Curved Hexahedral Block-Structure Generation by Advancing Front

Submission ID 2010

Abstract

This work aims to provide a method to generate block-structured meshes suitable for Computational Fluid Dynamics (CFD) simulations of flows around vehicles during atmospheric re-entry. This method takes as input a tetrahedral mesh of the domain, and the quadrangular block discretization of the vehicle surface. A linear blocking is obtained using an advancing front algorithm. This means it is incrementally created from the vehicle surface, layer by layer. Then, this linear blocking is curved, and we generate the final mesh. Some results of blocking and corresponding meshes generated with our algorithm are shown.

1 Introduction

Mesh generation is a critical component of numerical simulations. Here we consider the specific case of Computational Fluid Dynamics (CFD), and our aim is to provide a solution to automatically generate block-structured hexahedral meshes adapted to hypersonic flow simulations for atmospheric re-entry. To our knowledge, the generation of such meshes is usually handled by hand using dedicated software and is time-consuming.

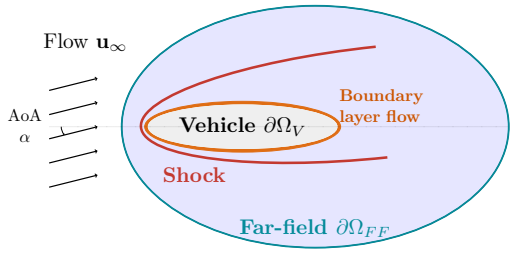


Figure 1: Traditional flow topology around a supersonic vehicle.

We restrict our approach to supersonic single-wall vehicles (Fig. 1) completely immersed in the fluid. We consider that the exterior boundary (far-field) is smooth.

2 Approach Overview

Our approach is based on the work of Ruiz-Gironés et al. [2, 3], where a method is presented to mesh the outer space around an object, using an advancing-front algorithm. A 2D pipeline based on this work and adapted to our constraints was proposed in [?][1]. In this paper, we extend it to generate a linear hexahedral block structure, as in [2], and we curve the obtained blocks. Starting from a tetrahedral discretization of the domain $\mathcal{M}_t$, and the quadrangular block discretization of the vehicle surface, the proposed algorithm builds a block structure by an **advancing front** process [3]. The initial front to advance corresponds to the quadrangular block discretization of the vehicle surface.

Definition 2.1. *A 3D* **front** *is a set of adjacent quadrangular faces*[2]*. Each edge of each quadrangular face of a front is shared by exactly two quadrangular faces on this front. In our pipeline, the quadrangular block discretization of the vehicle is the first* **front**, $\mathcal{F}_0$.

Definition 2.2. *A 3D* **layer** *is a set of adjacent hexahedral cells*[3]*. A* **layer** *is bounded by exactly two different* **fronts**. *For each quadrangular face of each hexahedral cell of a layer, either this face belongs to one of the two fronts, or this face is shared by exactly two hexahedra on this layer.*

3 Linear Block-Structure Generation

The block structure is generated by a successive extrusion of the layers, starting from the input discretization of the surface of the vehicle. The first step of the algorithm consists in computing some distance and vector fields on $\mathcal{M}_t$ to drive the extrusion algorithm.

3.1 Fields Computation As in [2], we compute three different distance fields. The first distance field d_v is the Euclidean distance of each node of $\mathcal{M}_t$ to the vehicle. The second distance field d_{FF} is the distance of each node of $\mathcal{M}_t$ to the outer boundary. The last distance field d is a combination of the two first distance fields. For each node of $\mathcal{M}_t$, we have:

$$d = \frac{d_V}{d_V + d_{FF}}. \tag{3.1}$$

[1] Anonymized. Will be add for the final version.

[2] Two faces are adjacent if they share at least one edge.

[3] Two hexahedra are adjacent if they share at least one quadrangular face.

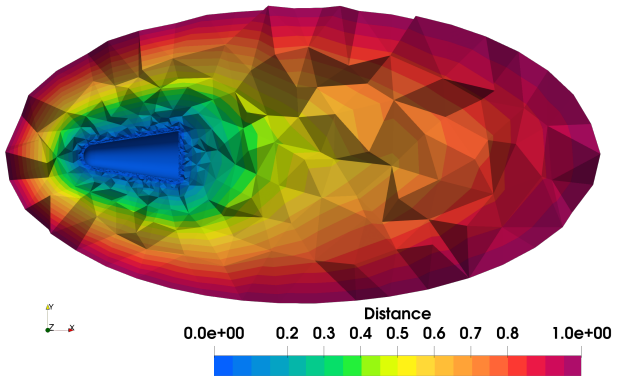


Figure 2: Distance field d computed on the tetrahedral mesh $\mathcal{M}_t$.

This field is normalized between $[0, 1]$. For every node on the vehicle, the distance is equal to 0, and for every node on the outer boundary, this distance field is equal to 1. The distance field d plotted on Figure 2 is used to compute the positions of the block corners of the next front (*i.e.* the new nodes created to generate the blocks of the layer). This ensures us a strong property: a **front** cannot separate. This means, by building a layer of hexahedra on a **front**, we ensure to provide a new **front** that respects the Definition 2.1. The gradient of the distance field d gives us a vector field on $\mathcal{M}_t$, that will lead the direction of the layers extrusion.

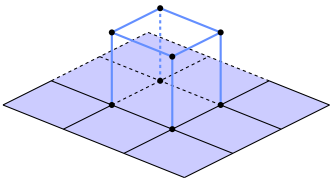
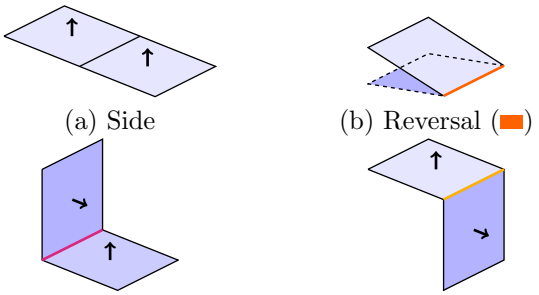


Figure 3: Pattern to apply on a regular quad. face of the front ( ) to create a new hex. block ( ).

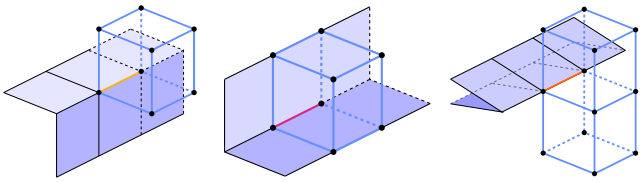
3.2 Layer Generation A **layer** $\mathcal{L}_i$ is generated from a **front** $\mathcal{F}_{i-1}$. Ideally, each quadrangular face of the **front** $\mathcal{F}_{i-1}$ generate one hexahedron (Fig. 3), and the set of new hexahedra forms the **layer** $\mathcal{L}_i$. However, due to geometrical features of the vehicle, we have to deal with some conflicts on the layers, leading to the expansion or contraction of blocks. To deal with those conflicts, we adopt the set of patterns introduced by Ruiz-Gironés et al. [2] to apply on specific configurations of edges of the front (Fig. 5, 6). The generation of a layer $\mathcal{L}_i$ is separated into three main steps:



(a) Side (b) Reversal ( ) (c) End ( ) (d) Corner ( )

Figure 4: Edges classification on a front ( ), according to the angle between the two outgoing normals of the adjacent faces of the edge.

1. **The geometric classification of the edges of the front** $\mathcal{F}_{i-1}$ to analyze and identify the features of the front $\mathcal{F}_{i-1}$ on which we want to build the layer. Each edge of the front is classified according to the value of the angle between the normal vectors of its two adjacent quadrangular faces (Fig. 4).
2. **The computation of paths of edges on the front** to identify on which nodes and edges we can apply patterns (Fig. 8).
3. **The application of the patterns** on nodes (Fig. 6), then on edges (Fig. 5), and finally build the hexahedra on regular faces of the front (Fig. 3).



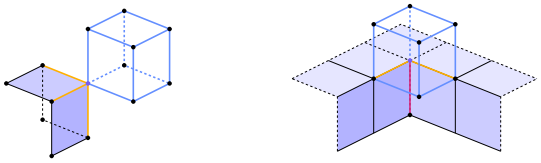
(a) Corner (b) End (c) Reversal

Figure 5: Patterns to apply on edges of the front ( ) according to their geometric classification. The created hexahedral blocks are plotted in blue ( ).

Definition 3.1. *A* **path** *is defined by a set of adjacent edges*[4] *on the same front. A valid* **path** *is bounded by two nodes of the front on which we can apply a* **node pattern**. *One same node can bound the path at each edge. All the edges of this path share the same classification, according to Figure 4.*

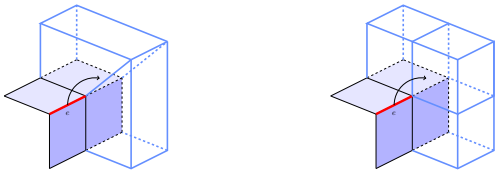
Step two is required to compute a set of edge paths considered valid to apply the patterns. As illustrated

[4]Two edges are adjacent if they share at least one node.



(a) 3 Corner (b) 2 Corner, 1 End

Figure 6: Examples of patterns to apply around one node of the front ( ), according to the classification of the adjacent edges around this node. The created hexahedral blocks are plotted in blue ( ).



(a) Without **corner** edge pattern (b) With **corner** edge pattern

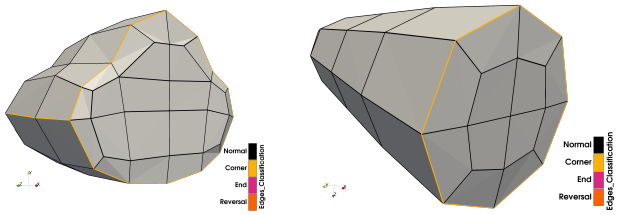
Figure 7: Topology propagation on a 3D layer. Depending if we apply a **corner edge pattern** (Fig. 5.a) on the adjacent **corner edge** of the front (a), or not (b), we will be forced to apply a certain pattern on the red edge too. Otherwise, the block structure will be invalid.

on Figure 7, we cannot decide which pattern to apply on which node or edge independently. We have to ensure first that the application of such a **node** or **edge pattern** will provide an acceptable neighborhood for the other blocks around. The computation of those paths depends on the available patterns (Fig. 5, 6). An example of valid paths selected after the geometric classification of the front edges is given on two different geometries in Figure 8. In this work, we also introduce the notion of **closed paths** (Fig. 8.b) to deal with the geometries of interest.

DEFINITION 3.2. *A* **closed path** *is a* **path** *that is not bounded by any particular node. All the nodes of this path are adjacent to exactly two edges of the front, and all the edges of a* **closed path** *share the same edge classification according to Figure 4. There is no* **node pattern** *to apply for those paths.*

4 Block Curving

As we work with a block structure, we have a few hexahedral block. The aim of curving the blocks is to improve the well-representation of the geometry before generating the final mesh to decrease the smoothing step. We decide to curve the block structure after the generation of the linear block structure, and we use



(a) Paths of corner edges (b) Closed path of corner edges on the RAM-C Spacecraft [4] geometry

Figure 8: Valid paths example.

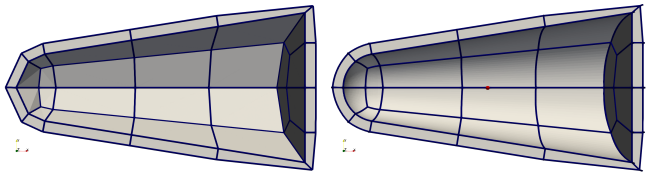
a Bézier representation of each hexahedral block [1] (Fig. 9). The degree of the blocks is uniform in each direction and for all the blocks.

(a) $p = 1$ (b) $p = 2$

Figure 9: Example of a Bézier Hexahedron of degree $p = 1$ (a) (*i.e.* linear block), and of degree $p = 2$ (b). For degree $p = 2$, there is one additional control point on each edge ( ), one on each face ( ), and one another hidden in the volume.

We compute the positions of the control points of the Bézier faces of the front $\mathcal{F}_0$ in order to interpolate some positions on the geometry. Then, in order to avoid potential intersections with the first front due to the very thin layer of blocks, we add the same offset on each control point of the opposite quadrangular face. As a result, we get a curved first layer of blocks (Fig. 10). The same operation is performed on the last front, corresponding to the surface blocking of the far-field.

Once all the blocks are curved, we generate a first mesh. For that, we use an **interval assignment** algorithm to compute the discretization of each curved block edge. According to the set of input parameters, each curved block edge computes its ideal final mesh



(a) Linear blocks (b) Curved blocks

Figure 10: Curving of the first layer of blocks.

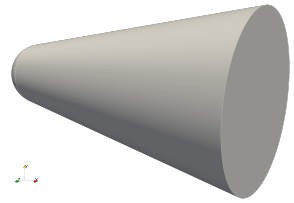
discretization. Then, as we expect to generate a conforming mesh, we solve an optimization problem to ensure that for all blocks, two opposite block edges will have the same discretization. Then, for each hexahedral block, based on the discretization in each direction, we use the parametric space of the Bézier hexahedron to compute the position of each final mesh node in the block. The final mesh generated is linear.

5 Results

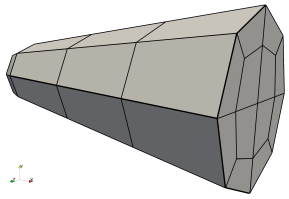
Our meshing algorithm is freely available and implemented in the open-source C++ meshing framework [5].

An example of the result of our algorithm is given in Figure 11, where we generate a mesh around the RAM-C stardust geometry (Fig. 11.a). The input quadrangular blocking surface is given in Figure 11.b. In Figure 11.c, we can see the second layer of blocks, built on the front represented before in Figure 8.d. On each **corner edge** (▬), an **edge pattern** is applied. In Figure 11.c, the hexahedra created by the **corner edge patterns** are plotted in orange (▬). The final linear blocking of Figure 11.d is generated with 5 layers of blocks. The final mesh of Figure 11.e is generated on the curved blocking (▬) and is composed of around $1,100,000$ hexahedral elements. With the implementation of our algorithm, it takes around $2m20$ to execute the full pipeline and generate both the blocking and the final mesh.

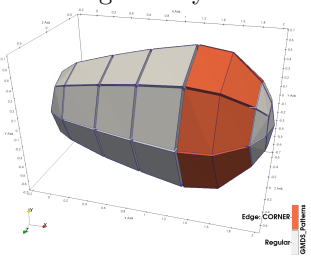
The use of some user parameters allows us to generate different blocking structures for the same geometry. For instance, we could choose to apply patterns on the very first layer of blocks or not. We can also decide not to apply any patterns on a part of the domain. Those parameters are taken into account during the computation of the valid **paths** on the **front**. This is particularly useful to control the blocking topology in the area of interest for the CFD (e.g. for the boundary layer computation).



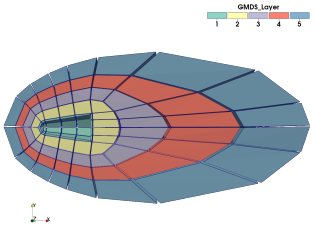
(a) RAM-C Stardust geometry



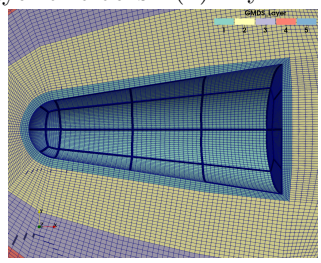
(b) Blocking of the surface



(c) Second layer of blocks



(d) Layers of linear blocks



(e) Final mesh and curved block edges (▬)

Figure 11: Blocking and mesh generated on RAM-C geometry.

References

[1] R. Feuillet, *Embedded and high-order meshes: two alternatives to linear body-fitted meshes*, PhD thesis, Université Paris-Saclay, 2019.

[2] E. R. Gironés, *Automatic hexahedral meshing algorithms: from structured to unstructured meshes*, PhD thesis, Universitat Politècnica de Catalunya (UPC), 2011.

[3] E. Ruiz-Gironés, X. Roca, and J. Sarrate, *The receding front method applied to hexahedral mesh generation of exterior domains*, Engineering with computers, 28 (2012), pp. 391–408.

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

[5] Anonymized. Will be add for the final version.