POLITECNICO DI TORINO

Master Degree inMechanical Engineering

Master Degree Thesis

Study of advanced parametric meshing techniques for gearbox dynamic analysis

Supervisors
Prof. Carlo Rosso
Ing. Luca Ronchiato

Candidate
Edoardo Di Giuda

July 2019

Summary

In Finite Element Analysis, the characteristics of the mesh (quality and classification) play a major role in the final result; such characteristics include: mesh quality, kinds of elements, mesh refinement, and many others.
It is true indeed that commercial softwares are able, often with good results, to create meshes for basically any kind of components. These softwares, however, present a fundamental limitation. They are not open-source: this means that their code is not directly available to the consumer and can not be modified in order to adapt the meshing process to specific cases and requirements. In specific applications, after the mesh is generated automatically by one of these softwares, it requires to be modified by hand for it to meet the required characteristics. This post process has two main issues: first of all it often requires several hours of additional work (and this represents a cost in terms of labor); second, it introduces the risk of human error. Hence, the possibility to end up, after hours of work, with an invalid geometry is more than tangible.
The objective of this thesis is to create a tool which takes care of this issues by making the process fully automatic. The application of the tool are very specific, as it was customized in order to create meshes for gearbox components, even though it can be easily adapted to different scenarios. The tool is not able to directly mesh 3D components; instead, it relies on the possibility to obtain a 3D component from the transformation of a 2D section of it.
For the application it was developed for, it proved to be very effective since many of the components (if not all of them) are either axis-symmetric or cyclic-symmetric.

Acknowledgements

I would like to thank my tutors, who helped me in the development of this project: Ing. Luca Ronchiato who provided fundamental technical support, and Prof. Carlo Rosso who has always been available any time I needed his help.

I would also like to thank my friends, who made these college years the best I could possibly hope for.

Lastly, I'd like to thank my family, whose support has been crucial in making me reach my objectives and goals.

Contents

List of Figures

List of Tables

Chapter 1

General Introduction

1.1 Introduction

This initial chapter will focus on the general purpose of this thesis. In particular, the reasons behind the need of a custom meshing algorithm will be explored, as well as the objectives that were proposed before the whole development process started. A brief introduction of the different chapters will follow, in order to make the reader aware of the general logic and the central idea behind this document.

1.2 Why a custom meshing algorithm?

There is plenty of commercial softwares that are able to create a three-dimensional mesh starting from a 3D model, so it appears natural to wonder what is the need to develop a custom meshing algorithm.
First of all, it is useful to point out the fact that such softwares are not open source and, therefore, they are not very versatile in terms of user options.
Another aspect is the specific application that the tool will be used for: dynamic analysis for gearbox components. These components, combined with this kind of analysis, usually require extra care in mesh generation; this means that, when commercial softwares are used, additional time is often spent in optimizing the mesh by hand (i.e. nodes placement, connectivity change, and so on).
Hence, it appears clearer why the idea of a custom tool for mesh generation is appealing, as it may results in a major process optimization in terms of time consumption and labor.
Moreover, as this tool is able to generate lighter meshes, not only we have a reduction in time spent to create the mesh itself, but also a reduction in computational time during analysis.
Considering the convenient characteristics that have just been introduced, the objective of this thesis is the development of a tool that, starting from a two-dimensional

geometry, is able to create a three-dimensional mesh with the highest possible number of hexahedral elements, which are the most suitable for this specific application. The reason why the starting point is a two-dimensional geometry is that is much easier to achieve a full-quadrilateral (or at least an almost full-quadrilateral) mesh that, after applying the proper transformations, can be converted into a three-dimensional full-hexahedral (or almost full-hexahedral) one.
The environment that was chosen to develop this tool is MATLAB, mainly because if this environment is properly exploited the whole process can be sped up thanks to the logic behind the vector operations.

1.2.1 Applications

A tool such as this can find multiple applications, thanks to its high versatility. As long as the final mesh can be obtained by transformation of a two-dimensional domain, the tool can be adapted to the case. In particular, it results to be particularly efficient when the model is symmetric with respect to a certain axis (that is the reason why it was developed for gearbox components).

1.3 Proposed achievements

Before the development process started, the objectives were discussed in such a way to divide the overall process into a series of smaller steps. The following list presents said steps:

1. Dividing the components in classes, so as to define the input parameters of each individual one: the classes that were defined are general axis-symmetric components, axial holes patterns, radial holes patterns, and transition regions.

2. Developing a 2D-meshing algorithm in MATLAB language: for this step a pre-existing algorithm for Delaunay triangulation was exploited so as to focus mainly on the transformation of the mesh into an all-quadrilateral one. The algorithm logic used to achieve this step was inspired by an algorithm called q-morph and it will be discussed later in a more detailed way.

3. Defining and developing the transformations to be applied to the two-dimensional mesh; the required transformations are: rotation for the general axis-symmetric solid, extrusion for flanges, and coordinates switch and projection for radial holes patterns and transition regions.

4. Implementing a circular pattern in order to make it easier to create the mesh for components that present a cyclic symmetry.

5. Implement a function to erase and substitute nodes that are within a certain tolerance one from the other: this function is required to "weld" multiple parts of a component and perform the previously cited circular pattern.

1.4 Introduction of other chapters

Now, a brief summary of the following chapters, and why they have been organized in such a way, will follow.
The first of the following chapters will be a brief theoretical introduction to FE analysis: after a brief discussion on the role that it plays in engineering and an overview of its equations, the attention will later be focused more on the mesh and its classifications, discussing pros and cons of the different kinds.
After that, a deeper discussion on meshing algorithms will be required, as it is the actual theoretical basis of the whole work. Also in this case, a general classification will be introduced and then the attention will be focused on two algorithms, which played the main role in the tool development: Delaunay triangulation first and q-morph later. The last one will be treated much more in detail with respect to the first one since the Delaunay triangulations represents just the starting point of the tool.
Then, Chapter 4 will treat the actual tool development, defining the starting point of the tool and the general logic behind the implemented algorithm.
The final chapter will introduce results and conclusion; in particular, test cases will be presented, together with computation time, in order to show the factors that mostly influence the tool's performance.

Chapter 2

FEM analysis and Mesh

2.1 Introduction

In the following chapter, a general overview on FEM analysis will be provided, focusing more on its importance in engineering-related fields than on the actual equations on which it relies. After that, the main topic will be discussed: the importance of the mesh and its general classification.

2.2 FE analysis

The term Finite Element Method comes from the main logic of the method itself: a certain region of interest, which may refer to a fluid, a solid component, or more in general to a region in space, is discretized into a set of smaller, simpler elements. Then, fundamental equations are applied to each one of those elements depending on the property that is being looked for (it can be the displacement, or the temperature, the velocity of a fluid, et caetera). The solutions for equations on each element are then combined in order to obtain the overall solution for the whole domain.
This method is very versatile and find application in a diverse range of problems: it can be useful in structural analysis, as well as fluid flow analysis, and heat transfer analysis (as well as many others). Of course the main interest for this thesis is structural analysis of mechanical components.
Finite Element Analysis (FEA) refers to the practical application of FEM in a certain problem. Such an analysis often requires a set of partial differential equations (PDE) to be solved in order to obtain the approximation of the exact solution of the problem. The aim of a good FEA is to reduce as much as possible such error and, therefore, to obtain a solution that is as close as possible to the exact one.
The problem, since it is expressed as a set of partial differential equations, is usually expressed in matrix form. Since, as said before, the main field of application of this tool will be modal analysis for gearbox components, let's take a look at the

formulation of a linear static analysis of an elastic problem:

$$\bar{\bar{A}}\bar{u} = \bar{f} \tag{2.1}$$

where $\bar{\bar{A}}$ is the stiffness matrix, $\bar{u}$ is the displacements vector (which is the unknown in the problem), and $\bar{f}$ is the forces vector (which is the vector of known factors). Of course, the previous equation can be more complex as additional factors (such as damping) are taken into account.
The last thing that is worth pointing out is that, in order to be able to solve such a problem, a set of well defined boundary conditions. There are different kinds of boundary conditions and, depending on the property that they define, they are classified as:

- **Dirichlet** boundary condition: the value of the variable of interest is well-defined on the boundary;
- **Neumann** boundary condition: the partial derivative of the variable of interest is well-defined on the boundary;
- **Robin** boundary condition: it is a combination of the previous two.

2.3 Mesh

In general, a mesh M can be defined as the discretization of a certain region of interest (to which a certain geometry is associated) into a series of smaller, simpler elements. The mesh is fully defined by a certain set of information, called a tuple. More in detail, the information describe the vertices V and their connectivity Q in such a way that each element of the mesh itself is fully defined. Formally, considering a generic triangular two-dimensional mesh, it is expressed as:

$$M(V, Q) = (\{v_1, v_2, v_3\}, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_1\}\}) \tag{2.2}$$

in which the first set of vertices defines the element itself, and the other three couples of vertices define the edges of the element.

2.3.1 Types of geometric domain

The first factor to be taken into account, when considering a meshing algorithm, is the type of domain to be meshed. This characteristic concerns the boundary of our region only, and not its internal portion.
The first distinction is made in terms of dimensions: it can either be two-dimensional or three-dimensional. Since this thesis does not approach any 3D meshing algorithm,

we will not include details on 3D domains.
When considering 2D domains, the four following types of geometries can be distinguished:

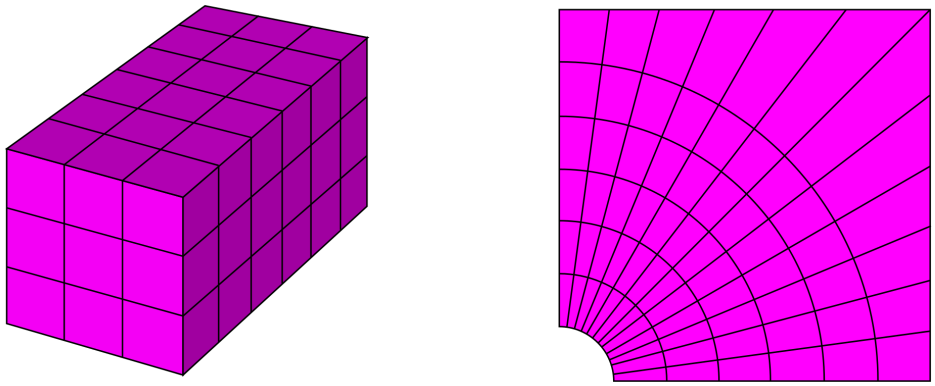
- Simple polygon: once a continuous set of edges is defined, the domain is the region enclosed in it;
- Polygon with holes: similar to the previous one, but a certain internal portion (or multiple portions) is not included in the domain;
- Multiple domain: it can be either a simple polygon or a polygon with holes, with the addition of internal boundaries; this comes in handy when a domain needs to be divided in multiple sub-domains with different geometric characteristics (such as mesh refinement, et caetera);
- Curved domains: any of the previous options, with the addition of curved sides.

Such a distinction is important in defining the limits of a meshing algorithm, in terms of types of geometry.

2.3.2 Types of meshes

As the type of geometry is a characteristics of the boundary, the type of mesh regards the disposition of nodes in the internal region and the way they are connected to one another.
Three main types of mesh can be identified:

- Structured: the nodes disposition follows a well-defined pattern; the main pros of this type of mesh are the fact that nodes are easily accessible and their storage requires less memory, however it is not very versatile in terms of geometry variability of the domain. In the following pictures, two examples of structured meshes:



(a) Cartesian grid (b) Curvilinear grid

Figure 2.1: Examples of structured grids

- Unstructured: in this case, nodes do not follow a predictable pattern, but at the same time they adapt way more easily to different kinds of geometry. In the following picture, an example of unstructured mesh:

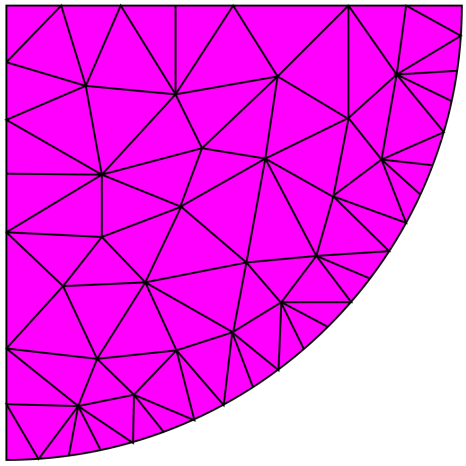


Figure 2.2: Example of an unstructured grid

- Hybrid: this type of mesh is a sort of combination of the previous kinds; the geometry is divided into sub-regions, whose disposition is unstructured, but each sub-region is organized in a structured fashion.

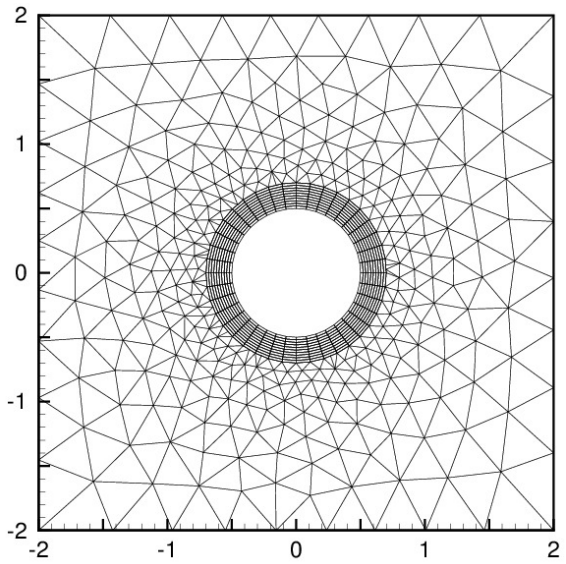


Figure 2.3: Example of a hybrid grid

2.3.3 Element Shape

Another important characteristic is element shape. The first thing that comes to mind is, of course, the number of edges of the element (when referring to a 2D domain) or the number of faces (when referring to a 3D domain).
In 2D, the most common elements are triangles and quadrilaterals; they can be usually combined in order to achieve a higher number of edges, such as pentagons.

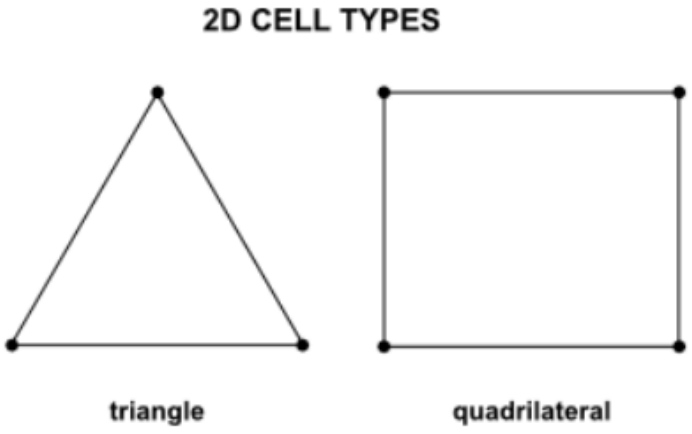


Figure 2.4: 2D cell types

In 3D, the most common elements are solid with four (tetrahedrons), five (triangle-based prisms or square-based pyramids) and six (hexahedrons) faces. They can even be generic polyhedrons, but they are not as common.

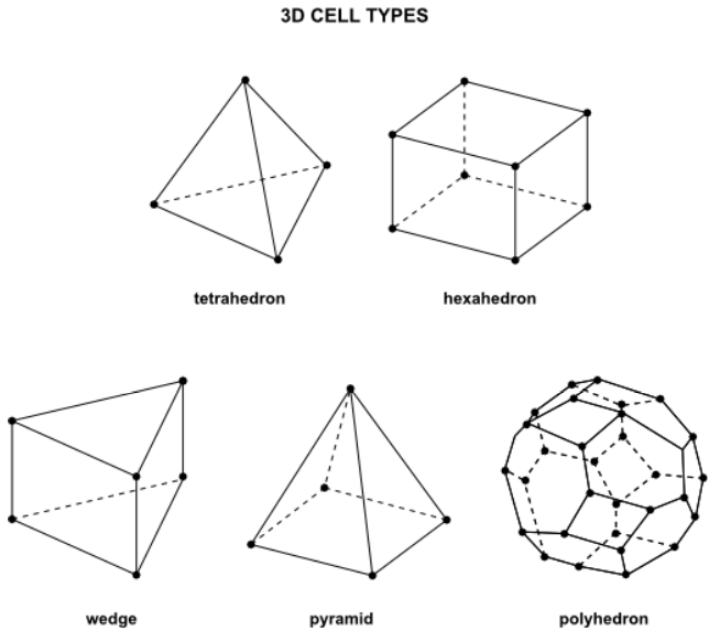


Figure 2.5: 3D cell types

Besides the number of edges or faces, the element shape is also characterized by a factor called aspect ratio. There is no unique definition of such factor. For the scope of this thesis, however, we will define the aspect ratio as the ratio of the largest to the smallest width of the element, where width is defined as the distance between parallel supporting hyperplanes.

2.3.4 Mesh topology

The last mesh characteristic that we are introducing is the mesh topology. According to its topology, we distinguish two kinds of mesh:

- Single-block: in this case, the mesh as a whole presents the same characteristics and there is no adaptation even in regions where it might be required to generate well-proportioned elements.

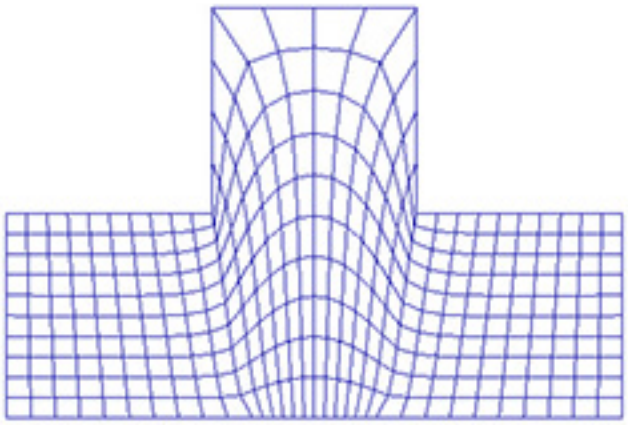


Figure 2.6: Example of a Single-block mesh

- Multi-block: in this case, instead, the mesh can be divided in different blocks and this allows to the definitions of characteristics based on user's needs.

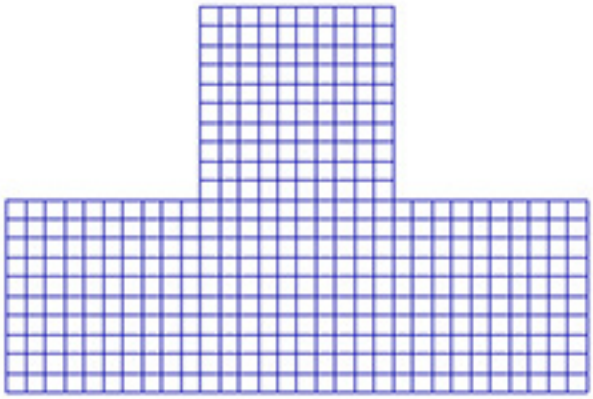


Figure 2.7: Example of a Multi-block mesh

2.4 Conclusions

Summarizing, after having introduced the main characteristics that define a mesh, we can define which one will be useful in our algorithm development:

- Type of geometric domain: for our particular case, we will consider a generic polygon with holes; there is also the possibility to introduce a multiple domain logic, but it was not used as it was not necessary.

- Type of mesh: as we are dealing with a different set of geometries, we will focus on the unstructured type of mesh, as it serves better our purpose.
- Element shape: in 2D we will take advantage of two shapes, triangle and quadrilaterals. Triangles represent the first step in our mesh generation, as the domain will first go through an algorithm called Delaunay triangulation; these triangle will later undergo a series of geometric operations and transformations that will convert them to quadrilaterals. In 3D, we will focus, instead, on hexahedrons and triangle-based prisms, as they are obtained through a set of geometric transformations of triangles and quadrilaterals.
- Mesh topology: at last, for our case, a simple single-block topology will be enough, without needing a more complex multi-block one.

Chapter 3

Meshing algorithms

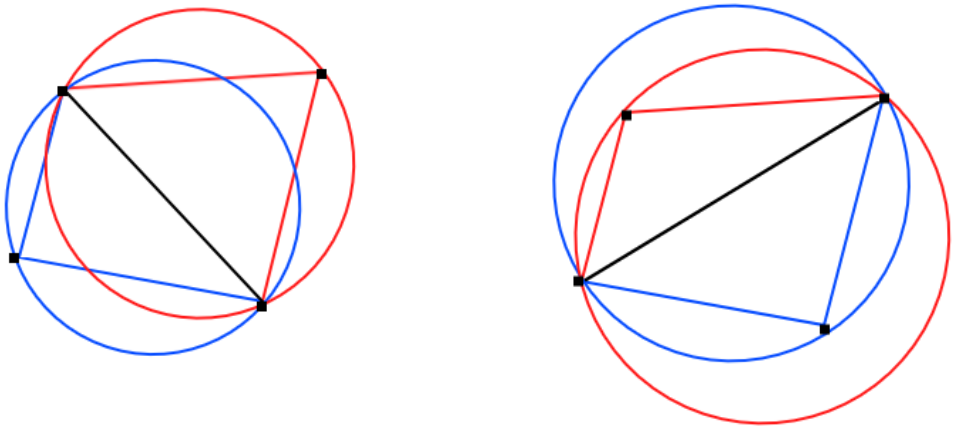
In this chapter, we will focus our attention on meshing algorithms. First of all, we will start with the introduction of a triangulation algorithm, called Delaunay triangulation. As this algorithm represents just the starting point of the tool, the main attention will be focused on the q-morph. This is an indirect algorithm which, starting from a given triangulation (obtained with the Delaunay one), will transform the domain in a set of quadrilaterals, by applying a series of transformation.

3.1 Delaunay triangulation

In order to fully understand the logic behind this algorithm, we need to introduce some theoretical concepts.

3.1.1 Empty Circle Property

Given a triangle, its *circumcircle* is defined as the unique circle passing through its three vertices. A triangulation is defined as a Delaunay triangulation if and only if the circumcircle of each triangle is empty; i.e. it does not contain any other point of any other triangle. In Figure 3.1 we can see as, starting from four points, we can connect them in order to form two triangles; however, only one of these configurations can be considered a Delaunay triangulation.



(a) Delaunay triangulation (b) Not a Delaunay triangulation

Figure 3.1: Delaunay vs non-Delaunay triangulation

As we can see, in Figure 3.1a both circumcircles do not contain any other point, while in Figure 3.1b both circumcircles contain an extra point which violates the requirement of the Delaunay triangulation.

3.1.2 Four points property

Another important characteristics of this triangulation is given by the *four points property*: for any four points in convex position and not laying on the same circle, there is exactly one Delaunay triangulation. This property proves to be very useful, if not fundamental, in every algorithm.

3.1.3 The algorithms

There is no unique way to generate a Delaunay triangulation, however all of the algorithms share the following steps:

1. Placing nodes on the boundary
2. Placing nodes on the inner region
3. Optimize inner nodes placing

There are different approaches to perform the second and third steps, based on a required value of mesh refinement:

- Nodes can be placed according to a series of structured grids, that are opportunely combined

- Nodes can be placed in successive layers, as a sort of advancing front starting from the boundary
- Nodes can be placed according to a random distribution, in a way that "over-populates" the region with internal nodes; nodes are then filtered out and refined

Now that both internal and boundary nodes are placed, the actual triangulation takes place. As stated before, there are different algorithms that are able to successfully carry out the process and reach a Delaunay triangulation. Since they are basically equivalent in terms of final results, we will go through an algorithm called *The Lawson Flip Algorithm.*
Since this algorithm is an indirect one, it requires a preliminar step where a generic triangulation is built. For this purpose, we will introduce the *Scan Algorithm.*

The Scan Algorithm A triangulation obtained by the *Scan Algorithm* usually presents elements with a very poor aspect ratio; it is performed in the following way:

1. This algorithm starts by placing four nodes, three colinear and an extra one, and by connecting the latter to the previous three.

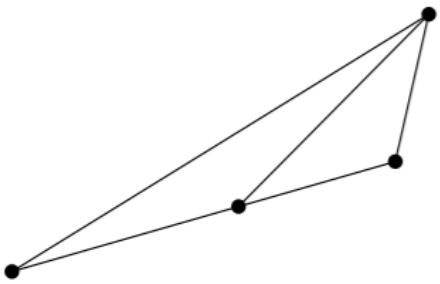


Figure 3.2: First step of a Scan triangulation

2. Further nodes, surrounding the original connections, are added, according to node density requirements, and connected to the "visible" ones; this means that the new nodes are connected in such a way that new resulting edges do not intersect old ones.

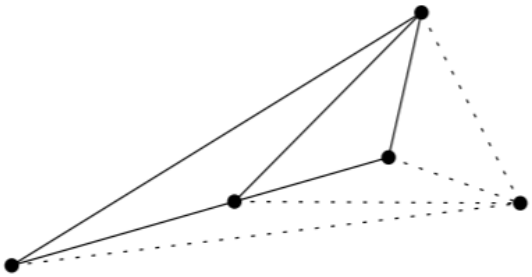


Figure 3.3: Further step of Scan triangulation

3. The process keeps going until the whole domain has been covered

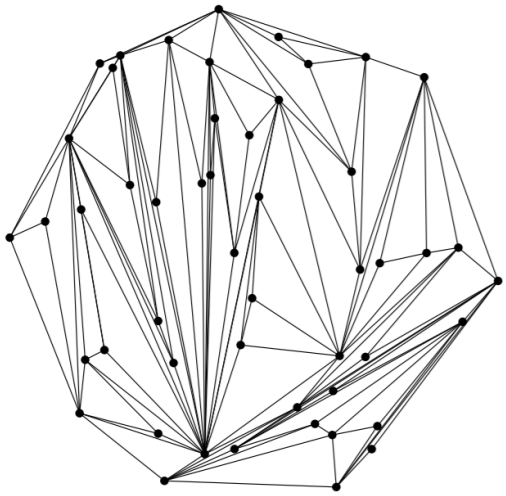


Figure 3.4: Complete Scan triangulation

The Lawson Flip Algorithm Starting from the previously built triangulation, the Lawson Flip Algorithm is a recursive process that proceeds as follows:

1. If any sub-triangulation in convex position that is not Delaunay is found, it is replaced with the corresponding Delaunay one by performing a *flip*, as shown in Figure 3.1;

2. Once a sub-triangulation has been "fixed", the search continues for another one;

3. The process continues until all the sub-triangulations are substituted with the corresponding Delaunay ones.

The overall result is much better with respect to the one obtained by the *Scan Algorithm*, as it can be observed in Figure 3.5:

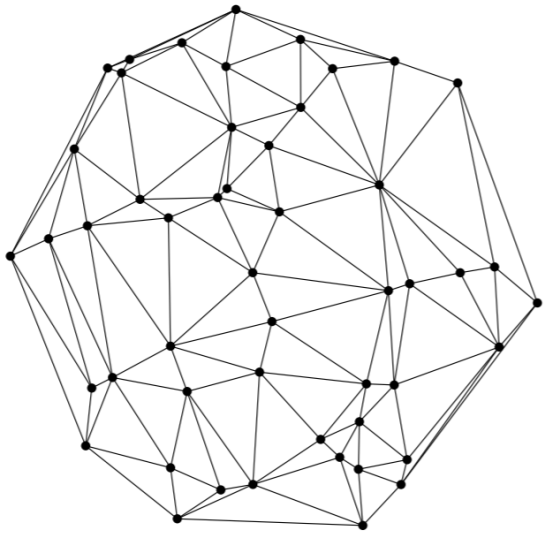


Figure 3.5: Complete Delaunay Triangulation

3.2 Q-morph

Now that the preliminar triangulation algorithm has been properly covered, the actual algorithm that is the fundamental logic of the developed tool can be accurately described more in detail.
The Q-morph, similarly to the Delaunay algorithm, is an indirect algorithm that requires a base triangulation to work with. It is defined as an advancing front method, meaning that quadrilaterals are formed starting from the domain's boundary towards the inner region. Quadrilaterals are placed one row after the other until the whole domain has been processed.
When defining the algorithm's logic, it can be divided in the following steps:

1. **Background mesh**: as previously stated, the algorithm requires a background triangular mesh; this step represents an important one, since the final size of the quadrilaterals will more or less match the one of the initial triangles; hence it is important to properly design the characteristics of the triangulation.

2. **Front definition**: as it was said before, the method is called an advancing-front method; this means that it is crucial to define the initial front. Of course it can be easily defined as the set of edges that make up the boundary; this translates as the set of edges which are part of only one triangle.

3. **Front edge classification**: now that the front has been defined, it needs to be classified; edges are classified according to their state. The state of an edge defines how the surrounding ones will be used in defining the quadrilateral; we will see what this means more in details later.

4. **Front edge processing**: this is the step where the quadrilateral is actually formed from the edge that is being processed. It is probably the most complex and fundamental step. In fact, it can be divided into a series of sub-steps:

   (a) *Side edge definition*: starting from the front edge, in order to form a quadrilateral, two side edges are required; depending on the edge's state, none, one, or two side edges will have to be defined according to the surrounding mesh. There are three main mechanisms to define a side edge: it can be defined as an existing edge in the existing mesh; it can be obtained by swapping diagonals; or it can be obtained by splitting an existing edge.

   (b) *Top edge recovery*: once the two sides are defined, the top edge must be recovered; also in this case, there are two options: the top edge might already exist in the background triangular mesh (in which case there is no need of further operations) or there may be the need of a series of swaps in order to recover it. The detailed mechanism will be explored later.

   (c) *Quadrilateral formation*: once the four sides of the quadrilateral are defined, the last step is to delete from the mesh the triangles that are within the quadrilateral itself, plus any residual edge and node.

   (d) *Local smoothing*: a local smoothing is applied, in order to improve the mesh quality in the region surrounding the newly formed quadrilateral; there are different options to choose from, when talking about mesh smoothing; the techniques that are used will be discussed more in detail later.

   (e) *Local front reclassification*: finally, the front is updated in the region surrounding the newly formed quadrilateral and the state is updated (as it may have changed due to the geometric transformations that were applied in the process).

5. **Topological clean-up**: once all the quadrilaterals have been processed, the mesh undergoes a topological clean-up; this operation consists in local change of connectivity in order to improve the mesh quality.

6. **Smoothing**: a last global smoothing is applied.

Now some of the previous steps will be discussed more in detail, as they represent a crucial point in the algorithm.

3.2.1 Front definition and classification

In Figure 3.6, the process of defining the state of a node on the current front is shown. The state of a node can be either 0 or 1; given a generic node N_k and its two consecutive front edges, its state is determined depending on the value of the angle α_k; this angle is the one formed by the mentioned front edges. The state is hence determined as follows:

$$state = \begin{cases} 1, & \text{if } \alpha_k < \frac{3}{4}\pi \\ 0, & \text{if } \alpha_k \geq \frac{3}{4}\pi \end{cases} \tag{3.1}$$

In the specific case represented in Figure 3.6, it is clear that the node is in state 0.

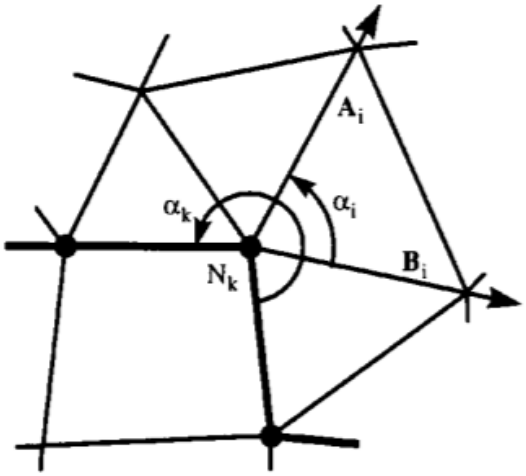
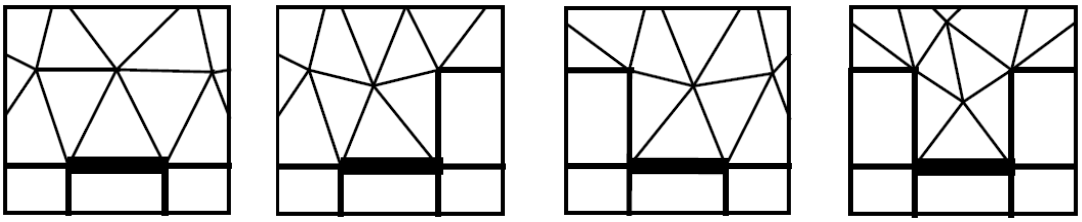


Figure 3.6: Node's state definition

As the edges are defined by two nodes, their state can be $'00'$, $'01'$, $'10'$, or $'11'$.



(a) '00' (b) '01' (c) '10' (d) '11'

Figure 3.7: Possible states of a front edge

3.2.2 Front edge processing

Among the different sub-steps that constitute this major point, the following ones are those which require some additional explanation: side edge processing, top edge recovery, and smoothing.

Side edge processing As previously stated, a node can be in a state that can be either 0 or 1 and it defines whether the side edge must be recovered or not. In case the node is in state 0, and hence it requires a side edge recovery process, different options are available. Let's first take a look at the general setup, and we will then see which are the discriminant factors that lean towards an option rather than the other.

General set-up We can observe a generic setup for the side edge selection in Figure 3.8; the node N_k is in state 0 and requires a side edge selection. As a first step, being E_{F1} and E_{F2} the front edges that share node N_k, the vector V_k is built in such a way that it passes through N_k and is parallel to the bisector of E_{F1} and E_{F2}. We can identify a certain tolerance angular region, by moving around V_k of a certain value ϵ.

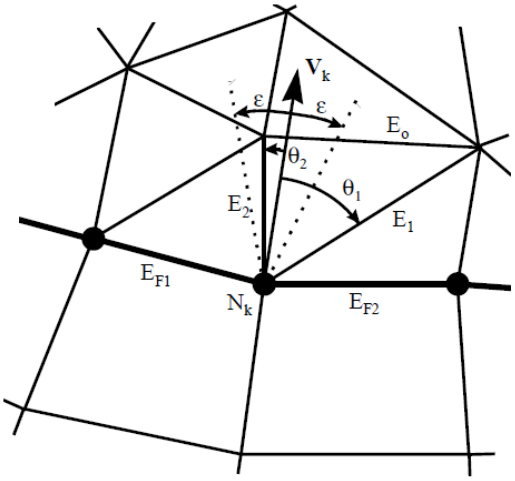


Figure 3.8: Side edge selection set-up

Now that the setup is clear, we can introduce the different options.

Edge selected from base triangulation In case an edge that already exists in the base triangulation falls in the tolerance region around V_k, this very edge is selected as the side edge; if there are multiple edges in this interval, the closest to V_k is selected.

If the previous operation is not possible (i.e. no edge falls in the tolerance interval), two options are available: the swap ans the split operations

Swap or Split In Figure 3.9, a situation where swap or split must be adopted is presented.

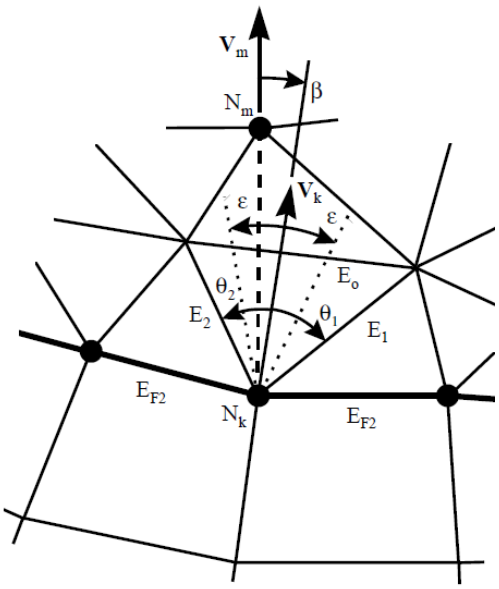


Figure 3.9: Swap or Split

As we can see, there is no edge falling within the tolerance interval around V_k; in this case, the following considerations are made: once E_1 and E_2 are found (they are the edges that contain the vector V_k), the triangle that is adjacent to the one delimited by the two edges is found; depending on the position of the furthest vertex N_m of said triangle, swap or split are applied. More in detail, in order to perform the swap operation, the following conditions must be met:

$$\begin{cases} \beta < \epsilon \\ \|N_k N_m\| < \sqrt{3}\frac{\|E_{F1}\|+\|E_{F2}\|}{2} \end{cases} \tag{3.2}$$

where β is the angle between vector V_k and the vector going from N_k to N_m, and $\|N_k N_m\|$ is the distance between N_k and N_m. If these conditions are met, then the diagonal that intersects V_k is swapped in favor of the one going from N_k and N_m, as it is shown in Figure 3.10.

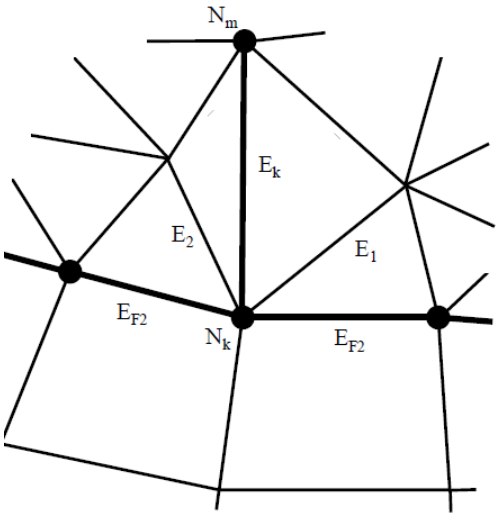


Figure 3.10: Swap

If the swap operation cannot be performed, due to a violation of one of the previous conditions, the split operation takes place. In this process, the edge that intersects vector V_k is split at the intersection point N_n itself. This creates two additional triangles, an additional point, and three additional edges, that need to be taken into account in the future.
Figure 3.11 shows the result of such operation

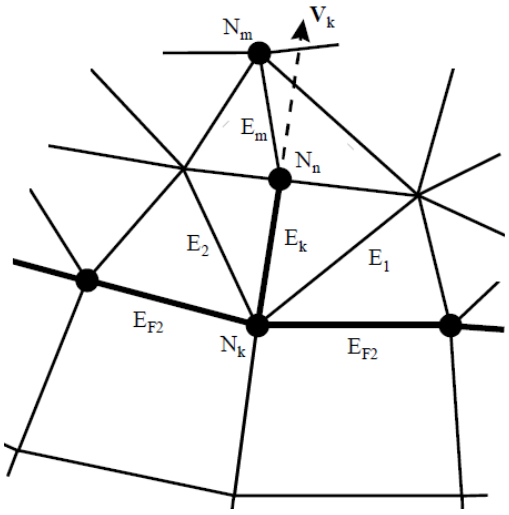


Figure 3.11: Split

Top edge recovery Now that the side edges are defined, there is nothing left, in order to have the four edges of our quadrilateral, than to recover the top edge. It

must be pointed out that this operation is not necessary and it is required if and only if the top edge does not exist in the base triangulation already. In which case, it is recovered by means of an iterative process.
In Figure 3.12, we can see a setup where the top edge recovery is required; in the picture we can identify different elements: N_C and N_D are the ending nodes of our side edges, the dashed line S is the line connecting the two nodes (i.e. the top edge to be recovered) and E_1,E_2, E_3, and E_4 are the edges intersecting S.

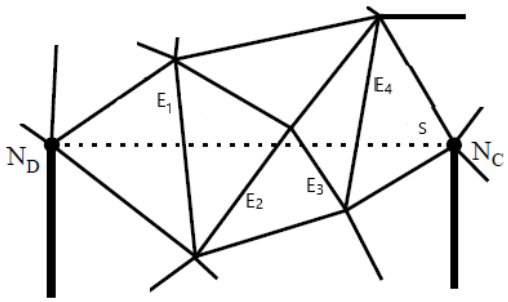


Figure 3.12: Top edge recovery

The process to recover the top edge is summarized in the following points:

1. The set of edges E_n, intersecting S, is stored in a list;
2. The first edge of the list is selected, and the two triangles that it divides are found;
3. The swap operation is performed;
4. The result of the swap operation is checked for consistency: it may happen that the triangles become inverted or that the swapped diagonal still intersects the line S;
5. If the swap operation is successful, the edge E is deleted by the list and the next one is selected; otherwise, the edge is put in the last position of the list and another edge is selected to attempt the swap;
6. the operation continues until the top edge is recovered or it fails multiple times; in the last situation the top edge is not recovered anymore and the q-morph proceeds with the processing of another front edge.

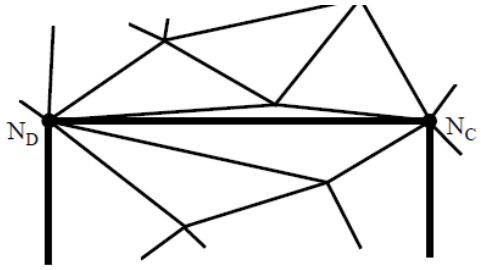


Figure 3.13: Top edge recovered

In Figure 3.13, the result of a successful top edge recovery is shown.
Since the quadrilateral-formation step is nothing but the elimination of the triangles, nodes, and edges that are within the four edges, we will pass directly to the smoothing process.

Smoothing Smoothing is essential to bring the mesh back to an acceptable condition, after it gets altered by the series of geometric transformations required to form a quadrilateral. However, depending on each node's specific connectivity, a different kind of smoothing is applied. In particular, we distinguish two different kinds of nodes: inner/outer nodes, and front nodes.
For the first class of nodes, a simple Laplacian smoothing is preferred, since it is not computationally expensive and it gives good results. For the second kinds of nodes, however, a simple Laplacian is too generic and does not provide certain crucial characteristics to the final result; therefore a very specific smoothing process, called modified isoparametric smoothing, is preferred.

Laplacian smoothing This kind of smoothing is applied to the nodes that are not locate on the current front. It is relatively fast and it is able to reach convergence in a few iterations. This process tends to move nodes to the centroid of the surrounding ones. If a certain node N_k is connected to a number of nodes m, then the Laplacian smoothing calculates the new position of point N_k as:

$$\vec{P}_{N_k} = \frac{\sum_{i=1}^{m} \vec{P_{N_i}}}{m} \tag{3.3}$$

where $\vec{P}_{N_k}$ is the vector corresponding to the position of node N_k and, similarly, $\vec{P_{N_i}}$ is the vector corresponding to the position of the nodes connected to N_k.

Modified isoparametric smoothing With respect to a simple Laplacian smoothing, this alternative method is way more complex, as it tries to provide additional characteristics to the final result; in particular, the reason why this method is applied to the nodes on the current front is that it is able to move nodes in such a

way that it create well-proportioned quadrilaterals and a smooth continuous front, at the same time. This method is applied to the nodes that are on the current front and are in contact with exactly two quadrilaterals. Let's go step-by-step through this process.

1. *Isoparametric smoothing*: the first step is calculating the change in position the the node would undergo, if a simple isoparametric smoothing was applied. Let's suppose that a certain node N_k, with position $\vec{V_k}$, is surrounded by a number n of quadrilaterals. The position given by its isoparametric smoothing is given by:

$$\vec{V_{k'}} = \frac{1}{n}\sum_{m=1}^{n}(\vec{V_{mj}} + \vec{V_{ml}} - \vec{V_{mz}}) \tag{3.4}$$

   where $\vec{V_{mj}}$ and $\vec{V_{ml}}$ are the positions of the nodes that are directly connected to N_k, and $\vec{V_{mz}}$ is the position of the node that is diagonally opposed to N_k. Therefore, we obtain a theoretical change in position equal to:

$$\Delta_A = \vec{V_{k'}} - \vec{V_k} \tag{3.5}$$

2. *Length adjustment*: we now calculate a modified value of Δ_A, based on the following formula:

$$\Delta_B = \vec{V_j} - \vec{V_k} + (\Delta_A + \vec{V_k} - \vec{V_j})\frac{l_D}{l_A} \tag{3.6}$$

   where l_A is the length of the edge $\bar{N_kN_j}$ if the isoparametric smoothing was applied, and is calculated as:

$$l_A = \|\vec{V_{i'}} - \vec{V_j}\|, \tag{3.7}$$

   l_D is an ideal length of $\bar{N_kN_j}$, based on the characteristics of the surrounding mesh, and calculated as:

$$\begin{aligned} l_D = &\frac{\|\vec{V_{k-1}} - \vec{V_{j-1}}\| + \|\vec{V_{j-1}} - \vec{V_j}\| + \|\vec{V_{k+1}} - \vec{V_{j+1}}\|}{4+n} + \\ &+ \frac{\|\vec{V_{j+1}} - \vec{V_j}\| + \sum_{i=1}^{n}\|\vec{V_k} - \vec{V_{ti}}\|}{4+n} \end{aligned} \tag{3.8}$$

   and $\vec{V_j}$ is the position of the node behind the front and connected to N_k. In order to fully understand of the previous formulas, Figure 3.14 can provide an easy reference:

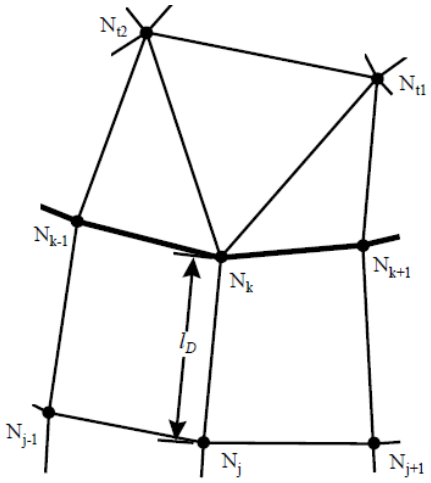


Figure 3.14: Length adjustment - Modified isoparametric smoothing

3. *Angular adjustment*: the following adjustment is probably the most complex, since it consists in a series of angles and intersection calculations. It is calculated as:

$$\Delta_C = \vec{P_{B2}} - \vec{P_k} \tag{3.9}$$

While $\vec{P_k}$ is just the vector connecting N_k to N_k, in order to explain what $\vec{P_{B2}}$ is and how it is calculated, it is necessary to introduce a graphic representation:

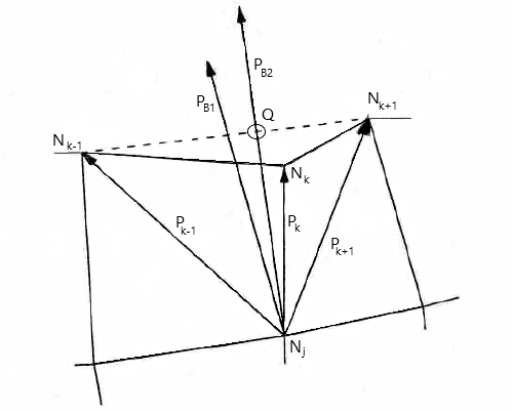


Figure 3.15: Angular adjustment - Modified isoparametric smoothing

First of all, the vectors $\vec{P_{k-1}}$, $\vec{P_k}$, and $\vec{P_{k+1}}$ are calculated as the vectors which go from N_j to respectively N_{k-1}, N_k, and N_{k+1}. Then, the vector $\vec{P_{B1}}$ is obtained by positioning its tail on N_j and orienting it in the direction of the bisector of the vectors $\vec{P_{k-1}}$ and $\vec{P_{k+1}}$. The angle of vector $\vec{P_{B2}}$ is obtained

by positioning its tail on N_j and orienting it in the direction of the bisector of vectors $\vec{P_k}$ and $\vec{P_{B1}}$. Finally, the point Q is determined as the intersection between $\vec{P_{B2}}$ and the line connecting N_{k-1} and N_{k+1}. The norm of $\vec{P_{B2}}$ depends on l_D and l_Q:

$$\|\vec{P_{B2}}\| = \begin{cases} \frac{l_Q + l_D}{2} & \text{if } l_D > l_Q \\ l_D & \text{otherwise} \end{cases} \tag{3.10}$$

where l_Q is the distance between N_k and Q.
We can now define the angular adjustment as:

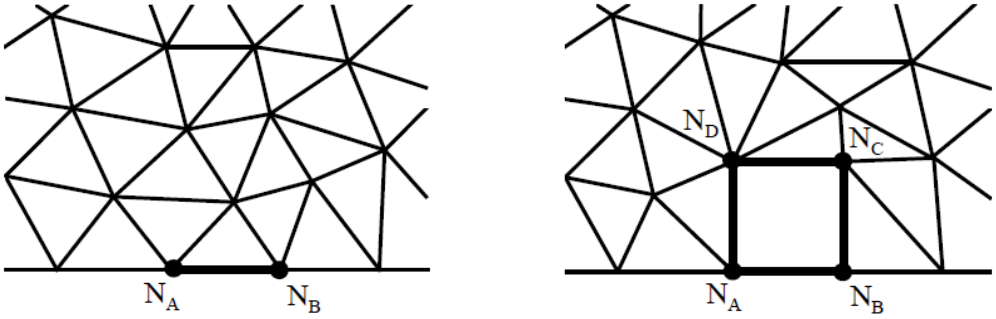
$$\Delta_C = \vec{P_{B2}} - \vec{V_k} \tag{3.11}$$

4. *Modified isoparametric smoothing*: the actual variation in position is finally calculated as an average between the length adjustment and the angular adjustment:

$$\Delta_k = \frac{\Delta_B + \Delta_C}{2} \tag{3.12}$$

Even though, as it can be seen, this method requires much more calculations with respect to the simple Laplacian, it is also true that it requires less iterations to reach convergence, and the nodes that requires it are much fewer than the common ones.

Front reclassification The formation of a new quadrilateral introduces the necessity to update the front information: in particular, both edges' connectivity and nodes' state must be updated. We can see in Figure 3.16 how the front changes after a quadrilateral is formed.



(a) Before (b) After

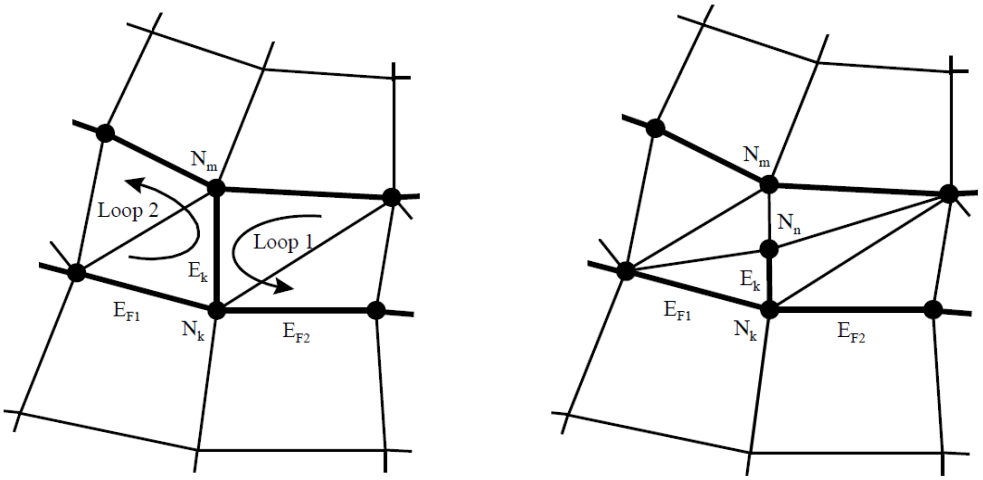
Figure 3.16: Front before and after a quadrilateral is formed

First of all, we notice that three new edges are introduced in the front: in Figure 3.16a, edge $\overline{N_A N_B}$ is part of the front; in Figure 3.16b, the edge $\overline{N_A N_B}$ is substituted by three new ones ($\overline{N_A N_D}$, $\overline{N_D N_C}$, and $\overline{N_C N_B}$). Second, the state of

nodes N_A and N_B passes from $'0'$ (as the angle between the subsequent edges is roughly π) to $'1'$ (as the angle becomes roughly $\frac{\pi}{2}$). Finally, two new nodes are introduced in the front (N_D and N_C) and their state is $'0'$ (as the angle between the edges is roughly $\frac{3}{2}\pi$).

3.2.3 Front closing

One of the most appealing algorithm of the 1-morph algorithm is that it is able to ensure an all-quadrilateral mesh, as long as the number of edges on the initial front (i.e. the boundary of the domain) is even. In case this condition is not met, the algorithm generates necessarily a single triangle.
However, even if the starting number of edges is even, in order to get the promised final result, extra care must be provided during side edge selection. It might happen (it actually always happens sooner or later during the process) that the edge, selected as the side edge, has the ending node on the same loop (a continuous, unbroken line of edges on the front) as the one on which the front edge is located. For a better understanding, let's take a look at Figure 3.17:



(a) Before split

(b) After split

Figure 3.17: Front closing

Node N_k requires the definition of a side edge, as it is in state $'0'$. If the edge selected as a possible side edge is E_k with the ending node N_m located on the same front, the following consideration must be made: If N_k and N_m are not on the same loop, then there is no problem and the edge is selected; however, if they belong to the same loop, it means that the selection of $_k$ as side edge will cause the unbroken loop to break down into two sub-loops (Loop 1 and Loop 2). Now, in order to ensure a final all-quadrilateral mesh, the number of edges on both resulting loops must be

even. If they are not even, edge E_k is split at its midpoint as in Figure 3.17b.
It is worth to point out the fact that, in such situations, it might happen that the edge that intersects the bisector during the side edge selection (the one named E_0 in Figure 3.8) is an edge on the current front. As such, since the swap or split operations would compromise the front itself, a larger value of ϵ is selected so that it is more likely to select an edge from the base triangulation.

3.2.4 Special cases

The logic that was described up to now is just a general set of instructions that the algorithm follows in order to process triangles into quadrilaterals. However, there might be some cases that require special attention and a more specific set of instructions in order to process them. We distinguish three different special cases: seaming, transition seam, and transition split. Let's introduce first the factors that determine when such cases come into play.

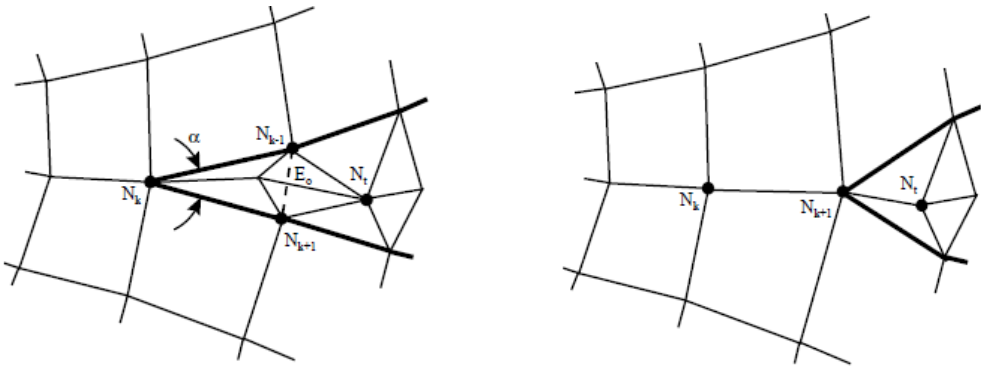
- *Smallest tolerable angle (ϵ)*: if the angle between two consecutive edges on the front is smaller that ϵ, seaming is applied; there are two different values for ϵ, and whether one or the other is considered depends on the number of quadrilateral that the central node shares.
- *Maximum length ratio (r_{max})*: if the ratio between the length of two consecutive edges on the front is greater than r_{max}, either transition seam or transition split is applied.
- *Number of quadrilaterals (n_Q)*: the number of quadrilaterals in contact with a node on the front determines the value of *epsilon* that must be considered for special cases application. In particular we usually have:

$$\epsilon = \begin{cases} \epsilon_1 & \text{if } n_Q > 5 \\ \epsilon_2 & \text{otherwise} \end{cases} \tag{3.13}$$

  with $\epsilon_1 < \epsilon_2$.

Seaming Being α the angle between two consecutive edges on the front, and r the ratio of lengths between the same two edges, the *seaming* operation is performed when the following conditions are met:

$$\begin{cases} \alpha < \epsilon \\ r \leq r_{max} \end{cases} \tag{3.14}$$



(a) Before seaming (b) After seaming

Figure 3.18: Seaming

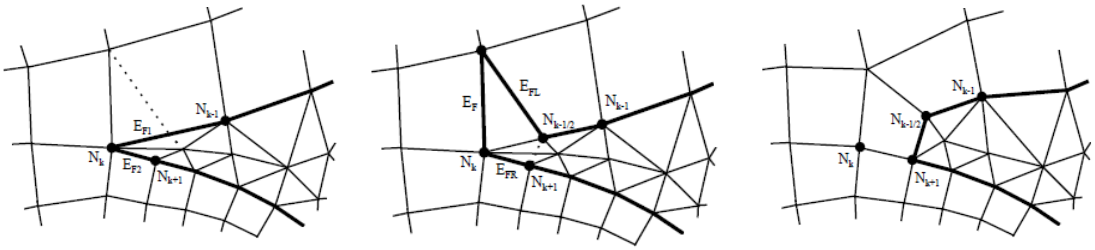
In Figure 3.18a, we can see a situation where *seaming* is required. The angle α, corresponding to the central node N_k is smaller than the limit ϵ and the ratio r does not exceed the limit value r_{max}. The operation proceeds in the following way:

1. Considering a counter-clockwise ordering of the nodes on the front, the node immediately before and immediately after the central one are found (respectively N_{k-1} and N_{k+1}, and so is the edge E_0 connecting the two.
2. If edge E_0 does not already exist in the base triangulation, it is recovered with a mechanism that is very similar with the one used to recover the top edge.
3. The new position of node N_{k+1} is calculated at the midpoint of E_0.
4. The triangles included in the region delimited by N_k, N_{k+1}, N_t, and N_{k-1}, as well as the nodes and the edges, are deleted from the triangulation. In addition, also the edges $\overline{N_k N_{k-1}}$ and $\overline{N_{k-1} N_t}$ are deleted, and the connectivity is modified in such a way that any residual edge connected to N_{k-1} results now connected to N_{k+1}.
5. Finally, smoothing is applied locally.

The final result of seaming can be observed in Figure 3.18b.

Transition seam When the following conditions are met, transition split is performed:

$$\begin{cases} \alpha < \epsilon \\ r > r_{max} \end{cases} \tag{3.15}$$



(a) Before transition seam (b) After transition seam (c) After quad formation

Figure 3.19: Transition seam

In Figure 3.19a, we can observe a situation where *transition seam* is required. In particular, the main difference between this operation and a normal *seaming* is that we are also able to deal with the transition of edge length between consecutive edges and smooth it out. The operation proceeds as follows:

1. As before, nodes N_{k-1} and N_{k-1} (and therefore edges E_{F1} and E_{F2}) are found as the node immediately before and immediately after the central one N_k.

2. The longest edge (in this case E_{F1}) is split at its midpoint $N_{k-\frac{1}{2}}$. This cause the formation of two new triangles: one given by splitting the triangle in contact with the long edge, and the other formed by portioning the quadrilateral in contact with the same edge.

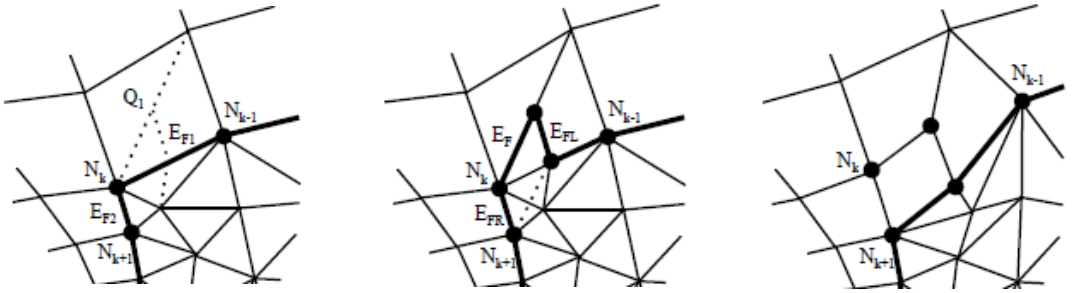
3. Once we reach the situation shown in Figure 3.19b, the front is processed by considering E_F as the front edge and E_{FL} and E_{FR} as side edges.

4. Finally, the mesh is locally smoothed.

We can observe the condition of the mesh, after the quadrilateral is formed and the mesh is smoothed, in Figure 3.19c.

Transition split The conditions required for the *transition split* to happen are the following ones:

$$\begin{cases} \alpha \geq \epsilon \\ r > r_{max} \end{cases} \tag{3.16}$$



(a) Before transition split (b) After transition split (c) After quad formation

Figure 3.20: Transition split

in Figure 3.21, the transformation which the mesh undergoes during *transition split* is shown. As we can see, a more smooth transition is required between consecutive edges, but the angle that they form is way too large to perform the *transition seam*. The process can be divided in the folloqing steps:

1. As usual, the nodes immediately after and immediately before the central one are found (N_{k+1} and N_{k-1} respectively).

2. Two new nodes are added to the mesh: the first one is obtained by splitting the long edge (in this case E_{F1} at its midpoint), and the other is positioned at the centroid of the quad in contact with the long edge.

3. As a result, two triangles and a quad are also added: one of the triangles come from the splitting of the triangle in contact with the long edge, while the quadrilateral and the other triangles come from the splitting of the quadrilateral that is in contact also with the long edge.

4. Then, the front is processed by considering E_F as the front edge, and E_{FL} and E_{FR} as side edges.

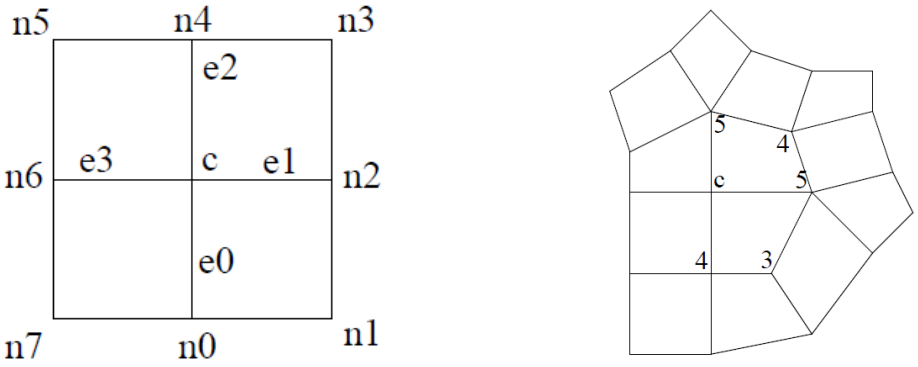
5. The mesh is locally smoothed.

In Figure 3.20c, the mesh after the whole process can be observed.

3.2.5 Topological clean-up and final smoothing

The final step is represented by an operation called *topological clean-up*. During this process, nodes and connectivity are changed throughout the whole mesh in order to reduce as much as possible the number of irregular nodes; by definition, in a quadrilateral mesh, a node is said to be irregular when it is connected to a number of nodes different from four.
There is a very diverse series of operations that actually take place during clean-up.

However, in the actual implementation, only one specific case was considered in order to reduce time consumption (usually a full cleanup requires as much time as full algorithm right before it is executed). The set of operations that are performed depends on the classification of the nodes. The nodes are classified according to their surrounding and their connections.
First of all, let's take a look at how the nodes are selected as eligible for a possible cleanup:



(a) Neighboring definition (b) Node's valence

Figure 3.21: Clean-up nodes classification

In Figure 3.21a, we consider e general node internal to the mesh c. We define the edges that are directly connected to the node as *neighboring edges* (in this case e_0, e_1, e_2, and e_3). We define the nodes surrounding the central node as *neighboring nodes* (in this case all nodes from n_0 to n_7); it is worth noticing that a node can be defined as a surrounding one even if it is not directly connected to c.
Now, we define the valence of a node as the number of quadrilaterals that the node itself shares. If a node's connectivity and position can not be changed during the process (i.e. boundary nodes) they are assigned the arbitrary value of 0.
In Figure 3.21b a practical example is shown: we have a central node connected to four quadrilaterals and it is surrounded by various irregular nodes. A certain "identification tag" will be attached to the node itself, and it will be defined in the following way: the first number is the valence of the central node; then, after a dash, the valences of the *neighboring nodes* are reported in a counterclockwise order. Therefore node c will have the following classification: $4 - 43545000$.
The classification is fundamental in defining the operations that will be performed on the node and its surrounding, as the clean-up process is based on looking for well-known patterns (identified by a specific tag) and acting on them. We will now introduce the only case that is considered in the tool during the cleanup step.

Two-edge node elimination In Figure 3.22, an example of such a case is presented. The central node c is connected to exactly two edges and shares exactly two quadrilaterals. Its classification is $2-4444$; however in general, this kind of clean-up is applied to any node whose classification is $2-xxxx$.

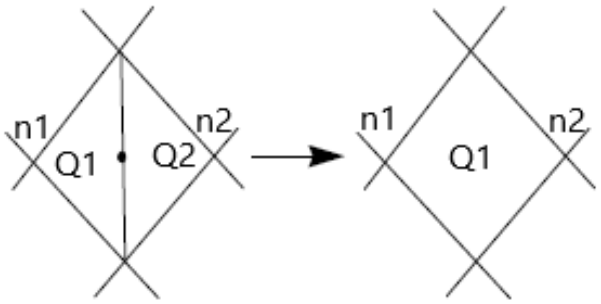


Figure 3.22: Two-edge nodes elimination

The mesh is processed in the following way (in the region surrounding the central node):

1. The central node is erased, as well as one of the two quadrilaterals.
2. In the residual quadrilateral, the central node is replaced by n_2.
3. Finally, the two edges connected to the central node are also erased, and smoothing is applied locally.

The reason why this case was the only one considered during cleanup is that it is quite easy and fast too individuate in the mesh, and it also removes critically misshaped quadrilateral elements (which are basically triangles before the cleanup is applied).

3.3 Conclusions

The two algorithms that were introduced in this chapter (i.e. the *Delaunay triangulation* and the *Q-morph*) are at the basis of the developed tool; they work back to back, as the domain first undergoes a process of triangulation, and then is processed according to the q-morph algorithm.
Since the triangulation will be performed by a pre-existing tool, the main attention will be focused on the q-morph implementation in MATLAB environment. In particular the next chapter will explain in general the development process for our specific application.

Chapter 4

Tool development

The need for this tool derived from the necessity to create meshes for gearbox components. Hence, a very specific set of components were identified, and the input parameters for each one of them were defined. Even though the developed tool is able to basically mesh any two-dimensional domain, the kinds of three-dimensional transformations in order to reach a 3D mesh are specific and differs for each class of components.

4.1 Pre-existing gears meshing tool

A previously developed tool had the task to create meshes for gears, given a certain set of input parameters that fully define the gear's geometry and its mesh refinement. The tool is able to create meshes for spur, helical, and conical gears by dividing the tooth's section in ordered regions (either three or five depending on the gear's characteristics), and meshing them individually according to a structured grid. Three-dimensional transformations are then applied in order to pass from a simple 2D section to the full 3D tooth first, and to the whole gear later.
The algorithm follows a completely different logic with respect to the ones that were introduced in Chapter 3, as the mesh does not share compatible characteristics with their process: first of all the mesh is structured, so it relies on the fact that a well-known geometry will be given as an input; second, the algorithm is not able to deal with domains with holes (according to the definition introduced in Chapter 2).
Therefore it is a very specific tool that performs a very specific task, with variability sufficient enough to span across a various set of input parameters, that changes the actual dimensions of the mesh but does not affect its topology.
In order to provide a clear picture of the interaction between this tool and the newly-developed one, it is worth to introduce the input parameters for a gear.

Geometric parameters:

- Number of teeth z
- Module m_n
- Pressure angle α
- Tip diameter d_{tip}
- Root diameter d_{root}
- Fillet type $fillet$
- Circular tooth thickness t_p
- Width w
- External/Internal diameter d_{ie}

Mesh parameters:

- Number of elements on the profile $N_{profile}$
- Number of elements on the fillet N_{fillet}
- Number of elements on upper rim $N_{rim-top}$
- Number of elements on lower rim $N_{rim-bottom}$
- Number of elements on the root N_{root}
- Number of elements along the width N_{width}

4.2 New tool

Of course the gear is just a small part of the whole component whose mesh is required. The objective of this newly developed tool is to create the remaining parts so that they can be opportunely merged together and obtain the final, fully-meshed components. It is necessary for the two tools to interact with one another.
First of all, it is important to specify which kind of component is treated and how we will divide it in its individual parts. The tool is applied to gearbox components, in particular to shafts. Usually a shaft can be divided in such a way that two classes of parts are found:

- *Axis-symmetric solids*: generic portions of the shaft that can be obtained by revolving a section around the main axis;
- *Cyclic-symmetric solids*: a portion where a specific feature follows a circular pattern and is repeated along the whole circumference.

However, defining just a generic cyclic-symmetric solid is not enough; an additional distinction is required, depending on the feature to be repeated along the circumference:

- *Axial holes pattern*: components that present holes in the axial direction;
- *Radial holes pattern*: components that present holes in the radial direction.

They differ in the set of input parameters and the way they are processed in 3D. Therefore we will deal with three different mesh categories: *axis-symmetric solids*, *axial holes patterns*, and *radial holes patterns*. For each one of them, the required geometric parameters will be introduced, as well as a whole series of parameters that affect the processing logic depending on how they are set. Since this last set of parameters is shared by each class, even though the actual values change, we will introduce them first.

4.2.1 Q-morph "tuning" parameters

These factors, which influence the way the algorithm process the front, are basically the parameters that were introduced in Section 3.2.2 and Section 3.2.4:

- β_{lim}: it was noticed that limiting the value of the bisector to a certain maximum helps having well-shaped quadrilaterals; selecting a value that is too close to $\frac{\pi}{2}$, however, increases the chance of applying swaps or splits, and locally affects the quality of the mesh. Leaving the value without constraint increases the risk of creating badly-shaped elements in correspondence of acute angles.

- $\Delta\epsilon_1$: this value represents the tolerance around the bisector for the selection of the side edge; an higher value increases the chance of selecting an edge that is not appropriate for the front edge that is being processed (similar to leaving no constraint to the bisector angle); on the contrary, selecting a value that is too low increases the chance of performing split operations and it would affect locally the quality of the mesh.

- $\Delta\epsilon_2$: the value of tolerance that is used in case we are not able to perform swap or split operations; the effects are basically the same as for $\Delta\epsilon_1$, with the difference that selecting an high value of $\Delta\epsilon_2$ reduces the chances for the algorithm to get stuck.

- r_{max}: the maximum acceptable ratio of consecutive edges' lengths; selecting a value that is too low would increase the risk to perform transition splits one after the other in the same region. This would cause some regions of the mesh to have an abundance of nodes and elements that are not properly sized with respect to the rest.

- ϵ_1: not to be confused with $\Delta\epsilon_1$, this is the angle considered for special cases, when the number of quadrilaterals in contact with the node is larger than five; a value that is too low would increase the chance to create elements with the shape of a rhombus, where a seaming would have probably fit better; on the contrary, a value too high would increase the risk to generate badly shaped quadrilaterals.

- ϵ_2: not to be confused with $\Delta\epsilon_2$, this is similar to ϵ_1 but for nodes that are in contact with a number of quadrilaterals that is smaller than or equal to four; the effects are similar to the ones of ϵ_1.

Proper examples of the effect on the resulting mesh will be given for each specific class, therefore motivating why certain values works better for certain geometries.

4.2.2 Axis-symmetric solids

This is probably the class of components that can be either the most complex or the easiest. This is due to the fact that basically we can have any kind of geometry, with a series of complex features. No matter how complex the domain gets, it should not be a problem for the tool as it relies on algorithms that very easily adapt to any shape and element's size.
The input parameters for this class is given in the form of two matrices:

- *Nodes matrix* N: this is an Nx2 matrix in which each row represents the x-y coordinates of each node on the domain's boundary.
- *Edges matrix* E: this is an Ex2 matrix in which each row represents an edge of the domain; in particular each row contains two indeces, which refer to the rows of the N matrix which contain the coordinates of the endpoints of the edge itself.

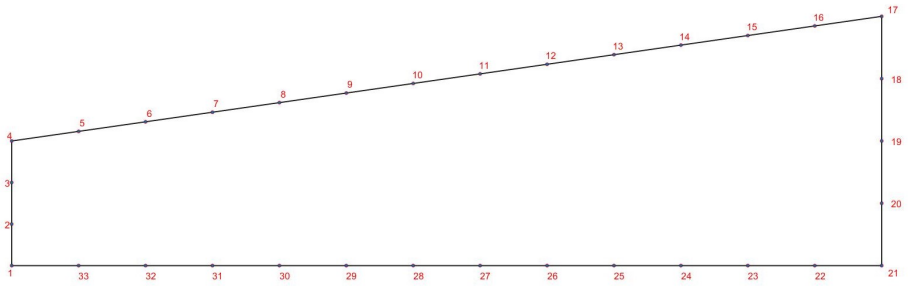
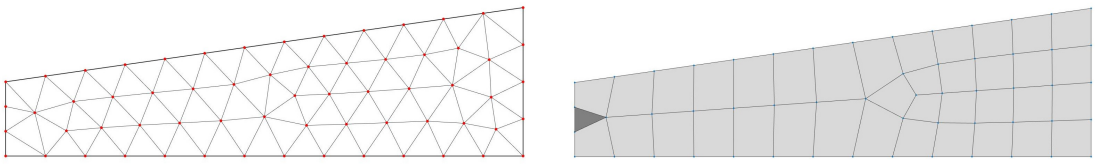


Figure 4.1: Input for an axis-symmetric solid

In Figure 4.1, a representation of the input matrices N and E can be observed.



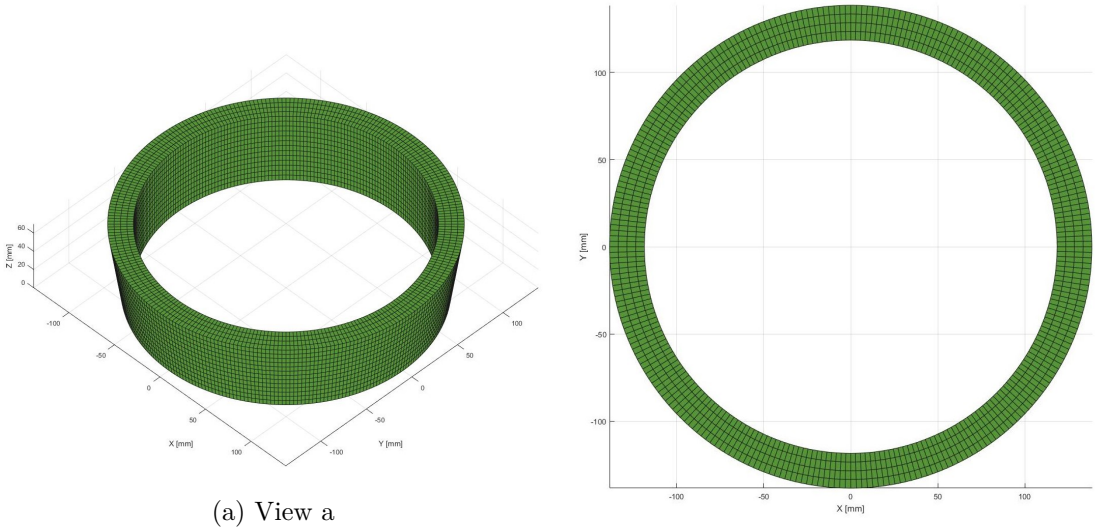
(a) After Delaunay

(b) After q-morph

Figure 4.2: 2D mesh of a generic axis-symmetric input

In Figure 4.2, the actual mesh-generating process is represented; in particular the effects of both main steps are clearly visible: first the Delaunay triangulation discretizes the domain into a series of triangles, as shown in Figure 4.2a, then the q-morph performs the transformations in order to obtain a quadrilateral mesh, as shown in Figure 4.2b. It is also worth noticing that the number of edges on the domain's boudnary is odd and, hence, a single triangle was generated as imposed by the algorithm's logic.
Once the two-dimensional domain is fully meshed, the solid is obtained by a simple rotation around the main axis. In order to define the angular spacing between the consecutive rotations of the original mesh, the following parameter is required:

- *Number of slices along the circumference* N_c: this defined in how many slices the components will be split; hence it defines how many times the original mesh will be repeated and the angular spacing between consecutive repetitions.
- *Twist angle* β: it might happen that the components is required to be merged with a gear that is either helicoidal or conical, and hence its profile needs to follow the twisted profile of the teeth. In order to do so, the angle β (in addition to two axial coordinates) is provided.

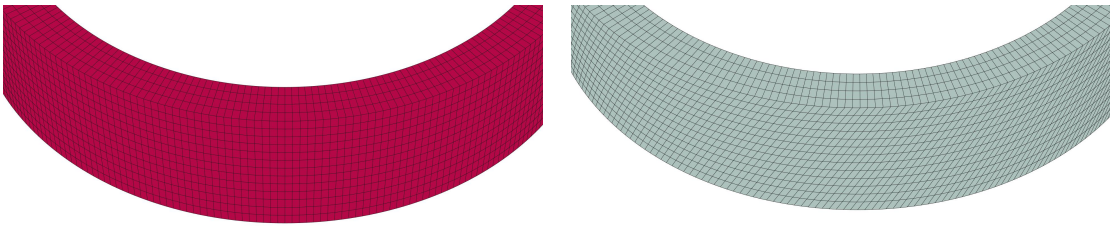


(a) View a

(b) View b

Figure 4.3: 3D mesh of an axis-symmetric solid

In Figure 4.4, the fully-meshed 3d component is shown in two different views. The following pictures show the effect of the input parameter β.



(a) $\beta = 0$ *deg* (b) $\beta = 15$ *deg*

Figure 4.4: Same solid with and without twist

4.2.3 Axial Holes pattern

Differently from a generic axis-symmetric solid, axial holes patterns rely on a more standard kind of geometry that can hence be obtained by a series of input parameters. In particular, the characteristics that are required for the definition of a geometry are:
Geometric parameters:

- *Thickness* t: this parameters determines the axial thickness of the component;
- *Internal radius* r_i *and external radius* r_o: considering the circular crown, this two parameters define its boundary;
- *Radial position of the hole* r_h: this parameter determines the circumference along which the centers of the holes lay on;
- *Radius of the holes* r_{hole}: determines the radius of the holes;
- *Number of holes* N_{hole}: this parameters defines how many holes will be placed on the circular crown; it also defines the angular region that will be meshed in 2D and, hence, how many times the feature will be repeated along the circular crown.

Mesh parameters:

- *Radial number of elements* N_r: this parameter defines how many elements will be placed on the boundary along the radius (i.e. on the segment connecting the inner circle with the outer circle);
- *Number of elements of the inner* N_{ri} *and external radius* N_{ro}: these two parameters define how many elements will be placed on the boundary, respectively on the inner and outer circumferences of the circular crown;

- *Number of elements on the holes* N_h: determines how many elements will be placed on the circumference of each hole;
- *Number of axial slices* N_t: the number of slices that the flange will be divided in, along its thickness.

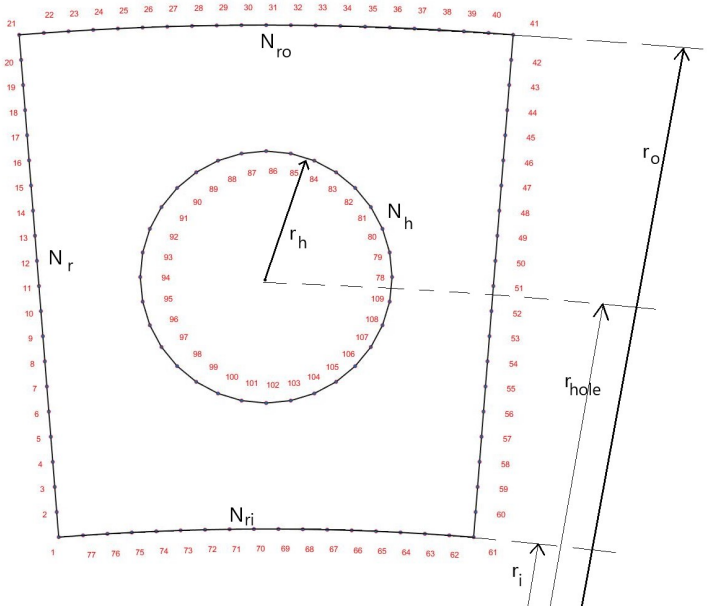
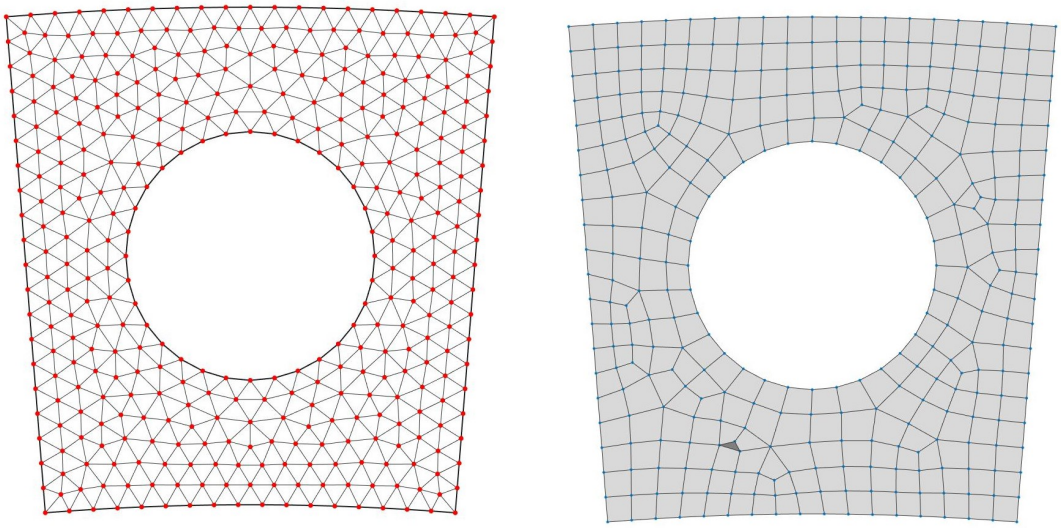


Figure 4.5: Input parameters of a flange

The flange is a very effective example that shows the actual potential of the 2D meshing algorithm; it is a domain that presents two initial boundaries, an internal and an external one, and it is managed with no problem by the q-morph.



(a) After Delaunay

(b) After q-morph

Figure 4.6: 2D mesh of a flange's portion

In Figure 4.6, the effect of the algorithm on the portion of a generic flange is observed. The triangular domain is shown in Figure 4.6a, while the final quadrilateral result is shown in Figure 4.6b. Also in this case, the initial number of edges on the boundary was odd and, hence, as a result a single triangle was generated.
Once the two-dimensional section is fully meshed, the transformation that is applied is nothing more than a simple extrusion. The 2D section is repeated for a number of times equal to N_t, each time opportunely spaced in order to cover the entire thickness of the flange.

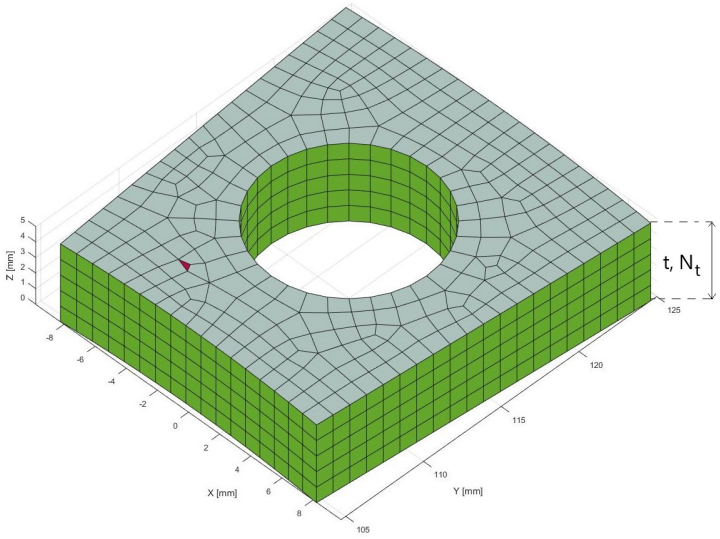
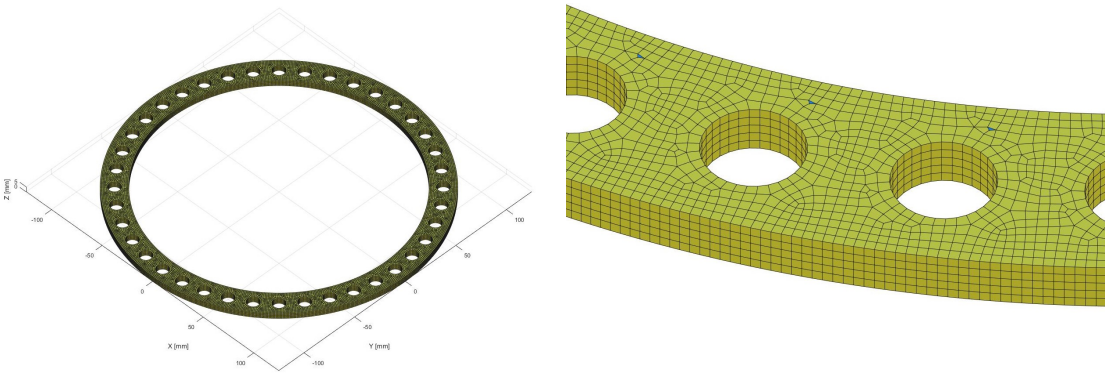


Figure 4.7: 3D mesh of the portion of a generic flange

Finally, in order to cover the whole circumference, the single feature is repeated following a circular pattern.



(a) View a (b) Particular

Figure 4.8: 3D mesh of a full flange

In Figure 4.8, the result of the full process to generate the mesh of a flange is shown. It is interesting to notice that the triangle, that was generated in the two-dimensional mesh elaboration, has now become a series of triangle-based prisms (highlighted in red in Figure 4.7, while in blue in Figure 4.8.
When applying the circular pattern, there is only one issue: the nodes at the interfaces overlap one with the other. This needs to be taken care of; however, since the mesh is an unstructured one, and nodes do not follow an predetermined order.

Fortunately it is well known where the duplicated nodes will end up in space and, hence, they will be opportunely selected by spacial filtering and deleted. Of course, it is also necessary that the deleted nodes are replaced by their original counterparts.

4.2.4 Radial holes pattern

Radial flanges are probably the most simple domain that the tool will have to deal with. They are similar to simple flanges, in the fact that they are geometries with holes, with the addition that the outer boundary is a simple rectangle and not a mix of straight lines and circumference arches.
The input parameter in order to fully define their geometry are:

- *Axial length* L: determines the
- *Internal* r_i *and external* r_o *radius*: They define the internal and external cylindrical surfaces that delimit the radial flange; their difference of course gives the radial thickness of the component;
- *Axial position of the holes* L_h: determines the axial distance of the circumference on which the centers of the holes lay;
- *Holes radius* r_h: determined the radius of each holes;
- *Number of holes* N_{hole}: defines the number of times that the single hole is repeated along the circumference; this also defines the angular region that the single feature occupies.

Mesh parameters:

- *Number of elements along the axial direction* N_L: this parameter defines how many elements will be placed at the interface between consecutive portions of the flange;
- *Number of circumference elements* N_c: defines how many elements will be a part of the circumference arches that delimit the flange along the axis;
- *Number of elements along the radius* N_r: defines in how many layers will be divided the flange, in the radial direction (similar to N_t for the simple flange);
- *Number of elements on the holes* N_h: defines in how many elements each single hole's circumference will be divided in.

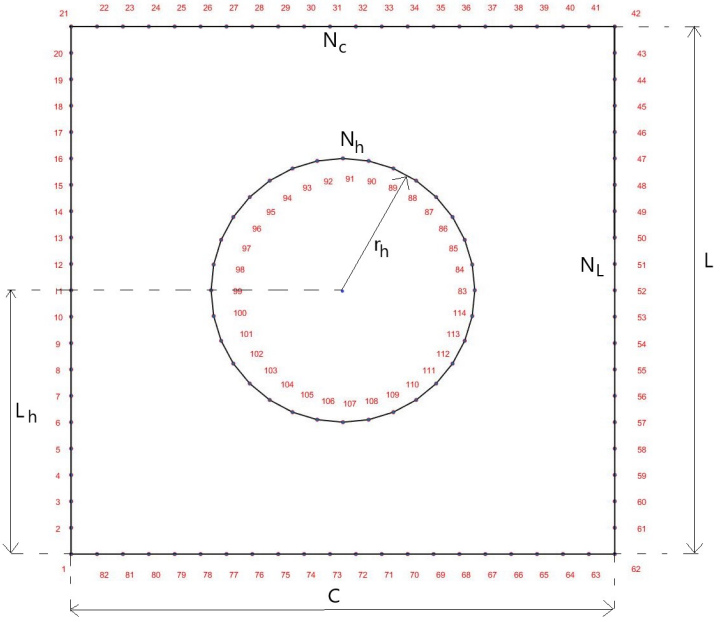
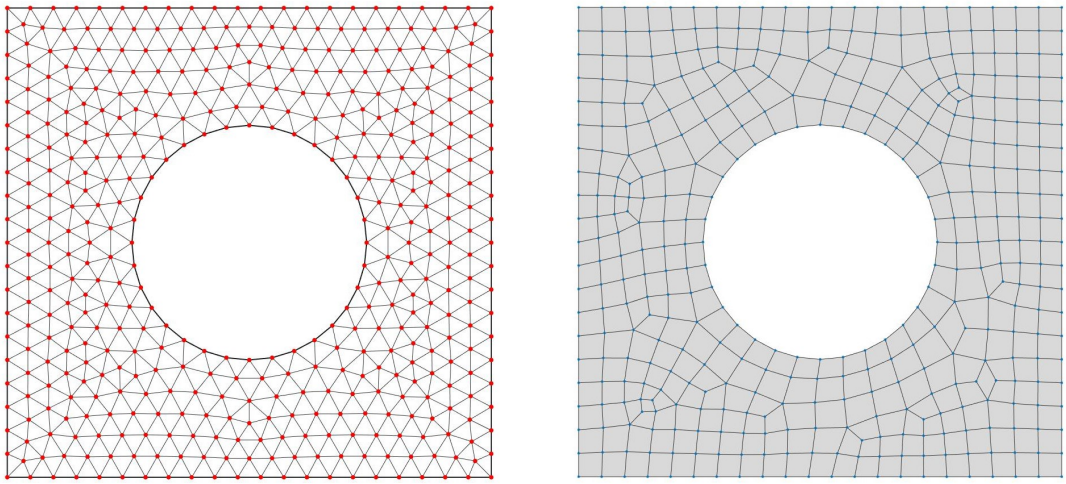


Figure 4.9: Input parameters of a radial flange

In Figure 4.9, a certain dimension is labeled as C, even thoud this is not an input parameter, it is derived frome them as:

$$C = \frac{2\pi}{N_{holes}} r_o \tag{4.1}$$

and it is the length of the arch of a single portion of the radial flange.



(a) After Delaunay

(b) After q-morph

Figure 4.10: 2D mesh of a radial flange's portion

In Figure 4.10, the final result of the two-dimensional meshing process is shown. As it can be noticed, the final mesh results to be quite regular, but it still presents some irregularities due to the different shape and number of edges on the two initial loops.

After the two-dimensional mesh is obtained, in order to transform it to the three-dimensional component, the following ordered set of geometric transformations is applied:

1. First a change of coordinates is performed in order to transform the flat domain to a portion of a cylindrical surface;
2. Then, points are projected towards the central axis; each projected set of nodes will be equally spaced along the radial direction in such a way to divide the whole thickness of the component according to the input parameter N_r;
3. The single portion is then repeated according to a circular pattern in order to cover the whole circumference.

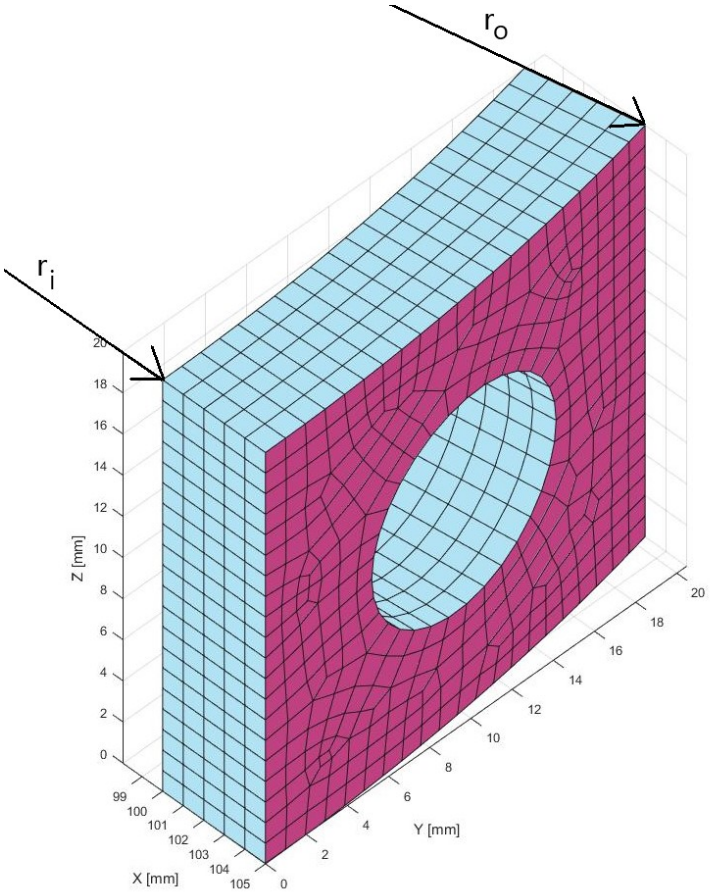
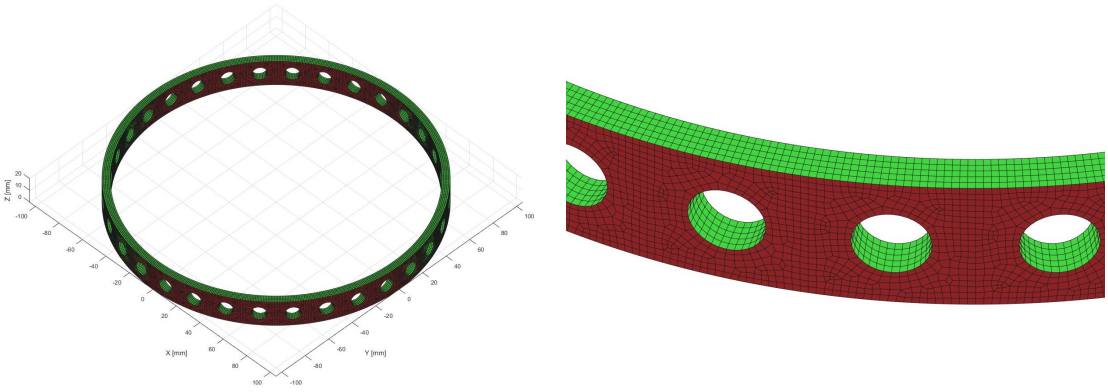


Figure 4.11: 3D mesh of the portion of a generic radial holes pattern



(a) View a (b) Particular

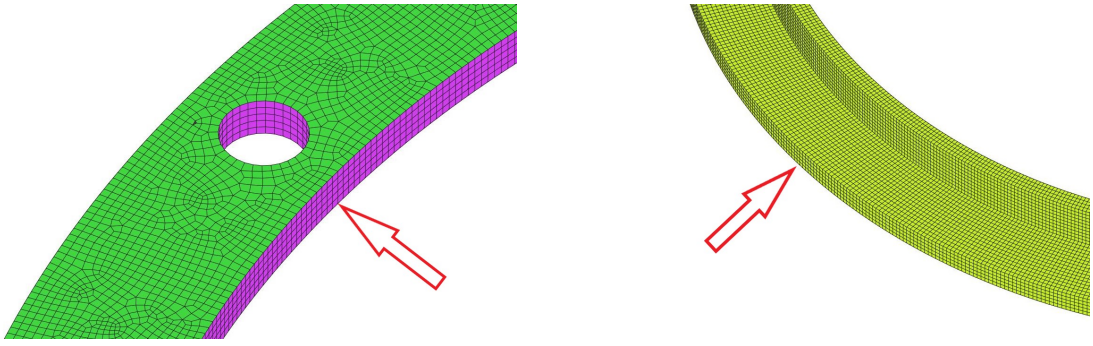
Figure 4.12: 3D mesh of a full radial holes pattern

4.3 Merging the parts

Once each component is fully meshed, they require to be bonded together. In order to achieve a proper merging of the different parts, it is crucial to provide the right information: a segment in space for each component must be defined so that the relative coordinates at which the bonding happens is determined; also the kind of bonding must be defined as we can have two different cases: equivalence or transition mesh. Figures will be shown for each specific example in order to make it clearer.

4.3.1 Equivalence

The term *equivalence* is used when referring to the process of deleting, or more properly merging, nodes that are within a certain tolerance distance one within the other in such a way that there are not overlapping or interfering geometries. When two parts are to be joined at a specific interface, there will be a whole set of nodes defined in both meshes, with a different index but with the same position in space. Duplicated nodes must hence be erased and the reference indices in both meshes must be properly updated.
As it was said before, since meshes are built according to a non-structured algorithm and hence nodes do not follow a pre-determined order, in order to properly identify overlapping nodes, without consuming too much time in the process, the nodes of both meshes are first filtered according to their position in space. In this way the amount of data to be processed is much liter with respect to the complete mesh. The overlapping nodes are then erased and references are updated.
In the following example, a flange and an axis-symmetric solid are to be merged: the merging surface of the flange is its internal cylindrical face, while the axis-symmetric solid will be merged on the most external cylindrical surface.



(a) Merging surface of the flange (b) Merging surface of the axis-symmetric

Figure 4.13: Reference surfaces for equivalence

The outcome of the operations is shown in Figure 4.16:

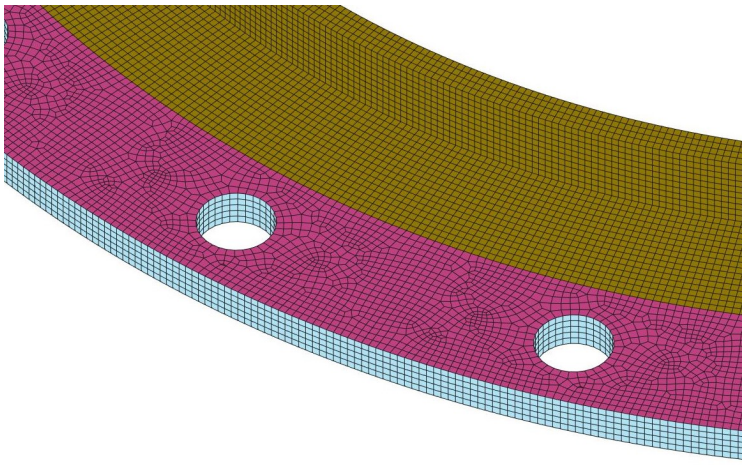


Figure 4.14: Equivalence outcome

In order to prove the effectiveness of the equivalence, the change in number of nodes is calculated and compared to the expected one. The feedback is also visual as the component does not present any flaws of gaps, even though a set of nodes has been erased from the mesh. This also proves that the updated references to the nodes' IDs was successful.

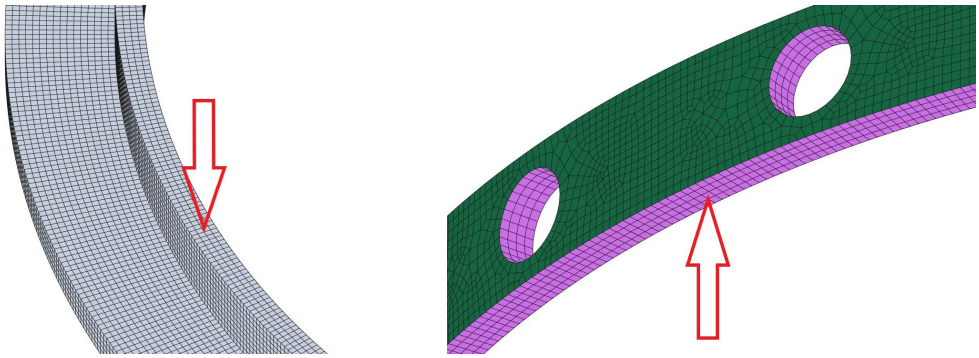
4.3.2 Transition mesh

The situation becomes more complex when a transition mesh is required. As it was said before, the solids that are being generated are either axis-symmetric or cyclic-symmetric; this means that each one of them will present a specific periodicity. As a result, it may happen sometimes (quite often actually) that the number of nodes along the circumference of a certain part differs from the one of the part it must be joined with. To solve this situation, a transition mesh is created.
Since the solution to the problem is not easy, not in a way to be universally valid at least, the following one was proposed as long as a specific condition is respected: it is required that the number of radial slices on the two portions to be joined is equal. In this way a strategy that is similar to the one used to generate the radial holes patterns is used:

1. First the number on nodes along both circumferences is calculated;
2. Then the greater common divider of the two numbers is found, so that the meshing algorithm is not applied to the whole surface but only to a small portions of it;
3. The surface to be meshed is built and then processed;

4. The mesh is then treated exactly as in the radial holes pattern: first there is a change in coordinates in order to adapt the planar domain to the original surface, then the two-dimensional mesh is projected in order to obtain the single three-dimensional portion of the transition mesh, and finally a circular pattern is applied in order to cover the whole circumference.

The risk of this solution is that it might happen that no common divider (other than one) is found and, hence, the whole cylindrical surface needs to undergo the 2D meshing algorithm. This would cause the total computational time to increase. However, since such a situation is quite rare, it is acceptable.



(a) Merging surface of the axis-symmetric (b) Merging surface of the radial holes

Figure 4.15: Reference surfaces for transition mesh

Once the parameters have been determined, the domain of the transition mesh can be assembled:

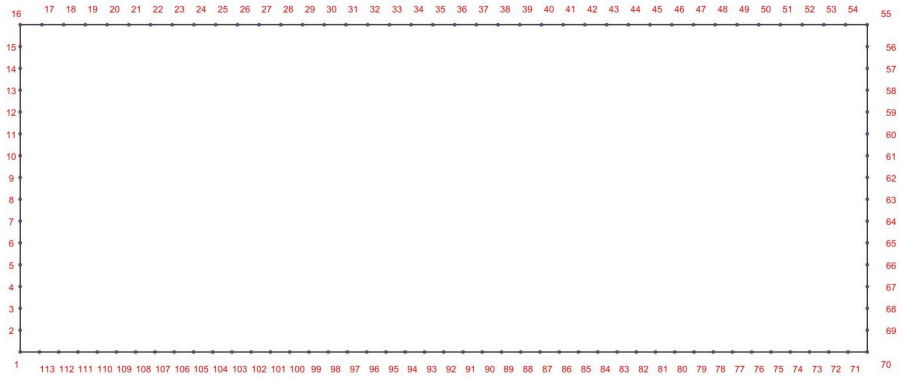
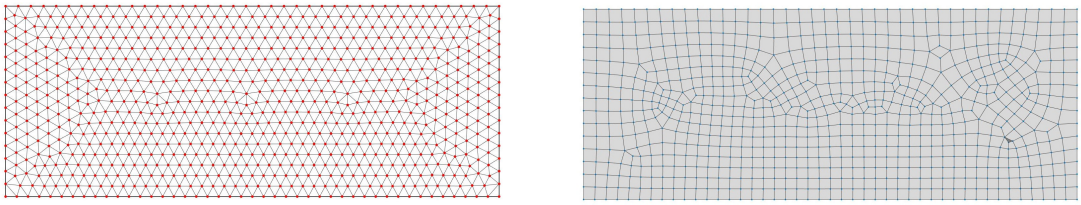


Figure 4.16: Domain of the transition mesh

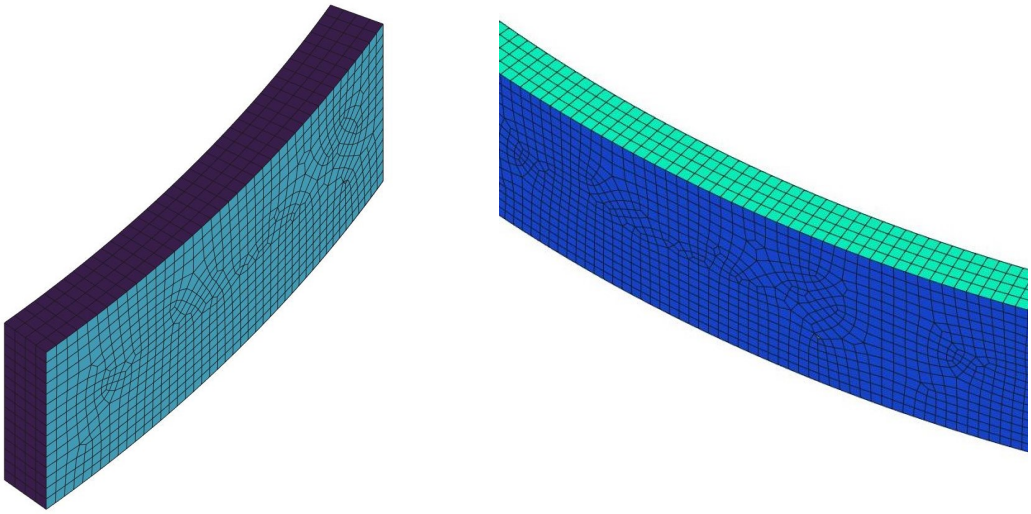
The domain is then processed as usual:



(a) After Delaunay (b) After q-morph

Figure 4.17: Processing the transition mesh

The 2D outcome results to be quite distorted and with a poor elements' quality; this is due to the fact that the upper and the lower edge are made by a different number of edges, while they have the same length.
Once the two-dimensional mesh is ready, the three-dimensional transformations are obtained, as well as the circular pattern.



(a) Single portion (b) Full transition mesh

Figure 4.18: Final transition mesh

Finally, the transition mesh is implemented in the geometry. The mesh is first properly moved in space in order to match the surfaces that it must join together. Then, the equivalence is applied on both surfaces in order to remove overlapping features ans avoid interference.

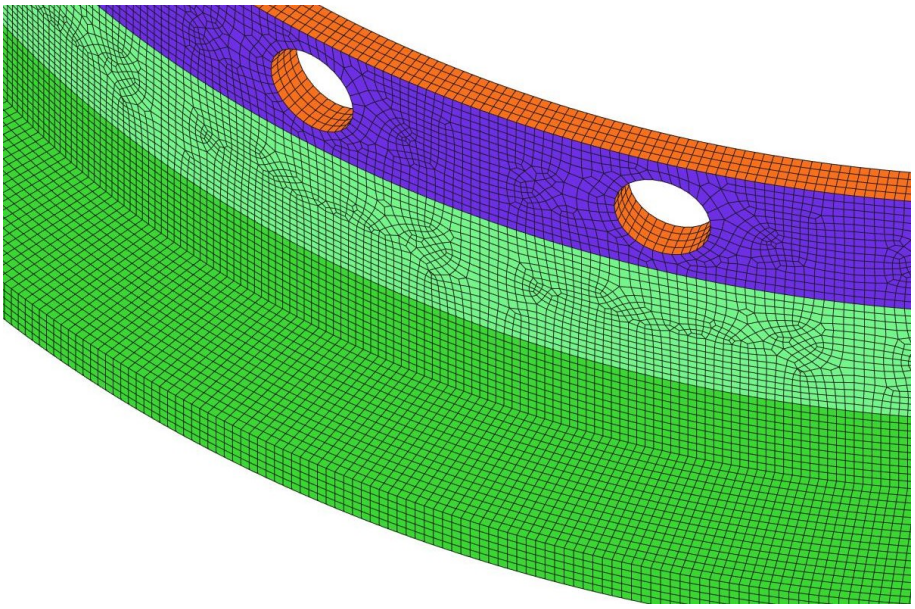


Figure 4.19: Implemented transition mesh

4.4 Test case

In order to prove the correct functioning of the tool, a simple test case was designed in such a way that each function was properly tested; the test case is made up by the following parts: a flange, a radial holes pattern, two axis-symmetric pieces, and a spur gear. The merging techniques are also tested: both equivalence and transition are introduced in the case. This test case also proves the correct interaction between the two distinct tools: the one that was developed for this thesis, and the one for gears' meshes.
In order to avoid repeating the previous steps, after the input parameters for each component will be provided, the geometry of each piece will be shown directly.

4.4.1 Flange

Table 4.1: Flange input parameters

Geometry parameters	
t $[mm]$	5
r_i $[mm]$	140
r_o $[mm]$	160
r_h $[mm]$	5
r_{hole} $[mm]$	150
N_{hole} [-]	20

Mesh parameters	
N_r [-]	20
N_{ri} [-]	44
N_{ro} [-]	51
N_h [-]	32
N_t [-]	5

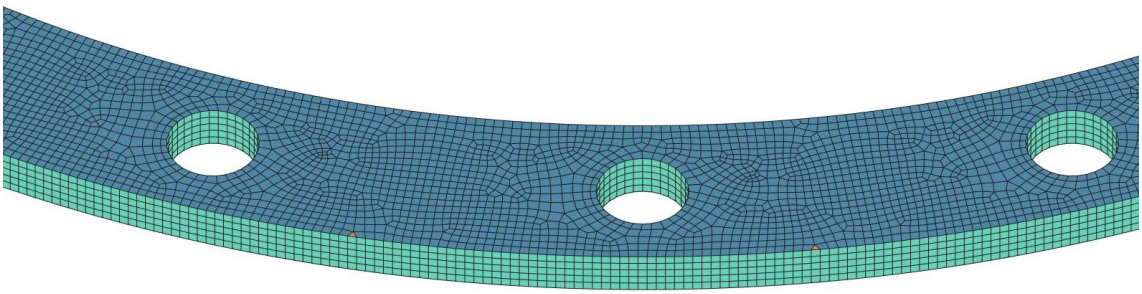


Figure 4.20: Flange - particular

4.4.2 Axis-symmetric 1

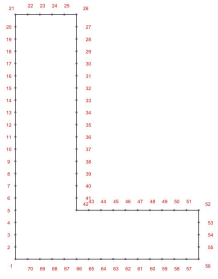


Figure 4.21: Axis-symmetryc 1

Table 4.2: Axis-symmetric 1 mesh input parameters

Mesh parameters	
N_c [-]	880
β $[deg]$	0

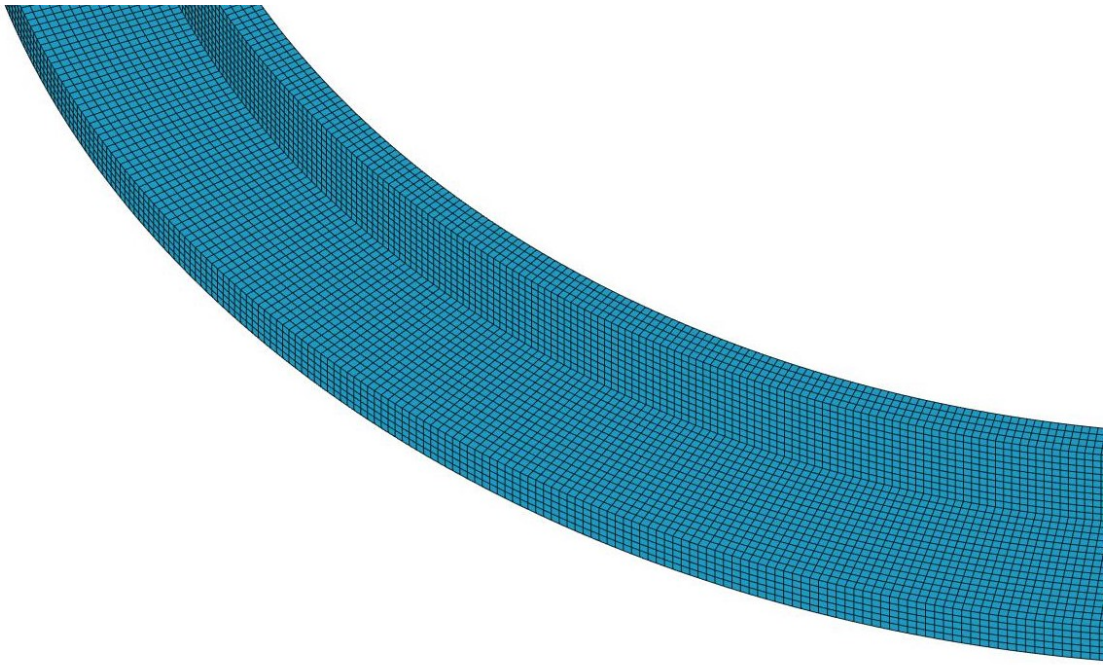


Figure 4.22: Axis-symmetric 1 - particular

4.4.3 Radial holes pattern

Table 4.3: Radial holes pattern input parameters

Geometry parameters	
L $[mm]$	15
r_i $[mm]$	120
r_o $[mm]$	124
L_h $[mm]$	7.5
r_h $[mm]$	5
N_{hole} [-]	20

Mesh parameters	
N_L [-]	15
N_c [-]	39
N_r [-]	5
N_h [-]	32

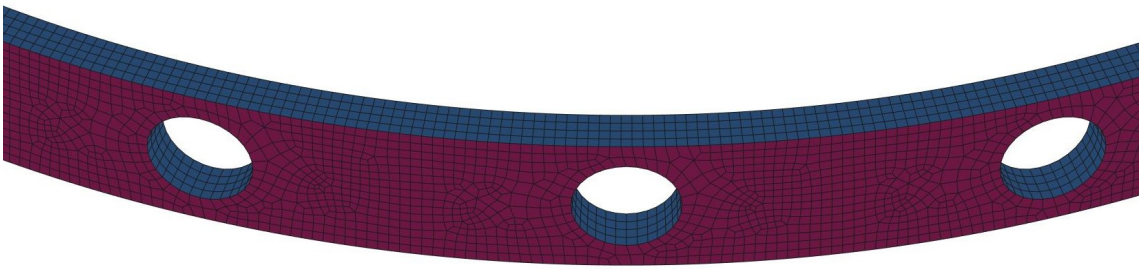


Figure 4.23: Radial holes pattern - particular

4.4.4 Axis-symmetric 2

Figure 4.24: Axis-symmetryc 2

Table 4.4: Axis-symmetric 2 mesh input parameters

Mesh parameters	
N_c [-]	880
β $[deg]$	0

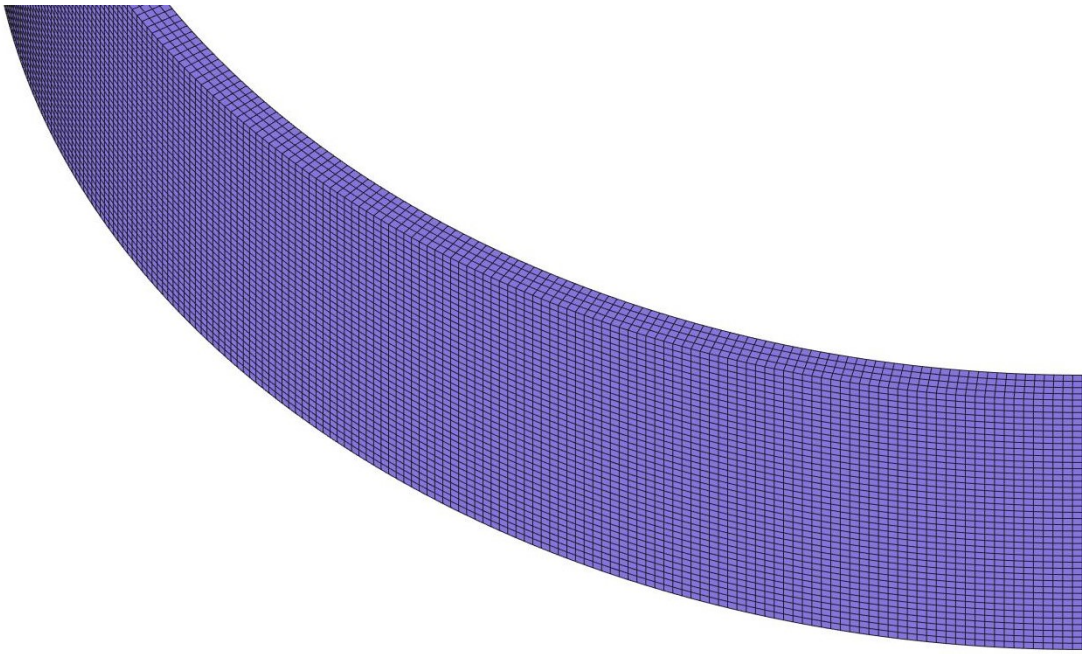


Figure 4.25: Axis-symmetric 2 - particular

4.4.5 Spur gear

Table 4.5: Spur gear input parameters

Geometry parameters	
z [-]	44
m_n $[mm]$	5
α_n $[deg]$	22.5
d_{tip} $[mm]$	210
d_{root} $[mm]$	230
$fillet$ [-]	full
t_p $[mm]$	7.785
w $[mm]$	40
d_{ie} $[mm]$	240

Mesh parameters	
$N_{profile}$ [-]	10
N_{fillet} [-]	8
$N_{rim-top}$ [-]	6
$N_{rim-bottom}$ [-]	4
N_{root} [-]	\
N_w [-]	40

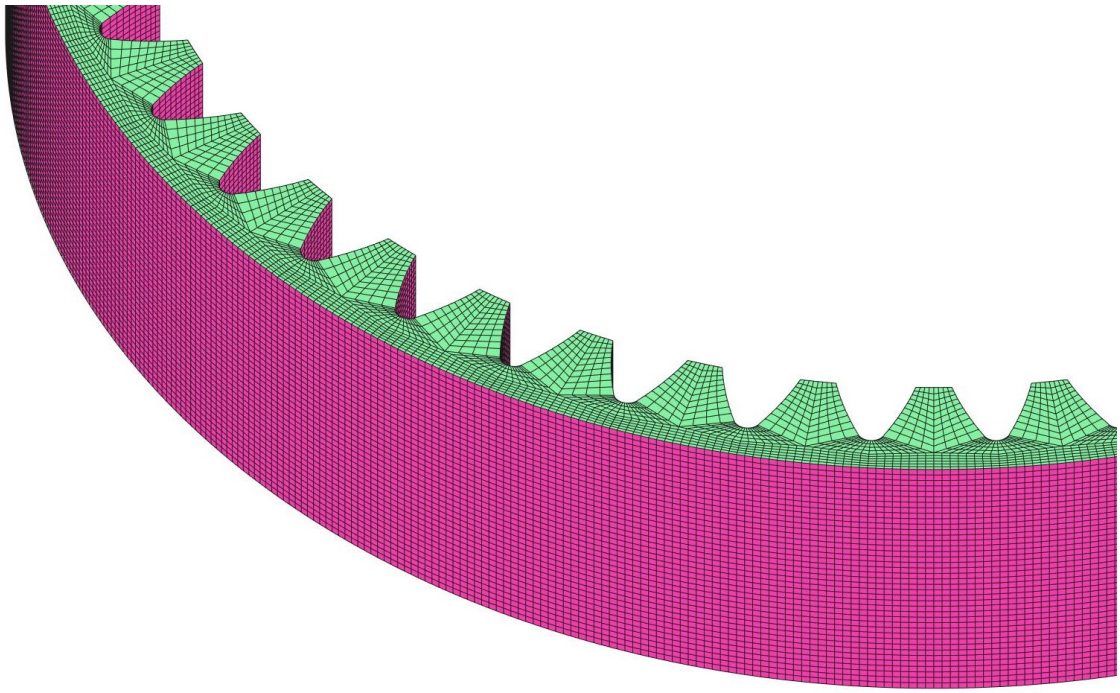


Figure 4.26: Spur gear - particular

4.4.6 Full component

Finally, the different parts were merged together by exploiting both equivalence and transition in the following order:

1. Flange - Axis-symmetric 1: *equivalence*
2. Axis-symmetric 1 - Radial holes pattern: *transition*
3. Radial holes pattern - Axis-symmetric 2: *transition*
4. Axis-symmetric 2 - Spur gear: *equivalence*

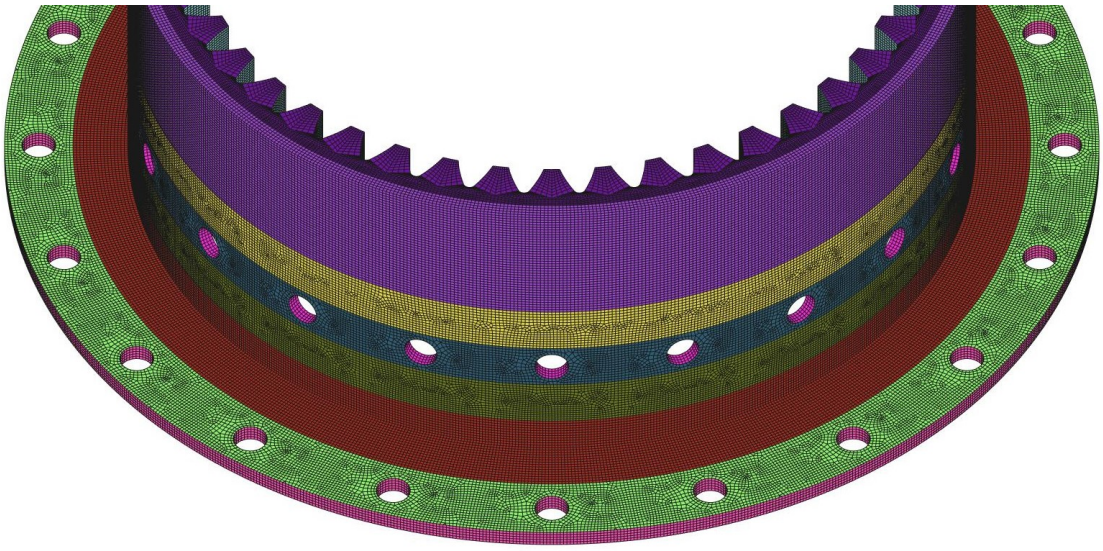


Figure 4.27: Fully-meshed test case - particular

The total elapsed time, from start to finish, was about 200 seconds (or 3 minutes and 20 seconds). This execution time was greatly influenced by the creation of the second transition mesh: due to a difference in periodicity between the radial holes pattern the axis-symmetric 2 a great portion of the cylindrical surface had to undergo the q-morph; this process alone took about 140 seconds (about 70% of the overall computational time).

Chapter 5

Results and conclusions

Two main aspects are of interest when describing the effectiveness of a meshing tool: mesh quality and computational time. The attention will also be focused, very rapidly, on the data conversion in such a way that the mesh can be used by FE analysis softwares.

5.1 Mesh quality

When evaluating the quality of a mesh, there is not a unique way to do it; there are different methods, each one as valid as the other. In this specific case, the distortion metric β will be adopted, as it provides useful information on both edges and angles of the elements. The metric is defined as follows: given a quadrilateral, four triangles can be formed by opportunely combining its four vertices; for the $i-th$ triangle with vertices A, B, and C the distortion metric α is calculated as follows:

$$\alpha_i = I2\sqrt{3}\frac{\|CA \times CB\|}{\|CA\|^2 + \|AB\|^2 + \|BC\|^2} \tag{5.1}$$

where

$$I = \begin{cases} 1 & \text{if the triangle is not inverted} \\ -1 & \text{if the triangle is inverted} \end{cases} \tag{5.2}$$

The distortion metric β is finally defined as:

$$\beta = \{min(\alpha_1, \alpha_2, \alpha_3, \alpha_4)\} \tag{5.3}$$

In this way, however, a perfect square would result in a distortion metric $\beta = \frac{\sqrt{3}}{2}$; in order to set a maximum scale equal to 1, the values are normalized in the following way:

$$\beta_n = 2\frac{\beta}{\sqrt{3}} \tag{5.4}$$

A value of β smaller than one would mean a quadrilateral with an angle greater than π. In order to provide a clear picture on the overall mesh, three values will be considered: maximum distortion metric β_{max}, minimum distortion metric β_{min}, and average distortion metric β_{avg}.

$$\beta_{max} = max\{\beta_j\} \tag{5.5}$$

$$\beta_{min} = min\{\beta_j\} \tag{5.6}$$

$$\beta_{avg} = \frac{\sum_{j=1}^{n} \beta_j}{n} \tag{5.7}$$

In particular, the attention will be focused on how the parameters that were introduced in Section 4.2.1 affect the mesh quality for each specific subclass. This will help understand why certain values were selected as the standard for the different geometries. Since, many tests were made in order to determine the best set of parameters for each class, only the final values will be shown, alongside with the resulting quality of the mesh.

5.1.1 Axis-symmetric

Since the geometry of these components does not follow any standard, the parameters were set in such a way to adapt the algorithm more easily and with little constraints:

Table 5.1: Process parameters and mesh quality for axis-symmetric solids

Process parameters	
β_{lim} $[deg]$	105
$\Delta\epsilon_1$ $[deg]$	35
$\Delta\epsilon_2$ $[deg]$	50
r_{max} [-]	2.5
ϵ_1 $[deg]$	25
ϵ_2 $[deg]$	35

Mesh quality	
β_{max} [-]	0.999
β_{min} [-]	0.423
β_{avg} [-]	0.958

5.1.2 Axial holes pattern

For axial holes pattern, the template domain always follows the same topology and, hence, is more predictable. A set of parameters which allowed the quads to follow the circular trajectory of the arches was selected.

Table 5.2: Process parameters and mesh quality for axial holes patterns

Process parameters	
β_{lim} $[deg]$	100
$\Delta\epsilon_1$ $[deg]$	20
$\Delta\epsilon_2$ $[deg]$	30
r_{max} [-]	2.5
ϵ_1 $[deg]$	30
ϵ_2 $[deg]$	45

Mesh quality	
β_{max} [-]	0.999
β_{min} [-]	0.314
β_{avg} [-]	0.929

5.1.3 Radial holes pattern

Similarly to axial holes, the domain has a predictable topology each time. In this case, however, since the outer loop is square-shaped, a set of parameters to ensure the selection of almost perfect quadrilateral was preferred.

Table 5.3: Process parameters and mesh quality for radial holes patterns

Process parameters	
β_{lim} $[deg]$	95
$\Delta\epsilon_1$ $[deg]$	30
$\Delta\epsilon_2$ $[deg]$	45
r_{max} [-]	2.5
ϵ_1 $[deg]$	20
ϵ_2 $[deg]$	30

Mesh quality	
β_{max} [-]	0.999
β_{min} [-]	0.400
β_{avg} [-]	0.919

5.1.4 Transition mesh

Also in this case the pattern is very predictable, as the shape of the domain can wither be a rectangle or an unrolled cone. However, in the latter case, the cone is shaped in such a way that, when unrolled, the circular arches have a long radius and, hence, they are almost flat. Due to this reasons, a set of parameters to ensure square elements was selected, with the freedom required to compensate the uneven number of nodes.

Table 5.4: Process parameters and mesh quality for transition meshes

Process parameters	
β_{lim} $[deg]$	95
$\Delta\epsilon_1$ $[deg]$	35
$\Delta\epsilon_2$ $[deg]$	50
r_{max} [-]	2.5
ϵ_1 $[deg]$	25
ϵ_2 $[deg]$	35

Mesh quality	
β_{max} [-]	0.997
β_{min} [-]	0.640
β_{avg} [-]	0.959

5.2 Computational time

In this section, the attention will be focused on how the computational time is affected by the different input parameters; as it can be easily guessed, the main factor is the number of elements of the mesh. In particular, the parameter that will be correlated with computational time is:

- *Number of triangles in the starting mesh* N_{tria}

Since the process that takes up most of the time is the implementation of the q-morph algorithm (whether it is for an input component or a transition mesh), the time required by this algorithm will be the base for time efficiency considerations. Since, in addition, the only mesh characteristic that affects the computational time is the number of starting triangles, and not the shape of the domain, a simple square domain will be considered as the starting point; the mesh refinement will be increased step by step, and the elapsed time recorded.
The results are summed up in the following figure:

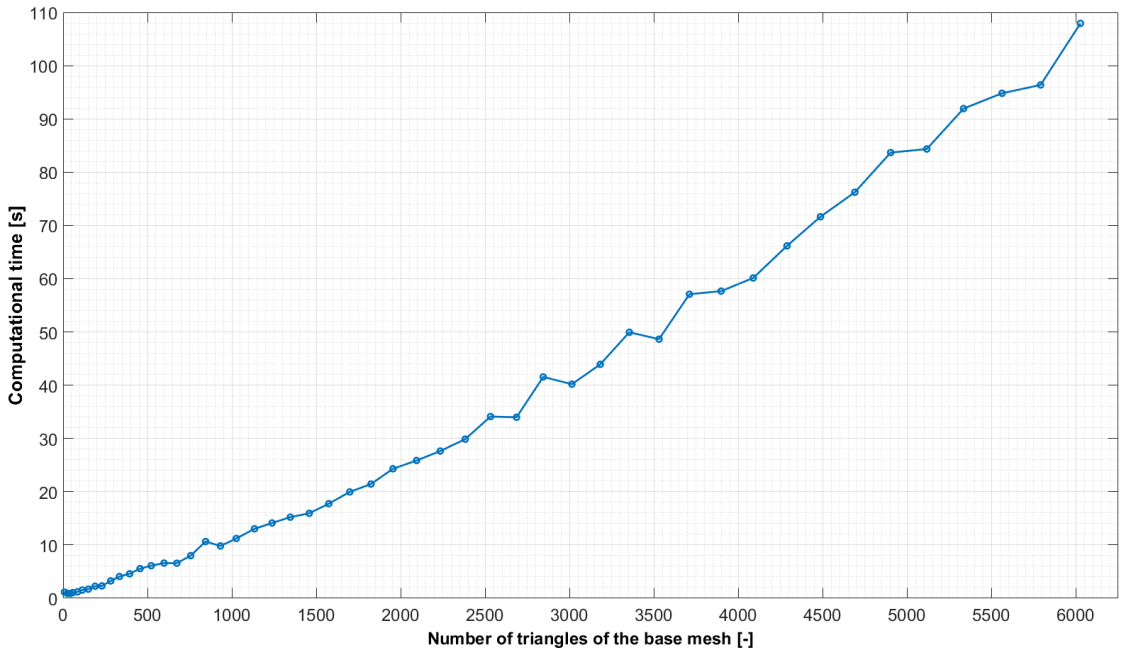


Figure 5.1: Performance

The computational time increases in an almost linear way as the number of elements rise. It might seem that the process becomes way too long, especially for a number of elements greater than 1500, as it takes more than 20 seconds for it to be completed. However, it must be remembered that the real strength of this tool is the fact that it relies on the symmetry of the components: considering the test case presented in Section 4.4, the sections of the different components did not exceed the 2000 triangles in the starting mesh; the real problem was represented by the transition mesh, which required a mesh of about 6000 triangles, as the parts to be connected did not present very compatible periodicity.

5.3 Conclusions

The application of the tool proved to be successful in different test cases. There is still room for improvement, especially regarding the qmorph performance. It represents, however, a valid and solid meshing tool and, with the proper modifications, it can be applied to a way broader range of possibilities.

Bibliography

[1] Steven J Owen et al. "Q-Morph: an indirect approach to advancing front quad meshing". In: *International Journal for Numerical Methods in Engineering* 44.9 (1999), pp. 1317–1340.

[2] Ted D Blacker and Michael B Stephenson. "Paving: A new approach to automated quadrilateral mesh generation". In: *International Journal for Numerical Methods in Engineering* 32.4 (1991), pp. 811–847.

[3] Scott A Canann, Joseph R Tristano, Matthew L Staten, et al. "An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral, and Quad-Dominant Meshes." In: *IMR*. Citeseer. 1998, pp. 479–494.

[4] Jörg-Rüdiger Sack and Jorge Urrutia. *Handbook of computational geometry*. Elsevier, 1999.

[5] Paul Kinney. "Cleanup: Improving quadrilateral finite element meshes". In: *6th International Meshing Roundtable*. 1997, pp. 437–447.

[6] Scott A Canann, SN Muthukrishnan, and RK Phillips. "Topological improvement procedures for quadrilateral finite element meshes". In: *Engineering with Computers* 14.2 (1998), pp. 168–177.