BLOCK-STRUCTURED QUAD MESHING FOR SUPERSONIC FLOW SIMULATIONS

Claire Roche[1,3] Jérôme Breil[1] Thierry Hocquellet[1] Franck Ledoux[2,3]

[1] *CEA-CESTA, Le Barp, France. claire.roche@cea.fr, jerome.breil@cea.fr, thierry.hocquellet@cea.fr*
[2] *CEA, DAM, DIF, F-91297, Arpajon, France. franck.ledoux@cea.fr*
[3] *LIHPC, CEA, Université Paris-Saclay, France*

ABSTRACT

Quad meshing is a very well-studied domain for many years. While the problem can be globally considered as solved, many approaches do not provide suitable inputs for Computational Fluid Dynamics (CFD) and in our case for supersonic flow simulations. Such simulations require a very strong control on the cell size and direction. To our knowledge, engineers ensure this control manually using interactive software. In this work we propose an automatic algorithm to generate full quadrilateral block structured mesh for the purpose of supersonic flow simulation. We handle some simulation input like the angle of attack and the boundary layer definition. Our approach generates adequate 2D meshes and is designed to be extensible in 3D.

Keywords: Quadrilateral mesh generation, blocking, supersonic flow, computational fluid dynamics

1. INTRODUCTION

Mesh generation is a critical component of a computational physics based analysis process. The mesh used for a simulation has a considerable impact on the quality of the solution, the stability, and the resources expended to complete the simulations. In this work, we consider the specific field of Computational Fluid Dynamics (CFD) and more precisely supersonic flow simulations. According to Chawner et al. [1], multi-block structured meshes provide the most accurate solutions for CFD. This is among the most popular meshing techniques for flow simulation [2]. But the generation of such meshes is very challenging and time-consuming for high-skilled engineers who can spend weeks or months to generate the adequate mesh using complex interactive tools. It is considered as one of the most time consuming step in the CFD process [1, 3].

The context of our work is the atmospheric (re)entry of a *vehicle* that can be a spacecraft (see Fig. 1 for an example). The geometric domain Ω we consider here is a sphere that surrounds the vehicle and our final goal is to pave the path to automatically generate the adequate block-structured meshes for supersonic flow simulations in 2D and 3D. We focus in this paper on the 2D case with the constraint that the different choices of the proposed solution do not meet specific restrictions to be extended in 3D. Dealing with supersonic flow simulation induces to consider in the meshing process the geometrical shape of Ω, but most importantly several simulation parameters (difference between vehicle front and back, boundary layers, angle of attack) that have a strong impact on the simulation results.

With these constraints in mind, we adapt current quadrilateral meshing techniques. Quad meshing is a very well-studied domain for many years. While the problem can be globally considered as solved if you look at recent results [4, 5, 6, 7], many methods do not provide suitable inputs for supersonic flow simulations. In our case, we require to have a *block-structured mesh* and to control the size, boundary orthogonality and cell direction in some areas. Most of the time, the mesh size can be controlled at the price of losing/degrading the mesh structure, while controlling boundary orthogonality is ensured with interactive software. In this work, we focus on a very demanding field, which is supersonic flow simulation codes. Such aerodynamics application require to handle thin boundary layers around the re-entrance vehicle, to control the cell size and orientation and to capture some shock areas.

1.1 State of the art

Due to the complexity of supersonic flow simulations, the grid density required to obtain the resolution of flow field gradients is unknown *a priori* [8]. Thereby some researchers concentrate on using mesh adaptation during the simulation [9]. Currently, **unstructured meshes** with high cell quality can be generated on complex geometries in a fully automatic way, which saves time. Unstructured meshes are also easier to adapt to specific metrics. But, in CFD, solvers may be less efficient in terms of memory, execution speed, and numerical convergence on this type of mesh topology [10].

Considering that multi-block structured meshes provide the most accurate solutions for CFD [1, 2], therefore those meshes are preferred. However, the generation of such meshes is very challenging and time-consuming for high-skilled engineers, especially in 3D. Fully automatic 3D multi-block structured mesh generation is a complex problem and currently there is no algorithm able to generate an ideal block topology. Then, other types of meshes may be used, such as **over-set grids** [11]. This makes easier the process of mesh generation on complex multi-component geometries. Even if these meshes provide a solution as accurate as the structured ones, it requires specific solvers with complex interpolation. **Hybrid meshes** for CFD (a thin layer of hexahedral elements near the wall and tetrahedral cells in the far field) are easier and faster to generate. Nevertheless, there is no proof that hybrid meshes provide a solution as accurate as block-structured or overset meshes.

In practice, the mesh quality is strongly linked to solver algorithms. Even if the same physic is solved, each solver has its quality criteria [3]. As explained by Chawner et al. [1], the mesh quality criteria for CFD simulations are always stated in a non-quantitatively way. For instance, terms like "*nearly orthogonal*", "*spacing should not be allowed to change too rapidly*", "*give consideration to skewness*", "*sufficiently refined*", "*adequate resolution*" and "*use high aspect ratios*" are used frequently and casually. Mesh generation relies on engineering experience. Thus, it is easier to check if a mesh is not "bad", instead of if it is "good". Indeed, an *a priori* mesh must at least pass the 'validity' requirements of the utilized flow solver (no negative volume cells, no overlapping cells, no void between cells, ...). The VERDICT library [12] is a reference software package for this type of mesh quality evaluation. In fact, the ultimate quality measure for a mesh is the global error measure on the quantity of interest after the simulation.

In order to find a way to generate a 2D quad block structure with an approach that extends to 3D, we can take a look at [4], [13] and [14] that provide complete surveys of existing techniques in 2D, 3D surfaces and 3D. Considering we expect to get a block-structured mesh, polycube-based methods and frame fields seems the most relevant. Polycubes were first used in computer graphics for seamless texturing of triangulated surfaces [15]. Many techniques [16, 17, 18, 19, 20, 21] improved first results. But the orientation sensitivy and the simple structure of a final coarse polycube does not fit our requirement. For several years now, frame fields have offered a promising solution for both quadrilateral and hexahedral mesh generation. They are computed as a continuous relaxation of an integer-grid map with internal singularities (that overcome some limitation of polycubes). The majority of frame field methods have three major steps: first they create and optimize a boundary-aligned frame field; then they generate an integer-grid map, which is aligned with previously defined frame field [22]; and finally, they extract integer isolines (in 2D) or isosurfaces (in 3D) to form an explicit block-structured mesh [23]. To the best of our knowledge, generating a 3D frame field remains challenging and state-of-the-art methods still fail to produce a hex-compatible frame field in 3D.

Considering that our application field is limited to the outer space that surrounds a single vehicle with a zone of interest near the vehicle wall, we can adopt the strategy proposed in [24] where they use an advancing-front approach to mesh such configurations in 3D. Such algorithm, like the paving algorithm in 2D [25] are relevant for our purpose. In the paving method, each boundary is previously meshed. In this work, since we do not have constraints far from the vehicle, only the vehicle wall is pre-meshed. Moreover, we differ from the original paving algorithm in creating new points: starting from a front point p, we transport p along a flow (defined by a vector field) to get the next point.

1.2 Main contributions

Generating adequate quadrilateral mesh for supersonic flow simulation requires to consider both the geometrical shape of the domain Ω but also some simulation parameters like the angle of attack, the thickness of the boundary layer, the distinct behaviour required in the front or the back of the vehicle and so on. Such conditions can be achieved manually in 2D. We propose to do it automatically in this work with the aim to extend it in 3D afterward. That's why our approach relies on the work of X. Roca et al. [24] and extend it to our special case with considering:

1. the boundary layer around the vehicle wall as a special area where we apply specific smoothing and discretization algorithms;
2. several geometry- and physics-based scalar fields that are mixed to control the mesh generation process;
3. test cases are proposed to compare other algorithm on given mesh quality criteria.

2. TERMINOLOGY AND PROBLEM STATEMENT

This work aims to propose an algorithm to automatically generate block-structured quadrilateral meshes for supersonic computational fluid dynamics.

2.1 Supersonic vehicle and environment

Figure 2 shows briefly the traditional flow topology observed during a supersonic flow simulation. The direction of the inflow is represented by the black vectors $\mathbf{u}_\infty$ ($\longrightarrow$)

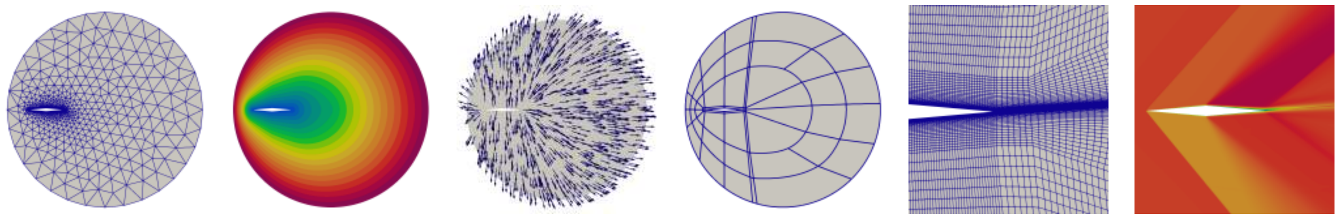

(a) Triangulation (b) Distance Field (c) Vector Field (d) Blocks (e) Mesh (f) Results

Figure 1: The main stages of our approach. Starting from a triangulation of the domain (a), we first generate and combine distance fields (b) and build a vector field that ensure wall orthogonality and the alignment with the angle of attack (c). Using those fields, we generate curved blocks (d) and a final quad mesh where the element size is carefully controlled in the boundary layer (e). Numerical simulation can then be launched (f).

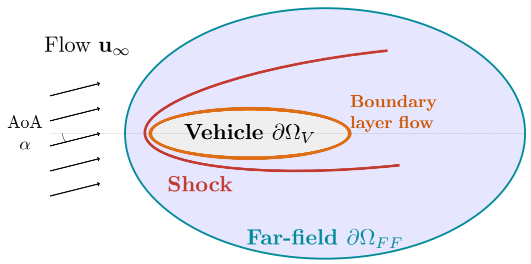


Figure 2: Flow around a supersonic vehicle.

and the angle of attack (AoA) α. Due to the effect of viscosity, a very thin boundary layer plotted in orange (🟧) develops on the wall. This region is characterized by very strong gradients of velocity and temperature. To compute an accurate solution of the Navier-Stokes (NS) equations, very thin and regular cells are needed in the wall-normal direction. Thus, structured mesh are well suited for this area. As few as possible singular nodes (nodes that are not of valence four) are admitted in the orange part of the mesh. In general, for CFD, gradients are calculated more accurately if the cells are aligned to the streamlines, particularly in the boundary layer, and along the shock represented in red (🟥) too. However, unlike boundary layers, mesh refinement is less restrictive near the shock to compute it accurately.

In this work, supersonic bodies are completely immersed in the fluid and a single wall is considered. The far field plotted in blue (🟦) is a smooth boundary (circle, ellipse), far from the physical phenomena to simulate. In this way, the flow structures around the vehicle do not impact the far-field boundary conditions. As the accuracy of the simulation is not needed in this area, there is no hard constraint on cell quality near the far-field boundary.

The thin region in front of the vehicle (on the left side of the vehicle in Fig. 2) is the key part that will govern the simulation. In this very specific zone, the mesh has to be as regular as possible, and the singular nodes are not admitted.

2.2 Approach overview

Let Ω be a 2D domain bounded by an inner boundary, the vehicle wall $\partial\Omega_V$ and an outer spherical boundary, the far-field boundary $\partial\Omega_{FF}$. Let α be the angle of attack of the vehicle, the aim of our approach is to automatically generate a quadrilateral block-structured mesh Q_Ω of Ω that captures the flow around the vehicle and the main flow direction defined by α. Other user parameters are the boundary layer thickness δ_{BL} along $\partial\Omega_V$, an edge size s_w on the wall $\partial\Omega_V$, a size $s_w^\perp$ of the first edge on the wall-normal direction and a global edge size s_G. To this purpose we propose the following method (see Fig. 1):

1. We first discretize the boundary curve $\partial\Omega_V$ (see Section 3.1). This stage requires to preserve geometric corners and maximum block edge size given as an input parameter.

2. Then we build several distance fields in order to drive the advancing front creation of block layers. Those fields are fused into a single one, called d, which have the property that any point $p \in \partial\Omega_V$ verifies $d(p) = 0$ and any point $p \in \partial\Omega_{FF}$ verifies $d(p) = 1$ (see Fig. 1.b and Section 3.2.1).

3. We extract a gradient field ∇d from one of the previously computed distance field and combine it with a constant vector field that represents the flow direction to produce the vector field $\mathbf{v}$ that captures both wall orthogonality and the flow direction (see Fig. 1.c and Section 3.2.2).

4. The scalar field d and the vector field $\mathbf{v}$ drive the creation of a quadrilateral block structure $\mathbf{B}$ where we create each node block in an advancing-front manner. Then curved blocks are created (see Fig. 1.d and Section 3.3).

5. We eventually generate cells of Q_Ω by distinguishing the first block layer where we control size transitions and wall orthogonality and the remaining blocks where we discretize blocks using a transfinite interpolation scheme in each block. To ensure to get edges of size about s_G we apply a simple interval assignment algorithm along non-constrained block edges (see Fig. 1.e and Section 3.4).

3. BLOCK-STRUCTURED MESH GENERATION ALGORITHM

Our approach is inspired by [24] where an advancing-front algorithm is proposed to mesh the outer space around an object. Starting from a set of block corners and edges on the wall of the vehicle, the algorithm uses distance fields and a vector field to control the layer extrusion process. To perform our algorithm, the input is a triangular mesh T_Ω of Ω. The first part of the algorithm aims to build the unstructured quadrilateral block topology, while the second part will produce the final mesh.

3.1 Vehicle wall block discretization

The first stage consists in discretizing $\partial\Omega_V$. To do it we traverse the vertices $v_0, ..., v_m$ of T_Ω located on $\partial\Omega_V$ and we select a vertex v_i if and only if it satisfies one of the following conditions:

- v_i is located on an extremum of the boundary profile, i.e; a vertex of $\partial\Omega_V$ that minimizes or maximizes the x or y coordinate.
- v_i is a geometric corner of $\partial\Omega_V$;
- the curvilinear distance from the previous selected vertex and v_i is greater than the limit length given as an input parameter.

Let us note that this approach does not guarantee that the boundary edges will have the same size. It is not an issue for our process since those edges will be refined later to get the final mesh.

3.2 Fields computation

Distance and vector fields are core components of our approach to drive the layer extrusion. The idea, inspired by [24] is to mix several fields to know how to insert block nodes during the layer creation process. In practice, those fields are discrete and defined at the vertices of T_Ω.

3.2.1 Distance fields computation

As in [24], we compute a distance field d by merging two distance fields: the first one is the distance from the vehicle boundary $\partial\Omega_V$; the second one is the distance from the far-field boundary $\partial\Omega_{FF}$. To compute those fields we solve the Eikonal equation [24] given by

$$\begin{cases} ||\nabla d|| & = f \text{ in } \Omega, \\ d_{|\mathscr{F}} & = 0. \end{cases} \tag{1}$$

In this equation, $\Omega \subset \mathscr{R}^n$ is the physical domain, f is a known function, $||\cdot||$ is the Euclidean norm, $\mathscr{F}$ is the front and d is the distance to this front. In this work, f is considered constant and equal to 1. The problem is solved on T_Ω.

The first field d_V in Figure 3.a is the distance field from the vehicle boundary $(\partial\Omega_V)$:

$$\begin{cases} ||\nabla d_V|| & = 1 \text{ in } \Omega \\ d_{V|\partial\Omega_V} & = 0 \end{cases} \tag{2}$$

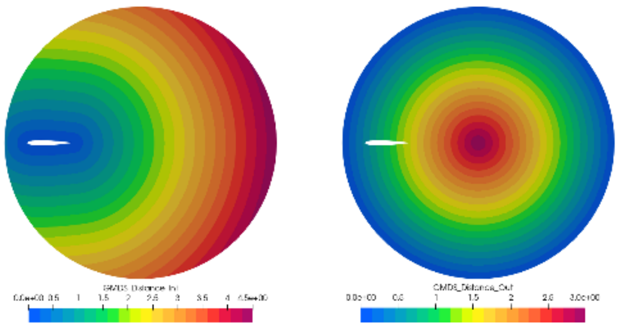
The second field d_{FF} in Figure 3.b is the distance from the far-field boundary $(\partial\Omega_{FF})$ in Figure 2 :

$$\begin{cases} ||\nabla d_{FF}|| & = 1 \text{ in } \Omega \\ d_{FF|\partial\Omega_{FF}} & = 0 \end{cases} \tag{3}$$

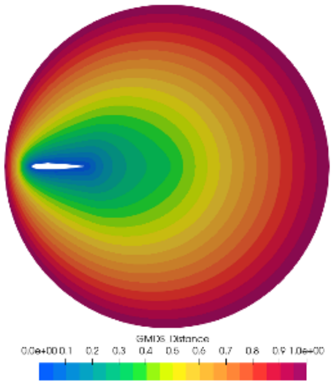
The third field d represented in Figure 3.c is a combination of the two fields d_V and d_{FF}:

$$d = \frac{d_V}{d_V + d_{FF}}. \tag{4}$$

The combination of the two distance fields makes it possible to obtain a field d normalized between $[0, 1]$ on the domain. This field d verifies $0 \leqslant d(x) \leqslant 1$, $\forall x \in \Omega$ and the boundary conditions $d_{|\partial\Omega_V} = 0$ and $d_{|\partial\Omega_{FF}} = 1$. This mixed field ensures us to reach the far-field for the same layer during the extrusion, it prevents the front to divide.



(a) d_V (b) d_{FF}



(c) d

Figure 3: The distance fields computed on the NACA 0012 airfoil geometry [26].

3.2.2 Vector field computation

In combination with the previously defined distance field d, we compute several vector fields to drive the layer creation. In supersonic flow simulation, we must particularly care about the front of the vehicle, its near boundary and global mesh direction in the back of the vehicle. We drive the mesh behaviour in the back area with the angle of attack α, which is flow-related information. To this purpose we define the vector field $\mathbf{u}_\infty$ as being constant on Ω and equals to the far-field flow direction, $\mathbf{u}_\infty = \begin{pmatrix} cos(\alpha) \\ sin(\alpha) \end{pmatrix}$.

For the front of the vehicle, we consider two possible options based on the previously-computed distance fields. The first vector field we use is the gradient of the distance field d_V, noted ∇d_V, and the second one is the gradient of the mixed distance field d, noted ∇d. We compute these vector fields at the vertices of T_Ω with the Least Squares fit of Directional Derivatives (LSDD) method [27]. Let us note $\mathbf{v_{front}}$ the vector field selected between ∇d_V and ∇d.

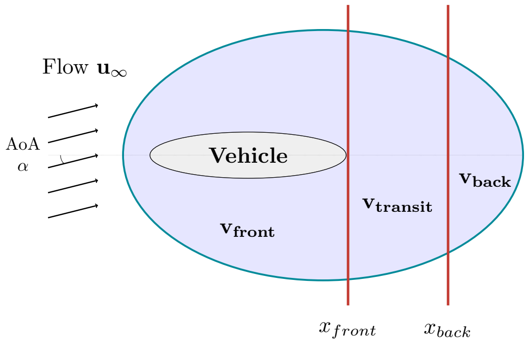


Figure 4: Linear transition between the front and back vector fields.

In order to consider both the front and back vector fields, we eventually compute the vector field $\mathbf{v}$ as a linear combination of those two vector field in a transition area (see Fig. 4). Two physical limits are set by the user in Ω, x_{front} and x_{back}. For each node $n_i \in T_\Omega$ at point $p = \{x_i, y_i, z_i\}$

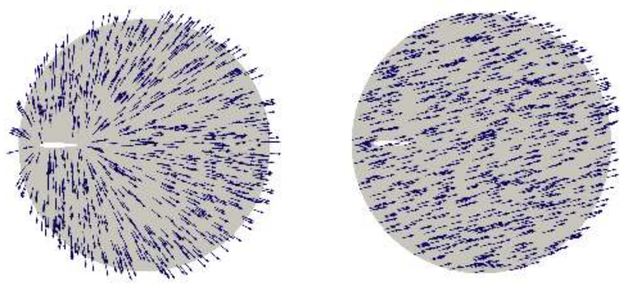
- if $x_i < x_{front}$, then $\mathbf{v_i} = \mathbf{v_{front,i}}$,
- if $x_i > x_{back}$, then $\mathbf{v_i} = \mathbf{v_{back,i}}$,
- if $x_{front} \leqslant x_i \leqslant x_{back}$, then $\mathbf{v_i} = (1-\theta)\mathbf{v_{front,i}} + \theta\mathbf{v_{back,i}}$,

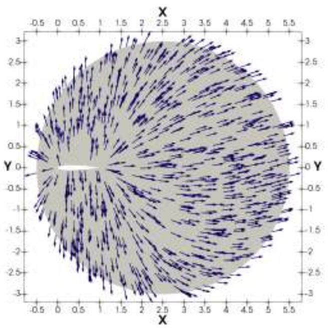
where $\theta = \frac{x_i - x_{front}}{x_{back} - x_{front}}$ is a damping parameter between 0 and 1. Figure 5 illustrates some of the different vector fields we use. Note that by default, we normalize all the vector fields as we only use the field direction and not its magnitude.

3.3 Blocking generation

We build the block structure using an advancing-front approach (see Algorithm 1). We know *a priori* the number of layers $N_{\mathscr{L}}$ that will be generated. At each step i, we build a complete layer of quadrilateral blocks, that we denote $\mathscr{L}_i$. The new inserted nodes and edges define the *extrusion front*, noted $\mathscr{F}_i$, built from the nodes of the front $\mathscr{F}_{i-1}$. They share common properties:

- All the nodes of $\mathscr{F}_i$ are at the same distance $d_{\mathscr{F}_i}$ considering the distance fields computed in Section 3.2.1. We use the distance field d_V for $\mathscr{F}_1$ and the distance field d for $\mathscr{F}_i$ with $i > 1$. This way, we ensure the nodes of the front $\mathscr{F}_1$ are all at a distance superior to the boundary layer thickness imposed by the user. In another hand, all the nodes $\mathscr{F}_i$ are on a same level set and the front can not separate.
- Each node of $\mathscr{F}_i$ is connected by edges to two other nodes on the front of $\mathscr{L}_i$;
- The front $\mathscr{F}_i$ forms a single loop.


(a) ∇d_V (b) $\mathbf{u}_\infty$



(c) $\mathbf{v}$

Figure 5: Vector fields computed on the NACA 0012 geometry. The vector field (c) is a mix from ∇d_V (a) for $x < 1.5$, and $\mathbf{u}_\infty$ (b) for $x > 5.0$. The damping zone is for $x \in [1.5, 5.0]$ and the angle of attack α is equal to 15°.

We generate the blocking structure by inserting one layer at a time in an independent way. The process is the same for all the layers (line 4 of Algorithm 1 and Section 3.3.1) except for the first boundary layer (line 1 of Algorithm 1 and Section 3.3.2).

Algorithm 1 Extrusion algorithm

```
Input: $T_\Omega$, $\mathscr{F}_0$ nodes, distance field $d$, distance field $d_V$,
    vector field $\mathbf{v}$, boundary layer thickness $\delta_{BL}$
Output: Blocking $\mathbf{B}$
1: $\mathscr{L}_1 \leftarrow$ compute1stLayer($\mathscr{L}_0$, $d_V$, $\mathbf{v}$, $\delta_{BL}$)
2: layer_step $\leftarrow 1/N_{\mathscr{L}}$
3: for all $i \in 2, ..., N_{\mathscr{L}}$ do
4:     $\mathscr{L}_i \leftarrow$ computeLayer($\mathscr{L}_{i-1}$, $d$, $\mathbf{v}$, i*layer_step)
5: end for
```

3.3.1 Block layer generation

We generate a complete layer of quadrilateral cells following Algorithm 2. The algorithm starts from the first front of nodes and edges $\mathscr{F}_0$ which corresponds to the discretization of $\partial\Omega_V$.

Front node location. Let us consider the generation of $\mathscr{F}_i$ from $\mathscr{F}_{i-1}$. For each node n^j_{i-1} of $\mathscr{F}_{i-1}$, we compute the ideal location of the next node n^j_i on $\mathscr{F}_i$ (Line 3 of the Algorithm 2) by solving the advection equation

$$\frac{\partial \mathbf{OM}}{\partial t} = \mathbf{v} \tag{5}$$

using a 4th order Runge-Kutta method. The origin node $\mathbf{O}$=n^j_{i-1} is advected along the direction of the vector field $\mathbf{v}$, until his distance d_{n^j} in the distance field d reach $d_{\mathscr{F}_i}$. We obtain then the point $\mathbf{M}$=n^j_i. Unlike [24], we define the position of a new node by decoupling the distance to be covered, provided by the distance field d, from the direction to be followed, provided by the vector field $\mathbf{v}$. This way, characteristics of the flow such as the angle of attack α are taken into account in the vector field built for the extrusion.

Once those positions computed, we check some validity rules to ensure that the quad blocks of the layer $\mathscr{L}_i$ have the adequate shape. Those rules are similar to the ones introduced by Blacker et al. [25] but used both geometric and physical criteria to classified nodes of $\mathscr{F}_{i-1}$: we consider the geometrical shape of the quadrilaterals and the alignment with vector field $\mathbf{v}$. According to this classification, we are going to insert or erase some nodes in $\mathscr{F}_i$. This process ensures a strong property of the computed layer: all the nodes of $\mathscr{F}_i$ are at the same distance $d_{\mathscr{F}_i}$ along the input distance field. This property ensures that the front cannot separate and that all the nodes will reach the outer boundary at the same time on the last layer.

Algorithm 2 ComputeLayer

```
Input: $T_\Omega$, $\mathscr{F}_i$ nodes, distance field $d$, vector field $\mathbf{v}$, distance $d_{\mathscr{F}_i}$ of the nodes in the distance field $d$
Output: Quad blocking, and a set of nodes and edges of the layer $\mathscr{F}_{i+1}$
 1: for all node $n^j_i \in \mathscr{F}_i$ do
 2:     $n^j_{i+1} \leftarrow$ ComputeIdealPosition($n^j_i$, $d_{\mathscr{F}_i}$, $d$, $\mathbf{v}$)
 3: end for
 4: while there is a singular node $n^k_i \in \mathscr{F}_i$ do
 5:     $n^k_i \leftarrow$ getSingularNode($\mathscr{F}_i$, singu_type)
 6:     if singu_type is 0 then
 7:         $\mathscr{F}_{i+1} \leftarrow \mathscr{F}_{i+1} + \{$insertQuadAtPoint($n^k_i$)$\}$
 8:     else if singu_type is 1 then
 9:         $\mathscr{F}_{i+1} \leftarrow \mathscr{F}_{i+1} + \{$contractQuadAtPoint($n^k_i$)$\}$
10:     end if
11: end while
```

Figure 6 represents the extrusion of regular blocks on a layer. In Figure 6.a, a small part of $\mathscr{F}_{i-1}$ (where i = 3) is plotted in red (━) and previously generated blocks (previous layers) in light blue (🟦). If there is no conflict on the layer due to block expansion or shrinking, then all the blocks are built in a regular way, and the front nodes of layer $\mathscr{L}_3$ are now the input for another step of the Algorithm 2 (see Fig. 6.b).

Block insertion. To avoid blocks of poor quality, we allow the insertion of blocks in areas specified by the user. As explained before, to create a layer $\mathscr{L}_i$, each node of $\mathscr{F}_{i-1}$ generates a node of $\mathscr{F}_i$ at the distance d_i in the distance field d, following the vector field $\mathbf{v}$. Let us consider two nodes n^j_{i-1} and n^k_{i-1} of $\mathscr{F}_{i-1}$ sharing an edge. They are respectively going to generate the nodes n^j_i and n^k_i of $\mathscr{F}_i$, and we may insert the block defined by $(n^j_{i-1}, n^j_i, n^k_i, n^k_{i-1})$. Depending on quality angle, we can reject this block. For instance, on Figure 7.a, the node n^0_2 will generate the node n^0_3 and the two adjacent blocks will not respect our angle quality. As a consequence, we would generate two nodes from n^0_2 and create an extra block (see Fig. 7.b). In this work, an extra block is inserted if the four following criteria are respected (using notations given on Fig. 7):

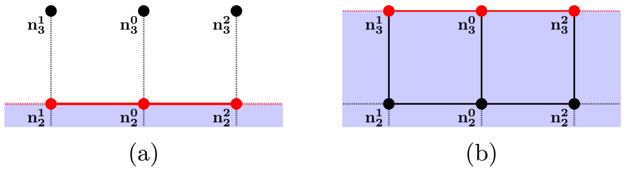


Figure 6: Regular layer computation.

1. The node is in an area where the insertion is allowed by the user;

2. The adjacent nodes on the same red layer $\mathscr{L}_2$ have not already inserted elements;

3. $\frac{\pi}{4} - \sigma_1 < \arccos\left(\frac{\mathbf{w}\cdot\mathbf{v_n}}{||\mathbf{w}||\cdot||\mathbf{v_n}||}\right) < \frac{\pi}{4} + \sigma_1$, where $\mathbf{v_n}$ is the value of the vector field at the position of the node;

4. $\arccos(\frac{\mathbf{w}\cdot\mathbf{a}}{||\mathbf{w}||\cdot||\mathbf{a}||}) + \arccos(\frac{\mathbf{w}\cdot\mathbf{b}}{||\mathbf{w}||\cdot||\mathbf{b}||}) > \frac{3\pi}{2}$.

Where $\sigma_1 = 0.174$ is an arbitrary tolerance corresponding to 10°. It is common to take the aspect ratio between two opposite edges as an insertion or shrinking criterion. However, for the applications of this work, there is no constraint on this specific ratio.

To compute the position of one of the two new nodes that are used to create the inserted block, we proceed as follows. To build the node n^3_3, the point n^0_2 in Figure 7 is advected following the constant vector $\frac{\mathbf{w}}{||\mathbf{w}||} + \frac{\mathbf{a}}{||\mathbf{a}||}$ until reaching the distance $d_{\mathscr{F}_i}$ in the distance field d. We do the same for the second point.

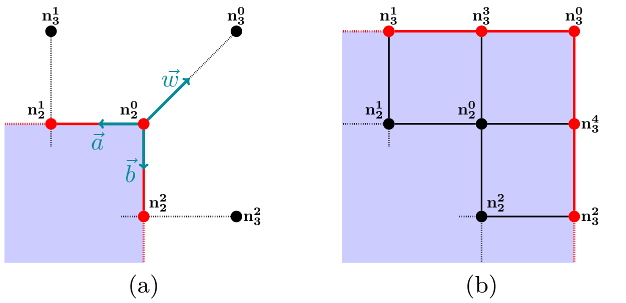


Figure 7: Block insertion.

Block shrinking. The block shrinking operation is the opposite of the block insertion. Considering three consecutive nodes n^j_{i-1}, n^k_{i-1} and n^ℓ_{i-1} of $\mathscr{F}_{i-1}$, that respectively generate the nodes n^j_i, n^k_i and n^ℓ_i of $\mathscr{F}_i$, we apply the shrinking process when we meet the configuration of Figure 8.a, where the three generated nodes are geometrically close. More specifically, we fuse the three generated nodes into a single one. To detect the places where the operation is necessary, the proximity of the ideal positions of the nodes of the next layer is controlled with a tolerance. After the fusion, the adjacent nodes on the layer (connected by an edge) are not able to perform an insertion or fusion operation.

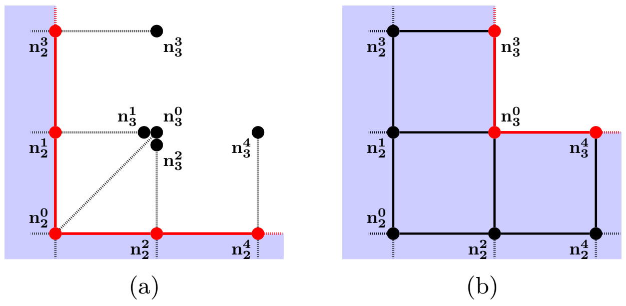


Figure 8: Block shrinking.

Figure 9 illustrates how blocks can be inserted and shrunk in a whole domain on the mars spacecraft geometry [28].

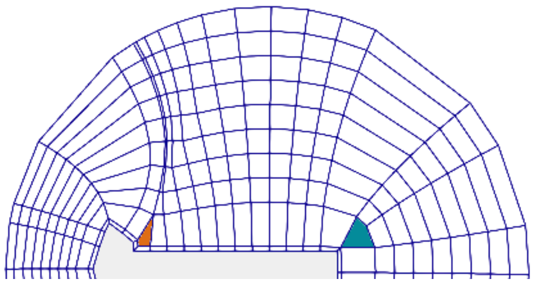


Figure 9: Block shrinking in orange (🟧) and insertion in blue (🟦) on the second layer.

3.3.2 Boundary Layer Extrusion

The boundary layer is a thin layer close to the wall where the fluid flow is dominated by viscosity effects. We take a particular care to this area, which we manage with the Algorithm 3. The distance field considered is d_{Ω_V}, and the distance of the layer is supposed to be higher than the thickness of the boundary layer δ_{BL}.

Boundary layer insertion. As explained before, insertions may be performed in the boundary layer. This operation must remain as occasional as possible to not introduce many singularities in the near boundary layer. The insertion is performed only in the case there is a very sharp angle on the geometry (for example the NACA 0012 airfoil in Fig. 3). This insertion is always a two-block insertion

Algorithm 3 Compute1stLayer

```
Input: $T_\Omega$, $\mathscr{F}_0$, expected boundary layer thickness $\delta_{BL}$,
    distance field $d_{\Omega_V}$, vector field $\mathbf{v}$
Output: Quad blocks of the layer $\mathscr{L}_1$
1: for all node $n^j_0 \in \mathscr{F}_0$ do
2:     $n^j_1 \leftarrow$ ComputeIdealPosition($n^j_0$, $\delta_{BL}$, $d_{\Omega_V}$, $\mathbf{v}$)
3: end for
4: $\mathbf{B} \leftarrow$ ComputeBlocks($\mathscr{L}_1$)
```

and is performed as shown in Figure 10. In Figure 10.a, the front considered for the extrusion is $\mathscr{F}_0$, and the nodes of this front are plotted in red. Let us remember that $\mathscr{F}_0$ is on the wall geometry. Each node of $\mathscr{F}_0$ computes the ideal position of the next node. At the position of the node n^0_0, a sharp angle is detected on the geometry surface. Then, two blocks are inserted. If the inserted upper right block of Figure 10 is considered, the two new block corners are placed this way. The first one, connected to n^1_1 by a block edge, is the position $p_{n^0_0}$ of the node n^0_0 advected at the distance δ_{BL} in the distance field d_{Ω_V} following a constant vector field equal to the vector $\mathbf{c_1}$ in Figure 10.a. The second block corner is placed at the position $p = p_{n^0_0} + l_1 \frac{\mathbf{c_1}+\mathbf{w}}{||\mathbf{c_1}+\mathbf{w}||}$. The second block of this insertion is built the same way, from the node n^2_0 on the other side of n^0_0. Figure 11 illustrates how this two-block insertion is performed on the boundary layer blocking.

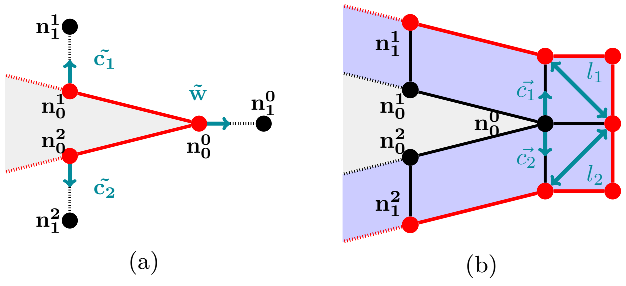


Figure 10: Block insertion on the boundary layer.

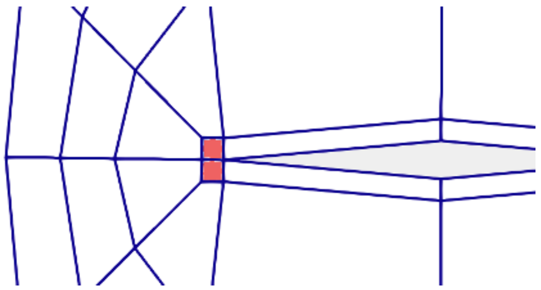


Figure 11: Insertion of two blocks (🟥) on the boundary layer for the diamond airfoil. This insertion in the front of the airfoil to ensure good quality elements in this sensitive area.

3.4 From blocks to quadrilaterals

Once the block structure built, we generate the final quad mesh. It requires to assign the right discretization to every block edges and to discretize each block with the appropriate regular grid. The boundary layer is meshed considering strong constraints about wall-orthogonality and aspect ratio.

3.4.1 Interval assignment

The interval assignment algorithm aims to select the number of mesh edges for each block edge. This is fundamentally an integer-valued optimization problem that was tackled in several works [29, 30, 31]. In particular, an incremental interval assignment using integer linear algebra is proposed by [31] and gives very satisfying results in terms of target size respect and speed performance. In this work, we follow a simple procedure that we describe thereafter. Even if the problem is initially composed of N integer unknowns, with N the number of block edges, it can be reduced considering the topological chords of the blocking. A topological chord $\mathscr{C}$ is defined as a set of opposite edges [32] (see Fig. 12). As a conformal mesh is expected, all the edges $\{e_i\}_{i=1..n_c}$ of the same chord $\mathscr{C}$ need to have the same discretization. Otherwise, the blocking discretization is not valid.

Then, starting from a block structure composed of N edges, the problem can be reduced to n integer variables, where n is the number of topological chords in the block structure. For instance, in the case of Figure 12, there are eight chords hence eight integer unknowns.

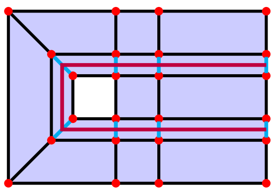


Figure 12: Purple topological chord $\mathscr{C}$ composed of the blue edges of the blocking.

This number of unknown is going to decrease again due to our application case where the thin boundary layer along the vehicle wall is handled specifically. Let us consider Figure 13 where the blue edges are on the vehicle wall and the green ones in the boundary layer. The discretization of the blue edges is controlled by an input parameter s_w that fixes a target length of each blue edge, and so propagate along the corresponding chords. The discretization of the green edges helps to capture the boundary layer flow. This discretization is again an input parameter, which strongly depends on the simulation. Again some unknowns are so removed.

It is important to notice that if two edges of a chord $\mathscr{C}_i$ have different hard constraints, the problem can not be solved and the mesh is not generated. In our case, it should not happen. The simple structure of our problem (full conform block structure) and the few number of hard constraints allows us in practice to avoid to build over-constrained systems.

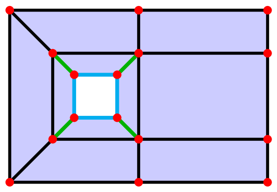


Figure 13: The block edges hard constrained in our algorithm are set in blue and green.

3.4.2 Boundary layer meshing

Boundary layer discretization. This part is about the discretization of the topological chords constrained by the blue and green edges in Figure 13. In this part, the block edges are linear. The objective of the blocking is to split the domain into a small set of blocks. As a direct consequence, when the geometry is curved (as the NACA 0012 airfoil), we do not obtain a good discretization of the vehicle wall and some parts of the boundary layer can be totally out of Ω. To solve it, we first create the mesh edges corresponding to the blue block edges in Figure 13: each block edge is linearly split by inserting k points $\{p_i\}_{i=1,\dots,k}$, that are projected onto $\partial\Omega_V$ afterward. For each point p_i, we keep the offset vector $\mathbf{v}_i^{proj}$ used to project p_i onto $\partial\Omega_V$. As the boundary layer is very thin, we apply the same offset vector $\mathbf{v}_i^{proj}$ for the mesh points used to linearly discretized the opposite block edge (see Fig. 14). By this way, we avoid to generate tangled meshes.

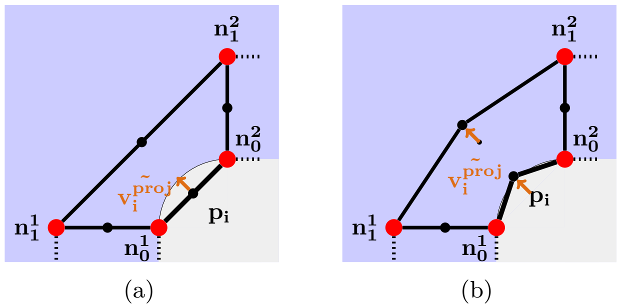


Figure 14: Boundary layer offset.

For the discretization of the boundary layer, we require three input user parameters, which are: s_w, the size of the final mesh edges on the wall vehicle; $n_w^\perp$, the number of mesh edges in the wall-normal direction; $s_w^\perp$ the size of the first mesh edge in the wall-normal direction. With these parameters, the edges are set uniformly in the streamwise direction.

Boundary layer smoothing. Even if the block edges were placed in an orthogonal way, the computation remains local for each node. As a consequence, there is no reason for the resulting mesh to be orthogonal to the wall (see

Fig. 16.a). At this stage, we perform a smoothing algorithms on the boundary layer blocks that have an edge on $\partial\Omega_V$. This smoothing aims to enhance the orthogonality of the first cells in the block. The smoothing algorithm [33] is performed on each block. It is a modification of the Line-Sweeping method introduced by J. YAO [34], which was specifically developed for structured meshes.

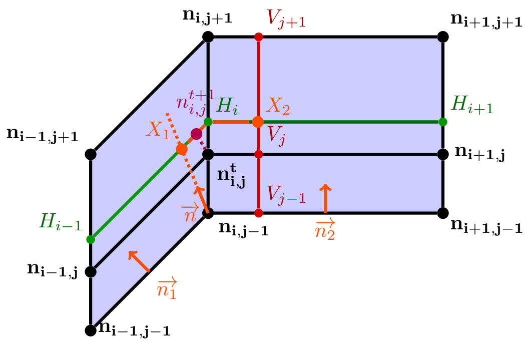


Figure 15: Modified Line-Sweeping method on an internal node in a block.

The Line-Sweeping method is a geometric, local, iterative, and fully explicit method that aims to uniformize the cell sizes of a block. Let **B** be a block of size $N_x \times N_y$, and $n_{i,j}$ be a node in **B** with $0 < i < N_x - 1$ and $0 < j < N_y - 1$[1]. To compute the new position at the iteration $t + 1$ by the Line-Sweeping, we consider the stencil made of the six black nodes on Figure 15. From this stencil, six points are computed, the three plotted in red (—) (V_{j-1}, V_j, V_{j+1}) and the three in green (—) (H_{i-1}, H_i, H_{i+1}). Red points are placed in the middle of each vertical branch. For instance, V_{j+1} is in the middle of the branch made up of the three nodes $n_{i-1,j+1}$, $n_{i,j+1}$, $n_{i+1,j+1}$. In the same way, the three green points are placed in the middle of each horizontal branch. For instance, H_{i-1} is at the middle of the branch $n_{i-1,j-1}$, $n_{i-1,j}$, $n_{i-1,j+1}$. From these six points, two branches of two segments each are built, the red one (V_{j-1}, V_j, V_{j+1}) and the green one (H_{i-1}, H_i, H_{i+1}). The Line-Sweeping places the new position of the node $n_{i,j}$ at the iteration $t+1$ as being the intersection of these two branches, represented by the orange point X_2. A damping coefficient $\theta_d \in [0, 1]$ chosen by the user can be added to enhance the convergence.

As the Line-Sweeping does not provide the near-wall orthogonality needed for this work, a modification was introduced in [33]. Assuming the block edge on the wall is at the index $j = 0$. For each node $n_{i,j}$ as the one in Figure 15, we compute the position X_2 with the Line-Sweeping method, and another orange point (X_1) is placed. Two vectors $\mathbf{n_1}$ and $\mathbf{n_2}$ ($\longrightarrow$) are computed, normal to the respective segments $[n_{i-1,j-1}, n_{i,j-1}]$, and $[n_{i,j-1}, n_{i-1,j-1}]$. Then, the sum $\mathbf{n} = \mathbf{n_1} + \mathbf{n_2}$ is considered to place the point X_1. This new point is at the intersection of the dashed orange line (- -) passing through the point $n_{i,j-1}$ and carried by the vector, and the green branch. A new orange branch $X_1, n^t_{i,j}$, X_2 (—) is considered. According to the index j of the node considered in the block, the new point $n^{t+1}_{i,j}$ is placed on the orange branch at the position $p^{t+1}_{i,j} = \alpha\gamma X_1 + (1-\gamma)X_2$. In this work, $\gamma = (\frac{j-1}{6(N_y-1)})^{0.01}$. This way, the closer the node is to the wall, the stronger the orthogonality is. Figure 16 illustrates how this smoothing stage improves the wall orthogonality.

[1] which means $n_{i,j}$ is not a boundary node.

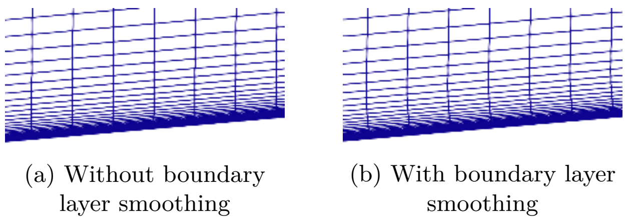

(a) Without boundary layer smoothing (b) With boundary layer smoothing

Figure 16: Comparison of the near wall mesh before and after smoothing.

Boundary layer refinement. In the wall-normal direction, a refinement law is performed on any chord containing a block with a block edge on the geometry (the green edges of Fig. 13). This implies the cells can be very anisotropic which is not a problem since gradients are in the wall-normal direction. Considering a vector of adjacent nodes $n_1, ..., n_{N+1}$. Each node n_i is a 1D point at the position l_i. According to the refinement law used, the new position of the node n_i is given by

$$l_i = l_1 + f_n(l_{N+1} - l_1) \tag{6}$$

where $f_n = 1 + \beta\frac{1-e^p}{1+e^p}$, $p = z(1 - \frac{i-1}{N})$, $z = log(r)$ and $r = \frac{\beta+1}{\beta-1}$.

From this law and a set of adjacent edges, the β parameter can be computed using a Newton method and three values: the sum of the length of the edges, the length of the first edge $s^\perp_w$, the number of nodes $n^\perp_w$. This refinement law is particularly adapted to the boundary layer where the size of the first cell can be very small. It avoids to generate too large cell size far from the boundary layer.

3.4.3 Block discretization

In the case there is no hard constraint on the given chord $\mathscr{C}$ composed of the edges $e_0, ..., e_{n_c}$, we get the number of mesh edges for the chord by minimizing:

$$F(t) = \sum_{e_i} \omega_i (t - T_i)^2, \tag{7}$$

where ω_i is the weight of the edge e_i and T_i is the ideal discretization of the edge e_i. To compute T_i, there is a target parameter s_G corresponding to the ideal default size of the edges of the final mesh. F is a second-degree polynomial in t made of positive terms that reaches a minimum when $\frac{F(t)}{\partial t} = 0$. We have

$$\frac{F(t)}{\partial t} = 2\sum_{e_i} \omega_i (t - T_i). \tag{8}$$

So the minimum is

$$t_0 = \frac{\sum_{e_i} \omega_i T_i}{\sum_{e_i} \omega_i} \quad (9)$$

Then, we choose the closest integer from t_0 as a solution and so the discretization of the edges of the chord $\mathscr{C}$.

3.4.4 Final curved blocking

It remains to finally mesh the blocks that are not in the boundary layers. All the block nodes are created and located using the advancing-front algorithm described in Section 1. To avoid discontinuities and low-quality cells, some block edges are not discretized linearly between two block nodes. In fact, we curve every block edge that has its two end points located on the same front $\mathscr{F}_i$ with $i > 1$. To do it, we build a control point p_C and the edge is represented by a quadratic Bézier curve.

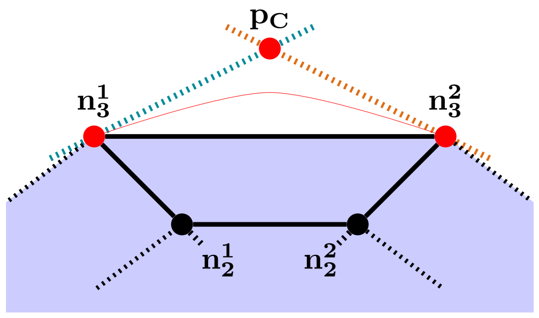
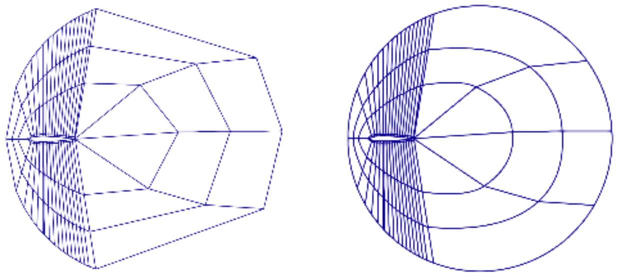


Figure 17: Curve block edges as a quadratic Bézier curve.

Let us consider the example of Figure 17, the block corners n_3^1 and n_3^2 are on the same front $\mathscr{F}_i$ and $i > 1$. Then, the block edge between them can be curved. To build the quadratic Bézier curve, we choose to insert the control point p_C as the intersection of two lines, the one plotted in blue (- -) and the one in orange (- -). The blue line is defined by the point at the block corner n_3^1, and the vector normal to $\mathbf{n_2^1 n_3^1}$. Then, the Bézier curve is controlled by (n_3^1, p_C, n_3^2). Figure 18 shows how the mesh blocks are curved with this procedure. After that, every edge e included curved ones, are subdivided according to the number of subdivision assign to their chord (see Section 3.4.3). We finally perform a transfinite interpolation scheme to generate the grid structured mesh in each block that is not located in the boundary layer.

4. RESULTS AND APPLICATIONS

To demonstrate the well-behaviour of our approach, we tested it onto different types of vehicles. Here, we focus on a selected set of samples but our heuristic was evaluated on a larger data set. We first checked the mesh quality and the impact of some key parameters as the angle of attacks and the ability to insert/contract quadrilateral blocks in each layers. Then we consider two validation cases. The first one is the well-studied case of the NACA 0012 airfoil [26]. The second one is a two-dimensional supersonic



(a) Linear blocking (b) Curved blocking

Figure 18: From linear (a) to curved blocks (b).

flow around a diamond-shaped airfoil. For this case, analytical solutions are available [35], [36]. The simulations are run using the open multiphysics simulation software SU2 [37] and our meshing algorithm is freely available and implemented in the open-source C++ meshing framework GMDS [38][2].

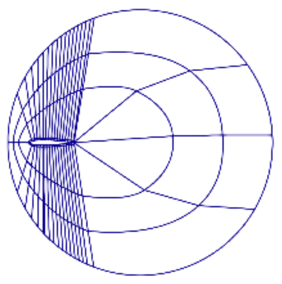
4.1 Mesh quality

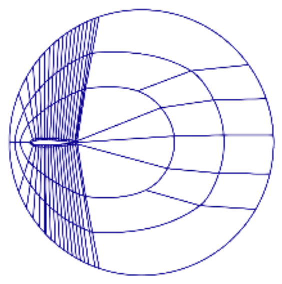
Figures 19.a and 19.b illustrate the block structures of two meshes generated with our algorithm. To generate these meshes, the used angle of attack is $\alpha = 0^\circ$. For the vector field computation, the transition area is set between $x_{front} = 1.5$ and $x_{back} = 5.0$. For the boundary layer meshing, we require a thickness of $\delta_{BL} = 4 \times 10^{-2}$m, $n_w^\perp = 100$ cells in the wall-normal direction and the size of the first cell at $s_w^\perp = 1 \times 10^{-8}$m. The minimum number of block corners on the wall is set to 33 and the number of layers is set to $N_\mathscr{L} = 4$. The size of the edges streamwise on the wall is $s_w = 1 \times 10^{-3}$m. In the rest of the domain, the edge size is set by default to $s_G = 1.2 \times 10^{-2}$m. The only difference between those two meshes is the permission to insert blocks during the layer extrusion process for the block-structure generation (see Fig. 19.b). The number of cells with high scaled jacobian (as defined in VERDICT [12]) increases for the mesh generated with inserted blocks (see Fig. 19.c and 19.d).

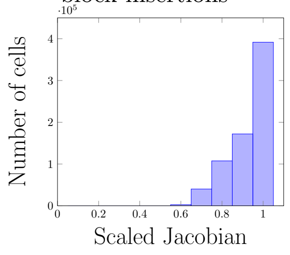
Figure 20 represents the same mesh generated in Figure 19.b but with an angle of attack $\alpha = 15^\circ$. Unlike the previous blocking, we see the block edges align with the flow behind the airfoil in Figure 20.a. Figure 20.b shows the quality of cells in the mesh. In comparison with Figure 19.d, the geometric quality of cells is not degraded by taking into account this angle.

Figure 21 shows a mesh generated on the diamond airfoil geometry. For this generation, the angle of attack is set to $\alpha = 0^\circ$. The vector field is computed with the parameters $x_{front} = 1.5$m and $x_{back} = 6.0$m. The number of layers is set to $N_\mathscr{L} = 4$ and the block insertions are allowed. For the boundary layer, a thickness of $\delta_{BL} = 5 \times 10^{-2}$m is required, with $n_w^\perp = 100$ cells in the wall-normal direction and the size of the first wall-normal edge is set to $s_w^\perp = 1 \times 10^{-9}$m. The size of the edges on the wall is set

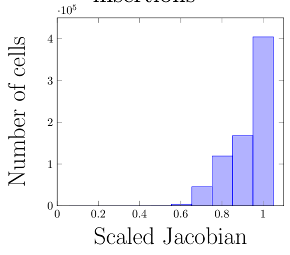
[2]https://github.com/LIHPC-Computational-Geometry/gmds.


(a) Blocking without block insertions


(b) Blocking with block insertions

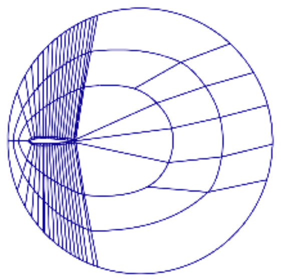


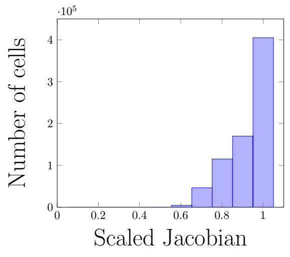
(c) Scaled jacobian of the cells of blocking (a)



(d) Scaled jacobian of the cells of blocking (b)

Figure 19: Mesh quality comparison between a block structure generated without (a) and with block insertions (b).


(a)



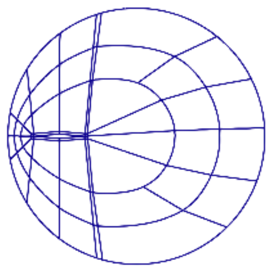
(b)

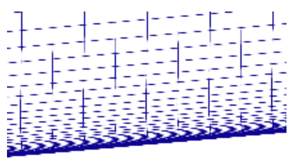
Figure 20: Mesh generated with an angle of attack $\alpha = 15^{\circ}$ (a) and the quality of the cells (b).

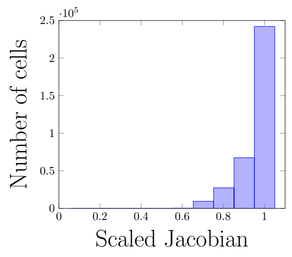
to $s_w = 4 \times 10^{-3}$m, and the default size of edges in the whole domain is $s_G = 1 \times 10^{-2}$m. We require at least 4 blocks in the boundary layer. In the boundary layer, 200 iterations of the modified line-sweeping smoother are performed with a damping parameter $\theta_d = 0.2$. This way, the mesh computed is orthogonal near the wall boundary (Fig. 21.b). The block structure in Figure 21.a shows two blocks inserted at each end of the geometry. Two additional blocks are inserted on the second layer, at the back of the airfoil. The algorithm provides good cells quality considering the scaled jacobian plotted in Figure 21.c.

4.2 Navier-Stokes equations

The Navier-Stokes equations are nonlinear partial differential equations used in fluid mechanics to describe the flow


(a) Blocking on the diamond airfoil


(b) Cells orthogonality near-wall boundary



(c) Scaled jacobian of the cells of the blocking (a)

Figure 21: Scaled Jacobian for the mesh generated on the diamond airfoil.

of a viscous and compressible fluid. The first equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0 \tag{10}$$

is the continuity equation. The momentum conservation equations are

$$\frac{\partial (\rho \mathbf{V})}{\partial t} + \nabla \cdot (\rho \mathbf{V}\mathbf{V}) = \nabla \cdot (\tau - pI) + \rho \mathbf{g}. \tag{11}$$

Then, the energy equation is given by

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho E \mathbf{V}) = \nabla \cdot ((\tau - pI) \cdot \mathbf{V}) + \rho \mathbf{g} \cdot \mathbf{V} - \nabla \cdot \mathbf{\Phi}. \tag{12}$$

In these equations, t is the time (s), ρ is the fluid density (kg.m^{-3}), $\mathbf{V}$ is the fluid particle velocity vector (m.s^{-1}), p is the pressure (Pa), τ is the viscous stress tensor, I is the unit tensor, $\mathbf{g}$ is the gravity vector (m.s^{-2}) and $\mathbf{\Phi}$ is the heat flux vector (J m^{-2} s^{-1}). In this work, we consider that the fluid is characterized by a perfect gas equation of state. The simulations are performed with the Reynolds Averaged Navier-Stokes (RANS) solver of SU2 [37]. Thus, the viscosity and turbulence are taken into account in the near-wall region. Here, the k-ω SST turbulence model is used [39].

4.3 Subsonic NACA 0012 airfoil

M_∞	AoA α	R_e	T_∞
0.3	15°	3×10^6	$293K$

Table 1: Simulation parameters for the subsonic NACA 0012 airfoil.

A simulation of the NACA 0012 airfoil is performed with the data set in Table 1 to validate the accuracy of the results on our generated mesh. The angle of attack is $\alpha = 15^\circ$, the Mach number is $M_\infty = 0.3$, the Reynolds number is set to $Re = 3 \times 10^6$ and the temperature $T_\infty = 293K$.

In Figure 22, the pressure coefficients

$$C_P = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty u_\infty^2} \qquad (13)$$

where p is the pressure and ρ the density, are compared. For a Reynolds number of $Re = 3 \times 10^6$, the experimental data set of Gregory and O'Reilly [40] seems to be the most appropriate for CFD validation [26]. These experimental data used as reference points for the surface pressure coefficients are plotted with black dots (•). The pressure coefficient plotted with red crosses (+) is the result of the simulation of this configuration on the mesh generated by our algorithm with the same parameters than the one of Figure 20. The two red curves are the result of the simulation plotted on both sides of the airfoil. However, the experimental data set gives only the result on one side of the airfoil. The results obtained with this configuration on our mesh are in good agreement with the experimental results.

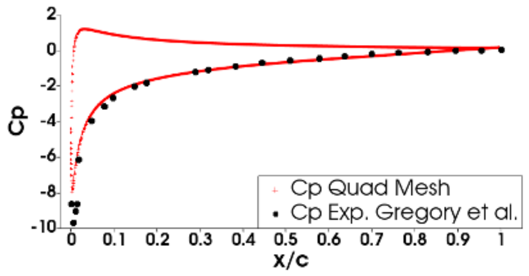


Figure 22: Pressure coefficient on the NACA 0012 airfoil for the simulation parameters in Table 1.

4.4 Supersonic diamond airfoil

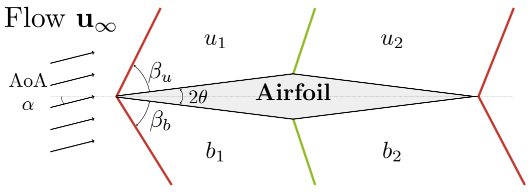


Figure 23: Scheme of the various zones and angles around the diamond airfoil [41].

In this part, a two-dimensional supersonic flow around a diamond-shaped airfoil is simulated. Figure 23 represents the geometry of the airfoil and the different areas and angles of the supersonic flow. Here, the viscosity effects are taken into account. As a consequence, the value of velocity on the wall is zero. For the case of the supersonic diamond airfoil, the analytical angles of the oblique shocks represented in red (🟥) are given by Liepmann et al. [36]. The shock direction depends on the Mach M_∞, the angle θ of the geometry, and the angle of attack α.

M_∞	AoA α	R_e	T_∞
1.5	3°	3×10^6	$293K$

Table 2: Simulation parameters for the supersonic diamond airfoil.

In this study, $\theta = 5^\circ$ and the chord of the airfoil is 1m. For the first simulation, the parameters in Table 2 are set. In Figure 24, the mach number distribution is plotted and compared to the analytical positions of the shocks. The value of the computed angle after the simulation is $\beta_u = 43.1^\circ$, and the value of the angle β_b is 45.7° which represents an error of 1° compared to analytical values. For this configuration, mach numbers are constant in the area u_1, b_1, u_2 and b_2. In the zone u_1, we reach a constant mach number of $M_{u_1} = 1.42$ in Figure 24, and $M_{b_1} = 1.19$ for the area b_1. These results are consistent with those given by the tables [35]. Regarding these results, the mesh generated by our algorithm captures the expected physics.

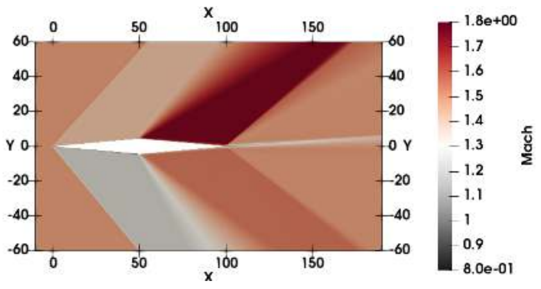


Figure 24: Mach-number distribution around a diamond-shaped airfoil immersed in a supersonic flow field at $M_\infty = 1.5$, $Re = 3 \times 10^6$ and $\alpha = 3^\circ$.

5. CONCLUSION

With this work, we propose a solution to automatically generate 2D quadrilateral block-structured meshes that are dedicated to flow simulation around a single vehicle. We take into account the geometrical shape of the domain and some simulation parameters like the angle of attack and the boundary layer thickness. First obtained results demonstrate that the generated meshes are usable for subsonic and supersonic flow simulations.

A few minor adjustments have to be made in the near future like improving the mesh size transition between the boundary layer and the other layers or integrating some mesh smoothing techniques. But the main part of the future work is twofold: first, we will extend the method to 3D. We are already able to generate the driving distance and vector fields in 3D and we will extend now the work of [24] to high-order blocks; second, we intend to use this approach in a loosely-coupled adaptive loop. We propose to iteratively adapt the distance and vector fields to encompass directional and size fields provided by a previous simulation code run.

6. References

[1] Chawner J.R., Dannenhoffer J., Taylor N.J. "Geometry, mesh generation, and the CFD 2030 vision." *46th AIAA Fluid Dynamics Conference*, p. 3485. 2016

[2] Ali Z., Tucker P.G., Shahpar S. "Optimal mesh topology generation for CFD." *Computer Methods in Applied Mechanics and Engineering*, vol. 317, 431–457, 2017

[3] Thornburg H. "Overview of the PETTT Workshop on Mesh Quality/Resolution, Practice, Current Research, and Future Directions." *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, p. 606. 2012

[4] Bommes D., Lévy B., Pietroni N., Puppo E., Silva C., Tarini M., Zorin D. "Quad-Mesh Generation and Processing: A Survey." *Computer Graphics Forum*, vol. 32, no. 6, 51–76, 2013. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12014

[5] Jezdimirovic J., Chemin A., Reberol M., Henrotte F., Remacle J. "Quad layouts with high valence singularities for flexible quad meshing." *CoRR*, vol. abs/2103.02939, 2021. URL https://arxiv.org/abs/2103.02939

[6] Reberol M., Georgiadis C., Remacle J. "Quasi-structured quadrilateral meshing in Gmsh - a robust pipeline for complex CAD models." *CoRR*, vol. abs/2103.04652, 2021. URL https://arxiv.org/abs/2103.04652

[7] Pietroni N., Nuvoli S., Alderighi T., Cignoni P., Tarini M. "Reliable Feature-Line Driven Quad-Remeshing." *ACM Trans. Graph.*, vol. 40, no. 4, jul 2021. URL https://doi.org/10.1145/3450626.3459941

[8] Alter S. "A Structured Grid Quality Measure for Simulated Hypersonic Flows." *42nd AIAA aerospace sciences meeting and exhibit*, p. 612. 2004

[9] Frey P.J., Alauzet F. "Anisotropic mesh adaptation for CFD computations." *Computer methods in applied mechanics and engineering*, vol. 194, no. 48-49, 5068–5082, 2005

[10] Secco N.R., Kenway G.K., He P., Mader C., Martins J.R. "Efficient mesh generation and deformation for aerodynamic shape optimization." *AIAA Journal*, vol. 59, no. 4, 1151–1168, 2021

[11] Chan W.M. "Overset grid technology development at NASA Ames Research Center." *Computers & Fluids*, vol. 38, no. 3, 496–503, 2009

[12] Knupp P.M., Ernst C., Thompson D.C., Stimpson C., Pebay P.P. "The verdict geometric quality library." Tech. rep., Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA, 2006

[13] Campen M. "Partitioning surfaces into quadrilateral patches: A survey." *Computer graphics forum*, vol. 36, pp. 567–588. Wiley Online Library, 2017

[14] Pietroni N., Campen M., Sheffer A., Cherchi G., Bommes D., Gao X., Scateni R., Ledoux F., Remacle J.F., Livesu M. "Hex-Mesh Generation and Processing: A Survey." *ACM Trans. Graph.*, jul 2022. URL https://doi.org/10.1145/3554920. Just Accepted

[15] Tarini M., Hormann K., Cignoni P., Montani C. "PolyCube-Maps." *ACM Trans. Graph.*, vol. 23, no. 3, 2004

[16] Gregson J., Sheffer A., Zhang E. "All-Hex Mesh Generation via Volumetric PolyCube Deformation." *Computer Graphics Forum*, vol. 30, no. 5, 1407–1416, 2011

[17] Livesu M., Vining N., Sheffer A., Gregson J., Scateni R. "PolyCut: Monotone Graph-cuts for PolyCube Base-complex Construction." *ACM Trans. Graph.*, vol. 32, no. 6, 171:1–171:12, 2013

[18] Hu K., Zhang Y.J. "Centroidal Voronoi tessellation based polycube construction for adaptive all-hexahedral mesh generation." *Computer Methods in Applied Mechanics and Engineering*, vol. 305, 405 – 421, 2016

[19] Huang J., Jiang T., Shi Z., Tong Y., Bao H., Desbrun M. "ℓ1-Based Construction of Polycube Maps from Complex Shapes." *ACM Trans. Graph.*, vol. 33, no. 3, 25:1–25:11, 2014

[20] Fang X., Xu W., Bao H., Huang J. "All-hex Meshing Using Closed-form Induced Polycube." *ACM Trans. Graph.*, vol. 35, no. 4, 124:1–124:9, 2016

[21] Fu X.M., Bai C.Y., Liu Y. "Efficient Volumetric PolyCube-Map Construction." *Computer Graphics Forum*, vol. 35, no. 7, 97–106, 2016

[22] Nieser M., Reitebuch U., Polthier K. "Cube-Cover–Parameterization of 3D Volumes." *Computer Graphics Forum*, vol. 30, no. 5, 1397–1406, 2011

[23] Lyon M., Bommes D., Kobbelt L. "HexEx: robust hexahedral mesh extraction." *ACM Trans. Graph.*, vol. 35, no. 4, 123, 2016

[24] Ruiz-Gironés E., Roca X., Sarrate J. "The receding front method applied to hexahedral mesh generation of exterior domains." *Engineering with computers*, vol. 28, no. 4, 391–408, 2012

[25] Blacker T.D., Stephenson M.B. "Paving: A new approach to automated quadrilateral mesh generation." *International journal for numerical methods in engineering*, vol. 32, no. 4, 811–847, 1991

[26] Rumsey C. "2DN00: 2D NACA 0012 Airfoil Validation Case." https://turbmodels.larc.nasa.gov/naca0012_val.html, 2021. [Online; accessed 5-August-2022]

[27] Mancinelli C., Livesu M., Puppo E. “A comparison of methods for gradient field estimation on simplicial meshes.” *Computers & Graphics*, vol. 80, 37–50, 2019

[28] Scalabrin L.C. *Numerical simulation of weakly ionized hypersonic flow over reentry capsules.* Ph.D. thesis, Citeseer, 2007

[29] Beatty K., Mukherjee N. “A Transfinite Meshing Approach for Body-In-White Analyses.” *Proceedings of the 19th International Meshing Roundtable.* 2010

[30] Gould J., Martineau D., Fairey R. “Automated Two-Dimensional Multiblock Meshing Using the Medial Object.” *Proceedings of the 20th International Meshing Roundtable.* Springer, 2011

[31] Mitchell S. “Incremental Interval Assignment by Integer Linear Algebra.” *proc. of the International Meshing Roundtable*, Oct. 2021. URL https://doi.org/10.5281/zenodo.5559025

[32] Staten M.L., Shepherd J.F., Shimada K. “Mesh matching–creating conforming interfaces between hexahedral meshes.” *Proceedings of the 17th International Meshing Roundtable*, pp. 467–484. Springer, 2008

[33] Roche C., Breil J., Olazabal M. “Mesh regularization of ablating hypersonic vehicles.” *8th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2022).* Oslo, Norway, Jun. 2022. URL https://hal-cea.archives-ouvertes.fr/cea-03783795

[34] Yao J. “A Mesh Relaxation Study and Other Topics.” Tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2013

[35] research staff A. “Raport 1135: Equations, tables, and charts for compressible flow.” Tech. rep., Ames Aeronautical Laboratory, 1953

[36] Liepmann H.W., Roshko A. *Elements of gasdynamics.* Courier Corporation, 2001

[37] Economon T.D., Palacios F., Copeland S.R., Lukaczyk T.W., Alonso J.J. “SU2: An open-source suite for multiphysics simulation and design.” *AIAA Journal*, vol. 54, no. 3, 828–846, 2016

[38] Ledoux F., Weill J.C., Bertrand Y. “GMDS: A generic mesh data structure.”

[39] Menter F.R. “Two-equation eddy-viscosity turbulence models for engineering applications.” *AIAA journal*, vol. 32, no. 8, 1598–1605, 1994

[40] Gregory N., O’reilly C. “Low-speed aerodynamic characteristics of NACA 0012 aerofoil section, including the effects of upper-surface roughness simulating hoar frost.” 1970

[41] Frapolli N., Chikatamarla S.S., Karlin I.V. “Entropic lattice Boltzmann model for gas dynamics: Theory, boundary conditions, and implementation.” *Physical Review E*, vol. 93, no. 6, 063302, 2016