UCLouvain

École polytechnique de Louvain

Non-orthogonal cross fields for quadrilateral meshing

Author: **Loïs Dermine**
Supervisor: **Jean-François Remacle**
Readers: **Mattéo Couplet, Alexandre Chemin, Estelle Massart**
Academic year 2023–2024
Master [120] in Mathematical Engineering

Abstract

During this master thesis, we study the problem of generating quad-meshes for geometries with very acute or obtuse angles. The aim is to avoid irregular nodes on the corners of the geometry and also to minimise the number of irregular nodes. We will attempt an approach using non-orthogonal crosses that is strongly inspired by *Desobry et al* (2021). We begin by showing different approaches to generating quadrilateral meshes. We will also introduce 2 models using frame fields. We will see the advantages and disadvantages of these approaches. We will continue by detailing the methodology and the adaptations made in comparison with *Desobry et al* (2021). Then, we will present the results obtained by specifying the parameters used. We conclude by summarising what was explained during the master thesis and by detailing the limitations of this approach.

Contents

Chapter 1

Introduction

Meshes are one of the essential tools for engineers. Indeed, when it comes to simulations, their use is mandatory. We often think of the fields of mechanical engineering and civil engineering, for the deformation of metal, plastic and concrete structures, etc. Automotive and aerospace engineering also come to mind, using meshes for airflow simulations (see Figure 1.1). But other sectors also use meshes in their work. Electrical engineers use meshes to analyse electrical components and model complex circuits. Biomedical engineers need to be able to model biological tissues and implants, as well as simulate blood flow to study some diseases.

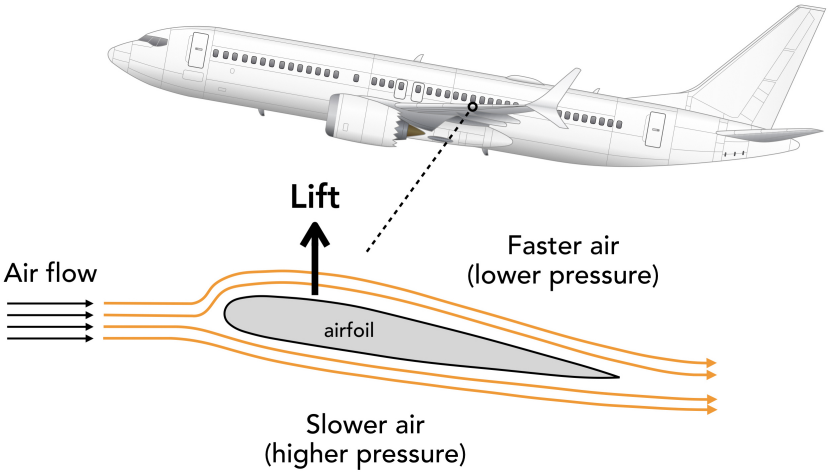


Figure 1.1: Airflow on an airfoil (© Dimitrios Karamitros)

When it comes to the use of meshes, it is essential to generate meshes that will produce the most accurate results possible. The size, shape and positioning in the geometry of the elements used are all important. A mesh with very small elements will generally be more accurate but will require longer calculation time. An element positioned strategically at the boundary of the geometry may lead to irregular nodes within it. If the elements are triangular or quadrilateral in the case of a plane or surface, or tetrahedral or hexahedral in the case of a volume, this will have an impact not only on the accuracy of the calculations, but also on the speed with which they are obtained and the amount of resources used by the computer. All these factors need to be taken into account, so it is all a question of compromise in the choices made. The aim is to move towards meshing techniques that give us the greatest possible accuracy in the results calculated from them. It is therefore normal to see a large number of different techniques that have been

developed over the years. However, it is often more difficult to generate quadrilateral and hexahedral meshes.
In this report, we focus on meshes using 2D elements, more specifically the use of quadrilateral elements on geometries with very acute and obtuse angles.
To return to the sectors mentioned earlier, this type of geometry corresponds, for example, to transistors, an aeroplane wing, a car bumper, turbine blades and so on. So once again we see the importance of the problem we are facing.
Using quadrilaterals to create meshes is not easy. It is more difficult to adjust the size and angles of quadrilateral elements. Having an extra node per element gives less flexibility in the arrangement of elements and makes it harder to adapt to different geometries. However, the use of quadrilateral elements can be preferred.

1.1 Triangle VS quadrilateral meshes

There are several techniques for generating triangular meshes on 2D planes and on surfaces of 3D geometries, so it is interesting to understand why a quadrilateral mesh has advantages, as it is more difficult to create an algorithm that generates quadrilateral meshes for any type of 2D geometry.
Firstly, for the same geometry and a fixed number of nodes, if we consider a quadrilateral mesh, its number of edges will be lower than on a regular triangular mesh. In fact, a quadrilateral will generally have 4 nodes and 4 edges, whereas the combination of 2 triangles has 4 nodes and 5 edges. More formally, using the Euler characteristic, which will be introduced later in the paper, we have

$$n - n_e + n_f = \chi \tag{1.1}$$

where n, n_e and n_f are respectively the number of nodes, edges and faces.
In the case of a triangular mesh, there are 3 edges per triangle, each belonging to 2 triangles, so we have $n_e = \frac{3}{2}n_f$. This gives

$$n = \frac{3}{2}n_f - n_f + \chi \approx \frac{1}{2}n_f \quad \Leftrightarrow \quad n_f \approx 2n \text{ and } n_e \approx 3n \tag{1.2}$$

In a quadrangulation, there are 4 edges per quadrilateral, each belonging to 2 quadrilaterals, so we have $n_e = 2n_f$. This gives

$$n = 2n_f - n_f + \chi \approx n_f \quad \Leftrightarrow \quad n_f \approx n \text{ and } n_e \approx 2n \tag{1.3}$$

We can consider $\chi \approx 0$ because it does not vary as a function of the number of nodes. As a result, its value is a constant which can be neglected for a large number of nodes. This shows us that quadrilateral meshes require fewer elements in their structure, and will therefore be more efficient in terms of computing speed.
This is the first comparison we can make, and the rest of this section aims to illustrate others.

1.1.1 Zlamal's minimum angle condition

The result introduced below was found by *Zlámal* (1968, [13] and [5]).
Let Ω be the representation of the domain of a geometry. Consider the equation :

$$-\sum_{i,j=1}^{2} \frac{\partial}{\partial x_i}\left(a_{ij}\frac{\partial u}{\partial x_j}\right) + cu = f \qquad \text{with } u = 0 \text{ on } \partial\Omega \tag{1.4}$$

where a_{ij}, c and f are continuous functions.
This corresponds to a generalised second-order elliptic equation, where the first-order terms are zero. The Laplace, Poisson and Helmholtz equations correspond to this scheme.
The aim is to find an error bound on the numerical solution. The numerical solution to the current problem is denoted by u^h.
Firstly, we will show that u^h is the closest solution to the analytical solution in a Sobolev space.
Secondly, we will work with a single triangle, and show that, for a continuous function with zeros on the edges of this triangle, its partial derivatives are bounded by its smallest angle.
Finally, these 2 results will be combined to show that the error of the numerical solution is bounded by the smallest angle of the set of triangles in a mesh.
We introduce a function $\tilde{u}$ belongs to H_2, a subspace of $W_2^{(1)}$, which is a Sobolev space. $\tilde{u}$ has the same values at nodes as u. *Zlamal* proved that :

$$D(u^h - u) \leq D(u - \tilde{u}) \tag{1.5}$$

where $D(z) = D(z, z)$ which is given by

$$D(\varphi, \psi) = \int_\Omega \left[\sum_{i,j=1}^{2} a_{ij} \frac{\partial \varphi}{\partial x_i} \frac{\partial \psi}{\partial x_j} + c\varphi\psi \right] dx \tag{1.6}$$

By Theorem 1 of [13], we know that for a function $\varphi(x_1, x_2)$ continuous on a closed triangle $\overline{T}$ and has bounded third-order derivatives on the interior of the triangle T, we have

$$|D^i \varphi(x_1, x_2)| \leq M_3 \qquad \text{with } |i| = 3 \tag{1.7}$$

with

$$D^i u = \frac{\partial^{|i|} u}{\partial x_1^{i_1} \partial x_2^{i_2}} \qquad \text{with } i = (i_1, i_2) \text{ and } |i| = i_1 + i_2 \tag{1.8}$$

Let the function φ be a function which cancels at the vertices, and at the midpoints of the edges of the triangle. Then the following inequality is valid on $\overline{T}$:

$$\left| \frac{\partial \varphi(x_1, x_2)}{\partial x_j} \right| \leq \frac{2}{\sin(\alpha)} M_3 c^2, \quad j = 1, 2, \quad |\varphi(x_1, x_2)| \leq M_3 c^3, \tag{1.9}$$

where α and c are respectively the smallest angle and the largest side of the triangle.
We apply the theorem described above to each of the triangles in the mesh with $\varphi = u - \tilde{u}$. $u(x_1, x_2)$ is a function with third-order derivatives which are bounded by M_3. This gives

$$[D(u - \tilde{u})]^{\frac{1}{2}} \leq \frac{C}{\sin(\theta)} M_3 h^2 \tag{1.10}$$

where θ and h are respectively the smallest angle and the largest side of all the triangles of the mesh. C is a constant that does not depend on the triangulation.

Combining results (1.5) and (1.10) with Friedrichs' inequality, *Zlámal* concludes that the numerical error is given by

$$\left|\left| u - u^h \right|\right|_{W_2^{(1)}} \leq \frac{C}{\sin(\theta)} M_3 h^2 \tag{1.11}$$

We work with geometries that have very acute and very obtuse angles, which will encourage the creation of triangular elements that also have very acute and very obtuse angles. Therefore the bound on the numerical error will be high in our case.
We know that this proof is only valid for triangular meshes. However, we make the hypothesis that quadrilateral elements will obtain a better bound because they favour orthogonality.

1.1.2 Accuracy of solutions

For this section, we will assume that we can project a result obtained for tetrahedra and hexahedra onto triangles and quadrilaterals. The Figure 1.2 offers us several comparisons between the errors in displacement and stress of a beam according to the elements used for the mesh. LT and LH stand for Linear Tetrahedra and Linear Hexahedra respectively.

Bending $\nu = 0.3$ Displacement			Bending $\nu = 0.3$ Stress		
DOF	LH	LT	DOF	LH	LT
567	0.72%		567	0.00%	
666		31.48%	666		21.23%
3075	0.08%		3075	0.00%	
3615		10.48%	3615		21.00%
Bending $\nu = 0.49$ Displacement			Bending $\nu = 0.49$ Stress		
DOF	LH	LT	DOF	LH	LT
567	6.56%		567	0.01%	
666		71.68%	666		66.77%
3075	3.20%		3075	0.01%	
3615		44.80%	3615		35.23%

(a) Bending model

Torsion $\nu = 0.3$ Displacement			Torsion $\nu = 0.3$ Stress		
DOF	LH	LT	DOF	LH	LT
567	15.65%		567	37.59%	
666		50.81%	666		77.82%
3075	5.26%		3075	8.59%	
3615		22.39%	3615		38.40%
Torsion $\nu = 0.49$ Displacement			Torsion $\nu = 0.49$ Stress		
DOF	LH	LT	DOF	LH	LT
567	26.41%		567	26.41%	
666		68.80%	666		68.80%
3075	5.44%		3075	5.44%	
3615		52.72%	3615		52.72%

(b) Torsion model

Figure 1.2: Error in displacement and stress at the reference position [2]

We can see that for a greater number of degrees of freedom in a triangular mesh than in a quadrilateral mesh, the accuracy of the result will be less good. When we look at the 2 tables, we can see that the difference between the errors of the 2 elements is not

negligible.
Therefore quadrilateral elements may be preferred because they are more precise for the same number of degrees of freedom.

1.2 Frame fields

A frame field, in the sense of *Panozzo et al* (2014, [10]), corresponds to the set of frames located on a domain.
A frame is a pair of tangent vectors to the domain, which have a certain length and angle with respect to a defined orthogonal basis, and their opposite vectors. The pair of two opposite vectors is called a direction. In this case, the domain is the geometry we want to mesh.

1.2.1 Cross fields

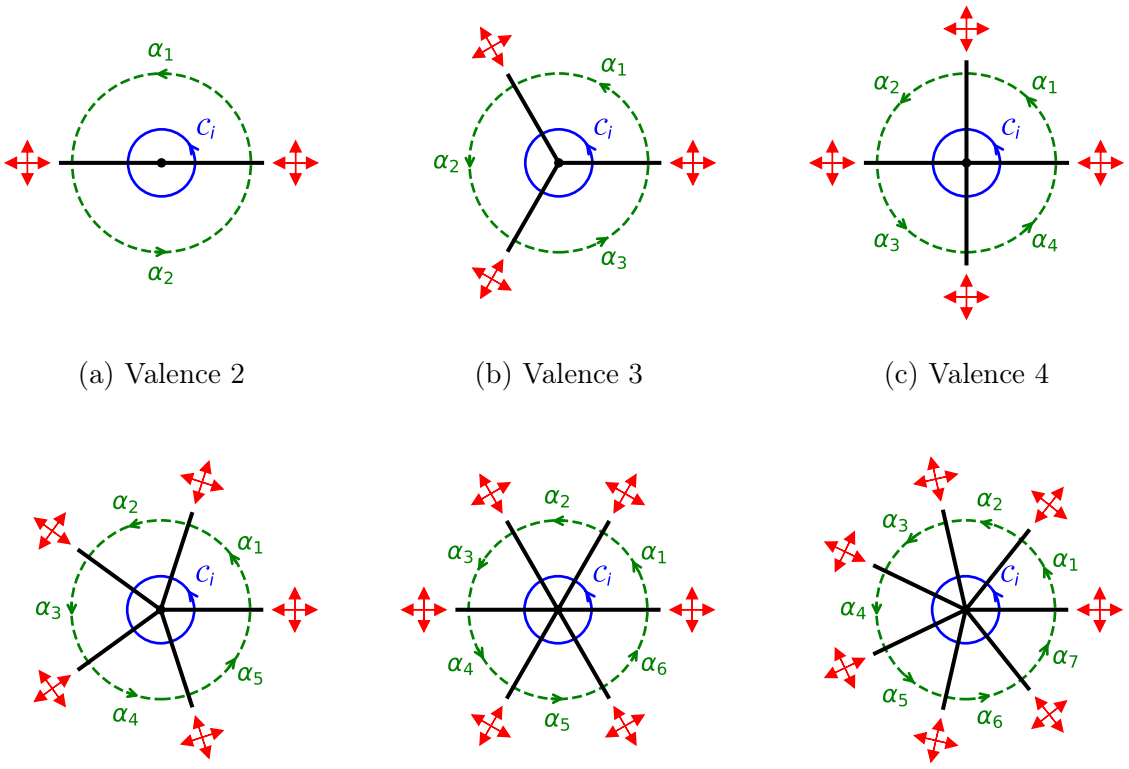
Approaches which generate quadrilateral meshes will generally use a more specific type of frame. These are crosses, which are frames whose vectors are orthogonal to each other and have a length of 1.
This type of frame is used, for example, in the *Ginzburg-Landau* method ([1] and [12]), which will be introduced later in this paper.
Cross fields are interesting because they can be transformed into quadrangulations using methods such as *Mixed-Integer Quadrangulation (MIQ)* proposed by *Bommes et al* (2009, [4]).

1.2.2 Non-orthogonal cross fields

For simple geometries, i.e. corners with angles close to 90° and 270°, orthogonal crosses will be used because we want to get as close as possible to square elements. In the current study, the geometries are more complex, so we will introduce non-orthogonal crosses.
In the sense of *Liu et al* (2011, [8]), non-orthogonal crosses are a generalisation of crosses for which orthogonality is not mandatory.
As we shall see later, this will allow us to impose quadrilaterals we want at the corners and then be able to smooth along the edges, which is not possible with orthogonal crosses.

1.3 Singularities

In the sense of *Panozzo et al*, a singularity of a frame field is a point that diverts the flow of tangential directions.
When a frame field is transformed into a quad-mesh, these points will create irregular nodes. These are nodes that do not have the expected valence.
For a quadrangulation, a node inside the geometry should ideally have a valence of 4, i.e. it should be adjacent to 4 quadrilaterals. In the case of a boundary node, the ideal is to have a valence of 2.
The valence of a node can be determined by the positioning of a set of frames. The Figure 1.3 shows this in the case of a cross field.



(a) Valence 2 (b) Valence 3 (c) Valence 4

(d) Valence 5 (e) Valence 6 (f) Valence 7

Figure 1.3: Representation of the valence of different nodes ([1])

1.3.1 Euler characteristic

Beaufort et al (2017, [1]) give us a detailed explanation of how to relate a geometry to the number of irregular nodes we will have when meshing that geometry.

Let's consider an orientable surface $\mathcal{S}$ in $\mathbb{R}^3$. The Euler characteristic for this surface is given by

$$\chi = 2 - 2g - b \tag{1.12}$$

where g is the genus of the surface. It can be seen as the number of "tunnels" through the surface, or the maximum number of cuttings along non-intersecting closed curves that won't make the surface disconnected. b is the number of components of the boundary $\partial\mathcal{S}$.
Here are some example of Euler characteristics :

- Disk ($g = 0$ and $b = 1$) : We have $\chi = 1$.
- Sphere ($g = 0$ and $b = 0$) : We have $\chi = 2$.
- Torus ($g = 1$ and $b = 0$) : We have $\chi = 0$.

The torus is the only closed surface with a zero Euler characteristic.
If we have a mesh on $\mathcal{S}$, the Euler characteristic links the number of nodes (n), the number of edges (n_e) and the number of faces (n_f). This equality is valid for both triangular and quadrilateral meshes.

$$\chi = n - n_e + n_f \tag{1.13}$$

This number was originally introduced for polyhedra, which all have a value of 2 as the Euler characteristic.
For a quadrangulation, *Beaufort et al* provides a link between this characteristic and the number of irregular internal nodes (n_k) and the number of irregular boundary nodes (n_{bk}). This gives us

$$\chi = \sum_k \frac{k}{4}(n_k + n_{bk}) \tag{1.14}$$

where k is an integer related to the valence of the nodes. For an internal node, we have a valence $v = 4 - k$ and for boundary nodes, we have $v = 2 - k$. We can therefore see that only irregular nodes will affect the value of the Euler characteristic.
We will use the equation (1.14) to judge the quality of our results.
For example, if we return to the case of the torus, we can conclude that the torus is the only closed surface that can be meshed without any irregular nodes.

Example with a disk

For a given geometry, the value of the Euler characteristic does not depend on the mesh itself. However, it does tell us how many irregular nodes to expect.
Let's take the example of a disk with a very simple triangulation (see Figure 1.4).

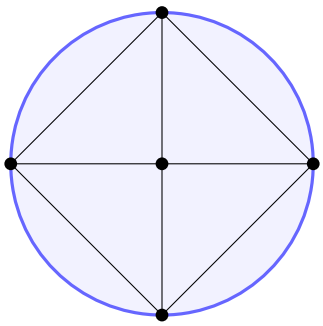


Figure 1.4: Basic quad-mesh of a circle

We can therefore see that in this case, we have $\chi = 5 - 8 + 4 = 1$. This result corresponds to the one found using the topological definition of the Euler characteristic that was firstly introduced.
We also know that $k \in \mathbb{Z}$ but quad-meshes generation models are designed to have $-1, 0$ and 1 as values most of the time. This means that terms of the equation (1.14) will most of the time have $\frac{-1}{4}, 0$ and $\frac{1}{4}$ as values. We therefore need at least 4 internal nodes with a valence of 3 or boundary nodes with a valence of 1.
Figure 1.5 shows 2 possible meshes configured in the best way.

1.4 Plan of the report

In this report, we will explain how the thesis was carried out. Firstly, we will look at the quadrilateral meshing techniques that have already been implemented. We will see why they are not perfect.
Next, we will describe the methodology that has been used, which is strongly inspired by *Desobry et al* (2021, [6]).

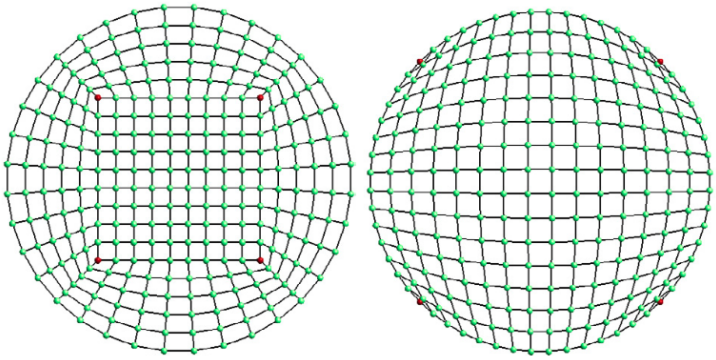


Figure 1.5: 2 differents meshes with 4 irregular nodes (red) of a disk [1]

After that, a section will be devoted to the results obtained.
The report will end with a conclusion of the thesis.

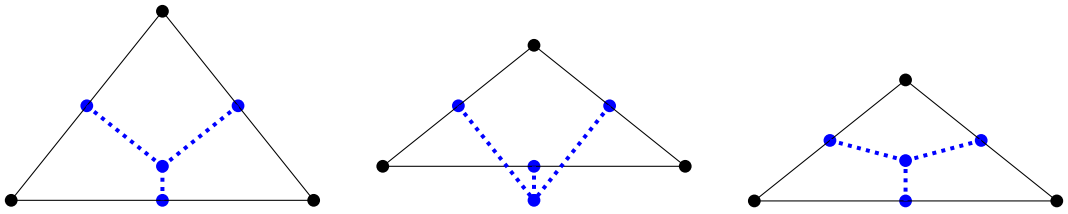
Chapter 2

State of the art

It is important to understand how a quadrilateral mesh can be generated. As mentioned in [9], there are 2 possible approaches.
Firstly, the direct approach, which consists of using techniques to immediately place quadrilaterals on the domain.
On the other hand, the indirect approach involves first a triangulation of the domain. Then, depending on the techniques used, the triangles are either combined or divided to form quadrilaterals. They can also just be a first step in the computations needed to form a quadrangulation. It is for this reason that our methodology will use a triangulation.

2.1 Basic indirect approach

The simplest way to form a quad-mesh is to cut the triangles in the initially generated mesh into quadrilaterals. Figure 2.1a shows a first approach using perpendicular bisectors of triangles.



(a) Perpendicular bisectors (works well)

(b) Perpendicular bisectors (doesn't work well)

(c) Medians

Figure 2.1: Representation of basic indirect approaches using different cuttings

However, it is important to note that in the case of our current study, we have to work on geometries with very acute or very obtuse angles. As a result, the triangles will also have the same type of constraints. Using perpendicular bisectors will not be robust in this case (see Figure 2.1b).
To solve this problem, we can use medians, for example (see Figure 2.1c). They can be adapted to any angle.

There are, of course, many other ways of splitting triangles.
It is true that this technique is simple, but it has one major disadvantage, a high number

of irregular nodes will be present in the final mesh. This can already be seen within the triangle itself, where the central node that is created has only 3 adjacent edges.

2.2 Automatic 2D quad-mesh generator using Bezier curves

In this section, we will describe a method for generating quadrilateral meshes using a direct approach, which requires a decomposition of the geometry. This method was proposed by *Talbert et al* (1990, [11]). Here are the different steps :

1. *Bezier* curves in the counterclockwise direction are used to define the boundary of the geometry. These curves are then linked to form a closed loop. If there are several boundaries, they are linked by cut lines added by the user. As the size of the continuous closed loop and the size of the desired elements are known, the boundary nodes are imposed.

2. We determine whether the loop is convex or concave. It is concave if at least one of the interior angles (the angle formed by a boundary node and its 2 neighbouring boundary nodes) is strictly greater than 180°.

3. If the loop is convex, then all the boundary nodes can be connected to each other without leaving the loop. In the concave case, we need to check for each pair of boundary nodes whether they are visible to each other (see Figure 2.2). To determine whether the boundary node is visible to the boundary node, proceed as follows:

   - Draw a straight line from the boundary node i to the boundary node $i-1$ (d_{i-1}), and a straight line from the boundary node i to the boundary node $i+1$ (d_{i+1}).
   - The line d_{i-1} is rotated clockwise until it reaches d_{i+1}. The nodes through which the line does not pass as it rotates are removed from the nodes visible to i.
     We proceed in the same way with a counterclockwise rotation for d_{i+1} up to d_{i-1}. After this, other nodes are removed from the list of those visible to i.

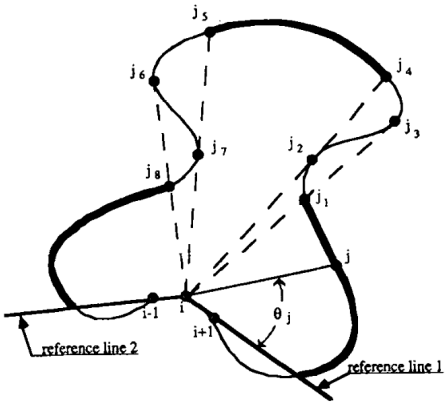


Figure 2.2: Representation of the visible nodes for node i [11]

4. The aim of this step is to determine the best splitting line (SL). To do this, a linear combination of 3 parameters is used:

$$SL = c_1\theta + c_2 L + c_3\epsilon \tag{2.1}$$

where :

- θ : The deviation of the split angles from the ideal (90°). We can mesure this by the following formula :

$$\theta = \frac{\sum_{i=1}^{4} \left|\theta_i - \frac{\pi}{2}\right|}{2\pi} \tag{2.2}$$

- L : The length of the splitting line
- ϵ : The error in element size matching (node placement)

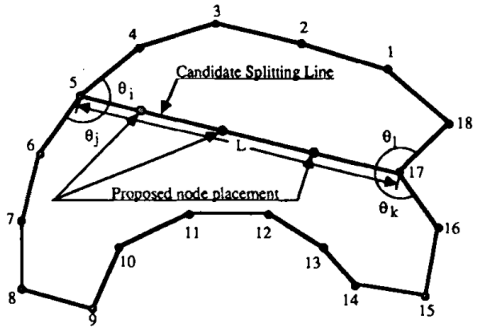


Figure 2.3: Representation of a candidate splitting line [11]

Talbert et al determined that the best weights are :

$$c_1 = 0.5 \qquad c_2 = 0.3 \qquad c_3 = 0.2$$

We need to select the candidate who minimises SL (see Figure 2.3).

5. When the closed loop has been split into 2 new loops, we check whether there are only 4 or 6 nodes for each of them. For closed loops where this is not the case, start again at step 2.
   If there are only 4 nodes, we have a quadrilateral, so that is it for that loop. In the case of 6 nodes remaining, a table is given where the mesh is imposed according to the configuration of those nodes (see Figure 2.4).

6. Smoothing

For this method, 2 steps are problematic.
First of all, at step 4, we want to minimise SL that involves the deviation of the split angles. We want θ_i as close to 90° as possible. As we have very acutes and obtuses angles in our geometry, we do not always want perpendicular angles to the boundary.
Secondly, we have an issue at step 5. When there are only 6 nodes left, most of the propositions in the table have irregular nodes.

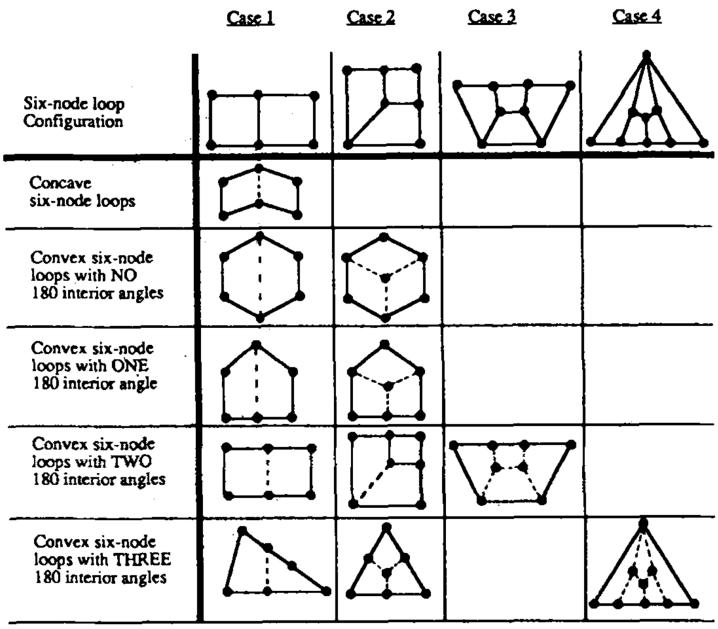


Figure 2.4: Preconfiguration of 6 nodes geometries [11]

2.3 Paving

Paving by *Blacker et al* (1991, [3]) is a method that uses a direct approach, progressing iteratively by forming the quadrilateral elements starting from the boundaries. Here are the different steps :

1. *Row choice* : Each time a row is filled with quadrilateral elements, a new row must be redefined. We build a row by selecting nodes from the paving boundary. The paving boundary is the boundary with all edges adjacent to only one quadrilateral element or to the boundary of the geometry. To create a row, we need to classify nodes by their angle inside the domain formed by the paving boundary (α).

   - Row end ($\alpha \leq \frac{\pi}{2} + \sigma_1$) : Node at which the row ends. It can only have one element adjacent to it.
   - Ambiguous row end/side ($\frac{\pi}{2} + \sigma_1 < \alpha \leq \pi - \sigma_2$)
   - Row side ($\pi - \sigma_2 < \alpha \leq \pi + \sigma_3$) : Node at which we have almost a continuous line. There are 2 elements adjacent to it.
   - Ambiguous row side/corner ($\pi + \sigma_3 < \alpha \leq \frac{3\pi}{2} - \sigma_4$)
   - Row corner ($\frac{3\pi}{2} - \sigma_4 < \alpha \leq \frac{3\pi}{2} + \sigma_5$) : Node corresponding to a classic corner, i.e. its outside angle is close to a right angle. There are 3 elements adjacent to it.
   - Ambiguous row corner /reversal ($\frac{3\pi}{2} + \sigma_5 < \alpha \leq 2\pi - \sigma_6$)
   - Row reversal ($2\pi - \sigma_6 < \alpha$) : Node that almost corresponds to a half-turn. There are 4 elements adjacent to it.

   Ambiguous categories contain nodes which could be put in 2 different categories. Parameters $\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6$ are tolerances on angles.

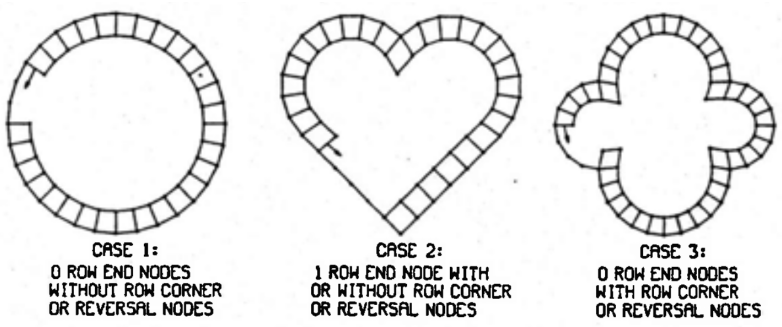


Figure 2.5: Geometries without 2 row end nodes for which methodologies are imposed to start the algorithm [3]

Some geometries have not at least 2 row end nodes (see Figure 2.5). Therefore, an initial row is defined to start the algorithm.

This technique only takes into account the current row, and does not attach any importance to the shape, position and orientation of the various boundaries of the geometry. We will look at how to rectify this later.
There is also an order to respect when creating rows. The rows just next to the last one created will be given priority. This is done in order to complete the quadrilateral mesh progressively from outside to inside the domain and to place the irregular nodes more at the centre of the geometry.

2. *Closure check* : In this step, we check if the *paving* algorithm is finished or not.

   - 0 remaining node : We stop the iterations.
   - 4 remaining nodes : We create a quad with those 4 nodes and then we stop the iterations.
   - 6 remaining nodes : As in *Automatic 2D quad-mesh generator using Bezier curves*, there are different configurations for 6 nodes (see Figure 2.6) :

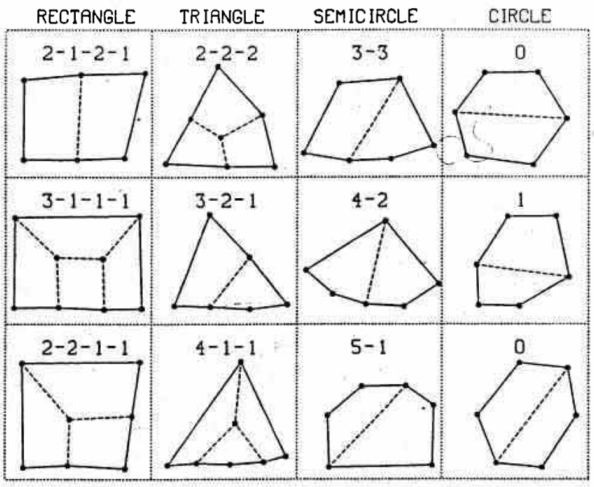


Figure 2.6: Preconfiguration of 6 nodes geometries [3]

After applying one of those configurations, we stop the iterations.

- More than 6 remaining nodes : We continue the iterations

3. *Row generation* : During this step, all the nodes are projected into the domain, creating a new row. Each node can create 1, 2, 3 or 4 new nodes, depending on the category to which it belongs. The projection is based on the angle of the current node, and creates 1, 2, 3 or 4 vectors from it, which are used to find the position and distance of the new nodes.

4. *Smoothing*

5. *"Repairs"* : This step is used to adjust any small problems caused by the previous steps and by the geometry itself.
   Seams are used to link elements that form a very small angle depending on their valence.
   Wedges can also be inserted in the case of a concave paving boundary, resulting in increasingly large elements. On the other hand, in the case of a convex paving boundary, tucks are formed to reduce the number of elements.
   There is also management of overlapping elements depending on whether they come from the same boundary or different boundaries.

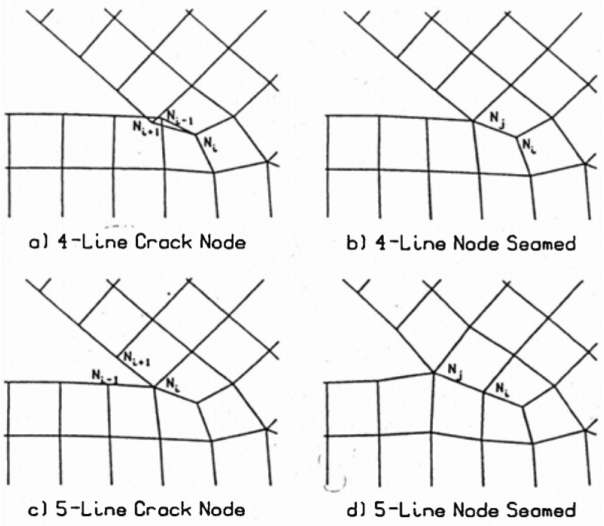


Figure 2.7: Example of an interior node seam [3]

6. Cleanup of completed mesh : The aim of this step is to improve the quality of the mesh. To do this, elements can be added or removed. This ensures that neighbouring elements are roughly the same size. This reduces the number of irregular nodes and eliminates elements with bad angles (i.e. very acute or very obtuse).

First of all, in step 2, we can again see that for all the possibilities of a quadrilateral mesh for 6 nodes, there are a lot of irregular nodes.
Overall, the *paving* method is quite difficult because there are many specific cases to take into account, as seen in steps 5 and 6. There are also many parameters and thresholds involved throughout the method. Slight changes in the value of these could create a completely different mesh. The problem with this is that it would have to be adapted to each of the geometries we want to mesh. This is not consistent with creating an algorithm where all the values are fixed for all the geometries.

If we now focus more on our own study, we can already see from Figure 2.7 in step 5, that very acute corners will produce irregular nodes. This is precisely what we would like to avoid.
In fact, whether it is seams, wedges, tucks or correcting overlaps, these modifications often produce irregular nodes. The problem with a method like *paving* is that it is a local rather than a global analysis of the geometry.
It is true that step 6 is there to correct all this, but it is not specified how to do it. We can assume that this is done manually when we see the final result.

2.4 Q-morph

Q-morph is an indirect approach proposed by *Owen et al* (1999, [9]). Steps are summarised below.

1. Selection of edges with only one adjacent triangle. Adjacent quadrilaterals are not taken into account in this count.

2. For each pair of neighbouring edges from the selection made in step 1 (the front edges), we calculate the bisector of the angle inside the domain between the 2 edges (V_k). We select edges where one of the nodes is the same as the starting point (node of origin (N_k) of this bisector, and which are not in the front edges).
   From these, we select the one which forms the smallest angle with the bisector. If this angle is less than ϵ, the edge is kept for the quadrilateral mesh.
   If this angle is greater than ϵ, select the edge opposite the origin node (E_0), and select the node opposite this edge (N_m). Plot the vector between the latter and the original node, and calculate the angle between this vector and the bisector :

   - if it is lower than ϵ, the edge E_0 is swap.
   - if it is greater than ϵ, the edge E_0 is split at the level of the intersection with V_k. This is illustrated in the Figure 2.8.

3. Now that the edges of the sides of the quadrilaterals are fixed, we need to recover the top edges to close these quadrilaterals using recovering techniques.

4. Smoothing the quadrilaterals.

5. If there are still triangles in the geometry, we return to step 1. Otherwise, the mesh is complete.

The final steps have been briefly explained because the problem with this method lies in step 2. To explain it properly, Figure 2.9 shows of a piece of boundary geometry with a very acute angle.

First of all, it is important to point out that in the method described, the angle at step 2 is usually fixed at $\epsilon = \frac{\pi}{6}$. In Figure 2.9, this angle has been taken. *Q-morph* favours orthogonality, and this is not what is expected with very acute or obtuse angles as seen on blue quadrilateral of the scheme. Orthogonality in relation to the boundary will, in this case, cause a significant number of irregular nodes in the final mesh.

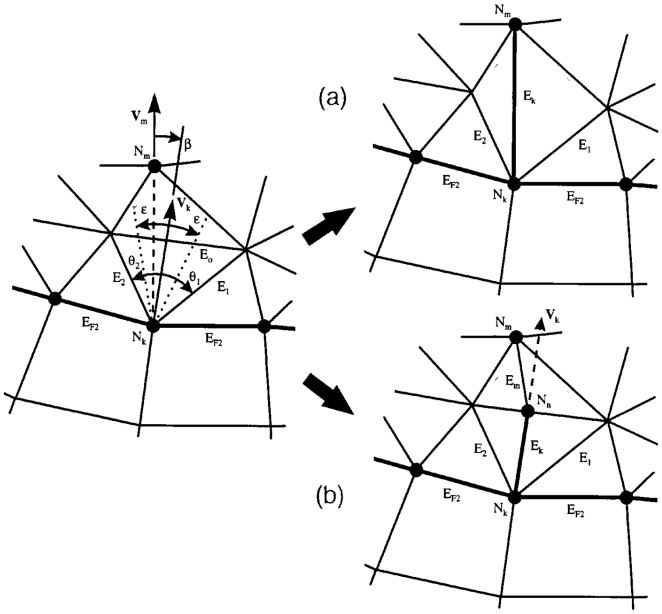


Figure 2.8: 2 types of perimeter intersection [9]

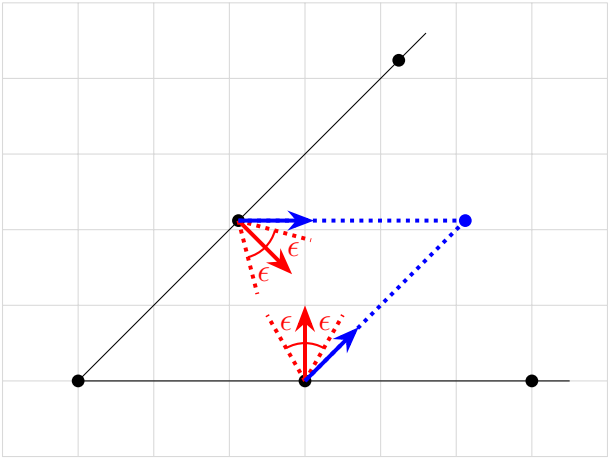


Figure 2.9: Iteration 1 of the *Q-morph* algorithm for a 45° corner

2.5 Ginzburg-Landau model

Our methodology that we will present later uses cross fields. It is therefore interesting to present a model which also uses cross fields. This is the *Ginzburg-Landau* model ([1] and [12]).
Consider a surface on which a triangular mesh has already been produced.
This method only uses orthogonal crosses. As a result, we only need one angle to describe a cross. Only fourth-order terms are needed to represent an orthogonal cross. This will be seen in the subsection *Representation of a cross*. The coefficients of the cross are therefore defined as follows:

$$\mathbf{f} = (\cos(4\theta_f), \sin(4\theta_f)) \equiv (f_1, f_2) \tag{2.3}$$

The angle θ_f can be calculated from any branch. The crosses on the boundary nodes are imposed with a branch parallel to the external normal, i.e. to the vector $\mathbf{n} =$

$(\cos(\theta_n), \sin(\theta_n))$. The coefficients of the boundary crosses are therefore :

$$\mathbf{f} = (\cos(4\theta_n), \sin(4\theta_n)) \quad \text{on } \Gamma \tag{2.4}$$

Once this initial condition has been imposed, we need to propagate these crosses within the domain. To do this, *Ginzburg* and *Landau* suggest the following equation:

$$E(f_1, f_2) = \underbrace{\frac{1}{2}\int_{\mathcal{S}} \left(|\nabla f_1|^2 + |\nabla f_2|^2\right) d\mathcal{S}}_{\text{smoothing}} + \underbrace{\frac{1}{4\epsilon^2}\int_{\mathcal{S}} \left(f_1^2 + f_2^2 - 1\right)^2 d\mathcal{S}}_{\text{penality}} \tag{2.5}$$

The aim is to minimise this expression to obtain the best cross field.
Penality prevents $\mathbf{f}$ from having a norm which is no longer equal to 1.
The smoothing term minimises the gradient. This ensures that 2 crosses of the cross field located next to each other are roughly similar, which avoids singularities.
The *Ginzburg-Landau* energy can be rewritten as follows:

$$E = \pi \left(\sum_{i=1}^{N} \text{index}(\mathbf{x}_i)^2\right) \log\left(\frac{1}{\epsilon}\right) + W + \mathcal{O}\left(\frac{1}{|\log(\epsilon)|}\right) \tag{2.6}$$

where

$$W = -\pi \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} \text{index}(\mathbf{x}_i)\text{index}(\mathbf{x}_j) \log|\mathbf{x}_i - \mathbf{x}_j| + R \tag{2.7}$$

with $\epsilon \to 0$. There is also a constraint to respect.

$$\sum_{i=1}^{N} \text{index}(\mathbf{x}_i) = 4\chi \tag{2.8}$$

The function index($\mathbf{x}$) returns an integer corresponding to the difference between the expected valence and the valence obtained. As a reminder, this is the value of k in

$$v = 4 - k \quad \text{on } \mathcal{S} \qquad \text{and} \qquad v = 2 - k \quad \text{on } \partial\mathcal{S} \tag{2.9}$$

where v is the valence.

The first term of E dominates when ϵ is small, which is indeed the case here. During minimisation, the function index($\mathbf{x}$), which is squared, will ensure that the singularities obtained have small indices.
Furthermore, the term W involves the distance between 2 points, $\mathbf{x}_i$ and $\mathbf{x}_j$. Minimisation will therefore seek to space out the singularities.

When coefficients have been obtained, the angle of each cross can be retrieved using

$$\theta_f = \frac{\text{atan2}(f_2, f_1)}{4} \tag{2.10}$$

This method gives relatively good results but favours orthogonality because of the crosses used.
We will see later that elements similar to the *Ginzburg-Landau* model will be used.

2.6 Surface deformation

Panozzo et al (2014, [10]) suggested a method that uses frame fields. A frame has the form $\langle \mathbf{v}, \mathbf{w}, \text{-}\mathbf{v}, \text{-}\mathbf{w} \rangle$. Each frame uses its own base $\mathbf{B}$. As each direction is linearly dependent on its opposite, we can rewrite each f_p by a matrix 2×2

$$\mathbf{V} = [\mathbf{v}, \mathbf{w}] \tag{2.11}$$

where $\mathbf{v}, \mathbf{w}$ correspond to the 2 branches whose angle is between 0 and π. Each column of $\mathbf{V}$ corresponds to the direction of a branch with respect to the basis $\mathbf{B}$.
It is also possible to express a frame as a combination of a cross :

$$f_p = \mathbf{W}\mathbf{x} = \langle \mathbf{W}\mathbf{u}, \mathbf{W}\mathbf{u}^{\perp}, -\mathbf{W}\mathbf{u}, -\mathbf{W}\mathbf{u}^{\perp} \rangle \tag{2.12}$$

where $\mathbf{W}$ is a SPD matrix (see Figure 2.10).

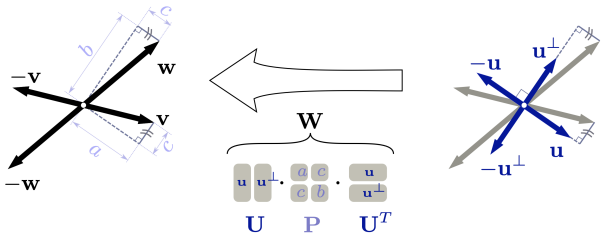


Figure 2.10: Canonical decomposition of a frame into a tensor $\mathbf{W}$ and a cross [10]

The method of *Panozzo et al* uses an indirect approach, so a triangular mesh is first generated. A frame is assigned to each triangle in the mesh.
Then, the aim is to generate a smooth frame field, i.e. each pair of neighbouring frames should be as similar as possible. If the frame field is smooth then $\mathcal{X}$ and $\mathcal{W}$, that are respectively a cross field and a SPD tensor field, are smooth. This smoothing must satisfy the constraints $\hat{f}_1$, ..., $\hat{f}_k$ which are imposed respectively on the triangles t_1, ..., t_k. These constraints can therefore be rewritten with $\hat{\mathbf{x}}_1$, ..., $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{W}}_1$, ..., $\hat{\mathbf{W}}_k$.
The goal is to interpolate the coefficients of $\mathcal{W}$ independently as harmonic scalar fields that satisfy the constraints. We use a face-based discrete Laplacian operator $\mathbf{L}^{\mathbf{B}}$ because we cannot interpolate directly. In fact, each constraint is expressed in a different basis.

$$\begin{aligned} &\mathbf{L}^{\mathbf{B}}(\mathbf{w}_1^T, \mathbf{w}_2^T, ..., \mathbf{w}_n^T)^T = \mathbf{0} \\ &\text{subject to } \mathbf{w}_j = \hat{\mathbf{w}}_j, \quad j \in \mathcal{C} \end{aligned} \tag{2.13}$$

where each $\mathbf{w}_i \in \mathbb{R}^3$ is a vector that contains the three entries of $\mathbf{W}_i$. $\mathcal{C}$ is the set of the constrained triangles and $\hat{\mathbf{w}}_j$ are the constraints. $\mathbf{L}^{\mathbf{B}}$ is a $3n \times 3n$ matrix, where n is the number of triangles in the mesh.
Following interpolation (2.13), we obtain a frame field which will enable us to create a quadrilateral mesh.

The method proposed by *Panozzo et al* gives conclusive results for obtaining frame fields.
Our method, which will be described in the next chapter, will use a process similar to that described above, but with non-orthogonal crosses instead of frames. We will also

impose constraints on our field, and we will try to smooth the cross field. However, we won't be mapping to orthogonal crosses.
One of the disadvantages of the *Panozzo et al* method is in obtaining the final mesh. A large difference in size between the triangles obtained following the deformation of the frames to find crosses can give numerical problems when using the *MIQ* method (2009, [4]).
Our method, which will be detailed in the next chapter, uses frames of unit length, which avoids this problem.

Chapter 3

Methodology

The aim of this section is to explain our methodology, which is strongly inspired by *Desobry et al* (2021, [6]). We will also detail the limits of this method and the solutions we propose.
For the whole methodology, we consider non-orthogonal crosses.

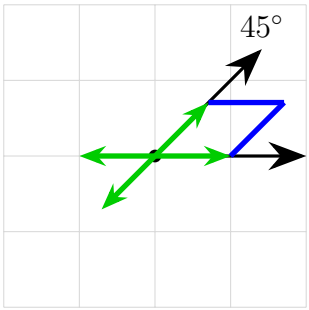
3.1 Geometry and mesh structure

3.1.1 Nodes

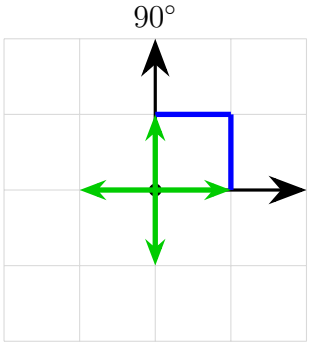
In order to solve the rest of the problem, we need to be able to distinguish between the different nodes. They have been separated into 3 categories:

- Category 0 : Nodes inside the geometry
- Category 1 : Boundary nodes of the geometry whose 2 adjacent edges form an angle between $(180 - \gamma)°$ and $(180 + \gamma)°$, or between $(360 - \gamma)°$ and $360°$. These nodes have respectively a valence of 2 and a valence of 4.
- Category 2 : Boundary nodes of the geometry whose 2 adjacent edges form an angle between $0°$ and $(180 - \gamma)°$, or between $(180 + \gamma)°$ and $(360 - \gamma)°$. These nodes have respectively a valence of 1 and a valence of 3.

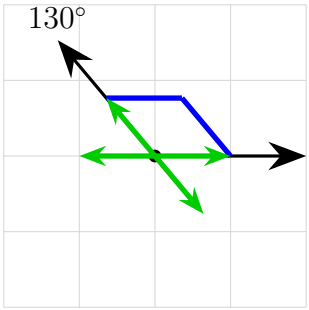
We can see that the angle γ should theoretically be between $0°$ and $90°$. In order to best decide on the value of this angle, it is important to know the results we want.
The arbitrary choice made is to take $\gamma = 45°$. This corresponds to the average of the 2 limits of the value of γ. To judge this choice, let's analyse what it will give us in terms of the crosses of certain critical angles.
These categories are similar to those of the *paving* method. Ambiguous categories have been removed, and there is now only one parameter instead of 6 for the *paving* method.
Meshes for different angles are shown on Figure 3.1. Directions in green are those that are fixed throughout the algorithm. For those in red, they may or may not be fixed during the algorithm. This will be explained in detail in the subsection *Penality*.
We impose certain crosses on the boundary because we want to avoid the appearance of singularities at the corners.



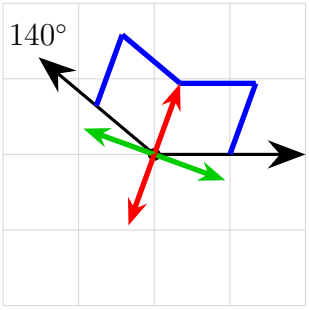
(a) Angle of 45° (valence 1)



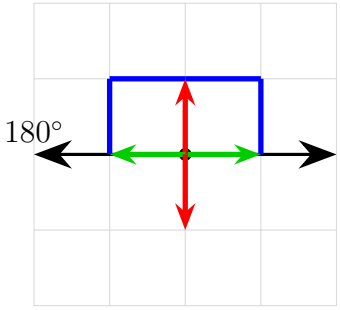
(b) Angle of 90° (valence 1)



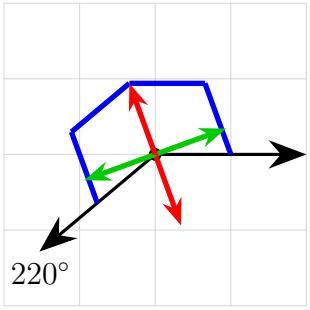
(c) Angle of 135° (valence 1)



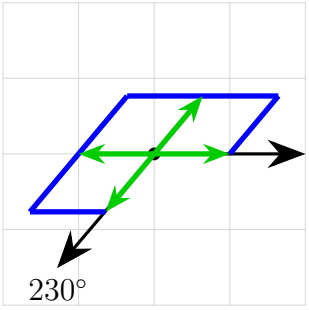
(d) Angle of 140° (valence 2)



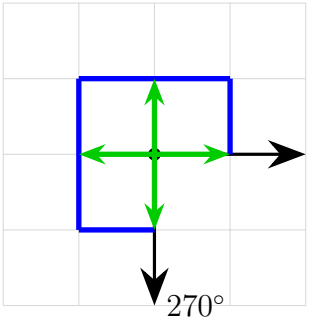
(e) Angle of 180° (valence 2)



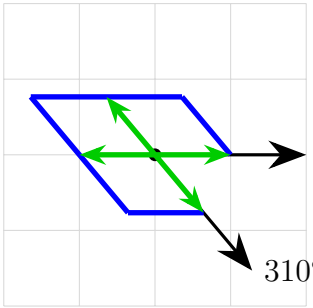
(f) Angle of 220° (valence 2)



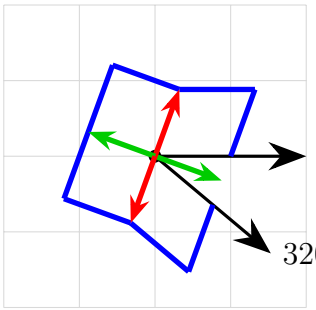
(g) Angle of 230° (valence 3)



(h) Angle of 270° (valence 3)



(i) Angle of 310° (valence 3)



(j) Angle of 320° (valence 4)

Figure 3.1: Illustration of the ideal mesh for different angles ($\gamma = 45°$)

Detection of angle values

In order to impose crosses on the corners, we first need to know their angle. The difficulty lies in finding out on which side of the edges adjacent to the corner the interior of the domain lies. To do this, we sort the boundary edges into a list. Then select one of these edges and find the triangle to which it belongs. We check whether the node not belonging to the boundary is located to the right of the selected edge. If it is, then we reverse the order of the list.
If there are several boundaries to the geometry, repeat these steps for each of them.
From now we know that the interior of the geometry is always to the left of any boundary edge.
Knowing this, it is easy to calculate the interior angle of each corner. We can then impose the desired crosses.

3.1.2 Half-Edges

In the context of meshes, we can quickly have a large number of triangles within the geometry being studied.
In order to keep a good complexity in the algorithm, we need to be able to go through these triangles efficiently. We will work with a structure that splits the edges into 2 half-edges. This allows us to move from one triangle to its neighbour by taking the twin half-edge. This is represented on Figure 3.2.

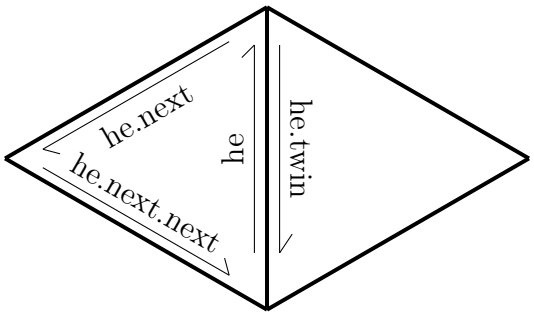


Figure 3.2: Representation of half-edges

More formally, the structure of a half-edge is represented in the code as follows:

```
class HalfEdge :
    def __init__(self, o, d, iT = None, n = None, t = None) :
        self.orig = o                # object Node
        self.dest = d                # object Node
        self.edgeId = -1             # tag only if it is a boundary edge
        self.incidentTriangle = iT   # object Triangle
        self.normal = None
        self.next = n                # object HalfEdge
        self.twin = t                # object HalfEdge
```

There is an attribute called `self.incidentTriangle` which plays an important role. This makes it easy to switch from half-edges to triangles, and vice versa.
Here is the structure of the triangles in the code :

```
class Triangle :
    def __init__(self, iE, id) :
        self.incidentEdge = iE       # object HalfEdge
        self.triangleId = id
```

3.1.3 Representation of a cross

Before looking for a mathematical representation of a cross, it's important to remember that the same cross can be numbered 24 different ways as shown on Figure 3.3.

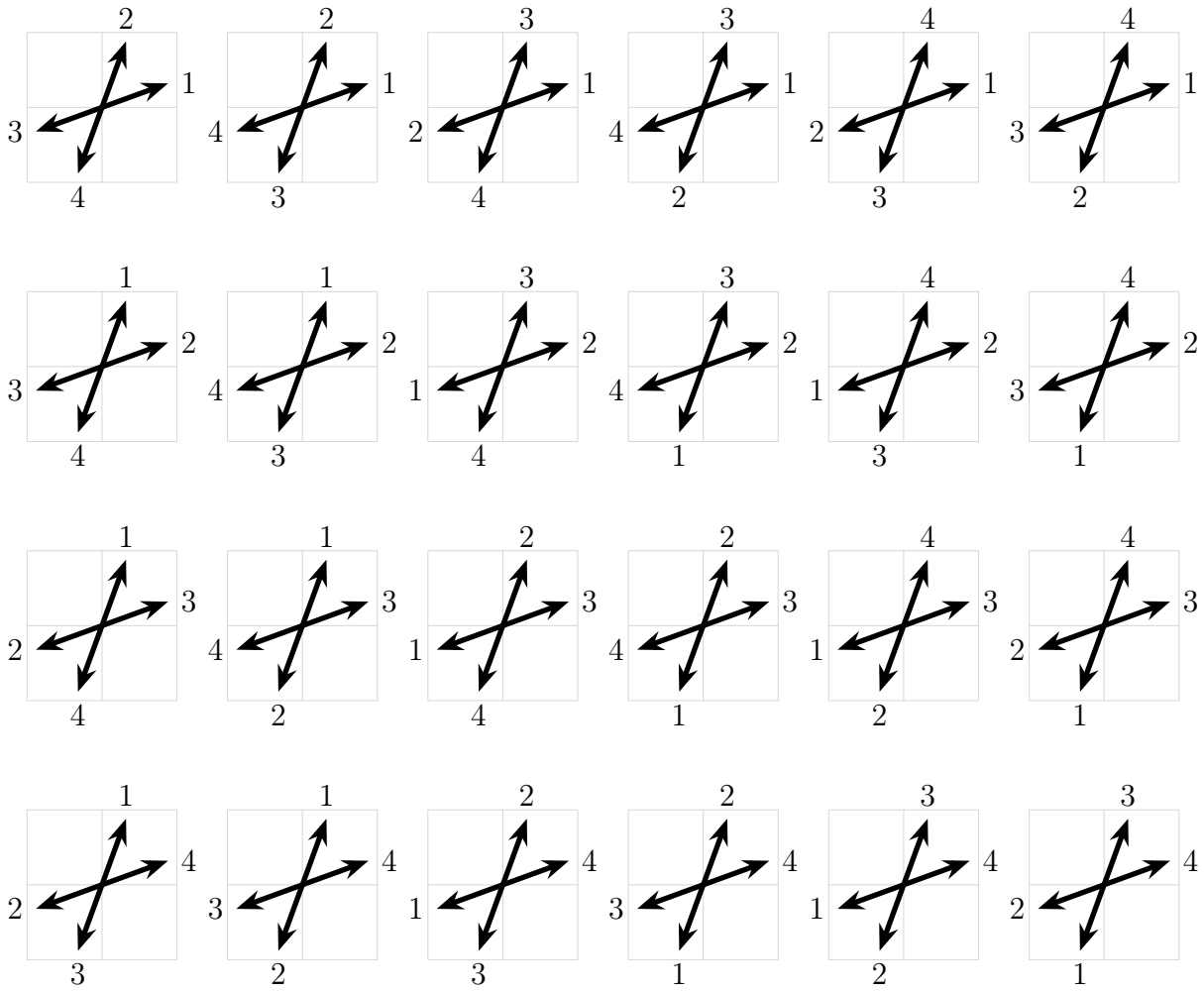


Figure 3.3: Different representations of a same cross

A non orthogonal cross is defined by 2 angles, α and β which are the angles of the first and second directions.
From [6], we have for a direction, the representation of a cross by the spherical polynomial $p(s) = \langle u, s\rangle^{2k}$. The even exponent means that there is no difference between 2 opposite vectors in the polynomial representation. As we have 2 directions for each cross, the polynomial becomes $p(s) = \langle u, s\rangle^{2k} + \langle v, s\rangle^{2k}$.

This polynomial belongs to the space

$$\mathbb{P}_{2k,N} := \left\{ \sum_{i=1}^{N} \langle u_i, s \rangle^{2k} \quad | \quad \forall u_i \in \mathbb{S}^n \right\} \tag{3.1}$$

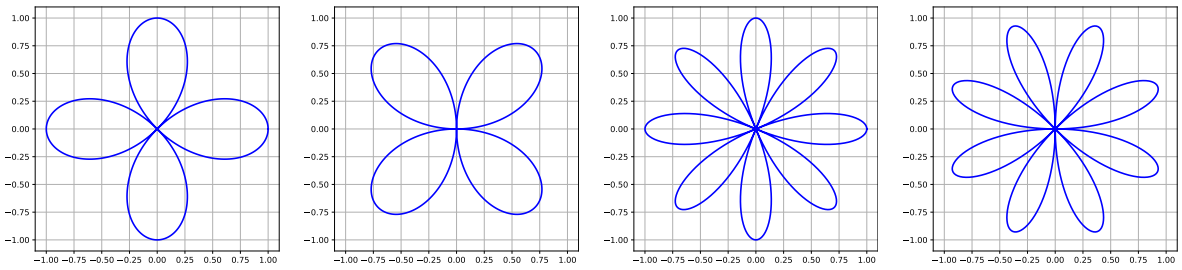
where $N = 2$.
This space is of finite dimension and can therefore be spanned by a finite number of orthogonal functions Y_ℓ, which will be related to the polynomial coefficients.
A space of orthogonal functions is a space with any 2 functions f and g respecting

$$\langle f, g \rangle = \frac{1}{L} \int_{-L}^{L} f(x) g(x) dx = 0 \tag{3.2}$$

We are in 2D so we will use the Fourier basis, which corresponds to

$$Y_{\ell,m}(s) = \begin{cases} \cos(\ell t) & \text{for } m = 0, \\ \sin(\ell t) & \text{for } m = 1, \end{cases} \tag{3.3}$$

where ℓ is the frequency index and t is an angle obtained by expressed s in polar coordinates (see Figure 3.4).



(a) $\cos(2t)$ (b) $\sin(2t)$ (c) $\cos(4t)$ (d) $\sin(4t)$

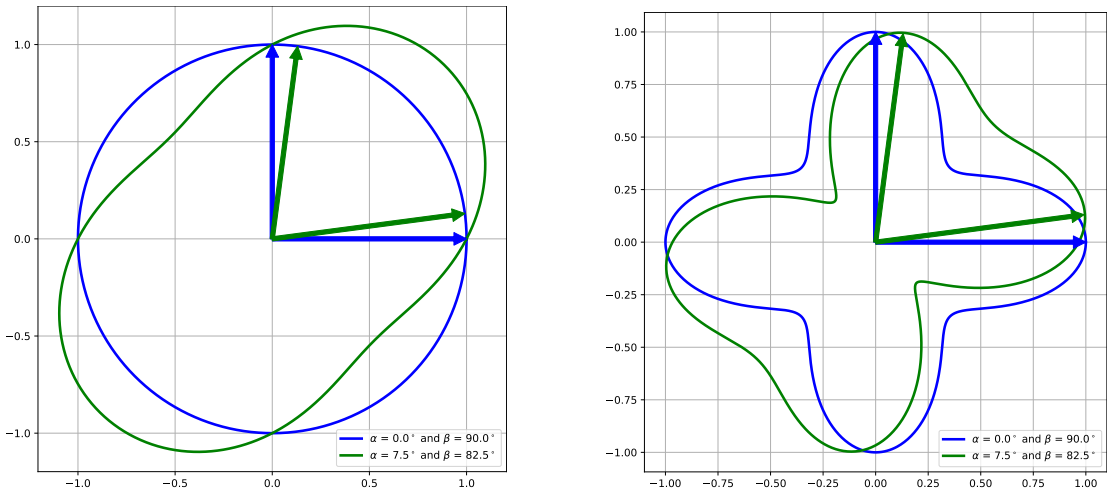
Figure 3.4: Representation of orthogonal functions of the Fourier basis ($\ell = 2$ and $\ell = 4$)

As a result, the polynomial representing a cross can be rewritten as follows :

$$p(s) = \sum_{\ell=1}^{L} c_{\ell,m}(\alpha, \beta) Y_{\ell,m}(s) \tag{3.4}$$

We need to determine the value of k, knowing that we want to minimise the number of coefficients in the polar polynomial that represents a cross. Assume that $k = 1$. In the case of a non-orthogonal cross, its representation is indeed unique. However, if our cross is orthogonal, its polar polynomial is a circle (see Figure 3.5a). This can easily proved by calculation.

$$\begin{aligned} \langle u, s \rangle^2 + \langle v, s \rangle^2 &= ||u||^2 \, ||s||^2 \cos^2(\alpha - t) + ||v||^2 \, ||s||^2 \cos^2\left(\left(\alpha + \frac{\pi}{2}\right) - t\right) \\ &= \cos^2(\alpha - t) + \cos^2\left((\alpha - t) + \frac{\pi}{2}\right) \\ &= \cos^2(\alpha - t) + (-\sin(\alpha - t))^2 \\ &= 1 \end{aligned} \tag{3.5}$$



(a) $k = 1$ (Degree 2) (b) $k = 2$ (Degree 4)

Figure 3.5: Representation of 2D crosses by their polar polynomial of different degrees

The polynomial therefore does not depend on the angle in the case of an orthogonal cross.

We need to take $k = 2$ to have a unique polynomial for each cross (see Figure 3.5b).

$$\begin{aligned}
\langle u, s\rangle^4 &= ||u||^4 \, ||s||^4 \cos^4(\alpha - t) \\
&= \cos^4(\alpha - t) \\
&= \left(\frac{1 + \cos(2(\alpha - t))}{2}\right)^2 \\
&= \frac{1}{4} + \frac{1}{2}\cos(2(\alpha - t)) + \frac{1}{4}\cos^2(2(\alpha - t)) \\
&= \frac{1}{4} + \frac{1}{2}\cos(2\alpha)\cos(2t) + \frac{1}{2}\sin(2\alpha)\sin(2t) + \frac{1}{4}\left(\frac{1 + \cos(4(\alpha - t))}{2}\right) \\
&= \frac{3}{8} + \frac{1}{2}\cos(2\alpha)\cos(2t) + \frac{1}{2}\sin(2\alpha)\sin(2t) + \frac{1}{8}\cos(4(\alpha - t)) \\
&= \frac{3}{8} + \frac{1}{2}\cos(2\alpha)\cos(2t) + \frac{1}{2}\sin(2\alpha)\sin(2t) \\
&\quad + \frac{1}{8}\cos(4\alpha)\cos(4t) + \frac{1}{8}\sin(4\alpha)\sin(4t)
\end{aligned} \tag{3.6}$$

From this equation (3.6), we deduce that the coefficients of the polynomial decomposition are :

$$\begin{cases}
c_{2,1} = 4\cos(2\alpha) + 4\cos(2\beta) \\
c_{2,2} = 4\sin(2\alpha) + 4\sin(2\beta) \\
c_{4,1} = \cos(4\alpha) + \cos(4\beta) \\
c_{4,2} = \sin(4\alpha) + \sin(4\beta)
\end{cases} \tag{3.7}$$

To understand the role of each of these coefficients, let's consider the case of an orthogonal cross. We have $\beta = \alpha + \frac{\pi}{2}$, which gives :

$$\begin{cases} c_{2,1} = 4\cos(2\alpha) + 4\cos(2\alpha + \pi) = 4\cos(2\alpha) - 4\cos(2\alpha) = 0 \\ c_{2,2} = 4\sin(2\alpha) + 4\sin(2\alpha + \pi) = 4\sin(2\alpha) - 4\sin(2\alpha) = 0 \\ c_{4,1} = \cos(4\alpha) + \cos(4\alpha + 2\pi) = \cos(4\alpha) + \cos(4\alpha) = 2\cos(4\alpha) \\ c_{4,2} = \sin(4\alpha) + \sin(4\alpha + 2\pi) = \sin(4\alpha) + \sin(4\alpha) = 2\sin(4\alpha) \end{cases} \tag{3.8}$$

We can therefore see that it is the coefficients of the second order terms that influence the orthogonality of a cross.

3.2 Creation of the objective function

3.2.1 Smoothness energy

Boundary nodes

In this step, we smooth the crosses along the entire boundary.
To do this, we need to select 2 consecutive category 2 nodes. Start and end nodes can be the same. Then, the methodology is as follows:

- Calculate the angle between the branch parallel to the current boundary and the next counterclockwise branch, for each of the 2 nodes selected (n_0 and n_p). These angles are respectively α_0 and α_p.
- Compute the distance between those 2 nodes using only boundary edges (d).
- Smooth the angles of the crosses on the current boundary. Let $p-1$ be the number of nodes between the 2 corners of category 2, and d_i the distance between n_0 and n_i, the current node.

$$\alpha_i = \frac{d - d_i}{d} * \alpha_0 + \frac{d_i}{d} * \alpha_p \qquad \text{with } i = 1, ..., p-1 \tag{3.9}$$

  where α_i corresponds to the angle between the nearest branch to the current boundary and the next counterclockwise branch for the node i of the current boundary.

If there are no category 2 nodes on the entire boundary, we let the crosses remain on the boundary as they were previously imposed.
We assume that the directions obtained are close to those we want to obtain for the final cross field.
The interest in replacing them, while they are still free, is important for the start of the optimisation, as explained in the subsection *Initial condition*.

Interior nodes

To propagate the boundary crosses within the domain in the best possible way, and to obtain the least possible irregular nodes, we need to ensure that 2 neighbouring crosses are roughly similar. To do this, we minimise the difference between the polynomials of 2 crosses.

Using results obtained previously, we have :

$$\begin{aligned} ||p_1(s) - p_2(s)||_2^2 &= \int_{\mathbb{S}^n} (p_1(s) - p_2(s))^2 ds \\ &= \sum_{\ell=2,4} \sum_{m=1,2} (c_{\ell,m}(\alpha_1, \beta_1) - c_{\ell,m}(\alpha_2, \beta_2))^2 \end{aligned} \tag{3.10}$$

Smoothness energy can be defined as the sum of the norms of the difference between the polynomials of 2 adjacent crosses.

$$E_s = \sum_{(ij) \in \mathcal{E}} ||p_1(s) - p_2(s)||_2^2 = \sum_{(ij) \in \mathcal{E}} \sum_{\ell=2,4} \sum_{m=1,2} (c_{\ell,m}(\alpha_i, \beta_i) - c_{\ell,m}(\alpha_j, \beta_j))^2 \tag{3.11}$$

A pair of nodes (i, j) belongs to the set ε if and only if these 2 nodes are adjacent.

To avoid problems with the numbering of crosses, *Desobry et al* suggest the change of variable $\bar{u} = (\cos(2\alpha), \sin(2\alpha))$ and $\bar{v} = (\cos(2\beta), \sin(2\beta))$. Vectors $\bar{u}$ and $\bar{v}$ have a constraint to respect which is that their norm must be unitary.
Multiplying the angle by 2 makes it possible to confuse the opposite branches, i.e. we can calculate α (or β) for any of the 2 branches of the pair formed by a direction.
More concretely, this means that for a cross $\{u, -u, v, -v\}$, this choice of change of variable will give the same value of sine and cosine whether we take the angle of the vector u or $-u$ (the same by v or $-v$) as they differ by a value of π.
This choice can be made because we see that angles in coefficients of (3.7) are only multiplied by even integers.
Therefore, polynomial coefficients become

$$\begin{cases} c_{2,1} = 4\bar{u}_1 + 4\bar{v}_1 \\ c_{2,2} = 4\bar{u}_2 + 4\bar{v}_2 \\ c_{4,1} = 2\bar{u}_1^2 + 2\bar{v}_1^2 - 2 \\ c_{4,2} = 2\bar{u}_1\bar{u}_2 + 2\bar{v}_1\bar{v}_2 \end{cases} \tag{3.12}$$

Knowing this, we can rewrite the equation (3.11).

$$E_s(\bar{u}, \bar{v}) = \sum_{(ij) \in \mathcal{E}} \sum_{\ell=2,4} \sum_{m=1,2} (c_{\ell,m}(\bar{u}_i, \bar{v}_i) - c_{\ell,m}(\bar{u}_j, \bar{v}_j))^2 \tag{3.13}$$

It is the second-order terms that affect orthogonality. In order to have control over these coefficients, we add a parameter to the smoothness energy.

$$\begin{aligned} E_s(\bar{u}, \bar{v}) = \lambda &\sum_{(ij) \in \mathcal{E}} \sum_{m=1,2} (c_{2,m}(\bar{u}_i, \bar{v}_i) - c_{2,m}(\bar{u}_j, \bar{v}_j))^2 \\ &+ \sum_{(ij) \in \mathcal{E}} \sum_{m=1,2} (c_{4,m}(\bar{u}_i, \bar{v}_i) - c_{4,m}(\bar{u}_j, \bar{v}_j))^2 \end{aligned} \tag{3.14}$$

3.2.2 Penality

Desobry et al suggests a boundary penality.

$$E_b(\bar{u}, \bar{v}) = \sum_{i \in \partial\mathcal{N}} (|\bar{u}_i|^2 - 1)^2 = \sum_{i \in \partial\mathcal{N}} (\cos^2(2\alpha_i) + \sin^2(2\alpha_i) - 1)^2 \tag{3.15}$$

$\mathcal{N}$ corresponds to the set of nodes in the geometry. As a result, $\partial\mathcal{N}$ is the set of the boundary nodes. In [6], they use $\mathcal{F}$. This corresponds to the set of faces in the geometry. For our model, we have chosen nodes for simplicity. This is essentially the same thing, because we also get a cross field at the end of the algorithm.

This energy only applies to the unfixed branches of the boundary nodes. The aim of this energy is to ensure that the free directions do not become degenerate, i.e. that their norm does not deviate too far from 1.
This allows the red branches (see Figure 3.1) to rotate slightly during minimisations. This makes the optimisation problem less constrained and gives better results.
However, this can cause problems. Looking at the lower boundary segment in Figure 3.6, we can see that some of the free directions of the boundary crosses have inverted during the algorithm, i.e. they have moved to the other side of the fixed direction.
This is why we propose a second strategy, which is to fix all the crosses located on boundary nodes. For this choice, the penality (3.15) is not used. The disadvantage of this strategy is that the optimisation problem is more constrained, and the objective function will give us a higher value.

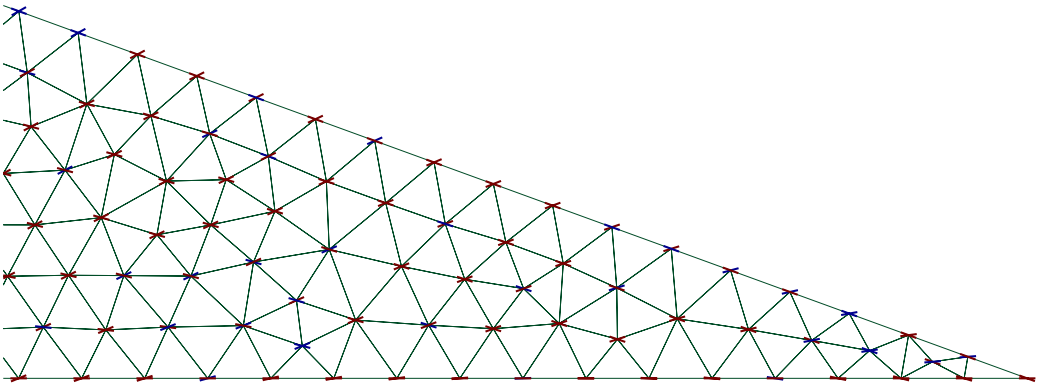


Figure 3.6: Comparison of singularities detection between cross field analysis (left) and smoothness energy visualisation on geometry (right).

Assume a non-orthogonal cross defined by 2 angles, α and β. We therefore have $\bar{u} = (\cos(2\alpha), \sin(2\alpha))$ and $\bar{v} = (\cos(2\beta), \sin(2\beta))$. To avoid errors such as those seen in Figure 3.6, the ideal would be to be able to bound the angle of the unfixed direction. This is not possible because we are not working directly with the angle, but with the variables $\bar{u}$ and $\bar{v}$.
The strategy we propose is to first run the algorithm leaving the red branches (see Figure 3.1) free, and to use the penality (3.15) by trying with increasingly larger values of the parameter b until we obtain a satisfactory result. b is a control parameter that will be introduced in *Objective function*.
If after trying several values of b, the result still produces an error as in Figure 3.6, then we suggest fixing the set of boundary crosses, and not using the penality in the objective function.

3.3 Optimization

3.3.1 Objective function

We have therefore just introduced the 2 terms needed for the optimisation. In order not to deviate too much from the given current result, *Desobry et al* suggest introducing an additional term to the objective function.
The objective function becomes

$$f(x,y) = E_s(x,y) + b * E_b(x,y) + \tau||(x,y) - (\bar{u},\bar{v})||_2^2 \tag{3.16}$$

where b is a parameter used to control the importance of the energy E_b. This parameter is zero when the crosses on the boundary nodes are fixed as explained in *Penality*.
The norm corresponds to the distance between the cross of the initial condition $((\bar{u},\bar{v}))$ and the new result of this same cross $((x,y))$. This avoids deviating too far from the initial condition. If the parameter τ is large, the solution obtained will be close to the initial condition, whereas if it is small, the solution will be further from it most of the time.

There are 2 steps on which we will iterate until we have a solution as precise as desired.

1. Obtain the solution to the minimisation problem
2. Projection of the results obtained so that the norms become unitary again.
   We know that the node i is described by $\{\bar{u}_i,\bar{v}_i\}$, so after the projection we want to have :
   $$||\bar{u}_i||_2 = 1 \text{ and } ||\bar{v}_i||_2 = 1 \qquad \forall i \in \mathcal{N} \tag{3.17}$$

3.3.2 Method

Now that the minimisation problem has been established, we need to find a method to solve it.
First of all, we see that our objective function is of class C^2 because all its partial derivatives of first and second order exist and are continuous. However, our problem is not linear.
Knowing this, we can use the *Newton-Raphson* method, which can be summarised by the following equation ([7]).

$$p_{s+1} = p_s - (K'(p_s))^{-1}K(p_s) \tag{3.18}$$

where $K'(p_s)$ is the Jacobian of K evaluated at p_s.
This equation allows us to find the roots of the function K iteratively.
We know that finding the minimum of a function corresponds to finding the roots of the derivative. For our objective function, this gives :

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1}\nabla f(x_k) \tag{3.19}$$

The problem lies with the Hessian. We know from inequality (3.20) that our objective function is not convex over the domain.

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \nleq tf(\mathbf{x}) + (1-t)f(\mathbf{y}) \qquad \text{with } t \in [0,1] \tag{3.20}$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors that represent random values of $(\bar{u},\bar{v})$.
In this case, the local extrema candidates (i.e. $\nabla f(x) = \mathbf{0}$) may be saddle points. To

counter this, we could use a *gradient descent* method, but using only the gradient slows down convergence. Instead, we will use a quasi-Newton method that approximates the Hessian. This is more robust for non-convex problems.

$$p_{s+1} = p_s - (\hat{K}'_s)^{-1} K(p_s) = p_s - \hat{M}'_s K(p_s) \tag{3.21}$$

where $\hat{M}'_{s\in\mathbb{N}}$ is a sequence of approximations to $(K'(p_s))^{-1}$.
In our case, we have :

$$x_{k+1} = x_k - B_k \nabla f(x_k) \qquad \text{where } B_k \in \mathbb{R}^{n\times n} \tag{3.22}$$

B_k is a sequence of approximations to $(\nabla^2 f(x_k))^{-1}$.
There are obviously several proposed approximations to B_k. Firstly, rank-one matrices which are expressed as follows :

$$B_{k+1} = B_k + uv^T \qquad \text{where } u, v \in \mathbb{R}^n \tag{3.23}$$

These types of matrices are quite simple but are less accurate if there are large changes in the Hessian over the iterations.
This is why we use a rank-two matrix. The aim is to maintain the symmetrical structure of the approximation by using 2 consecutive approximations.
We have chosen to use the *Broyden-Fletcher-Goldfarb-Shanno* (*BFGS*) method. More specifically, we used the *L-BFGS-B* method from `scipy.optimize.minimize`. This version does not need to store the entire inverse Hessian approximation.

3.3.3 Initial condition

For methods such as *L-BFGS-B*, an initial condition is required.
As a result of (3.20), the minimum found may simply be local and not global. However, it is possible to use different techniques to help obtain a global minimum.
As mentioned earlier, the red branches (see Figure 3.1) have been smoothed to give an initial direction. We assume that this is a good initial condition because these branches are close to what we want to achieve.
As far as the interior of the geometry is concerned, it is very difficult to determine a good initial condition. We would have to rely on the crosses fixed on the boundary of the geometry, which would mean minimising the smoothing energy, and therefore solving the optimisation problem. This is not possible.
Initially, our idea was to generate orthogonal crosses :

$$\mathbf{V} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.24}$$

This notation follows that introduced by *Panozzo et al* in the equation (2.11). The problem with this initial condition is that the interior nodes, which are present in a greater majority than the boundary nodes, will give a zero smoothing energy inside the domain. This risks leading us to a solution that is not a global minimum, as we want the boundary conditions to influence the first minimisation performed.
The final choice we made was to generate random values between 0 and $\frac{\pi}{2}$ for angles of each orthogonal cross. We use orthogonal crosses to avoid degeneration as much as possible. If we generate 2 random directions, it is possible that they will overlap, or that they will be very close to overlapping.
This choice obviously does not guarantee a global minimum, which is why we generate several initial conditions to increase our probability of obtaining the best result.

3.4 Streamlines

When the optimisation problem has converged, we obtain the final crosses needed to create the quadrilateral mesh. To get an initial idea of what this will look like, we use streamlines. The aim is to start from certain points on the surface and see how these points evolve by following the directions indicated by the crosses. This can be represented by a differential equation describing the evolution of a position.

$$\vec{v} = \frac{d\vec{x}}{dt} \tag{3.25}$$

By discretizing this, we can obtain the following coordinates of our point X_k.

$$\vec{V_k} = \frac{X_{k+1} - X_k}{\Delta t} \qquad \Rightarrow \qquad X_{k+1} = X_k + \vec{V_k}\Delta t \tag{3.26}$$

We therefore need X_0 and $\vec{V_0}$ in order to start the iterations. For the initial position condition we need, it was arbitrarily decided to take equidistant points on each boundary. The distance between the starting points of the streamlines is chosen by the user. We also need an initial direction. We therefore need to create a cross for our point X_0. To do this, we interpolate the crosses of the 2 nodes of the edge on which it is located. Next, we need to decide which branch to use. We take the one which is most perpendicular to the boundary edge and which goes towards the interior of the domain. This vector corresponds to $\vec{V_0}$.
When we have this, we obtain the point X_1 but we notice that we also need $\vec{V_1}$ for the rest. In fact, we need to establish a new cross at each iteration. To determine $\vec{V_k}$, we first create the cross located at node X_k by interpolating the 3 nodes of the triangle in which it is located. Let's take $\vec{W_i}$, the vector representing the branch i of this cross.

$$\vec{V_k} = \arg\max_{W_i} \langle \vec{V_{k-1}}, \vec{W_i} \rangle \tag{3.27}$$

To complete a streamline, there are 2 possibilities. Either the iterations stop when the maximum number of iterations has been reached, or the point X_{k+1} obtained is outside the surface.

It is important to specify that the streamlines only give an approximation of the final mesh that will be obtained. An example of streamline is shown on Figure 3.7.

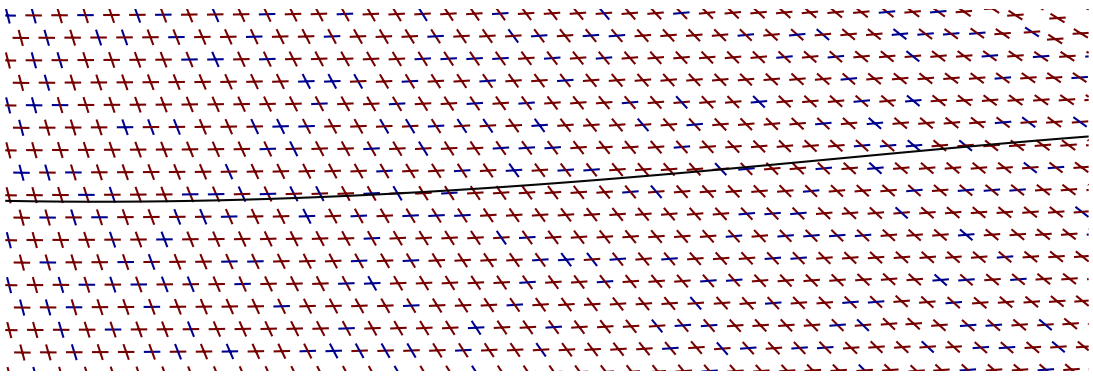


Figure 3.7: Streamline in a non-orthogonal cross field

3.5 Singularities detection

Figure 1.3 showed us how to determine the valence that a node will have from the placement of orthogonal crosses. It is therefore possible to determine the singularities of a cross field, to get an idea of the positioning of irregular nodes.
The detection method is applied to each triangle in the mesh.

- Selection of a starting node (n_0), which is one of the 3 vertices of the triangle.
- We randomly select one of the 4 branches of the starting node. A branch is given by $b_{i,j}$ where i and j respectively correspond to the index of the node and the index of the branch. Therefore, the starting node is $b_{0,j}$ (j has a value between 0 and 3).
- Then we iteratively perform the following equality 3 times :

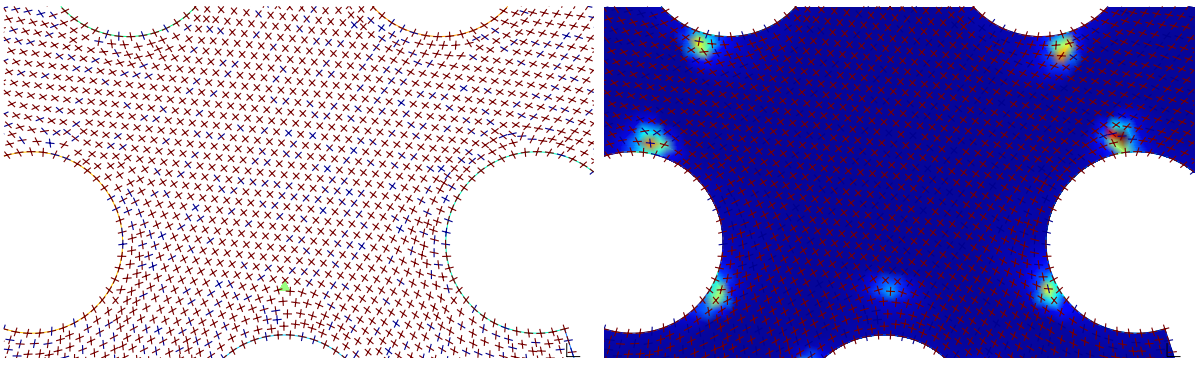
$$b_{(i+1)\%3,j} = \arg \max_{b_{(i+1)\%3,k}} \langle b_{i,j}, b_{(i+1)\%3,k} \rangle \tag{3.28}$$

  where % is the modulo symbol. Nodes of the triangle are selected counterclockwise.

- A new $b_{0,j}$ is found. We can determine the type of singularity by comparing it to the initial one.
  Let j_s be the starting branch and j_f the final branch, then we have

$$k = \begin{cases} 1 & \text{if } j_f - j_s = 1 \text{ or } j_s = 3, j_f = 0 \\ 0 & \text{if } j_s = j_f \\ -1 & \text{if } j_s - j_f = 1 \text{ or } j_s = 0, j_f = 3 \end{cases} \tag{3.29}$$

In our case, the crosses are non-orthogonal, so the results may differ depending on the branch chosen at the start. To improve our result, we repeat the algorithm 2 times starting from 2 adjacent branches. This gives us one result for each direction. If the 2 results are not similar, then the value of k is chosen arbitrarily in the order 0, 1 and -1. However, it is possible to be more critical by adding another analysis to this. By plotting the smoothness energy for each triangle, it is possible to see the positioning of singularities. A singularity appears when adjacent crosses are not smooth, and so their coefficients, as introduced in the equation (3.12), are significantly different. As a result, the smoothness energy has a high value for these triangles.
In Figure 3.8a, there is only one singularity, whereas in reality there are many more. This can be seen on Figure 3.8b where each zone with a high smoothness energy represents a singularity.
This geometry (shuriken) will be studied in more detail in chapter *Results*.



(a) Singularities detected (b) Smoothness energy on each triangle

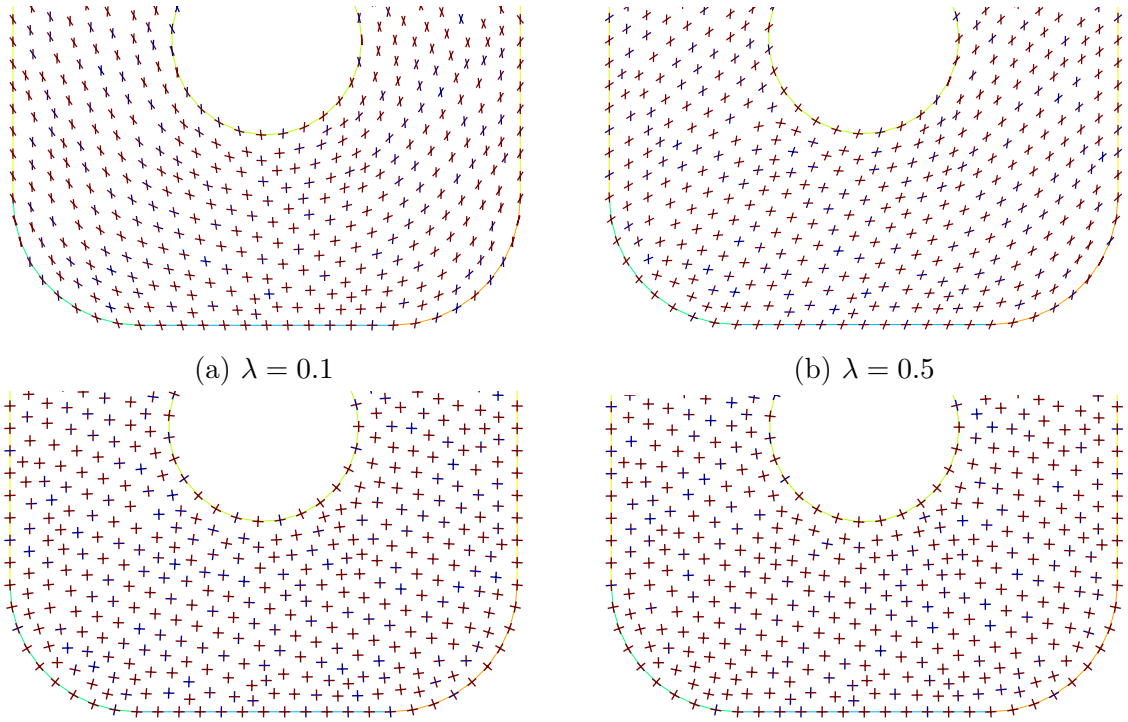
Figure 3.8: Comparison of singularities detection between cross field analysis (left) and smoothness energy visualisation on geometry (right).

Chapter 4

Results

For each result, we will specify the strategy used to obtain it, i.e. whether the crosses on the boundary have been completely fixed or whether certain directions have been left free. For all results, when it is mentioned that some boundary directions have been left free, these are the red directions of the category 1 nodes as illustrated in Figure 3.1.
It is important to specify that the results presented in this chapter are not final meshes. It is an idea of what the mesh could look like using streamlines.
For all results, the value of τ is 0.5.

Parameter λ



(a) $\lambda = 0.1$ (b) $\lambda = 0.5$

(c) $\lambda = 1.0$ (d) $\lambda = 1.5$

Figure 4.1: Cross fields obtained using our method for different values of λ

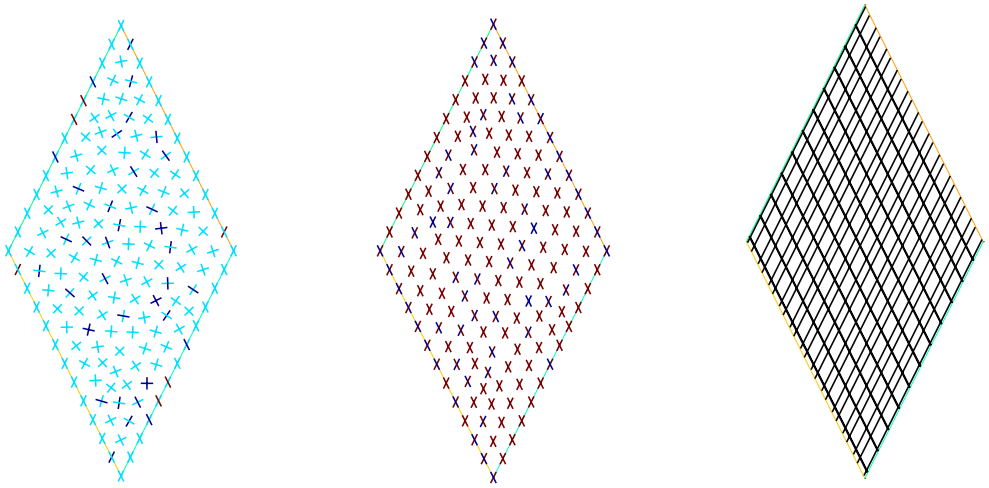
Figure 4.1 shows the results we obtained by varying the value of λ. These are close to those obtained by *Desobry et al* but are not exactly the same, although the parameters

used are the same. We also generated an orthogonal cross field as an initial condition as suggested.
In our case, for $\lambda = 0.1$ and $\lambda = 0.5$ we see inversions in the sense of what was explained in the subsection *Penality*. These results are therefore not correct.
When λ is higher, othogonality is increasingly favoured. We approach a result such as obtained by the *Ginzburg-Landau* model ([1] and [12]).
For all results, we take $\lambda = 1.5$ as suggested by *Desobry et al.*

4.1 Meshes with no interior irregular node

4.1.1 Diamond

We start with a very simple geometry that has 2 very acute angles and 2 very obtuse angles.



(a) Initial cross field (b) Final cross field (c) Streamlines

Figure 4.2: Result of our methodology on a geometry without interior irregular node (diamond)

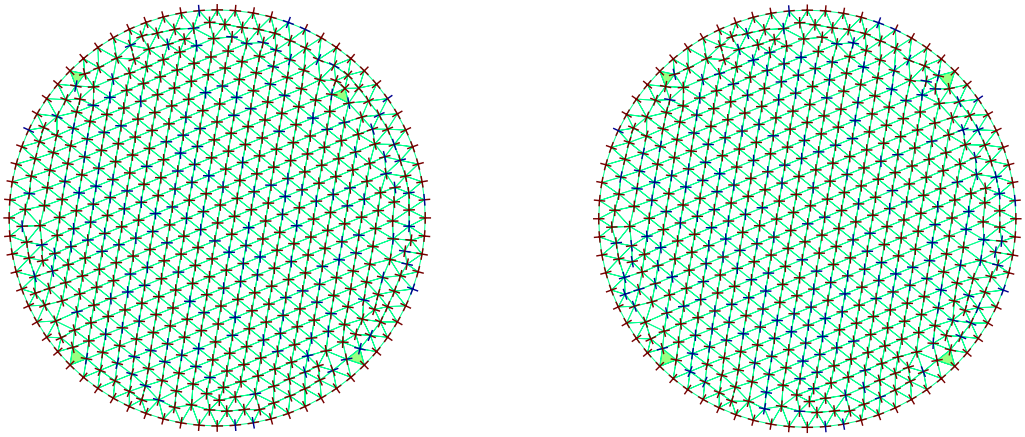
We can see that the boundary condition of the diamond is made up of the same cross for all the nodes on the boundary. Knowing this, we want all the crosses inside the geometry to become similar to those on the boundary.
In the result obtained, there are no singularities on the cross field, and all the crosses are similar (see Figure 4.2).
Whether with certain free boundary directions, or with all the fixed boundary crosses, the result obtained is always the same.

4.2 Meshes with interior irregular node

4.2.1 Disk

The disk is an important geometry because it does not have any category 2 nodes as presented in the subsection *Nodes*.

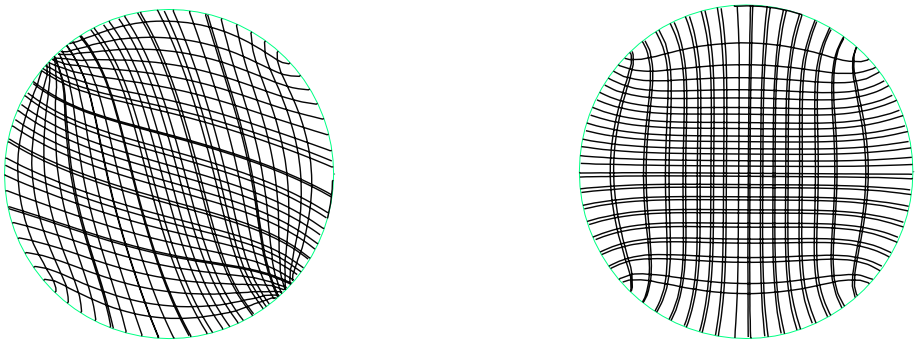
As seen in Figure 1.5, the best quad-mesh of a disk consists of 4 irregular nodes, which is indeed the case here (see Figure 4.3).



(a) Local minimum (b) Global minimum

Figure 4.3: Comparison between a local (left) and a global (right) minimum

Figure 4.3a shows an example of a local minimum of the objective function. This solution was obtained using a single initial condition for the resolution. This shows the advantage of using several initial conditions and keeping the best one.



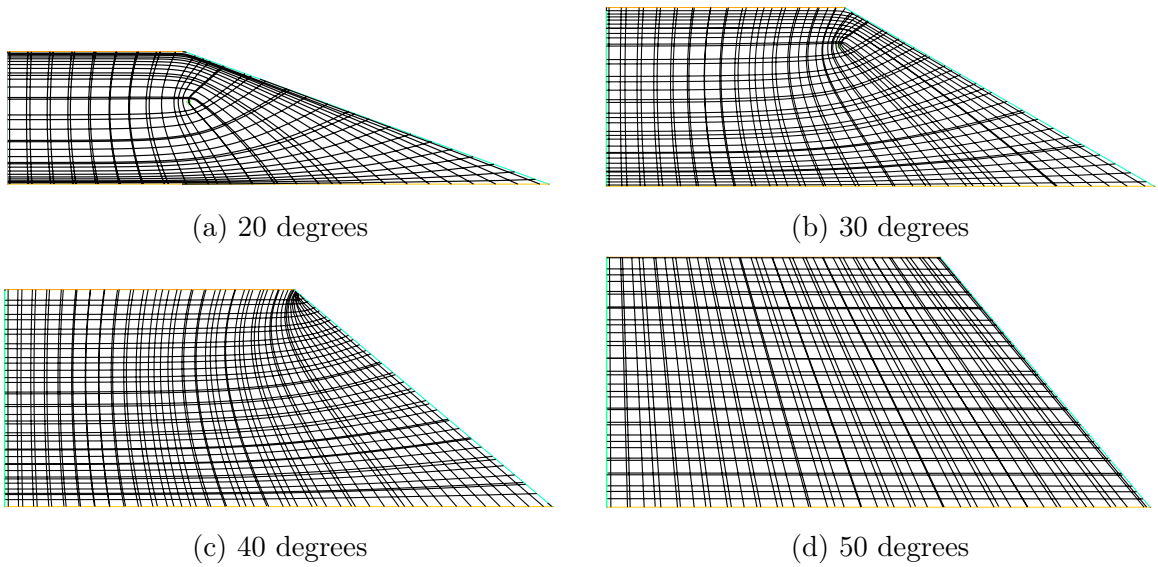
(a) $b = 0$ (b) $b = 10$

Figure 4.4: Results using different values of b (circle)

The best placement for singularities is on the boundary. This is justified by the smoothing energy term of the objective function, which tends to minimise the difference between the coefficients of a cross and those of its adjacent nodes. In most cases, a boundary node will have fewer neighbours than an interior node.
To get a good result, taking $b = 10$ is a good choice (see Figure 4.4).

4.2.2 Trapezoids

The angles mentioned in Figure 4.5 labels correspond to the angles of the corners at the bottom right of the trapezoids. The trapezoid with an angle of 50° is the only one with no singularity. Streamlines starting on one side necessarily end on the opposite side,



(a) 20 degrees (b) 30 degrees

(c) 40 degrees (d) 50 degrees

Figure 4.5: Representation of the valence of different nodes [1]

whereas this is not the case for trapezoids with angles of 40, 30 and 20 degrees. This is the reason why there is a singularity on the cross field.
The fact that it appears for an angle between 40 and 50 degrees is easily justified by the assumptions made earlier.
The trapezoid has an Euler characteristic of 1. From equation (1.14), we know that our problem must respect

$$\sum_k \frac{k}{4}(n_k + n_{bk}) = 1 \tag{4.1}$$

The 2 corners on the left of the trapezoid always have a valence of 1. The bottom right-hand corner, whose angle is varied, also always has a valence of 1. However, the top right-hand corner has a valence of 1 when its angle α is less than 135°, and a valence of 2 otherwise. By replacing this in the previous equation, we obtain :

$$\begin{cases} 1 + \sum_k \frac{k}{4} n_k = 1 & \text{if } \alpha \leq 135° \\ \frac{3}{4} + \sum_k \frac{k}{4} n_k = 1 & \text{otherwise} \end{cases} \tag{4.2}$$

If we want to minimise the number of irregular nodes in the first case, we have $n_k = 0$ if $k \neq 0$. In the second case, we necessarily need at least 1 node of valence 3 inside the geometry. This explains the presence of a singularity on the cross field.

For the trapezoid with an angle of 50 degrees, the conditions on b and leaving some boundary directions free or not, do not change the cross field obtained. For other angles, the crosses must be fixed on the boundary, otherwise an inversion will occur.

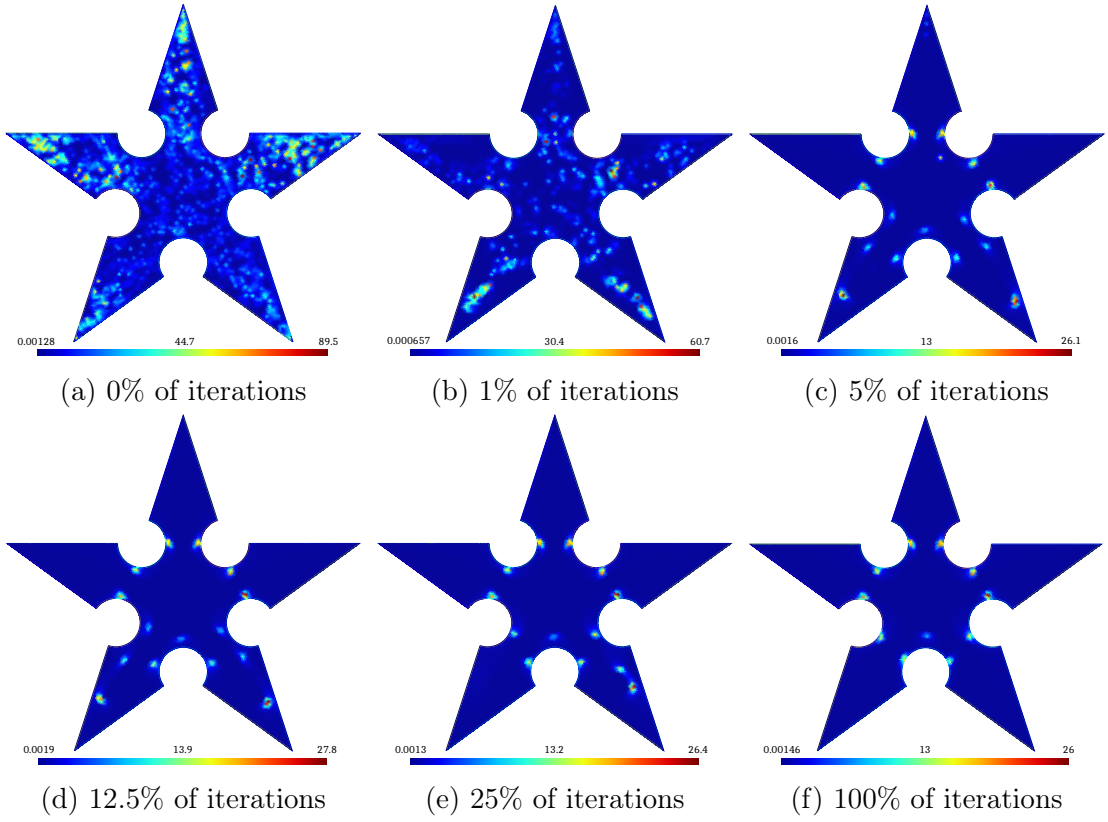
4.2.3 Shuriken

The shuriken is an interesting geometry because it combines curved edges and very acute angles.
Like the previous geometries, the Euler characteristic of the shuriken is 1. With the

crosses of the category 2 nodes imposed, we know that the 15 corners will have a valence of 1, which gives :

$$\frac{15}{4} + \sum_k \frac{k}{4} n_k = 1 \tag{4.3}$$

Ideally, we would like to find 11 singularities that give irregular nodes with $k = -1$ at the end of the algorithm.

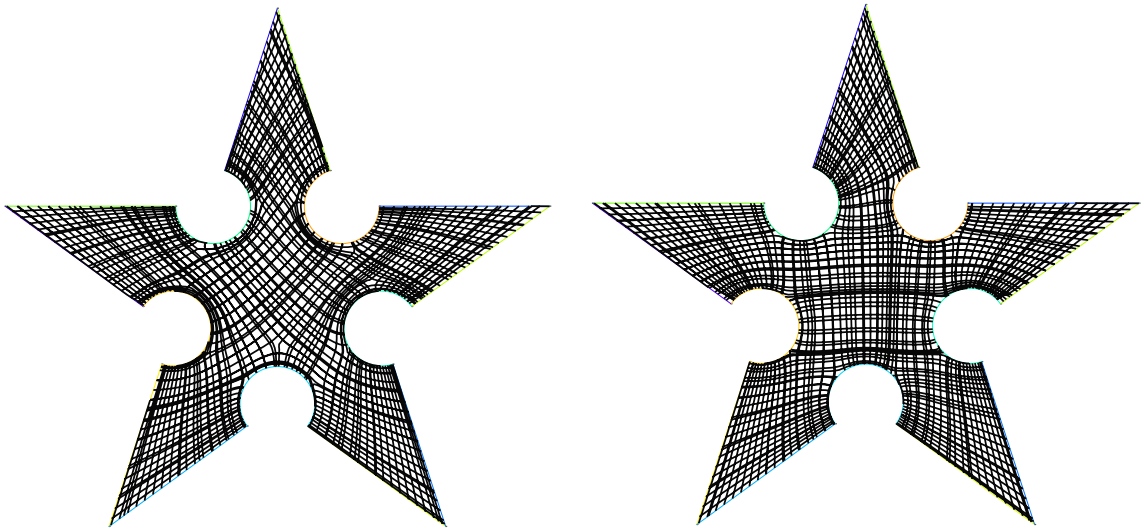


(a) 0% of iterations (b) 1% of iterations (c) 5% of iterations

(d) 12.5% of iterations (e) 25% of iterations (f) 100% of iterations

Figure 4.6: Evolution of the smoothness energy on the domain (Shuriken)

This is indeed the case (see Figure 4.6), but, unlike *Desobry et al*, we do not have a singularity at the centre of the geometry. This is located slightly below.
We could put forward several hypotheses to justify this difference.
Firstly, despite our different initial conditions, it is possible that we still obtain a local minimum.
Secondly, our geometry is not exactly the same as in [6]. This could explain the slight difference in positioning.
However, the result obtained is satisfactory as it minimises the number of irregular nodes as desired.
To obtain these results, some boundary directions were left free. Unlike the disk, the result is better when $b = 0$ (see Figure 4.7).
This is a disadvantage of the method described, as the result depends heavily on the choices (fixed directions and parameter b) made.



(a) $b = 0$ (b) $b = 10$

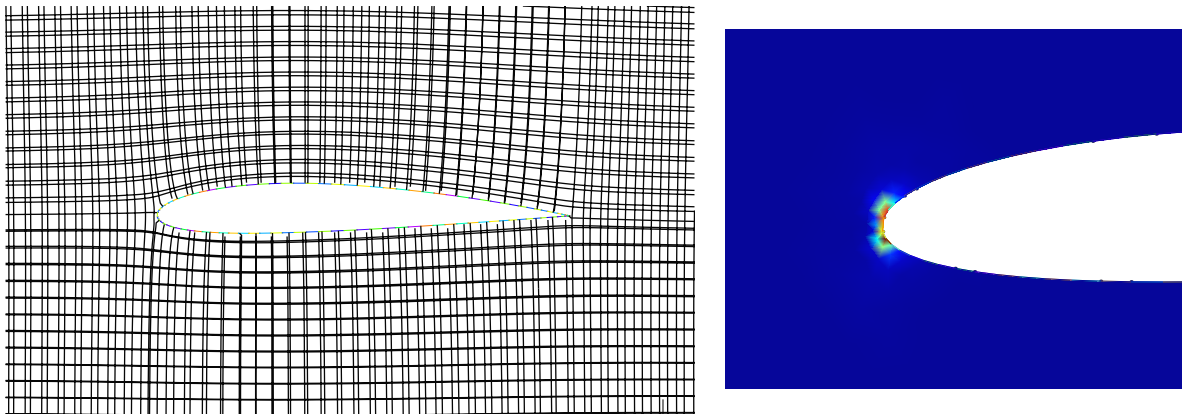
Figure 4.7: Results using different values of b (shuriken)

4.2.4 NACA airfoil 2412

In this subsection, we will study a geometry that is more in line with what an engineer might encounter in his work. We take a NACA airfoil similar to that shown in the intro on Figure 1.1. This geometry has an Euler characteristic of 0. There are 5 corners, 4 of which have a valence of 1, which are not shown in Figure 4.8. In fact, these corners are those of the square which is the outer limit of the "infinite" domain. The fifth corner is the trailing edge angle of the NACA airfoil, which has a valence of 4. So we have :

$$\frac{1}{2} + \sum_k \frac{k}{4} n_k = 0 \tag{4.4}$$

The ideal is therefore to have 2 singularities giving irregular nodes with $k = -1$. This is what we obtain with our algorithm by leaving some boundary directions free and by setting $b = 0$. The 2 singularities are at the front of the NACA airfoil (see Figure 4.8b).



(a) Streamlines (b) Smoothness energy at the front

Figure 4.8: Visual analysis of the results obtained for the NACA airfoil

Chapter 5

Conclusion

In conclusion, through this master thesis, we have explored the world of quadrilateral mesh generation. We focused on a more specific case, namely geometries with very acute and very obtuse angles. We first demonstrated that this type of geometry is present in many areas of engineering. We showed why quadrilaterals can be more appropriate elements than triangles.
We then looked at the literature to discover different approaches to generating quadrilateral meshes. After having seen the advantages and disadvantages of these different methods, we detailed our methodology which is strongly based on *Desobry et al* (2021, [6]). We began by discussing the results we would like to obtain at the boundary of our geometry. We then went on to gradually construct the objective function that allows us to propagate the non-orthogonal crosses on the boundary inside the domain. Once the optimisation problem had been created, we applied it to different geometries. We were able to see that our model has certain limits.

First of all, there are 4 variables per node for our objective function. As a result, obtaining a minimum can be slow. Multiple initial conditions significantly increase total execution time as well. Furthemore, there is not just one minimisation, as we alternate between a minimisation step and a projection step.

Otherwise, when the boundary crosses are not completely fixed, there can be problems with inversion of the branches on the boundary. However, if they are fixed all the time, it is possible not to obtain the ideal mesh. In addition, the model is subject to the adaptation of a parameter (b) by the user. All this means that our method is not completely automated.
It would be ideal to be able to limit the angles of the free branches of the boundary. This would not allow the branches on the boundary to be inverted, but it would favour obtaining the ideal quadrilateral mesh.
To conclude, the methodology presented in this master thesis gives very good results but requires adjustments by the user. This does not fully meet the objective of creating an automatic mesher.

Bibliography

[1] Pierre-Alexandre Beaufort et al. "Computing cross fields A PDE approach based on the Ginzburg-Landau theory". In: *Procedia Engineering* 203 (2017), 219–231. ISSN: 1877-7058. DOI: 10.1016/j.proeng.2017.09.799. URL: http://dx.doi.org/10.1016/j.proeng.2017.09.799.

[2] Steven E. Benzley et al. "A Comparison of All Hexagonal and All Tetrahedral Finite Element Meshes for Elastic and Elasto-plastic Analysis". In: 2011. URL: https://api.semanticscholar.org/CorpusID:59426732.

[3] 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 (Sept. 1991), 811–847. ISSN: 1097-0207. DOI: 10.1002/nme.1620320410. URL: http://dx.doi.org/10.1002/nme.1620320410.

[4] David Bommes, Henrik Zimmer, and Leif Kobbelt. "Mixed-integer quadrangulation". In: *ACM Transactions on Graphics* 28.3 (July 2009), 1–10. ISSN: 1557-7368. DOI: 10.1145/1531326.1531383. URL: http://dx.doi.org/10.1145/1531326.1531383.

[5] Jan Brandts et al. "On angle conditions in the finite element method". In: *SeMA Journal* 56.1 (Sept. 2011), 81–95. ISSN: 2254-3902. DOI: 10.1007/bf03322598. URL: http://dx.doi.org/10.1007/BF03322598.

[6] David Desobry et al. "Designing 2D and 3D Non-Orthogonal Frame Fields". In: *Computer-Aided Design* 139 (Oct. 2021), p. 103081. ISSN: 0010-4485. DOI: 10.1016/j.cad.2021.103081. URL: http://dx.doi.org/10.1016/j.cad.2021.103081.

[7] Haelterman, Robby. "Analytical study of the Least Squares Quasi-Newton method for interaction problems". eng. PhD thesis. Ghent University, 2009, XXI, 224. URL: {http://lib.ugent.be/fulltxt/RUG01/001/333/190/RUG01-001333190_2010_0001_AC.pdf}.

[8] Yang Liu et al. "General planar quadrilateral mesh design using conjugate direction field". In: *ACM Transactions on Graphics* 30.6 (Dec. 2011), 1–10. ISSN: 1557-7368. DOI: 10.1145/2070781.2024174. URL: http://dx.doi.org/10.1145/2070781.2024174.

[9] S. J. Owen et al. "Q-Morph: an indirect approach to advancing front quad meshing". In: *International Journal for Numerical Methods in Engineering* 44.9 (Mar. 1999), 1317–1340. ISSN: 1097-0207. DOI: 10.1002/(sici)1097-0207(19990330)44:9<1317::aid-nme532>3.0.co;2-n. URL: http://dx.doi.org/10.1002/(SICI)1097-0207(19990330)44:9<1317::AID-NME532>3.0.CO;2-N.

[10] Daniele Panozzo et al. “Frame fields: anisotropic and non-orthogonal cross fields”. In: *ACM Transactions on Graphics* 33.4 (July 2014), 1–11. ISSN: 1557-7368. DOI: 10.1145/2601097.2601179. URL: http://dx.doi.org/10.1145/2601097.2601179.

[11] Jeffrey A. Talbert and Alan R. Parkinson. “Development of an automatic, two-dimensional finite element mesh generator using quadrilateral elements and Bezier curve boundary definition”. In: *International Journal for Numerical Methods in Engineering* 29.7 (May 1990), 1551–1567. ISSN: 1097-0207. DOI: 10.1002/nme.1620290712. URL: http://dx.doi.org/10.1002/nme.1620290712.

[12] Ryan Viertel and Braxton Osting. “An Approach to Quad Meshing Based on Harmonic Cross-Valued Maps and the Ginzburg–Landau Theory”. In: *SIAM Journal on Scientific Computing* 41.1 (Jan. 2019), A452–A479. ISSN: 1095-7197. DOI: 10.1137/17m1142703. URL: http://dx.doi.org/10.1137/17M1142703.

[13] Miloš Zlámal. “On the finite element method”. In: *Numerische Mathematik* 12.5 (Dec. 1968), 394–409. ISSN: 0945-3245. DOI: 10.1007/bf02161362. URL: http://dx.doi.org/10.1007/BF02161362.

Appendix A

Use of online tools

A.1 ChatGPT

ChatGPT was used as part of this master's thesis. It was never used for writing this paper but for secondary tasks. Python codes to generate the graphs and some of the LaTex layouts were generated with ChatGPT. It was also used to reformulate explanations given in various papers, or to quote examples when theoretical developments were given. ChatGPT has been used to find well-known formulas such as orthogonal functions, the convexity criterion, etc.

A.2 DeepL

Deepl was used for part of the translation of this master thesis. It was also used to reformulate sentences and find synonyms.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**