Hexahedral curved block-structured mesh generation for atmospheric re-entry

Claire Roche

université PARIS-SACLAY

Hexahedral Curved Block-Structured Mesh Generation for Atmospheric Re-Entry

Génération de Maillages Hexaédriques Structurés par Blocs Courbes pour la Rentrée Atmosphérique

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580, Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique Mathématique
Graduate School : Informatique et sciences du numérique, Référent : Université d'Évry Val d'Essonne

Thèse préparée dans l'unité de recherche **LiHPC** (Université Paris-Saclay, CEA), sous la direction de **Franck LEDOUX**, Directeur de Recherche, le co-encadrement de **Jérôme BREIL**, Ingénieur-Chercheur, et de **Thierry HOCQUELLET**, Ingénieur-Chercheur

Thèse soutenue à Paris-Saclay, le 1er octobre 2024, par

Claire ROCHE

Composition du jury

Membres du jury avec voix délibérative

Éric ANGEL Professeur des Universités, Université d'Évry Paris-Saclay	Président & Examinateur
Héloïse BEAUGENDRE Professeure des Universités, Bordeaux INP	Rapporteure & Examinatrice
Xevi ROCA Chef d'Équipe, Barcelona Supercomputing Center	Rapporteur & Examinateur
Jeanne PELLERIN Cadre Scientifique, TotalEnergies, entreprise	Examinatrice

THESE DE DOCTORAT
NNT : 2024UPASG053

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Génération de Maillages Hexaédriques Structurés par Blocs Courbes pour la Rentrée Atmosphérique

Mots clés: Maillage Hexaédrique, Structure de Blocs Courbes, Mécanique des Fluides Numérique, Hypersonique

Résumé: Le Commissariat à l'Énergie Atomique et aux Énergies Alternatives (CEA) s'intéresse à la simulation d'écoulements fluides en régime supersonique et hypersonique dans le cadre de la rentrée atmosphérique. Pour ce faire, un code de simulation numérique dédié y est développé. Pour répondre à des contraintes fortes, ce code ne prend en entrée que des maillages hexaédriques structurés par blocs. Ce type de maillage est compliqué à générer, c'est le plus souvent réalisé à la main via l'utilisation de logiciels interactifs dédiés. Pour des géométries industrielles complexes, la génération d'un maillage est très couteuse en temps. A l'heure actuelle, la génération automatique de maillages hexaédriques est un sujet de recherche ouvert et complexe.

Dans le cadre de ces travaux de thèse, nous proposons une méthode permettant de générer des maillages structurés par blocs courbes de domaines fluides autour de géométries dédiées pour les problématiques visées. Cette méthode a d'abord été prototypée dans le cadre de domaines 2D, puis étendue au cas 3D. Ici, la méthode est présentée dans le cas général, en dimension n. Elle se découpe en plusieurs étapes qui sont les suivantes. Dans un premier temps, une structure de blocs linéaire est obtenue par extrusion d'une première discrétisation de la paroi. Ces travaux sont une extension des travaux proposés par Ruiz-Girones et al. [RG11, RGRS12]. Une fois cette structure de blocs linéaire obtenue, nous proposons deux manières distinctes de courber les blocs afin d'améliorer la représentation de la géométrie, et de limiter le lissage sur le maillage final. La première est à travers d'un processus de lissage de maillage à topologie fixe à l'aide d'un problème d'optimisation, auquel un terme de pénalité est ajouté pour aligner certaines arêtes du maillage aux interfaces. Dans notre processus, nous appliquons cette méthode de lissage à la structure de blocs pour l'aligner sur la surface du véhicule. Cette méthode étant pour l'instant trop couteuse en temps de calcul dans le cas 3D, nous proposons une seconde manière de courber les blocs, à travers une représentation à l'aide de courbes polynomiales de Béziers. Nous appliquons cette fois des opérations géométriques et locales afin d'aligner les blocs à la géométrie. Enfin, en partant du principe que les blocs sont représentés à l'aide de courbes de Bézier, nous générons un maillage final sur ces blocs courbes sous différentes contraintes. Finalement, nous évaluons la qualité des maillages générés à travers des critères purement géométriques, en étudiant l'impact des différents paramètres de notre méthode sur le maillage final. Nous évaluons également les maillages générés par la simulation d'écoulements fluides sur ces maillages, avec la comparaison à des données expérimentales, analytiques, ainsi qu'à des calculs de référence.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title: Hexahedral Curved Block-Structured Mesh Generation for Atmospheric Re-Entry

Keywords: Hexahedral Mesh, Curved Block Structure, Computational Fluid Dynamics, Hypersonic

Abstract: The French Alternative Energies and Atomic Energy Commission (CEA) studies Computational Fluid Dynamics (CFD) in the case of supersonic and hypersonic flows. To do so, a dedicated code is developed. To deal with strong constraints, this code only performs on block-structured meshes. This specific type of mesh is complicated to generate, and this generation is usually handled by hand using dedicated interactive software. In the case of industrial complex geometries, the mesh generation is highly time-consuming. Currently, automatic hexahedral mesh generation is a complex open research field.

In this thesis work, we propose a method to generate block-structured hexahedral meshes on curved blocks of the fluid domain around vehicles, dedicated for the problems of interest. This method is first proposed in 2D and then extended to 3D. Here, it is presented in the generic nD case. This method is composed of several steps. First, a linear block structure is extruded from a first vehicle surface discretization. This work is an extension of the previous work of Ruiz-Girones et al. [RG11, RGRS12]. Once this linear block structure is generated, we propose two different methods to curve the block structure to improve the vehicle surface representation and limit the smoothing on the final mesh. The first method is through an algorithm of mesh adaptation at fixed topology by solving an optimization problem to which a penalty term is added to align certain mesh edges to an implicit interface. Our global approach uses this method to align the block structure to the vehicle surface. This method remains time-consuming in 3D, so we proposed a second method to curve our block structure through polynomial Bézier curves. Considering our blocks as Bézier blocks, we apply geometric and local operations to align the high-order block structure to the vehicle surface. Then, considering we curved the block structure using Bézier elements, we generate a mesh on the curved blocks under constraints. Finally, the generated mesh quality is evaluated in two ways. The first one is through purely geometrical criteria analysis. The second one is through numerical simulations of flows around vehicles on our meshes, comparing the simulation results to experimental data, analytical results, and reference simulations.

AKNOWLEDGEMENTS

I would like to thank the jury members Éric ANGEL, Xevi ROCA, Héloïse BEAUGENDRE, and Jeanne PELLERIN, for accepting to evaluate my PhD work.

Je souhaite remercier Franck du fond du cœur. Pour m'avoir fait confiance en premier lieu, puis pour m'avoir encadrée et accompagnée tout au long de la thèse. Merci pour tous les échanges stimulants. Je me suis énormément amusée pendant ces trois années, merci.

Évidemment, je remercie Simon d'avoir porté une multiple casquette, de collègue de thèse, à ami, puis presque encadrant. Merci d'avoir pris tout ce temps pour discuter et travailler avec moi. J'espère maintenant que je trouverai la force au plus profond de moi de te mettre des buts à l'escalade. Merci aussi à toi Flo.

Merci à Nico pour tout. Je resterai à jamais impressionnée par, entre autres, tes hautes compétences et connaissances en Git. J'ai adoré discuter avec toi tout au long de cette thèse, visiter Washinton DC, aller voir un match de NBA, parler lecture et photographie.

Plus généralement, merci aux personnes du laboratoire et service de Franck pour m'avoir accueillie parmi eux pour la deuxième partie de mon aventure de thèse.

Un grand merci à François d'être toujours présent et enthousiaste. C'est à chaque fois un plaisir de parler de sciences avec toi.

De nouveau merci à Jeanne, pour tout.

Je n'oublie pas les personnes qui étaient là quand mon long périple à commencer, et qui m'ont donné l'envie et le courage de prendre cette voie. Merci à Paul, Coco et Stéphane.

Merci à Maxime le californien, et à Julien pour vos précieux cours de CFD, même si je ne suis pas l'élève la plus appliquée.

Je souhaite remercier chaudement Thierry et les membres de son laboratoire, qui m'ont accueillie dans leur couloir pendant plus d'une année. J'ai beaucoup rigolé, et plus qu'un soutien moral, vous m'avez encouragée à ouvrir ma curiosité sur beaucoup de sujets. Un merci tout particulier à Valentin pour ses bruits d'animaux ainsi que ses précieux cours de C++.

Merci à tous les copains de Teratec. Merci à Valentin, d'avoir partagé un long bout de chemin et de nombreuses pauses café avec moi (et pour m'avoir fait découvrir, avec Nico, la meilleure série de livres de tout les temps). Merci à Paul, d'être incroyablement épuisant et bavard mais également un vrai rayon de soleil à Teratec. Merci pour toutes tes attentions et ta gentillesse. Merci à Clément d'avoir fait briller le

soleil du sud dans le ciel un peu gris du nord. Merci aux personnes du meilleur open-space, Julie, Alexiane, Axelle, Alexandre, Victor (j'espère que les feuilletés aux saucisses étaient à la hauteur de tes espérances). Merci à cette incroyable équipe de doctorants de Teratec, Sébastien, Manu, Mina, Gabriel, Luc, etc, et aussi aux futurs doctorants, Arzhela et Tristan. Je vous souhaite à tous le meilleur pour la suite, et plein de réussite.

I want to acknowledge Ketan for being such a great supervisor, and a fun person. Thank you to the MFEM team members for welcoming me to their team during the summer of 2023 at the Lawrence Livermore National Laboratory. I want to thank Christine Z., and Kyle for saving my first days in the U.S. I want to thank Jon, Mike, Anton, Francesco, Leonardo, Peter, Clément R, Kyle (again), Asya, Kenneth & Kim, Taoli, Jaryd, Alper, Sam, Gabriel, Akash, Shahbaj, Rajib, Abi, and all the other interns from the LLNL summer 2023. Thank you for all the crazy adventures, for the cornhole games, the tacos & karaoke nights, and your support in my quarter coins collection!

Of course, I want to express a special thanks to Yu & Anh. You made me have the best summer of my life exploring California, and I am glad I made friends for life. I can't wait to explore more of the world with you.

Merci à mes super copines, Léa, Mira et Margaux. Merci à mes super copains de grimpe (et plus), Brice, Coco, Thomas, et toute l'équipe. Merci à Thomas V.

Merci à ma petite Lilou, et à ma grosse Rainette.

J'aimerais remercier mes parents, mon frère, et toute ma famille pour leur soutien depuis la maternelle. Pour m'avoir toujours encouragée à faire ce que j'aime. Tout particulièrement, je souhaite remercier ma mamie Annie, pour son soutien toutes ces années. Tu es un symbole de force et d'amour.

Last but not least, I want to thank Yann. Mon amour, je te remercie d'avoir été à mes côtés tout du long, et de m'avoir soutenue et supportée de manière inconditionnelle.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TERMS AND ACRONYMS

Acronyms

AoA Angle of Attack. 16, 38, 59, 72, 75, 77, 167–169, 184

CAD Computer Aided Design. 36, 57, 63, 74, 112

CCF Sphere-Cone-Cylinder-Flare. 15, 16, 134, 164, 165

CEA French Alternative Energies and Atomic Energy Commission. 42, 43, 139, 147, 150, 153, 155

CFD Computational Fluid Dynamics. 12, 38, 40, 45, 58–61, 68, 71, 139, 147, 152–155, 160

CST Class/Shape Function Transformation. 162

DOFs Degrees of Freedom. 14, 113, 116, 117

HyTRV Hypersonic Transition Research Vehicle. 16, 162, 163

LLNL Lawrence Livermore National Laboratory. 111

LSDD Least Squares fit of Directional Derivatives. 76

NASA National Aeronautics and Space Administration. 37

NS Navier-Stokes. 58, 150

RANS Reynolds Averaged Navier-Stokes. 58, 147, 148

TFI Transfinite Interpolation. 49–51, 111, 126

TMOP Target-Matrix Optimization Paradigm. 14, 112–117, 119

Glossary

***p*-adaptivity** High-order mesh element degree adaptation. 112

***r*-adaptivity** Mesh adaptation at constant topology. 112, 113

RÉSUMÉ EN FRANÇAIS

1 Introduction

Un **maillage** peut être définit comme la représentation discrète d'un espace physique continu en un ensemble finit d'éléments simples.

En informatique, les maillages sont utilisés principalement dans deux domaines. Le premier est celui de la **représentation graphique** (par exemple pour des films d'animation, ou bien des jeux vidéos), où le maillage sert à représenter de manière discrète des personnages, des objets, etc.. Le deuxième est la **simulation numérique**, et c'est d'ailleurs dans ce cadre que s'inscrivent ces travaux de thèse.

La simulation numérique a connu un essort important ces dernières années. Il s'agit d'un domaine rassemblant des outils mathématiques afin de reproduire ou de prédire un comportement physique par des moyens informatiques. La simulation numérique est utilisée quotidiennement, par exemple pour des prédictions météorologiques, mais également des simulations de tremblement de terre, la conception d'automobiles, d'avions, etc.. Au cours de ces dernières années, la simulation numérique a pris une place importante car cela permet d'accéder à des quantités difficiles, voir impossibles à mesurer expérimentalement, cela permet également de modifier des paramètres facilement, parmi d'autres avantages.

Mécanique des Fluides pour la Rentrée Atmosphérique

La rentrée atmosphérique fait référence à l'entrée d'un véhicule (par exemple un débris spatial ou une navette spatiale) dans l'atmosphére d'une planète. Lors de cette rentrée dans l'atmosphère, l'objet est soumis à de très fortes conditions de température et de pression. La vitesse supersonique ou bien hypersonique (c'est-à-dire au dessus de la vitesse du son) de l'objet peut mener à l'ablation de sa surface, voir même à sa désintégration.

La Figure 1 illustre les phénomènes observés lors d'un écoulement fluide autour d'un véhicule ( ) en régime supersonique ou hypersonique. La direction du fluide à l'infini est décrite par le vecteur $\overrightarrow{u_\infty}$ ( ) et l'Angle d'Attaque (AA) α. Du fait des effets de viscosité, une très fine couche limite représentée en orange ( ) se développe autour de la surface de l'objet. C'est une zone caractérisée par un très fort gradient de vitesse et de pression dans la direction orthogonale à la paroi. En général, afin de calculer ce phénomène de manière précise, il est nécessaire d'avoir des mailles alignées avec le sens de l'écoulement, ainsi que des mailles très fines dans la couche limite. Une onde de choc ( ) se développe également. Une onde de choc est une discontinuité du fluide caractérisée par un changement brutal de pression, tem-

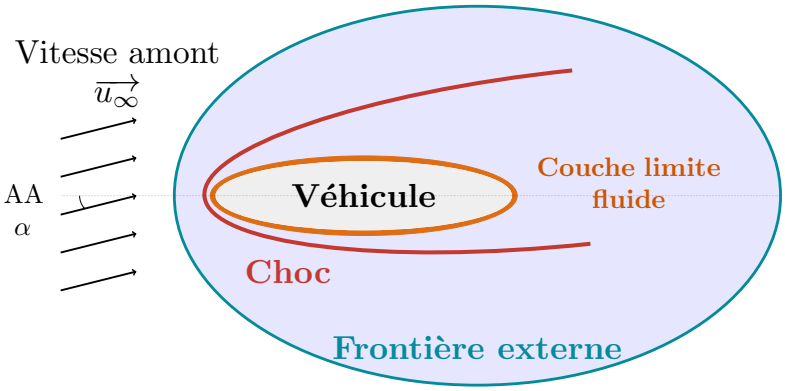


Figure 1: Topologie traditionelle d'un fluide autour d'un véhicule supersonique.

pérature, et de densité du milieu. Afin de calculer des résultats précis, il est nécessaire d'avoir des maillages très réguliers, surtout dans la partie amont du véhicule.

Afin d'étudier ce type de phénomènes, le Commissariat à l'Énergie Atomique et aux Énergies Alternatives (CEA) développe un code dédié pour le calcul d'écoulements fluides autour de véhicules en régime supersonique ou hypersonique. Afin de répondre à des contraintes fortes, ce code ne prend en entrée que des maillages structurés par blocs. Á l'heure actuelle, la génération de ce type de maillages en 3D est très compliqué. Le plus souvent, elle est réalisé à la main à l'aide de logiciels interactifs dédiés et peut nécessiter plusieurs semaines de travail pour des géométries industrielles complexes. Pour ces raisons, de nombreuses recherches portent sur l'automatisation de la génération de maillages hexaédriques.

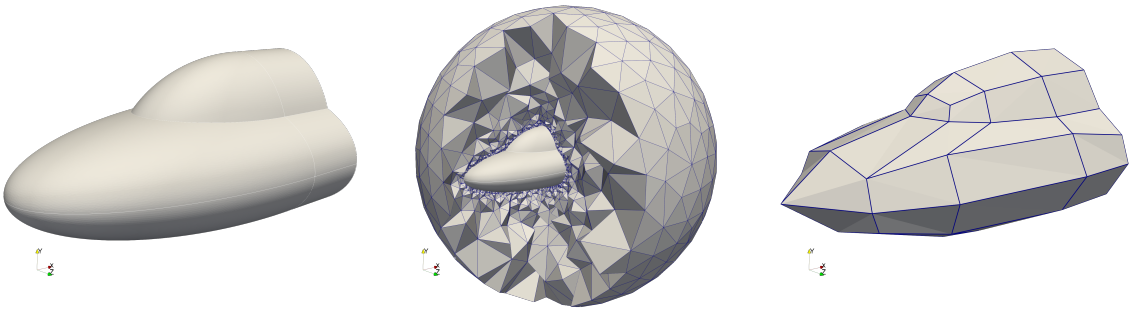
Le but de ces travaux de thèse est donc de proposer une méthode permettant de générer des maillages hexaédriques 3D structurés par blocs dédiés aux écoulements fluides autour de véhicules en régime supersonique. Pour ce faire, nous avons travaillé sous certaines hypothèses sur le domaine. Tout d'abord, les véhicules sont complétement immergés dans le fluide ( ), et le véhicule ne posséde qu'une seule paroi. De plus, la frontière extérieure ( ) est lisse (par exemple circulaire ou elliptique).

Principales Étapes de notre Approche

Dans le but de générer des maillages 3D hexaédriques structurés par blocs dédiés pour des applications de mécanique des fluides dans le cadre de la rentrée atmosphérique, nous avons prototypé un algorithme de maillage en 2D que nous avons étendu au cas 3D. Notre méthode de maillage est découpée en plusieurs étapes, et s'appuie sur les données d'entrée suivantes:

1. Un premier maillage du domaine fluide en un ensemble de simplex (i.e., triangles en 2D, et tétrahèdres en 3D) comme celui proposé en Figure 2.b;
2. Le découpage en blocs de la surface du véhicule similaire à celui de la Figure 2.c;
3. Un ensemble de paramètres pour contrôler les différentes étapes de l'algorithme.

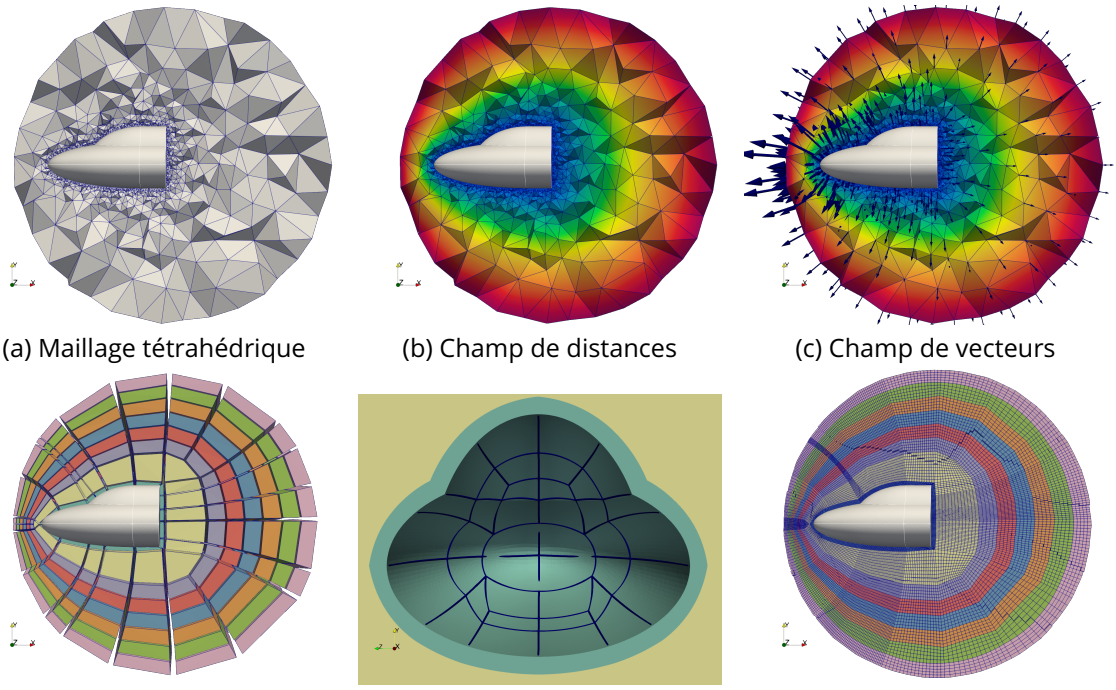
Les principales étapes de notre approche, illustrée en 3D sur la Figure 3, sont les suivantes. Sur la première représentation discrète du domaine fluide (voir Fig. 3.a), on calcule des champs de distance (voir Fig. 3.b), ainsi qu'un champ de vecteurs (voir Fig. 3.c). Ces champs vont



(a) Vehicule (b) Maillage tétrahédrique du fluide autour du véhicule (a) (c) Discrétisation en blocs de la surface du véhicule (a)

Figure 2: Données d'entrée de notre méthode.

nous permettre de guider l'algorithme de génération de blocs, qui seront extrudés couche par couche, à partir de la discrétisation initiale de la surface du véhicule. La Figure 3.d représente une vue suivant un plan de coupe d'une structure de bloc extrudée sur 8 couches. Chaque couleur représente une couche de blocs différente. Cette méthode est inspirée des travaux de Ruiz-Gironés et al. [RG11, RGRS12] et adaptée à nos problématiques.



(a) Maillage tétrahédrique (b) Champ de distances (c) Champ de vecteurs

(d) Blocs linéaires (e) Blocs courbes (f) Maillage hexaédrique

Figure 3: Principales étapes de notre approche illustrées dans le cas 3D.

Une fois la première structure de blocs est générée, on ajoute une étape afin de courber les blocs dans le but d'améliorer la représentation de la géométrie (voir Fig. 3.e). Cette étape permet d'éviter l'appel à un algorithme de lissage sur le maillage final, qui serait couteux en temps étant donné que le maillage finil peut comporter un nombre important d'éléments. Pour courber les blocs, dans le cadre de ces travaux de thèse, deux méthodes ont été étudiées.

- La première méthode s'inscrit dans la cadre d'un algorithme permettant de faire du lissage de maillage à topologie fixe par la résolution d'un problème d'optimisation global.

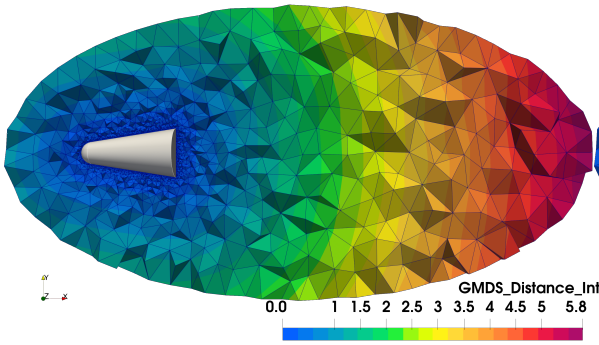
- La seconde approche consiste à considérer nos blocs comme étant des blocs de Bézier, auxquels nous appliquons un ensemble d'opérations géométriques et locales afin d'aligner les faces de blocs à la surface de la géométrie.

En considérant que nous avons courbé notre structure de blocs à l'aide d'éléments de Bézier, nous générons un maillage sur cette structure à l'aide d'un algorithme dédié (voir Fig. 3.f).

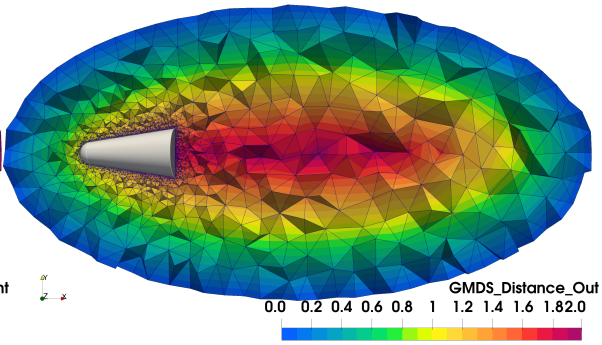
2 Génération de Structure de Blocs par Avancées de Fronts

Les champs de distances et vecteurs sont des composantes clés dans notre approche. L'idée, inspirée de Ruiz-Gironés et al. [RG11, RGRS12] est de mélanger différents champs afin de guider le processus d'extrusion par couches de blocs. Ces champs sont discrets, calculés sur un maillage de support composé d'éléments simples (triangles en 2D et tétraèdres en 3D) du domaine fluide à mailler.

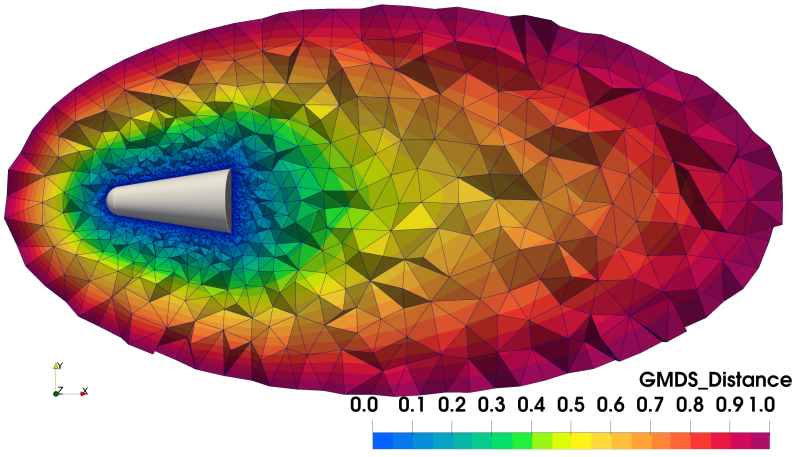
Calcul des Champs



(a) Distance Euclidienne à la surface du véhicule d_V



(b) Distance Euclidienne à la frontière externe d_{FF}



(c) Champs de distances combinées d

Figure 4: Champs de distances autour d'un véhicule 3D.

Tout d'abord, trois champs de distance sont calculés. Le premier, d_V, illustré en Figure 4.a représente la distance Euclidienne de chaque nœud du maillage support à la surface du véhicule. Le second champ, d_{FF} illustré en Figure 4.b est calculé comme étant la distance Euclidienne de chaque nœud du maillage support par rapport à la frontière externe. Le dernier champ de distance d, illustré en Figure 4.c, est une combinaison des deux premiers champs tel que:

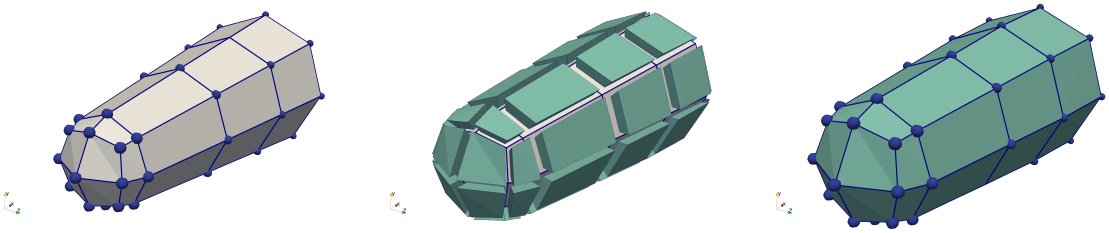
$$d = \frac{d_V}{d_V + d_{FF}}. \tag{1}$$

Ce dernier champs d vérifie $0 \leq d(x) \leq 1$, $\forall x \in \Omega$, et les conditions limites sont données par $d_{|\partial\Omega_V} = 0$ and $d_{|\partial\Omega_{FF}} = 1$.

En addition de ces champs de distances, un champ de vecteurs est calculé sur le maillage de support du domaine. Ce champs de vecteurs est calculé entre autre à l'aide des gradients des différents champs de distances, et il nous permet de guider la direction de l'extrusion de chaque bloque dans une couche. Ce champs nous permet donc d'assurer certaines propriétés, comme l'orthogonalité des arêtes de blocs à la paroi du véhicule.

Génération d'une Couche

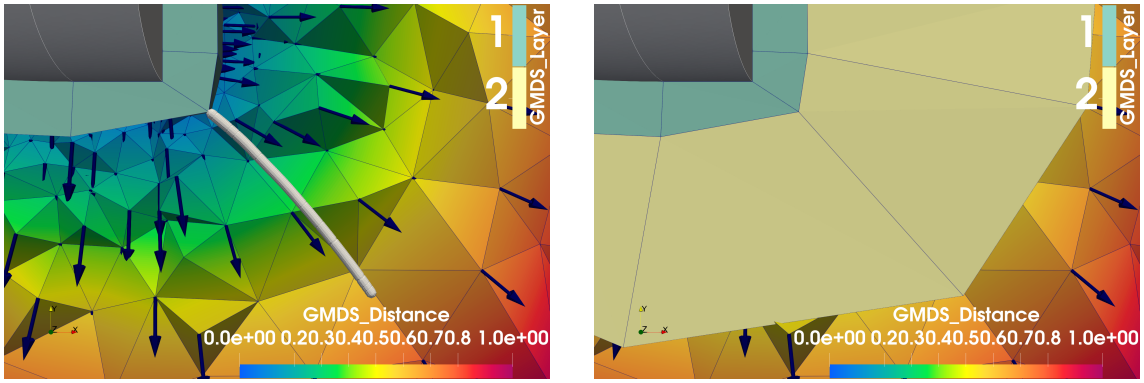
Dans le but de générer une structure de blocs autour d'un véhicule, nous adoptons une stratégie d'avancée de fronts basée sur les travaux de Ruiz-Gironés et al. [RG11, RGRS12], dans lesquels ils proposent cette approche pour générer un maillage hexahédrique autour d'un véhicule, à partir d'un découpage en quadrangles de la surface du véhicule, et d'une frontière extérieure non pré-maillée, et lisse. Ici, nous utilisons cette méthode modifiée pour générer la structure de blocs, qui contient moins d'éléments, ce qui est plus contraignant. Pour ce faire, nous adaptons certaines étapes de l'approche à nos besoins spécifiques.



(a) Front (b) Couche (c) Nouveau front

Figure 5: Génération d'une couche régulière de blocs en 3D. Chaque quadrangle du front (a) créé un unique bloc hexahédrique par extrusion de cette face. Cet ensemble de blocs ( ) représentés par une vue éclatée (b) forme une couche de blocs. Cette couche fournit un nouveau front (c).

A chaque étape de l'algorithme, une couche complète de blocs est générée sur un front (voir Fig. 5.a). La création de chaque couche est indépendante des précédentes. Un front est définit ici comme un ensemble de quadrangles, d'arêtes et de sommets de blocs qui respectent certaines propriétés. Par exemple, chaque arête de bloc du front est adjacente à strictement deux faces quadrangulaires sur ce front. Afin de créer une couche régulière, chaque face quadrangulaire du front est extrudée pour créer un bloc hexahédrique (voir Fig. 5.b). Cette couche fournit un nouveau front (voir Fig. 5.c) qui respecte les mêmes propriétés qu'énoncées précédemment. Ainsi, on peut construire une nouvelle couche sur ce front, et ainsi de suite, jusqu'à ce que la frontière extérieure soit atteinte.



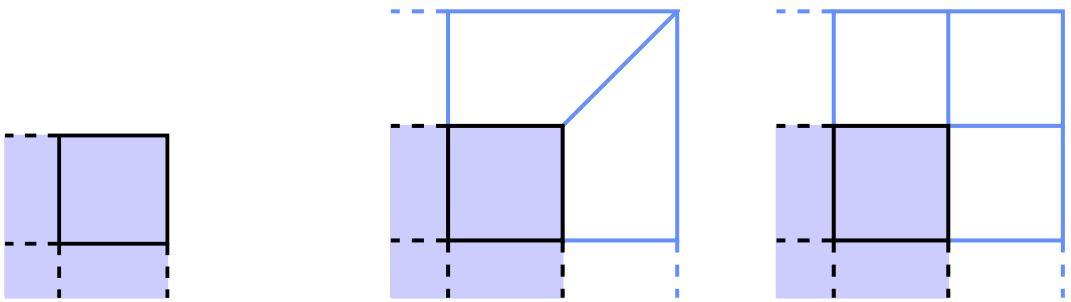
(a) Trajectoire d'un sommet de bloc (b) Nouveaux blocs ( )

Figure 6: Calcul de la position d'un sommet de bloc d'un front en 3D, à partir de la position du sommet de bloc du front précédent. En (a), la première couche de bloc ( ) est préalablement construite. Pour calculer la position du prochain sommet de bloc, la position du sommet est advectée suivant le champ de vecteur ( ), jusqu'à ce qu'une distance cible soit atteinte. Une fois les positions calculées, la nouvelle couche de blocs ( ) est créée.

Le calcul de la position des sommets d'une couche, à partir des positions des sommets de la couche précédente est illustré sur un cas réel présenté en Figure 6. Sur cet exemple pratique, le sommet de bloc est advecté suivant le champ de vecteurs représenté par les flèches bleues foncées ( ), jusqu'à ce qu'une valeur cible soit atteinte dans le champ des distances représenté en fond. Cette distance cible est la même pour tous les sommets de blocs d'un front. La trajectoire discrète effectuée par le sommet de bloc est représentée sur la Figure 6.a à l'aide des points blancs. Comme on peut le voir, cette trajectoire ne suit pas spécifiquement une droite. Enfin, sur la Figure 2.15.b, la couche de blocs construite à l'aide de cette position calculée est illustrée à l'aide d'un plan de coupe suivant le plan $(O, \overrightarrow{X}, \overrightarrow{Y})$.

Chaque sommet d'un front est placé à la même iso-valeur dans le champ des distances combinés. Ceci nous assure qu'un front ne peut pas se séparer, et qu'au cours du processus, chaque front généré respectera bien les propriétés d'un front.

Gestion de Conflits sur une Couche

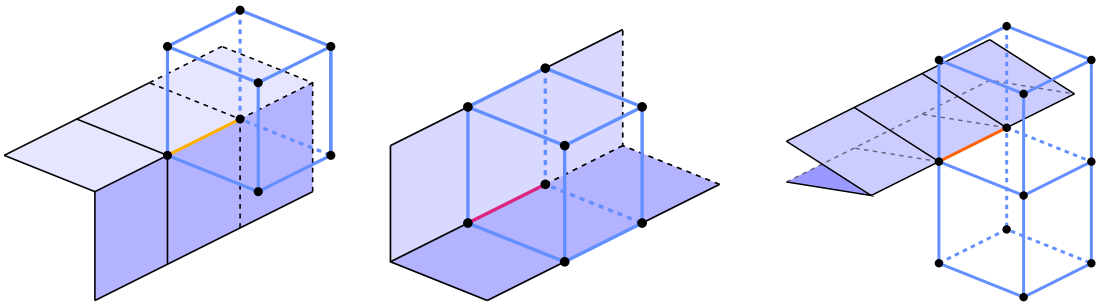


(a) (b) (c)

Figure 7: Exemple de gestion d'un conflit dans le cas d'une expansion du domaine physique en 2D [BS91]. Dans le cas régulier, chaque arête créé un bloc tracé avec les arêtes bleues ( ) sur la couche (b). Cependant, pour améliorer la qualité géométrique des blocs, un bloc suplémentaire est créé sur un sommet de blocs (c).

Sur une couche, il est possible d'avoir envie de gérer des conflits dûs à l'expansion ou la contraction du domaine physique. Le cas de la Figure 7 illustre une expansion du domaine physique en 2D. Dans ce cas, deux possibilités s'offrent à nous en suivant la méthode du Paving [BS91]. La première consiste à créer des blocs de manière régulière (voir Fig. 7.b), ou bien insérer un bloc sur un sommet de bloc (voir Fig. 7.c).

En 3D, le nombre de combinaisons possibles de blocs à construire autour d'un sommet de bloc ou d'une arête de bloc se démultiplient. Un échantillon de ces combinaisons est illustré en Figure 8. Pour référence, la combinaison présentée en Figure 8.a représente l'extension naturelle de la combinaison présentée en Figure 7.c en 2D. La différence principale est qu'en 2D, l'opération est locale au sommet, alors qu'en 3D, l'utilisation d'une telle combinaison de blocs sur une arête ou un sommet se propage dans la structure topologique de la couche. Ainsi, en plus d'utiliser une liste de critères locaux aux sommets d'un front pour appliquer une combinaison de blocs permettant d'améliorer la qualité des blocs, il est nécessaire de vérifier préalablement que l'utilisation de ces combinaisons va fournir une couche de blocs cohérentes, dans laquelle tous les blocs sont bien connectés.



(a) Insertion de bloc sur une arête ( ) (b) Contraction de blocs sur une arête ( ) (c) Insertion de deux blocs sur une arête ( )

Figure 8: Exemples de combinaisons de blocs ( ) appliquées sur un front ( ) pour gérer des conflits en 3D.

2D

Si la méthode globale de génération de la structure de blocs par avancée de fronts est la même en 2D et en 3D, la différence principale est lors de l'étape de traitement des conflits sur une couche, qui est plus direct en 2D. Pour ce faire, des règles similaires à celles présentées pour la méthode de Paving introduite par Blacker et al. [BS91] sont utilisées. Cette approche 2D présentée dans [RBHL23] constitue un prototype fonctionnel que nous avons ensuite étendu au cas 3D.

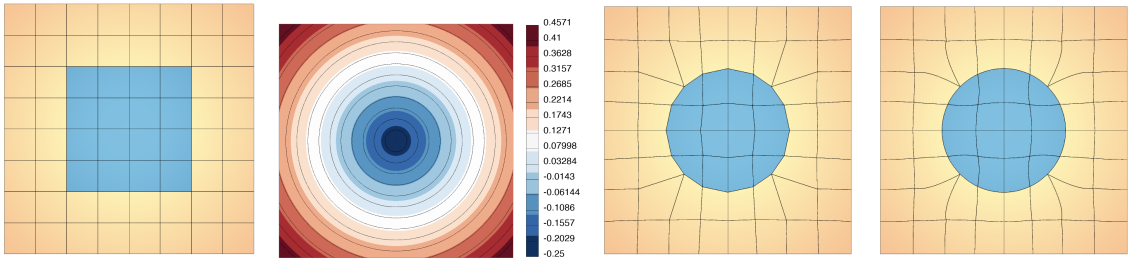
3 Courbure d'une Structure de Blocs & Génération de Maillages

Dans cette partie, nous nous intéressons à la représentation courbe d'une structure de blocs. Au lieu de considérer nos arêtes de blocs hexahédriques comme étant des segments droits, ces arêtes sont représentées ici à l'aide de polynômes. Cela permet entre autre de réduire l'erreur (c'est-à-dire l'écart) entre les blocs (ainsi que le maillage final) et la géométrie.

Dans le cadre de cette thèse, nous avons étudié deux approches afin de courber une structure de blocs linéaire. La première s'inscrit dans un processus d'adaptation de maillages développé dans MFEM[1] [mfe, AAB+21, BKMT23, AAB+24], bibliothèque notoire pour le calcul par méthode des éléments finis. La seconde approche étudiée s'appuie sur une représentation des blocs à l'aide de courbes de Bézier, et d'un ensemble d'opérations locales et géométriques afin d'aligner ces blocs à la géométrie.

Adaptation de Maillages d'Ordres Mixtes pour l'Alignement d'Interfaces

La première méthode s'inscrit dans une chaîne utilisant un algorithme d'adaptation de maillages développé dans MFEM [mfe, AAB+21, BKMT23, AAB+24]. Dans [DKK+19], une méthode est introduite pour adapter un maillage à topologie fixe (c'est-à-dire sans changer le nombre d'éléments du maillage ni la connectivité entre ces éléments) reposant sur la minimisation d'une fonction d'énergie construite à l'aide de métriques de qualité. En mettant à disposition différentes métriques [Knu20, Knu22], cela permet d'optimiser un maillage en se focalisant sur la taille des mailles, leurs formes, ou d'autre critères. Dans d'autres travaux [BKMT23], un terme de pénalité est ajouté à la fonction objective. Le but est de permettre l'alignement d'un maillage à une interface représentée à travers d'une fonction de distance discrète sur un maillage support.



(a) Maillage initial (b) Fonction distance σ (c) Maillage final d'ordre $p=1$ (d) Maillage final d'ordre $p=2$

Figure 9: Example d'alignement à une interface à l'aide de l'algorithme développé dans MFEM, dans le cadre de maillages d'ordres uniformes. Le maillage initial (a) est composé d'éléments quadrangulaires. Le domaine est séparé entre deux matériaux, représentés en bleu ( ) et en jaune ( ). La frontière entre ces deux matériaux est donnée par l'iso-valeur 0 de la fonction distance σ (b) sur un maillage de support.

La Figure 9 illustre l'alignement d'un maillage quandrangulaire initial (voir Fig. 9.a) représentant un domaine partagé entre deux matériaux. L'interface entre ces deux matériaux est représentée à travers l'iso-valeur 0 de la fonction distance σ en Figure 9.b. Ici, cette interface est circulaire. Cette méthode permet d'aligner un maillage dont les éléments ont tous le même ordre sur le domaine. Le maillage de la Figure 9.c est obtenu en utilisant l'algorithme pour optimiser et aligner le maillage en considérant que tous les éléments sont d'ordre 1 (c'est-à-dire linéaires). Dans le cas de la Figure 9.d, le maillage aligné est d'ordre 2 (c'est-à-dire que tous les éléments sont d'ordre 2). Dans ce second cas, on peut voir que l'interface circulaire entre les deux matériaux est mieux représentée. Cependant, ce qui n'est pas visible, c'est le

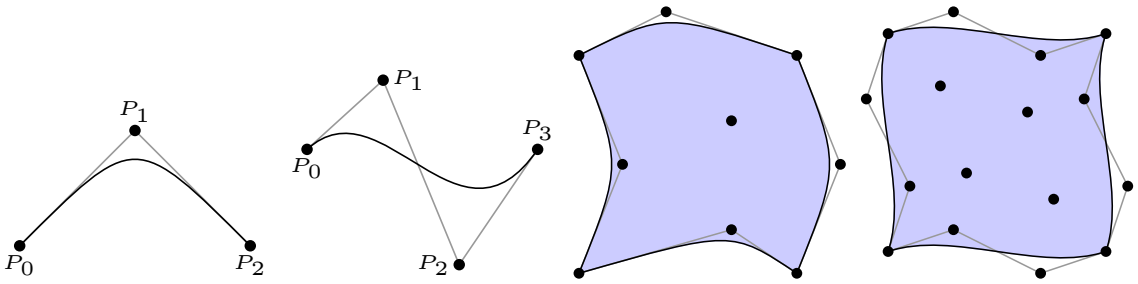
[1] https://github.com/mfem/mfem

nombre de degrés de liberté utilisés pour représenter le maillage. Dans le cas du maillage en Figure 9.c, chaque élément est représenté par 4 points. Dans le cas du maillage d'ordre 2 de la Figure 9.d, 9 points sont nécessaires pour représenter chaque quadrangle, ce qui augmente grandement le nombre de degrés de liberté sur le maillage global.

Dans ces travaux [MDK+24], une boucle itérative permet d'adapter l'ordre des éléments autour de l'interface que l'on souhaite représenter, afin d'utiliser des mailles d'ordre élevé dans les zones de forte courbure, tout en gardant des éléments d'ordre faible dans les autres parties du domaine. Ceci résulte en un maillage avec des éléments d'ordres mixtes dans le domaine, qui comporte moins de degrés de liberté qu'un maillage d'ordre élevé uniforme sur le domaine, mais une erreur d'interpolation entre le maillage et l'interface similaire.

Cette méthode a montré de bons résultats en 2D et 3D, sur des maillages avec des types d'éléments différents, et sur des problèmes d'intérêt pratique. Cependant, en 3D, cette méthode est pour l'instant trop couteuse en temps.

Courbure des Blocs par Polynômes de Bézier



(a) Courbe de Bézier d'ordre $p=2$ (b) Courbe de Bézier d'ordre $p=3$ (c) Quadrangle de Bézier d'ordre $p=2$ (d) Quadrangle de Bézier d'ordre $p=3$

Figure 10: Courbes de Bézier (▬) d'ordre $p=2$ (a), et d'ordre $p=3$ (b). Quadrangles de Bézier (▬) d'ordre $p=2$ (c), et d'ordre $p=3$ (d). Les P_i (●) sont les points de contrôle de chaque courbe.

Une seconde approche est étudiée dans ces travaux afin de courber les blocs, basée sur une représentation des blocs par des polynômes de Bézier. Dans les travaux de Feuillet [Feu19], différents éléments de Béziers ainsi que leurs propriétés sont explicités. Ici, nous utilisons la formulation des quadrangles et hexahédres de Bézier pour les blocs. Les courbes de Bézier [Bez86, DC59, GB12, Feu19] sont des polynômes bien étudiés, sur lesquelles multiples propriétés sont connues, et algorithmes disponibles. La Figure 10 présentes des exemples de courbes de Bézier de différents ordres (voir Fig. 10.a et b), ainsi que deux exemples de quadrangles de Béziers (voir Fig. 10.c et d). Une courbe de Bézier d'ordre p est définie par un ensemble de $p+1$ points P_i, appelés points de contrôle. Par extension, un bloc hexahédrique de Bézier d'ordre p est représenté à l'aide de $(p+1)^3$ points de contrôle. Ces positions ne sont pas interpolées, sauf pour les deux extrémités. Un ensemble d'opérations locales et géométriques sont effectuées afin d'aligner les faces de blocs correspondantes au véhicule à la surface géométrique de celui-ci. Ces modifications sont propagées pour assurer de ne pas générer des blocs courbes croisés.

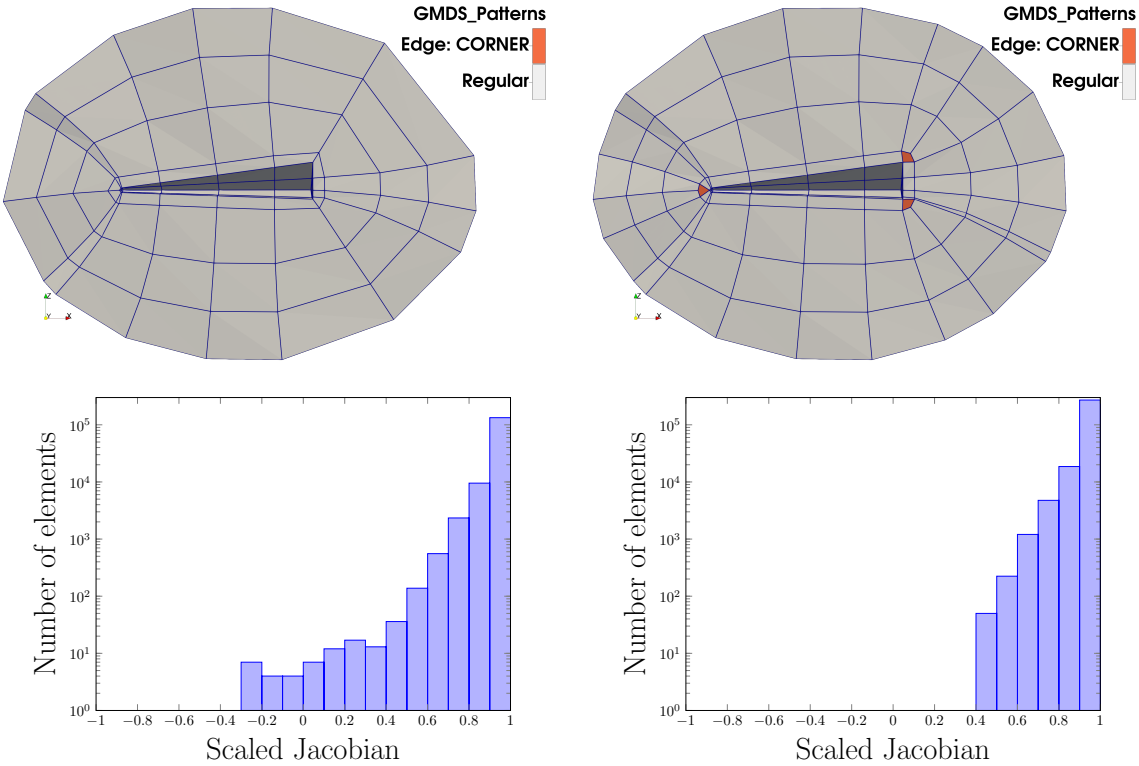
La génération du maillage final s'effectue à l'aide d'un algorithme d'optimisation sur la structure de blocs pour répondre aux contraintes imposées en entrée. A partir d'une arête

de blocs, on calcule l'ensemble des arêtes de blocs opposées dans la structure. Cet ensemble d'arêtes partage la même discrétisation finale. Pour chaque ensemble d'arêtes, un problème d'optimisations sous contraintes est résolu afin de déterminer le nombre final de mailles dans le maillage. Une fois les nombres de mailles fixés, le maillage final est généré en utilisant l'espace paramétrique de chaque bloc de Bézier.

4 Analyse de la Qualité des Maillages Générés

Dans ces travaux, la qualité des maillages hexaédriques structurés par blocs courbes générés est évaluée de deux manières différentes. La première est par le biais de critères purement géométriques sur ces maillages. La seconde est par la simulation numérique d'écoulements fluides sur les maillages générés.

La première étude, basée purement sur des critères géométriques bien connus tels que le *Scaled Jacobian* et la *Skewness* comme introduits dans la bibliothèque VERDICT [KET+06], permet de mettre en avant sur plusieurs géométries différentes à la fois les possibilités de notre méthode, avec un éventail de structures de bloc générés autour d'une même géométrie en modifiant les paramètres de la méthode. Cette étude permet également de mettre en avant la qualité des maillages générés sur ces différentes structures de bloc, ainsi que l'impact des paramètres de l'algorithme sur ce résultat final.



(a) Couches régulières (b) Couches avec gestion de conflits

Figure 11: Comparaison de la qualité géométrique de deux maillages générés sur deux structures de blocs différentes (a), et (b).

La Figure 11 présente une comparaison entre deux structures de blocs et maillages générés

avec cette méthode autour de la même géométrie. Les deux structures sont représentées ici en utilisant un plan de coupe suivant le plan $(O, \overrightarrow{X}, \overrightarrow{Z})$. Sur la Figure 11.a, la structure de blocs est extrudée de manière régulière, comme expliquée précedemment. Dans la seconde structure présentée en Figure 11.b, des blocs sont insérés sur une couche, représentés en orange (▬), pour améliorer la qualité géométrique des blocs. En conséquence, si on regarde les diagrammes de la qualité géométrique des maillages générés sur chaque structure, la qualité des mailles dans le second cas est améliorée.

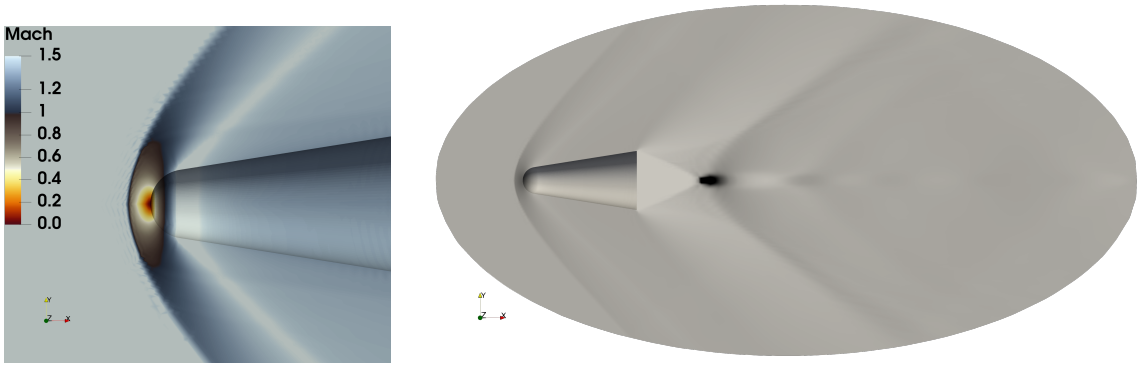
Dans une seconde partie, la qualité des maillages générés est étudiée à travers la simulation numérique. Pour ce faire, deux codes de calcul différents sont utilisés. Le premier, SU2 [SU2, EPC+16], est un code multi-physiques développé en C++ dont les sources sont ouvertes. Le second est un code Navier-Stokes stationnaire et instationnaire développé par le CEA, dédié pour des calculs de mécanique des fluides en régime hypersonique autour de corps de rentrée. Pour valider les maillages générés, des calculs ont été effectués en 2D ainsi qu'en 3D, en régime subsonique, supersonique, et hypersonique. Les résultats obtenus ont été comparés à

- des données expérimentales mises à disposition par la NASA;
- des solutions analytiques;
- des résultats de référence obtenus sur d'autres maillages générés à la main.

Pour les différentes configurations étudiées, les résultats obtenus sur nos maillages sont cohérents avec les données de référence. Ceci nous permet de valider l'utilisation de nos maillages pour des cas d'intérêt pratique.

M_∞	AA α	p_∞	T_∞
$1,5$	$0°$	$101.325,0\text{N.m}^{-2}$	$288,15\text{K}$

Table 1: Paramètres de simulation pour le fluide autour du RAM-C II 3D.



(a) Champ de Mach autour du nez de l'objet

(b) Champ de pression adimensionné

Figure 12: Résultats de la simulation autour du RAM-C II 3D.

Parmi les simulations effectuées, nous présentons ici un cas particulier d'un écoulement supersonique 3D effectué avec le code SU2 [SU2, EPC+16]. Les conditions infinies amonts du fluides sont celles explicitées en Table 1. Sur la Figure 12, les résultats de la simulation sont présentés à l'aide d'un plan du coupe suivant le plan $(O, \overrightarrow{X}, \overrightarrow{Y})$. Plus précisemment, sur la

Figure 12.a, le champ de Mach est tracé autour du nez de l'objet. Sur la Figure 12.b, le champ de pression adimensionné est représenté. D'après [And89], ces résultats correspondent à ce à quoi nous pouvions nous attendre.

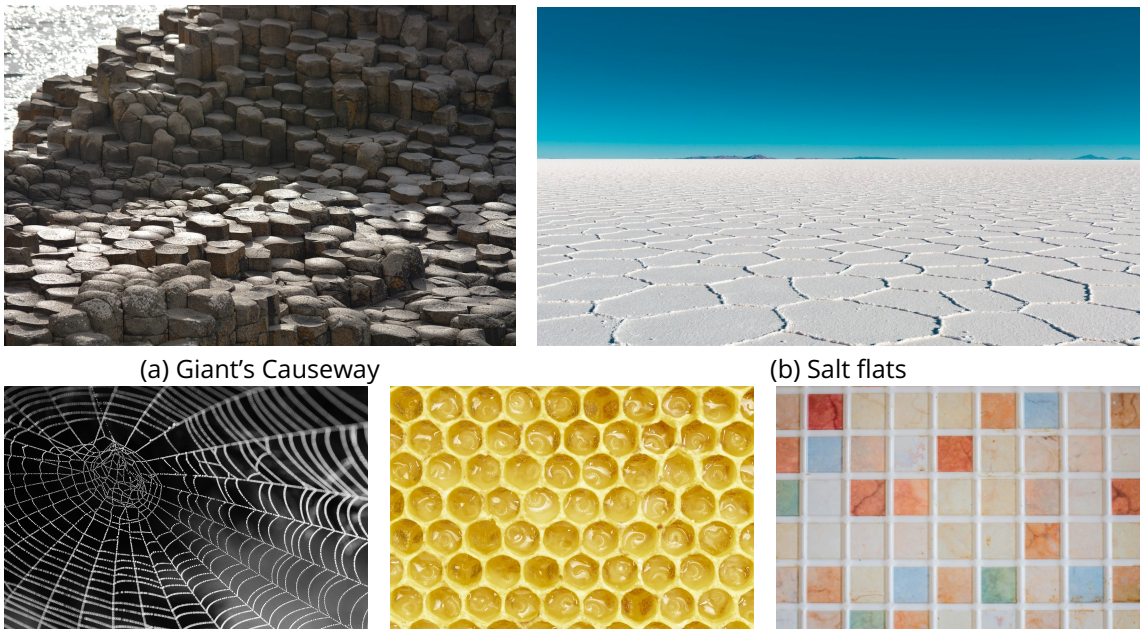
Ces résultats couplés aux autres permettent de montrer tout d'abord que nos maillages passent le test de validité de différents solveurs, c'est-à-dire qu'ils sont capable de lire nos maillages en entrées et qu'ils sont capables de les utiliser pour calculer. Ensuite, la comparaison aux diverses données montre que les solutions calculées sur ces maillages sont cohérentes.

5 Conclusion & Perspectives

Dans ces travaux, nous avons proposés une méthode automatique pour générer des maillages hexahédriques structurés par blocs courbes par une méthode d'avancée de fronts autour d'un véhicule avec une paroi unique, complétement immergés dans le fluide, dédiés pour des calculs de mécaniques des fluides autour de corps de rentrée. Plus spécifiquement, la méthode a été développé en 2D puis étendue au cas 3D. La structure de blocs est générée par extrusion par couches d'un premier maillage de blocs du véhicule jusqu'à la frontière extérieure. Cet algorithme inspiré des travaux de Ruiz-Girones et al. [RG11, RGRS12] est adapté ici à nos applications. Une fois cette structure de blocs hexahédriques linéaires générée, elle est courbée. Pour ce faire, deux approches sont étudiées. La première est dans le cadre d'une boucle d'adaptation du degré polynomial des éléments autour de la surface de la géométrie, en utilisant un algorithme d'optimisation de maillages développé dans MFEM [mfe, AAB+21, AAB+24]. Cette méthode a montré de bons résultats mais elle reste trop couteuse en temps en 3D. Ansi, une seconde méthode a été étudiée dans le cadre de ces travaux, en considérant les blocs comme des blocs de Bézier, que l'on aligne à la surface de la géométrie. La dernière partie de ces travaux est dédiée à l'étude et la validation des maillages générés. Pour ce faire, une première analyse purement géométrique est menée sur la qualité géométrique des éléments des maillages, en fonction des différentes topologies de blocs générées. Le processus de validation est complété par la simulation numérique, avec l'appuie de calculs effectués avec deux codes différents. Le premier code, SU2 [SU2, EPC+16] est un code développé en C++ spécialisé dans la résolution de systèmes d'équations aux dérivées partielles, notamment pour des applications de mécanique des fluides, le second est un code développé par le CEA. Différents types d'écoulements 2D et 3D ont été étudié, en régime subsonique, supersonique, et hypersonique. Les résultats obtenus ont été comparés à des données expérimentales de la littérature, des solutions analytiques, ainsi qu'à des résultats obtenus sur des maillages de référence. Nous avons ainsi montré que les maillages générés par notre méthode permettent d'obtenir des résultats de simulation cohérents avec la physique d'intérêt.

INTRODUCTION

A **mesh** can be defined as the discrete representation of a physical domain in a set of finite and simple elements. Figure 13 illustrates mesh examples in nature. Here, floors of the Giant's Causeway in Ireland (see Fig. 13.a), and the salt flats of the Death Valley (see Fig. 13.b) are marked by a set of polygonal shapes. In the same way, spider nets (see Fig. 13.c) form quadrangles, or honeycomb (see Fig. 13.d) form assemblies of regular hexagonal shapes. The last example of the mosaic (see Fig. 13.e) represents a planar surface paved using tiles. Here, it looks very regular but you may imagine various representations of diverse shapes (e.g., animals, trees, persons, etc.) using little tiles.



(a) Giant's Causeway (b) Salt flats

(c) Spider net (d) Honeycomb (e) Mosaic

Figure 13: Meshes in nature.

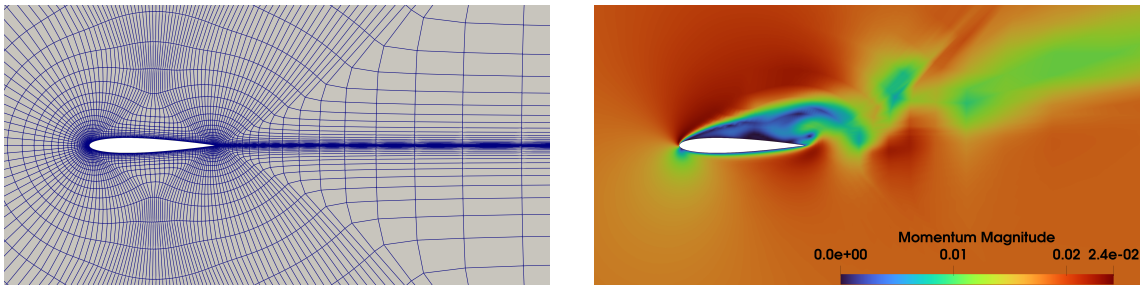
In computer science, meshes are mainly used in two different contexts to represent a continuous domain through a discrete finite set of points. The first one is **graphic representations**, such as in animation or video games (to represents characters, etc.). The second one is **numerical simulation**, which is the point of interest of this manuscript.

1 Numerical Simulation in a Nutshell

Numerical simulation is a field that consists in gathering many mathematical tools to reproduce or predict, using a computer, a physical phenomenon. This is used in various contexts of everyday life, for instance numerical weather forecast to predict the weather, earthquake simulations, car crash simulations, airplanes design, etc.. During the last decade, numerical simulation has taken much more importance contributing to the emergence of the digital industry. With numerical simulation, it is easier to modify some parameters and access quantities that are difficult or impossible to measure. It can also be used in a context where "real" experiments are not possible (e.g., tsunami effect prediction, star-meteorite collision, etc.). The process of numerical simulation can be divided into several steps.

The first step consists in identifying the physical phenomena to study. Then, to pick the **mathematical modeling**, namely the continuous equations that represent the phenomena, which we want to solve. For a same phenomenon, multiple different mathematical modelings can exist and choosing one is already an important step (e.g., in case of a flow simulation, do we want to take viscosity effects of the flow into account or not). Usually, those mathematical models are complex partial differential equation systems for which an exact solution is unknown[2]. However, many different **numerical methods** are used to approximate the solutions of those equations. As we expect to approximate the solution using a computer, we can not solve the continuous equations directly: we have to discretize continuous equations both in space and time. Such discretizations are handled by the **numerical schemes**. The best known are the Finite Differences, the Finite Elements, and the Finite Volumes, but many other methods exist. Those schemes give the formula that approximates the solution of the equations in each cell or points of the space discretization (i.e., a mesh).

As space, time is continuous, and the equations can not be solved on an infinite time step. A time step has to be set to compute the solution on a finite number of time periods. Let us note that this time step is not necessarily constant and can evolve during the simulation.



(a) Mesh around airfoil (b) Simulation result

Figure 14: Flow simulation around an airfoil [SU2].

Another critical step for numerical simulation is to define the **geometry** to use for the simulation (e.g., the shape of an airfoil as in Fig. 14.a, the shape of the bridge in case of a simulation of a flow under a bridge, etc.). The geometric modeling step is done using Computer Aided Design (CAD) software. Then, the **meshing** step sets the physical points on which the discrete equations are solved. The number of points, the relations between the points, and

[2]For some well-known systems, the research of an analytical solution remains an open problem.

their positions are set during this step and form what we refer to as a **mesh** (see Fig. 14.a). Figure 14.b represents the resulting field computed at each point of the mesh 14.a. Depending on the physical phenomenon (e.g., electromagnetism, mechanic, fluid mechanic, etc.), and the numerical scheme used, the type and quality of mesh needed differ. Meshing is a core and critical component of the numerical analysis process. In industrial cases, the mesh generation is often handled by placing the mesh points and connecting them by hand, using dedicated interactive software. This process is highly time-consuming, and this is the reason why some researches focus on the automation of mesh generation. If the mesh does not respect criteria depending on the physical phenomena, the numerical method, and the physical domain, then the simulation may fail.

This work focuses on automatic mesh generation and is dedicated to atmospheric re-entry flow simulations around supersonic and hypersonic re-entry bodies.

2 Computational Fluid Dynamics for Atmospheric Re-Entry

Figure 15: Artistic representation of a capsule re-entry [esa].

Atmospheric entry refers to the entry of an object or vehicle (e.g., space debris, spacecraft, etc.) from outer space into a planet's atmosphere. Figure 15 represents an artistic view of a capsule re-entry. During atmospheric entry, the object suffers from high temperature and pressure conditions. The supersonic or hypersonic velocity (i.e., superior to the sound speed) of the object can lead to its ablation or disintegration.

Vehicles have been sent into space for years to provide information about our environment. They provide, for instance, pictures and collect samples to study. The capsule must be slowed before landing so inhabited space vehicles can return to Earth. In other cases, it is not necessary. For example, Stardust was a robotic spacecraft launched by National Aeronautics and Space Administration (NASA) in 1999 to collect dust samples from a comet. It was the first spacecraft to bring samples from a comet to Earth. To bring back the spacecraft, it was necessary to

design it to withstand Earth's atmospheric re-entry. The rounded vehicle right part in Figure 15 represents the heat shield, which is supposed to ensure the capsule's integrity.

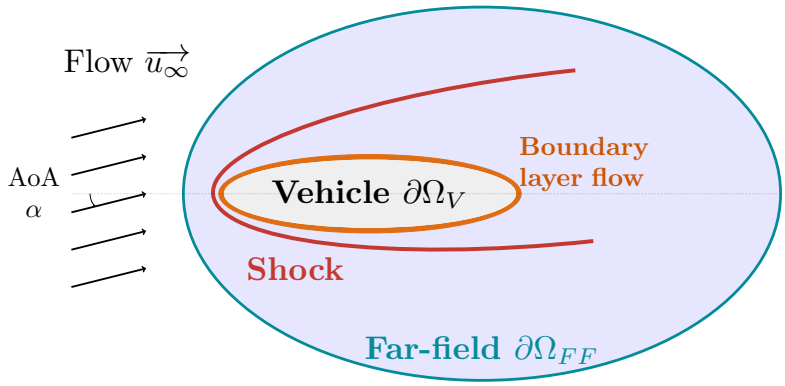


Figure 16: Traditional flow topology around a supersonic vehicle.

Figure 16 shows briefly the traditional flow topology observed during a supersonic or hypersonic flow simulation around a vehicle ( ). The direction of the inflow is represented by the black vectors $\overrightarrow{u_\infty}$ ( ) and the Angle of Attack (AoA) α. Due to the effect of viscosity, a very thin boundary layer plotted in orange ( ) develops along the wall. Extreme gradients of velocity and temperature characterize this region. In general, for Computational Fluid Dynamics (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. A shock wave is a flow discontinuity characterized by an abrupt change in pressure, temperature, and density of the medium. To compute an accurate solution of the fluid dynamics equations, very thin and regular cells are needed along the wall in the wall-normal direction. Thus, structured meshes are well-suited for this area. As few as possible, singular nodes (not valence four nodes) are admitted in the orange part of the mesh. However, mesh refinement is less restrictive near the shock to compute it accurately, unlike boundary layers.

In this work, vehicles 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 strong 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. 16) is the key part that will govern the simulation. In this very specific zone, the mesh has to be as regular as possible, and singular nodes are not admitted.

3 Principal Steps of our Approach

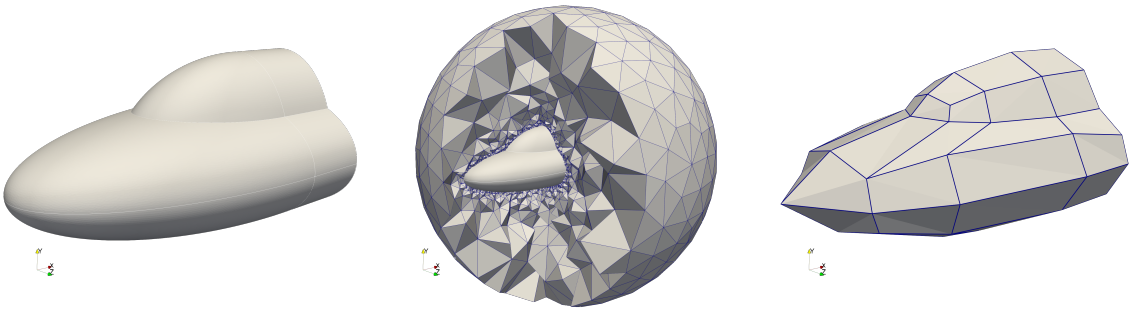
We aim to generate 3D hexahedral block-structured meshes dedicated for atmospheric re-entry CFD simulations. Such meshes are created before any simulation (i.e., we don't know the potential shock position, etc.). To generate those meshes, we propose a meshing algorithm that relies on three inputs:

1. A tetrahedral (or triangular in 2D) discretization of the fluid domain to mesh around the

vehicle (see Fig. 17.b).

2. The vehicle surface block discretization (see Fig. 17.c).

3. A set of user parameters to control some algorithm features.



(a) Vehicle surface

(b) Tetrahedral mesh of the fluid domain to mesh around vehicle (a)

(c) First block discretization of vehicle surface (a)

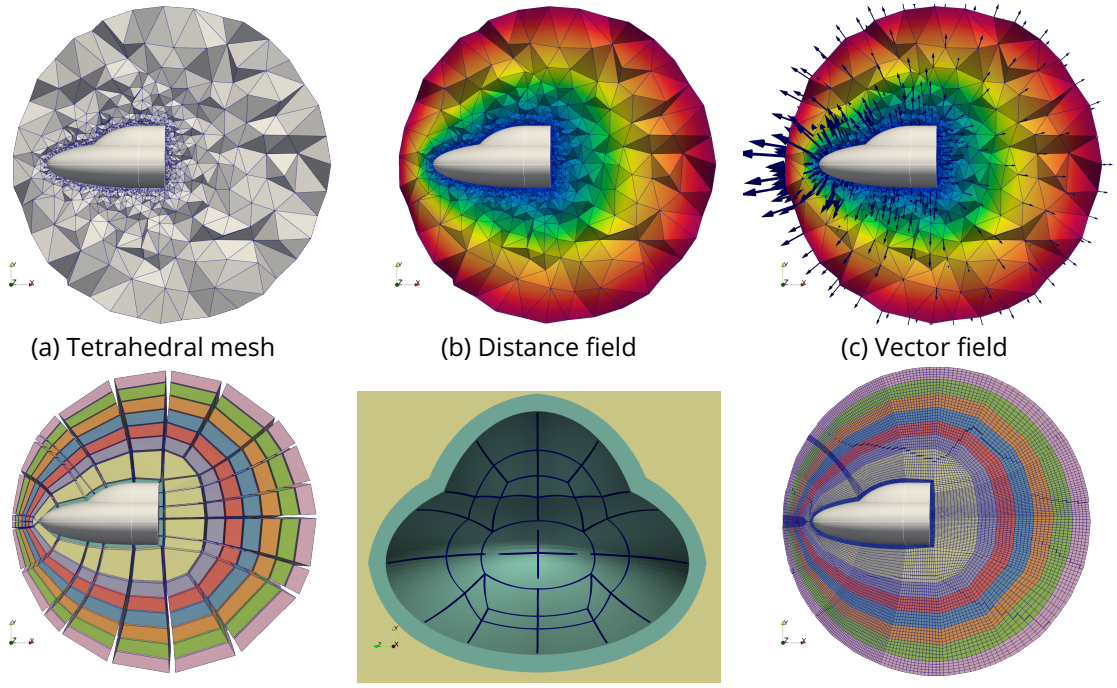
Figure 17: Inputs of our method.

Starting from the first tetrahedral mesh of the physical domain as the one of Figure 18.a (this type of meshes can easily be generated using the free open software GMSH [gms, GR09] [GR09] for instance), and the block wall surface discretization. We first generate a linear block structure around the vehicle using an extrusion algorithm that is an extension of the work done by Ruiz-Gironés et al. [RG11, RGRS12]. To do so, we compute some distance fields (see Fig. 18.b) and a vector field (see Fig. 18.c) that will lead the block extrusion direction. Then, starting from a first block discretization of the surface, we extrude the block structure layer by layer (see Fig. 18.d, where the block structure is extruded on 8 different layers, each one represented by a different color).

Once the linear block structure is generated, we add a step to represent our blocks as high-order blocks (see Fig. 18.e). This step aims to improve the geometry representation because as we work with a coarse block structure, there is an important gap between the real curved surface of the geometry and the straight cells of the blocks. This step also allows us to avoid the smoothing step on the final mesh. The final mesh can be made up of a large amount of cells. Suppose this final mesh is generated on the linear cells. In that case, we will need to apply a projection step to align the cells on the boundaries of the geometry and then apply a smoothing step to untangle and improve the quality of the cells around the vehicle. Applying a smoothing algorithm on such a mesh might be very time-consuming. This is the reason why we use high-order blocks. In this manuscript, two different strategies are studied to curve the blocks:

1. The first one uses Bézier cells and applies a set of geometric and local operations around the boundaries.

2. The other method considers the problem as a global problem. It solves an optimization problem to align a high-order block structure around the boundaries while carefully controlling the polynomial degrees of the elements around the geometry.



(a) Tetrahedral mesh (b) Distance field (c) Vector field

(d) Linear blocks (e) Curved blocks (f) Final hexahedral mesh

Figure 18: Main stages of our approach illustrated in 3D.

We decided to base our mesh generation pipeline on the work of Ruiz-Gironés et al. and we performed various modifications to fit our requirements.

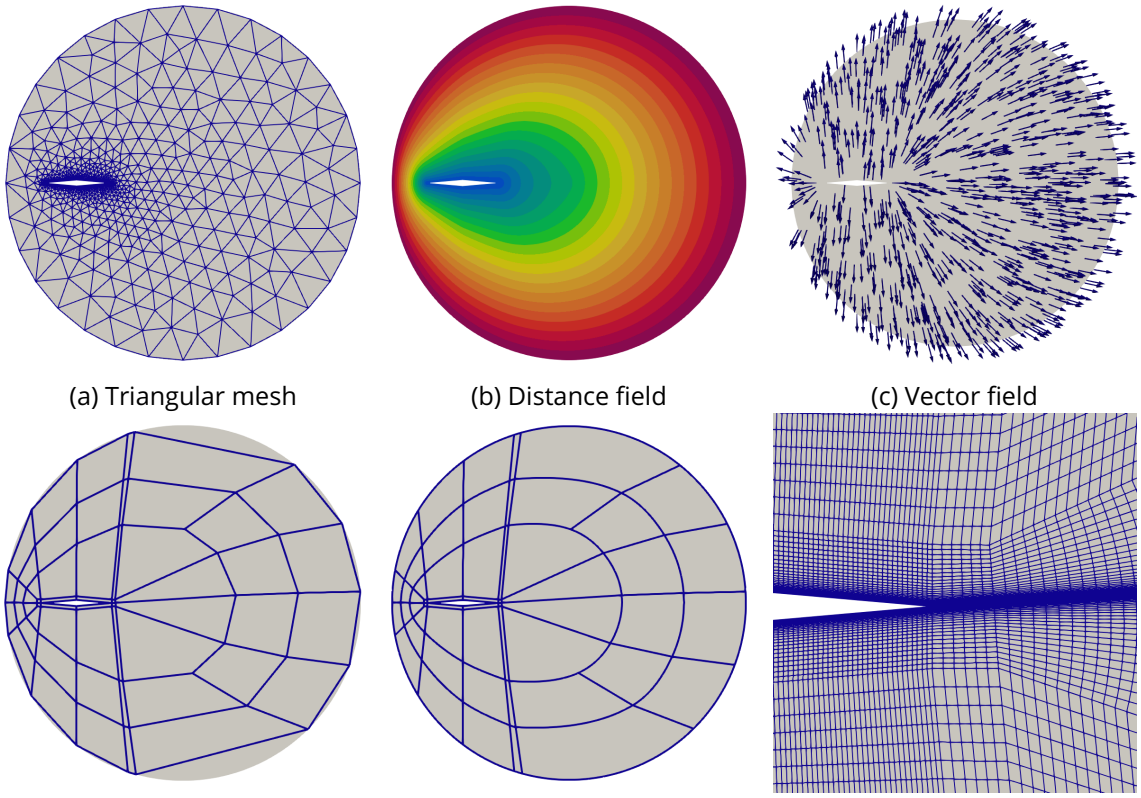
1. First of all, we adapt the work proposed in [RG11, RGRS12] to generate a coarser block structure, that is more constraining than a mesh. Our pre-meshed block surface can be anisotropic (i.e., with important aspect ratio between quadrangular cells) while the pre-meshed surface looks relatively regular in [RG11, RGRS12].

2. We introduce, in addition to the 3 different distance fields (see Fig. 18.b and Fig. 19.b), a vector field that leads the directions for the extrusion of the blocks (see Fig. 18.c and Fig. 19.c).

3. We only consider half of the patterns used to solve conflicts on a layer, but we introduce the notion of **closed path**.

4. We curve the block structure and use an interval assignment algorithm to generate the final mesh. In contrast, authors of [RG11, RGRS12] use templates to refine hexahedral cells (which could degrade the blocking structure).

Considering this global pipeline, this manuscript is divided into four main parts. Chapter 1 is a general part to introduce different notations and definitions linked to the fields of meshing and Computational Fluid Dynamics in the specific case of supersonic and hypersonic flow mechanics. This chapter contains all the information necessary for a good understanding of this manuscript. An already informed reader may want to skip it. The next three chapters correspond to the main steps of our approach presented in Figures 18 and 19. The method was first designed in 2D to be extensible to 3D. The main steps of the whole method are the same (see Fig. 18 and Fig. 19), even if there are a few minor differences. For this reason, in



(a) Triangular mesh (b) Distance field (c) Vector field

(d) Linear blocks (e) Curved blocks (f) Final block-structured mesh

Figure 19: Main stages of our approach illustrated in 2D.

this manuscript, the general method is presented in nD, where n refers to the physical domain dimension. Dedicated comments related to the 2D or 3D method are presented in specific frames when needed. The extrusion algorithm to generate the linear block structure is presented in Chapter 2 (see corresponding steps on Fig. 18.b, c, and d). Chapter 3 is dedicated to the curving of a linear block structure using two different approaches and the final discretization of the curved block structure into a mesh (see corresponding steps on Fig. 18.e and f). In Chapter 4, we discuss the generated mesh quality through purely geometrical criteria analysis and the application of numerical simulation on meshes generated with our method. Let us note that our meshing algorithm developed and presented here is freely available and implemented in the C++ framework GMDS[3] [gmd, LWB08].

In addition to this manuscript, we provide a detailed list of the communications linked to this thesis work.

Peer-Reviewed Papers

- Roche C, Breil J, Hocquellet T, Ledoux F. Block-structured quad meshing for supersonic flow simulations. International Meshing Roundtable 2023 (SIAM IMR23), Amsterdam, The Netherlands, March 2023. [RBHL23]

- Mittal K, Dobrev V.A., Knupp P, Kolev T, Ledoux F, Roche C, Tomov V.Z. Mixed-Order

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

Meshing using rp-adaptivity for Surface Alignment with Implicit Geometries. International Meshing Roundtable 2024 (SIAM IMR24), Baltimore, USA, March 2024. [MDK+24]

Research Note

- Roche C, Breil J, Calderan S, Hocquellet T, Ledoux F. Curved Hexahedral Block Structure Generation by Advancing Front. SIAM International Meshing Roundtable Workshop (SIAM IMR24), Baltimore, USA, March 2024.

Proceeding

- Roche C, Breil J, Olazabal M. Mesh regularization of ablating hypersonic vehicles. In 8th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2022), Oslo, Norway, June 2022. [RBHL23]

Talks

- Roche C, Breil J, Hocquellet T, Ledoux F. Block-structured quad meshing for supersonic flow simulations. SIAM International Meshing Roundtable Workshop (SIAM IMR23), Amsterdam, The Netherlands, March 2023.
- Roche C, Breil J, Hocquellet T, Ledoux F. Advancing-front block structure generation for atmospheric re-entry simulations. 57th 3AF International Conference on Applied Aerodynamics, High speed aerodynamics, from transonic to hypersonic, Bordeaux, France, March 2023.
- Roche C, Breil J, Hocquellet T, Ledoux F. Level-Set for CFD Quad Meshing. 22th Computational Fluids Conference (CFC2023), Cannes, France, April 2023.
- Roche C, Breil J, Calderan S, Hocquellet T, Ledoux F. Curved Hexahedral Block Structure Generation by Advancing Front. SIAM International Meshing Roundtable Workshop (SIAM IMR24), Baltimore, USA, March 2024.
- Mittal K, Dobrev V.A., Knupp P, Kolev T, Ledoux F, Roche C, Tomov V.Z. Mixed-Order Meshes through rp-Adaptivity for Surface Fitting to Implicit Geometries. SIAM International Meshing Roundtable Workshop (SIAM IMR24), Baltimore, USA, March 2024.
- Roche C, Breil J, Hocquellet T, Ledoux F. Génération de maillages hexaédriques structurés par blocs pour la rentrée atmosphérique. Journée des Doctorants de la DAM, the French Alternative Energies and Atomic Energy Commission, Avrainville, France, April 2024.

Posters

- Roche C, Breil J, Hocquellet T, Ledoux F. Automatic hexahedral mesh generation for atmospheric re-entry. Journée des doctorants, the French Alternative Energies and Atomic Energy Commission, Arcachon, France, May 2022. Best Poster Award.

- Roche C, Breil J, Hocquellet T, Ledoux F. Automatic 2D curved block-structured mesh generation for atmospheric re-entry. Scientific evaluation of the French Alternative Energies and Atomic Energy Commission in atmospheric re-entry, Bordeaux, France, November 2022.
- Roche C, Breil J, Hocquellet T, Ledoux F. Block-structured 2D mesh generation for supersonic flow simulation. Scientific evaluation of the French Alternative Energies and Atomic Energy Commission in high performance computing, Paris, France, December 2022.
- Roche C, Breil J, Hocquellet T, Ledoux F. Block-Structured Quad Meshing for Supersonic Flow Simulations. SIAM International Meshing Roundtable Workshop (SIAM IMR23), Amsterdam, The Netherlands, March 2023.

Chapter 1

DEFINITIONS & STATE OF THE ART

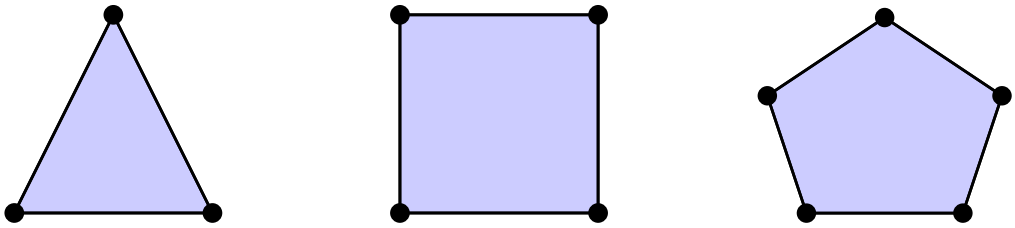
This chapter remains general and allows us to introduce the two fields related to this work: hexahedral mesh generation and Computational Fluid Dynamics (CFD) in the specific case of atmospheric re-entry. An advised reader may want to skip this chapter. We first explain what a mesh is and define multiple terms related to meshes and mesh-related techniques. We discuss how we represent and manipulate a mesh and the geometry in this work. Then, we dedicate a section to present the well-known mathematical modeling used in CFD and a discuss about mesh generation and quality for CFD. Finally, in the next section, we go through existing state-of-the-art techniques to generate hexahedral meshes automatically.

Contents

1.1 Mesh Definitions & Terminology

In this section, we go through a non-exhaustive list of mesh definitions and many words and adjectives used to qualify a mesh. We also remember some tools widely used in mesh manipulations.

1.1.1 Mesh Generalities



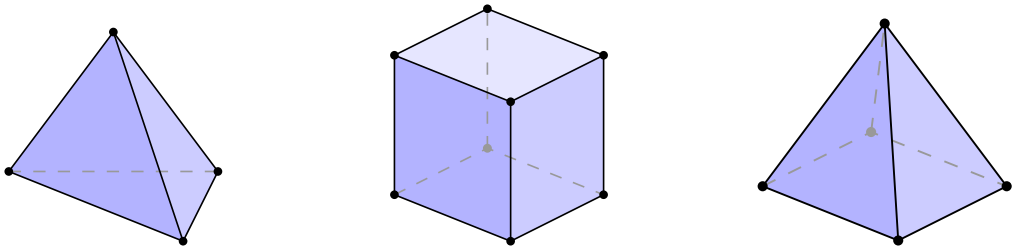
(a) Triangle (b) Quadrangle (c) Polygon

Figure 1.1: Examples of usual 2-cells.

A **mesh** can be defined as a discretization of a continuous geometric space Ω of dimension n in a collection of simpler elements K [FG99]. More specifically, $\mathcal{M}$ is a mesh of the domain Ω if:

$$\Omega = \overline{\bigcup_{K \in \mathcal{M}} \overset{\circ}{K}}, \tag{1.1}$$

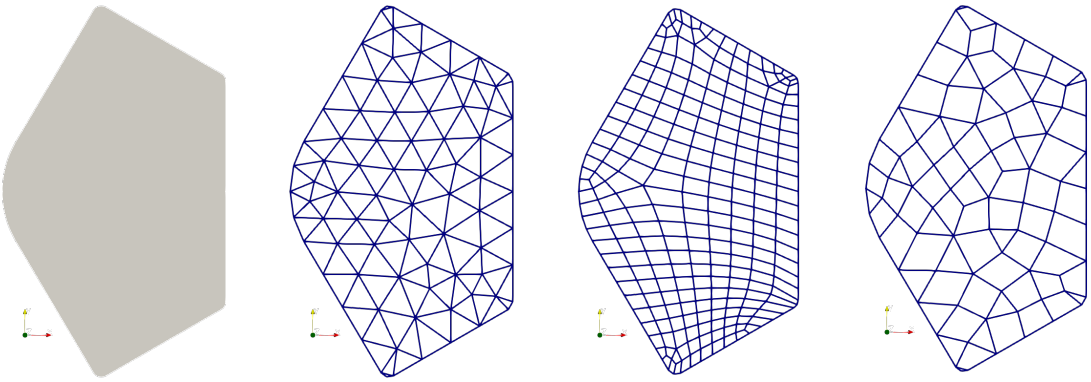
with $K \neq \emptyset \ \forall K \in \mathcal{M}$ and $\overset{\circ}{K_1} \cap \overset{\circ}{K_2} = \emptyset$, $\forall K_1, K_2 \in \mathcal{M}$. It means that the interior of all the elements K of $\mathcal{M}$ are not empty, the intersection of the interiors of two different elements is empty, and it wholly covers Ω. The continuous domain Ω is also called the **geometry**.



(a) Tetrahedron (b) Hexahedron (c) Pyramid

Figure 1.2: Examples of usual 3-cells.

A mesh is composed of a set of simpler elements of **dimension** n, called **n-cells**, where n is also the dimension of the mesh. Examples of usual 2-cells and 3-cells are given respectively in Figure 1.1 and Figure 1.2. An n-cell is bounded by **n–1-cells** (e.g., a hexahedron is a 3-cell bounded by 6 quadrangular 2-cells). Usually, 0-cells are referred as **nodes** (e.g., a hexahedron is a 3-cell defined by 8 nodes), and 1-cells as **edges** (e.g., a hexahedron has 12 edges). The interface between two n-cells are $n-1$-cells. When the dimension n is equal to 2 (**2D**), the n-cells are 2D shapes (*e.g.* quadrangles, triangles, polygons, ...). While in dimension $n = 3$ (**3D**), the n-cells are 3D shapes (*e.g.* hexahedra, tetrahedra, pyramids, polyedra, ...). In the 3D case, the $n-1$-cells are 2D shapes. Let us note that the $n-1$-cells are not necessarily planar. The



(a) Geometry (b) Triangular Mesh (c) Quadrangular Mesh (d) Hybrid Mesh (quadrangular and triangular n-cells)

Figure 1.3: Example of a triangular mesh (b), a quadrangular mesh (c), and a hybrid mesh (d) inside the geometry (a).

cells are also called **entities** or **elements**. The relation between two different mesh entities is referred to as **connectivity**.

Meshes are classified into different categories depending on their topology. According to Frey et al. [FG99], a **structured mesh** is a mesh whose connectivity table is of finite difference type. If we take the particular case of a 3D mesh, each node can be stored in a matrix $M_{l,m,n}$, where l, m, and n are the number of nodes in each direction. Those nodes are connected by cells in a way that, given a node $n_{i,j,k}$ of the mesh, by going through the index i+1, j+1, k+1, then i–1, j–1 and k–1, we are back on the node $n_{i,j,k}$. In other words, structured meshes are, topologically, grids. This type of mesh storage is less memory-consuming, and the access to the adjacent elements is direct. There is no need for special connectivity tables to reach the adjacent elements. By opposition, a mesh that does not have this trivial connectivity is called an **un-structured mesh** (see Fig. 1.3.b, c, and d).

A mesh composed of n-cells of at least two different geometric shapes (e.g., quadrangular and triangular in Fig. 1.3.d) is called an **hybrid mesh**.

(a) Conformal (b) Non-Conformal

Figure 1.4: Example of conformal mesh (a), and non-conformal mesh (b).

A mesh is defined as **conformal** if the intersection between two elements of this mesh is empty or a common node, edge, or face to the two elements. Considering the example given in Figure 1.4, the mesh (a) is conformal while the quadrangular mesh (b) is non-conformal. For instance, the intersection of the two n-cells Q_1 and Q_2 is, among other things, composed of the red node (●). This red node participates to define the quadrangular n-cell Q_2 but does not

belong to the quadrangular n-cell Q_1.

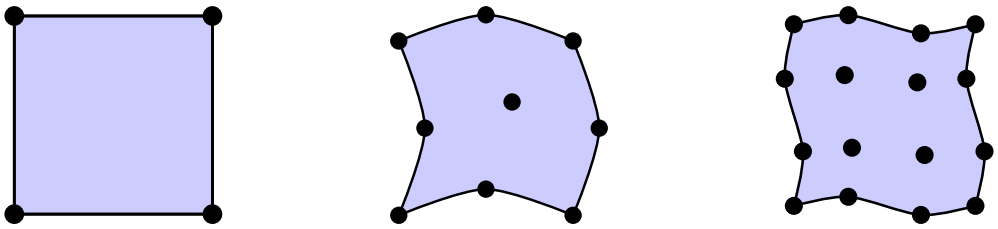
Finally, different words are used to qualify the geometrical properties of a mesh. For example, a mesh is qualified as **anisotropic** if its cell size is not uniform in space and direction. For example, the mesh is considered anisotropic if some cells are stretched out in one direction.

(a) Blocks ( ) (b) Mesh on blocks ( )

Figure 1.5: Blocking (a), and an example of mesh generated on this block structure (b).

A **blocking**, or **block structure** (see Fig. 1.5.a) is usually considered as a coarse mesh of quadrangular (2D) or hexahedral (3D) cells. Those cells are referred to as **blocks**, and their nodes (●) as **block corners**. A **blocking** is meant to be subdivided to provide a mesh (see Fig. 1.5.b). As we may observe, the blocking can be unstructured. However, the mesh generated in each block is structured. One can notice that to obtain a conformal final mesh, two opposite block edges must share the same discretization (i.e., the same number of mesh nodes). The main advantage of working with blocks is that we need to manipulate a few elements, and the structure can benefit from the structured mesh storage by blocks. However, this can lead to an insufficient representation of the geometry. Moreover, working with a coarse structure may lead to a shortage of degrees of freedom while applying meshing tools and methods (such as smoothing).

1.1.2 High-Order Meshes



(a) p_1 (b) p_2 (c) p_3

Figure 1.6: High order quadrangular cells. A cell is plotted in light blue ( ), the degrees of freedom of the cell are plotted with black dots (●) while the edges of the element are plotted in black ( ).

A **high-order $n-$cell** is a cell represented by nonlinear elements. Usually, in the case of a quadrangular cell of order 1, the cell is described by 4 degrees of freedom, or nodes, as in Figure 1.6.a. All the edges of the cell are linear. In some cases, a cell can be considered nonlinear. This means that more degrees of freedom are needed to represent the cell. For instance, in case of a quadrangular cell of degree $p = 2$, 9 degrees of freedom are used to define the quadrangular cell (see Fig. 1.6.b), and 16 degrees of freedom for a quadrangular

cell of degree $p=3$ (see Fig. 1.6.c), etc. Increasing the order of the mesh elements increases the number of degrees of freedom, increasing the memory and computing time consumption. Polynomials represent each edge of the element. Depending on the representation used for the cells, a high-order cell can be defined only with the degree of freedom on its edges. A cell of order $p>1$ can also be referred to as a **curved cell**.

A **high-order mesh**, or **curved mesh**, is a mesh composed of high-order cells. A high-order mesh can comprise cells of different degrees in the domain. By opposition, a mesh composed only of cells of degree $p=1$ will be referred to as **low-order mesh**, or **linear mesh**.

1.1.3 Meshing Tools

Mesh Adaptation

Mesh adaptation consists of many methods and algorithms that aim to modify a mesh. Those methods can be divided into different subcategories.

- $h-$**adaptivity** is a category of methods concerning mesh adaptation at non-constant topology. This means the mesh's topology will change in terms of node number (degree of freedom) and cells to improve some metrics. The metric to optimize can be a mesh cell quality criteria or a field. To achieve this optimization, the method can, for instance, split or fuse some cells (e.g., Adaptive Meshing Refinement).
- $r-$**adaptivity** regroups a family of methods concerning mesh adaptation at constant topology. This means we do not change the connectivities between the different elements of the mesh; we keep the exact same number of elements from the beginning to the end of the process. However, the positions of the nodes change to improve some specific metrics (e.g., size fields, cell's shape, etc.). Those methods can also be referred to as **mesh smoothing**.
- $p-$**adaptivity** is a set of methods concerning mesh adaptation by modifying the polynomial order of the elements of the mesh.

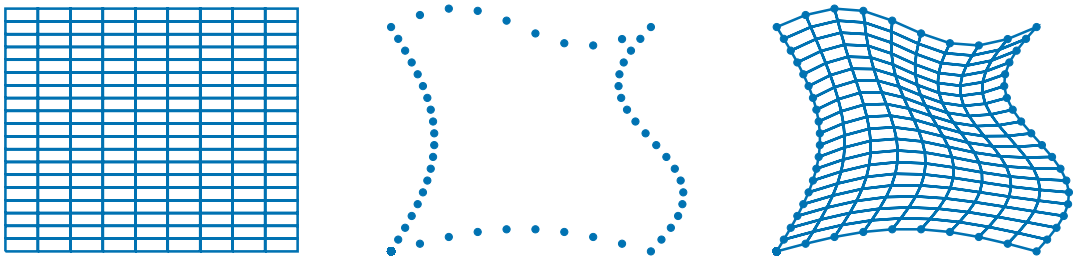
Linear Transfinite Interpolation 2D

Transfinite Interpolation (TFI) [TSW98, Smi99] is a well-known method to build a function over a planar domain by imposing a given function on the boundary. In this section, we redevelop the linear Transfinite Interpolation formulation. Other types of interpolation exist (e.g., Lagrangian TFI) but are not discussed in this document. For complete information on TFI, the reader should look at [TSW98, Smi99].

Let us consider a 2D grid of $N_i \times N_j$ points with known boundary positions. Linear Transfinite Interpolation is a way to compute the inner positions of the grid points by only using the boundary points coordinates. We note the function Φ, the mapping function to compute the physical coordinates of each point (ξ_i, η_j) of the grid parametrical space (see Fig. 1.7.a), where $\xi_i = \frac{i-1}{N_i-1} \in [0,1]$, and $\eta_j = \frac{j-1}{N_j-1} \in [0,1]$, with $i \in [\![1, N_i]\!]$, and $j \in [\![1, N_j]\!]$.

According to these notations, for $i \in [\![1, N_i]\!]$, and $j \in [\![1, N_j]\!]$, the physical position of the corresponding grid point (see Fig. 1.7.c) is computed using only the physical coordinates of boundary points (see Fig. 1.7.b) with the formula:

$$\begin{aligned}\Phi(\xi_i, \eta_j) =& (1-\xi_i)\Phi(0,\eta_j) + \xi_i\Phi(1,\eta_j) + (1-\eta_j)\Phi(\xi_i,0) + \eta_j\Phi(\xi_i,1) \\ &- (1-\xi_i)(1-\eta_j)\Phi(0,0) - (1-\xi_i)\eta_j\Phi(0,1) \\ &- \xi_i(1-\eta_j)\Phi(1,0) - \xi_i\eta_j\Phi(1,1).\end{aligned} \tag{1.2}$$



(a) Parametrical space (b) Boundary point positions in physical space (c) Final grid in physical space

Figure 1.7: Transfinite interpolation 2D example on a 10×20 grid of points.

Linear Transfinite Interpolation 3D

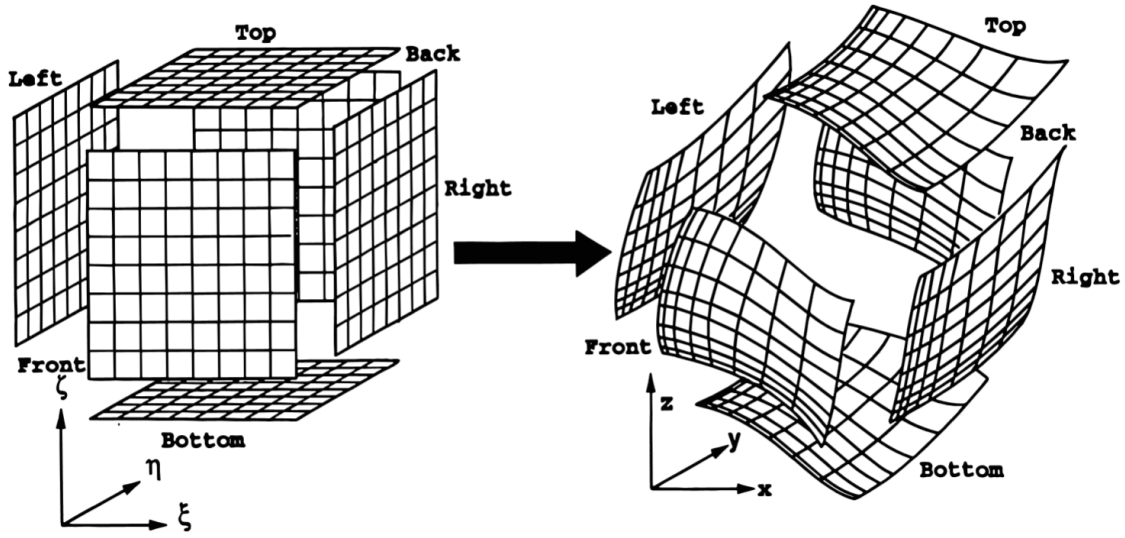
This part extends the linear Transfinite Interpolation given before to 3D grids. As before, we use the same notations as in [TSW98]. Let us consider a grid of $N_i \times N_j \times N_k$ points. We note the function Φ the mapping function to compute the physical coordinates at each point (ξ_i, η_j, ζ_k) of the grid in parametrical space (see Fig. 1.8.a), where $\xi_i = \frac{i-1}{N_i-1} \in [0,1]$, $\eta_j = \frac{j-1}{N_j-1} \in [0,1]$, and $\zeta_k = \frac{k-1}{N_k-1} \in [0,1]$, with $i \in [\![1, N_i]\!]$, $j \in [\![1, N_j]\!]$, and $k \in [\![1, N_k]\!]$.

According to these notations, $\xi_0 = 0$, and $\xi_{N_i} = 1$, $\eta_0 = 0$ and $\eta_{N_j} = 1$, $\zeta_0 = 0$, and $\zeta_{N_k} = 1$.

$$\begin{cases} U(\xi_i,\eta_j,\zeta_k) = & (1-\xi_i)\Phi(0,\eta_j,\zeta_k) + & \xi_i\Phi(1,\eta_j,\zeta_k) \\ V(\xi_i,\eta_j,\zeta_k) = & (1-\eta_j)\Phi(\xi_i,0,\zeta_k) + & \eta_j\Phi(\xi_i,1,\zeta_k) \\ W(\xi_i,\eta_j,\zeta_k) = & (1-\zeta_k)\Phi(\xi_i,\eta_j,0) + & \zeta_k\Phi(\xi_i,\eta_j,1) \end{cases} \tag{1.3}$$

$$\begin{cases} UW(\xi_i,\eta_j,\zeta_k) = & (1-\xi_i)(1-\zeta_k)\Phi(0,\eta_j,1) + & (1-\xi_i)\zeta_k\Phi(0,\eta_j,1) \\ + & \xi_i(1-\zeta_k)\Phi(1,\eta_j,0) + & \xi_i\zeta_k\Phi(1,\eta_j,1) \\ UV(\xi_i,\eta_j,\zeta_k) = & (1-\xi_i)(1-\eta_j)\Phi(0,0,\zeta_k) + & \xi_i\eta_j\Phi(1,1,\zeta_k) \\ + & \xi_i(1-\eta_j)\Phi(1,0,\zeta_k) + & (1-\xi_i)\eta_j\Phi(0,1,\zeta_k) \\ VW(\xi_i,\eta_j,\zeta_k) = & (1-\eta_j)(1-\zeta_k)\Phi(\xi_i,0,0) + & (1-\eta_j)\zeta_k\Phi(\xi_i,0,1) \\ + & \eta_j(1-\zeta_k)\Phi(\xi_i,1,0) + & \eta_j\zeta_k\Phi(\xi_i,1,1) \end{cases} \tag{1.4}$$

and



(a) Parametrical space (b) Physical space

Figure 1.8: Transfinite interpolation 3D boundary surfaces [TSW98].

$$\begin{aligned} UVW(\xi_i,\eta_j,\zeta_k) = \quad & (1-\xi_i)(1-\eta_j)(1-\zeta_k)\Phi(0,0,0) + \quad (1-\xi_i)(1-\eta_j)\zeta_k\Phi(0,0,1) \\ + \quad & (1-\xi_i)\eta_j(1-\zeta_k)\Phi(0,1,0) + \quad \xi_i(1-\eta_j)\zeta_k\Phi(1,0,1) \\ + \quad & \xi_i\eta_j(1-\zeta_k)\Phi(1,1,0) + \quad \xi_i\eta_j\zeta_k\Phi(1,1,1) \\ + \quad & (1-\xi_i)\eta_j\zeta_k\Phi(0,1,1) + \quad \xi_i(1-\eta_j)(1-\zeta_k)\Phi(1,0,0) \end{aligned} \tag{1.5}$$

Finally, for $i \in [\![1, N_i]\!]$, $j \in [\![1, N_j]\!]$, and $k \in [\![1, N_k]\!]$, the physical position of the corresponding point is computed using only the boundary points in the physical space (see Fig. 1.8.b) using the formula:

$$\begin{aligned} \Phi(\xi_i,\eta_j,\zeta_k) = \quad & U(\xi_i,\eta_j,\zeta_k) + \quad V(\xi_i,\eta_j,\zeta_k) + \quad W(\xi_i,\eta_j,\zeta_k) \\ - \quad & UW(\xi_i,\eta_j,\zeta_k) - \quad UV(\xi_i,\eta_j,\zeta_k) - \quad VW(\xi_i,\eta_j,\zeta_k) \\ + \quad & UVW(\xi_i,\eta_j,\zeta_k). \end{aligned} \tag{1.6}$$

The corresponding algorithms to compute the point's positions over a 2D or a 3D grid using the linear TFI are presented in Appendix A.

1.1.4 Mesh Quality

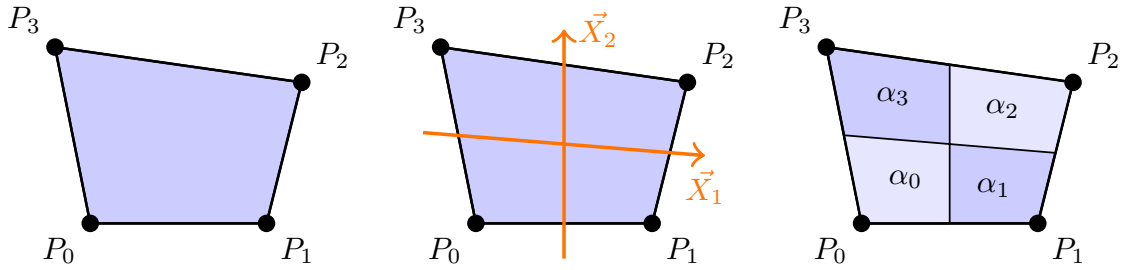
Mesh quality is a very discussed topic [KET+06, Knu00, Knu01, Knu07] regarding mesh generation and adaptation. In our case, the quality of a mesh can be evaluated on two different levels. The first one is the result of the numerical simulation on the mesh of interest, and all the information given by this simulation (e.g., errors in the domain). The second level is a pure geometric analysis of the cells of the mesh before any simulation. This is what is explored in this section.

This part presents some purely geometric quality criteria introduced in the well-known Verdict Geometric Quality Library [KET+06]. If they introduce many criteria over

different element types, we focus here on the scaled Jacobian and the skewness only on quadrangular 2-cells and hexahedral 3-cells.

Quadrangle Scaled Jacobian

This criterion is fully explained by KNUPP [KET+06, Knu00] and is re-introduced here as a reminder.



(a) Quadrangle (b) Principal axes (c) Areas

Figure 1.9: Notations used in [KET+06] to compute the Scaled Jacobian of a quadrangle.

Given the quadrangle of Figure 1.9.a, let us note the vectors corresponding to the edges

$$\begin{cases} \overrightarrow{L_0} = \overrightarrow{P_1} - \overrightarrow{P_0} \\ \overrightarrow{L_1} = \overrightarrow{P_2} - \overrightarrow{P_1} \\ \overrightarrow{L_2} = \overrightarrow{P_3} - \overrightarrow{P_2} \\ \overrightarrow{L_3} = \overrightarrow{P_0} - \overrightarrow{P_3} \end{cases} \tag{1.7}$$

and the edges lengths $L_i = ||\overrightarrow{L_i}||$, $\forall i \in \{0,1,2,3\}$. A normal vector is associated with each corner

$$\begin{cases} \overrightarrow{N_0} = \overrightarrow{L_3} \times \overrightarrow{L_0} \\ \overrightarrow{N_1} = \overrightarrow{L_0} \times \overrightarrow{L_1} \\ \overrightarrow{N_2} = \overrightarrow{L_1} \times \overrightarrow{L_2} \\ \overrightarrow{N_3} = \overrightarrow{L_2} \times \overrightarrow{L_3} \end{cases}, \tag{1.8}$$

and the associated normalized vector are given by $\hat{n}_i = \frac{\overrightarrow{N_i}}{||\overrightarrow{N_i}||}$, $\forall i \in \{0,1,2,3\}$. The two principal axes of the quadrangle (see Fig. 1.9.b) are

$$\begin{cases} \overrightarrow{X_1} = (\overrightarrow{P_1} - \overrightarrow{P_0}) + (\overrightarrow{P_2} - \overrightarrow{P_3}) \\ \overrightarrow{X_2} = (\overrightarrow{P_2} - \overrightarrow{P_1}) + (\overrightarrow{P_3} - \overrightarrow{P_0}) \end{cases}, \tag{1.9}$$

they define a "center" normal $\overrightarrow{N_c} = \overrightarrow{X_1} \times \overrightarrow{X_2}$ of unit-length $\hat{n}_c = \frac{\overrightarrow{N_c}}{||\overrightarrow{N_c}||}$.

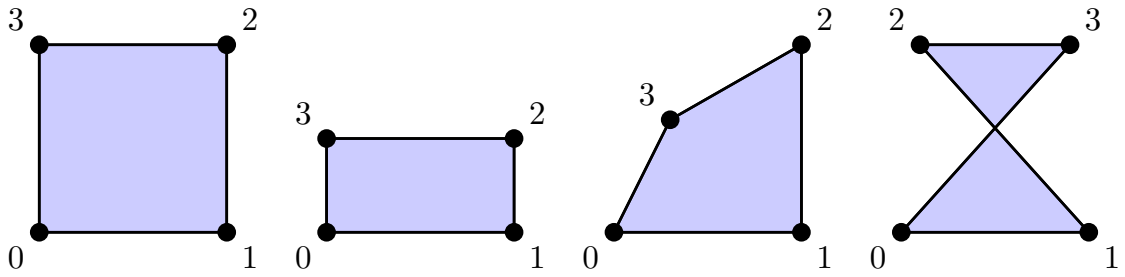
Finally, four areas are introduced, corresponding to the subdivision of the quadrangle into four quadrangles (see Fig. 1.9.c), associated with each vertex. They are noted

$$\forall i \in \{0,1,2,3\}, \ \alpha_i = \hat{n}_c \cdot \overrightarrow{N_i}. \tag{1.10}$$

According to these notations, the value of the scaled Jacobian is given by

$$q_{sj} = \min\left(\frac{\alpha_0}{||\overrightarrow{L_0}||||\overrightarrow{L_3}||}, \frac{\alpha_1}{||\overrightarrow{L_1}||||\overrightarrow{L_0}||}, \frac{\alpha_2}{||\overrightarrow{L_2}||||\overrightarrow{L_1}||}, \frac{\alpha_3}{||\overrightarrow{L_3}||||\overrightarrow{L_2}||}\right) \tag{1.11}$$

The range of the scaled Jacobian as defined in Equation (1.11) is $[-1, 1]$. For a square or a rectangle (see Fig. 1.10.a and .b), the value of the scaled jacobian is equal to 1. If the scaled jacobian of the cell takes a negative value (see Fig. 1.10.d), the cell is considered as **inverted**. It can also be referred to as **tangled** or **overlapping**.



(a) $q_{sj} = 1$ (b) $q_{sj} = 1$ (c) $q_{sj} = 0.5547$ (d) $q_{sj} = -0.743294$

Figure 1.10: Examples of scaled Jacobian over different quadrangular cells.

Quadrangle Skew

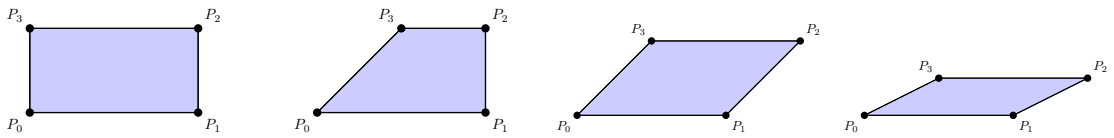
In order to compute the skew of a quadrangle adapted from [Rob87], and presented here as in [KET+06], we introduce the normalized vectors of principal axes (see Eq. (1.9)):

$$\begin{cases} \hat{X}_1 = \frac{\overrightarrow{X_1}}{||\overrightarrow{X_1}||} \\ \hat{X}_2 = \frac{\overrightarrow{X_2}}{||\overrightarrow{X_2}||} \end{cases}, \tag{1.12}$$

then, the skewness is defined as

$$q_s = |\hat{X}_1 \cdot \hat{X}_2|. \tag{1.13}$$

The quadrangle skew normal range is between $[0, 1]$. For a unit square, this value is equal to 0. Figure 1.11 represents different quadrangles and their corresponding skew value.



(a) $q_s = 0$ (b) $q_s = 0.447214$ (c) $q_s = 0.707107$ (d) $q_s = 0.894427$

Figure 1.11: Skew examples over different quadrangular cells.

Hexahedral Scaled Jacobian

This criterion is also fully explained by KNUPP [Knu00, KET+06] and is re-introduced here as a reminder.

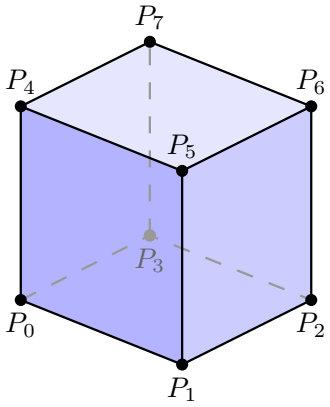


Figure 1.12: Notations used in [KET+06] to compute the Scaled Jacobian of a hexahedron.

Given the hexahedron of Figure 1.12, let us note the vectors corresponding to the edges

$$\begin{array}{lll} \overrightarrow{L_0} = \overrightarrow{P_1} - \overrightarrow{P_0} & \overrightarrow{L_4} = \overrightarrow{P_4} - \overrightarrow{P_0} & \overrightarrow{L_8} = \overrightarrow{P_5} - \overrightarrow{P_4} \\ \overrightarrow{L_1} = \overrightarrow{P_2} - \overrightarrow{P_1} & \overrightarrow{L_5} = \overrightarrow{P_5} - \overrightarrow{P_1} & \overrightarrow{L_9} = \overrightarrow{P_6} - \overrightarrow{P_5} \\ \overrightarrow{L_2} = \overrightarrow{P_3} - \overrightarrow{P_2} & \overrightarrow{L_6} = \overrightarrow{P_6} - \overrightarrow{P_2} & \overrightarrow{L_{10}} = \overrightarrow{P_7} - \overrightarrow{P_6} \\ \overrightarrow{L_3} = \overrightarrow{P_0} - \overrightarrow{P_3} & \overrightarrow{L_7} = \overrightarrow{P_7} - \overrightarrow{P_3} & \overrightarrow{L_{11}} = \overrightarrow{P_7} - \overrightarrow{P_4} \end{array} \tag{1.14}$$

with the corresponding lengths $L_i = ||\overrightarrow{L_i}||$, $\forall i \in [\![0, 11]\!]$. The principal axes are given by

$$\begin{cases} \overrightarrow{X_1} = (\overrightarrow{P_1} - \overrightarrow{P_0}) + (\overrightarrow{P_2} - \overrightarrow{P_3}) + (\overrightarrow{P_5} - \overrightarrow{P_4}) + (\overrightarrow{P_6} - \overrightarrow{P_7}) \\ \overrightarrow{X_2} = (\overrightarrow{P_3} - \overrightarrow{P_0}) + (\overrightarrow{P_2} - \overrightarrow{P_1}) + (\overrightarrow{P_7} - \overrightarrow{P_4}) + (\overrightarrow{P_6} - \overrightarrow{P_5}) \\ \overrightarrow{X_3} = (\overrightarrow{P_4} - \overrightarrow{P_0}) + (\overrightarrow{P_5} - \overrightarrow{P_1}) + (\overrightarrow{P_6} - \overrightarrow{P_2}) + (\overrightarrow{P_7} - \overrightarrow{P_3}) \end{cases}. \tag{1.15}$$

Nine 3×3 Jacobian matrices are defined, using the edge vectors $\overrightarrow{L_i}$ as the columns of each matrix

$$\begin{cases} A_0 = (\ \overrightarrow{L_0}, & -\overrightarrow{L_3}, & \overrightarrow{L_4}) \\ A_1 = (\ \overrightarrow{L_1}, & -\overrightarrow{L_0}, & \overrightarrow{L_5}) \\ A_2 = (\ \overrightarrow{L_2}, & -\overrightarrow{L_1}, & \overrightarrow{L_6}) \\ A_3 = (\ \overrightarrow{L_3}, & -\overrightarrow{L_2}, & \overrightarrow{L_7}) \\ A_4 = (\ \overrightarrow{L_{11}}, & \overrightarrow{L_8}, & -\overrightarrow{L_4}) \\ A_5 = (-\overrightarrow{L_8}, & \overrightarrow{L_9}, & -\overrightarrow{L_5}) \\ A_6 = (-\overrightarrow{L_9}, & \overrightarrow{L_{10}}, & -\overrightarrow{L_6}) \\ A_7 = (-\overrightarrow{L_{10}}, & -\overrightarrow{L_{11}}, & -\overrightarrow{L_7}) \\ A_8 = (\ \overrightarrow{X_1}, & \overrightarrow{X_2}, & \overrightarrow{X_3}) \end{cases}. \tag{1.16}$$

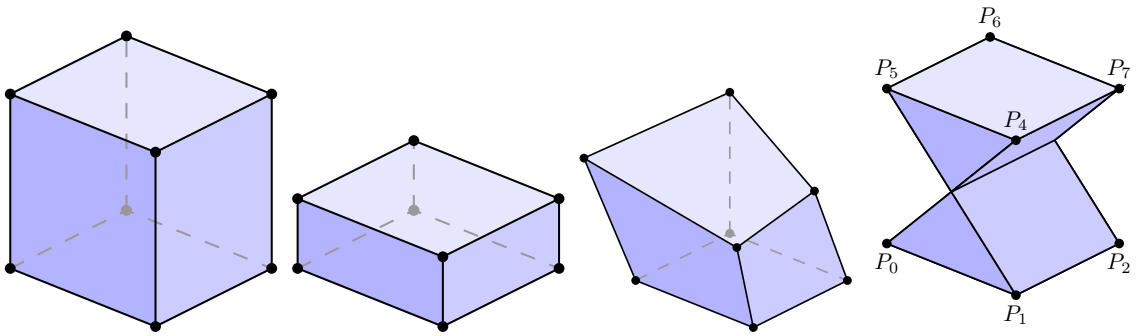
Given A one of these matrix defined by the column vectors $\overrightarrow{v_1}$, $\overrightarrow{v_2}$, and $\overrightarrow{v_3}$. Then, the normalized version of the Jacobian matrix A is defined by

$$\hat{A} = \left(\frac{\overrightarrow{v_1}}{||\overrightarrow{v_1}||}, \frac{\overrightarrow{v_2}}{||\overrightarrow{v_2}||}, \frac{\overrightarrow{v_3}}{||\overrightarrow{v_3}||} \right). \tag{1.17}$$

Then, $\forall i \in [\![0, 8]\!]$, the determinant of the normalized Jacobian matrix $\hat{A}_i$ is $\hat{\alpha}_i = \det(\hat{A}_i)$.

According to these notations, the value of the scaled Jacobian of the hexahedral cell is given by

$$q_{sj} = \min_{i \in \{0,1,\ldots,8\}} \{\hat{\alpha}_i\}. \tag{1.18}$$

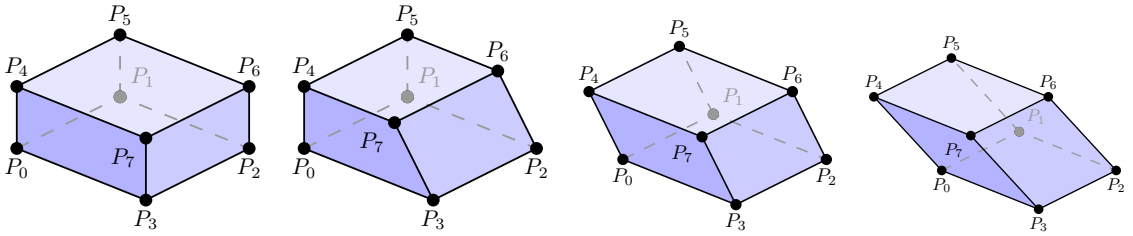


(a) $q_{sj} = 1$ (b) $q_{sj} = 1$ (c) $q_{sj} = 0.1126$ (d) $q_{sj} = -0.707107$

Figure 1.13: Scaled Jacobian examples over different hexahedral cells.

Analogously to the scaled Jacobian over a quadrangle, the range of the scaled Jacobian over a hexahedron as defined in Equation (1.18) is $[-1, 1]$. For a cube or a rectangular cuboid (see Fig. 1.13.a and .b), the scaled Jacobian value equals 1. If the scaled Jacobian of the cell takes a negative value (see Fig. 1.10.d), the cell is considered as **inverted**. It can also be referred to as **tangled** or **overlapping**.

Hexahedral Skew



(a) $q_s = 0$ (b) $q_s = 0.351123$ (c) $q_s = 0.6$ (d) $q_s = 0.868243$

Figure 1.14: Skew examples over different hexahedral cells.

In order to compute the skew of a hexahedron adapted from [TF89], and presented here as in [KET+06], we introduce the normalized vectors of principal axes (see Eq. (1.15)):

$$\begin{cases} \hat{X}_1 = \frac{\overrightarrow{X_1}}{||\overrightarrow{X_1}||} \\ \hat{X}_2 = \frac{\overrightarrow{X_2}}{||\overrightarrow{X_2}||} \\ \hat{X}_3 = \frac{\overrightarrow{X_3}}{||\overrightarrow{X_3}||} \end{cases}, \tag{1.19}$$

then, the skewness is defined as

$$q_s = \max\{|\hat{X}_1 \cdot \hat{X}_2|, |\hat{X}_1 \cdot \hat{X}_3|, |\hat{X}_2 \cdot \hat{X}_3|\}. \tag{1.20}$$

As for a quadrangle, the skew-normal range is between $[0, 1]$, and the skew value of a cube is 0. Figure 1.14 presents different hexahedra and their corresponding skew value.

1.2 Mesh & Geometry Representation

Mesh representation is how the cells of a mesh are expressed and linked through the different connectivities. The representation chosen significantly impacts how different manipulations are implemented on the mesh. Various ways exist to represent meshes and geometry with different benefits and drawbacks. For instance, some are meant to optimize memory consumption, while others intend to make implementing some topological operations easier. The choice of a method to represent meshes also depends on the type of meshes and elements to use. This section focuses on the **cellular representation** of meshes used in GMDS[1] [gmd, LWB08], and how the geometry is handled.

1.2.1 Cellular Mesh Representation

In [LWB08], an n-dimension cellular **mesh model** can be defined by cells and connectivities between the different cells. When a mesh is created in GMDS [gmd], a mesh model has to be picked. The model defines the cell's dimensions used and the kind of connectivities supported between the selected cells of different dimensions. The cellular representation is re-introduced here, but the reader could refer to [Gar02] for more details.

(a) Elementary mesh model

(b) One another connectivity is added to the model (a)

Figure 1.15: Elementary mesh models in GMDS [gmd].

The most elementary model available to represent a mesh of dimension n is illustrated in Figure 1.15.a. This model only represents the cells of dimension n and the mesh nodes. As the connectivities supported are represented here using the black vectors ($\longrightarrow$), it is understood that an n-cells knows the nodes it is connected to, but the nodes don't know to which cells they are connected to. In case it is needed to know for a node to which n-cell it is connected to, a new connectivity must be set in the mesh model. This new model is illustrated in Figure 1.15.b.

Due to the methods we want to implement in this work, we decided to work with a complete mesh model. In this model, an i-cell, with $i \in [\![0, 3]\!]$, knows all cells of dimension $j \neq i$ it is adjacent to. The main advantage is that we can access all types of cells of our mesh and all the cells connected to a specific cell when needed. The main drawback is the memory consumption. However, we can eliminate this because we use this representation to manipulate the block structure, which is considered as being a very coarse mesh. The other main constraint is that when we create a cell, we have to build the cells of lower dimensions that do not exist in the mesh yet and pay attention to giving all the connected cells the information to be connected to those new cells. For instance, if one wants to create a quadrangular 2-cell, first, we need to check if the 1-cells of this quadrangular cell have already been created or if we have to build it. Then, we must initialize all the connectivities corresponding to the 2-cell and the new 1-cells

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

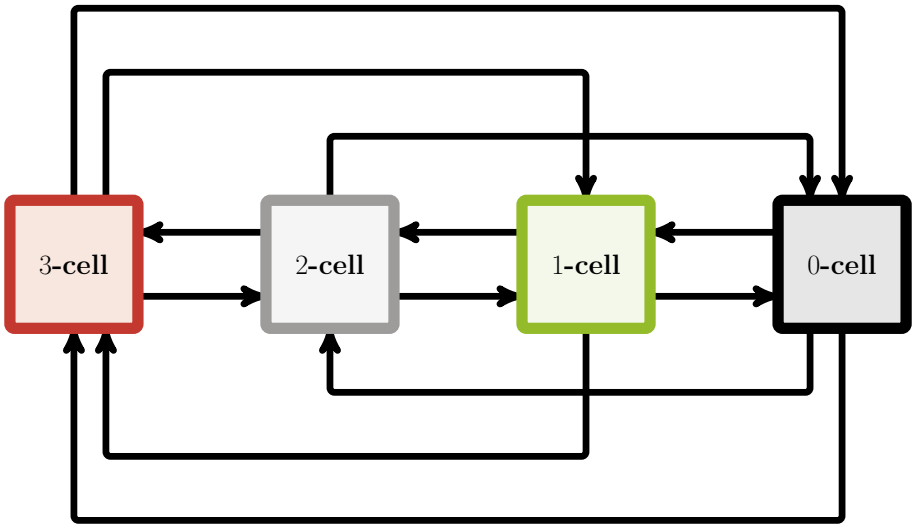


Figure 1.16: Very complete cellular mesh model representation for a mesh of dimension $n = 3$ in GMDS [gmd].

(i.e., node) created.

1.2.2 Geometry Representation & Geometric Classification

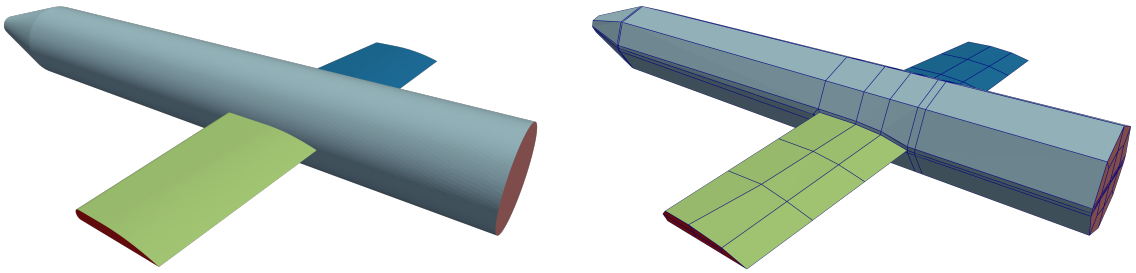
This subsection provides a basic understanding of the geometric representation through a background mesh in GMDS [gmd] and its geometric classification, not a formal definition. This work doesn't use a Computer Aided Design representation of the physical domain. Instead, a first mesh is used as the geometry reference (see Fig. 1.17.a).

A physical domain Ω is represented by its boundaries. The geometric classification labels the geometric features of the physical domain into different geometrical entities:

- **Geometrical Point**: geometrical entity of dimension 0;
- **Geometrical Curve**: geometrical entity of dimension 1;
- **Geometrical Surface**: geometrical entity of dimension 2.

In our case, the geometric classification consists in attributing to each mesh cell the geometry dimension and entity on which this cell relies on. Let us note that an n-cell lies on a geometric entity of dimension p with $n \leq p$. For instance, a 2-cell (e.g., quadrangle) can not live on a geometrical curve.

Figure 1.17 represents a practical example of surface classification. In Figure 1.17.a, each color represents a set of the background mesh cells classified on the same geometric surface. Using this definition of the surface, we attribute to each coarse 2-cell of the Figure 1.17.b coarse mesh a value corresponding to the surface on which it lies. Only geometric surfaces are represented in this case. But on the geometry of Figure 1.17.a, geometric curves and geometric points are also defined. Each node of Figure 1.17.b is associated with a geometric point, curve, or surface. Each edge is associated with a curve or a surface. If a node is located at the intersection between different geometric entities, we choose the one of the lowest dimension.



(a) Geometric surfaces (b) 2-cells lying on the geometric surfaces

Figure 1.17: Surfaces geometric classification. Each color of (a) represents a different geometric surface. Each quadrangular 2-cell of (b) is associated with one geometric surface of (a).

This geometric classification is particularly useful while performing mesh modifications to be sure to remain on the initial geometric entity.

1.3 Computational Fluid Dynamics for Atmospheric Re-Entry

1.3.1 Mathematical Modeling for Computational Fluid Dynamics

Among the different mathematical modeling existing for Computational Fluid Dynamics, we focus on two different systems of equations in this work: the Navier-Stokes and the Euler equations. The Navier-Stokes (NS) equations are nonlinear partial differential equations used in fluid mechanics to describe the flow of a viscous and compressible fluid. The first equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho \overrightarrow{V} \right) = 0 \tag{1.21}$$

is the continuity equation. The momentum conservation equations are

$$\frac{\partial \left(\rho \overrightarrow{V} \right)}{\partial t} + \nabla \cdot \left(\rho \overrightarrow{V} \overrightarrow{V} \right) = \nabla \cdot (\tau - pI) + \rho \overrightarrow{g}. \tag{1.22}$$

Then, the energy equation is given by

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot \left(\rho E \overrightarrow{V} \right) = \nabla \cdot \left((\tau - pI) \cdot \overrightarrow{V} \right) + \rho \overrightarrow{g} \cdot \overrightarrow{V} - \nabla \cdot \overrightarrow{\Phi}. \tag{1.23}$$

In these equations, t is the time (s), ρ is the fluid density (kg.m^{-3}), $\overrightarrow{V}$ 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, $\overrightarrow{g}$ is the gravity vector (m.s^{-2}), and $\overrightarrow{\Phi}$ is the heat flux vector (J.m^{-2}.s^{-1}). This work considers that a perfect gas equation of state characterizes the fluid.

In Computational Fluid Dynamics, the Reynolds Averaged Navier-Stokes (RANS) equations formulation is often used. The RANS equations are obtained using Reynolds decomposition of the flow quantities. Each flow variable is expressed as the sum of a mean and fluctuation terms. The **Euler** equations are obtained by considering the Navier-Stokes equations without

the viscosity effects (i.e., by removing the viscous stress tensor τ in Eq. (1.22) and (1.23)) and the heat flux.

In CFD, the **Mach number** is defined as

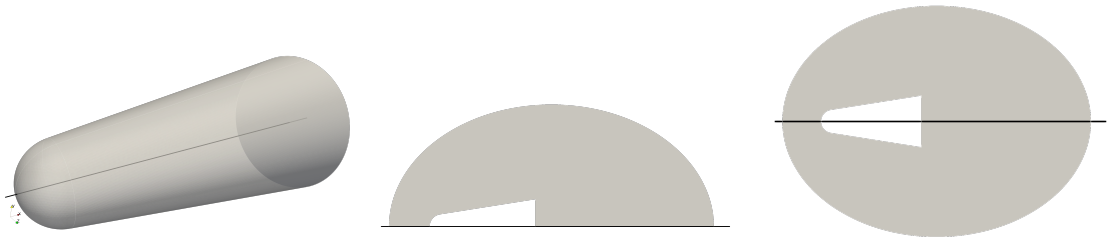
$$M = \frac{u}{c}, \tag{1.24}$$

with u the local flow velocity, and c the speed of sound in the medium. Considering this definition, at $M = 1$, the local flow velocity equals the sound speed in the medium. In case $M_\infty = \frac{u_\infty}{c} < 1$, where u_∞ is the magnitude of $\overrightarrow{u_\infty}$ (see Fig. 16), we refer to the vehicle as **subsonic**, and in case $M > 5$, the vehicle is referred as **hypersonic**. In between, the object is considered as **supersonic**.

The Need of 3D Meshes

Usually, many geometries of interest are **axisymmetric**, which means symmetrical around an axis of revolution (see Fig. 1.18.a). To handle those geometries when the flow Angle of Attack (AoA) is null, many solvers propose a way to simulate a flow around a 3D axisymmetric geometry by only computing the solution of the equations around a slice of the domain and imposing a symmetry condition on the boundary corresponding to the symmetry axis (▬) (see Fig. 1.18.b). This is equivalent to considering the problem as a 2D problem. However, emerging non-symmetrical geometries arouse the need for real 3D simulation and so 3D hexahedral meshes. Also, in case the AoA is not equal to 0°, the geometry may be axisymmetric, but the simulation needs to be a 3D simulation, as the flow around the vehicle is not symmetric anymore.

Let us recall that this work aims to provide a solution to generate 3D meshes. For this reason, and as the algorithm was first elaborated in 2D, we decided to work on the 2D case with constraints similar to those encountered in 3D. To do so, we considered the geometry completely immersed in the fluid (see Fig. 1.18.c), even if it is not the natural approach for some 2D geometries treated in CFD. However, a post-processing approach is proposed to cut half part of the domain to provide a suitable block-structured mesh of the domain of Figure 1.18.b when possible (see Appx. D).

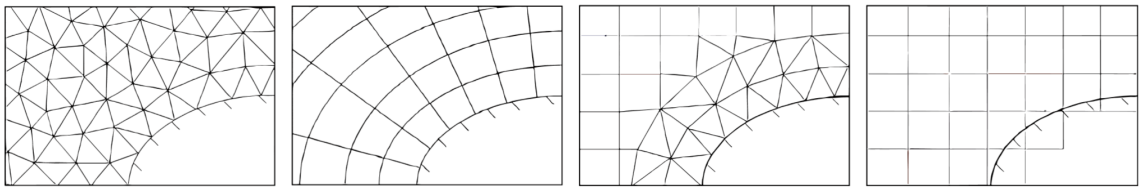


(a) 3D axisymmetric geometry (b) Corresponding 2D fluid domain for simulation (c) Corresponding 2D meshed domain in this work

Figure 1.18: Example of 3D axisymmetric geometry (a) around the black axis (▬).

1.3.2 Mesh for Computational Fluid Dynamics

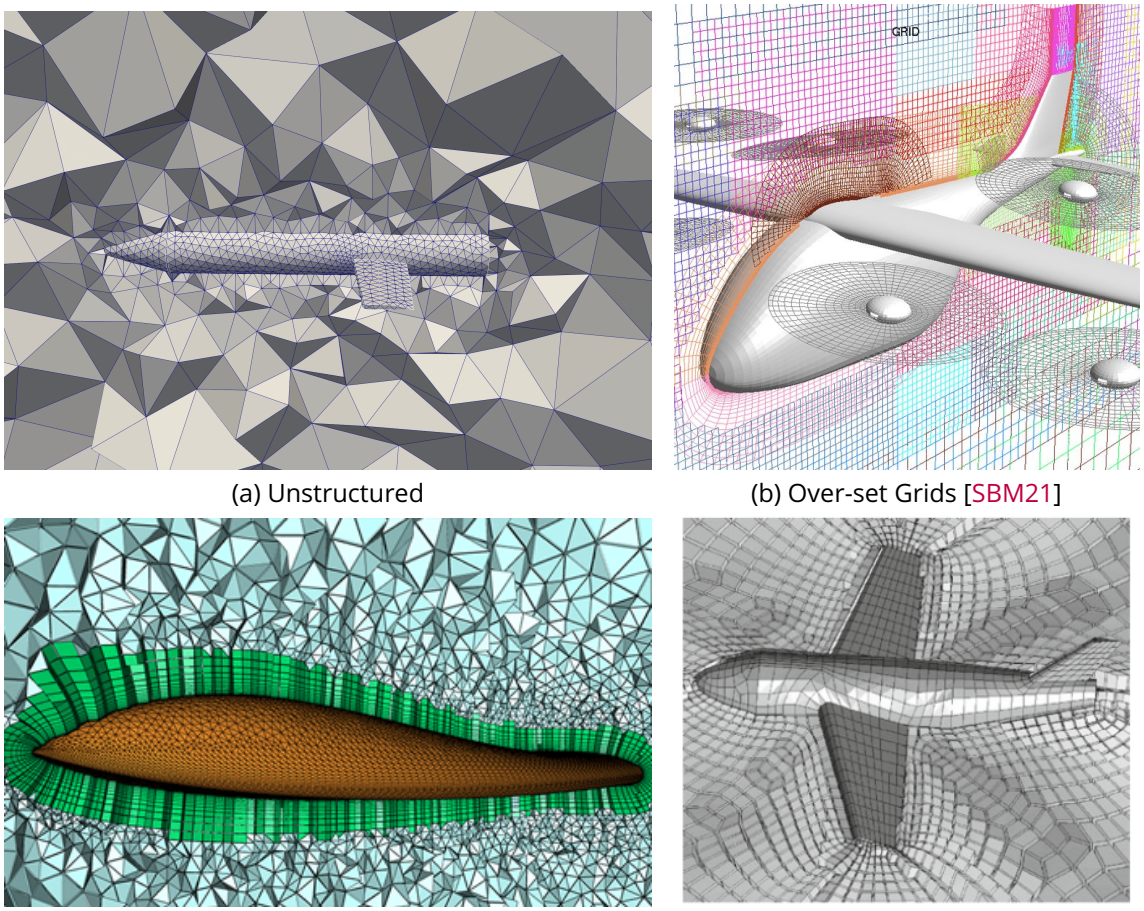
Mesh generation is a critical component of a computational physics-based analysis process. The mesh used for a simulation has a considerable impact on the quality of the solution, the stability,



(a) Unstructured (b) Structured (c) Unstructured with mixed elements (d) Immersed-body

Figure 1.19: Body-fitted meshes (a), (b), and (c), and immersed-body mesh (d) in Computational Fluid Dynamics [SD13].

and the resources expended to complete the simulations. This work considers the specific field of Computational Fluid Dynamics (CFD) and, more precisely, supersonic and hypersonic flow simulations. Some works use immersed-body meshes (see Fig. 1.19.d) to reduce the meshing process. An immersed-body mesh is a regular grid intersecting the vehicle surface. In our case, we expect to generate a body-fitted mesh (see Fig. 1.19.a, b, and c), which means a mesh that follows the boundary surfaces.



(a) Unstructured (b) Over-set Grids [SBM21]

(c) Hybrid [cad] (d) Hexahedral [RG11, RGRS12]

Figure 1.20: Examples of classical mesh topologies used in Computational Fluid Dynamics.

According to Chawner et al. [CDT16], multi-block structured meshes provide the most accurate solutions for CFD. This is among the most popular meshing techniques for flow simulation [ATS17]. However, generating such meshes is very challenging and time-consuming for

high-skilled engineers who can spend weeks or months generating the adequate mesh using complex interactive tools. It is considered one of the most time-consuming steps in the CFD process [CDT16, Tho12].

Due to the complexity of supersonic and hypersonic flow simulations, the grid density required to obtain the resolution of flow field gradients is unknown *a priori* [Alt04]. Thereby, some researchers concentrate on using mesh adaptation during the simulation [FA05]. Currently, **unstructured meshes** (see Fig.1.20.a) 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 Computational Fluid Dynamics, solvers may be less efficient regarding memory, execution speed, and numerical convergence on this type of mesh topology [SKH+21].

Considering that multi-block structured meshes provide the most accurate solutions for Computational Fluid Dynamics [CDT16, ATS17], therefore those meshes are preferred. However, generating 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, no algorithm can generate an ideal block topology. Then, other types of meshes may be used, such as **over-set grids** [Cha09] (see Fig.1.20.b). This makes the process of mesh generation on complex multi-component geometries easier. Even if these meshes provide a solution as accurate as the structured ones, they require the use of specific solvers with complex interpolation. **Hybrid meshes** (see Fig.1.20.c) for Computational Fluid Dynamics (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 over-set meshes.

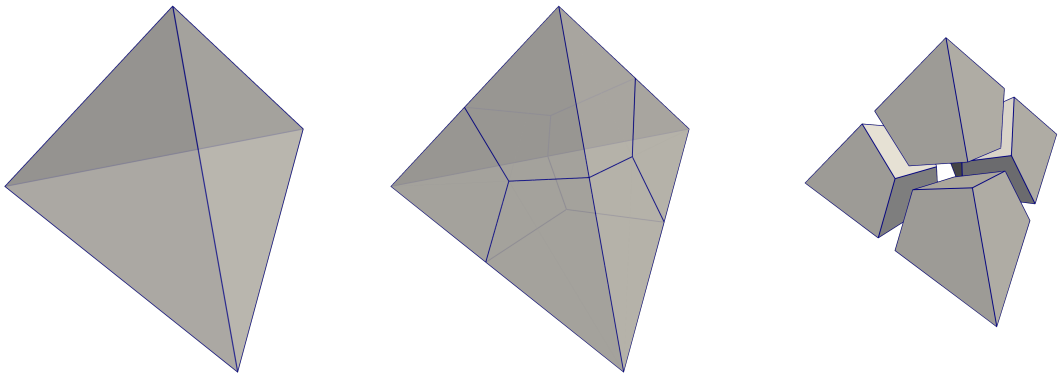
In practice, the mesh quality is strongly linked to solver algorithms. Even if the same physics is solved, each solver has its quality criteria [Tho12]. As explained by Chawner et al. [CDT16], the mesh quality criteria for Computational Fluid Dynamics simulations are always stated in a non-quantitatively 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 (i.e., no negative volume cells, no overlapping cells, no void between cells, etc.). The VERDICT library [KET+06] 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.

1.4 Hexahedral Mesh Generation: State of the Art

To find a way to generate a 2D quad block structure with an approach that extends to 3D, we can take a look at [BLP+13], [Cam17] and [Owe98, PCS+22] that provide complete surveys of existing techniques for 2D quadrangular mesh generation, quadrangular 3D mesh generation of surfaces and hexahedral 3D mesh generation of volumes. Considering we expect to get a block-structured mesh, polycube-based methods, frame fields, and advancing fronts seem the most relevant. In this section, we present the main concepts of several hexahedral mesh generation

algorithms in a non-exhaustive way.

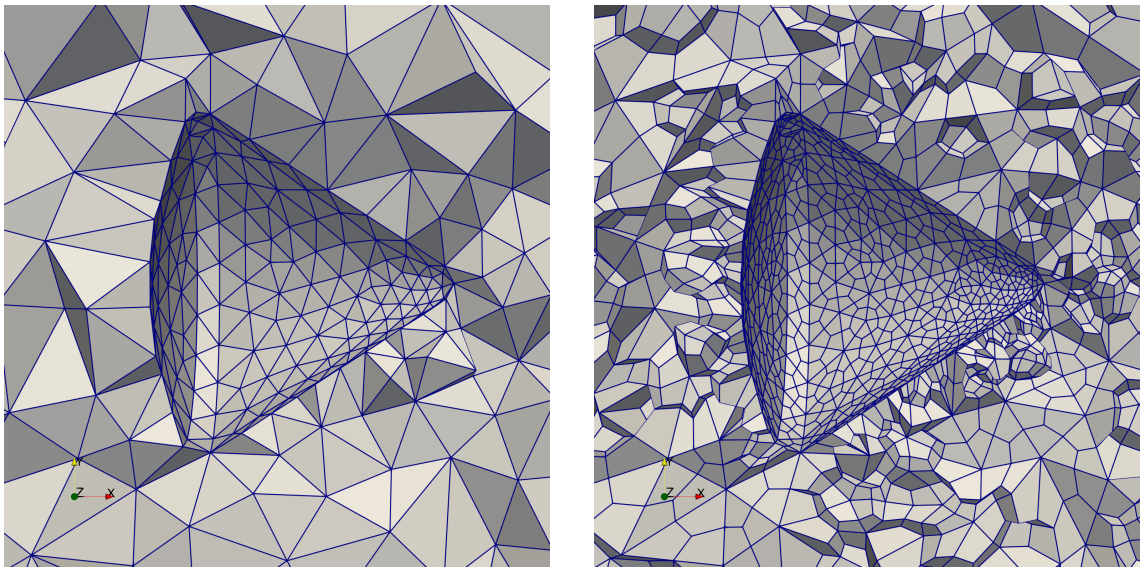
THex



(a) Tetrahedron (b) Tetrahedron (a) split into 4 hexahedra (c) Shrink view of the 4 different hexahedra (b)

Figure 1.21: THex method: how to split a tetrahedron into 4 hexahedra.

The most naive way to generate a full hexahedral mesh is to start from a tetrahedral mesh of the physical domain and split every tetrahedron of this mesh into 4 different hexahedra (see Fig. 1.21). However, this solution would lead to a poor-quality mesh unsuitable for numerical simulation (see Fig. 1.22).



(a) Tetrahedral mesh (b) Hexahedral mesh

Figure 1.22: Example of a mesh generated using THex method.

Sweeping

Introduced in [SS96, Bla96, LG97, SCO99, MBST96], the sweeping strategy consists in, given a closed physical domain Ω where we can identify two opposite surfaces, one serving as source

and the other as target, to provide a 2D quadrangular mesh of the source surface and extrude it to the target surface (see Fig. 1.23). This method is suitable for CAD models generated by extrusion, but for more complex geometries, it may be challenging. Other research proposes first to automatically decompose the domain into different sweepable parts first [LGT01].

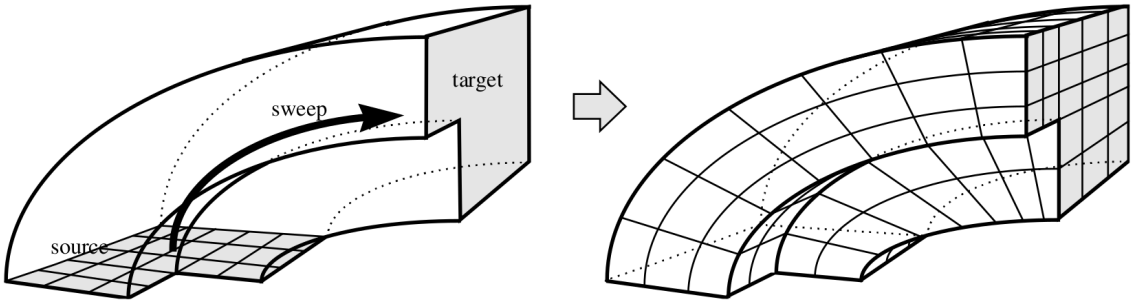


Figure 1.23: Sweeping [Owe98].

Overlay Grids

First introduced by SCHNEIDERS [Sch96], then studied in many works [Sch97, SLK04, ZB05, Mar09, She09, ISS09, HQZ13, GSP19, LPC21, PLC+21], the main principle of **overlay grids methods** is to immerse the geometry into a regular grid, to eliminate the cells out of the geometry, and to apply specific operations on the hexahedra, which intersect the boundaries of the geometry (see Fig. 1.24). This category of method leads to a very regular hexahedral mesh in the domain, with very poor quality hexahedral cells near the boundaries of the geometry.

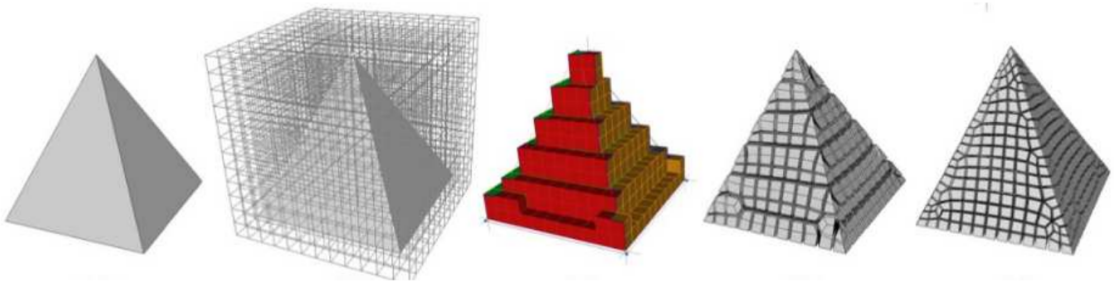
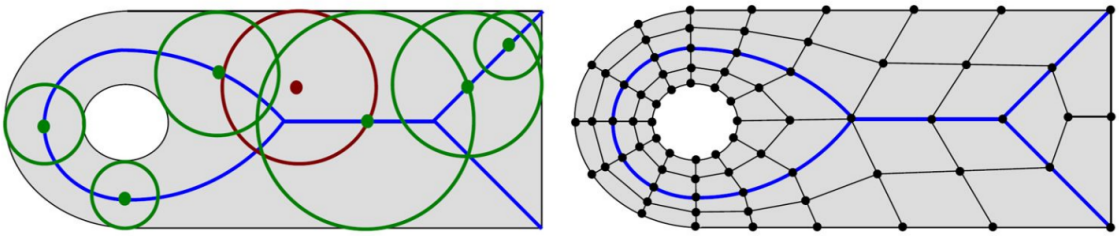


Figure 1.24: Overlay grid method [Kow13]

Medial Axis

This approach uses the medial object to extract a block structure [PAS95, PA97]. The medial object is defined on a domain Ω as the set of points $p \in \Omega$ such as it exists a sphere centered on point p that touches at least two distinct points of the boundary of the domain $\partial\Omega$. This set of points forms a "*skeleton*" of Ω and can be used to subdivide Ω into several sub-domains (see Figure 1.25.a). Then, those sub-domains are subdivided to form the mesh [PA97, Qua16] (see Figure 1.25.b).

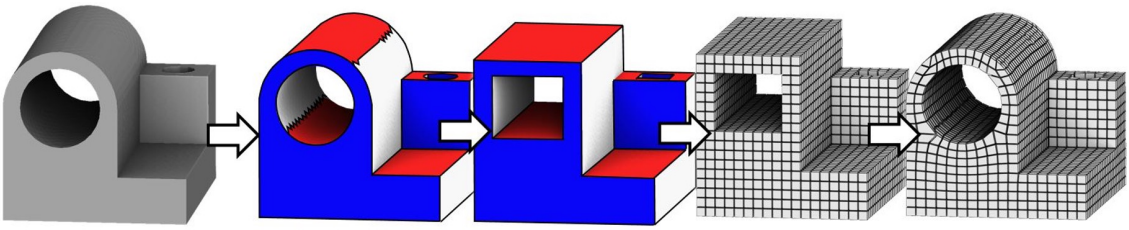


(a) (b)

Figure 1.25: Medial axis method [WL16].

Polycube

Polycube methods were first used in computer graphics for seamless texturing of triangulated surfaces [THCM04].



(a) (b) (c) (d) (e)

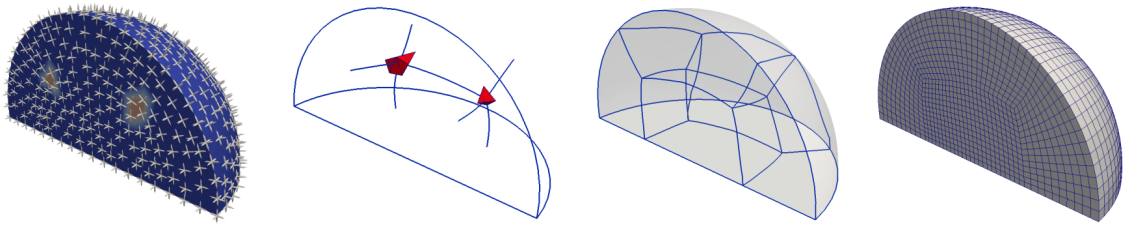
Figure 1.26: Polycube based approach [Pro22].

A polycube can be seen as a heap of cubes. The polycube method relies on four main steps. Starting from a domain Ω (see Fig. 1.26.a), a first triangular mesh T_Ω of the surface is generated. This mesh is colored according to 6 different colors, corresponding to the 6 principal axis $(\pm\overrightarrow{X}, \pm\overrightarrow{Y}, \pm\overrightarrow{Z})$ in 3D (see Fig. 1.26.b). A main axis must be assigned to each T_Ω triangle to do so. A set of adjacent triangles sharing the same color will correspond to a polygonal face of the polycube. This colored mesh is then deformed (see Fig. 1.26.c), and then a grid is extracted (see Fig. 1.26.d). Finally, the inverse transformation is applied on the grid to obtain the final mesh of Figure 1.26.e.

Many techniques [GSZ11, LVS+13, HZ16, HJS+14, FXBH16, FBL16] improved first results. However, the orientation sensitivity and the simple structure of a final coarse polycube do not fit our requirements.

Frame Fields

For several years, **frame fields** methods [KLF16, RSL16, GJTP17, LZC+18, PBS20] have offered a promising solution for quadrangular and hexahedral mesh generation. They are computed by solving the continuous relaxation problem of an integer-grid map with internal singularities (that overcome some limitations of polycubes). The majority of frame field methods have three major steps: first they create and optimize a boundary-aligned frame field (see Fig. 1.27.a); then they generate an integer-grid map, which is aligned with previously defined frame field [NRP11b, NRP11a]; and finally, they extract integer isolines (in 2D) or isosurfaces (in 3D) (see Fig. 1.27.c) to form an explicit block-structured mesh [LBK16] (see Fig. 1.27.d).



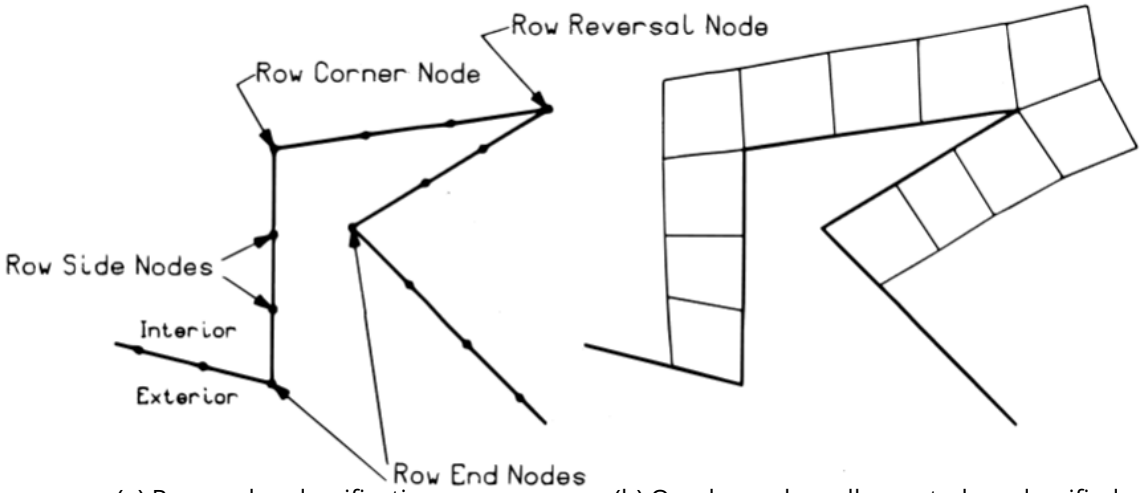
(a) Frame field (b) Singularities (c) Block structure (d) Mesh

Figure 1.27: Frame fields approach [Cal22].

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.

Advancing Fronts

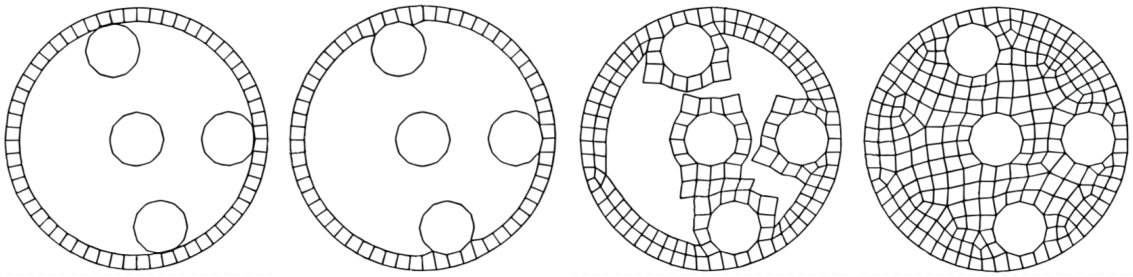
The **paving** method was introduced by Blacker et al. [BS91] to generate unstructured 2D quadrangular meshes of arbitrary domains. They start from pre-meshed boundaries and create rows of elements one at a time, following some geometrical rules. It is decomposed into several main steps. First, they select a row (which is a sequence of adjacent nodes and edges), classify the row nodes according to the angle between the two adjacent edges (see Fig. 1.28.a), and build quadrangular cells according to this classification (see Fig. 1.28.b). Then, they recompute a new row and perform the same operations again. In Figure 1.28.a, we see that a row node can take four different values during the classification according to the angle: side, end, corner, or reversal. A node classified as "side" will create one new node (see Fig. 1.28.b). A node classified as "corner" will create three new nodes, an "end" node will not create any node, and a "reversal" node will create five new nodes. It can also be seen in terms of the creation of quadrangular cells. One quadrangular cell is inserted on a row node corner, while two are inserted on a reversal node. On the other hand, two quadrangular cells are fused in one on an end node.



(a) Row nodes classification (b) Quadrangular cells created on classified row (a)

Figure 1.28: Paving row nodes classification [BS91].

In practice, they may perform those topological operations during the row creation due to the geometrical shape of the row or after the row creation to improve the mesh quality before applying a specific smoothing algorithm. They seam quadrangular cells in the domain during the process to avoid overlapping rows, and they finally mesh the remaining voids in the domain using patterns of elements based on the number of edges that surround the remaining void. An example of a complete paving sequence is presented in Figure 1.29.



(a) Intersections with inner holes (b) Cells seamed to holes (c) Second paved rows (d) Final mesh

Figure 1.29: Example of full paving sequence [BS91].

The **plastering** algorithm [SCB92, Can92, Bla96, BM93] was first introduced as the 3D extension of the paving algorithm to generate hexahedral mesh. The main principle of this approach is to start from the pre-meshed boundaries of the domain and to build cells one by one starting from the boundaries (see Fig. 1.30) until a remaining void is meshed using hexahedra patterns. One main issue is dealing with front collisions during the advancing process. Owen et al. [OSCS99] proposed the **q-morph** 2D approach based on the transformation of a triangular mesh over the domain to create quadrangular elements. They extended it to **h-morph** in 3D [OS00] using an hybrid mesh made of tetrahedra and hexahedra during the process. If it has been demonstrated that it is possible to provide a full hexahedral mesh of a remaining void under the constraint to have an even number of quadrangular cells on the boundary [Mit96], it may result in a very poor quality mesh, not usable for numerical simulation.

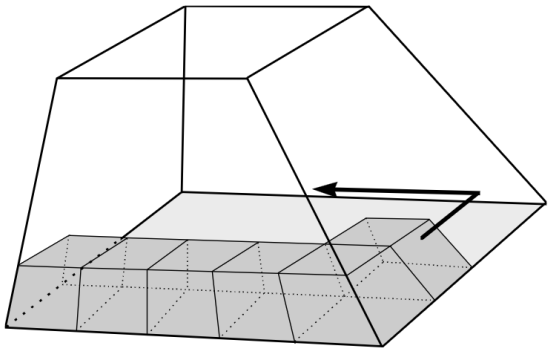
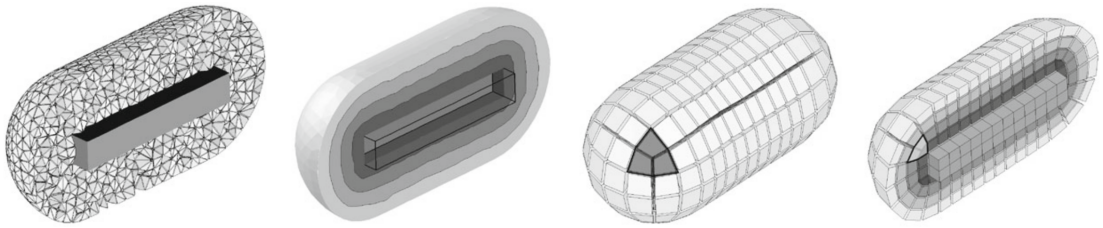


Figure 1.30: Plastering [Owe98].

Other works propose an unconstrained version of plastering, starting from non-pre-meshed boundaries, but directly from the geometry [SOB05, SKOB06, SKO+10]. Advancing front methods allow to keep control on mesh quality and size near the boundaries, and provide high quality around those areas. However, those methods may generate internal voids that can not be meshed with hexahedra or full hexahedral meshes of poor quality. Usually, a hex-dominant

mesh is extracted (i.e., a mesh mainly composed of hexahedral cells with a low proportion of pyramids and tetrahedral cells).

The receding front approach, proposed by Ruiz-Gironés et al. [RGRS12, RG11] is an advancing front approach that aims to generate an hexahedral mesh of an outer domain around a geometry. They start from the pre-meshed inner geometry boundary and relax constraints while advancing towards an outer smoothed non-meshed boundary (see Fig. 1.31).



(a) (b) (c) (d)

Figure 1.31: Receding front [RG11, RGRS12]. A tetrahedral mesh of the domain is generated (a), on which three different distance fields are computed (b). Using those distance fields, a hexahedral mesh (c,d) is generated layer by layer starting from the pre-meshed inner boundary.

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 by Ruiz-Gironés et al. [RG11, RGRS12] where they use an advancing-front approach to mesh such configurations in 3D. Such an algorithm, like the paving algorithm in 2D [BS91], is relevant for our purpose. For our work, we do not consider the paving method as being relevant. Indeed, it starts from a pre-meshed boundary, while 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 our way of 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.

Summary

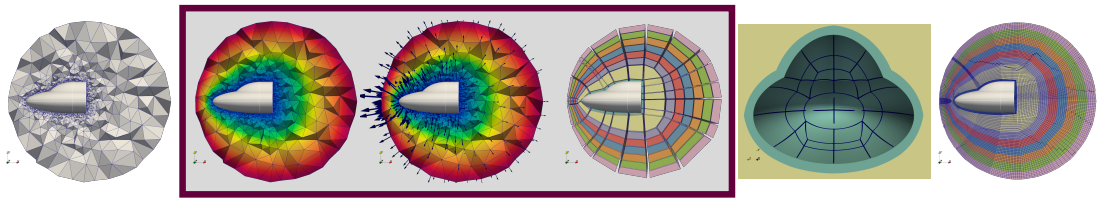
In this chapter, we introduced the notations and definitions necessary to understand the following chapters. We gave typical mesh definitions, exhibited examples of different topologies, and reviewed techniques widely used in mesh generation and adaptation. We provided information about the computing representation of a mesh and a physical domain, which is necessary for understanding some meshing method choices. We went through some notions related to Computational Fluid Dynamics associated with the context of this work. We discussed mesh quality in terms of purely geometrical and simulation-related quality. The last section was dedicated to hexahedral mesh generation state-of-the-art. Several methods were presented, focusing eventually on advancing front methods because our work belongs to this category of methods.

Chapter 2

ADVANCING FRONT FOR LINEAR BLOCK STRUCTURE GENERATION

In this chapter, the algorithm to generate the linear block structure is presented. This algorithm relies on the previous work of Ruiz-Gironés et al. [RG11, RGRS12] and is adapted here to specific constraints due to our application to hypersonic fluid mechanic. The block structure is built starting from the vehicle wall by extruding layers of hexahedral blocks. This process is driven by different quantities computed on the domain. In the first part, the computation of those fields is described. Then the process to extrude one block layer is introduced. Once the general process of building one block layer is explained, we will deal with the conflict management process on a layer. This process consists of a list of operations performed on a layer in order to add or delete blocks to improve the geometric quality of the layer blocks. If the algorithm proposed in this chapter generalizes to 2D and 3D, it differs for the conflict management stage. At the end of this chapter, we will know how to generate the linear hexahedral block structure, and the perspectives will be explained.

Contents



(a) (b) (c) (d) (e) (f)

Figure 2.1: Principal steps of our approach illustrated in 3D. In this chapter, we deal with the linear blocking generation by an extrusion of layers, corresponding to steps (b), (c) and (d).

We want to generate a block structured mesh around a flying vehicle, with particular care to the mesh cell sizes and directions close to the vehicle wall. The input is a first physical domain discretization using simplices (see Fig. 2.1.a). In other words, this input is a triangular (or tetrahedral in 3D) mesh $\mathcal{M}_T$ of the fluid domain Ω to mesh, around the vehicle of interest. This mesh can be considered as a **background mesh** as it defines our physical domain. The vehicle wall block discretization is also given as an input, as we want a strong control on it. The far-field boundary is not pre-meshed, which provides flexibility in the chosen approach.

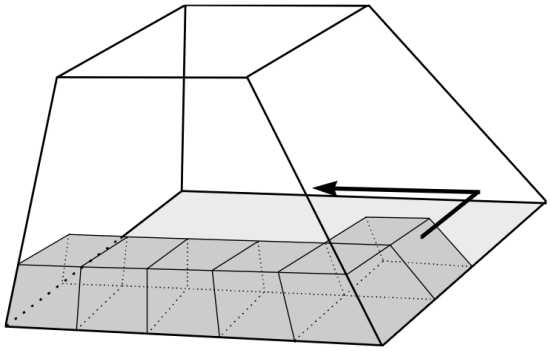


Figure 2.2: Plastering process [Owe98].

Due to our constraints presented above, the most relevant methods to generate the block structure belong to the category of advancing-front algorithms. Originally, the main principle of advancing-front methods is to start from a pre-meshed boundary and incrementally insert quadrangular or hexahedral cells into the domain. The method was first proposed for 2D quadrangular mesh generation through paving [BS91], then extended to 3D hexahedral mesh generation with the algorithm called plastering [BM93]. This process fills the domain step by step, until a remaining void is finally meshed, using specific quadrangular or hexahedral patterns. Sometimes, in 3D, this remaining internal void is too complex to be meshed with all-hexahedral cells. The advancing-front methods starting from a pre-meshed boundary are called constrained. As it can be over-constrained for hexahedral mesh generation, unscontrained methods [SOB05, SKO+10] propose to relax the problem by generating the mesh starting from the geometry, and not a first discretization of the boundaries. In the state of the art, considering both constrained and unconstrained approaches, we identify three different categories of advancing-front methods. First one relies on advancing the structure element by element (see Fig. 2.2). A second category of methods aims to generate the mesh by portions of elements (such as unconstrained plastering where they anticipate layers before creating them) [SOB05, SKO+10]. The last category regroups methods to generate the mesh by extrusion of complete layers of elements [RG11, RGRS12]. Some advancing-front methods are based on the use of patterns of

cells in order to improve the geometrical quality of the created cells (see Fig. 2.3), or to avoid potential conflicts due to contraction of a physical domain.

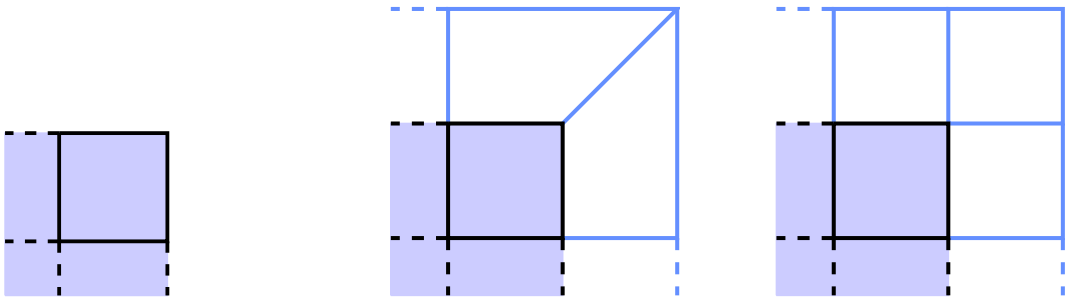


Figure 2.3: Example of conflict management in the paving method [BS91] in case of domain expansion. In order to improve the geometric quality of the new blue cells ( ) of (b), we can insert another cell (c).

Among the methods of the last category, we have a particular interest in the receding-front method proposed by Ruis-Gironés et al. [RG11, RGRS12], which has several features. This method is used in practice to mesh outer space around vehicles (e.g., Computational Fluid Dynamics (CFD) for instance) and is limited to geometric domains with smooth outer boundaries (i.e., no sharp features). This method can be considered as a mix between constrained and unconstrained methods, as it starts from a pre-meshed boundary (i.e., the discretization of the vehicle wall) to advance towards the non pre-meshed outer boundary. It relaxes constraints while making possible to keep a high control on the mesh quality around the vehicle. The method proposed and summarized in Algorithm 1 is decomposed as follow. In [RG11, RGRS12], they first generate and combine distance fields by solving the Eikonal equation on a background simplicial mesh $\mathcal{M}_T$, and use the combined resulting field to expand the pre-meshed vehicle boundary in the volume. They know *a priori* the number $N_{\mathcal{L}}$ of layers to build, and at each step of the algorithm, they generate a full layer of hexahedral cells on the precedent one (see Algo. 1, l. 4). They avoid the potential problem of separation of a front by the use of the distance field to control the layers expansion. On a layer, they perform a specific conflict management based on a similar principle as presented in Figure 2.3. They apply some set of hexahedra at particular location to improve the hexahedra quality over a layer.

Algorithm 1 Receding-Front Method [RG11, RGRS12]

Require: Background mesh $\mathcal{M}_T$, first front $\mathcal{F}_0$, combined distance field d, distance field from the vehicle d_V
Ensure: Hexahedral mesh $\mathcal{M}_H$
1: ScalarField $d_V \leftarrow$ solveEikonal($\mathcal{M}_T$, $\partial\Omega_V$)
2: ScalarField $d_{FF} \leftarrow$ solveEikonal($\mathcal{M}_T$, $\partial\Omega_{FF}$)
3: ScalarField $d \leftarrow$ combineDistanceFields(d_V, d_{FF})
4: $\Delta d \leftarrow 1/N_{\mathcal{L}}$
5: **for all** $i \in 1, ..., N_{\mathcal{L}}$ **do**
6: $\quad \mathcal{L}_i \leftarrow$ computeLayer($\mathcal{F}_{i-1}$, d, $i\Delta d$)
7: **end for**

Due to our constraint and the capabilities of the method, we decide to base our work on

the previous work of Ruis-Gironés et al. [RG11, RGRS12]. We highlight several minor differences:

1. First of all, we use this method to generate a block structure which is coarser and much constrained than a mesh. Our pre-meshed block surface can be anisotropic (i.e., with important aspect ratio between quadrangular cells) while the pre-meshed surface looks relatively regular in [RG11, RGRS12].
2. We do not compute the distance fields (see Algo. 1, l. 1-2) the same way.
3. We introduce, in addition to the 3 different distance fields (see Fig. 2.1.b), a vector field (see Fig. 2.1.c) that will lead the directions for the extrusion of the blocks. This vector field allows us to control the block direction extrusion. Through this vector field, we can impose the block alignment to the vehicle for the first block layer (i.e., to provide wall-orthogonal blocks), and to the flow Angle of Attack (AoA) in other parts of the domain.
4. We generate the very first block layer with particular conditions, not as the other layers.
5. The way we manage conflicts on a layer slightly differs.

Algorithm 2 Block Structure Generation Layer by Layer

Require: Background mesh $\mathcal{M}_T$, first front $\mathcal{F}_0$, boundary layer thickness δ_{BL}
Ensure: Block structure $\mathbf{B}$
1: ScalarField $d_V \leftarrow$ EuclideanDistance($\mathcal{M}_T$, $\partial\Omega_V$)
2: ScalarField $d_{FF} \leftarrow$ EuclideanDistance($\mathcal{M}_T$, $\partial\Omega_{FF}$)
3: ScalarField $d \leftarrow$ combineDistanceFields(d_V, d_{FF})
4: VectorField $\overrightarrow{v} \leftarrow$ computeVectorField($\mathcal{M}_T$, d_V, d)
5: $\mathcal{L}_1 \leftarrow$ compute1stLayer($\mathcal{F}_0$, d_V, $\overrightarrow{v}$, δ_{BL})
6: $\Delta d \leftarrow 1/N_{\mathcal{L}}$
7: **for all** $i \in 2, ..., N_{\mathcal{L}}$ **do**
8: $\quad \mathcal{L}_i \leftarrow$ computeLayer($\mathcal{F}_{i-1}$, d, $\overrightarrow{v}$, $i\Delta d$)
9: **end for**

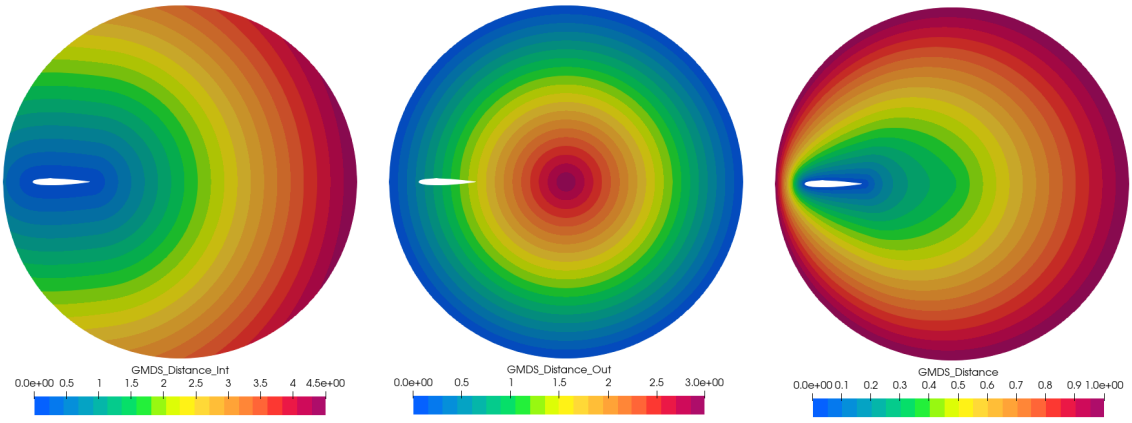
If we look back at the full pipeline of our method presented in Figure 2.1, this chapter focuses on steps (b), (c), and (d). The principle steps of our process are summarized in Algorithm 2. In this chapter, we re-introduce the receding-front method proposed by Ruis-Gironés et al. [RG11, RGRS12], and explain how we adapt it to our constraints to generate the hexahedral block structure. We first designed a suitable 2D pipeline, then extend it to the 3D case. For this reason, the chapter presents the method for the general nD case and the outline is decomposed as follows. In the first Section 2.1, we explain how the distance and vector fields are computed (see respectively Fig. 2.1.b and c) on the input background mesh $\mathcal{M}_T$ (see Fig. 2.1.a). The second Section 2.2 is meant to explain how we build a regular block layer, and the full block structure by extruding some layers, starting from the vehicle. A special treatment is performed in order to deal with conflicts due to expansion or contraction of the domain on a layer. To do so, we first examine the front on which we want to build a layer, then we decide where to create block cells. We know *a priori* which blocks will be created in

the layer, before starting its construction. The way conflicts are handled on a layer differs in 2D and 3D. For this reason, this step is presented separately, respectively in Section 2.3 for 2D and Section 2.4 for 3D. Obtained block structures generated around three different geometries are presented in Section 2.5. Finally, some identified perspectives related to this part of the work are addressed in Section 2.6.

2.1 Fields Computation

Distance and vector fields are core components of the approach to drive the block layer extrusion. The idea, inspired by [RG11, RGRS12] is to mix several fields to know how to insert block corners during the layer creation process. In practice, those fields are discrete and defined at the nodes of the background mesh $\mathcal{M}_T$. In this section, we present the process to compute those fields.

2.1.1 Distance Fields



(a) Euclidean distance to the vehicle d_V

(b) Euclidean distance to the far-field d_{FF}

(c) Combined distance field d

Figure 2.4: Distance fields computed around the 2D NACA 0012 airfoil geometry (see Appx. B).

As in [RG11, RGRS12], we compute a distance field d by merging two distance fields: the first one encodes the distance from the vehicle boundary $\partial\Omega_V$, the second one encodes the distance from the far-field boundary $\partial\Omega_{FF}$. In the previous work of Ruiz-Gironés et al. [RG11, RGRS12], the Eikonal Equation given by

$$\begin{cases} ||\nabla d|| &= f \text{ in } \Omega, \\ d_{|\mathcal{F}} &= 0, \end{cases} \tag{2.1}$$

is solved to compute the distance fields on the domain. In this equation, $\Omega \subset \mathcal{R}^n$ is the physical domain, f is a known function (considered as constant and equal to 1), $||\cdot||$ is the Euclidean norm, $\mathcal{F}$ is the front and d is the distance to this front. The problem is solved on the background mesh $\mathcal{M}_T$.

The first field d_V is the distance field from the vehicle boundary ($\partial\Omega_V$) :

$$\begin{cases} ||\nabla d_V|| &= 1 \text{ in } \Omega \\ d_{V|\partial\Omega_V} &= 0. \end{cases} \tag{2.2}$$

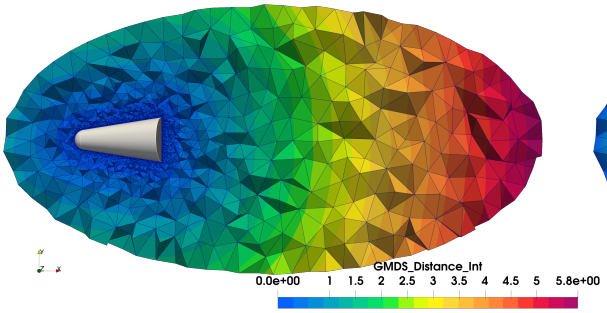
The second field d_{FF} is the distance from the far-field boundary ($\partial\Omega_{FF}$):

$$\begin{cases} ||\nabla d_{FF}|| & = 1 \text{ in } \Omega \\ d_{FF|\partial\Omega_{FF}} & = 0. \end{cases} \tag{2.3}$$

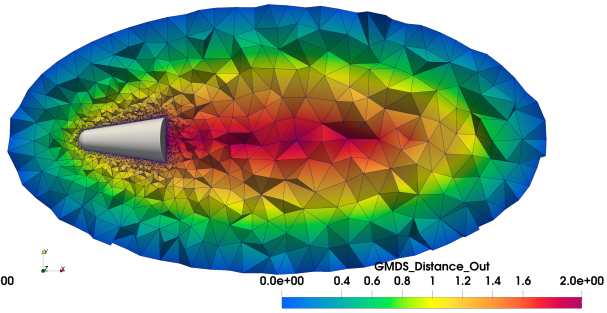
In this work, the fields d_V (see Fig. 2.4.a and Fig. 2.5.a) and d_{FF} (see Fig. 2.4.b and Fig. 2.5.b) are simply computed as the Euclidean distance of each node of the background mesh $\mathcal{M}_T$, respectively to the vehicle and the far-field boundaries. The main reason is computational cost. We do not work with a Computer Aided Design (CAD) representation of the physical domain but through a discretization given by the background mesh $\mathcal{M}_T$. For this reason, if the background mesh is not thin enough around the vehicle wall, we have a terrible representation of its boundary. So, we need many cells around the vehicle wall, which increases the computational cost of the method proposed in [RG11, RGRS12]. The two methods are not equivalent in some specific cases (e.g., in the case of a multi-component object, if we need to compute the distance to only one component), and it is discussed in the perspectives section at the end of this chapter. Due to our specific constraint, the fields computed using the Euclidean distance ensure the same properties.

The third and last field d represented in Figure 2.4.c and Figure 2.5.c is computed the same way as in [RG11, RGRS12], this field is a combination of the two fields d_V and d_{FF}:

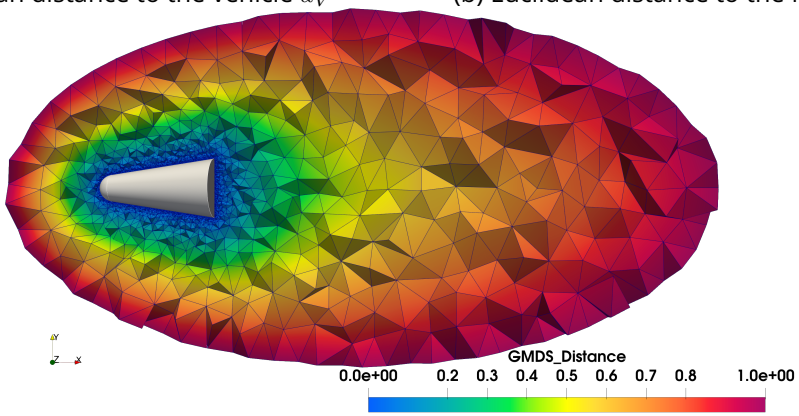
$$d = \frac{d_V}{d_V + d_{FF}}. \tag{2.4}$$



(a) Euclidean distance to the vehicle d_V



(b) Euclidean distance to the far-field d_{FF}



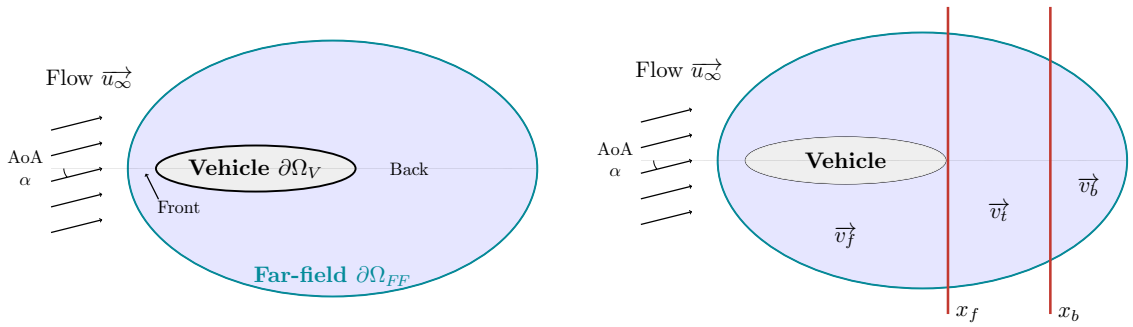
(c) Combined distance field d

Figure 2.5: Distance fields computed around the 3D RAM-C II geometry (see Appx. B).

This field d verifies $0 \leq d(x) \leq 1$, $\forall x \in \Omega$ and the boundary conditions $d_{|\partial\Omega_V} = 0$ and $d_{|\partial\Omega_{FF}} = 1$. It ensures us to reach the far-field for the same layer during the extrusion, it

prevents the front to divide. Let us note that the combined distance field d, and the distance field to the vehicle d_V are used in the extrusion algorithm, while d_{FF} is only computed to build d.

2.1.2 Vector Fields



(a) Different areas of domain to mesh ( ) (b) Vector field $\overrightarrow{v}$ computed piecewise

Figure 2.6: The vector field used to lead the extrusion direction over the domain is computed piecewise on the physical domain to mesh ( ). Physical limits ( ) used to compute the resulting vector field $\overrightarrow{v}$ are set by user parameters. $\overrightarrow{v_t}$ is computed as a linear transition between $\overrightarrow{v_f}$ and $\overrightarrow{v_b}$ on $[x_f, x_b]$.

In addition to the three previously defined distance fields d, d_V, and d_{FF} introduced in the work of Ruiz-Gironés et al. [RG11, RGRS12], we use a vector field computed on the background mesh $\mathcal{M}_T$. Let us remind that we expect in this chapter to generate a block structure starting from the vehicle wall ( ) of Figure. 2.6.a to the far-field ( ) of the physical domain ( ). In our approach, this vector field drives the blocks extrusion direction. The resulting vector field aims to ensure block orthogonality close to the vehicle wall ( ) of Figure 2.6.a, regular blocks in the front vehicle part (see Fig. 2.6.a), while trying to align the blocks with some privileged directions (e.g., the flow Angle of Attack α) behind the vehicle (see back area of Fig. 2.6.a). In this subsection, we compute and mix different vector fields to respect the conditions listed before. Each vector field is computed at nodes of the background mesh $\mathcal{M}_T$.

Far-field Flow Direction

In supersonic flow simulation, we want to care particularly about the front of the vehicle, its near boundary, and the global mesh direction in the vehicle's back area. We drive the mesh behavior in the back area with the Angle of Attack (AoA) α, which is flow-related information. To this purpose, we define the vector field $\overrightarrow{u_\infty}$ as constant on the domain Ω and equal to the far-field flow direction, $\overrightarrow{u_\infty} = \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}$. In practice, this vector field is used as the back vector field $\overrightarrow{v_b}$ of $\overrightarrow{v}$ (see Fig. 2.6.b).

Distance Fields Gradients

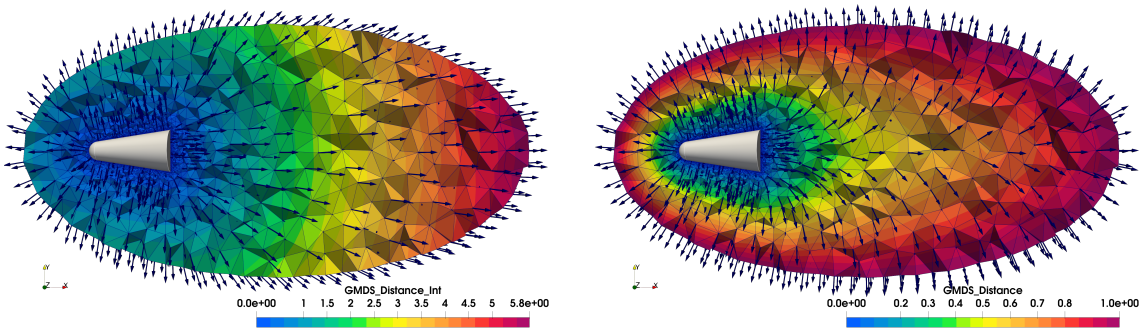
For the front of the vehicle, two possible options are considered based on the previously computed distance fields. The first vector field we use is the gradient of the distance field d_V, noted ∇d_V, and the second one is the gradient of the mixed distance field d, noted ∇d. We compute these vector fields at the vertices of $\mathcal{M}_T$ with the Least Squares fit of Directional Derivatives (LSDD) method [MLP19]. Let us consider n_i a node of $\mathcal{M}_T$, and $\{n_0, ..., n_{k_i}\}$ the set of nodes adjacent[1] to n_i. The first order Taylor's expansion of any function f is given by

$$f(n_j) - f(n_i) \approx \nabla f \cdot (n_i - n_j), \forall j \in [0, ..., k_i]. \tag{2.5}$$

Wa can use the Equation (2.5) to compute the gradient of f by solving a linear system. Most of the time, k_i is higher than the problem's dimension. We consider the $k_i \times d$ matrix A_i of the $(n_j - n_i)$ coefficients. We solve the linear system of size $d \times d$:

$$A_i^T W_i A_i \nabla f(v_i) = A_i^T W_i D_i, \tag{2.6}$$

where D_i is the vector of length k_i representing the coefficients $f(n_j) - f(n_i)$, W_i is a $k_i \times k_i$ diagonal matrix of weights, with $W_i(j,j) = \frac{1}{{d_{ij}}^2}$, and d_{ij} is the Euclidean distance between the nodes n_i and n_j (i.e., the size of the edge shared by n_i and n_j).



(a) Distance field from the vehicle gradient ∇d_V

(b) Combined distance field gradient ∇d

Figure 2.7: The gradient of two different distance fields of interest around RAM-C II 3D geometry (see Appx. B). The corresponding vector field is plotted using dark blue vectors (⟶). The background color map represents the distance field used to compute the vector field.

A dedicated pre-process may be required on the background mesh $\mathcal{M}_T$ for n-cells that have all their nodes located on the vehicle wall. The value of the distance field will be equal to 0 on all the nodes of this cell. As a result, the gradient of the distance field will be null over this cell. So if a block corner of the first front has to go through this cell during the extrusion algorithm, it can stop its trajectory path.

In the algorithm, we can decide to compute the gradient of the Euclidean distance field to the vehicle d_V (see Fig. 2.7.a), or the combined distance field d (see Fig. 2.7.b). In the first case, the vector field is orthogonal to the vehicle. In practice, the front vector field $\overrightarrow{v_f}$ (see Fig. 2.6.b) is chosen between ∇d_V and ∇d.

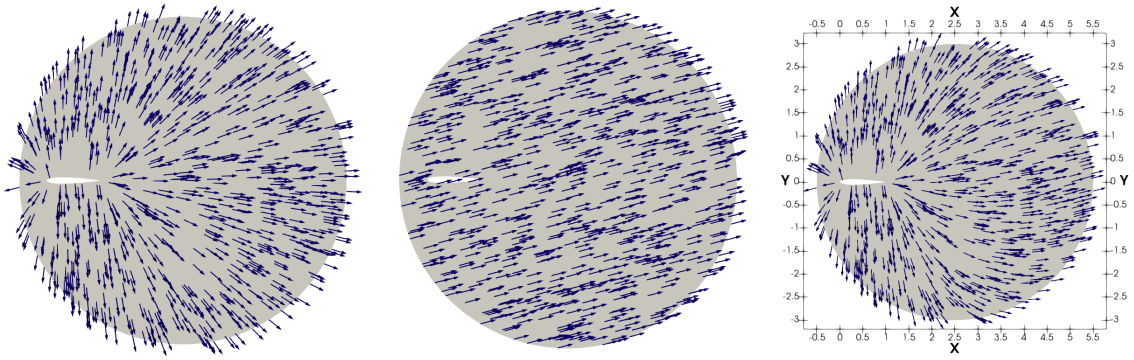
[1] Two nodes are considered as being adjacent if they share one edge.

Combined Vector Field

In order to consider both the front and back vector fields, we eventually compute the vector field $\overrightarrow{v}$ as a linear combination of those two vector fields in a transition area (see Fig. 2.6.b). Two physical limits ( ) are set by the user in the physical fluid domain Ω ( ), x_f and x_b. For each node $n_i \in \mathcal{M}_T$ at position $p = \{x_i, y_i, z_i\}$, we compute the local value of the vector field $\overrightarrow{v_i}$ as

$$\begin{cases} \overrightarrow{v_i} = \overrightarrow{v_{f,i}} & \text{if } x_i < x_f, \\ \overrightarrow{v_i} = \overrightarrow{v_{b,i}} & \text{if } x_i > x_b, \\ \overrightarrow{v_i} = (1-\theta)\overrightarrow{v_{f,i}} + \theta\overrightarrow{v_{b,i}} & \text{if } x_f \leq x_i \leq x_b. \end{cases} \tag{2.7}$$

Where $\overrightarrow{v_{f,i}}$, and $\overrightarrow{v_{b,i}}$ are respectively the values of selected vector fields $\overrightarrow{v_f}$ and $\overrightarrow{v_b}$ at node n_i. A damping parameter $\theta = \frac{x_i - x_b}{x_b - x_f}$ between 0 and 1 controls the transition area. Figure 2.8 illustrates some of the different vector fields we use[2]. Note that by default, we normalize all the vector fields as we only use the field direction and not its magnitude. Due to the vector field computation, our method is sensitive to the geometry's input orientation through the background mesh $\mathcal{M}_T$.



(a) $\overrightarrow{v_f} = \nabla d_V$ (b) $\overrightarrow{v_b} = \overrightarrow{u_\infty}$ (c) $\overrightarrow{v}$

Figure 2.8: A practical example of vector fields computed around the 2D NACA 0012 geometry. The vector field (c) is a mix from ∇d_V (a), and $\overrightarrow{u_\infty}$ (b), with $x_f = 1.5$ and $x_b = 5.0$. The Angle of Attack α used in (b) is equal to 15°.

2.2 Block Layers Extrusion

At this algorithm stage, we computed the distance and vector fields necessary to lead our extrusion algorithm on our input background mesh $\mathcal{M}_T$. This sections explains how to build a regular block layer starting from the wall surface discretization.

[2] A study is performed in Appendix C in order to illustrate different vector fields generated and combined around a test case, and the impact of the different vector fields on the block structures generated. As this study presents block structure, the reader may want to move forward to the next sections before checking this Appendix.

2.2.1 Surface Geometry Block Structure

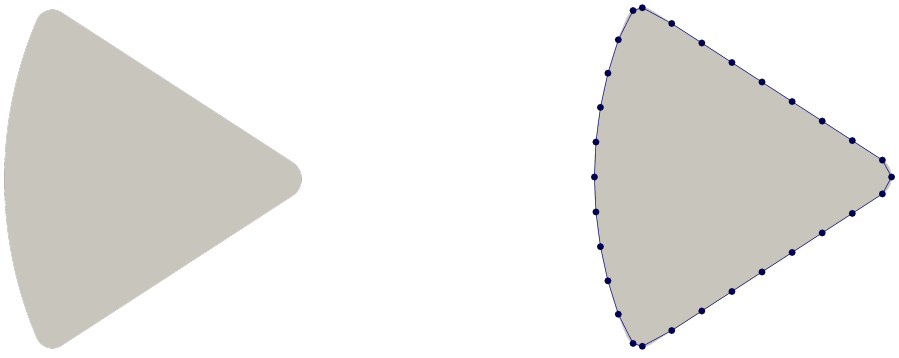
The discretization of the vehicle wall is a set of n–1-cells and is considered as the very first step of our algorithm (see Fig.2.9 and Fig. 2.10). The first block layer will be built on this very first discretization. We proposed a way to compute it in 2D in this work (see the next 2D frame). In 3D, we start from a pre-meshed vehicle wall (see the 3D frame).

2D

The wall surface block discretization is a set of block corners and block edges. To automatically discretize the vehicle geometry (corresponding to the boundary $\partial\Omega_V$), we traverse the nodes $n_0, ..., n_m$ of the triangular background mesh $\mathcal{M}_T$ located on $\partial\Omega_V$ in an ordered way. We select a node n_i if and only if it satisfies at least one of the following conditions:

- n_i is located on an extremum of the boundary profile (i.e., a node of $\partial\Omega_V$ that minimizes or maximizes the x or y coordinate).
- n_i is a geometric corner (geometric point) of $\partial\Omega_V$;
- the curvilinear distance from the previously selected node and n_i is greater than the maximal length given as an input parameter.

The starting node on $\mathcal{M}_T$ is usually an extremum of the boundary when possible, or a characteristic node. Let us note that this approach does not guarantee that the boundary edges will have the same size, or that the block discretization of the boundary will be symmetric, even though the geometry is symmetric. It is not an issue for our process since those are block edges and they will be refined later to get the final mesh. Here, the block corners are set at positions corresponding to nodes of the background mesh $\mathcal{M}_T$. The discretization of the vehicle is automatic here. We can easily imagine taking the set of block edges and corners corresponding to the discretization of the vehicle geometry as an input, in order to keep control over those elements.



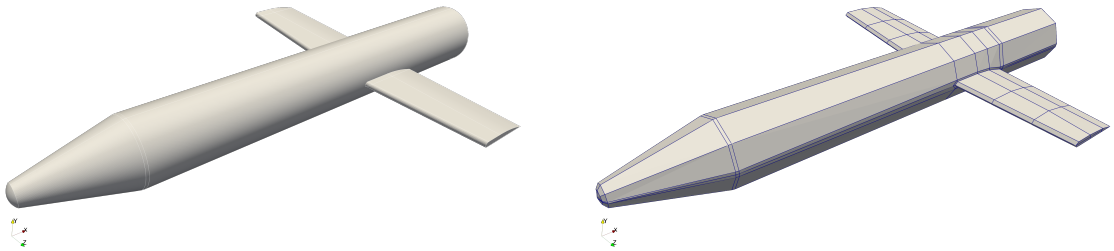
(a) 2D geometry ( ) (b) 2D vehicle block edges ( ) and block corners ( )

Figure 2.9: Automatic 2D geometry block structure (front $\mathcal{F}_0$) generation example around Apollo geometry (see Appx. B).

3D

The wall surface block discretization is a set of block corners, block edges and quadrangular block faces. The generation of a quadrangular surface blocking is an open field of research, but you can find some methods in the survey of CAMPEN [Cam17]. It is worth considering improving the algorithm by using one of these methods to generate the blocking surface. However, due to our application, it is needed to keep control over this surface blocking. Here, the vehicle surface discretization is given as an input. In this work, most of the input surface block structures were generated using **MGX**[a], an interactive software dedicated to block-structured hexahedral mesh generation. An example of input discretization is given on Figure 2.10.b.

[a] https://github.com/LIHPC-Computational-Geometry/magix3d



(a) 3D geometry ( ) (b) 3D surface blocking ( )

Figure 2.10: Surface geometry block structure (front $\mathcal{F}_0$) example.

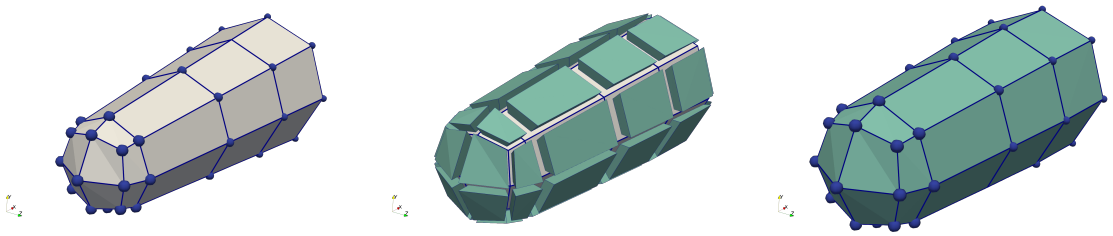
This block decomposition of the vehicle surface is considered as the first **front** $\mathcal{F}_0$ (see Def. 2.2.1) in the algorithm. As we may see in the example of a 3D front represented in Figure 2.10.b, each block edge of the front is adjacent to exactly two quadrangular blocks (n–1-cells) on this front and respect the given definition. Then, this front is a suitable input for our algorithm.

Definition 2.2.1 (**Front**). In dimension n, a **front** is defined as a set of adjacent n–1-cells (of blocks)[3] that shares some properties. Each n–2-cell of this front is adjacent to exactly two n–1-cells on this same front (see Fig. 2.10.b).

2.2.2 Generation of One Layer

At each step of the algorithm, a complete new block layer is generated on a front (see Fig. 2.11). The computation of a new layer is independent of the computation of the precedent layer. We generate a complete block layer following Algorithm 3. The algorithm aims to build a block layer on a front $\mathcal{F}_i$ and to provide a new front $\mathcal{F}_{i+1}$ that respects the same definition. To produce a block layer (n-cells), each n–1-cell of a front (see Fig. 2.11.a) is extruded to create a block. This set of blocks forms a **layer** (see Def. 2.2.2 and Fig. 2.11.b) denoted $\mathcal{L}_i$. This block layer provides us with a new front (see Fig. 2.11.c) noted $\mathcal{F}_{i+1}$ that shares the same properties as the precedent one. This way, we can build a new layer on this front, etc., until the outer boundary of the domain (far-field) is reached.

[3] Two n–1-cells are adjacent if they share at least one n–2-cell.



(a) Front (b) Layer (c) New front

Figure 2.11: A practical example of one regular block layer generation on a 3D front. Each quadrangular block of the front (a) creates one hexahedral block by extrusion of itself. This set of new hexahedral blocks ( ) represented by the exploded view in (b) forms a block layer. This block layer provides a new front (c).

Definition 2.2.2 (Layer). In dimension n, a **layer** is a set of adjacent n-cells (blocks)[4]. A layer is bounded by exactly two different fronts. For each block corner of each n-cell of the layer, this block corner belongs to one of the two fronts. For each n–1-cell of a layer, either this n–1-cell belongs to one of the two fronts, or this n–1-cell is shared by exactly two n-cells of this layer.

Algorithm 3 Build One Regular block layer on a Front

Ensure: Blocks $\mathbf{B}$ of the layer $\mathcal{L}_i$, and the set of block corners and n–1-cells of front $\mathcal{F}_{i+1}$

1: **for all** block corner $n_i^j \in \mathcal{F}_i$ **do**
2: $n_{i+1}^j \leftarrow$ computeIdealPosition(n_i^j, $d_{\mathcal{F}_{i+1}}$, d, $\mathbf{v}$)
3: **end for**
4: **for all** block n–1-cell $\in \mathcal{F}_i$ **do**
5: $\mathbf{B} \leftarrow \mathbf{B}+$ buildBlock(n–1-cell)
6: **end for**

New Front Block Corner Location

To build the layer $\mathcal{L}_{i+1}$, the block corner positions of the new front $\mathcal{F}_{i+1}$ are computed from the positions of the block corners of the precedent front $\mathcal{F}_i$. All the block corners of the front $\mathcal{F}_{i+1}$ are located at the same iso-value d_{i+1} on the distance field of interest.

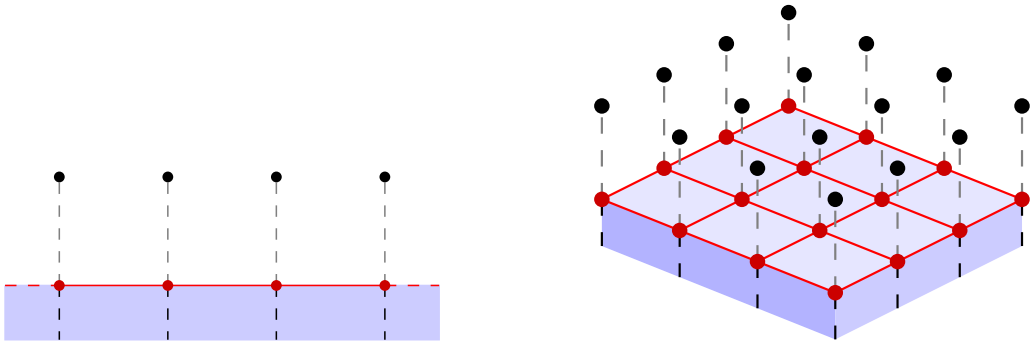
Let us consider the generation of a block layer $\mathcal{L}_{i+1}$ starting from the front $\mathcal{F}_i$. For each block corner n_i^j of $\mathcal{F}_i$, we compute the ideal location of the next block corner n_{i+1}^j, which belongs to the next front $\mathcal{F}_{i+1}$ (see Algo. 3, l. 3) by solving the advection equation

$$\frac{\partial \overrightarrow{OM}}{\partial t} = \overrightarrow{v} \tag{2.8}$$

using a 4^{th}-order Runge-Kutta method (see Eq. (2.9)):

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{2.9}$$

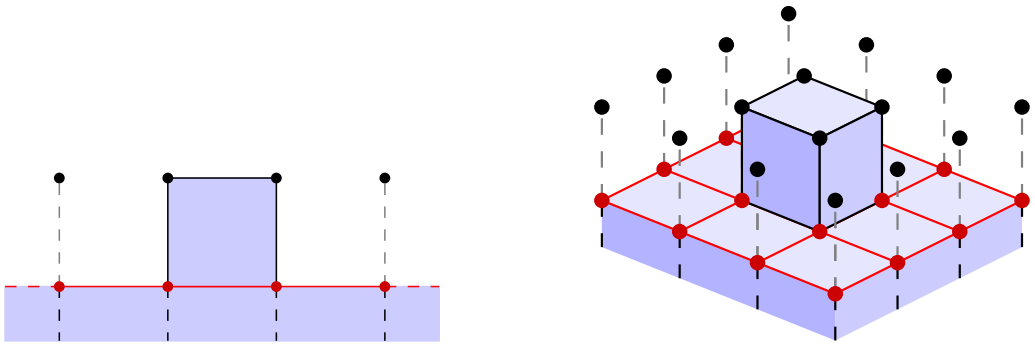
[4]Two n-cells are adjacent if they share at least one n–1-cell.



(a) 2D front (b) 3D front

Figure 2.12: Local view of a front (▬). Each block corner of the front (●) computes the ideal position of the next block corner of the next front (●).

where $t_{n+1} = t_n + h$, h is the step size, $k_1 = f(t_n, y_n)$, $k_2 = f(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2})$, $k_3 = f(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2})$, and $k_4 = f(t_n + h, y_n + hk_3)$. The origin block corner $\mathbf{O}{=}n_i^j$ is advected along the direction of the vector field $\overrightarrow{v}$ until its distance d_{n^j} in the distance field d reaches d_{i+1}. We obtain then the point $\mathbf{M}{=}n_{i+1}^j$. Unlike [RGRS12], we define the position of a new block corner by decoupling the distance to be covered, provided by the distance field d, from the direction to be followed, provided by the vector field $\overrightarrow{v}$. 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. While through the use of the combined distance field, we ensure a strong property on the computed layer: all the block corners of a front $\mathcal{F}_i$ are located at the same distance $d_{\mathcal{F}_i}$ along the input distance field. This property ensures that the front cannot separate (i.e., the layer created will always provide a new front that respects the Definition 2.2.1) and that all the block corners will reach the outer boundary at the same moment on the last layer.

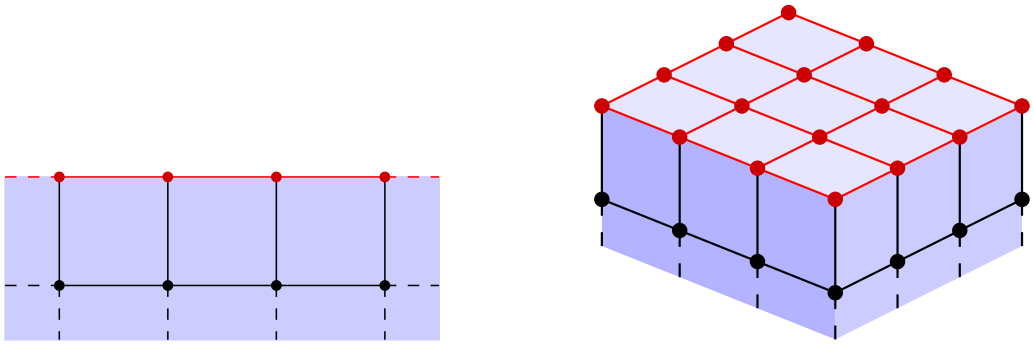


(a) One block of the new layer created on the 2D Front (b) One block of the new layer created on the 3D Front

Figure 2.13: An n-cell (block) is created on the front (▬). This n-cell belongs to the new layer. Ideally, each n–1-cell of the front creates one n-cell.

Figure 2.12 presents a schematic local view of a nD front (▬) of n–1-cells. The blocks created in the precedent layer are plotted in blue (▬). Each block corner of the front (●) computes the ideal position in the next front (●). Trajectories, computed using the advection equation, followed by the front nodes are plotted using dashed gray lines (▪▪▪).

Then, for each n–1-cell of the front, we create one block (see Fig. 2.13) according to the ideal positions computed previously. Figure 2.13 represents the extrusion of regular blocks on a nD layer. In Figure 2.13.a, a small part of a front $\mathcal{F}_{i-1}$ is plotted in red (▬), and previously



(a) 2D layer (b) 3D layer

Figure 2.14: A layer is defined as the set of n-cells (blocks) created on the n–1-cells of a front. This layer is bounded by two fronts, the first one is the one used to build the layer. The other one is the new front ( ).

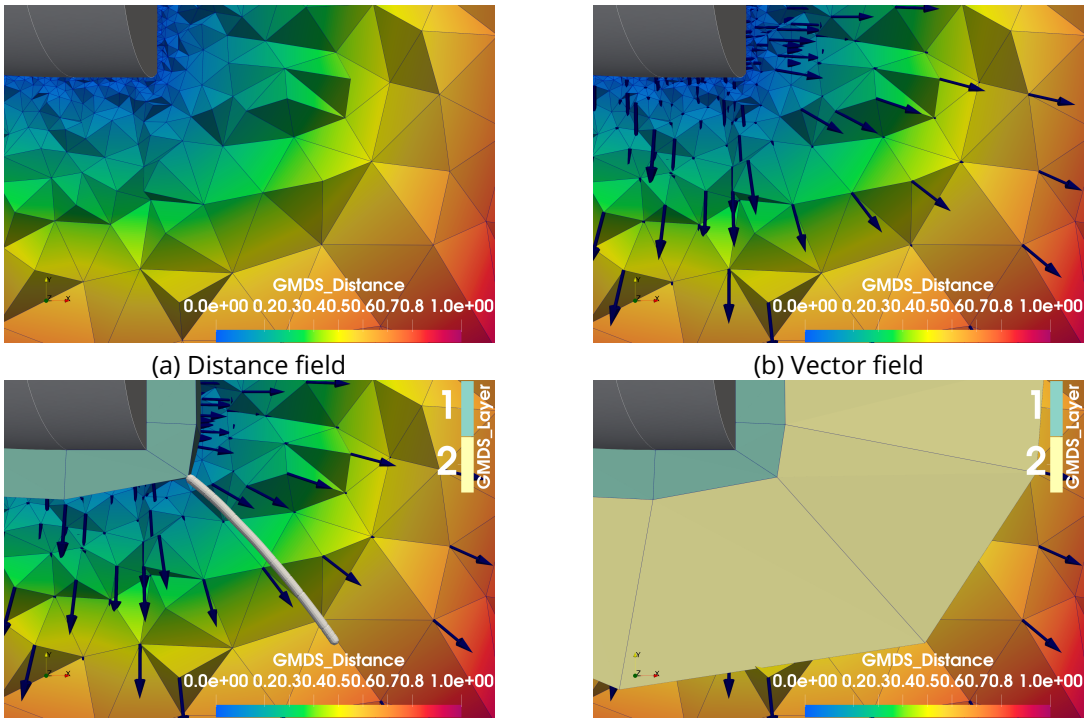
generated blocks (previous layers) in light blue ( ). If there is no conflict on the layer due to block expansion or shrinking, all the blocks are built regularly, and the front block corners of layer $\mathcal{L}_i$ (see Fig. 2.14) are now the input for another step of Algorithm 3 (see Fig. 2.13.b). We can note that the computation of each layer is independent, and the main principle of building a layer is the same in 2D and 3D. In 2D, each block edge of the front generates one quadrangular block of the layer (see Fig. 2.13.a), while in 3D, each quadrangular block of the front creates a hexahedral block (see Fig. 2.13.b).

A practical example of the computation of the position of a block corner in the next front, starting from a front is shown in Figure 2.15. The block corner is advected along the vector field of Figure 2.15.b until the target distance of the front is reached in the distance field of Figure 2.15.a. The discrete trajectory followed by the block corner is plotted using white dots in Figure 2.15.c. A clip view along the plan $(O,\overrightarrow{X},\overrightarrow{Y})$ of the blocks of the layer computed is represented in Figure 2.15.d.

Generation of the First Layer

As mentioned before, the boundary layer is a very thin layer close to the vehicle wall. In this area, the fluid flow is dominated by viscosity effects, which generate very thin boundary layer with high gradient of velocity in the normal wall direction. We take particular care in this area, which we manage with Algorithm 4. The distance field considered for this layer is the Euclidean distance to the vehicle d_{Ω_V} and not the combined distance field d as before. This allows us to set all the orthogonal block edges to the wall at the same distance of the wall. The distance of the layer is supposed to be higher than the boundary layer thickness δ_{BL}.

A practical example of the block-structured extrusion on six different layers is performed around the Apollo geometry (see Appx. B) in 2D (see Fig. 2.16). The distance field is computed on the domain, and we compute the discretization of the vehicle automatically (see Fig. 2.16.a). Then, the first block layer is created on this front, with the specific Algorithm 4. Once this first block layer is computed carefully, the second block layer is created (see Fig. 2.16.c). As we may observe, all the block corners of this layer are located at the same distance on the combined distance field. The final block structure of Figure 2.16.d is built on six different layers.



(a) Distance field (b) Vector field

(c) Block corner trajectory (d) New blocks ( )

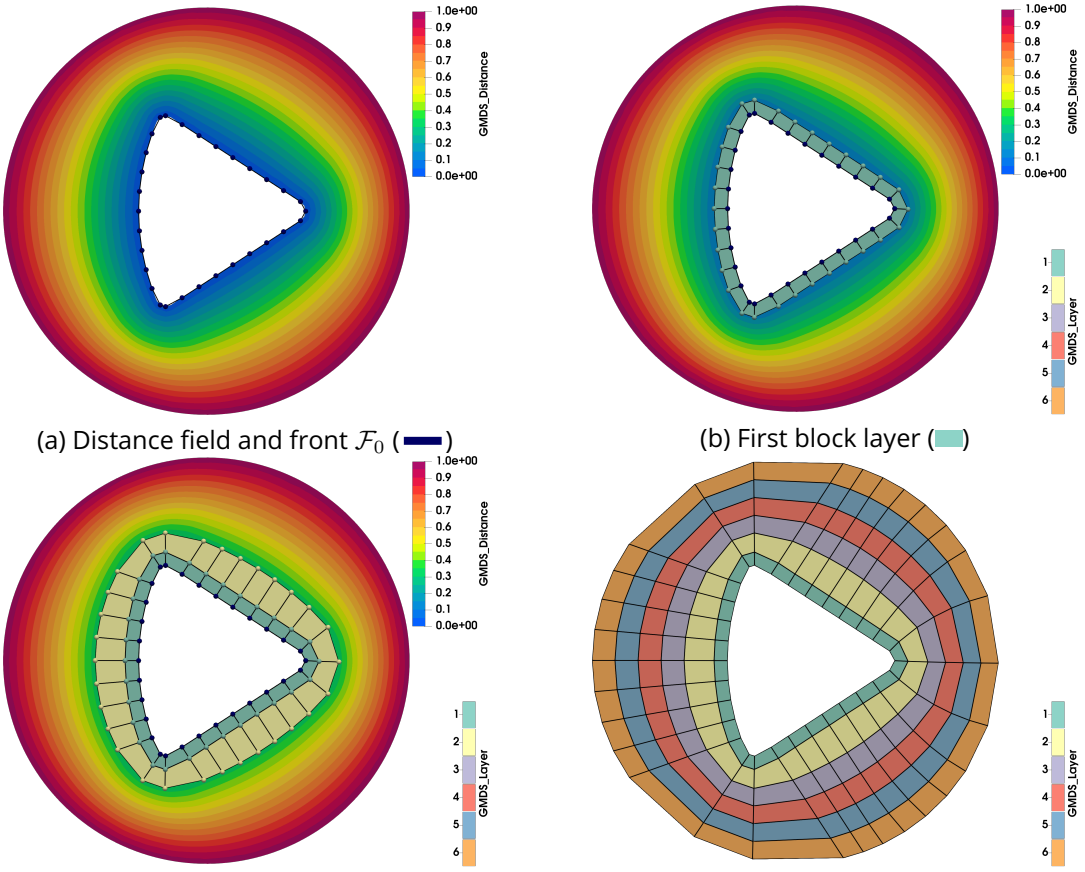
Figure 2.15: 3D Front block corner location computed by advection of the precedent block corner. In (c), the first block layer ( ) is built. To compute the position of the next block corner, the considered block corner is advected along the path represented with white dots. Once the positions are computed for each block corner of the front, the new block layer ( ) is created.

Algorithm 4 Build the First Block Layer

Require: Background mesh $\mathcal{M}_T$, first front $\mathcal{F}_0$, expected boundary layer thickness δ_{BL}, distance field from the vehicle d_{Ω_V}, vector field $\mathbf{v}$

Ensure: Blocks of the first layer $\mathcal{L}_1$ $\mathbf{B}$

1: **for all** block corner $n_0^j \in \mathcal{F}_0$ **do**
2: $n_1^j \leftarrow$ computeIdealPosition(n_0^j, δ_{BL}, d_{Ω_V}, $\mathbf{v}$)
3: **end for**
4: $\mathbf{B} \leftarrow$ computeBlocks($\mathcal{L}_1$)



(a) Distance field and front $\mathcal{F}_0$ ( )

(b) First block layer ( )

(c) Second block layer ( )

(d) Block-structure extruded on six layers

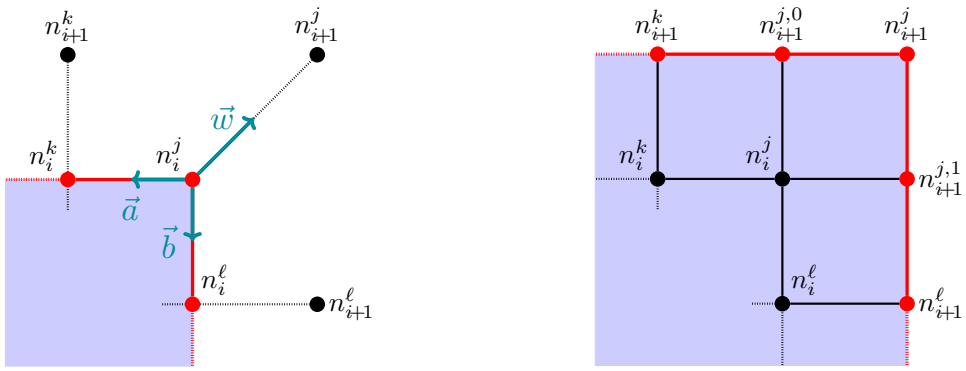
Figure 2.16: A practical example of the extrusion of blocks around Apollo 2D geometry (see Appx. B). Starting from the automatically generated front $\mathcal{F}_0$ in blue ( ) of (a), the first layer ( ) is built (b), then the second layer ( ) in (c). Here, the block structure is extruded along six different layers (d).

2.3 Conflict Management on a 2D Layer

The generation of a regular block layer was presented in the precedent section (see Sec. 2.2.2), where the algorithm is the same in 2D and 3D. However, we have to deal with the expansion or contraction of blocks on the domain. It may lead to stretched blocks in case of domain expansion, with an important opposite edge length ratio. In the case of domain contraction, it may lead to overlapping blocks. The geometric quality of the block impacts the geometric quality of the mesh generated. To increase the blocks' geometric quality, it is necessary to handle those conflicts. To do so specific operations are performed to improve the block quality on the layer while ensuring to provide an output front that respects the definition. This is crucial if we want to construct a new layer on this front.

Starting from a front $\mathcal{F}_i$, and once the positions of the next block corners are computed (see Sec. 2.2.2), we first check some validity rules to ensure that the blocks of the layer $\mathcal{L}_{i+1}$ we want to create have adequate shapes. Those rules are similar to the ones introduced by Blacker et al. [BS91] for the paving method, but use both geometric and physical criteria to classify block corners of $\mathcal{F}_i$. We consider here the geometric shapes of the quadrangular blocks and their alignment with the vector field $\overrightarrow{v}$. As in the paving method, according to this classification, three different operations are used. We can insert one block, fuse two blocks in one, or insert two blocks in the layer $\mathcal{L}_{i+1}$ on a block corner of the front.

Block Insertion



(a) Front $\mathcal{F}_i$ and ideal positions of each next block corner of front $\mathcal{F}_{i+1}$

(b) Extra block built on the layer at block corner n^j_i

Figure 2.17: Example of block insertion performed on block corner n^j_i of the front $\mathcal{F}_i$ (▬) of (a). The block corner n^j_i creates two more block corners, $n^{j,0}_{i+1}$, and $n^{j,1}_{i+1}$ on the new front $\mathcal{F}_{i+1}$ (▬) of (b).

To avoid blocks of poor quality, we allow the insertion of blocks on a layer in areas specified by the user. As explained before, to create a layer $\mathcal{L}_{i+1}$, each block corner of the front $\mathcal{F}_i$ generates a block corner of the front $\mathcal{F}_{i+1}$ at the distance d_{i+1} in the distance field d, following the vector field $\overrightarrow{v}$. Let us consider two block corners n^j_i and n^k_i of the front $\mathcal{F}_i$ connected by a block edge. They are supposed to generate two block corners, respectively n^j_{i+1} and n^k_{i+1}, of the next front $\mathcal{F}_{i+1}$. We may create the regular block on this block edge, defined by the four block corners $(n^j_i, n^j_{i+1}, n^k_{i+1}, n^k_i)$. Depending on this block angle quality, we can reject this block. For

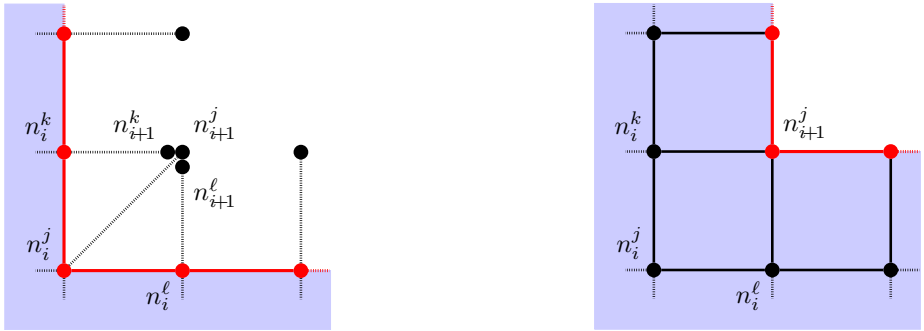
instance, in Figure 2.17.a, the block corner n_i^j of the front $\mathcal{F}_i$ will generate the block corner n_{i+1}^j. The two regular blocks supposed to be created on the layer around the block corner n_i^j will not respect our target angle quality. As a consequence, we would generate two additional block corners from the block corner n_i^j (here, $n_{i+1}^{j,0}$, and $n_{i+1}^{j,1}$), and insert an extra block on the layer (see Fig. 2.17.b). In this work, an extra block is inserted on the block corner n_i^j if the four following criteria are respected (using notations given in Fig. 2.17):

1. The block corner is in an area where the insertion is allowed by the user;
2. The adjacent block corners on the same red front have not already inserted or removed elements;
3. $\frac{\pi}{4} - \sigma_1 < \arccos\left(\frac{\overrightarrow{w}\cdot\overrightarrow{v_n}}{||\overrightarrow{w}||\cdot||\overrightarrow{v_n}||}\right) < \frac{\pi}{4} + \sigma_1$, where $\overrightarrow{v_n}$ is the value of the vector field at the position of the block corner n_i^j;
4. $\arccos\left(\frac{\overrightarrow{w}\cdot\overrightarrow{a}}{||\overrightarrow{w}||\cdot||\overrightarrow{a}||}\right) + \arccos\left(\frac{\overrightarrow{w}\cdot\overrightarrow{b}}{||\overrightarrow{w}||\cdot||\overrightarrow{b}||}\right) > \frac{3\pi}{2}$;

where $\sigma_1 = 0.174$ is an arbitrary tolerance corresponding to 10°. It is usual 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, but one can notice it will impact the discretization of the final mesh.

To compute the position of one of the two new block corners used to create the inserted block, we proceed as follows. To build the block corner $n_{i+1}^{j,0}$ (see Fig. 2.17.b), the position of the block corner n_i^j is advected following the constant vector $\frac{\overrightarrow{w}}{||\overrightarrow{w}||} + \frac{\overrightarrow{a}}{||\overrightarrow{a}||}$ until reaching the distance d_{i+1} in the distance field d. We do the same to compute the location of the second block corner ($n_{i+1}^{j,1}$).

Block Shrinking



(a) Front $\mathcal{F}_i$ and ideal positions of each next block corner of front $\mathcal{F}_{i+1}$

(b) Only one block is generated around the block corner n_i^j instead of two

Figure 2.18: Block shrinking example (2D). Each block corner (●) of the front (▬) of (a) computes the ideal position of the next block corner (●). Three potential new block corners are too close, so they are merged to provide only one new block instead of two quadrangular blocks on the new layer (b).

The block shrinking operation can be considered as the opposite of the block insertion. Considering three consecutive block corners n_i^j, n_i^k, and n_i^ℓ of the front $\mathcal{F}_i$, that respectively

generate the block corners n_{i+1}^j, n_{i+1}^k, and n_{i+1}^ℓ of the front $\mathcal{F}_{i+1}$. We apply the shrinking process when we meet the configuration of Figure 2.18.a, where the three generated block corners are geometrically close. More specifically, we fuse the three generated block corners into a single one. To detect the places where the operation is necessary, the proximity of the ideal positions of the block corners of the next layer is controlled with a tolerance. After the fusion, the adjacent block corners on the layer (i.e., connected by an edge) are not able to perform an insertion or fusion operation anymore. In Figure 2.18.a, block corner n_i^j was supposed to be adjacent to two different quadrangular blocks on the layer, but those two blocks are fused. So the block corner is only adjacent to one quadrangular block on the new layer (see Fig. 2.18.b).

Figure 2.19 illustrates a practical example of how blocks can be inserted or shrunk in a layer. The domain represents the fluid around the Mars Spacecraft geometry [Sca07] ( ) (see Appx. B). On the second block layer, two blocks are fused in one ( ), while one block is inserted on the right part of the geometry ( ).

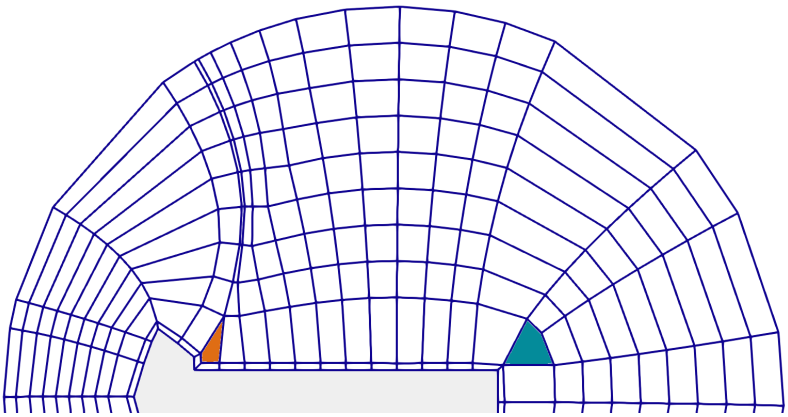


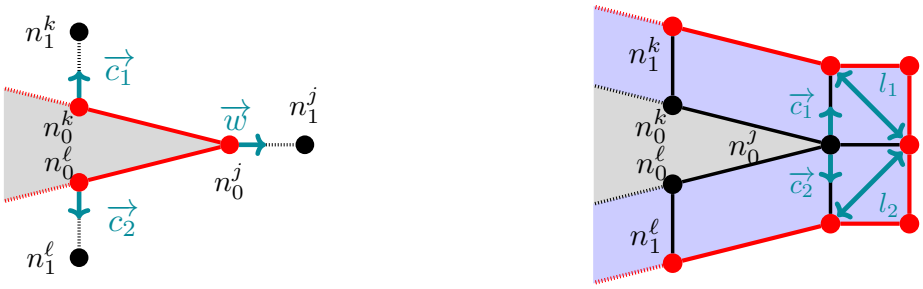
Figure 2.19: Blocks around Mars Spacecraft geometry (see Appx. B). Only one side of the 2D geometry is represented here ( ). On the second layer of the quadrangular block structure, two blocks are fused in one ( ), and a block is inserted on a block corner of the front ( ).

Double Block Insertion

(a) Diamond-shaped airfoil (b) NACA 0012 airfoil [Rum21]

Figure 2.20: Examples of 2D geometries with very sharp angles.

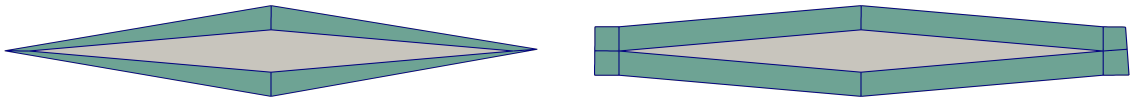
As explained before, we expect to generate a block structure as regular as possible, so we may want to avoid any insertion or shrinking operations on a layer. At least, in the very thin first block layer, we completely disable the two operations presented before. We consider that, regarding the geometries treated in this work, the first block layer will be thin enough to avoid conflicts. However, we still need to solve a very specific type of conflict, due to sharp geometries (e.g., the 2D NACA 0012 airfoil back in Fig. 2.20.b). Otherwise, the geometry feature will introduce very sharp blocks, and the resulting mesh may have very poor quality cells (or overlapping cells).



(a) (b)

Figure 2.21: Double block insertion on the boundary layer. There is a very sharp angle on the geometry ( ). To create a valid block layer on the first front ( ) of (a), we perform the insertion of two blocks at block corner n_0^j (b).

The double block insertion is similar to the one exposed before, but this time two blocks are inserted on a block corner (see Fig. 2.21). If the insertion is not performed, it can lead to tangled invalid meshes. In Figure 2.21.a, the front considered for the extrusion is $\mathcal{F}_0$ (i.e., the wall geometry block discretization), and the corresponding block corners of this front are plotted in red ( ). This operation can only be performed on this front $\mathcal{F}_0$. Each block corner of the front $\mathcal{F}_0$ computes the ideal position of the next block corner. At the position of block corner n_0^j, a sharp angle is detected on the geometry surface. Then, two blocks are inserted. If the inserted upper right block of Figure 2.21 is considered, the two new block corners are placed this way. The first one, connected to n_1^k by a block edge, is the position p_0^j of block corner n_0^j advected at the distance δ_{BL} in the distance field d_{Ω_V} following a constant vector equal to the vector $\overrightarrow{c_1}$ in Figure 2.21.a. The second block corner is placed at position $p = p_0^j + l_1 \frac{\overrightarrow{c_1}+\overrightarrow{w}}{||\overrightarrow{c_1}+\overrightarrow{w}||}$. The second block of this insertion is built the same way, from the block corner n_0^ℓ on the other side of n_0^j. Figure 2.22 illustrates a practical example of this insertion of two blocks at a block corner on the boundary layer blocking, around a diamond-shaped airfoil geometry (see Fig. 2.20.a).



(a) (b)

Figure 2.22: Practical example of 2D double block insertions. Insertions of two blocks on the boundary layer ( ) of a 2D diamond-shaped airfoil ( ). The two insertions in the front and back of the airfoil are performed to ensure good quality elements in those sensitive areas. Without the blocks inserted, the block structure obtained in (a) has very poor-quality elements.

Finally, in order to handle the different types of conflicts, Algorithm 5 is performed to build a layer on a 2D front, where the three different operations are performed if necessary. Figure 2.23 represents a practical example of this 2D algorithm with conflict management to generate a block structure around the NACA 0012 geometry presented before. Starting from the automatic block structure generated in Figure 2.23.a, the first block layer of Figure 2.23.b is built. On this first block layer, an insertion of two blocks at a block corner of the front $\mathcal{F}_0$ is performed, on the right part of the geometry. This layer provides a new front on which the second block layer is computed (see Fig. 2.23.c). On this layer, two blocks are inserted on two different block corners of the front $\mathcal{F}_1$, on the right of the geometry. The final block structure

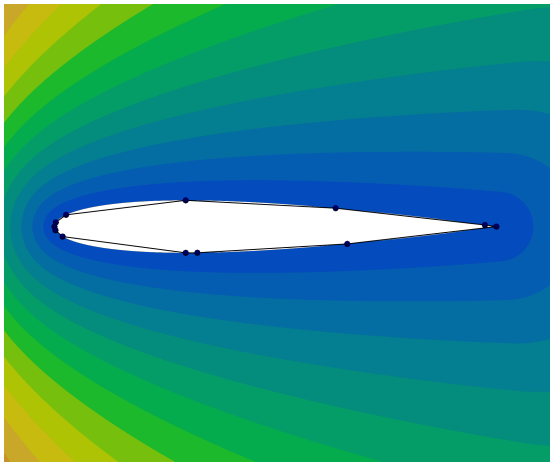
Algorithm 5 Build One Block Layer on a 2D Front

Require: Background mesh $\mathcal{M}_T$, front $\mathcal{F}_i$, distance field d, vector field $\overrightarrow{v}$, distance d_{i+1} of the next front block corners in the distance field d
Ensure: Quadrangular blocks $\mathbf{B}$ of the layer $\mathcal{L}_i$, and the set of block corners and edges of front $\mathcal{F}_{i+1}$

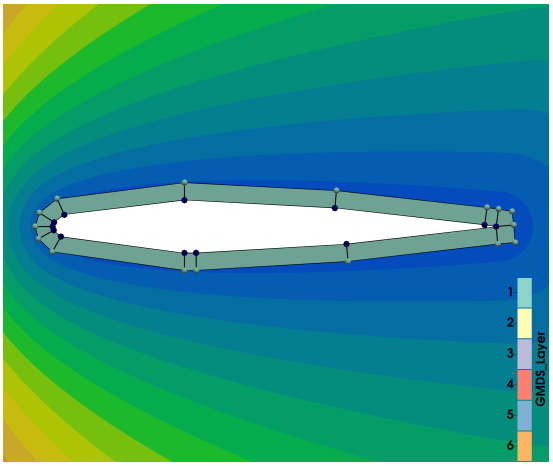
for all block corner $n_i^j \in \mathcal{F}_i$ **do**
  $n_{i+1}^j \leftarrow$ computeIdealPosition(n_i^j, $d_{\mathcal{F}_i}$, d, $\overrightarrow{v}$)
end for
while there is a conflict at block corner $n_i^k \in \mathcal{F}_i$ **do**
  $n_i^k \leftarrow$ getBlockCorner($\mathcal{F}_i$, conflict_type)
  if conflict_type is *expansion* **and** $i > 1$ **then**
    $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_{i+1} + \{$insertQuadBlockAtCorner($n_i^k$)$\}$
  else if conflict_type is *contraction* **and** $i > 1$ **then**
    $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_{i+1} + \{$contractQuadBlockAtCorner(n_i^k)$\}$
  else if conflict_type is *sharp geometry* **and** $i = 0$ **then**
    $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_{i+1} + \{$insertTwoQuadBlocksAtCorner(n_i^k)$\}$
  end if
end while

generated on six different layers of blocks is represented in Figure 2.23.

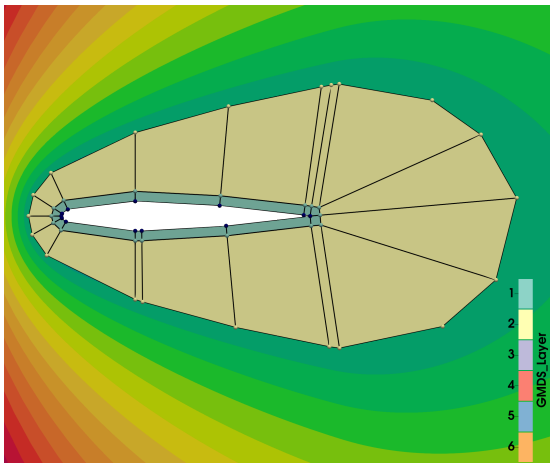
We can note several interesting facts about this method. First, if all the fronts and layers respect the definitions given before, they are not all the same size in terms of the number of elements. For instance, if an insertion of one quadrangular block is performed on a layer, this leads directly to the addition of two new n–1-cells in the next front. The second interesting fact is we can easily generate various block structures with different topologies or sizes just by changing the input parameters. If we take the example of the second block layer generated around the NACA 0012 geometry (see Fig. 2.23.c), we could get rid of the block insertions if the user decides they are not mandatory.



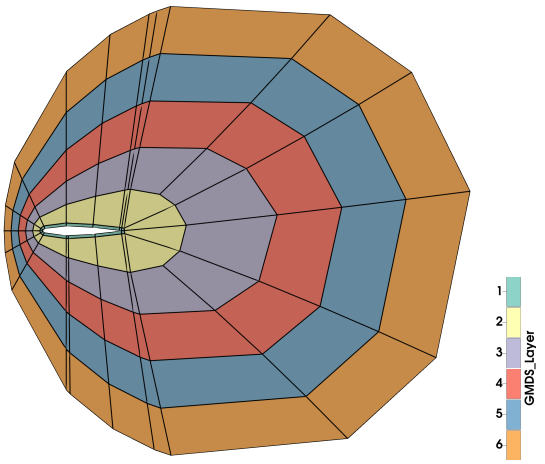
(a) Distance field and front $\mathcal{F}_0$ ( )



(b) First block layer ( )



(c) Second block layer ( )

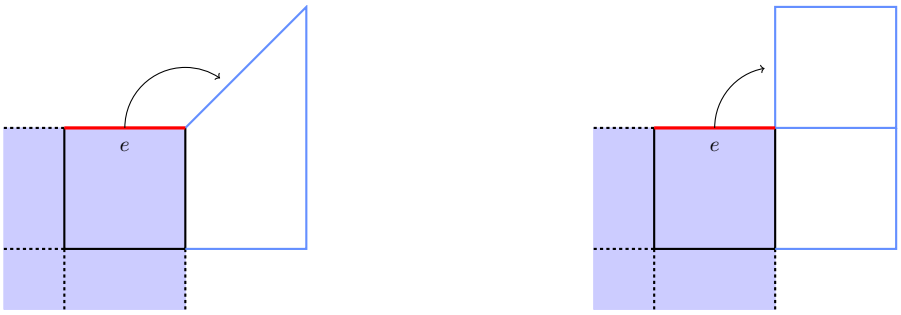


(d) Block-structure extruded on six layers

Figure 2.23: Practical example of the extrusion of blocks around NACA 2D geometry. Starting from the automatically generated front $\mathcal{F}_0$ plotted in blue ( ) in (a), the first layer ( ) is built (b). On this layer, an insertion of two blocks is performed on the right side of the geometry to deal with its sharp angle. Then the second layer ( ) is generated (c). The final block structure is then extruded regularly for the next layers. Finally, the six different layers (d) are generated.

2.4 Conflict Management on a 3D Layer

As this 2D approach was designed to be extensible to 3D, many steps are performed the same way in the 2D algorithm and the 3D algorithm. For instance, we compute the distance and vector fields with the same algorithm in 2D and 3D. But instead of working on a triangular background mesh, we work on a tetrahedral background mesh. The main difference is in the way we handle conflicts on a layer.

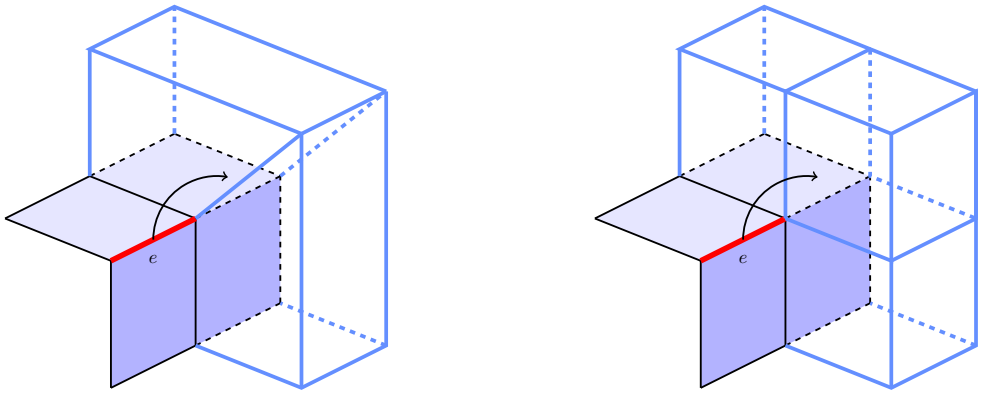


(a) Without insertion (b) With insertion

Figure 2.24: Topology Preservation 2D. The quadrangular blocks of the precedent layer are filled in light blue ( ), and the block edges are plotted in black. The blocks of the layer in construction are plotted in blue ( ).

In the paving algorithm, we can only perform three different operations on a layer to solve conflicts and the three different operations are performed at block corners (see Sec. 2.3). At a block corner, you can choose to insert a quad, to insert two quads, or to shrink two quads into one. In 3D, the different configurations are more numerous and complicated. First, the different operations can be performed at block corners or edges. In addition, in the 2D case, if you decide to insert or not a quadrangular block at a block corner, it does not change the topology for the edges of the front connected to the block corner, there is a preservation of the structure. If we take a look at Figure 2.24 if an insertion is performed (b) or not (a), in both cases, when the red block edge e ( ) of the front will have to create its quadrangular block, the block will connect to a block edge shared by the two fronts. Now consider the 3D case of an insertion of a hexahedron on a block edge represented in Figure 2.25.b (which is the natural extension of the insertion of a quadrangular block at a block corner in 2D). The red block edge e ( ) has to insert a hexahedron too in order to connect to this inserted hexahedron via the block face. In a way, this insertion will be propagated on a set of adjacent block edges of the front. While in the case there is no insertion, such as in Figure 2.25.a, the edge e will not be able to insert any hexahedron, because there is no face to reconnect to properly.

So in 2D, the insertion (or contraction) of a quadrangular block is a local problem, while in 3D we need to consider first a global problem on the whole front before performing the operations. The main idea is to ensure the global coherence of the topology of the layer before applying the local patterns, on each block corners and edges. To build a full layer of hexahedral blocks on a front, we first analyze and classify all the block edges of the front. Then, according to this classification, we compute on which block corners and edges we can insert or remove hexahedra. Once we know where to build each hexahedron, then we build the full layer of hexahedra.



(a) Without insertion (b) With insertion

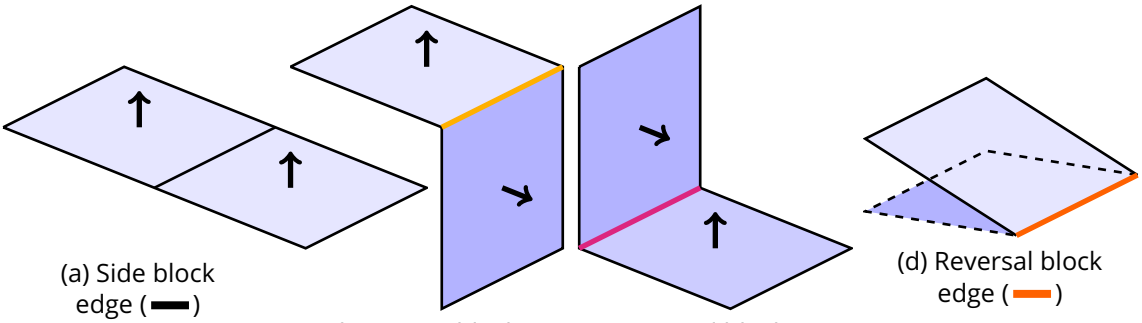
Figure 2.25: Topology propagation in 3D.

2.4.1 Front Edges Classification

To compute a block layer, we take a front as input. A **3D front** is a set of quadrangular faces, edges, and block corners. We first detect the geometrical features of the front and classify all the edges of the front. The topology of the layer of hexahedra will be defined by solving a global problem on this front. In our case, as a front can not separate, each edge of the front is connected to exactly two quadrangular block faces on this front.

The very first step of the algorithm is to compute the classification of each block edge of the front, according to the angle Φ between the outgoing normal vectors to its two adjacent quadrangular block faces on the front. As shown in Figure 2.26, an edge can be classified as

$$\begin{cases} \text{(b) } \textbf{CORNER} & \text{if } \frac{\pi}{4} \leq \Phi < \frac{3\pi}{4}, \\ \text{(d) } \textbf{REVERSAL} & \text{if } \frac{3\pi}{4} \leq \Phi < \frac{5\pi}{4}, \\ \text{(c) } \textbf{END} & \text{if } \frac{5\pi}{4} \leq \Phi < \frac{7\pi}{4}, \\ \text{(a) } \textbf{SIDE} & \text{otherwise.} \end{cases} \tag{2.10}$$

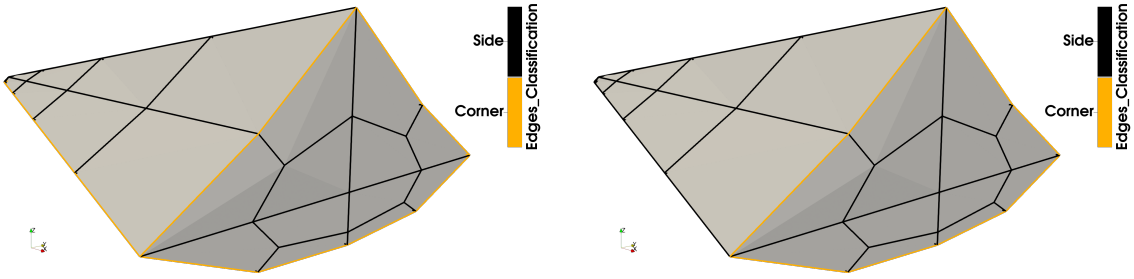


(a) Side block edge ( ) (b) Corner block edge ( ) (c) End block edge ( ) (d) Reversal block edge ( )

Figure 2.26: Block edges classification on a front ( ).

Definition 2.4.1 (**Feature block edge**)**.** A **feature block edge** is a block edge of the front that is not classified as side. Its classification is either corner, end, or reversal (see Fig. 2.26).

Definition 2.4.2 (**Feature block corner**)**.** A block corner is defined as a **feature block corner** if it is connected to at least three feature block edges on the front.



(a) (b)

Figure 2.27: Example of block edges classification on a front. The corner block edges are represented in yellow ( ), and the side block edges in black ( ). The front (a) is classified following the same process as in [RG11, RGRS12], respecting Equation (2.10). While the front (b) is classified with the specific directive to avoid feature block edges on the left part of the front.

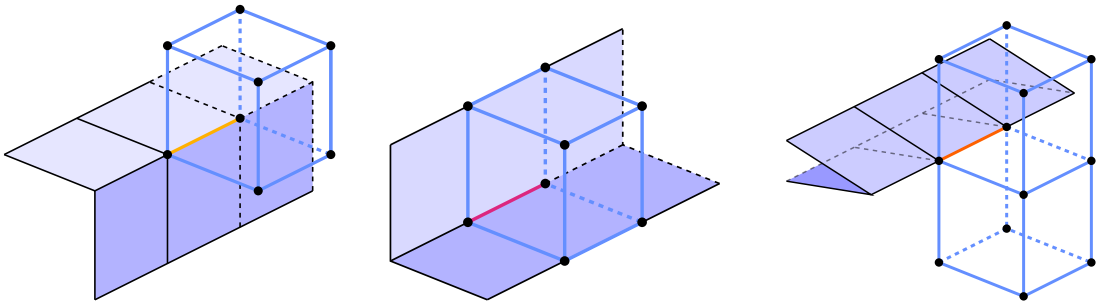
A practical example of this classification is illustrated on the front of Figure 2.27. The classified front of Figure 2.27.a is obtained using the rules of Equation (2.10). This front is composed of a set of corner block edges plotted using yellow ( ) edges, and a set of side block edges plotted in black ( ). Let us note that, unlike in the work of Ruiz-Gironés et al., we can choose here to declassify an edge (i.e., we may decide that a corner, end, or reversal edge will be considered as a side edge), according to some physical parameters. As for the 2D algorithm, we want to introduce a few singularities in the block structure topology. So the user can decide to declassify block edges of the front depending on their location in the domain, considering the quality of the blocks will be enough without adding or removing some blocks in the layer. So, for instance, the classified front of Figure 2.27.a becomes the classified front of Figure 2.27.b if we want to consider all the block edges on the left part of the figure as regular edges. A set of corner block edges ( ) in Figure 2.27.a has been declassified, to be considered as side block edges ( ) in Figure 2.27.b. This choice can impact the whole block structure and makes it possible to generate different topologies of blocking around the same vehicle, examples will be given later.

2.4.2 Patterns Considered to Solve Conflicts

The algorithm aims to apply a set of **patterns** of hexahedra on the front to solve the potential conflicts due to expansion or contraction on a layer.

Definition 2.4.3 (**Pattern**)**.** A **pattern** is defined as a set of hexahedral blocks created around a block corner or a block edge of a front $\mathcal{F}_i$. This set of blocks belongs to the block layer $\mathcal{L}_i$ built on front $\mathcal{F}_i$.

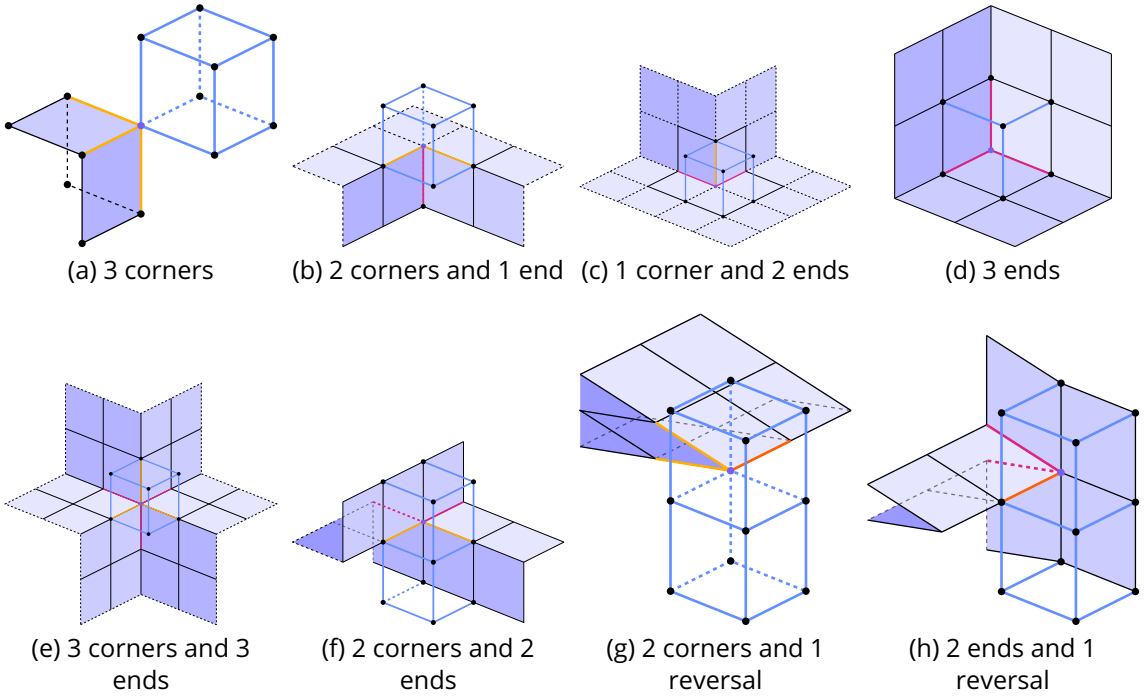
As this work is based on the work of Ruiz-Gironés et al. [RG11, RGRS12], we decided to implement the same patterns they did to solve conflicts on a layer. We first have the three patterns on block edges presented in Figure 2.28. They can be seen as a natural extension of the three 2D patterns on block corners, but on block edges. We can insert a block on a corner block edge (see Fig. 2.28.a), fuse two blocks to only create one on an end block edge (see Fig. 2.28.b), or create two new blocks on a reversal block edge (see Fig. 2.28.c). As explained before, the use of one of those patterns on a block edge will need to be propagated on the adjacent block edges.



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

Figure 2.28: Conflict management 3D: patterns applied on a feature block edge of the front ( ) depending on its classification.

In addition to the patterns on feature block edges, eight different patterns are introduced to deal with configurations of feature block edges around the block corners of a front (see Fig. 2.29). Each configuration represents a feature block corner ( ) of the front ( ) adjacent to at least three feature block edges on this front. Around each of these eight configurations, we can create a set of hexahedral blocks ( ) that belong to the new block layer. Some patterns only create one block, while some others create two different blocks.



(a) 3 corners (b) 2 corners and 1 end (c) 1 corner and 2 ends (d) 3 ends

(e) 3 corners and 3 ends (f) 2 corners and 2 ends (g) 2 corners and 1 reversal (h) 2 ends and 1 reversal

Figure 2.29: Conflict management 3D: patterns to apply on feature block corners ( ) of the front ( ). For one configuration of feature block edges around a feature block corner, the set of block(s) created around the block corner is represented with blue edges ( ).

Similarly to the 2D case, some patterns provide less number of n–1-cells on the next front than on the front they are applied (see Fig. 2.28.a, c, Fig. 2.29.a, g, etc.), while some of them will provide a larger number of n–1-cells on the next front (see Fig. 2.28.b, Fig. 2.29.d, etc.). For instance, the pattern around three corners of the front (see Fig. 2.29.a) will provide three new n–1-cells on the next front, and seven block corners on it. The pattern applied around three end block edges of the front (see Fig. 2.29.d) will not provide any n–1-cell on the next

front, and only one block corner on it. It is important to note that the patterns aim to deal with specific conflicts on a part of the front. When it is done, all the quadrangular block cells of the front are supposed to generate a block in the layer by extrusion of itself as presented before (see Fig. 2.30). However, due to some specific patterns around block edges or block corners (e.g., the pattern on an end block edge in Fig. 2.28.b), some quadrangular blocks of the front will not generate any additional hexahedral block on the layer.

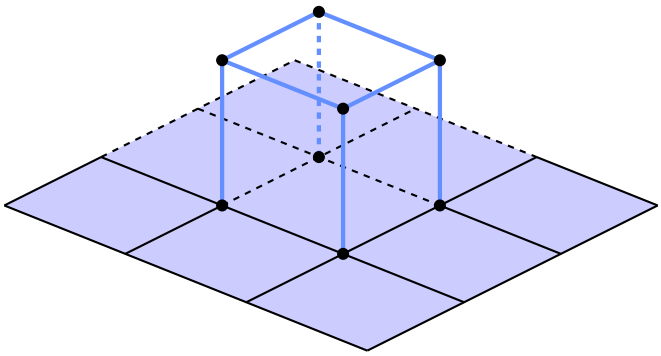


Figure 2.30: Pattern on face.

As in many methods where several cases are considered, there is always the reasonable question of the completeness of the cases listed. As explained in [RGRS12], there exist additional patterns to take into account other arrangements of block edges around a block corner of a front. However, they are not considered in this work. The reason why we can afford to take those extra cases into account is that we first compute on the front which patterns we can apply and were, before to build the corresponding hexahedra, and this computation step respects the list of patterns we implemented.

2.4.3 Paths of Feature Block Edges on a Front

To compute *a priori* on which feature block edge and feature block corner we can apply patterns on a front, we compute a set of **paths** (see Def. 2.4.4).

Definition 2.4.4 (Path). A **path** is a set of adjacent block edges (and block corners) on a 3D front sharing the same classification on this front. A path starts by a feature block corner and stops on a feature block corner. Each block corner in this path is adjacent to exactly two feature block edges on this path or is a bounding feature block corner. A path can start and end on the same feature block corner.

To compute a path, we start from a feature block corner and one of its adjacent feature edges. Then, we try to connect to another feature block corner according to Algorithm 6. Let us note that the path computation stops here when the last block corner checked is adjacent to less than two feature edges of the same classification type, or if the last block corner is a feature block corner as defined before. In the first case, we decide in this work to consider the path as non-valid because it does not respect the Definition 2.4.4 given before. In this case, we declassify all the block edges of this path (i.e., we consider them as side edges). This implies we have to perform again the path computation algorithm on all the block edges of the front, according to this new classification. This is one of the differences with the work

Algorithm 6 Single Path Computation

Require: Classified front $\mathcal{F}_i$, starting block corner n_s, starting block edge e_s
Ensure: Path $\mathcal{P}$
1: $n \leftarrow e_s$.oppositeNode(n_s)
2: $e \leftarrow e_s$
3: $\mathcal{P}$.add(e_s)
4: **while** n.nbrClassifiedEdges() $= 2$ **and** $e = e_s$.classification() **do**
5: $e \leftarrow n$.nextEdge()
6: $\mathcal{P}$.add(e)
7: **end while**
8: $n_e \leftarrow n$

presented in [RGRS12], where they choose to consider those specific paths as *semi-paths*, and they introduce additional patterns to deal with those configurations. In fact, to decide if a path is valid here, the path has to respect some conditions:

1. The two bounding feature block corners have to be connected to at least three feature block edges of the front, and respect one of the eight configurations listed previously (see Fig. 2.29).

2. All the edges of the path have to share the same classification according to previous Section 2.4.1.

Algorithm 7 All Paths Computation

Require: Front $\mathcal{F}_i$
Ensure: List of all valid paths on the front $\mathcal{F}_\mathcal{P}$
1: **while** List is not valid **do**
2: clear($\mathcal{F}_\mathcal{P}$)
3: validList $\leftarrow$ true
4: **for all** valid feature block corner n_s **do**
5: **for all** valid feature edge e_s adjacent to n_s **do**
6: $\mathcal{P} \leftarrow$ PathComputation(n_s, e_s) ▷ (see Algo. 6)
7: **if** $\mathcal{P}$ is *valid* **then**
8: $\mathcal{L}_\mathcal{P} \leftarrow$ add(PathComputation(n_s, e_s))
9: **else**
10: declassifyEdges($\mathcal{P}$)
11: validList $\leftarrow$ false
12: **end if**
13: **end for**
14: **end for**
15: **end while**

This approach is more restrictive than in [RG11, RGRS12]. In case a path is classified as invalid, the starting and ending block corners of the path are considered invalid too. The direct consequence is that front block corners must be re-classified and potentially valid paths are

now not valid anymore. When they treat those invalid paths with special patterns, we decide in this work to declassify all the edges of the paths (to classify them as side edges, see Algo. 7, l. 10) and to recompute all the paths.

After the computation of all paths from one feature block corner to another, we decide to compute the potential loops of feature block edges on the front, called here **closed paths** (see Def. 2.4.4). A closed path is a path of adjacent feature block edges containing no feature block corner. This means that all the block corners of a closed path are adjacent to strictly two feature block edges of the front. All the feature block edges of a closed path have to share the same classification according to Section 2.4.1. To our knowledge, the closed paths were not considered in the work of Ruiz-Girones et al.. Due to our application, those configurations are mandatory because the back part of many of our geometries is affected by these particular paths (e.g., the right part of the vehicle presented in Fig. 2.31).

Definition 2.4.5 (**Closed path**)**.** A **closed path** is a set of adjacent feature block edges of a front sharing the same classification (corner, end, or reversal). A closed path does not contain any feature block corner. This means that every block corner of a closed path is adjacent to exactly two feature block edges on the front.

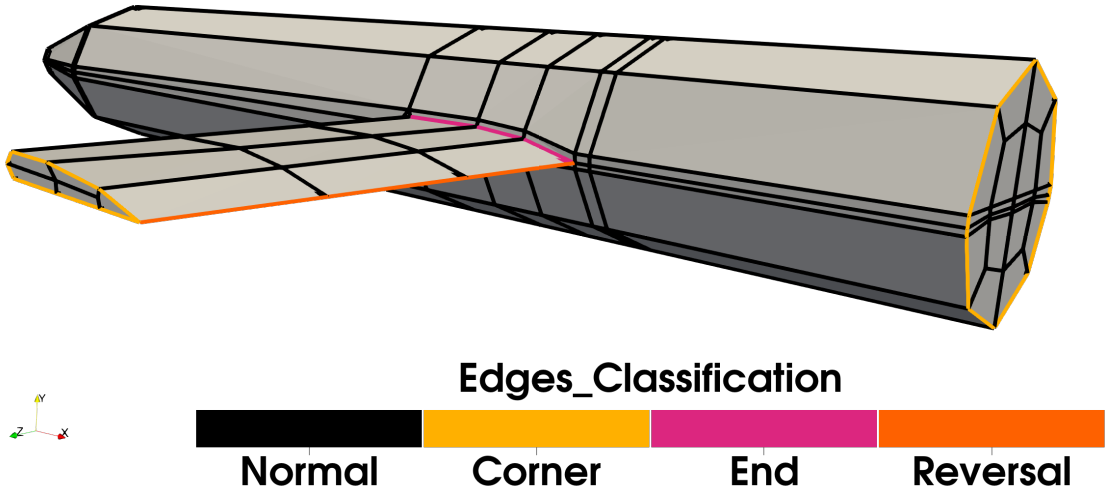


Figure 2.31: Practical example of block edges classification on a front. The corner block edges are represented in yellow (—), the end block edges in purple (—), the reversal block edges in orange (—), and the side block edges in black (—).

Considering the practical example of front classification presented in Figure 2.31, three different types of paths presented before are illustrated. The most traditional path is represented here with the three reversal block edges (—). This path is bounded by two different feature block corners. The first one, on the left side, is a block corner adjacent to one reversal block edge (—) and two corner block edges (—) (see Fig. 2.29.g for the corresponding pattern). The other side of this reversal path is bounded by a block corner adjacent to one reversal block edge (—) and two end block edges (—) (see Fig. 2.29.h for the corresponding pattern). Another interesting path is the one composed of corner block edges (—) on the wing (left part of the figure). This path starts and ends on the same feature block corner, shared by the precedent path. The last type of path illustrated here is a closed path at the back of the vehicle (right part of the figure). This path is a corner block edges (—) path and it is not bounded

by any feature block corner. Each block corner of this path is adjacent to exactly two corner block edges on the front and some side block edges.

Once all the paths are computed on the front, we know *a priori* at each block corner, block edge, and block quadrangular face of the front which pattern will be applied. The creation of the hexahedra of the layer is performed in three steps:

1. **Patterns on feature block corners**. We start with the creation of a set of new hexahedra on each marked feature block corner of the front according to the eight valid configurations of feature block edges around a block corner (see Section 2.4.2).
2. **Patterns on feature block edges**. Then we apply the block edge patterns presented in Section 2.4.2 depending on the classification of the edges, and the set of valid paths computed before.
3. **Patterns on remaining block faces**. Finally, hexahedra are created regularly (by extrusion of the face) on all the remaining faces of the front.

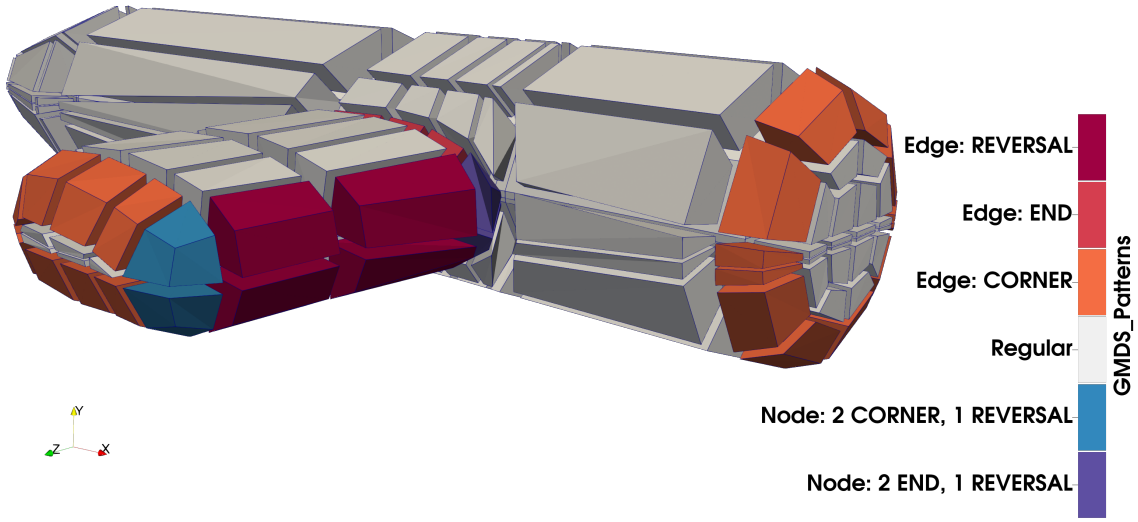


Figure 2.32: Practical example of a block layer generated on the previously classified front of Figure 2.31. The layer is represented using a shrunk view. The hexahedra built on a block face of the front are represented in gray (), while the hexahedral blocks built using patterns are plotted using different colors. Here, two sets of blocks are built using patterns on the block corners of the front, and plotted in blue () and purple (). The other colored blocks are built using patterns on the feature block edges of the front.

The layer of hexahedral blocks generated on the classified front of Figure 2.31, after the paths computation, is presented in Figure 2.32. The hexahedra generated using patterns are represented in different colors (corresponding to the patterns applied) while the hexahedra created regularly (by extrusion of a quadrangular block face of the front) are plotted in gray (). Here, the hexahedral blocks of the layer are represented using a shrunk view in order to see the patterns applied on front block edges classified as end (), and the two hexahedral blocks created by the pattern applied on the block corner adjacent to two end block edges and one reversal block edge of the front (). As we may see, the hexahedra built using those patterns

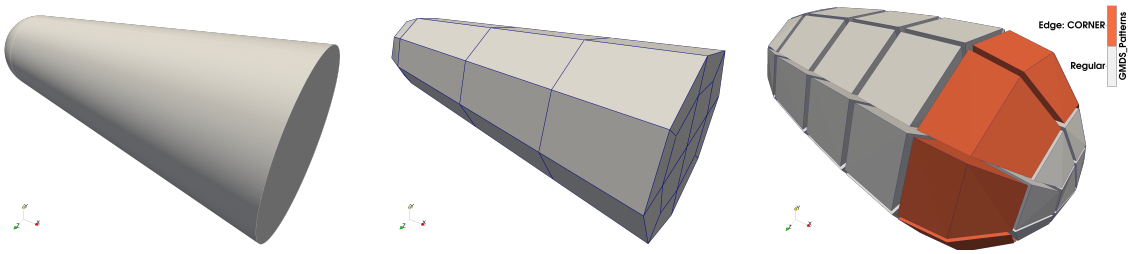
do not provide any block face on the next front. In this practical example, the use of the two other patterns is illustrated. The pattern on corner block edges provides one hexahedron on each block edge ( ), while the one applied on each reversal block edge provides two hexahedra each per block edge ( ). The last pattern illustrated in this example is the one around a block corner adjacent to two corner block edges, and one reversal block edge. This pattern builds two hexahedral blocks on the layer plotted here in blue ( ).

The operations presented to deal with potential conflicts on a layer ensure the layer of block-cells $\mathcal{L}_i$ built this way on front $\mathcal{F}_i$ provides a new front $\mathcal{F}_{i+1}$ which respects strictly the definition of a front. So this new front is suitable for performing a new step of the algorithm, i.e., to build a new block layer on it.

2.5 Results

This section aims to exhibit some results of this algorithm around vehicles and to comment on the possibilities of the method. Results are shown in this part around three different 3D examples: the RAM-C II vehicle, a double ellipsoid, and a caretwing (see Appx. B). The study of the double ellipsoid geometry shows how to generate different block topologies around the same input geometric surface by modifying the input parameters of the algorithm. The final result around the caretwing illustrates more complex patterns used on a layer.

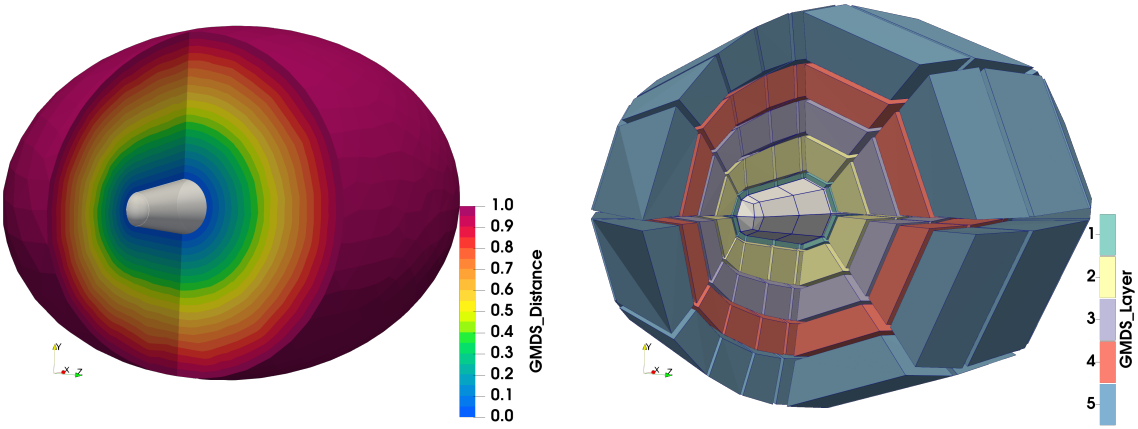
RAM-C II



(a) RAM-C II vehicle (b) Front $\mathcal{F}_0$ (c) Second block layer

Figure 2.33: Practical example of second block layer (c) generated generated around RAM-C II geometry (a) (see Appx. B) using the initial front (b).

Here is a practical example of a block structure generated around the RAM-C II vehicle (see Fig. 2.33.a). This vehicle has a conical shape with a spherical nose (left part of the figure). The vehicle is oriented in such a way we will refer to the spherical part as the nose, or front part, and the disk as the back. The quadrangular block discretization of the vehicle used as our first front is given in Figure 2.33.b. The first block layer is extruded regularly (i.e., no patterns are used), while we allow patterns in the second block layer. As a result, we obtain in the second block layer of Figure 2.33.c a closed path of hexahedral blocks on the back of the vehicle. This is the only layer built using patterns. The final block structure composed of five different block layer is presented in Figure 2.34.b. The block structure is generated on the fluid domain represented in Figure 2.34.a by the background mesh $\mathcal{M}_T$, and the combined distance field computed on it. Here, we decided to represent the domain using two clip plans for visualization purposes, but the block structure is generated on the plain domain.

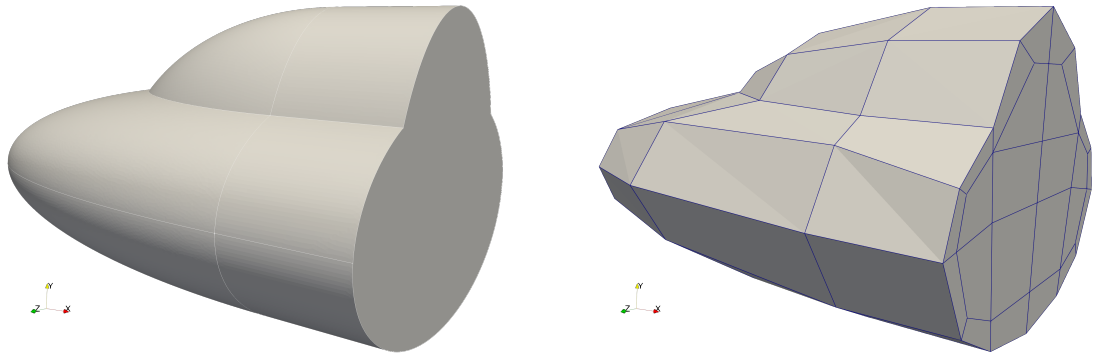


(a) Distance field d (b) Blocks

Figure 2.34: Practical example of block structure extruded on five layers (b) generated around RAM-C II geometry (see Fig. 2.33.a). The combined distance field computed on the background mesh is plotted in (a). Here, we cut a quarter of the domain for visualization purposes, but the domain meshed around the geometry and the resulting blocking are plain.

Double Ellipsoid

Let us now consider the Double Ellipsoid geometry [AADLC91] of Figure 2.35.a and its surface discretization given in Figure 2.35.b. Using the algorithm detailed before, we can generate various different block topologies around a same vehicle, and we illustrate here some example around this specific geometry. Figure 2.36 shows three different block topologies generated using the same starting front $\mathcal{F}_0$ of Figure 2.35.b.

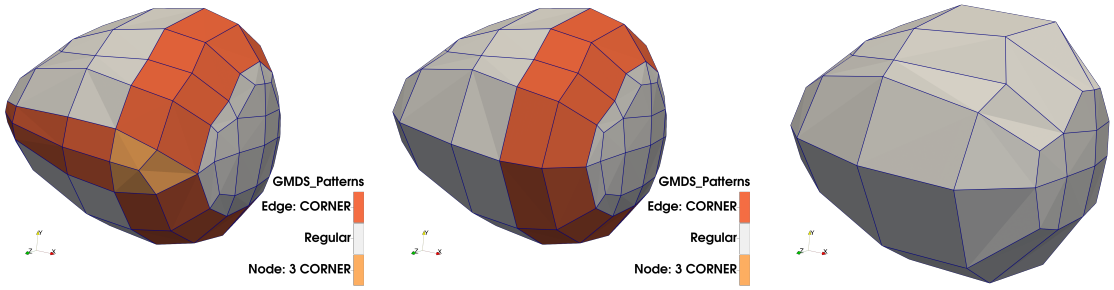


(a) Double Ellipsoid geometry (b) Double Ellipsoid front $\mathcal{F}_0$

Figure 2.35: Double Ellipsoid 3D input geometry [AADLC91] and block discretization of the vehicle surface.

Figure 2.36.a represents the second block layer generated of a block structure extruded on eight different layers. This layer is the only one of the blocking where patterns are used. The first block layer $\mathcal{L}_1$ is generated in a fully regular way on front $\mathcal{F}_0$ (i.e., each quadrangular cell of the front generates a hexahedral block, no patterns are used). This first regular layer provides a new front on which patterns are applied to provide the second block layer of Figure 2.36.a. The only reason why the patterns are not used in the first block layer is because we decided to disable the conflict management on the first layer to generate this specific block structure.

The result of Figure 2.36.b is obtained by starting the algorithm again with slightly different



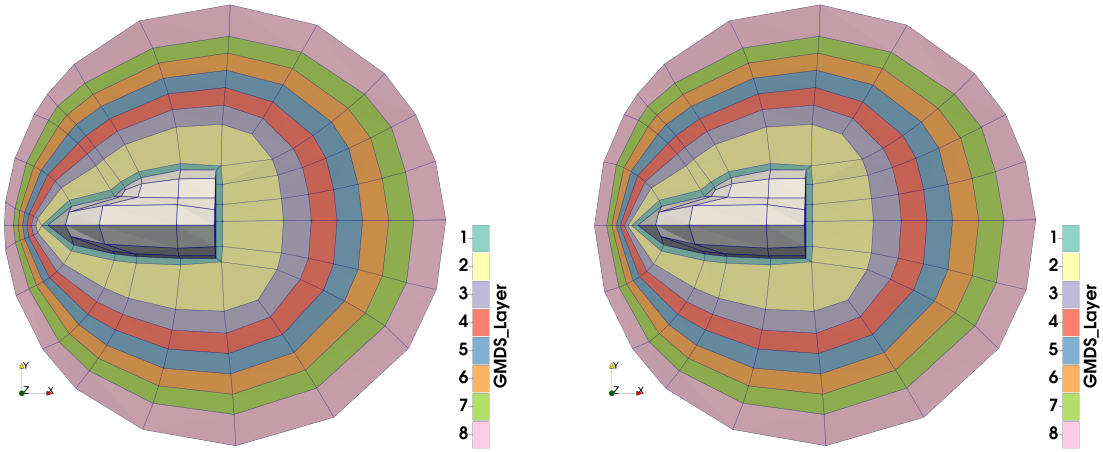
(a) Patterns all over the layer (b) Closed path on the back only (c) Regular layer

Figure 2.36: Three different second block layer topologies generated on the same front.

parameters. We still disable the use of patterns on the first block layer, as for Figure 2.36.a. We also disable the use of patterns on the front part of the geometry (corresponding to the left part of the plot). So now, on the second block layer, there is a closed path of corner edges on the back of the geometry.

Finally, the result plotted in Figure 2.36.c is the second block layer generated around the same geometry and front $\mathcal{F}_0$ as before. However, no patterns are allowed here. As a consequence, the second layer is built with only regular blocks, as presented in Section 2.2.2.

Only three different block topologies are presented in Figure 2.36 on the specific second block layer. It is easily to imagine different block topologies, with for instance the same three variations on the first block layer. This is possible by slightly changing the input parameters of the method.



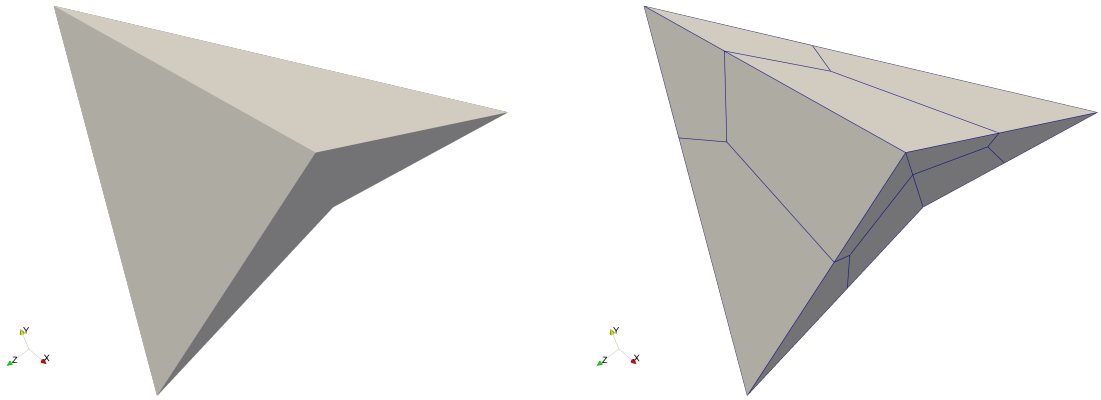
(a) Corresponding blocking of Fig. 2.36.a (b) Corresponding blocking of Fig. 2.36.b

Figure 2.37: Two different blocking topologies generated around Double Ellipsoid vehicle. The block structure is extruded on eight different block layers, each color corresponds to one layer. Blocking (a) corresponds to the full block structure of Figure 2.36.a, and blocking (b) corresponds to Figure 2.36.b. Figure 2.36 focuses on the second block layer, plotted here in yellow ( ).

A clip view along the plan $(O,\vec{X},\vec{Y})$ of the eight different layers of final block structure corresponding to Figure 2.36.a, and b are plotted respectively in Figure 2.37.a, and b. Each color represents a different block layer. The difference between the two block structures presented here are in the second block layer plotted before. As you may see on the left parts of Figure 2.37, this difference impact the whole block structure (and the final number of blocks). This change

of parameters also impact the final number of blocks. The starting front of Figure 2.35.b is made of 76 block $n{-}1$-cell. If the whole block structure is generated in a regular way (i.e., without conflicts), the resulting block structure should be made of 608 hexahedral blocks (this is the case of the final block structure presented in Fig. 2.36.c). In the first case presented here (see Fig. 2.37.a) the resulting block structure is composed of 932 blocks, while this is 790 hexahedral block for the second one (see Fig. 2.37.b). As explained before, the use of patterns can increase (or decrease) the number of blocks on the final structure.

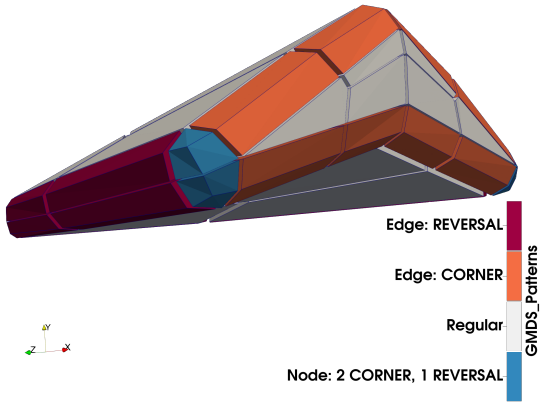
Caretwing



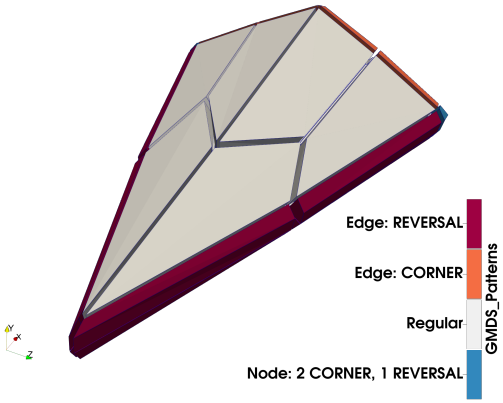
(a) Caretwing geometry (b) Caretwing front $\mathcal{F}_0$

Figure 2.38: Caretwing 3D input geometry (see Appx. B) and block discretization of the vehicle surface.

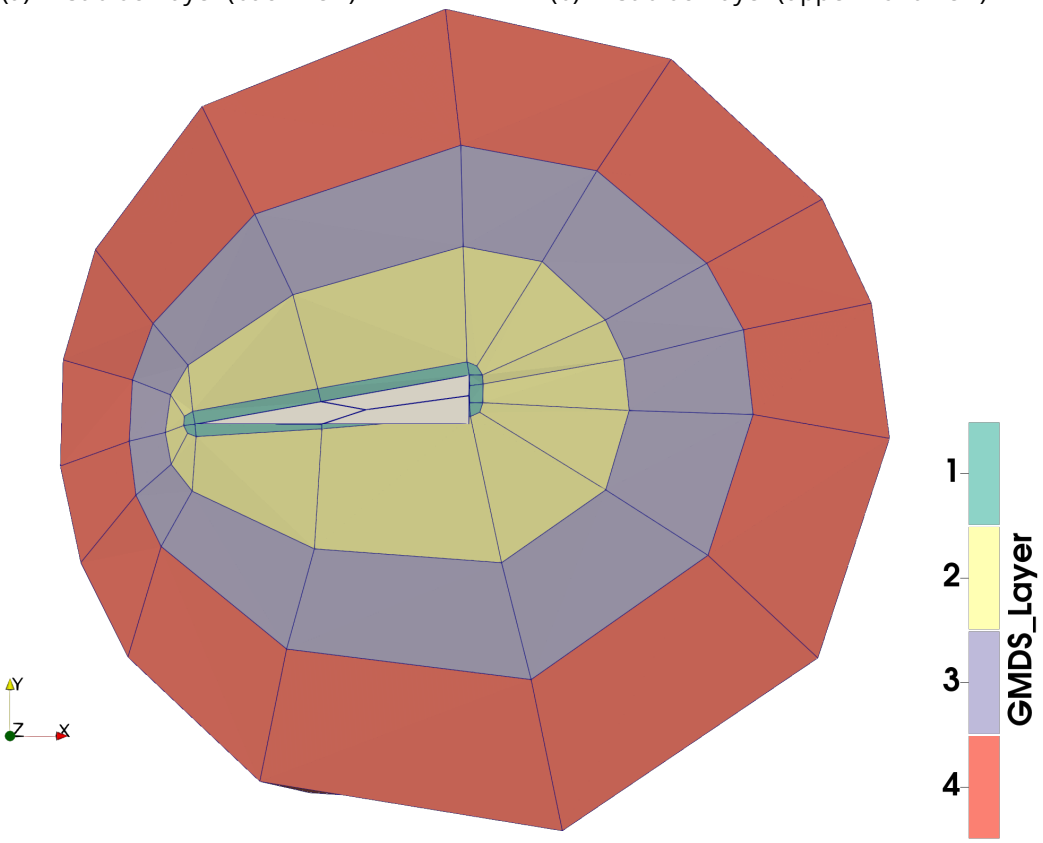
The last geometry presented in this section is a caretwing (see Fig. 2.38.a). We start from the the front discretization given in Figure 2.38.b. The first block layer built on the front $\mathcal{F}_0$ is presented in Figure 2.39.a and b. On this first block layer, there are hexahedral blocks generated by patterns to solve conflicts. The ones created on corner block edges are plotted in orange (■), the ones on reversal block edges are plotted in purple (■). The last pattern illustrated on this test case is on two different blocks corners adjacent to two corner block edges and one reversal block edge each. The hexahedra introduced by this pattern are plotted in blue (■). Once this first layer is generated, the other layers are generated layer by layer. The final block structure, extruded on four layers is represented in Figure 2.39.c in clip view (i.e., only one half of the blocking in plotted here) around the plain representation of the first front $\mathcal{F}_0$ (■). If we take a look at the blocks on the right part of the first layer (■), we may see the inserted blocks.



(a) First block layer (back view)



(b) First block layer (upper front view)



(c) Clip view of the 4 layers of linear blocks

Figure 2.39: Linear blocks extruded on 4 different layers around Caretwing geometry.

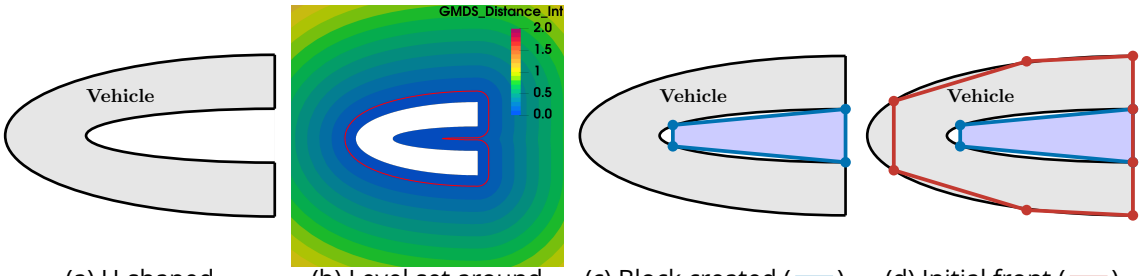
2.6 Perspectives

In this chapter, we presented our work, based on the previous work of Ruiz-Girones et al. [RG11, RGRS12] to generate a linear block structure around a vehicle under several constraints on our physical domain. In the future, we expect to improve this work to get rid of some of the constraints. This chapter opens many perspectives, and among the ones we list bellow, we decide to give specific details about potential future works in this section.

Multi-Vehicle Geometries

One of our constraints on the inputs is we work with a "quasi convex" single-wall vehicle. In some specific cases, we may want to mesh two separated vehicles or handle vehicles with strong non-convex areas (like a U-shaped vehicle, see Fig. 2.40). To do that, we think we can take advantages of the computed distance fields to find particular features in the domain where the level set functions can meet. The idea would be to detect these particular areas, and use an other approach (e.g., medial axis) to first generate a block structure between the two elements. Once the two walls are joined, we can extract a common peel, and use it as the first front for our algorithm. Here are the main stages we expect for this approach:

1. Compute a distance field on physical domain (see Fig. 2.40.b);
2. Detect an iso-line in the field where the front could meet (see red iso-line ( ) in Fig. 2.40.b);
3. Use medial axis method to build a first block-structure to connect the different objects (see Fig. 2.40.c);
4. Use our algorithm to build the full block-structure of the domain, considering the different objects unified as one vehicle on which the algorithm can be applied (see Fig. 2.40.d).



(a) U-shaped vehicle ( ) (b) Level set around vehicle (a) (c) Block created ( ) in non-convex detected area (d) Initial front ( ) for our advancing front algorithm

Figure 2.40: Main stages of pre-treatment approach considered to handle non-convex vehicles.

Improve Distance Fields Computation

The distance field is a core component of the method, as it is through this distance field we compute the vector field to lead the block extrusion direction. If this field is not computed

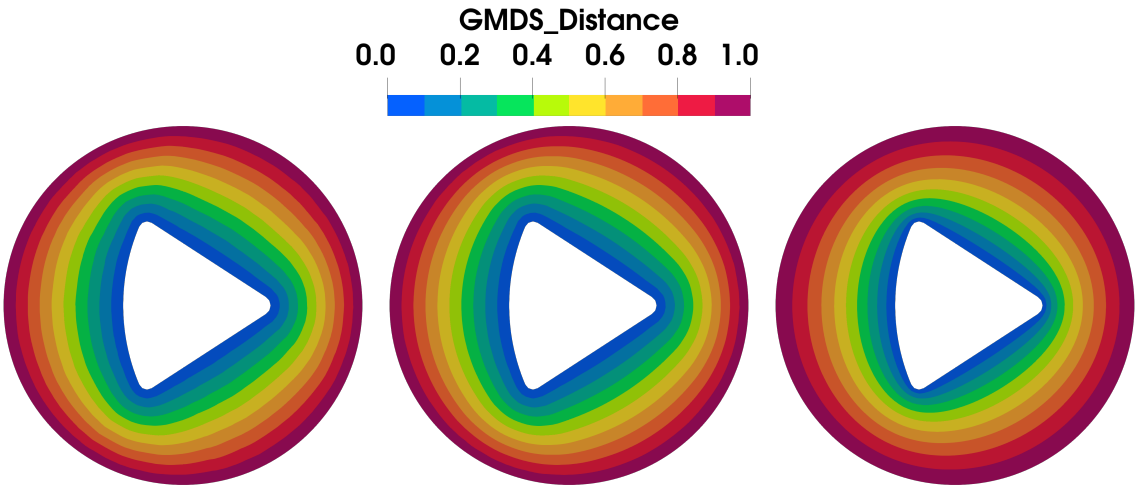
accurately, then the vector field is not and it impacts the result of the algorithm. One solution to improve the distance field computation accuracy is to use a more refined background mesh. However, it increases the time consumption of the distance field computation step.

We expect to work more on the distance field computation to find a method to compute the fields faster and more accurately. We tried three different approach so far:

1. Compute the distance field the same way they did in Ruiz-Gironés et al. [RG11, RGRS12] (see Fig. 2.41.a). But it was time-consuming because our background mesh has to be refined around the vehicle surface, and not accurate enough.

2. Compute the Euclidean distance (see Fig. 2.41.b). It is accurate enough under condition we have a huge amount of points on the background mesh $\mathcal{M}_T$ located on the geometry (and in our case, we need to, because the geometry is represented by the discrete input background mesh of simplices $\mathcal{M}_T$). However, the field computed with this method is not the same as the one before (e.g., in case of multi-objects).

3. Compute the distance field by solving a heat flux system (see Fig. 2.41.c), but this method was also time-consuming.



(a) Fast Marching [Set99] (b) Euclidean distance (c) Heat flux system

Figure 2.41: Levet set computation methods.

Figure 2.41 represents three distance fields computed with the previous methods. In the case of Figure 2.41.a and b, two distance fields are previously computed using the corresponding method; one is the distance from the vehicle surface, and the other is the distance from the far-field boundary. Then, the two distance fields are combined to provide a normalized distance field between 0 and 1. If the two fields look similar, it takes 10 times less time to compute the field using the Euclidean distance than the Fast Marching method in this case (which takes around 0.2s here). In addition, the field of Figure 2.41.a is more noisy. We are fully aware that the problem solved in the two cases is different, and the level set result on different vehicles will differ. In the case of Figure 2.41.c, the heat flux equation is solved with specific boundary conditions, similarly to what is done in [CWW17]. However, even if there is only one field to

compute with this method, it takes around 3 minutes with this simple 2D mesh. Once again, we can see around the boundaries that this field is not the same as the two before.

In a more practical way, the course given by CHARLES DAPOGNY[5] gives some tools for considering more relevant methods for level set computing.

Automatic Generation of Surface Block-Structure

For now, for the 3D case, the first front (the quadrangular block-structure of the vehicle) is taken as an input. We consider it is generated by hand using a dedicated interactive software. We proposed a way to generate the first front automatically in 2D and it could be interesting to have the same in 3D.

The generation of a quadrangular surface blocking is an open field of research, but we can find some methods in the survey of CAMPEN [Cam17]. It would be interesting to consider generating the decomposition surface of the vehicle in an automatic way as an alternative of this needed input. In addition, it would be interesting to propose a method that takes into account the geometric surface, but also takes into consideration the list of patterns on a layer, to provide a suitable input to build a layer.

[5] https://internationalmeshingroundtable.com/imr31/short-courses/#dapogny

Summary

This chapter focused on the generation of a block structure around a vehicle, by an advancing front approach based on the previous work of Ruiz-Girones et al. [RG11, RGRS12]. This approach is atypical because it relies on extruding a block structure layer by layer while advancing front methods usually rely on an extrusion cell by cell. In this chapter, we detailed the different steps of this method. First, the computation of the distance and vector fields on the background mesh to lead the creation of a block layer (see Sec. 2.1). Then, we explained how we compute a regular block layer (see Sec. 2.2), and how to handle conflicts on a block layer in 2D (see Sec. 2.3) and 3D (see Sec. 2.4). If the approach is inspired by the previous work of Ruiz-Gironés et al. [RG11, RGRS12], we performed various minor modifications to the method explained in this chapter.

1. The method was previously used for hexahedral mesh generation while it is used here for block structure generation. We potentially have to deal with anisotropic input block surface (i.e., with important aspect ratio between quadrangular cells) while the pre-meshed surface looks relatively regular in [RG11, RGRS12].
2. We use a vector field to lead the directions for the blocks extrusion.
3. We don't consider a specific type of feature edges called semi-edges in the original work to solve conflicts on a layer, but we introduce instead the additional notion of closed path.

Finally, the last part of this chapter was dedicated to several identified perspectives on this work and potential avenues for future works. At this stage of the algorithm, we have a full linear block structure around the vehicle of interest. We expect now to generate a final mesh by subdividing the blocks using grids. However, as we work with coarse blocks, if we subdivide them we would need to smooth it to align with the geometry. To avoid this smoothing step, we decide to curve our blocks first. In the next chapter, we will propose a curved representation of the blocks and a way to align the curved blocks to the geometry. Then the process of going from the curved blocks to the final mesh will be fully detailed.

Chapter 3

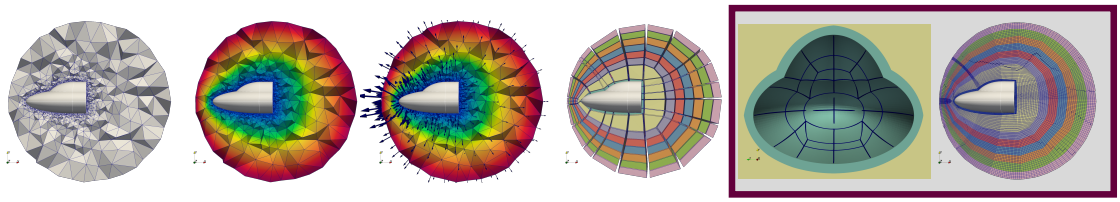
BLOCK STRUCTURE CURVING

At this stage of the method, we have a complete linear block structure. However, as we work with a blocking, there are few elements, and they need to be subdivided. As there are few elements, the geometric surface is not well represented. Besides, a smoothing step to align the final mesh to the physical domain will be mandatory if the linear blocking is discretized using grids. A smoothing algorithm on such a mesh is highly time-consuming, and we expect to avoid this step by first considering our block structure as curved and aligned onto the vehicle surfaces. To do so, two different approaches are considered in this work. The first is through *rp*-adaptivity by considering a global problem on the block structure and solving an optimization problem to improve the quality of the blocks and align them to a surface while carefully controlling the polynomial degree of elements adjacent to the vehicle surface. A second approach is studied by considering our blocks as Bézier blocks to align using local operations. Once the blocks are curved, the discretization is set in each direction to generate the final mesh.

Constributions

- Mittal, K., Dobrev, V. A., Knupp, P., Kolev, T., Ledoux, F., Roche, C., Tomov, V. Z. (2024). *"Mixed-Order Meshes through rp-adaptivity for Surface Fitting to Implicit Geometries."* SIAM International Meshing Roundtable (SIAM IMR 2024). [MDK+24]

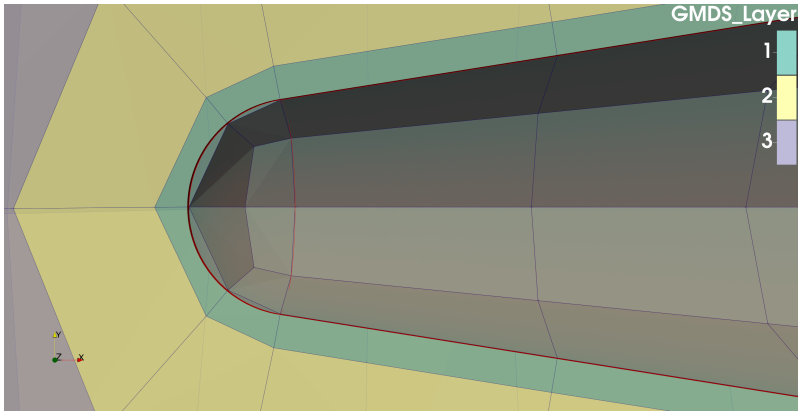
Contents



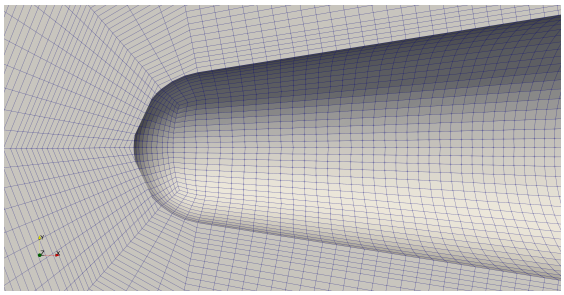
(a) (b) (c) (d) (e) (f)

Figure 3.1: Principal steps of our approach illustrated in 3D. In this chapter, we deal with the block structure curving (e) and the final mesh generation (f) stages.

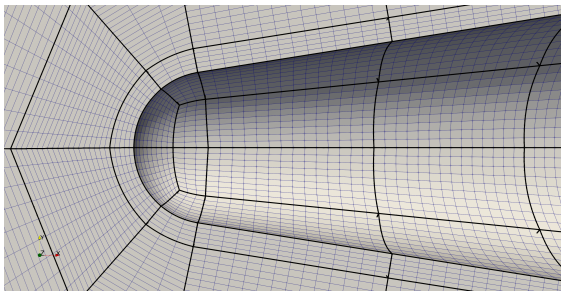
This chapter describes the two final steps of our global approach represented through the pipeline of Figure 3.1. The precedent work of Chapter 2 provides us with a linear block structure, which is composed of a few hexahedral blocks. Consequently, the geometrical error can be too large and difficult to be controlled. If the final mesh is generated on this linear blocking, a time-consuming smoothing step will be needed to reduce the geometrical error.



(a) Linear blocks and vehicle surface ( )



(b) Mesh generated on linear blocks (a)



(c) Mesh generated on previously curved blocks (degree $p=2$)

Figure 3.2: Example of a 3D linear block structure (a) generated around RAM-C II geometry ( ) (see Appx. B), and a mesh generated on this linear block structure (b). Another option of mesh generation on a Bézier block structure (c) is proposed. Curved block edges are plotted using thick black lines ( ).

The geometrical error is the error between the mesh, which is a discrete representation of a physical space, and the boundary of interest. Considering the Figure 3.2.a, the geometrical error may be represented by an error function computed between the red vehicle surface ( ), and the linear blocks ( ). It exists two main way to reduce this geometrical error, the first one is by increasing the discretization, which is not wanted here because we work on a block

structure. The second approach is to curve the cells to approximate the interface, and this is the solution explored in this chapter.

Figure 3.2.a represents a clip view along the plan $(O,\overrightarrow{X},\overrightarrow{Y})$ of a block structure generated using the algorithm presented in previous Chapter 2 around RAM-C II geometry (see Appx. B). Here, the RAM-C II boundary $\partial\Omega_v$ is plotted in red (▬). We generate the mesh on this linear block structure using Transfinite Interpolation in every block. Then, we project the nodes corresponding to the boundaries onto the vehicle surface. The resulting mesh is plotted in Figure 3.2.b. As we may see on the left part of the vehicle, some cells overlap because of the cells' anisotropy. The mesh presented here as an example is coarse but already composed of around $475,000$ n-cells. We adopt a high-order representation of the block structure to avoid dealing with overlapping mesh cells and smoothing the final mesh. The final mesh generated (see Fig. 3.2.c) on the black curved blocks (▬) is obtained without any smoothing on the whole mesh.

Curving the blocks aims to improve the representation of the geometry before generating the final mesh to decrease this smoothing step. In this work, we curve the block structure after its linear generation and then align it with the vehicle surface. To do so, we explored two different methods. The first approach considers the problem as global on the block structure to align it on an implicit surface representation while carefully controlling the polynomial degrees of blocks around the interface (see Sec. 3.1). This method results in a mixed-order block structure adapted to the vehicle surface. This method shows good result for both 2D and 3D meshes, using different topologies of elements on problems of practical interest. However, this solution remains time consuming in 3D. For this reason, we decided to consider a second approach to curve the blocks, which is more local and geometric. To do so, we use a Bézier representation of each hexahedral block [Bez86, DC59, GB12, Feu19] (see Sec. 3.2). The degree of the blocks is uniform in each direction and the same for all the blocks. Using geometric operations, we align the n–1-cells corresponding to the vehicle surface, then propagate those modifications in the block structure to avoid invalid curved blocks. Finally, considering that we have a Bézier representation of blocks that aligns with the geometry, we will discuss how we generate the final mesh (see Sec. 3.3).

3.1 Mixed-Order Mesh Adaptivity for Surface Alignment to Implicit Geometries

The first approach considered to curve the linear block structure was studied with the MFEM team at the Lawrence Livermore National Laboratory (LLNL). MFEM[1] [mfe, AAB+21, BKMT23, AAB+24] is an open-source C++ modular finite element methods library that handles arbitrary high-order finite element meshes and spaces.

Computational analysis with finite element method requires geometrically accurate meshes. There is existing capability from the LLNL to do interface alignment in MFEM framework [AAB+21, DKK+19, BKMT23, AAB+24], but it only works for elements of uniform orders. However, using the same polynomial order for all n-cells away from curvilinear surfaces increases the computational cost of memory and time consumption without increasing accuracy. This work aims

[1] https://github.com/mfem/mfem

to provide a solution to elevate the degree of the elements around the interface we want to align with, but only if necessary. In this work, the interface between the different materials is represented by the 0-isovalue of a level-set function defined on a denser background mesh. Starting from a low-order multi-material mesh, we first compute the alignment error at the interface, then increase the polynomial order of the interface elements if the error is above a desired threshold, and then perform the optimization and alignment algorithm. We repeat those three steps several times until the target fitting error is reached on all the n–1-cells or the maximum order allowed is reached. The results show that we can get the same maximum fitting error around the interface while using this adaptive approach rather than the uniform order one, and save an important amount of degree of freedom. We also show that this method generalizes to meshes with different types of cells (quadrangular, triangular), to 3D meshes, and to problems of practical interest.

Usually, the generation of body-fitted high-order meshes relies on the generation of a linear mesh, curved *a posteriori* by elevating the mesh to a higher polynomial degree and projecting the higher-order nodes corresponding to the boundary of the domain on the surfaces of this physical domain [XSHM13, FP16, MGSP15, GPRPS16, TLR16, PSG16, RGR22, MCM+20]. These approaches focus on uniform-order meshes over the domain. Karman et al. [KKSW22] tackle the mixed-order meshes problem through mesh generation. In this work, the degree of some mesh elements is elevated to represent the geometry accurately. Starting from a valid linear mesh, they elevate the degree of surface elements according to a deviation metric testing process by placing the new nodes on the CAD boundary surfaces. Then, the perturbation on those surfaces is propagated into the volumes using a transfer process. As the n-cells in the boundary layer near the geometry surface are very thin (see Fig. 3.3), curving only the elements adjacent to the surface can lead to overlapping cells in the volume. To avoid this phenomenon, they propagate the element orders in the volume. This results in a gradation of mesh element order into the volume represented by gray shades in Figure 3.3.

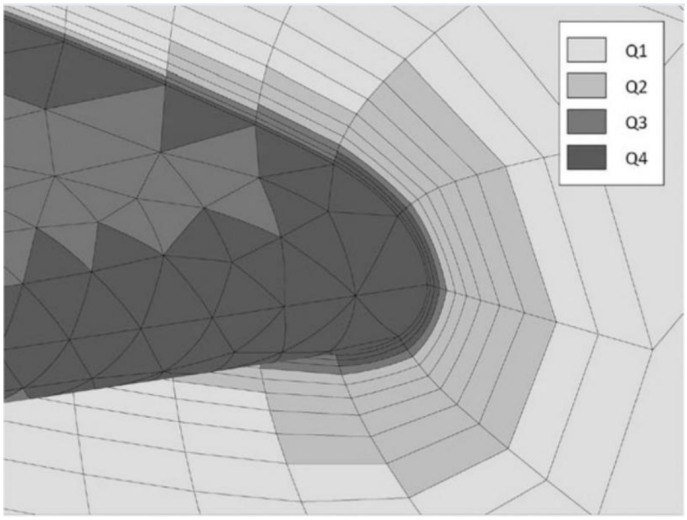


Figure 3.3: Mesh elements order propagated in the volume around the Onera M6 wing [KKSW22]. According to the legend, each gray shade corresponds to a different polynomial degree.

In this section, we introduce the main components of the TMOP-based *r*-adaptivity framework [DKK+19]. Then, we explain the principle of the adaptation of this method to perform surface alignment [BKMT23] and finally go through details of the *p*-adaptivity approach using

this framework [MDK+24] with various practical numerical results.

3.1.1 Target-Matrix Optimization Paradigm (TMOP)

In this section, we present non-exhaustively the mesh representation in MFEM framework, the Target-Matrix Optimization Paradigm (TMOP) for *r*-adaptivity for interface alignment to implicit geometries, and the generalization to mixed-order meshes of arbitrary dimension $n = 2$ or 3. We advise the reader to refer to corresponding works [DKK+19, BKMT23, MDK+24] for more technical details, deeper results analysis, and additional test cases.

Mesh Representation in Finite Element Framework

In this section, the physical domain $\Omega \subset \mathbb{R}^n$ of dimension n is discretized as by a mesh $\mathcal{M}$, which is an union of N_E curved n-cells Ω^e of order p, with $e \in [\![1, N_E]\!]$. The position of an n-cell in the mesh $\mathcal{M}$ is given by a matrix $\mathbf{x}_e$ of size $n \times N_p$ whose columns represent the n-cell Degrees of Freedom (DOFs) coordinates. We select a set of scalar basis functions $\{\bar{w}_i\}_{i=1}^{N_p}$ on the reference element $\bar{\Omega}^e$. Considering $\mathbf{x}_e$, the map between the reference $\bar{\Omega}^e$ and physical Ω^e n-cell (see Fig. 3.4) is given by

$$\mathbf{x}(\bar{\mathbf{x}}) = \Phi_e(\bar{\mathbf{x}}) \equiv \sum_{i=1}^{N_p} \mathbf{x}_{e,i} \bar{w}_i(\bar{\mathbf{x}}),\ \bar{\mathbf{x}} \in \bar{\Omega}^e,\quad \mathbf{x} = \mathbf{x}(\bar{\mathbf{x}}) \in \Omega^e. \tag{3.1}$$

Here, $\mathbf{x}_{e,i}$ is the i-th column of $\mathbf{x}_e$ (i.e., the i-th degree of freedom of the n-cell Ω^e). In this section, $\mathbf{x}$ is the position function defined by Equation (3.1).

The Jacobian A of the mapping function Φ_e given in Equation (3.1) at any reference point $\bar{\mathbf{x}} \in \bar{\Omega}^e$ is defined by

$$A(\bar{\mathbf{x}}) = \frac{\partial \Phi_e}{\partial \bar{\mathbf{x}}}(\bar{\mathbf{x}}) = \sum_{i=1}^{N_p} \mathbf{x}_{e,i} [\nabla \bar{w}_i(\bar{\mathbf{x}})]^T . \tag{3.2}$$

A is a matrix $n \times n$, with

$$A_{a,b}(\bar{\mathbf{x}}) = \frac{\partial x_a(\bar{\mathbf{x}})}{\partial \bar{x}_b} = \sum_{i=1}^{N_p} \mathbf{x}_{i,a} \frac{\partial \bar{w}_i(\bar{\mathbf{x}})}{\partial \bar{x}_b},\quad a, b \in [\![1, n]\!] . \tag{3.3}$$

r-Adaptivity Through TMOP

The Target-Matrix Optimization Paradigm aims to optimize the mesh (i.e.,, modify the mesh node positions without changing the mesh topology) to match a user-defined transformation W in the best way. This transformation corresponds to the target (or ideal) geometric properties of each n-cell of the mesh. This transformation can be written as a combination of four geometric components [Knu22]:

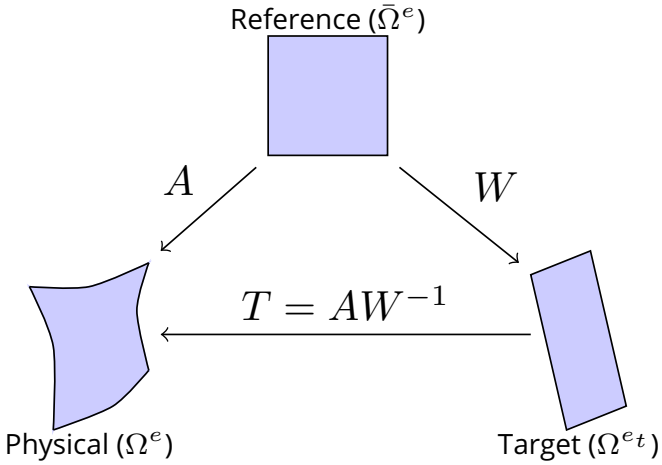


Figure 3.4: Representation of the major Target-Matrix Optimization Paradigm (TMOP) matrices [MDK+24].

$$W_{n\times n} = \underbrace{\zeta}_{\text{[volume]}} \circ \underbrace{R_{n\times n}}_{\text{[rotation]}} \circ \underbrace{Q_{n\times n}}_{\text{[skewness]}} \circ \underbrace{D_{n\times n}}_{\text{[aspect-ratio]}} . \tag{3.4}$$

Using the Equations 3.2 and 3.4, the mapping from the target element to the current coordinates (see Fig. 3.4) is

$$T = AW^{-1}. \tag{3.5}$$

A **quality metric** $\mu(T)$ is introduced to compare the geometric parameters between the transformations A and W. Different metrics are used to optimize different components of T. For instance, $\mu_2 = \frac{|T|^2}{2\det(T)} - 1$ is a *shape* metric that depends on the skewness and aspect ratio only. Here, $|T|$ is the Frobenius norm of T. Numerous different quality metrics are used, and we can find more details in [Knu20, Knu22]. Using the chosen mesh quality metric, the optimization problem is minimizing the global **objective function**

$$F_\mu(\mathbf{x}) = \sum_{\Omega^e \in \mathcal{M}} \int_{\Omega^{e_t}} \mu(T(\mathbf{x}))d\mathbf{x}_t, \tag{3.6}$$

where Ω^{e_t} is the target element corresponding to the element Ω^e (see Fig. 3.4).

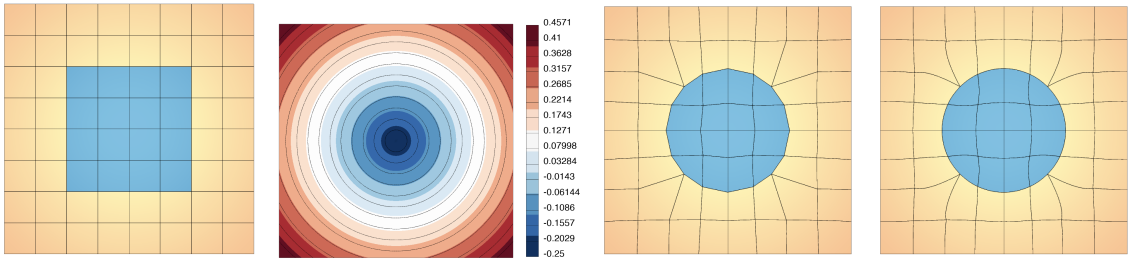
r-Adaptivity for Interface Alignment to Implicit Geometries

In previous work [BKMT23], a **penalty-type term** F_σ is added to the previous formulation (see Eq. 3.6)

$$F(\mathbf{x}) = \underbrace{\sum_{E\in\mathcal{M}} \int_{E_t} \mu(T(\mathbf{x}))d\mathbf{x}_t}_{F_\mu} + \underbrace{w_\sigma \sum_{s\in S} \sigma^2(\mathbf{x}_s)}_{F_\sigma}, \tag{3.7}$$

which aims to align the set S of n–1-cells and n–2-cells to the 0-isovalue of a level set function σ defined on a background mesh $\mathcal{M}_T$. Here, $\mathbf{x}_s$ is the position of each node s of the S n–1-cells set. We refer to the interface as implicit because it is represented by the discrete 0-isovalue

of σ. The penalization weight is given by w_σ. The method handles high-order meshes but only with uniform element order over the mesh (i.e., all the n-cells of the mesh share the same polynomial degree).



(a) Initial mesh (b) Level set function σ (c) Mesh fitted, uniform element order $p=1$ (d) Mesh fitted, uniform element order $p=2$

Figure 3.5: Practical example of interface alignment using TMOP. The initial 2D mesh (a) is composed of coarse quadrangular cells. The domain is separated between two different materials represented with blue (■) and yellow (■) cells. The border between the two materials is given by the 0-isovalue of the level set function σ (b).

Let us consider the practical example of Figure 3.5. The physical square domain is separated between two different materials. The 0-isovalue of the level set function σ (see Fig. 3.5.b) gives the border between the two materials. Here, the interface is a circle. The initial mesh comprises coarse quadrangular cells (see Fig. 3.5.a). This initial mesh is labeled to assign a material value to each quadrangular n-cell. Here, the two different materials are represented in blue (■) and yellow (■). The set S of n–1-cells to align is defined by the edges shared by one blue quadrangular cell on one side and one yellow quadrangular cell on the other. Figure 3.5.c shows a result of mesh optimization with uniform element degrees $p=1$ (i.e., linear cells), while Figure 3.5.d illustrates the result of the mesh optimization with uniform element degrees $p=2$ over the domain. As we may see, the circular interface is better approximated in the second case, with elements of degrees $p=2$. However, far from the interface (e.g., the top or bottom corners), the cells are shaped the same, but the number of degree of freedom necessary to represent those cells is higher in case $p=2$. So, we potentially need to increase the polynomial degree of mesh elements in order to compute the interface accurately. Still, we want to have a low polynomial degree where it is not mandatory to avoid a large number of degrees of freedom.

3.1.2 *rp*-Adaptivity for Interface Alignment

This part aims to adapt the polynomial degree of each n-cell of the mesh in order to represent the interface in an accurate way with high-order cells in areas of high curvatures and low-order cells in areas far from the curvatures. This results in a mesh with fewer degrees of freedom more suitable for fast numerical analysis.

The process is fully explained in Algorithm 8. Given a mesh $\mathcal{M}$ to optimize, and a background mesh $\mathcal{M}_T$ with a level set function σ whose isovalue 0 represents an interface we want to be aligned with, the first step is to label the mesh $\mathcal{M}$ to assign a material value to each n-cell of the mesh (see previous work [BKMT23] for more information). We compute the set

Algorithm 8 Successive Polynomial Order Increase

Require: Mesh to optimize $\mathcal{M}$, background mesh $\mathcal{M}_T$, level set function σ, maximum polynomial degree of elements p_{max}, polynomial degree increment Δp, maximal fitting error e_{max}
Ensure: Fitted mesh $\mathcal{M}$

```
1: M ← materialsLabeling()                                   ▷ (see [BKMT23])
2: M ← interfaceAlignment(M_T,σ)                             ▷ TMOP
3: for i=1, p_max−1 do
4:   for all n−1-cells f ∈ S do
5:     e_f ← computeAlignmentError(f, σ, M_T)
6:     if e_f > e_max then
7:       el ← getElements(f)   ▷ Get the two n-cells adjacent to the n−1-cell f
8:       el_0 ← setElementOrder(el_0.order+Δp)
9:       el_1 ← setElementOrder(el_1.order+Δp)
10:    end if
11:  end for
12:  M ← interfaceAlignment(M_T,σ)                           ▷ TMOP
13: end for
```

S of n–1-cells shared by two n-cells that belong to two different materials[2]. Then, we successively increase the polynomial degree of the n–1-cells by Δp if the threshold error e_{max} of alignment over the n–1-cell is not reached. At each step of the main loop on polynomial orders (see Algo. 8, l. 3), the interface alignment is performed after the elevation of polynomial degrees (see Algo. 8, l. 11). This is on this mixed-order aligned mesh that the alignment error is computed in the next step of the loop (see Algo. 8, l. 4).

As the problem formulated in Equation (3.7) aims to align the DOFs of the mesh to the interface to compute the error alignment (see Algo. 8, l. 4) we need to check the error at more points than only the error at each DOFs. To do so, we compute the integrated error e_f over the n–1-cell f to verify the alignment of the n–1-cell to the interface. This integrated error

$$e_f = ||\sigma||^2_{\mathcal{L}^2,f} = \int_f \sigma^2(\mathbf{x}) \tag{3.8}$$

is the squared $\mathcal{L}^2$ norm of the level-set function σ on that n–1-cell. This integration requires interpolating the discrete level set function σ at the quadrature points associated with the n–1-cell. This process is detailed in [MDK+24].

Let us note that a n–1-cell shared by two adjacent n-cells is constrained by the lowest polynomial degree of the two adjacent n-cells (see Fig. 3.6). So if a n–1-cell is located at the interface between two different materials, its two adjacent n-cells share the same polynomial degree.

Figure 3.7 illustrates a result of the Algorithm 8. Here, we want to align the initial 2D mesh (see Fig. 3.7.a) to a rounded-square interface represented by the 0-isovalue of the level set function σ (see Fig. 3.7.b). The input mesh is linear (i.e., uniform order $p = 1$), and the

[2] A n–1-cell is adjacent to one or maximum of two n-cells. In the first case, the n–1-cell borders the domain. In the second case, the n–1-cell is in the inner part of the mesh.

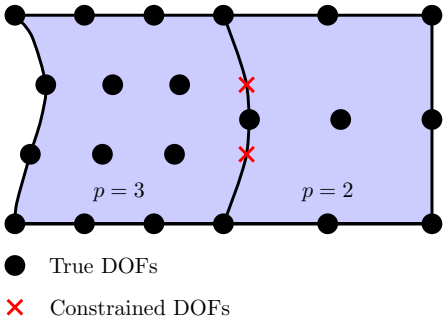
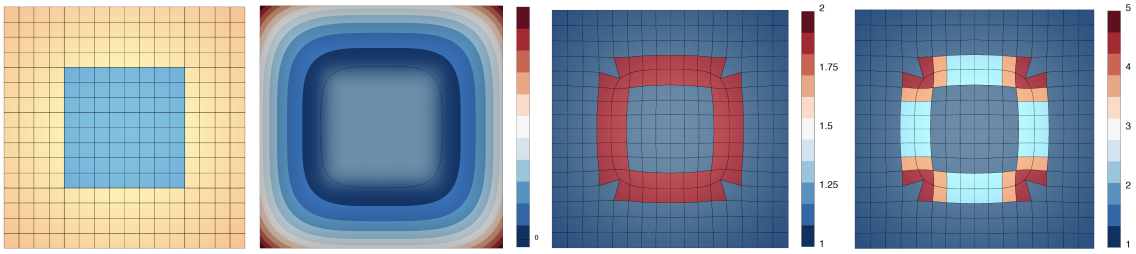


Figure 3.6: Constrained Degrees of Freedom (DOFs) on a n–1-cell shared by two n-cells with different polynomial degrees [MDK+24].

maximal degree allowed is $p_{max} = 5$. The increase polynomial order step for this test case is $\Delta p = 1$. After one iteration of the main loop (see Algo. 8, l. 3), the polynomial orders of the n-cells are plotted in Figure 3.7.c. All the n-cells far from the interface remain at a low polynomial degree $p = 1$, while the polynomial order of every n-cells around the interface is elevated to $p = 2$. The final aligned mesh obtained after 4 steps of the loop is plotted in Figure 3.7.d. As we may see, the final polynomial degree far from the interface remains low while it increases in the area of high curvature. This mesh is composed of elements of degree $p = 2$, 3, or 5 around the interface, and of linear n-cells far from it.



(a) Initial mesh (b) Level set function σ (c) Polynomial orders of aligned mesh after 1 step (d) Polynomial orders of aligned mesh after 4 steps

Figure 3.7: Example of 2D mixed-order mesh alignment to the implicit interface (b). The mesh is composed of n-cells of degree $p = 2$ around the interface and $p = 1$ everywhere else after one iteration (c). After several iterations, the polynomial degree around the interface varies from $p = 2$ to $p = 5$ in areas of high curvatures (d).

Regarding the final number of Degrees of Freedom, we can see Figure 3.8 where five different meshes are compared. The five 2D meshes, corresponding to the four red points (●) and the blue square (■) have the same number of n-cells. This graph represents the maximum alignment error over a n–1-cell of the mesh, according to the number of nodes of the mesh. The red points (●) correspond to meshes with uniform order over the mesh. At the same time, the blue value represents the mixed-order mesh of Figure 3.7.d. As expected, with the mixed-order mesh, we approximate the interface with the same accuracy as with uniform high-order meshes, and we save a large number of degrees of freedom.

The time-consuming step of the algorithm corresponds to the interface alignment part (i.e., call to TMOP algorithm, see Alg. 8, l. 13), and the more the mesh contains degrees of freedoms (i.e., nodes), the more it is time consuming. Indeed, each time we raise the polynomial degree of an element, it is mandatory to align the mesh to the interface again. However, the

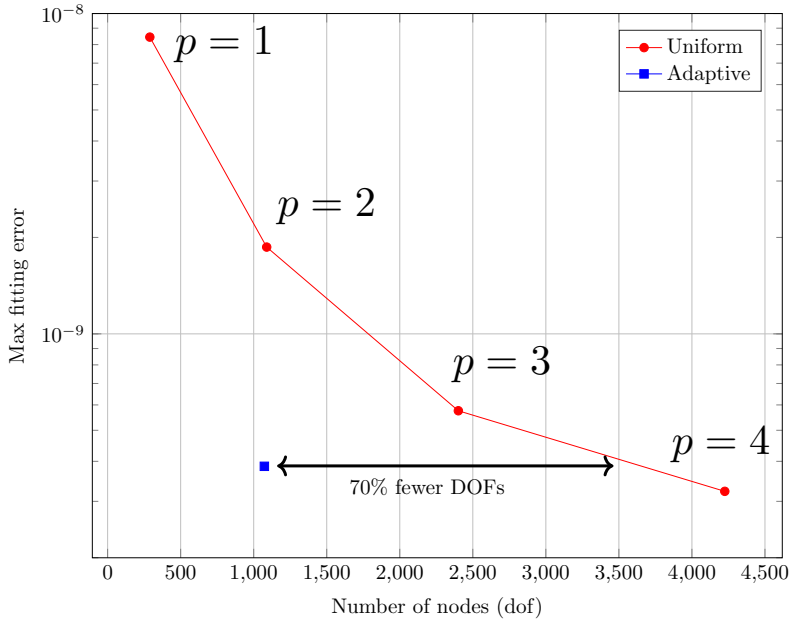


Figure 3.8: Maximal interpolated error comparison between uniform and mixed-order mesh aligned to an implicit interface. The integrated error of the mixed-order mesh Figure 3.7.c is plotted here with the blue square (■).

fitting is not required if the polynomial degree is decreased. That is why the strategy proposed in Algorithm 9 consists in increasing first the polynomial degree of every n-cells around the interface of an arbitrary value, performing the fitting algorithm, and then in decreasing step by step the degrees. For each n–1-cell f of polynomial degree p_f of the interface, we create two artificial elements of degree $p < p_f$ on the two n-cells of degree p_f. We use those two fake elements to compute the potential error between the interface and the n–1-cell (see Alg. 9, l. 4). If the local error is below the input threshold maximal error, then the polynomial degree of the two elements is decreased (see Alg. 9, l. 5–9).

In addition to Algorithms 8 and 9 presented before, we can decide to control the gap of polynomial degrees between two adjacent n-cells. This way, once the cells' orders are set around the interface, the degrees are propagated in the domain. As the n–1-cell shared by two adjacent n-cells is constrained by the lower degree (see Fig. 3.6), if the polynomial degree difference between two adjacent n-cells is important, the n–1-cell may be over-constrained. As mentioned in the previous work of Karman et al. [KKSW22], this transition is of practical interest for anisotropic mesh cells around the interface alignment.

3.1.3 Application to Block Structures

In MFEM, the method presented in this section is used to adapt high-order meshes. In this manuscript, our framework uses it on the linear block structures generated using the method of Chapter 2 to smooth it and align it with the vehicle surface. To use the method, we consider our block structure as a coarse mesh of quadrangular (2D) or hexahedral (3D) cells. As the block structure will be refined when generating the final mesh, we are less strict about the maximal alignment error allowed than before. The background mesh $\mathcal{M}_T$ on which the level set function σ is represented is roughly the same as the one used in Chapter 2. However,

Algorithm 9 Successive Polynomial Order Decrease

Require: Mesh $\mathcal{M}$, background mesh $\mathcal{M}_T$, level set function σ, minimal polynomial order p_{min}, maximal polynomial order p_{max}
Ensure: Fitted mesh $\mathcal{M}$

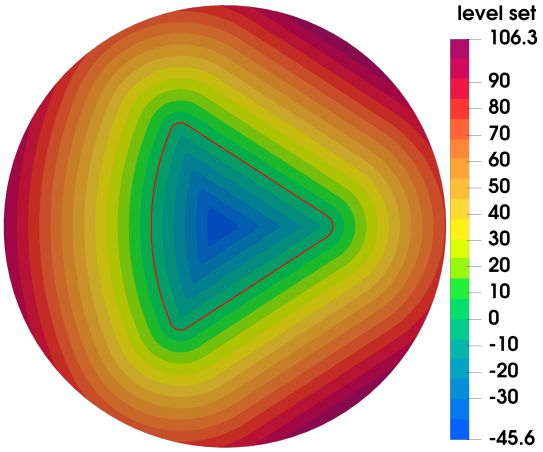
$\mathcal{M} \leftarrow$ materialsLabeling() ▷ (see [BKMT23])
for all n–1-cells f at the interface of two different materials **do**
  el $\leftarrow$ getElements(f) ▷ Get the two n-cells adjacent to n–1-cell f
  $el_0 \leftarrow$ setElementOrder(p_{max})
  $el_1 \leftarrow$ setElementOrder(p_{max})
end for
$\mathcal{M} \leftarrow$ interfaceAlignment($\mathcal{M}_T$, σ) ▷ TMOP
for all n–1-cells f at the interface of two different materials **do**
  for $i = 1, p_{min}-1$ **do**
    $e \leftarrow$ computeAlignmentError(f, σ, $\mathcal{M}_T$, $p_f - i$)
    if $e < e_{max}$ **then**
      el $\leftarrow$ getElements(f) ▷ Get the two elements adjacent to the face f
      $el_0 \leftarrow$ setElementOrder(el_0.order-1)
      $el_1 \leftarrow$ setElementOrder(el_1.order-1)
    end if
  end for
end for

we need to add elements inside the vehicle and compute the value of the level set on those elements, too. To calculate the value of σ, we consider all the nodes on the vehicle surface at distance 0; then, we compute the Euclidean distance of the inner nodes to this surface. We need to slightly modify our block structure to provide suitable input to the MFEM framework. This pre-processing differs depending on the block structure dimension n, which is explained below in the dedicated 2D and 3D frames.

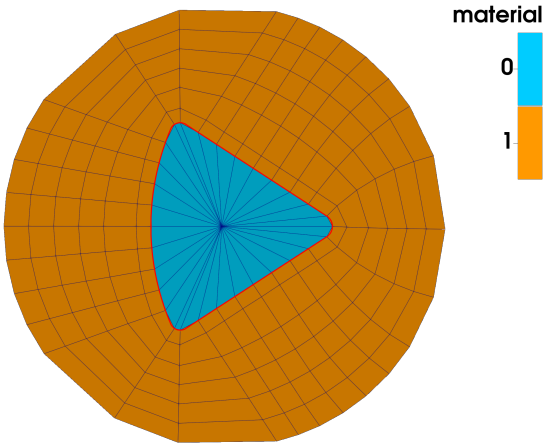
2D

Ghost triangular block-cells are added to mesh the inside of the vehicle in order to use the interface alignment algorithm (see Fig. 3.9.b). Those cells are built only to align the block edges on the surface. They are removed afterward.

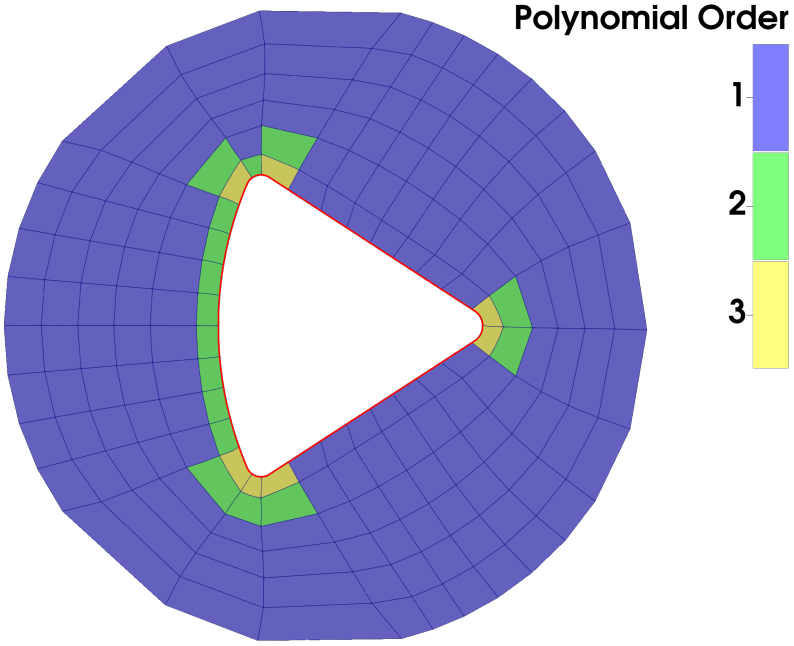
Figure 3.9 presents a 2D linear block structure made of quadrangular orange cells (▬) (see Fig. 3.9.b) generated with the method presented in Chapter 2. Here, we adapt the block structure by adding the blue triangular cells (▬) to give a suitable input to MFEM. In this example, the vehicle surface is represented by the red iso-line (▬) in the level set field of Figure 3.9.a. The resulting adapted block structured is presented in Figure 3.9.c with the corresponding polynomial degree of each block cell.



(a) Level set function σ with iso-value 0 ( ) on background mesh $\mathcal{M}_T$



(b) Inital blocks, with triangular *ghost* block cells ( )

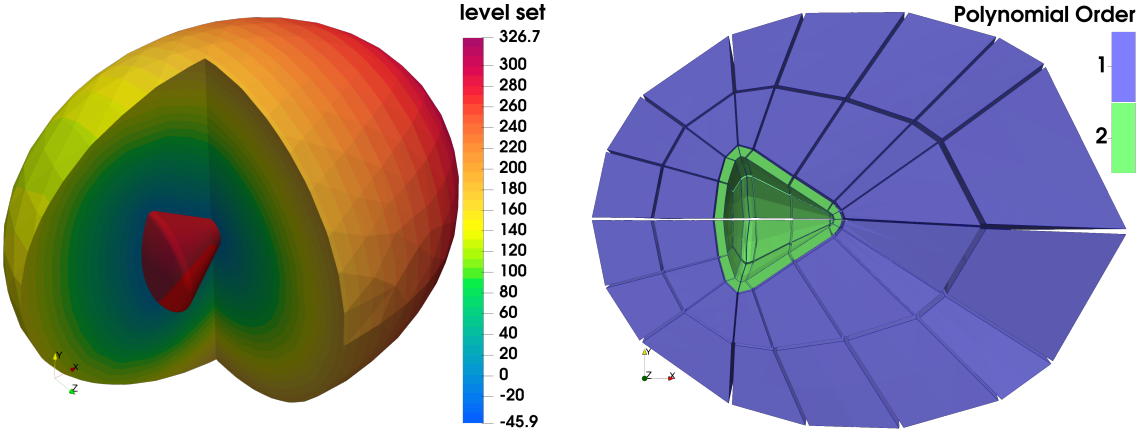


(c) Optimized final block orders, aligned to the red interface ( )

Figure 3.9: Practical example of *rp*-adaptivity on a block structure generated around Apollo 2D vehicle using the pipeline presented in Chapter 2.

3D

As MFEM does not support high-order pyramids, we can not extend the 2D naive approach to 3D. Instead, a very thin *ghost* layer of hexahedral block-cells is extruded inward the vehicle in order to use the interface alignment algorithm. Those cells are built only to align the block edges on the surface. They are removed, then. This method illustrates the framework capabilities of our 3D problem of practical interest. However, in the case of complex geometries, it may be challenging to generate this inward hexahedral blocks layer.



(a) Level set function σ with iso-value 0 (—) on background mesh $\mathcal{M}_T$

(b) Optimized final block orders, aligned to the red interface (—) (a)

Figure 3.10: Practical example of *rp*-adaptivity of a block structure generated around Apollo 3D vehicle.

The example of Figure 3.10 illustrates the possibilities of the algorithm for 3D problems of practical interest. The linear block structure is generated using the method of Chapter 2 and made of 224 hexahedral blocks, including an extruded layer of *ghost* blocks inward of the vehicle. As the block structure will be refined later, we do not search for a very low integrated error as in [MDK+24]. Diverse 2D and 3D mesh alignment examples are presented in [MDK+24].

For the two presented cases in Figure 3.9 and 3.10, the algorithm is performed to align the cells to the vehicle surface. We can also imagine performing the same algorithm to align the cells with the outer boundary.

3.2 Curving of the Structure Using Bézier Elements

The previous work of Section 3.1 provides an accurate solution to curve an existing linear block structure to align it to an interface. The method shows good results for both 2D and 3D test cases. However, this solution remains time-consuming, especially for 3D cases. For this reason, we propose a second way to curve a block structure in this section. This method relies on a Bézier representation of our blocks. It is less time-consuming than the approach presented in Section 3.1 but provides less numerical accurate results. This is not a problem in our case, as the approximation of the vehicle surface remains sufficient to generate a mesh on it.

In this part, we present the Bézier block representation we work with and how we manage to approximate the boundaries using these Bézier blocks.

3.2.1 Bézier Elements

This section presents the Bézier elements we use in this work in a non-exhaustive manner. The reader can find more details in the work of FEUILLET [Feu19], where the high-order formulations are exposed on various other elements, with complete definitions and properties of the elements. The main advantage of Bézier elements is that they are well-studied simple analytical functions. Multiple works rely on those functions; they are well-documented, many good properties of the functions and their derivatives are known, and algorithms are easily available.

Bézier Curve

A **Bézier curve** γ of order p is defined by a set of $p+1$ **control points** $(P_0, ..., P_p)$. The control points are of arbitrary dimension. In the parametric space, we can represent the physical curve by the relation

$$\gamma(u) = \sum_{i=0}^{p} B_i^p(u) P_i, \tag{3.9}$$

where $u \in [0, 1]$, and B_i^p is the $i-th$ **Bernstein polynomial** of degree p, defined by:

$$B_i^p(u) = \binom{p}{i} u^i (1-u)^{p-i}; \tag{3.10}$$

where $\binom{p}{i} = \frac{p!}{i!(p-i)!}$ is a **binomial coefficient**.

(a) Order $p=2$ (b) Order $p=3$

Figure 3.11: Example of Bézier curves (▬) of degree $p=2$ (a), and of degree $p=3$ (b), with P_i the control points (●) of each curve.

Let us notice that $P_0, ..., P_p$ are named control points because they control the shape of the curve. However, the curve does not interpolate those points, except for $\gamma(u=0) = P_0$, and $\gamma(u=1) = P_p$ (see Fig. 3.11). For $u \in [0, 1]$,

$$\frac{\partial B_i^p}{\partial u}(u) = \binom{p}{i} (i u^{i-1}(1-u)^{p-1} - (p-i)u^i(1-u)^{p-i-1}). \tag{3.11}$$

One particular Bézier curve property of interest is the tangent to the curve extremities (see Eq. 3.11). A Bézier curve of degree $p > 1$ is tangent to vector $\overrightarrow{P_0P_1}$ at interpolated control point P_0, and tangent to vector $\overrightarrow{P_pP_{p-1}}$ at interpolated control point P_p (see Fig. 3.11).

The DE CASTELJAU'S method in Algorithm 10 is an optimized recursive algorithm proposed to evaluate the Bézier curve at the parametric point $u \in [0, 1]$.

Algorithm 10 DeCasteljau

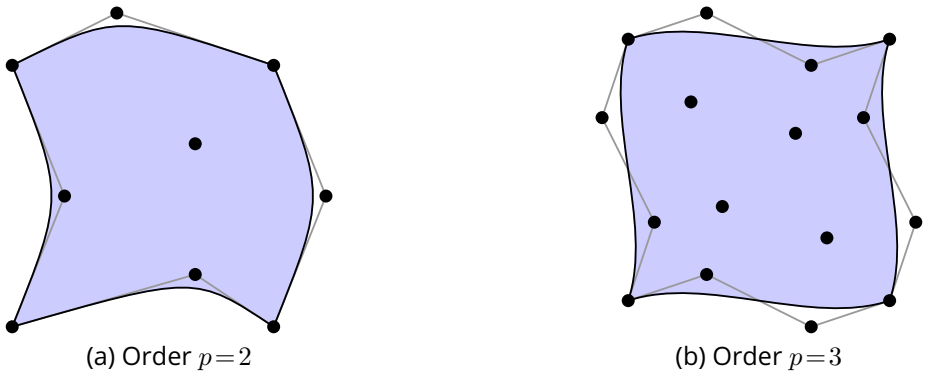
Require: Degree $p \in \mathbb{N}^*$, Control Points P_i, Parameter $u \in [0, 1]$
Ensure: $P(u)$
1: **if** $p = 0$ **then**
2: **return** P_0
3: **else**
4: **for all** $i \in \{0, p-1\}$ **do**
5: $\hat{P}_i = (1-u)P_i + uP_{i+1}$
6: **end for**
7: **return** $DeCasteljau(p-1, \hat{P}_i, u)$
8: **end if**

Bézier Quadrangle

It is possible to define a Bézier quadrangle only with four Bézier curves, considering that the quadrangle is defined by its four boundaries. However, as we want to consider our blocks as Bézier elements, we expect to use the Bézier definition of the elements to generate the inner mesh of each block. For this reason, we also consider the inner control points of each Bézier element.

Let us note $P_{p,m}$ the matrix of size $p \times m$ of the control points. Then we can express the Bézier quadrangle by its control points in the parametric space:

$$\forall u, v \in [0, 1], \quad \gamma(u, v) = \sum_{i=0}^{p} \sum_{j=0}^{m} B_i^p(u), B_j^m(v) P_{i,j}. \tag{3.12}$$



(a) Order $p=2$ (b) Order $p=3$

Figure 3.12: Example of a Bézier quadrangles ( ) of degree $p=2$ (a), and of degree $p=3$ (b), with the corresponding control points (●).

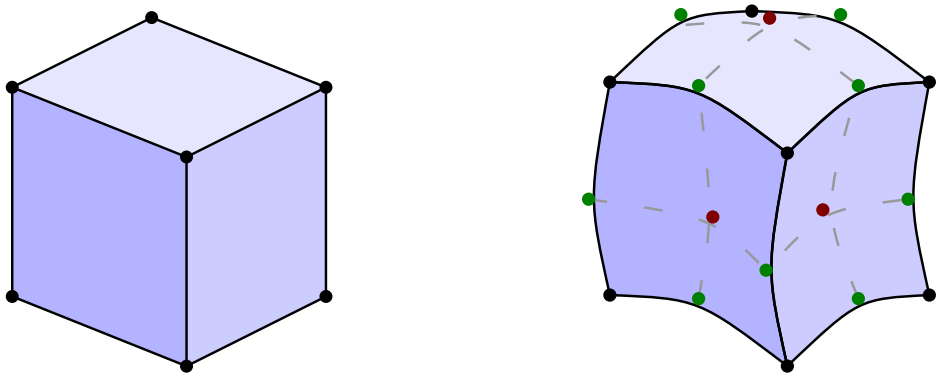
The restriction of a Bézier quadrangle to a boundary is a Bézier curve. Figure 3.12 shows examples of Bézier quadrangles of uniform degree per direction (i.e., degree $p \times p$).

Bézier Hexahedron

Let us note $P_{p,m,l}$ the matrix of size $p \times m \times l$ of the control points. Then, we can express the Bézier hexahedron (see Fig. 3.13) by its control points in the parametric space:

$$\forall u, v, w \in [0,1], \gamma(u,v,w) = \sum_{i=0}^{p} \sum_{j=0}^{m} \sum_{k=0}^{l} B_i^p(u), B_j^m(v) B_k^l(w) P_{i,j,k}, \tag{3.13}$$

where $B_l(u)$ is the l-th order Bernstein polynomial.



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

Figure 3.13: 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 here in the volume.

Theoretically, we can imagine having a different order for a Bézier quadrangular or hexahedral element in each direction. However, for a sake of simplicity, this work considers that the degree is the same in each direction. Through misuse of language, we will refer to an element of degree p in each direction (i.e., a Bézier hexahedral element of degree $p \times p \times p$) as an element of degree p.

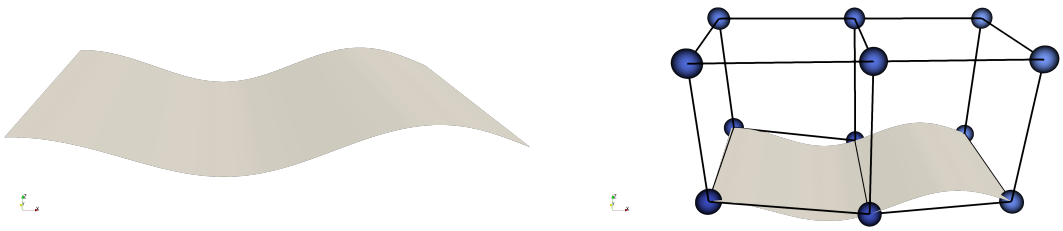
3.2.2 Block Curving to Interpolate Boundaries

The first step is to compute the ideal order needed to represent the geometry. To do so, we select the set of n–1-cells corresponding to the front $\mathcal{F}_0$ located on the geometry, and the front $\mathcal{F}_{N_L}$ (the last front, corresponding to the far-field).

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

2D

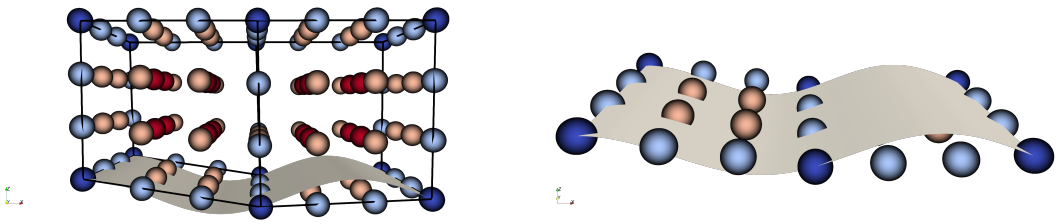
For a better understanding, the reader may refer to Appendix F where the process is explained in 2D for curving one Bézier edge to approximate the geometry. In this Appendix, different methods are explored to curve a Bézier edge, and the impact on the final curved edge is studied.



(a) Part of the vehicle surface (b) 2 blocks of the linear block structure

Figure 3.14: Practical example of a part on the vehicle surface (a) and two linear hexahedral blocks previously generated (b). The block corners are here represented in dark blue (●).

The process is geometric and local and can be decomposed as follows. Considering a small part of the surface such as Figure 3.14.a which is discretized by the two linear hexahedral blocks of Figure 3.14.b. We elevate the degree of all the blocks of the block structure to the same value p (e.g., $p=3$ in Fig. 3.15.a). The $(p+1)^3$ control points of each block are initially set uniformly on the linear blocks. Here, the control points corresponding to the hexahedral block corners are plotted in dark blue (●). The control points over the block edges are plotted in light blue (●), the control points over the faces are in orange (●), and the control points in the volume are plotted in red (●).



(a) Initial control points (elements of degree $p=3$), set uniformly on the linear blocks (b) (b) Initial control points on the n–1-cells corresponding to the geometric surface

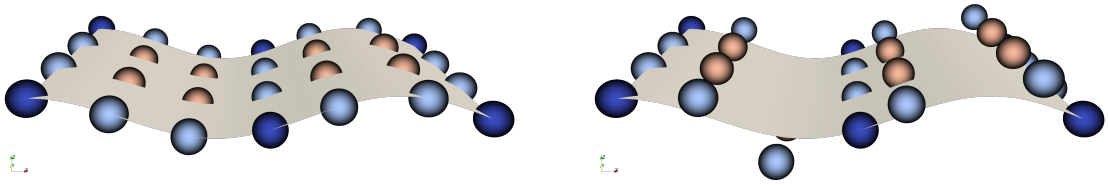
Figure 3.15: The two hexahedral blocks of Figure 3.14.a are represented here with Bézier hexahedral blocks of degree $p=3$ (a) (i.e., degree $p=3$ in each direction). The control points corresponding to the vehicle surface are represented in (b). The control points plotted in light blue (●) are the control points over the block edges, in orange (●) over the block faces, and in red (●) in the volume.

The initial control points $\{P_{i,j}\}_{i,j\in[\![0,p]\!]}$ of the n–1-cells corresponding to the vehicle wall (see Fig. 3.15.b) are projected on the geometric surface of Figure 3.14.a. We note $A_{i,j}$ the projected position of control point $P_{i,j}$ onto the vehicle surface $\partial\Omega_v$. This projected set of points is presented in Figure 3.16.a. We consider this set of points as the physical positions to interpolate with our Bézier surfaces. Using those positions, we compute the physical coordinates of the corresponding control points $P_{i,j}$. To do so, we invert the $(p+1)^2$ equations system

built using the Bézier expression over a quadrangular surface (see Eq. (3.12)). We consider the control points as uniform in the parametric space. With that say, we can express the system as

$$\forall i,j \in [\![0,p]\!], \quad A_{i,j} = \gamma\left(u = \frac{i}{p}, v = \frac{j}{p}\right) = \sum_{i=0}^{p}\sum_{j=0}^{p} B_i^p(u), B_j^p(v) P_{i,j}, \tag{3.14}$$

with P, the matrix of control points we want to compute, and $\gamma(u,v)$ is determined by the physical interpolated positions of Figure 3.16.a. To interpolate the positions of Figure 3.16.a, the resulting control points computed are the ones of Figure 3.16.b. Once those positions are computed, we re-compute the positions of the inner control points using Transfinite Interpolation to avoid overlapping curved blocks. The final control points of the 2 Bézier blocks considered are represented in Figure 3.17.a.

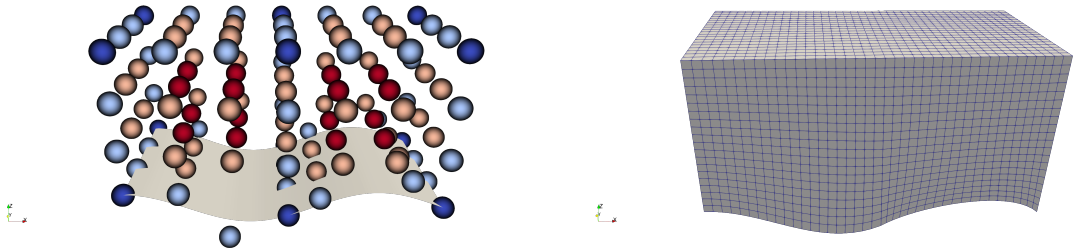


(a) Points to interpolate on the geometric surface, computed as the projection of the control points of (a) onto the geometry

(b) Control points computed to interpolate positions of (a)

Figure 3.16: The control points corresponding to the vehicle surface (see Fig. 3.15.b) are projected onto the vehicle surface. This set of points (a) are the positions we want our Bézier surfaces b_s to interpolate. The Bézier surfaces control points positions (b) are computed to interpolate the physical positions (a).

As the first block layer can be very thin, we add the same offset as the control points moved in Figure 3.16.b to the opposite n–1-cells in the blocks. This step is not illustrated here and could be included between Figure 3.16.b and Figure 3.17.a. This step avoids overlapping curved blocks for anisotropic blocks. Using the parametric space of those two Bézier blocks, an example of the resulting mesh is represented in Figure 3.17.b. As we may notice, for the right part block, this process may not be sufficient to approximate accurately the geometric surface.



(a) Control points of the 2 Bézier blocks

(b) Final mesh generated on the 2 Bézier blocks (a)

Figure 3.17: Final control points of the 2 hexahedral Bézier blocks (a) with a practical example of the final mesh generated on those blocks (b).

Now comes the legitimate degree p value choice question. In this work, we perform a preliminary study on the linear block structure to pick the degree p. The process is performed

Algorithm 11 Block Structure Bézier Polynomial Degree Computation

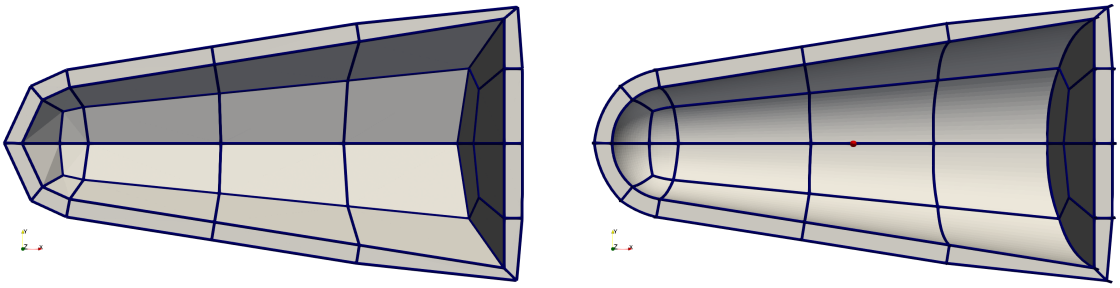
Require: Maximal polynomial degree p_{max}, threshold error e_{max}, set of block n–1-cells S located onto the vehicle wall, geometric vehicle surface $\partial\Omega_v$
Ensure: Bézier blocks polynomial degree p

```
for all block n–1-cell s ∈ S do
    p_s ← 1                                   ▷ init polynomial degree over n–1-cell s
    b_s ← buildBézierCell(p_s)                ▷ computes control points to interpolate ∂Ω_v positions
    e_s ← error(b_s, ∂Ω_v)                    ▷ error between Bézier cell of degree p_s and ∂Ω_v
    if e_s > e_max and p_s < p_max then
        p_s ← p_s + 1
    else
        if p_s > p then
            p ← p_s                           ▷ Eq. (3.15)
        end if
    end if
end for
```

as presented in Algorithm 11. We first select the set S of Bézier block n–1-cells (i.e., block edges if $n = 2$, and block surfaces if $n = 3$) located onto the surface geometry. We choose a maximal approximation error threshold e_{max} between a quadrangular Bézier surface and the physical vehicle boundary. Using this, for each quadrangular Bézier surface $b_s \in S$, we incrementally increase the degree of the Bézier surface $p_s = 1, 2, ..., p_{max}$ until the maximal degree p_{max} allowed or the maximal error e_{max} is reached. The polynomial degree needed for the local Bézier surface b_s is p_s. Then the uniform order over the block structure is chosen as the maximal polynomial degree needed over the Bézier surfaces corresponding to the vehicle surface

$$p = \max_{b_s \in S} \{p_s\}. \tag{3.15}$$

In this part, the error e_s over a Bézier surface b_s is computed as the maximal deviation between a sample of points $a_{i,j} = b_s(u_i, v_j)$, with $i, j \in [\![0, N_T]\!]$ and u_i, v_j chosen uniformly over the parametric space, and their projection onto the surface geometry. The sample number of comparison points chosen N_T is arbitrary.



(a) Linear first block layer (b) Curved first block layer

Figure 3.18: Example of a first linear block layer (a) generated around RAM-C II curved using Bézier blocks of degree $p=2$ to align to the vehicle wall (b).

The result of this algorithm is illustrated in Figure 3.18 on the first block layer. The linear first block layer (see Fig. 3.18.a) generated using the algorithm presented in Chapter 2 is curved *a posteriori* using Bézier blocks of degree $p=2$. The final aligned high-order first block layer is plotted in Figure 3.18.b. For visualization purposes, the layer is represented using a clip view here.

3.3 Mesh Generation from the Curved Block Structure

Once the curved block structure is built, we generate the final mesh. It requires assigning the proper discretization to every block's edge and discretizing each block with the appropriate regular grid. The boundary layer is meshed considering strong constraints about wall-orthogonality and aspect ratio. In this section, we consider that the block structure is represented using Bézier elements (see Sec. 3.2).

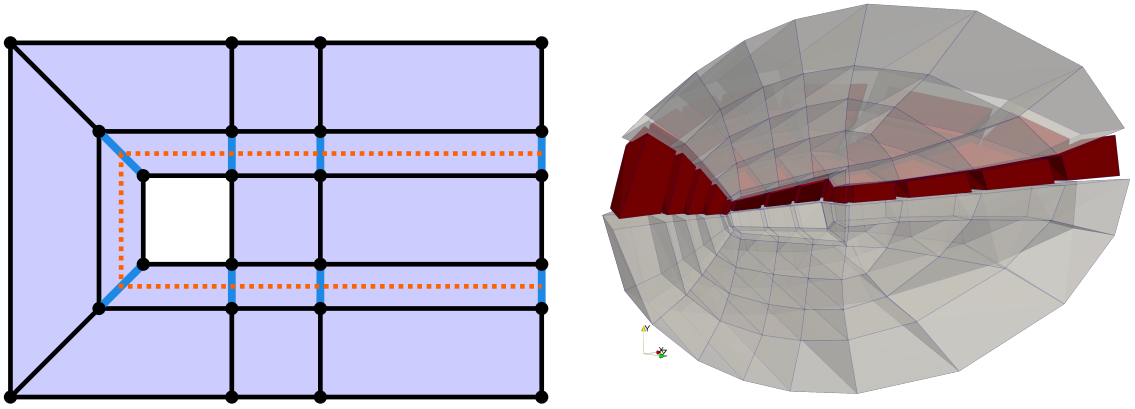
3.3.1 Interval Assignment

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 [BM10, GMF11, Mit21]. In particular, an incremental interval assignment using integer linear algebra is proposed by [Mit21] 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 $\mathcal{C}$ (see Def. 3.3.1) is defined as a set of opposite edges [SSS08] (see Fig. 3.19). We perform the computation of a blocking topological chord such as in Figure 3.19 in two steps: First we select a block edge like one of the blue ones (▬) on Figure 3.19.a; then, traversing adjacent blocks through opposite block edges, all the blue block edges (▬) of the orange chord (▬) are obtained. The process is the same no matter the dimension n. Figure 3.19.b represents the set of red blocks (▬) adjacent to the block edges of the same topological chord. For visualization purposes, the blocking is represented here using a clip view. In theory, a block in dimension n is included at most in n different topological chords. As a conformal mesh is expected, all the block edges $\{e_i\}_{i=1..n_c}$ of the same chord $\mathcal{C}$ need to have the exact same discretization. Otherwise, the blocking discretization is not valid.

Definition 3.3.1 (**Topological Chord**)**.** A **topological chord** is a set of adjacent opposite edges [SSS08] (see Fig. 3.19).

Then, starting from a block structure composed of N_e block edges, the problem can be reduced to n_e integer variables, where n_e is the number of topological chords in the block structure. For instance, in the case of Figure 3.19.a, we count eight different topological chords, hence eight integer unknowns.

This number of unknowns decreases again due to our application case, where the thin boundary layer along the vehicle wall is handled specifically. Let us consider Figure 3.20,



(a) Topological chord $\mathcal{C}$ ( ) composed of the blue block edges ( ) on 2D block structure

(b) Topological sheet ( )

Figure 3.19: Examples of 2D and 3D topological chords.

where the yellow block edges ( ) are on the vehicle wall, and the purple ones ( ) are in the boundary layer. The discretization of the yellow block edges ( ) is controlled by an input parameter s_w that fixes a target length of each final mesh edge on the vehicle wall and propagates along the corresponding chords. The discretization of the purple block edges ( ) helps to capture the boundary layer flow. This discretization is again an input parameter $s_w^{\perp}$, which strongly depends on the simulation. Again, some unknowns are so removed.

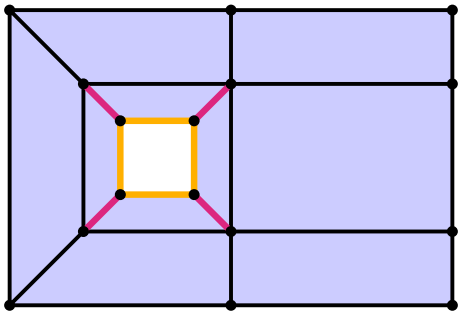


Figure 3.20: The block edges hard constrained in our algorithm are set in yellow ( ) and purple ( ).

It is essential to notice that if two block edges of a chord $\mathcal{C}_i$ have different hard constraints, the problem can not be solved, and the mesh is not generated. In our 2D case, it should not happen. The simple structure of our problem (full conform block structure) and the small number of hard constraints allow us to avoid building over-constrained systems in practice. However, for the 3D case, we choose the maximum number of final mesh cells over the conflicted edges as the final discretization for the whole chord. This may result in an over-discretized mesh in some parts of the domain.

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

$$F(t) = \sum_{e_i \in \mathcal{C}} \omega_i (t - T_i)^2, \tag{3.16}$$

where ω_i is the weight of the block edge e_i and T_i is the ideal discretization of the block edge e_i. To compute T_i, a target parameter s_G corresponds to the ideal default size of the target

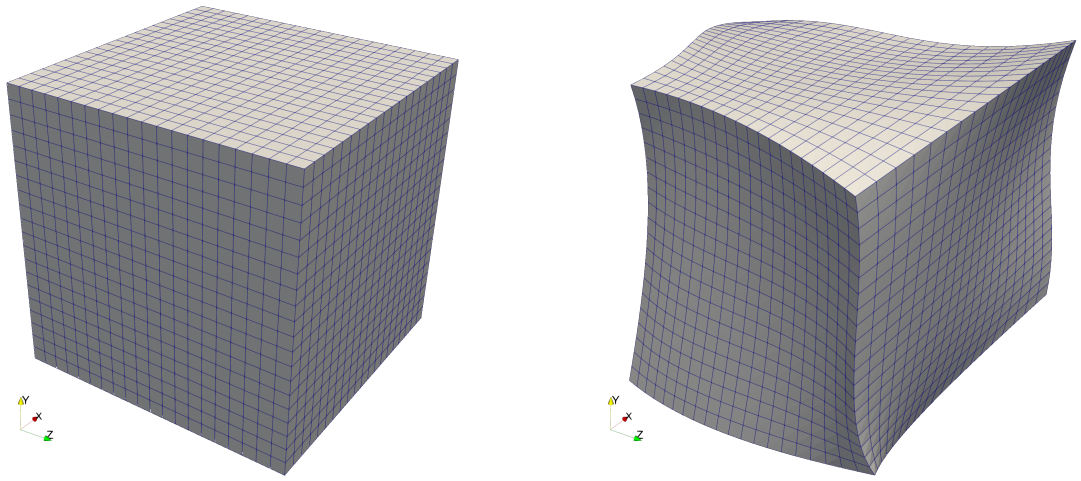
final mesh edges. We use the length of the Bézier block curved edge e_i to compute T_i. To estimate a Bézier edge length, we discretize the curve using a large sample of points and sum the discrete segment lengths. F is a second-degree polynomial in t made of positive terms that reaches a minimum when $\frac{F(t)}{\partial t} = 0$. We have

$$\frac{F}{\partial t}(t) = 2 \sum_{e_i \in \mathcal{C}} \omega_i (t - T_i). \tag{3.17}$$

So, the minimum is reached for

$$t_0 = \frac{\sum_{e_i \in \mathcal{C}} \omega_i T_i}{\sum_{e_i \in \mathcal{C}} \omega_i}. \tag{3.18}$$

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

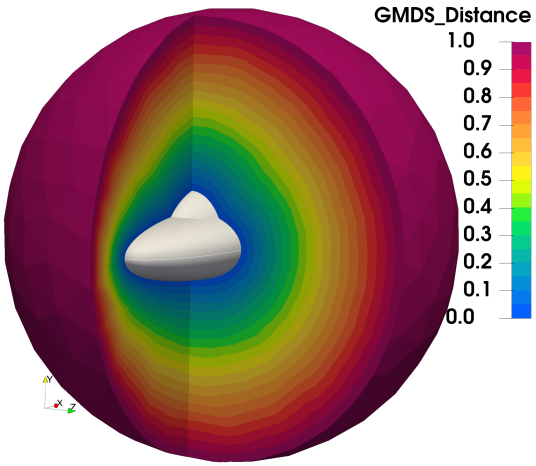


(a) Block discretization in parametric space (b) Block discretization in physical space

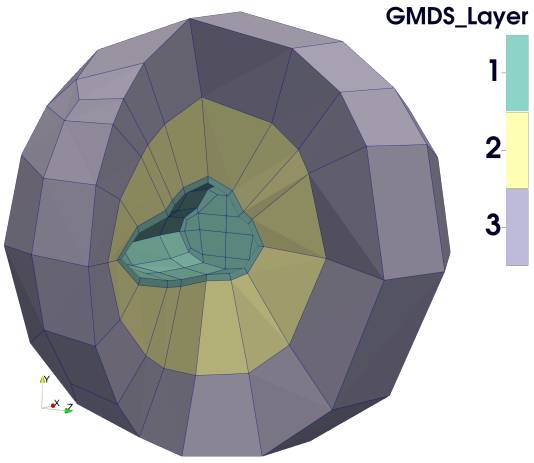
Figure 3.21: 3D Bézier block discretization in the parametric space (a), according to the number of edges in each direction given by the interval assignment algorithm. The final block discretization in physical space (b) is obtained using the parametrization of the Bézier block.

Once the discretization is set on all the block edges, we can generate the mesh in each block. To do so, given a block $\mathbf{B}$, we discretize the parametric space $[0, 1]^n$ uniformly (see the 3D example in Fig. 3.21.a), according to the previously computed number of mesh cells in each direction (using the interval assignment algorithm). Then, we compute the final coordinates of each mesh node in the physical space using the parametrization of the block (see Fig. 3.21.b).

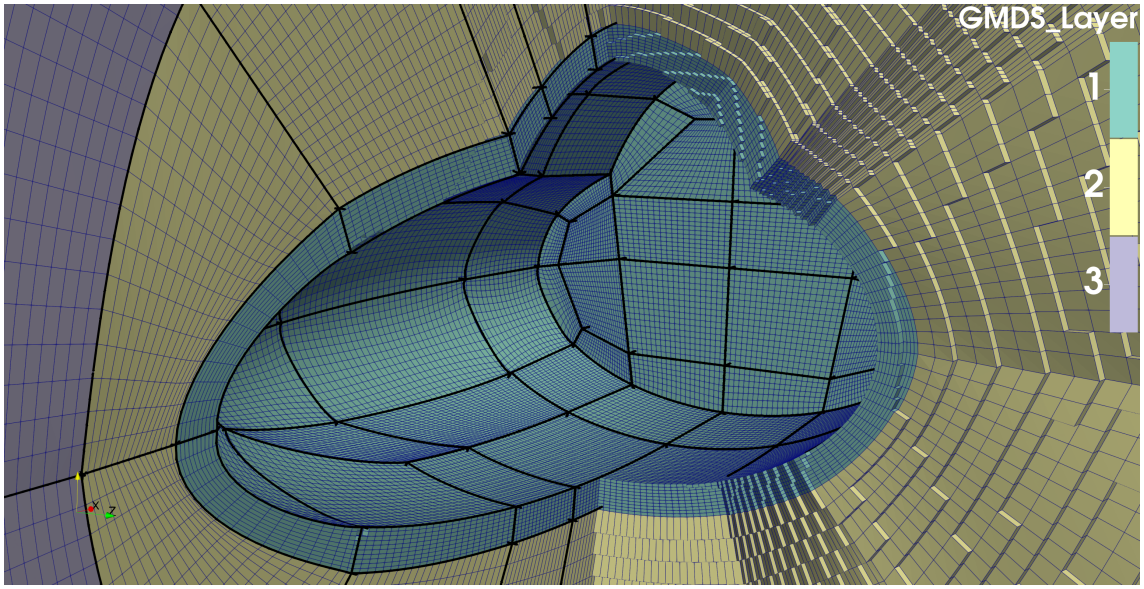
A practical example of mesh generation is presented in Figure 3.22 around a Double Ellipsoid geometry (see Figure 3.22.a). The linear block structure is generated on 3 different block layers using the method presented in Chapter 2. We use this linear block structure to obtain the curved block structure represented with black curved edges (▬) in Figure 3.22.c, around the final mesh generated. The block structure is made of 270 blocks, and the final mesh (see Fig. 3.22.c) is composed of around $1,500,000$ hexahedral cells. The computation of this mesh takes about 3 minutes with the implementation of our algorithm, starting from the different fields computation to the final mesh generation, including mesh quality metrics computation.



(a) Geometry ( ) and physical domain



(b) Linear blocks extruded over 3 layers



(c) Final mesh generated on curved blocks ( )

Figure 3.22: Mesh generation example around 3D Double Ellipsoid geometry (a). The linear blocking (b) is curved using Bézier elements of degree $p=3$ (c).

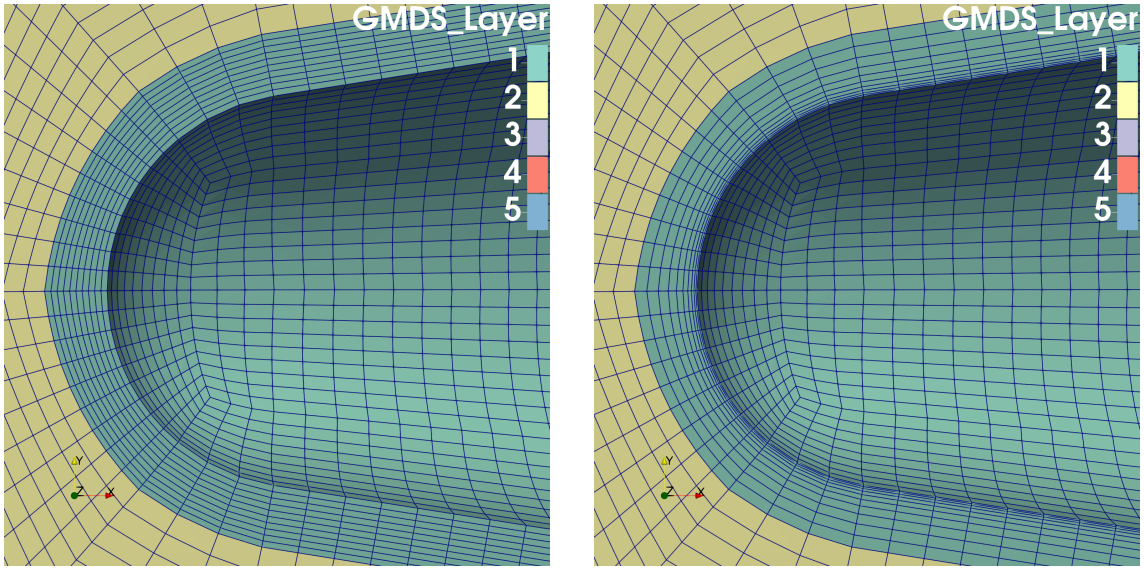
Refinement law for first block layer final mesh

In the wall-normal direction, a refinement law is performed on any opposite block edges set containing at least one block edge orthogonal to the vehicle wall (i.e., a block edges with one block corner on front $\mathcal{F}_0$, with the other block corner on front $\mathcal{F}_1$). This implies the cells can be very anisotropic, which is not a problem since gradients are in the wall's 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

$$l_i = l_1 + f_n(l_{N+1} - l_1), \tag{3.19}$$

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 $s_w^{\perp}$, the number of nodes $n_w^{\perp}$. This refinement law is particularly adapted to the boundary layer where the size of the first cell can be tiny. It avoids generating too large cell sizes far from the boundary layer.



(a) Without refinement (b) β law refinement

Figure 3.23: Example of the same mesh without refinement (a) or refined (b) in the boundary layer blocks generated around RAM-C II vehicle.

Figure 3.23 illustrates an example of the refinement law applied on the first layer blocks. Here, the mesh generated without refinement (see Fig. 3.23.a), or with refinement (see Fig. 3.23.b) are topologically equivalent (i.e., they share the same number of elements, and same connectivities). However, in case of Figure 3.23.b, we impose the first orthogonal mesh edge size and apply the β refinement law presented before.

2D

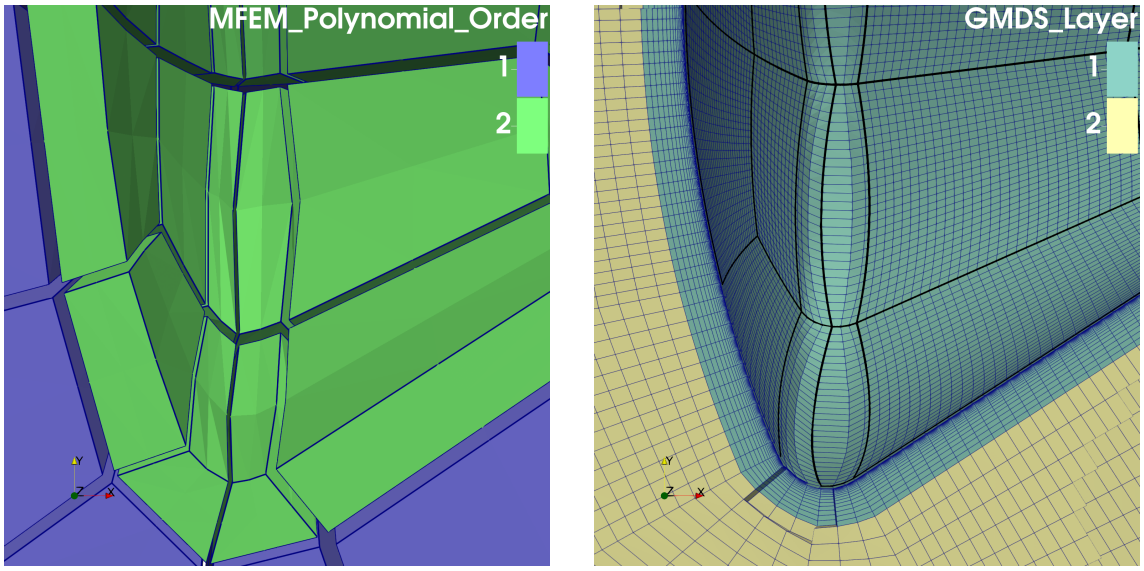
A mesh smoothing may be perform on blocks of the first block layer connected to the vehicle surface by an edge lying on the vehicle surface. This smoothing (see Appx. G) aims to orthogonilize cells to the wall. This smoothing is performed before the refinement, but the result of the smoothing algorithm, and refinement step is illustrated on a 2D mesh in Appendix G.

3.4 Perspectives

In this chapter, we presented how we curve a linear block structure using two different methods; the first one is in MFEM framework through *rp*-adaptivity, while the second approach relies on Bézier block representation and local and geometric operations. In the last section, we explained how we generate a mesh on a curved Bézier block structure. This work offers many perspectives. Here, we decide to focus on some of them and to provide details about potential future works.

Surface Block Edge Curving Using Geodesics

Through the Bézier block surfaces curving presented in Section 3.2, and the Bézier block edges curving analysis of Appendix F, we can conclude that the approach chosen here to compute the positions of control points is not sufficient to approximate the vehicle surface, and we could improve it by changing our approach. One lead we could explore is to use the notion of geodesic [BSK21, GR24] in order to build the curved block edges.



(a) MFEM high-order blocks (b) GMDS Bézier blocks and resulting mesh

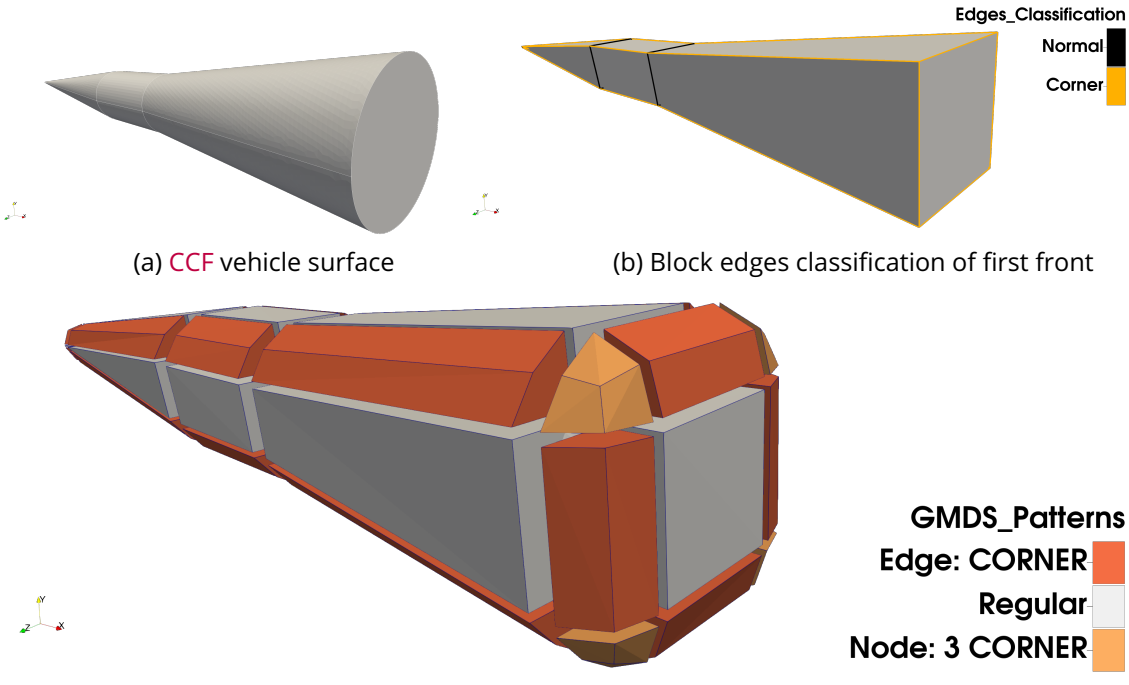
Figure 3.24: Comparison of high-order blocks around Apollo vehicle (see Appx. B).

Through our approach using Bézier block elements, and the approach through *rp*-adaptivity, we arrived at the conclusion that the criterion over minimizing the approximation error is not sufficient. Figure 3.24 represents the same block structure topology generated around

Apollo vehicle (see Appx. B) with *rp*-adaptivity (see Fig. 3.24.a), and using Bézier blocks (see Fig. 3.24.b). The two pictures represent the same clip view of the blocks among the plan $(O,\overrightarrow{X},\overrightarrow{Y})$. In Figure 3.24.a, only the high-order blocks are represented using a shrinked view[3] with their corresponding polynomial orders, while in Figure 3.24.b the Bézier block edges are plotted in black (▬), and the final mesh in blue (▬). In both cases, to curve the block structure, we focus on the geometrical error between the surface vehicle and the block surfaces. However, if we pay attention in both cases to the vertical block edges in the center, we may observe that even if the curved edges are on the vehicle surface, their directions are not as expected. To correct this effect, we expect to use the notion of geodesic over the vehicle surface, by also imposing some privileged directions linked to the surface curvature to lead the edges constructions.

This raises two other questions. The first one is, for now, with Bézier blocks, we use uniform degrees in all directions and all over the Bézier block structure. We could discuss about the potential choice of having non-uniform degree per direction first, and then, by block, in a similar way to what they do in MFEM framework. In addition, to curve our Bézier elements, we consider the control points as set uniformly in the parametrical space of the element. We could imagine a more strategic way to set the control points in the parametrical space in order to reduce the approximation error between a geometric curve and a Bézier curve for instance.

Curved Advancing Front Algorithm



(a) CCF vehicle surface

(b) Block edges classification of first front

(c) Linear first block layer

Figure 3.25: Block edges classification of first front on Sphere-Cone-Cylinder-Flare vehicle (see Appx. B).

[3]Here, the block edges of Fig. 3.24.a looks linear due to artefact visualization. The green blocks (▬) are of polynomial order $p=2$, so they are represented piecewise linearly using $p+1$ points per direction.

In this chapter, we presented how we manage to curve a linear block structure to approximate a vehicle surface. We expect to go further by building directly a curved block structure with the advancing front method presented in Chapter 2, starting from the curved first front. The main idea would consist in performing the block edges classification on a curved front.

If we consider the practical example given in Figure 3.25, the vehicle surface (see Fig. 3.25.a) is discretized with the input front of Figure 3.25.b. When we performed the block edges classification on this linear front, some block edges are classified as corner ( ) according to the angle between the two adjacent block $n-1$-cells while considering the vehicle surface of Figure 3.25.a, there is no such feature on the geometry. This classification leads to unwanted inserted blocks (see Fig. 3.25.c). If for now we propose a solution to unable the use of patterns in certain parts of the domain, classified a previously curved front could make it automatic in those situations.

Starting from a curved front, we could generate directly a curved block structure, layer by layer, using the same algorithm as in Chapter 2. As express in the advancing-front algorithm presented, when we compute the positions of the next front block corners starting from a front, to compute a layer, the advected block corner trajectory computed with the Runge-Kutta method does not follow a straight line. We could use this curved trajectory to generate the curved Bézier edges between two fronts. Then, when building the layer, we could imagine aligning the curved Bézier n-1-cell of the front to the iso-value of the distance field corresponding to the front.

Edge Size Transition

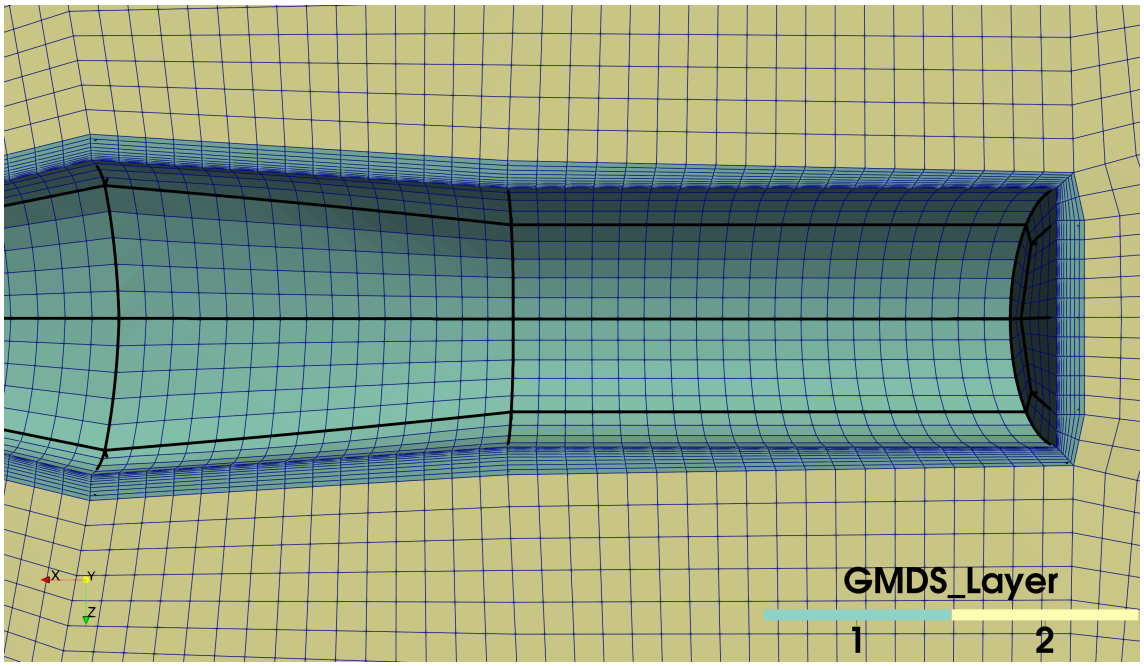


Figure 3.26: Brutal edge size transition between two different layers around HIFiRE-5 vehicle (see Appx. B).

If we manage through the β-law refinement to have a controlled transition of cell edge size in the first block layer ( ) (see Fig. 3.26), corresponding to the boundary block layer,

this size transition is not controlled everywhere. In the given example of Figure 3.26, we see a mesh generated around HIFiRE-5 vehicle, with the curved Bézier block edges of degree $p=2$ plotted in thick black (▬). The picture represents a clip view of the mesh along the plan $(O,\overrightarrow{X},\overrightarrow{Z})$ and focus on the two first block layers. As we see here, there is a brutal edge size transition between the first green block layer (▬), and the second one in yellow (▬). It would be advantageous to propagate the mesh size over the block layers, starting from the second one until the last one, by getting the size of the edge mesh cells of the precedent layer and applying a similar refinement law in the block layer, similar to what is perform for the first block layer.

High-Order Block Smoothing

The aim is to smooth the control points of high-order blocks to:

- Improve the mesh cells continuity between adjacent blocks,
- Improve the blocks' quality.

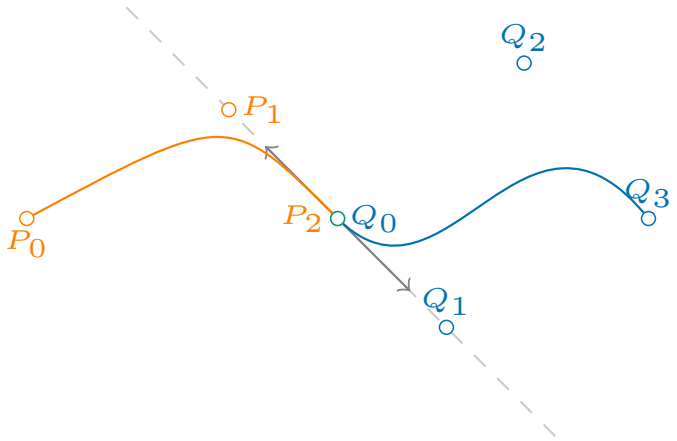
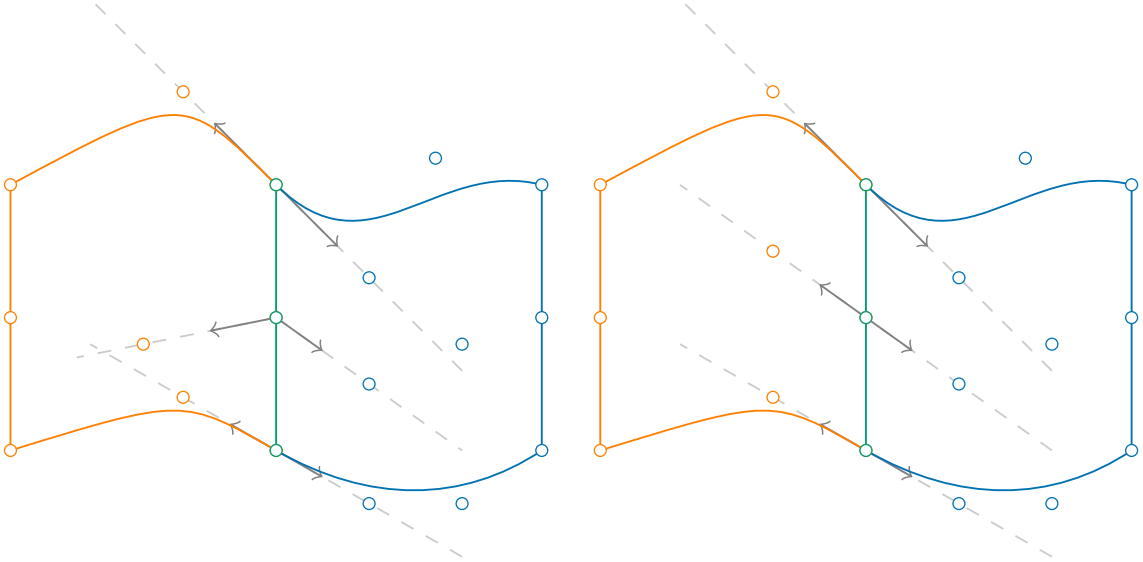


Figure 3.27: Continuity between two adjacent Bézier edges.

Considering the two adjacent Bézier edges of Figure 3.27. The orange one (▬) of degree $p=2$ is represented through the orange control points $\{P_i\}_{i\in[\![0,2]\!]}$ (○), and the blue one (▬) of degree $q=3$ is represented with the blue control points $\{Q_j\}_{j\in[\![0,3]\!]}$ (○). They share one control point (i.e., $P_2=Q_0$) represented in green (○). Using the Bézier curve properties, we know we conserve the continuity between the two curved edges at point $P_2=Q_0$ if the control points P_1, $P_2=Q_0$, and Q_1 are aligned.

By imposing the same principle on control points over Bézier quadrangular blocks (see Fig. 3.28) or hexahedral blocks, we may improve the mesh cell continuity when generating the final mesh using the parametrical space of the Bézier elements. Here, the green Bézier edge (▬) shared by both Bézier quadrangles is represented as straight, but in practice, it could be curved as the others. The orange block (▬) is a Bézier quadrangle of degree 2×2. The blue one (▬) is of degree 2×3.

The example of Figure 3.28 only illustrates two adjacent blocks in 2D, but we may imagine other adjacent blocks on each Bézier edge. This results in cross-constraints around some control

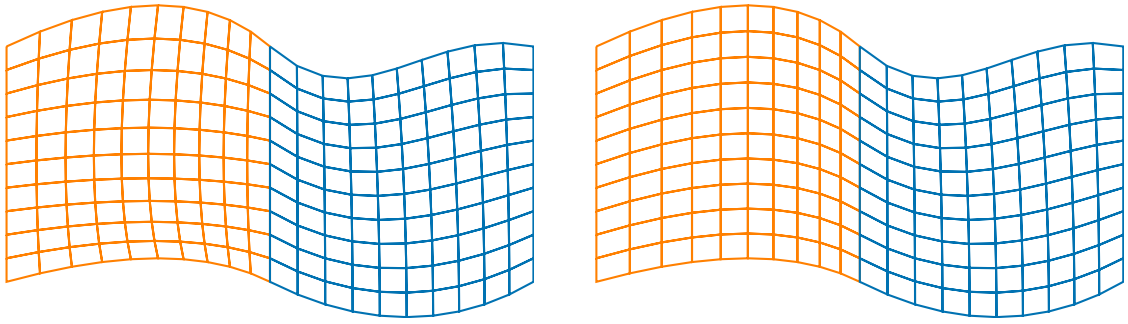


(a) Non-aligned inner control points (b) Aligned inner control points

Figure 3.28: Continuity between two adjacent Bézier quadrangles.

points. We think that we probably need to have Bézier blocks of at least degree $p > 3$ to relax the alignment constraints around the control points corresponding to block corners.

Figure 3.29 illustrates an example of a mesh generated on the two adjacent Bézier blocks of Figure 3.28. Each block is discretized by a 10×10 grid, with points set uniformly in the parametric space. The physical positions illustrated in Figure 3.29 were obtained using the mapping functions of each Bézier block. The mesh of Figure 3.29.a is generated on the non-aligned control points of Figure 3.28.a. The mesh of Figure 3.29.b is generated using the aligned control points given in Figure 3.28.b. As a result, the boundary mesh nodes of the two Bézier quadrangles are located at the same positions, but the mesh cells' continuity at the interface between the two blocks is better in the second case.



(a) With non-aligned control points (b) With aligned control points

Figure 3.29: Meshes (a) generated on non-aligned control points (see Fig. 3.28.a), and (b) on aligned control points (see Fig. 3.28.b).

Summary

In this chapter, we presented how, starting from a linear block structure around a vehicle, we curve the block structure before generating the final mesh. In order to curve the block structure, we explored two possibilities. The first one is through *rp*-adaptivity by solving a global problem on the block structure. It aims to improve the block geometric quality, and to align a set of $n-1$-cells with an interface (i.e., the vehicle surface in our case) while carefully controlling the polynomial degree of elements around the interface. As this method remains time consumming in 3D, we proposed a second approach based on a Bézier representation of blocks. We curve the structure using local and geometrical operations to align to the vehicle surface, and then propagate the modifications in the volume to avoid tangled blocks. Finally, we generate the mesh by solving an optimization problem to set the discretization of each block edge, and then use the parametric space of our Bézier block to compute the positions of the mesh nodes in the physical space.

Chapter 4

BLOCK-STRUCTURED MESHES QUALITY ANALYSIS

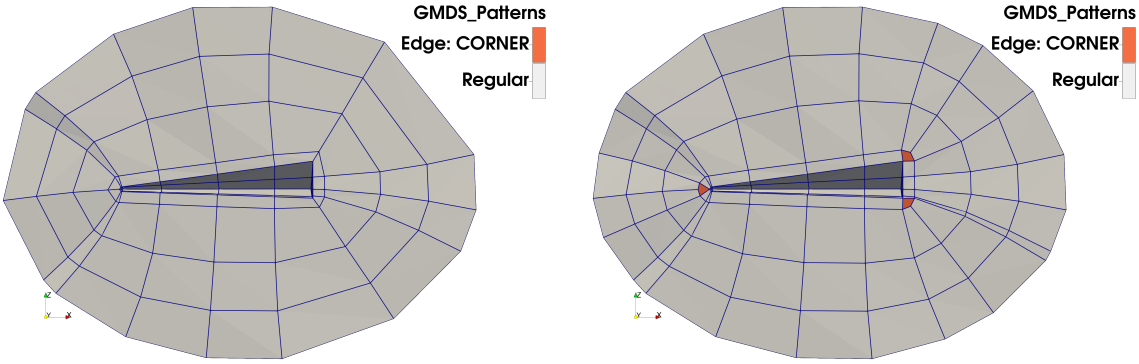
This last chapter discusses the block-structured final mesh quality generated by our method. We illustrate this quality through two main axes. The first one is a purely geometrical analysis of final hexahedral mesh cells. We compare different block structure topologies generated and the resulting final mesh quality. The second way to evaluate mesh quality presented here is through practical results of numerical simulations performed on meshes generated with our method. The first step was to perform 2D numerical simulations over well-known geometries using the open-source Computational Fluid Dynamics code SU2 [SU2, EPC+16]. The aim was to ensure that our meshes passed the validity check of the solver and then to show that we captured the right phenomena. To do so, we simulated a subsonic flow around the well-known NACA 0012 airfoil and a supersonic flow around a 2D diamond-shaped airfoil. In the first case, we compare our simulation results to an experimental data set given by NASA. In the second case, we compare our results to analytical solutions for shock position. The second step was to perform a 2D hypersonic flow simulation around a vehicle using the target CEA CFD code. A comparison is done using a mesh generated by hand as a reference. In the last section, we perform a 3D supersonic flow simulation around the RAM-C II vehicle using SU2.

Contents

4.1 Geometrical Mesh Quality

This section proposes an overview of different block structures and resulting mesh quality comparisons and analysis based on purely geometrical quality criteria. We only present mesh quality evaluations for 3D vehicles. We propose first to analyze the impact of block patterns on the resulting mesh quality, then the effect of smoothing the linear block structure (see Appx. E) prior to generating the curved block structure and the resulting mesh.

4.1.1 HyTRV



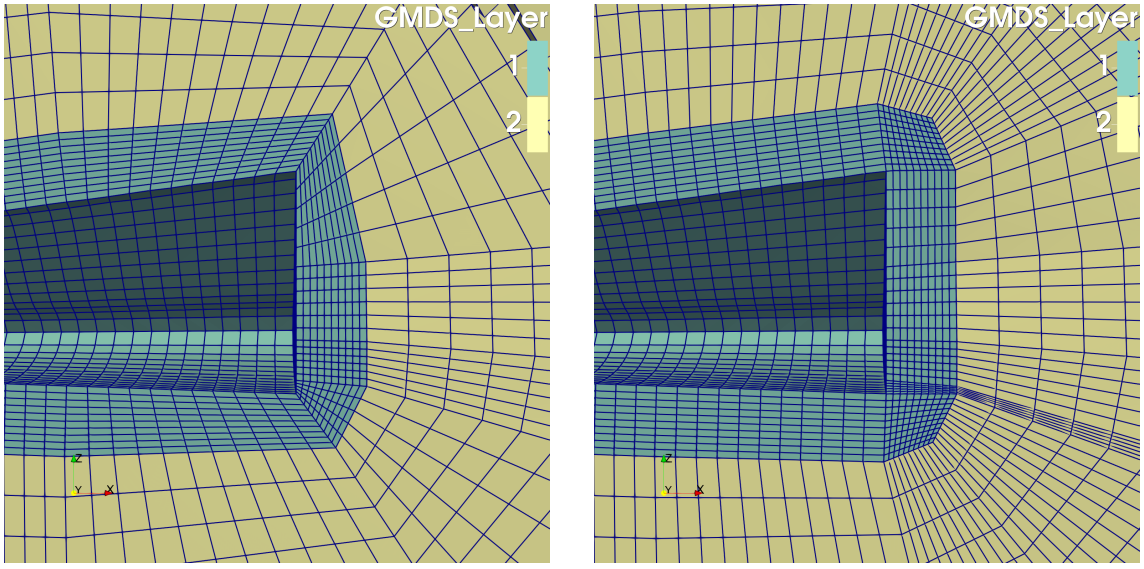
(a) Without patterns (b) With patterns

Figure 4.1: Two block structures generated with 4 block layers around HyTRV, without patterns (a), and with patterns on first block layer (b).

Here, we propose to study the geometrical quality of a mesh generated around HyTRV (see Appx. B) depending on the patterns used to create the block structure. Figure 4.1 represents a clip view along the plan $(O, \overrightarrow{X}, \overrightarrow{Z})$ of two different block structures generated using the same parameters, except for Figure 4.1.a, patterns are not allowed during the extrusion (i.e., every block layer is regular). In the case of Figure 4.1.b, the number of block layers is the same (and equal to 4), input background mesh and wall surface block discretization (first front) are the same, the distance and vector fields are computed the same way, etc. The only difference is we allow patterns on the first block layer. As a result, we see orange hexahedral blocks (▬) inserted by patterns on the first block layer. Two patterns inserted on block corners are also used in this example but are not visible in this image. In this example, we do not apply any smoothing procedure on the two linear block structures.

To generate the curved block structure and the resulting mesh, in both cases, the parameters used are the same (i.e., curved Bézier block degree, target size of final mesh edges on wall, target default size of final mesh edges in the domain, number of cell in boundary block layer, etc.). The Bézier blocks are of degree $p = 2$. However, the topological structure of the two blockings results in very different final meshes. In the case of Figure 4.1.a, the resulting mesh (see Fig. 4.2.a) has $113,920$ elements, and the minimal value of the scaled Jacobian is around -0.29. For this mesh, it takes around 35 seconds to execute the full algorithm, from the fields computation on the tetrahedral background mesh $\mathcal{M}_T$, to the hexahedral mesh generation, including the mesh quality evaluation.

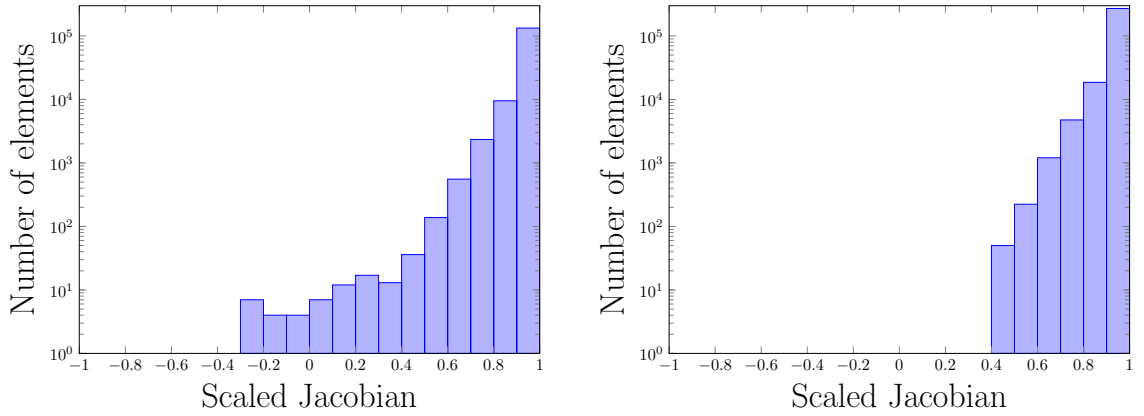
In the case of Figure 4.1.b, the final mesh (see Fig. 4.2.b) comprises $226,320$ mesh cells,



(a) Without patterns (b) With patterns

Figure 4.2: Two meshes (a), and (b) generated respectively on block structures of Figure 4.1.a, and Figure 4.1.b.

but the minimal scaled Jacobian equals 0.42. The final mesh cell difference number is due to the patterns (▬) used in the first block layer. As we want many cells in this block layer, the number of mesh cells for block edges of the first block layer (▬) in the wall-normal direction is imposed by a user parameter. In the case of Figure 4.1.b, this refinement propagates in other blocks in the domain due to the blocks inserted (▬) in this first block layer. This phenomenon is illustrated on the corresponding mesh of Figure 4.2.b. The mesh is presented using a clip view along the plan $(O, \vec{X}, \vec{Z})$ as for Figure 4.1.b. On the top and bottom right part of the domain, we can see the increase of mesh cells density due to the propagation of the discretization of the first block layer. For this second mesh, it takes around 1 minute to execute the full algorithm.

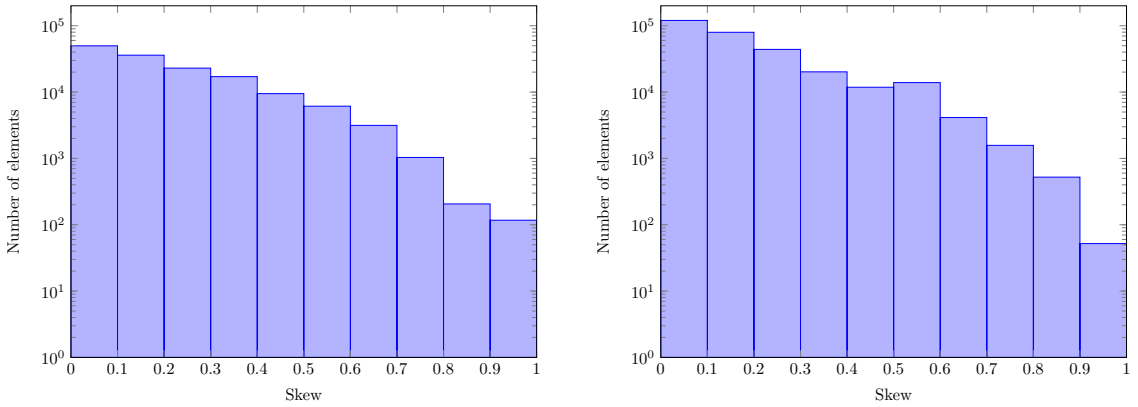


(a) Without patterns (b) With patterns

Figure 4.3: Scaled Jacobian of resulting meshes generated on block structures of Figure 4.1 around HyTRV.

Figure 4.3 shows the scaled Jacobian distribution on both meshes. In case patterns are not allowed (see Fig. 4.1.a), this results in inverted final mesh cells (see Fig. 4.3.a). While in case patterns are allowed (see Fig. 4.1.a), the resulting scaled Jacobian distribution over

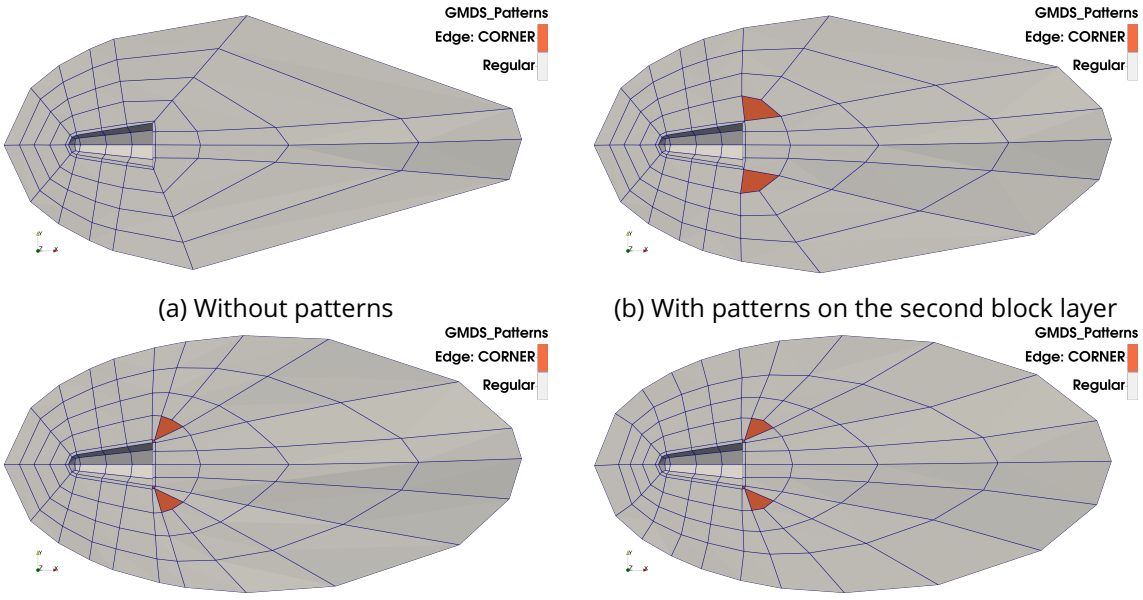
the mesh improves (see Fig. 4.3.b). In this case, using patterns on the first block layer also enhances the skewness of mesh elements (see Fig. 4.4). The execution time for the entire method, including the distance and vector fields computation on the background mesh, until the geometrical quality computation on the final mesh takes about 1 minute. If the user judges that the final mesh quality or block topology is not convenient, it is reasonable to change the input parameters and run the algorithm again.



(a) Without patterns (b) With patterns

Figure 4.4: Skewness of resulting meshes generated on block structures of Figure 4.1 around HyTRV.

4.1.2 RAM-C II

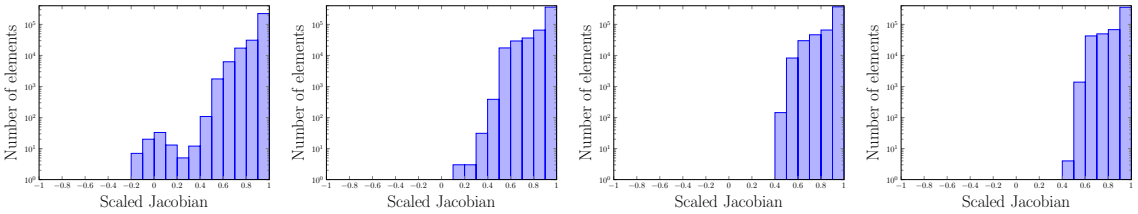


(a) Without patterns (b) With patterns on the second block layer

(c) With patterns (d) With patterns and smoothing

Figure 4.5: Four block structures generated with 5 block layers around RAM-C II, without patterns (a), with patterns allowed except on the first block layer (b), and with patterns allowed everywhere (c). The last block structure (d) is obtained with patterns allowed all over the domain and using 4 smoothing iterations on the linear block structure.

This subsection compares four different meshes obtained around RAM-C II (see Appx. B) vehicle. The four corresponding linear block structures are represented in Figure 4.5 using a clip view along the plan $(O,\overrightarrow{X},\overrightarrow{Y})$. To generate the four meshes, we use the same parameters (i.e., vector field computation, default mesh edge size, etc.), except we play here with patterns over the layers. Figure 4.5.a represents the most regular block structure without any patterns applied. In Figure 4.5.b, we allow patterns over the layers but not on the first block layer. Consequently, a closed path of corner block edges is computed, and orange (▬) is inserted on the second block layer. In Figure 4.5.c, we allow patterns on all layers. As a result, a closed path is detected on the first front, and corresponding orange hexahedral blocks (▬) are inserted. In this case, patterns are also inserted on the second block layer. The last linear block structure of Figure 4.5.d is obtained by applying 4 smoothing iterations (see Appx. E) to the linear block structure of Figure 4.5.c.

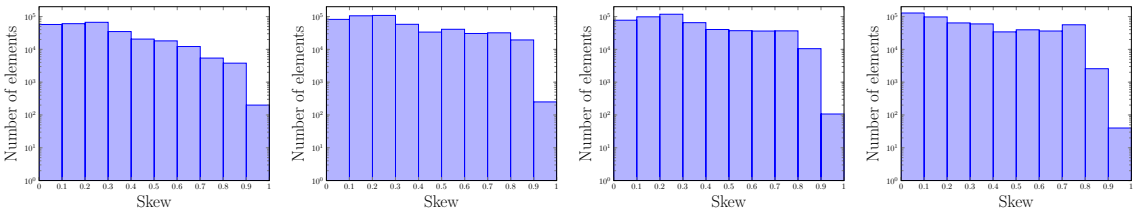


(a) Without patterns (b) With patterns on the second block layer (c) With patterns (d) With patterns and smoothing

Figure 4.6: Scaled Jacobian of resulting meshes generated on block structures of Figure 4.5 around RAM-C II.

The curved blocks of all four linear block structures are computed the same way, and every Bézier block is of degree $p = 2$, and the final meshes are generated using the same target sizes. All four meshes have an approximate number of $500,000$ cells. If we take a look at the scaled Jacobian distribution in Figure 4.6, using patterns in different layers improves this quality criterion and avoids inverted cells (see Fig. 4.6.a).

Adding patterns also improves the skewness distribution (see Fig. 4.7.a and b). But in case we allow insertions all over the domain (see Fig. 4.5.c), it increases the skewness of some elements (see Fig. 4.7.c). This is partly due to the flattened inserted blocks on the second block layer (see Fig. 4.5). By smoothing this linear block structure (see Fig. 4.5.d), it improves both the scaled Jacobian distribution (see Fig. 4.6.d) and the skewness distribution (see Fig. 4.7.d) of the mesh.



(a) Without patterns (b) With patterns on the second block layer (c) With patterns (c) With patterns and smoothing

Figure 4.7: Skewness of resulting meshes generated on block structures of Figure 4.5 around RAM-C II.

Figure 4.8 represents the mesh generated on the block structure of Figure 4.5.c and corresponds to the scaled Jacobian and skewness distributions of Figure 4.6.c and Figure 4.7. The

RAM-C II vehicle surface is represented in gray ( ). The mesh is represented using two clip views along plans $(O,\overrightarrow{X},\overrightarrow{Y})$ and $(O,\overrightarrow{Y},\overrightarrow{Z})$ for visualization purpose.

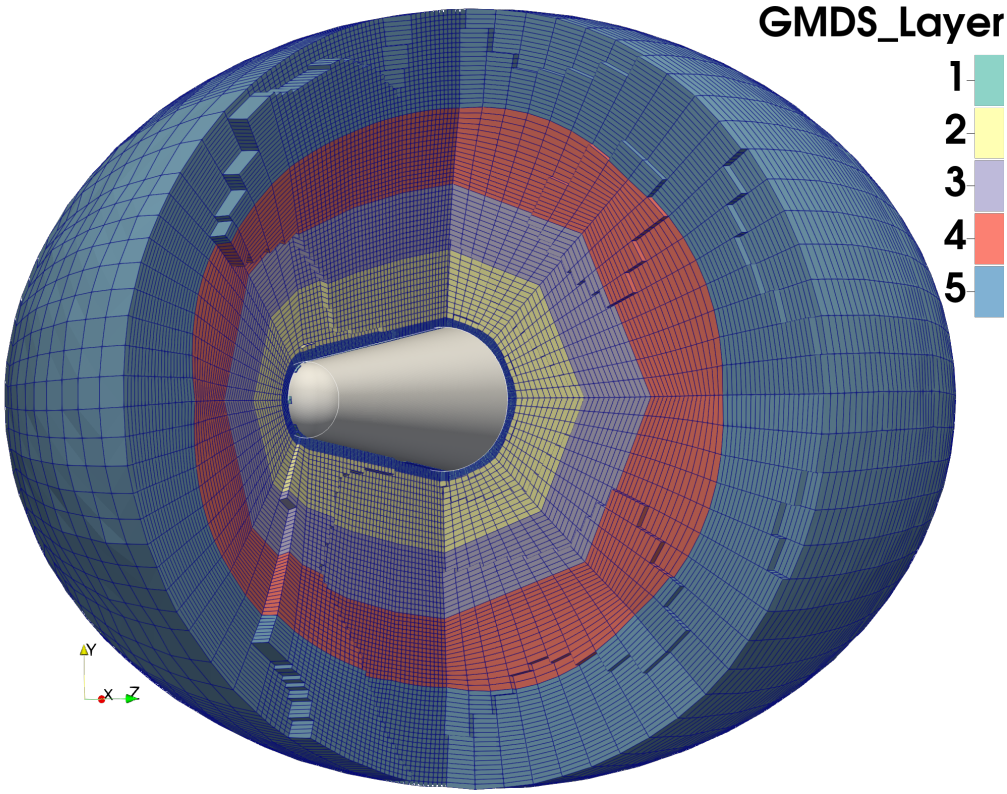
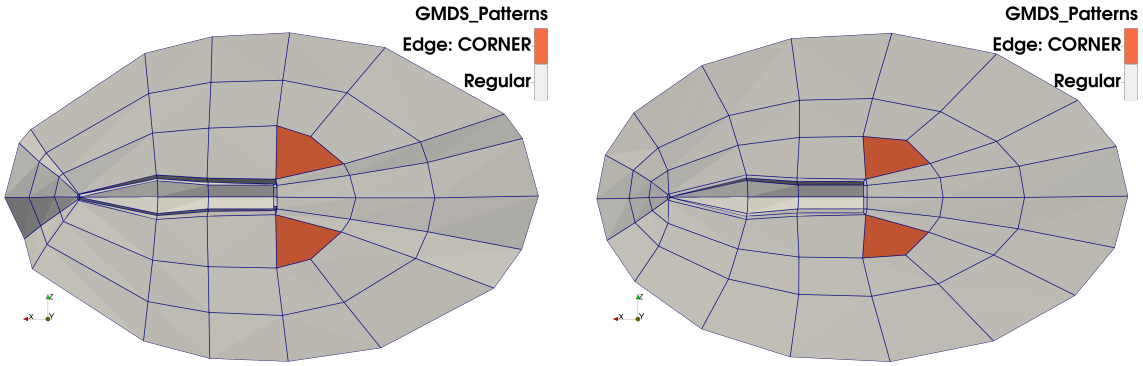


Figure 4.8: Mesh generated on block structure 4.5.c around RAM-C II.

4.1.3 HIFiRE-5

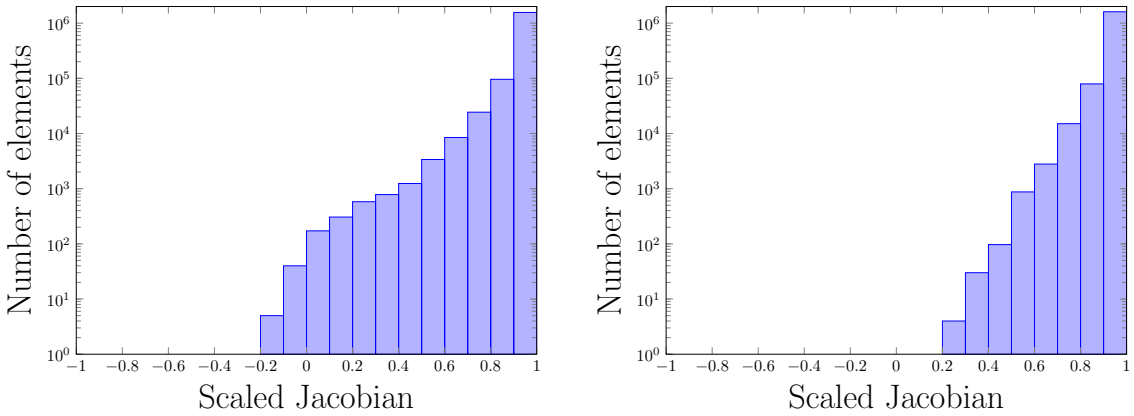


(a) Without smoothing (b) With smoothing

Figure 4.9: Two block structures generated with 4 block layers around HIFiRE-5, without smoothing (a), and with smoothing (b).

Here, we compare two different meshes generated around HIFiRE-5 vehicle (see Appx. B) using the same parameters and with the same block structure topology. But in one case, we perform 10 smoothing iterations (see Appx. E) on the linear block structure before curving the blocks and generating the final mesh. The non-smoothed block structure is represented

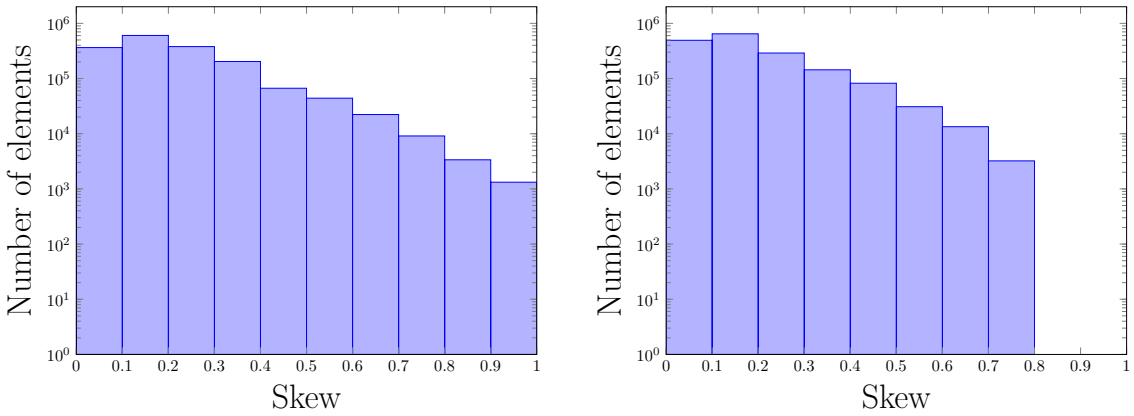
in Figure 4.9.a using a clip view along the plan $(O, \vec{X}, \vec{Z})$. In contrast, the smoothed one is represented in Figure 4.9.b. In both cases, the block structure topology is the same, with patterns inserted (■) on the second block layer. Due to how we compute the final discretization of the block structure, two block structures with the same topology may have a different final number of mesh cells, as is the case here.



(a) Without smoothing (b) With smoothing

Figure 4.10: Scaled Jacobian distrobution on final meshes generated on non-smoothed (a), and smoothed (b) block structure of Figure 4.9 around HIFiRE-5.

In Figure 4.10a, we plot the scaled Jacobian distribution over the final meshes generated on the non-smoothed block structure (see Fig. 4.9.a), and Figure 4.10.b corresponds to the scaled Jacobian distribution over the final mesh generated on the smoothed block structure (see Fig. 4.9.b). If we lose the block alignment to the vector field after smoothing, generating the final mesh on the smooth block structure allows us to avoid inverted mesh cells. In the first case, the mesh comprises approximately $1,700,000$ cells, and the worst cell has a scaled Jacobian equal to -0.13. In the case of the smoothed block structure, the final mesh is made of approximately the same number of cells, but the worst element has a scaled Jacobian of 0.28. Finally, if we consider the skewness distribution of mesh cells given in Figure 4.11, the smoothing also improves the quality of the generated mesh cells. For the two meshes generated, the full computation time is around 3 minutes and 30 seconds.



(a) Without smoothing (b) With smoothing

Figure 4.11: Skewness distribution on final meshes generated on non-smoothed (a), and smoothed (b) block structure of Figure 4.9.

4.1.4 Vehicle with wings

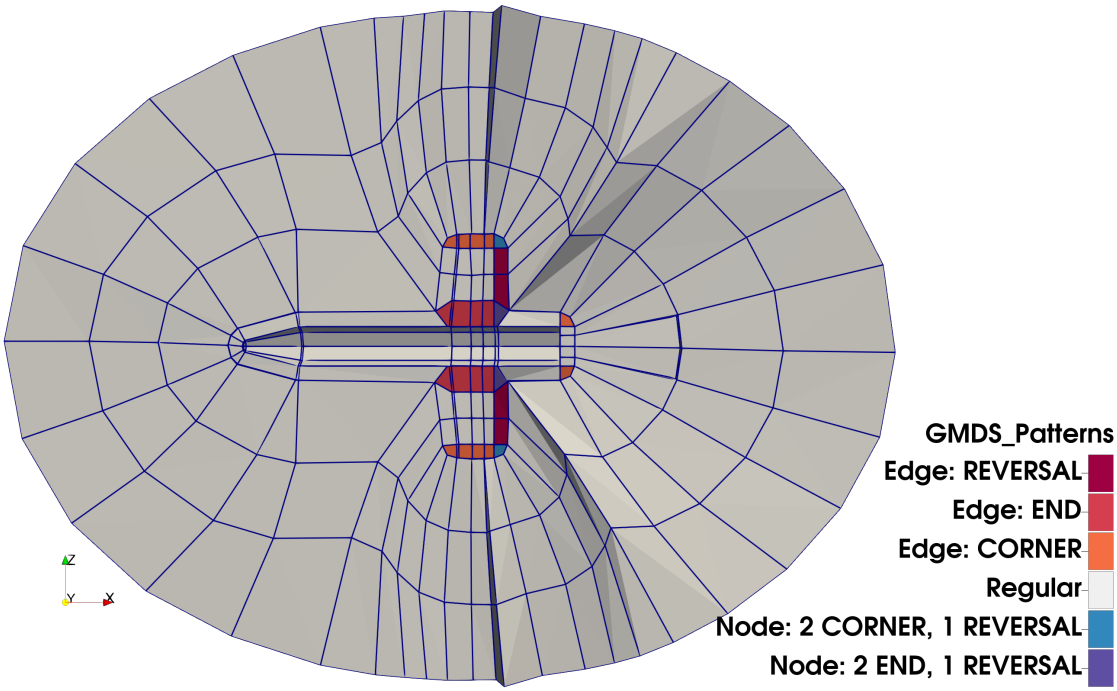
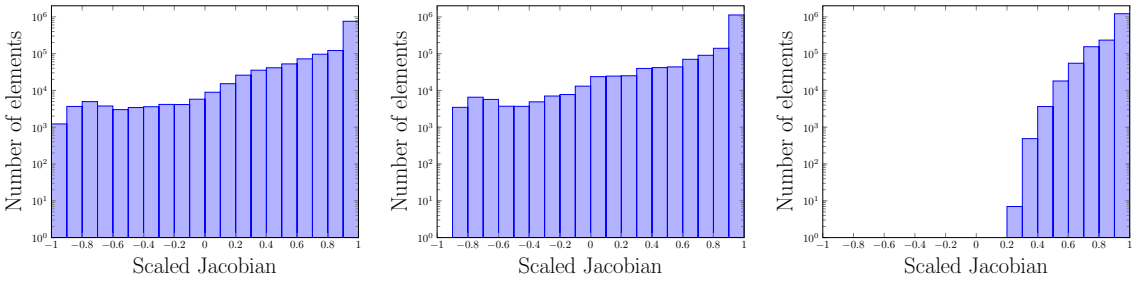


Figure 4.12: Block structure around the vehicle with wings.

This vehicle is challenging because the wings create a non-convex domain. As a consequence, we need to solve conflicts due to the contraction of the domain, and those operations are mandatory. Otherwise, we can not reach an acceptable final mesh quality even after smoothing. This section compares three meshes generated around a geometry with wings. Unlike before, we only present the block structure of one of them in Figure 4.12 for clarity.

Figure 4.12 represents a block structure generated around a vehicle with two wings. This block structure is generated along 4 layers, and patterns are used in the first block to solve conflicts due to sharp geometrical features on the vehicle surface. Here, the block structure is represented in a clip view along the plan $(O, \vec{X}, \vec{Z})$. The vehicle surface is symmetric along this same plan.



(a) Without patterns (b) With patterns (c) With patterns and smoothing

Figure 4.13: Scaled Jacobian distribution on final meshes generated on non-smoothed (a), and smoothed (b) block structure of Figure 4.12 around vehicle with wings.

In Figure 4.13, we present the scaled Jacobian distribution of the three different meshes. Once again, the three meshes are generated using the same global parameters. For the first

one of Figure 4.13.a, the block structure topology is generated regularly without any patterns over the layers. The second result in Figure 4.13.b refers to the mesh generated with the same global parameters, but this time, patterns are used in the first block layer. Unlike in the previous example, adding patterns in a layer is insufficient to improve the resulting mesh's quality. But by applying 14 iterations of the smoothing algorithm (see Appx. E), we obtain the final result of Figure 4.13.c. This third mesh is generated on the block structure of Figure 4.12. We create a mesh with no inverted cells by adding patterns and smoothing the block structure (see Fig. 4.13.c).

4.2 Application to Computational Fluid Dynamics Simulations

In this section, we evaluate the mesh quality generated with our algorithm on numerical simulation practical examples. To do so, we detail a sample of simulations run on 2D and 3D meshes for subsonic, supersonic, and hypersonic vehicles, with two different Computational Fluid Dynamics (CFD) code: SU2 [SU2, EPC+16], and the legacy Navier-Stokes steady/unsteady solver of the French Alternative Energies and Atomic Energy Commission (CEA). Our first quality criterion is that two different solvers accept our meshes as input and can compute a solution on them. So, our meshes pass the solver validity check. Then, we compare results obtained on our meshes to experimental data sets, analytical solutions, and reference solutions obtained on a different mesh generated by hand.

SU2 is an open multiphysics simulation software developed by Stanford University that provides computational analysis tools for numerical simulation. SU2 is an unstructured solver, and in this work, we adapted our output meshes to express them as unstructured hexahedral meshes instead of block-structured ones. Remember that a subsonic vehicle refers to a Mach number $M_\infty < 1$, a hypersonic vehicle refers to $M_\infty > 5$, and a vehicle is considered supersonic in between.

4.2.1 Subsonic 2D NACA 0012 Airfoil

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

Table 4.1: Simulation parameters for the subsonic NACA 0012 airfoil.

A simulation of a flow around the NACA 0012 airfoil (see Appx. B) is performed with the data set provided in Table 4.1 to validate the accuracy of the results on our generated mesh. This simulation uses the Reynolds Averaged Navier-Stokes (RANS) solver of SU2 [SU2, EPC+16] and the k-ω SST turbulence model [Men92, Men94]. The angle of attack is $\alpha = 15°$, the Mach number is $M_\infty = 0.3$, the Reynolds number is set to $Re = 3 \times 10^6$, and the temperature $T_\infty = 293$K.

In Figure 4.14, pressure coefficients

$$C_P = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty u_\infty^2} \tag{4.1}$$

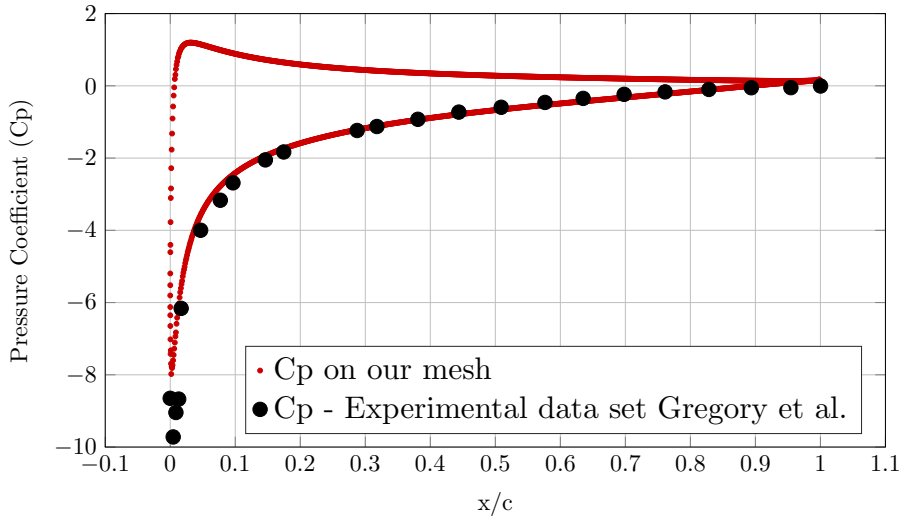


Figure 4.14: Pressure coefficient on the NACA 0012 airfoil for the simulation parameters of Table 4.1.

where p is the pressure, p_∞ is the upstream pressure, ρ is the density, and $\overrightarrow{u_\infty}$ is the upstream flow velocity, are compared. For a Reynolds number of $Re = 3 \times 10^6$, the experimental data set of Gregory and O'Reilly [GO70] seems to be the most appropriate for CFD validation [Rum21]. These experimental data used as reference points for the surface pressure coefficients are plotted with black dots (●). The pressure coefficient plotted with red dots (●) results from the simulation of this configuration performed with SU2 on the mesh generated by our algorithm. The two red curves result from the simulation plotted on both sides of the airfoil. However, the experimental data set provides information only on one side of the airfoil. The results obtained with this configuration on our mesh are in good agreement with the experimental results.

We also give interest here in the y^+ value, which is defined in the reference book [Cou89] as

$$y^+ = \frac{yU_\tau}{\mu}, \tag{4.2}$$

with U_τ the friction velocity. The y^+ value represents the undimensionalized mesh size. In this case, the $y_1^+ < 1$, which is the undimensionalized wall orthogonal first cell size. This means the mesh resolution in the boundary layer is sufficient to capture the expected phenomena[1].

4.2.2 Supersonic 2D Diamond Shaped Airfoil

This part simulates a two-dimensional supersonic flow around a diamond-shaped airfoil inspired by [FCK16]. This simulation uses the Reynolds Averaged Navier-Stokes (RANS) solver of SU2 and the k-ω SST turbulence model [Men92, Men94]. Figure 4.15 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

[1] Let us note that in case this resolution is not sufficient, we can easily re-generate the mesh by only changing the input parameter corresponding to the wall orthogonal first mesh cell size, or the number of mesh cells in the boundary layer blocks.

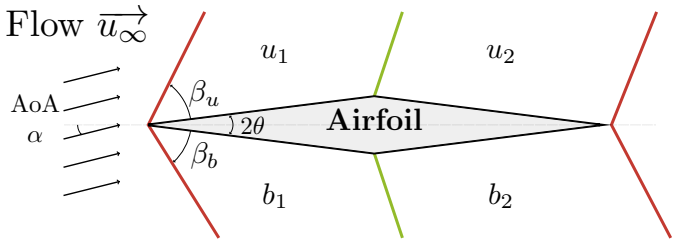


Figure 4.15: Scheme of the various zones and angles around the diamond airfoil [FCK16].

M_∞	AoA α	R_e	T_∞
1.5	3°	3×10^6	293K

Table 4.2: Simulation parameters for the supersonic diamond airfoil.

oblique shocks represented in red (▬) are given by Liepmann et al. [LR01]. The shock direction depends on the Mach M_∞, the angle θ of the geometry, and the angle of attack α. In this study, $\theta = 5°$ and the airfoil chord is 1m. Table 4.2 are set for this simulation.

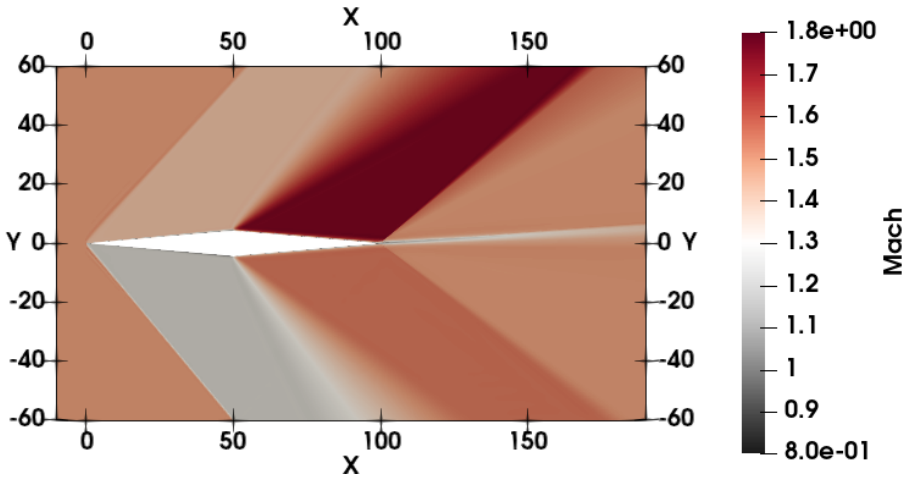


Figure 4.16: Mach-number distribution around a diamond-shaped airfoil immersed in a supersonic flow field at $M_\infty = 1.5$, $Re = 3 \times 10^6$ and $\alpha = 3°$.

In Figure 4.16, 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 $\beta_u = 43.1°$, 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 4.16, and $M_{b_1} = 1.19$ for the area b_1. These results are consistent with those given by the tables [rs53] (see Appx. H). Regarding these results, the mesh generated by our algorithm captures the expected physics.

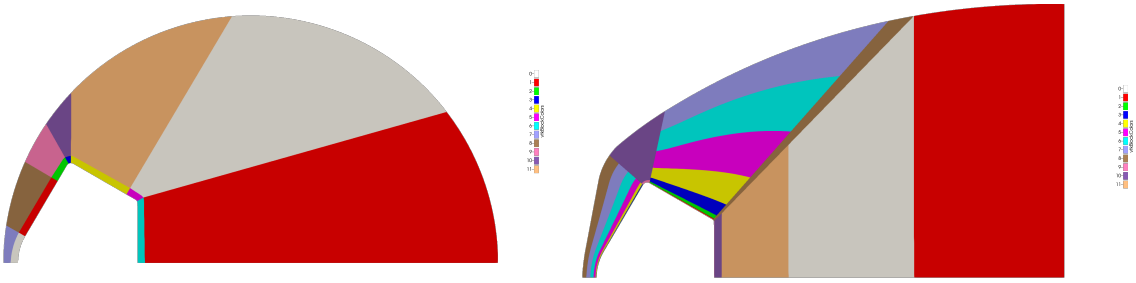
4.2.3 Hypersonic Stardust 2D

In this subsection, we present a numerical simulation result on a block-structured mesh generated with our 2D algorithm (see Fig. 4.17.a) and compare it to a simulation run on a block-structured mesh generated by hand using the dedicated software ICEM-CFD [ICE] (see Fig. 4.17.b). For both meshes, the first wall-normal cell size is set to 1.10^{-6}m. We use here

M_∞	AoA α	R_e	T_∞
10.0	0°	10^6	270K

Table 4.3: Simulation parameters for the hypersonic Stardust.

the CEA legacy code to simulate a hypersonic flow around Stardust (see Appx. B). We solve the Navier-Stokes equations using the physical parameters of Table 4.3 and expect to capture the shock position.



(a) GMDS block structure (automatic) (b) ICEM-CFD block structure (hand-made)

Figure 4.17: Two different block structures generated around the axisymmetric Stardust geometry (see Appx. B). Each color stands for a different block.

Figure 4.18 illustrates the simulation result on both meshes. In Figure 4.18.a and b, the pressure fields are plotted. The shock position computed on the two meshes is at the same position. In Figure 4.18.c and d, the velocity magnitude field is presented with the streamlines (▬). The phenomena captured on our mesh are the same as the ones on the mesh generated by hand using ICEM-CFD [ICE].

4.2.4 Supersonic RAM-C II 3D

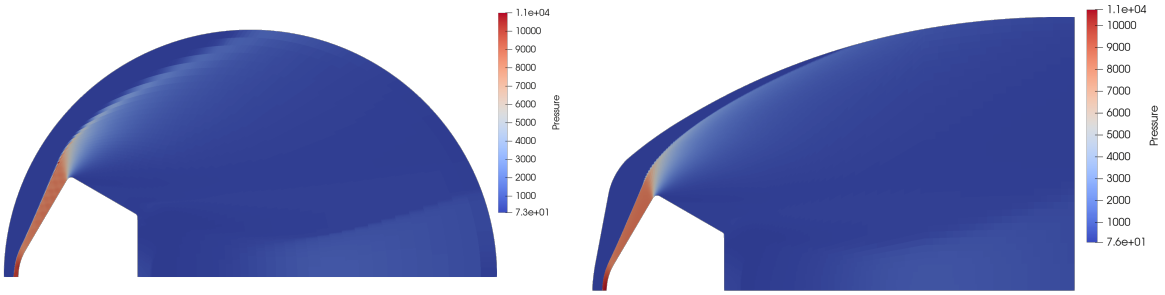
M_∞	AoA α	p_∞	T_∞
1.5	0°	$101,325$Pa	288.15K

Table 4.4: Simulation parameters for the supersonic RAM-C II 3D.

This last simulation is performed using SU2 around the 3D RAM-C II geometry (see Appx. B). The simulation parameters are presented in Table 4.4. The Euler equations are solved (i.e., the viscosity effects are not considered).

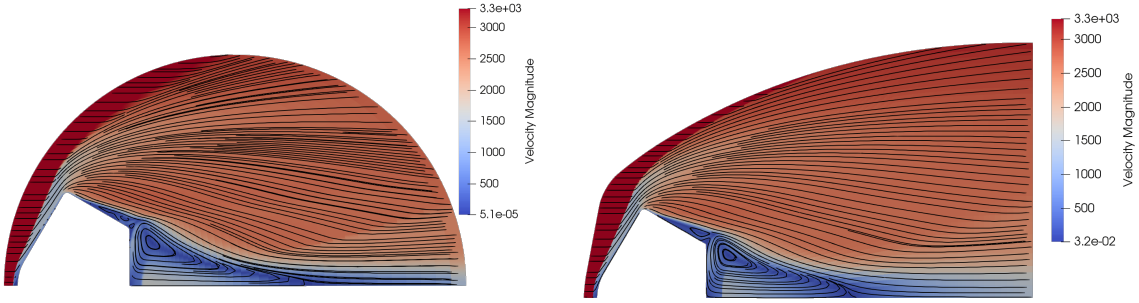
Figure 4.19.b presents the Mach field computed around RAM-C II nose. On the left side of the geometry, around the vehicle nose, we see a curve detached shock wave. This shock is normal on the vehicle symmetry axes at the stagnation point (vehicle surface point where the velocity is null), for this reason, the flow is subsonic in the area between the shock wave and the vehicle nose. Behind the more oblique portions of the shock wave, the flow is supersonic. These observations are coherent with the explanations given in [And89] (see Fig. 4.19.a).

Figure 4.20 presents the undimensionalized pressure field around RAM-C II. Regarding the wake (i.e., the flow behind the vehicle), the phenomena observed are similar to [MJK64]. In this area, we can see the field recombination.



(a) Pressure field on GMDS mesh

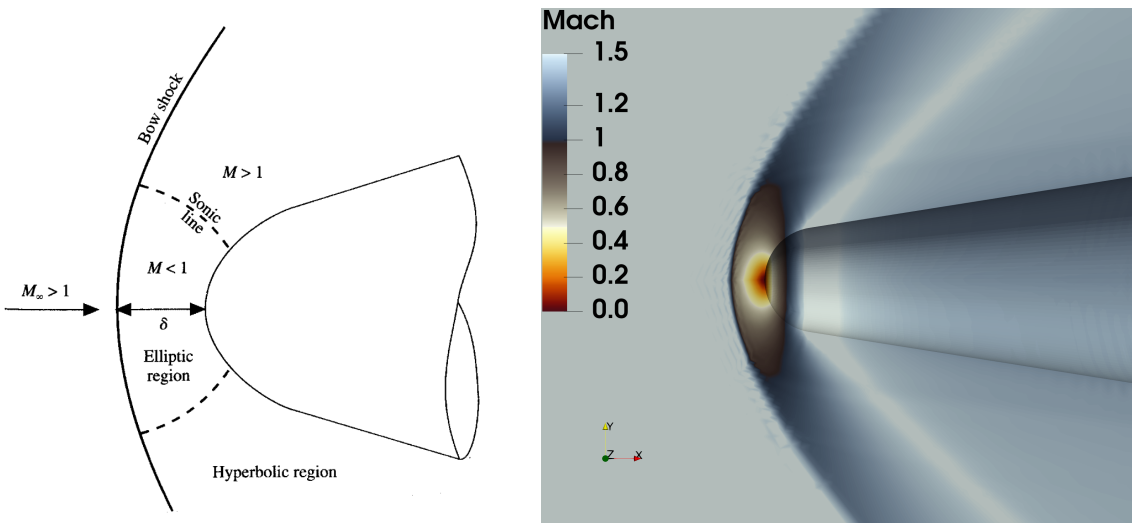
(b) Pressure field on ICEM-CFD mesh



(c) Velocity magnitude on GMDS mesh

(d) Velocity magnitude ICEM-CFD mesh

Figure 4.18: Pressure fields and velocity magnitude after simulations around Stardust geometry on Figure 4.17 meshes using legacy code.



(a) Schematic of the flowfield over a blunt body moving at supersonic or hypersonic speeds [And89]

(b) Mach field around RAM-C II nose

Figure 4.19: Mach field around vehicle stagnation point in case of supersonic flow simulations.

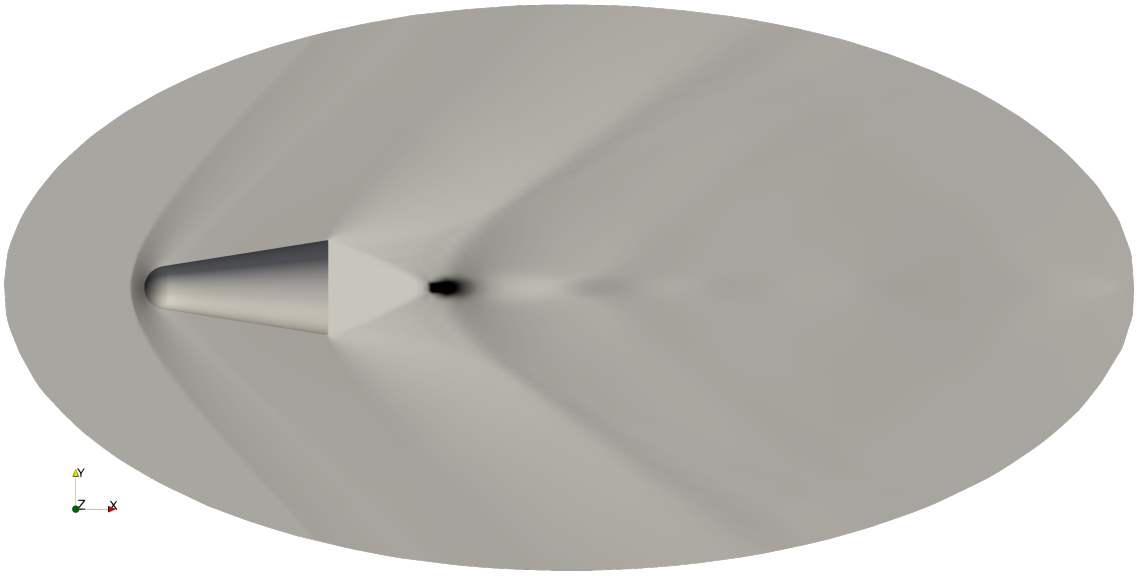


Figure 4.20: Undimensionalized pressure field around supersonic RAM-C II 3D vehicle using the simulation parameters of Table 4.1.

This simulation aims to show that 3D meshes generated with our algorithm pass the validity requirement for a CFD solver and are usable to capture some phenomena related to supersonic flows, such as the shock position, and the global flow topology around the vehicle.

Summary

In this chapter, we studied the mesh quality of meshes generated with the method presented in precedent chapters from two angles: using geometrical analysis of meshes n-cells and numerical simulation results performed on the meshes.

The first part was dedicated to purely geometrical quality criteria over the mesh cells. We presented multiple vehicles, and for each, different block structure topologies and corresponding final meshes were generated. This part illustrates the capabilities and flexibilities of our approach to generate various block structures and meshes around the same vehicle and analyses the impact on the mesh quality generated.

In the second part, we presented different results of numerical simulations using two different Computational Fluid Dynamics software: SU2, which is open-source software, and the Navier-Stokes solver from the French Alternative Energies and Atomic Energy Commission. The various examples of 2D and 3D subsonic, supersonic, and hypersonic flow simulations demonstrate the validity of our meshes as solvers can perform on them. In addition, comparing results obtained on our meshes to experimental data, analytical solutions, and reference solutions on other meshes illustrate that the meshes generated with our algorithm are usable to capture phenomena of interest.

This chapter's central perspective is to simulate hypersonic 3D flows around vehicles using the CEA legacy Navier-Stokes code.

CONCLUSION

In this work, we proposed a full and complete pipeline to generate hexahedral block-structured meshes for fluid domains around a single vehicle dedicated to Computational Fluid Dynamics (CFD) atmospheric re-entry study. More specifically, we proposed a suitable 2D method that is extensible to the 3D case and presented the general process in the nD case.

The first part of this work, presented in Chapter 2, focuses on generating a linear block structure using an advancing front approach. As we expected a strong control of mesh size and quality around the vehicle, we adopted an advancing front approach. We based our work on the previous work of Ruiz-Girones et al. [RG11, RGRS12], where they introduce the receding front method to generate a hexahedral mesh of the outer space around a vehicle, starting from a pre-meshed vehicle boundary, and a non-pre-meshed outer boundary, which allows to relax some constraints. We take an input first discretization of the domain into simplices as background mesh to compute some quantities necessary for the extrusion algorithm and the first surface block discretization of the vehicle boundary. Then, the algorithm extracts the block structure by generating the blocks layer by layer, starting from the vehicle surface. In this work, we introduce additional constraints due to our applications. This was performed to control the extrusion direction through a specific vector field, to control the first block layer with particular conditions, and to control the introduced blocks in the structure to improve the block's quality.

Once the linear block structure generated, we focused in Chapter 3 on the block curving stage. As we work with a block structure, we have a few hexahedral elements that do not represent the vehicle's geometric surface accurately. As we expect to generate a body-fitted mesh, if we split the blocks and project the nodes onto the surface, a time-consuming step would be mandatory on the internal nodes to improve the mesh quality. For this reason, we decided first to curve the block structure. To do so, we studied two different approaches. The first one is through *rp*-adaptivity, in the finite element framework MFEM [mfe, AAB+21, AAB+24]. Starting from a *r*-adaptivity pre-existing algorithm meant to do high-order mesh (with uniform degree over the domain) interface alignment to an implicit representation of a surface, we took it one step further to adapt the polynomial degree of elements around the interface in areas of higher curvature. This method provides good results in 2D and 3D for different mesh topologies and on problems of practical interest. However, in 3D, it remains slow for now. For this reason, we worked on a second approach to curve the blocks: using a Bézier representation of our blocks. After introducing the representation of the elements, we explained how we manage to curve the blocks to represent the vehicle surface and how we propagate the information in the volume to avoid overlapping cells. Considering we have a high-order Bézier block structure, we explained how we generate the resulting mesh using an interval assignment algorithm with specific constraints and using the parametrical space of the Bézier blocks.

Finally, in Chapter 4, we analyzed meshes generated with our method through two main axes: purely geometric criteria and numerical simulation results. For the first part, we presented a set of vehicles of interest, different block structure topologies obtained using our method, and the final mesh quality. Through this analysis, we show the impact of the method parameters on the final geometric mesh quality. However, we keep in mind that the main quality criterion is the numerical simulation result. The second part of this chapter was dedicated to the results of numerical simulations of flows around vehicles performed on meshes generated with our method. This section illustrated 2D and 3D simulation results using two different CFD codes. There are simulations of subsonic, supersonic, and hypersonic flows around vehicles. Results are compared to experimental data sets, analytical results, and reference results obtained on reference meshes. Through those numerical simulations, we illustrated that our meshes pass the validity solvers check (i.e., they can compute on the given meshes) and the results are coherent with the physical phenomena of interest.

1 Perspectives

We previously listed perspectives related to each chapter. In Chapter 2, we gave details about how we expect to:

- Handle multi-vehicle and more non-convex geometries through a specific pre-treatment process;
- Improve distance fields computation, for numerical accuracy and time-consumption purpose;
- Propose a way to generate the vehicle surface discretization automatically (i.e., the first front taken as an input in the 3D pipeline).

In Chapter 3, we proposed leads to:

- Improve the block edge curving to represent the vehicle surface, to not only focus on the approximation error, but also on the block edges direction;
- Extend the algorithm of Chapter 2 to generate directly a curved block structure using the advancing front process;
- Improve the edge size transition between two adjacent blocks of the block structure;
- Smooth the curved block structure to improve the block and resulting mesh quality, and to improve the continuity between cells of adjacent blocks.

The main perspective of Chapter 4 is to use one 3D block-structured mesh generated with our method for a numerical simulation using the French Alternative Energies and Atomic Energy Commission CFD code. In addition to the perspectives listed in each corresponding chapter, we introduce here one major perspective to this work. This work aims to generate a block-structured mesh before any simulation. The next step is to get the result of a first simulation, and to inject it in our generation process. Such a process may be qualified of "weak adaptation"

procedure: we iteratively mesh the whole domain and run the simulation code until getting the "most code-adapted" mesh. Among the information we could get back from a simulation code, we are particularly interested in:

1. the boundary layer thickness, to know if the first block layer thickness is sufficient to capture the phenomena, and to know if we need to add more mesh cells.
2. The shock position, to align blocks and mesh cells to the shock position, and refine if needed.

A similar attempt was recently done in 2D in [DWQ24].

A remaining question is also to define what means to have the "most code-adapted" mesh. We think that there is not a unique answer; and specific criteria should be defined for every simulation code. Studying works done on triangular and tetrahedral mesh adaptation techniques seems mandatory [FA05, LAA10, AL16].

APPENDIX

Contents

A Transfinite Interpolation

Algorithm 12 Linear Transfinite Interpolation 2D

Require: $\mathbf{G}[0 : m-1, 0 : n-1]$ a 2D grid of $m \times n$ points, with the positions of the points initialized on the four boundaries of the grid

Ensure: $\mathbf{G}$ with inner positions of points set

$d_i \leftarrow 1/(m-1)$
$d_j \leftarrow 1/(n-1)$
for all $i \in 1, ..., m-2$ **do**
  $c_i \leftarrow i \times di$
  for all $j \in 1, ..., n-2$ **do**
    $c_j \leftarrow j \times dj$
    $\mathbf{G}[i,j] \leftarrow (1-c_i)\mathbf{G}[0,j] + c_i\mathbf{G}[m-1,j] + (1-c_j)\mathbf{G}[i,0] + c_j\mathbf{G}[i,n-1]$
    $-(1-c_i)(1-c_j)\mathbf{G}[0,0] - (1-c_i)c_j\mathbf{G}[0,n-1]$
    $-c_i(1-c_j)\mathbf{G}[m-1,0] - c_ic_j\mathbf{G}[m-1,n-1]$
  end for
end for

Algorithm 13 Linear Transfinite Interpolation 3D

Require: $\mathbf{G}[0:m-1,0:n-1,0:p-1]$ a 3D grid of $m \times n \times p$ points, with the positions of the points initialized on the six boundaries of the grid

Ensure: $\mathbf{G}$ with inner positions of points set

$d_i \leftarrow 1/(m-1)$
$d_j \leftarrow 1/(n-1)$
$d_k \leftarrow 1/(p-1)$
for all $i \in 1, ..., m-2$ **do**
  $c_i \leftarrow i \times di$
  for all $j \in 1, ..., n-2$ **do**
    $c_j \leftarrow j \times dj$
    for all $k \in 1, ..., p-2$ **do**
      $c_k \leftarrow k \times dk$
      $\mathsf{U} \leftarrow (1-c_i)\mathbf{G}[0,j,k] + c_i\mathbf{G}[m-1,j,k]$
      $\mathsf{V} \leftarrow (1-c_j)\mathbf{G}[i,0,k] + c_j\mathbf{G}[i,n-1,k]$
      $\mathsf{W} \leftarrow (1-c_k)\mathbf{G}[i,j,0] + c_k\mathbf{G}[i,j,p-1]$
      $\mathsf{UW} \leftarrow (1-c_i)(1-c_k)\mathbf{G}[0,j,0] + (1-c_i)c_k\mathbf{G}[0,j,p-1]$
        $+c_i(1-c_k)\mathbf{G}[m-1,j,0] + c_ic_k\mathbf{G}[m-1,j,p-1]$
      $\mathsf{UV} \leftarrow (1-c_i)(1-c_j)\mathbf{G}[0,0,k] + c_ic_j\mathbf{G}[m-1,n-1,k]$
        $+c_i(1-c_j)\mathbf{G}[m-1,0,k] + (1-c_i)c_j\mathbf{G}[1,n-1,k]$
      $\mathsf{VW} \leftarrow (1-c_j)(1-c_k)\mathbf{G}[i,0,0] + (1-c_j)c_k\mathbf{G}[i,0,p-1]$
        $+c_j(1-c_k)\mathbf{G}[i,n-1,0] + c_jc_k\mathbf{G}[i,n-1,p-1]$
      $\mathsf{UVW} \leftarrow (1-c_i)(1-c_j)(1-c_k)\mathbf{G}[0,0,0] + (1-c_i)(1-c_j)c_k\mathbf{G}[0,0,p-1]$
        $+(1-c_i)c_j(1-c_k)\mathbf{G}[0,m-1,0] + c_i(1-c_j)c_k\mathbf{G}[m-1,0,p-1]$
        $+c_ic_j(1-c_k)\mathbf{G}[m-1,n-1,0] + c_ic_jc_k\mathbf{G}[m-1,n-1,p-1]$
        $+(1-c_i)c_jc_k\mathbf{G}[0,n-1,p-1] + c_i(1-c_j)(1-c_k)\mathbf{G}[m-1,0,0]$
      $\mathbf{G}[i,j,k] \leftarrow \mathsf{U} + \mathsf{V} + \mathsf{W} - \mathsf{UW} - \mathsf{UV} - \mathsf{VW} + \mathsf{UVW}$
    end for
  end for
end for

B Geometries Guide

In this Appendix, we present geometries studied in different works linked to hypersonic Computational Fluid Dynamics, used here to experiment with our meshing algorithm. More details are available about the origins of the geometries and numerical and physical results but we only provide here the geometries description to allow reproducibility and comparison with our meshing method[2].

B.1 Apollo Experimental Model

According to SCALABRIN [Sca07], this geometry comes from an experimental study of hypersonic flows during the 1960s to validate models used in the design of the Apollo spacecraft [GB68]. This geometry is included in our geometry database of interest. This geometry is reproducible using Figure B.1 with the nose radius $R_n = 168.86$mm, the base radius $R_b = 70.36$mm, the cap radius $R_c = 8.37$mm, the shoulder radius $R_s = 7.04$mm, the length of the capsule $L = 123.28$mm, and the cone angle $\alpha_c = 33^\circ$. The 3D geometry may be obtained by a complete revolution of this 2D geometry around the axis.

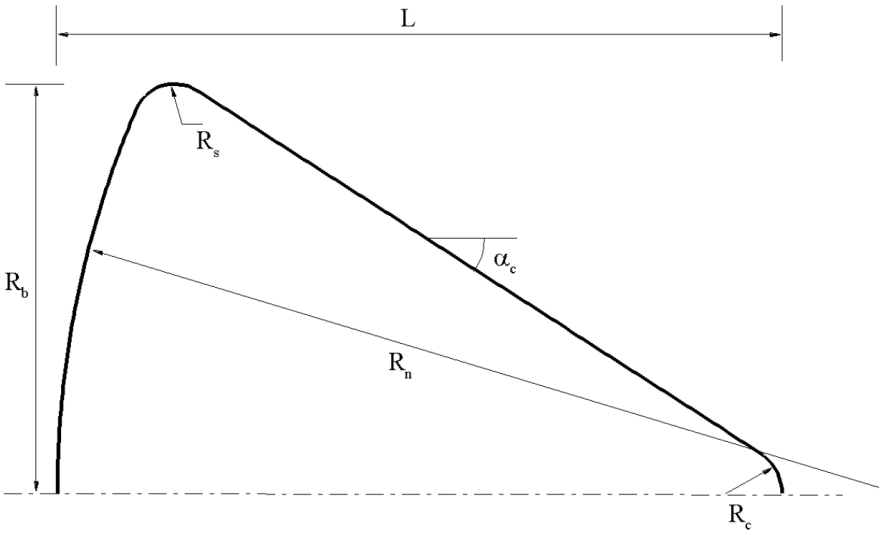


Figure B.1: Apollo experimental model [Sca07].

B.2 Caretwing

The Caretwing geometry (see Fig. B.2) is given in [Küc65]. Here, it is reproduce using $\delta = \arctan(0.1)$, $\tau = 0.06$, $l = 1.0$, $S = \frac{1.0}{(30.0\tau)^2}$, and $\sigma = \frac{10.162570\pi}{180.0}$.

[2]Some geometries are available on https://github.com/claireroche/Geometries, with different file formats.

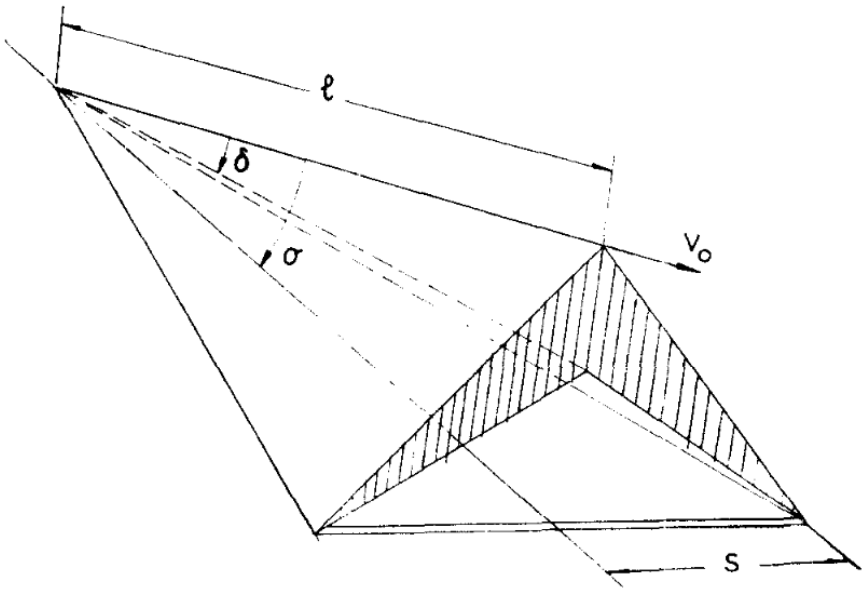


Figure B.2: Caretwing [Küc65].

B.3 Double Ellipsoid

The analytical description of the geometry of Figure B.3 is provided in [DGP12] and given here as a reminder:

$$\begin{cases} \left(\frac{x}{0.06}\right)^2 + \left(\frac{y}{0.025}\right)^2 + \left(\frac{z}{0.015}\right)^2 = 1 & \text{if } x \leq 0, \\ \left(\frac{x}{0.035}\right)^2 + \left(\frac{y}{0.0175}\right)^2 + \left(\frac{z}{0.025}\right)^2 = 1 & \text{if } x \leq 0, \text{ and } 0 \leq z, \\ \left(\frac{y}{0.025}\right)^2 + \left(\frac{z}{0.015}\right)^2 = 1 & \text{if } 0 \leq x \leq 0.016, \\ \left(\frac{y}{0.025}\right)^2 + \left(\frac{z}{0.015}\right)^2 = 1 & \text{if } 0 \leq x \leq 0.016, \\ \left(\frac{y}{0.0175}\right)^2 + \left(\frac{z}{0.025}\right)^2 = 1 & \text{if } 0 \leq z. \end{cases} \tag{4.3}$$

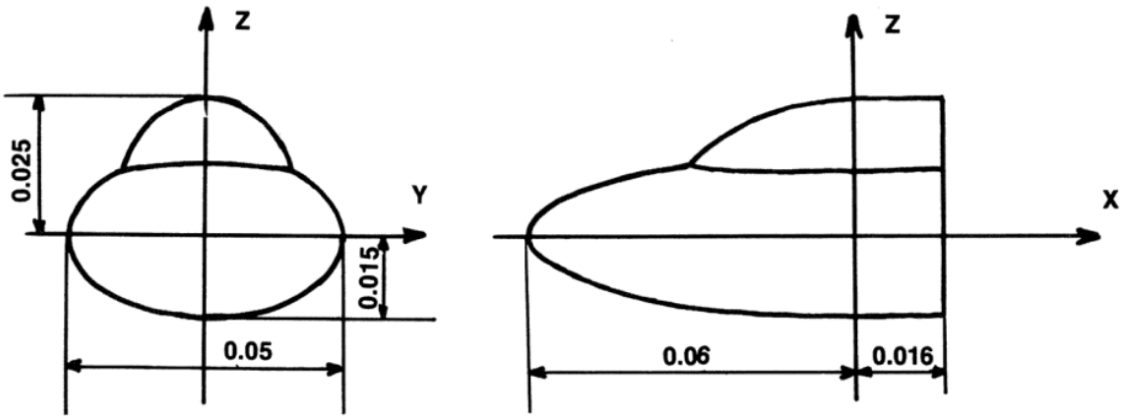


Figure B.3: Double ellipsoid [DGP12].

B.4 HIFiRE-5

According to [JAK15], HIFiRE-5 (see Fig. B.4) is a dedicated geometry for aerothermodynamic experiments. It is an elliptic cone with a $2:1$ aspect ratio. It has a 7° half-angle on the minor axis and a 2.5mm radius nosetip. The payload instrumented elliptic cone section length is 0.86m.

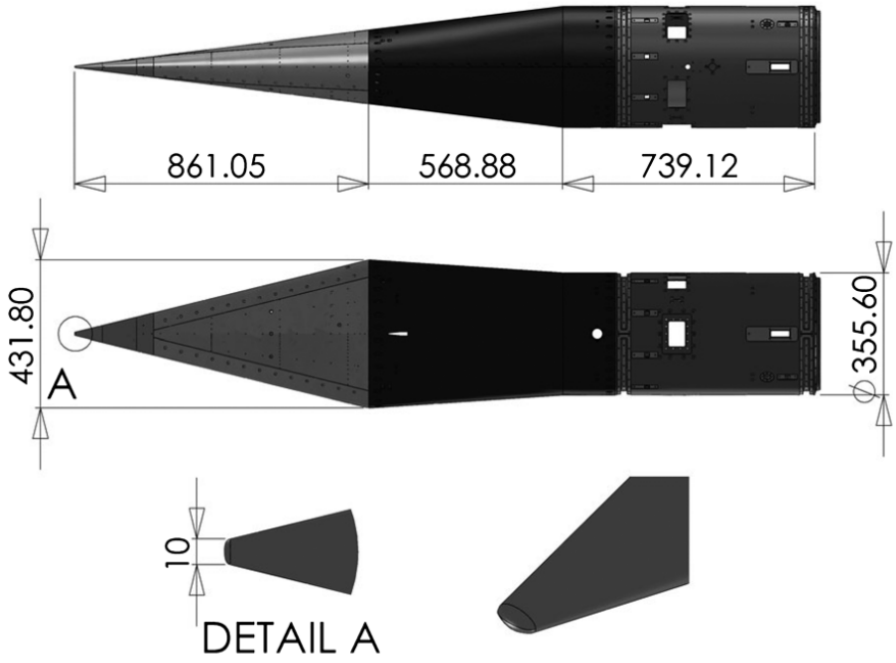


Figure B.4: HIFiRE-5 vehicle with dimensions in mm [JAK15].

B.5 Hypersonic Transition Research Vehicle (HyTRV)

The Hypersonic Transition Research Vehicle (HyTRV) is a 3D geometry described in [QLYT21]. The vehicle's shape is illustrated in Figure B.5. The length of the body is $1,600$mm, its head is a $1:2:1$ ellipsoid, with its major and minor axis respectively 80mm and 40mm. The bottom half section of the body is a $1:4$ ellipse and the major and minor axis are 60mm and 150mm, respectively. The upper part of the lifting body is given by a Class/Shape Function Transformation (CST) curve and elliptic curve. The upper part curve function is:

$$z = \alpha \cdot z_C + (1-\alpha) \cdot z_E, \tag{4.4}$$

where z is the vertical coordinate, $\alpha = 0.72$, z_E is the coordinate given by the $1:4$ ellipse, z_C is the coordinate given by the CST function, and the analytical formula of the CST is:

$$\begin{cases} z_C &= \zeta \cdot Z_x \\ \zeta &= 2^8 \cdot \eta^4 \cdot (1-\eta)^4, \ \eta \in [0,1] \\ y &= (\eta - \frac{1}{2}) \cdot Y_x \end{cases} \tag{4.5}$$

$Z_x = 270$mm is the upper surface curve height, and $Y_x = 600$mm is the bottom surface width.

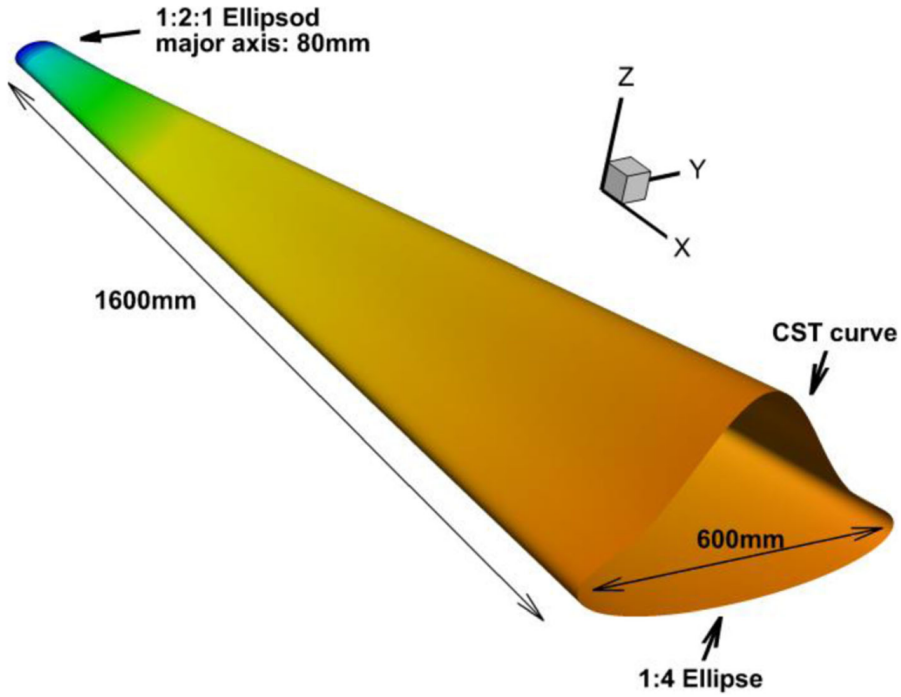


Figure B.5: Hypersonic Transition Research Vehicle (HyTRV) [QLYT21].

B.6 Mars Entry Spacecraft Experimental Model

According to SCALABRIN [Sca07], this geometry is an experimental model used in a wind tunnel. This geometry is reproducible using Figure B.6 where the nose radius $R_n = 12.7$mm, the base radius $R_b = 25.4$mm, the corner radius $R_c = 1.27$mm, the frustum radius $R_f = 15.24$mm, the sting radius $R_s = 10.32$mm, the sting length $L_s = 116.86$mm, the nose cone $\alpha_n = 70^\circ$, and the frustum cone angle $\alpha_f = 40^\circ$. The 3D geometry may be obtained by a full revolution of this 2D geometry around the symmetry axis.

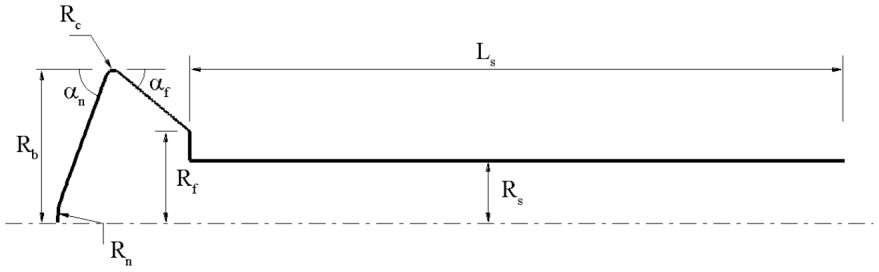


Figure B.6: Mars entry spacecraft model [Sca07].

B.7 NACA 0012 Airfoil

The NACA 0012 airfoil is a well-known and studied geometry [Rum21]. The exact NACA 0012 airfoil formula is given by

$$y = \pm 0.6(0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1015x^4). \tag{4.6}$$

Figure B.7: NACA 0012 airfoil [Rum21].

B.8 RAM-C II Spacecraft

According to SCALABRIN [Sca07], this geometry comes from 1960's communications blackout studies. To reproduce the geometry, we can use Figure B.8 where the nose radius $R_n = 0.1524$m, the cone angle $\alpha_c = 9^\circ$, and the vehicle length is $L = 1.3$m.

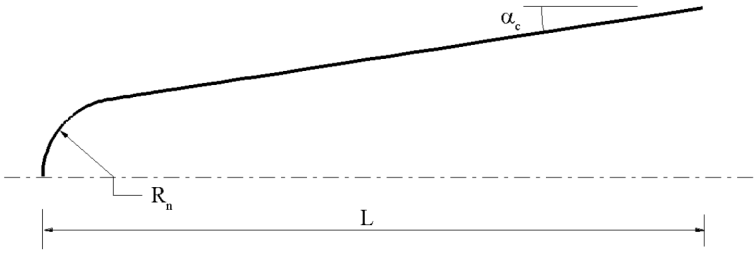


Figure B.8: RAM-C II spacecraft [Sca07].

B.9 Sphere Cone Cylinder Flare (CCF)

(a) CCF nose (b) CCF vehicle

Figure B.9: Sphere-Cone-Cylinder-Flare (CCF) vehicle.

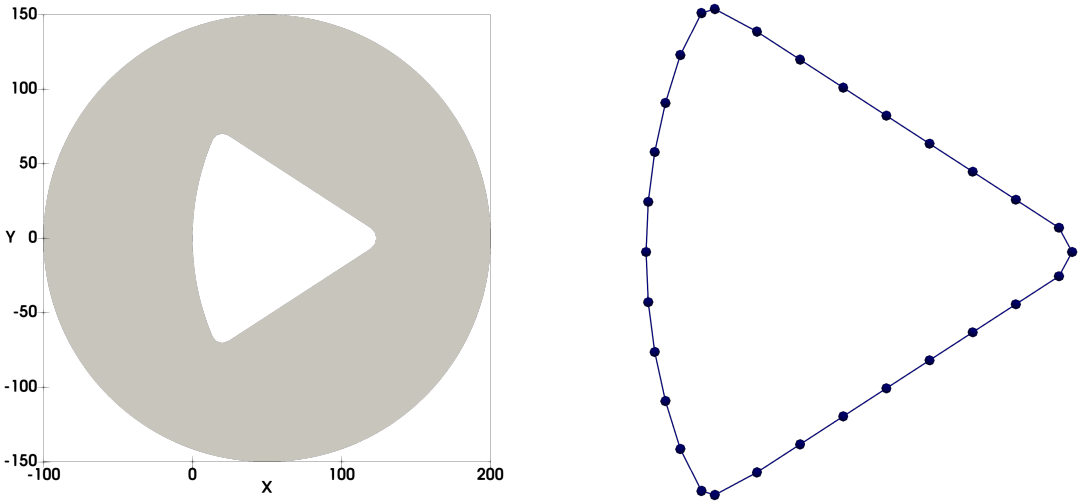
The Sphere-Cone-Cylinder-Flare (CCF) vehicle of Figure B.9 presented in [EBSB19] is an axisymmetric geometry (i.e., the 3D model is obtained by a rotation around the symmetry axis). In addition to the main characteristics given in Table 4.5, they define the small nose radius $R_n = 0.1$mm (see Fig. B.9.a), the half-cone angle is 5°, and the flare angle is equal to 3.5°.

	x (m)	y (m)
Nose-Cone junction	0.0920×10^{-3}	0.0997×10^{-3}
Cone-Cylinder junction	0.398147	0.034925
Cylinder-Flare junction	0.525670	0.034925
End of flare	0.764456	0.049530

Table 4.5: Sphere-Cone-Cylinder-Flare model coordinates [EBSB19].

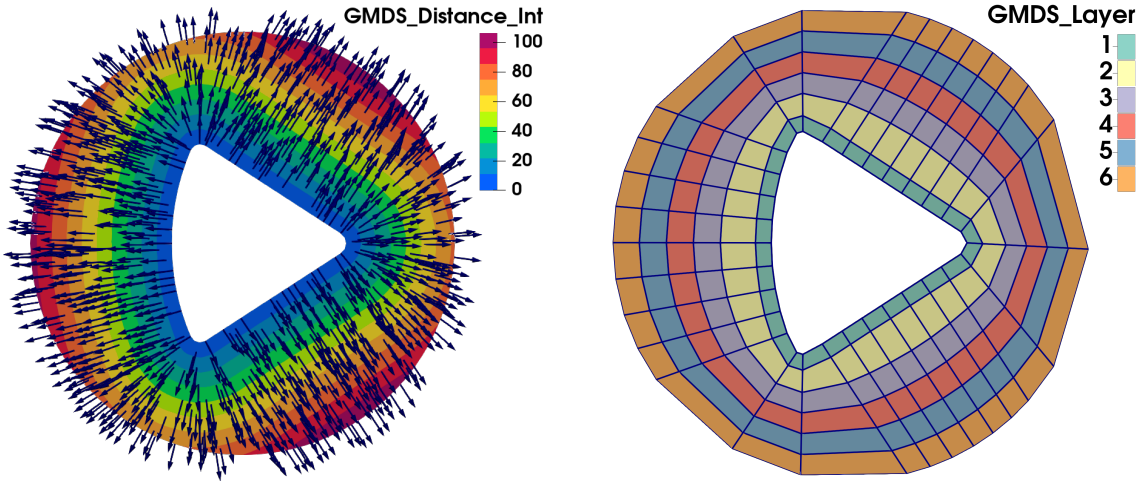
C Vector Field Impact Analysis

This Appendix gives examples of the different vector fields $\overrightarrow{v}$ choice of computation in 2D and the impact on the resulting block structure. We focus here on the linear block structure generation around the Apollo vehicle [Sca07] (see Appx B). The physical domain to mesh is represented in gray ( ) in Figure C.10.a, where the far-field is a circle. We do not vary the domain in this part, and the vehicle wall discretization used is always the same (see Fig. C.10.b). Each time, the block structure is extruded on 6 different layers, using the same parameters, except for the vector field computation. For the next figures, the vector field $\overrightarrow{v}$ is represented using dark blue vectors ( ). For visibility purposes, the vectors are not plotted at each $\mathcal{M}_T$ node but only on $1,500$ selected nodes.



(a) Physical domain to mesh (b) Vehicle wall discretization

Figure C.10: Physical 2D domain to mesh around Apollo vehicle (a), starting from the block discretization (b).

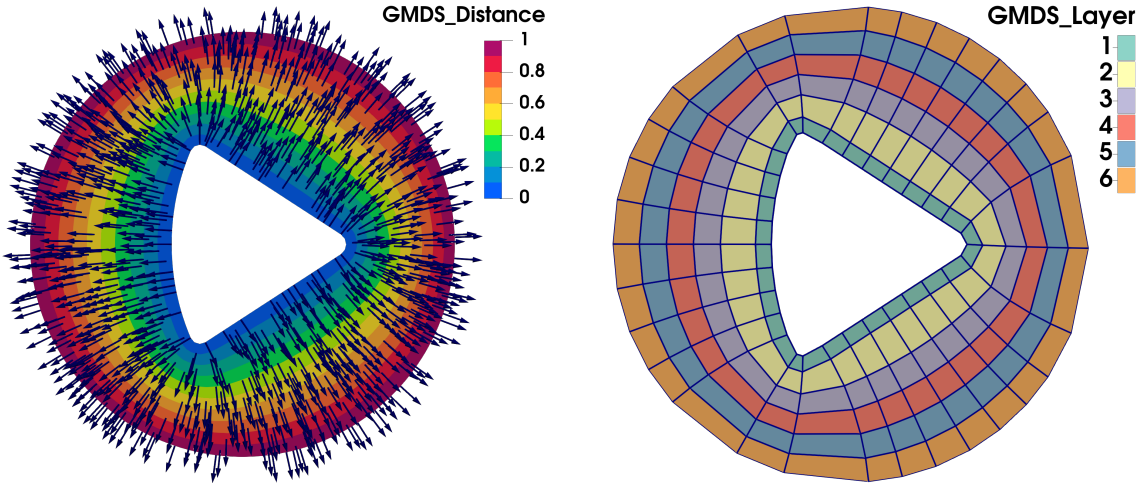


(a) Distance field d_V and vector field ∇d_V on $\mathcal{M}_T$ (b) Final blocks

Figure C.11: Block structure generation around Apollo geometry using vector field $\overrightarrow{v} = \nabla d_V$.

The first naive example is the generation of the block structure using the gradient of the

scalar distance field from the vehicle ∇d_V as the vector field to lead the extrusion over all the domain (see Fig. C.11.a). This field is orthogonal to the vehicle wall by construction, so it ensures the block edges orthogonality to the wall on the first layer of blocks ( ). Figure C.12 represents a second example of block structure generation, where this time the vector field $\overrightarrow{v}$ is chosen equal to the combined distance field gradient ∇d. The resulting block structure (see Fig. C.12.b) has an elliptic shape while the precedent one (see Fig. C.11.b) can be considered more straight.



(a) Distance field d and vector field ∇d on $\mathcal{M}_T$ (b) Final blocks

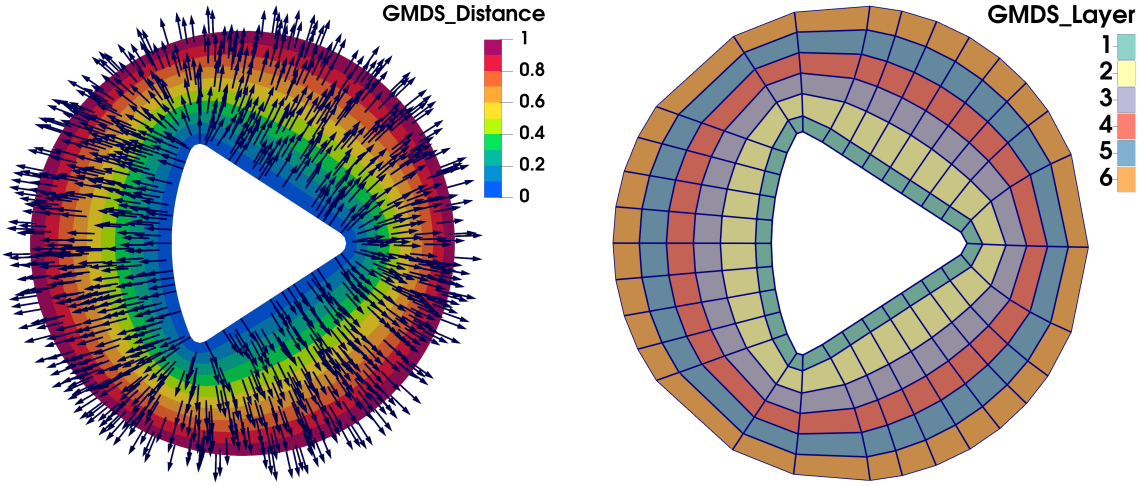
Figure C.12: Block structure generation around Apollo geometry using vector field $\overrightarrow{v} = \nabla d$.

We also decided to exhibit three examples of vector field combinations here. The first one plotted in Figure C.13.a is a combination of the two precedent vector fields ∇d_V and ∇d (see Fig. C.13.a). The damping area considered is between $x_f = 0.0$ and $x_b = 50.0$ (see Fig. C.10.a). As expected, we observe the straight behavior of the first test case Figure C.11.b on the front part of the domain (see Fig. C.13.b), while we observe the more elliptic block structure on the back part of the domain (see Fig. C.13.b). We can easily imagine the result by inverting the two different fields to obtain an elliptic-like block structure on the front part, using ∇d, while having straight blocks on the back part using ∇d_V, and adjusting the transition limits as wanted. This generates many possibilities, but we do not use this combination of fields in practice.

In fact, we are more interested in aligning the back part of the block structure to the flow direction. To do so, we usually combine ∇d or ∇d_V as the front vector field $\overrightarrow{v_f}$, with the constant vector field $\overrightarrow{u_\infty}$ as the back vector field $\overrightarrow{v_b}$. Figure C.14 represents the combination of $\overrightarrow{v_f} = \nabla d$ with $\overrightarrow{v_b} = \overrightarrow{u_\infty}$. In this case, the Angle of Attack (AoA) used to compute $\overrightarrow{u_\infty}$ is $\alpha = 0^\circ$. The transition area is between $x_f = 50.0$, and $x_b = 200.0$ If we compare this result to the one of Figure C.12.b where the block structure was generated using ∇d_V over all the domain, we may see the blocks on the back part trying to align.

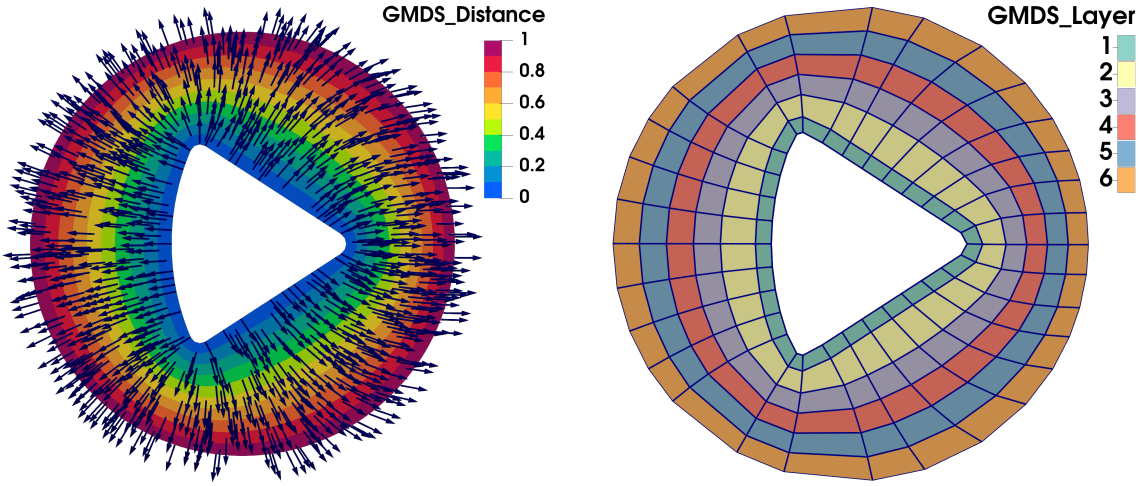
The last result presented in this part is Figure C.15. As for the one before, we combine here the gradient of a distance field, which is ∇d_V this time, to the flow direction $\overrightarrow{u_\infty}$. This time, the damping zone is between $x_f = 80.0$, and $x_b = 200.0$. Here, the AoA is set to $\alpha = 30^\circ$. As expected, it impacts the back block structure, with blocks trying to align with this deviation.

This short study shows multiple possibilities for computing the vector field easily, and we



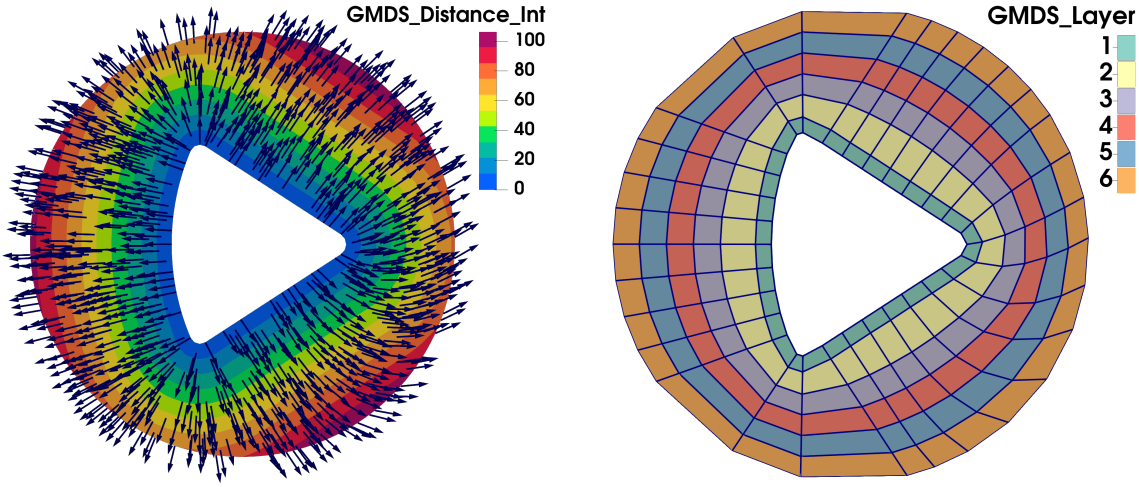
(a) Distance field d and vector field $\overrightarrow{v}$ on $\mathcal{M}_T$ (b) Final blocks

Figure C.13: Block structure generation around Apollo geometry using combined vector field $\overrightarrow{v}$ where $\overrightarrow{v_f} = \nabla d_V$ and $\overrightarrow{v_b} = \nabla d$, with $x_f = 0.0$ and $x_b = 50.0$.



(a) Distance d and vector fields $\overrightarrow{v}$ on $\mathcal{M}_T$ (b) Final blocks

Figure C.14: Block structure generation around Apollo geometry using combined vector field $\overrightarrow{v}$ where $\overrightarrow{v_f} = \nabla d$ and $\overrightarrow{v_b} = \overrightarrow{u_\infty}$, with $x_f = 50.0$ and $x_b = 200.0$. With an Angle of Attack (AoA) $\alpha = 0^\circ$.



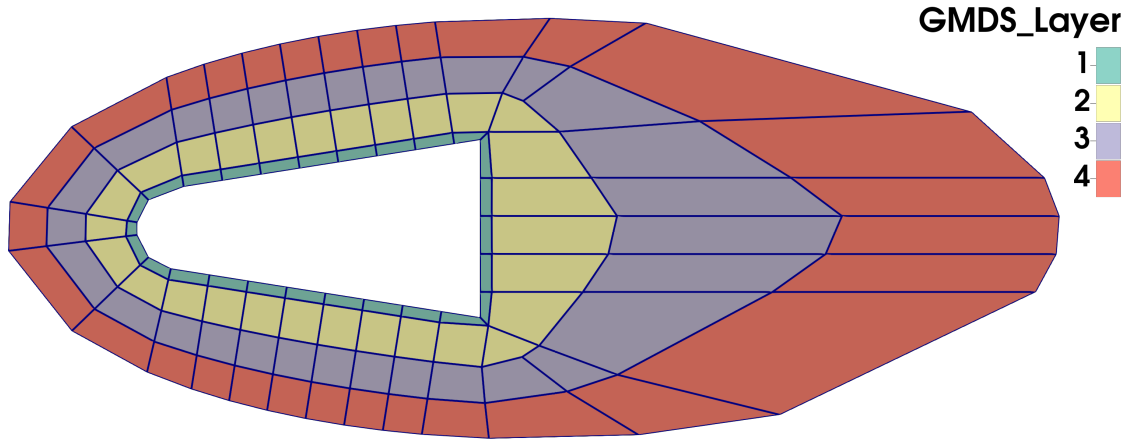
(a) Distance d_V and vector fields $\overrightarrow{v}$ on $\mathcal{M}_T$ (b) Final blocks

Figure C.15: Block structure generation around Apollo geometry using combined vector field $\overrightarrow{v}$ where $\overrightarrow{v_f} = \nabla d_V$ and $\overrightarrow{v_b} = \overrightarrow{u_\infty}$, with $x_f = 80.0$ and $x_b = 200.0$. With an Angle of Attack (AoA) $\alpha = 30^\circ$.

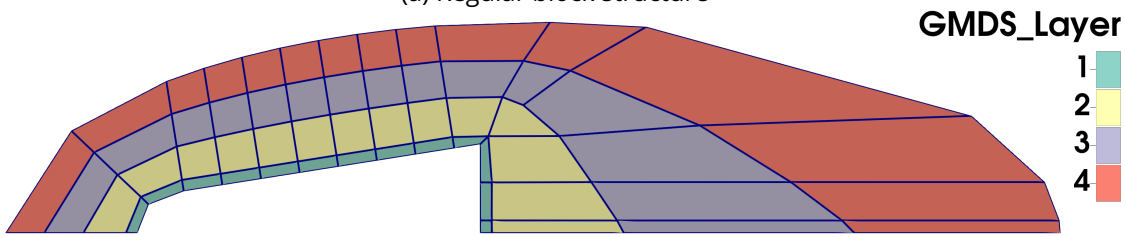
may imagine other combinations. Here, we presented the main combinations used in the manuscript. If we don't encode one version of the vector field computation, it is because, in some cases, we prefer one version to another. For instance, in the case of an anisotropic domain behind the vehicle, some vector fields help to propagate the block structure in the domain, while others fail.It is essential to understand that if the vector field computation gives flexibility in the method, it is a sensitive step. As the blocks follow the vector field to generate, a vector field such as constant $\overrightarrow{u_\infty}$ over the domain would lead to self-intersecting blocks and non-meshed parts of the domain. We can note that for this 2D case, the execution time of the whole algorithm (including the discretization of the wall surface, the computation of distance and vector fields, the block structure generation, the block structure curving, and the mesh generation) does not exceed 8 seconds. So, if the vector field is not well conditioned, changing the parameters and re-running the generation remains reasonable.

D Axisymmetry Block-Structured Mesh Generation

Using a user parameter, it is possible to try to generate the equivalent of an axisymmetric block structure and resulting mesh in 2D. This relies on a pre-process and a post-process procedure. As a pre-process, we first check on the discretization of the vehicle surface if there are block corners on the symmetric axis. If not, the block corners are created. We still generate the block structure all over the vehicle with the condition to remain on the axis for block corners of a front generated on previous block corners corresponding to the axisymmetric constraint (as there is no reason the block corners stay on the axis, depending on the input vector field).



(a) Regular block structure



(b) Axisymmetric block structure

Figure D.16: Axisymmetric block structure generation around RAM-C II (see Appx. B).

We add a post-process step to cut half of the resulting block structure to provide the axisymmetric mesh. We note here that it can fail because the method was not meant to handle those cases. For instance, the structure cannot be cut if there is an insertion or a fusion of blocks on the block corner on the axis.

Figure D.16 exhibits an example of a block structure generation around RAM-C II vehicle (see Fig. D.16.a). Using the exact same parameters but by choosing the axisymmetric option, the resulting block structure is given in Figure D.16.b. Looking at the wall surface discretization, we can see the two additional block corners added on each vehicle side on the axis. The resulting mesh generated on the linear block structure Figure D.16.b is plotted on Figure D.17. The block structure degree is elevated to $p=2$.

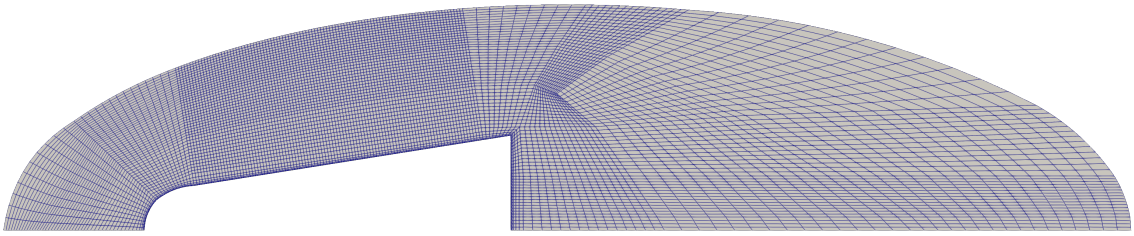
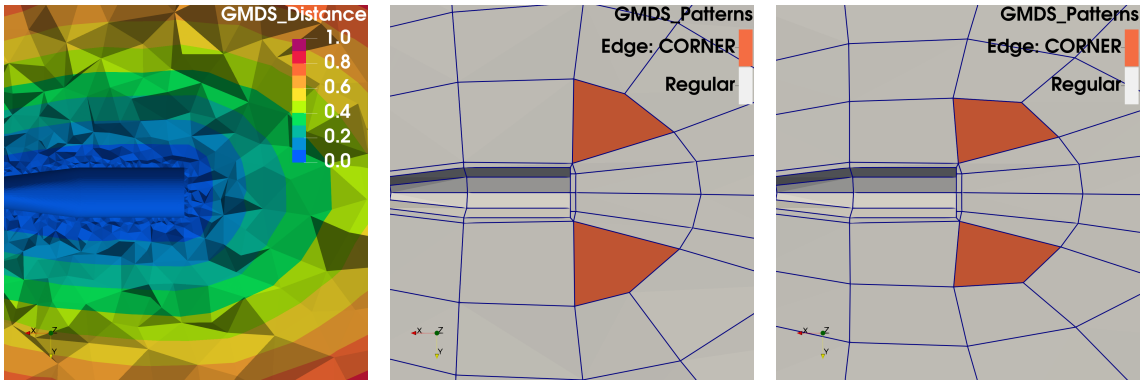


Figure D.17: Axisymmetric mesh generated around RAM-C II (see Appx. B) on Figure D.16.b block structure topology.

E Block Structure Smoothing

As every block corner of the same front is located on the same isoline in the distance field (see Fig. E.18.a), it may result in flattened blocks when patterns are used on a layer (see Fig. E.18.b, with flattened orange blocks (■) inserted on the second block layer). This phenomenon does not appear on the first block layer, corresponding to the boundary block layer, because the selected distance field is set to d_V (i.e., the Euclidean distance to the vehicle). Also, the block quality may be increasingly poor during the process over the layers. This is mainly due to approximation during the trajectory computation due to a coarse background mesh and a naive interpolation formula over the tetrahedral cells.



(a) Combined distance field (b) Without smoothing (c) 10 smoothing iterations

Figure E.18: Non-smoothed block structure (b) around HIFiRE-5 (see Appx. B) and smoothed block structure (c). Block corners of (b) inserted orange blocks (■) are located on the same isoline (a).

For these two reasons, we add a geometric local iterative smoothing step on the linear block structure. This iterative process is weighted according to the block corner front value. This way, the positions of the block corners on the first layers, around the vehicle surface, are barely moved, while block corners on the outer boundary benefit from much freedom.

At each iteration i, for each block corner n_c at position p_c in the block structure $\mathbf{B}$, we compute the new position of the block corner according to:

$$p_c^{i+1} = (1-\theta)p_c^i + \theta\frac{1}{N_c}\sum_{n_k \in S_c} p_k^i, \tag{4.7}$$

where p_c^i is the position of n_c at iteration i, S_c is the set of adjacent block corners to n_c[3], N_c is the size of S_c, and θ is a damping parameter computed according to the front index of n_c in the block structure. Here,

$$\theta = 0.3\frac{f_c}{N_{\mathcal{L}}}, \tag{4.8}$$

where $N_{\mathcal{L}}$ is the total layers number, and f_c is the index of the front to which the block corner n_c belongs. This smoothing only applies for block corners with a front index $f_c > 1$. The block corners located on the vehicle surface (at $f_c = 0$) and on the first block layer corresponding to the boundary layer (at $f_c = 1$) remain at the exact same location during the whole process. For the block corner located on the last front, the damping parameter is equal to 0.3. The

[3]Two block corners are adjacent if they share one block edge

adjacent block corners selected to smooth the block corners of this specific front are not the same (see Alg. 14).

In practice, 4 iterations of the Algorithm 14 are sufficient to smooth the block structure. However, more iterations may be necessary on some specific block structures made of many blocks. Figure E.18.c illustrates the result of Figure E.18.b block structure after 10 iterations of the smoothing algorithm.

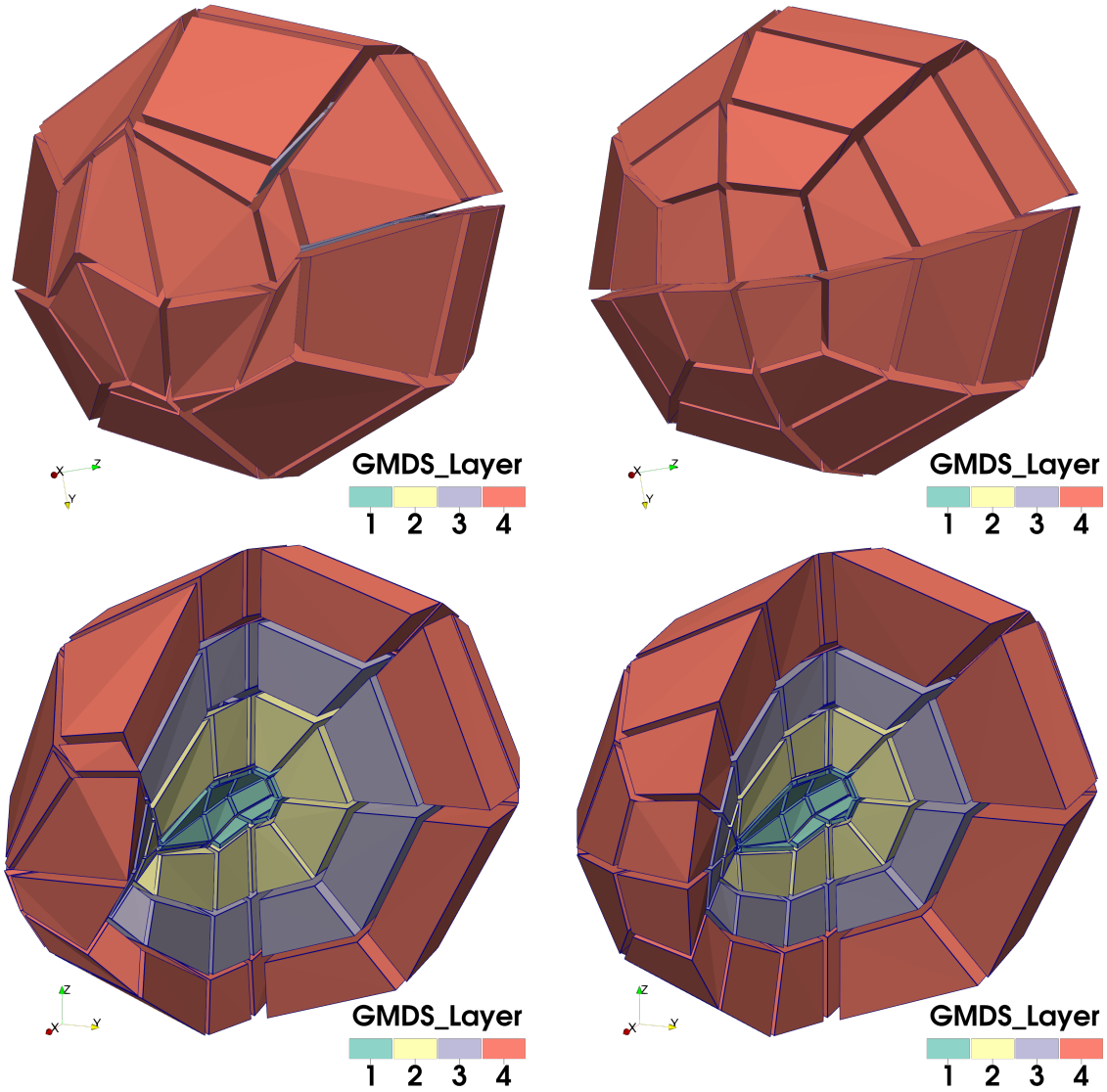
Algorithm 14 Linear Block Structure Smoothing

Require: Block structure $\mathbf{B}$, number of iterations N
Ensure: Smoothed block structure $\mathbf{B}$

$N_{\mathcal{L}} \leftarrow \mathbf{B}$.nbr_layers()
$i \leftarrow 0$
$\mathbf{B}_0 \leftarrow \mathbf{B}$
$\mathbf{B}_1 \leftarrow \mathbf{B}$
while $i < N$ **do**
  $f_c \leftarrow \mathbf{B}$.front_value($n_c$)
  for all block corner n_c in $\mathbf{B}$ **do**
    $p \leftarrow (0, 0, 0)$
    $N_c \leftarrow 0$
    $\theta \leftarrow 0.3\ f_c$ / $N_{\mathcal{L}}$
    if $f_c < N_{\mathcal{L}}$ **then**
      for all block corner n_k adjacent to n_c **do**
        $p \leftarrow p + \mathbf{B_0}$.point($n_k$)
        $N_c \leftarrow N_c + 1$
      end for
    else if $f_c = N_{\mathcal{L}}$ **then**
      for all block corner n_k adjacent to n_c and located on last front **do**
        $p \leftarrow p + \mathbf{B_0}$.point($n_k$)
        $N_c \leftarrow N_c + 1$
      end for
    end if
    $\mathbf{B_1}$.point(n_c) $\leftarrow (1 - \theta)\mathbf{B_0}$.point($n_c$) $+ \theta p / N_c$
  end for
  $\mathbf{B}_0 \leftarrow \mathbf{B}_1$
  $i \leftarrow i + 1$
end while
$\mathbf{B} \leftarrow \mathbf{B}_1$

Figure E.19.a and Figure E.19.b represent the same block structure than in Figure E.18.b and Figure E.18.c respectively. Here, we see the impact on the last layer blocks' shapes before and after smoothing. The more smoothing steps are performed, the more we lose the alignment

to the vector field in some parts of the physical domain. However, this does not affect the first layer blocks.



(a) Without smoothing (b) With smoothing

Figure E.19: Non-smoothed block structure (a) around HIFiRE-5 (see Appx. B) and smoothed block structure (b).

F Bézier Block Edge Curving Analysis

In this appendix, we compare different methods to compute the positions of control points of a Bézier curve to approximate a geometric curve. We keep in mind that we study methods in 2D, but we want them to be extendable to the 3D case, knowing we do not have access to a parametric representation of the geometric surface to approximate. This analysis explores different naive methods to compute control point positions of Bézier edges to approximate a given geometric curve. It illustrates why the way we decide to compute these positions is not sufficient and why the problem may not be easy.

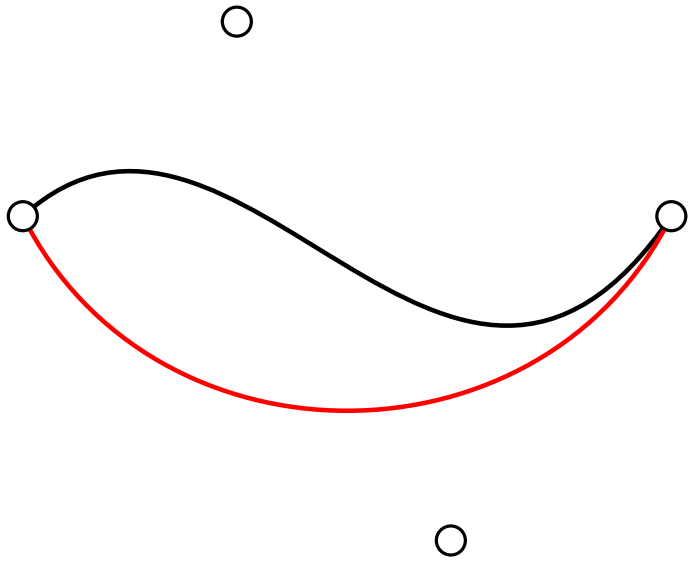


Figure F.20: Geometric curve ( ), and curved Bézier block edge ( ) defined by its white control points (○).

Knowing a geometric curve ( ), we want a strategic way to compute the positions of the control points $P_0, ..., P_p$ (○) of a Bézier edge of degree p to minimize the error between the Bézier edge ( ) and the geometric curve.

Let us remind that a Bézier curve of degree p can be expressed in the parametric space with its control points P_i by the relation

$$\gamma(u) = \sum_{i=0}^{p} B_i^p(u) P_i \tag{4.9}$$

where $u \in [0, 1]$, and B_i^p is the $i-th$ Bernstein polynomial of degree p, defined by:

$$B_i^p(u) = \binom{p}{i} u^i (1-u)^{p-1} \tag{4.10}$$

where $\binom{p}{i} = \frac{p!}{i!(p-i)!}$ is a binomial coefficient.

If we take a set of $p+1$ points $\{A_i\}_{0 \leq i \leq p}$ to interpolate, we can solve a system to compute the positions of the control points P_i needed to build the Bézier curve of degree p to interpolate the points A_i. In the next section, given a Bézier curve of arbitrary degree $p > 0$, we propose 3 different methods to compute the set of control points P_i to interpolate a geometric curve.

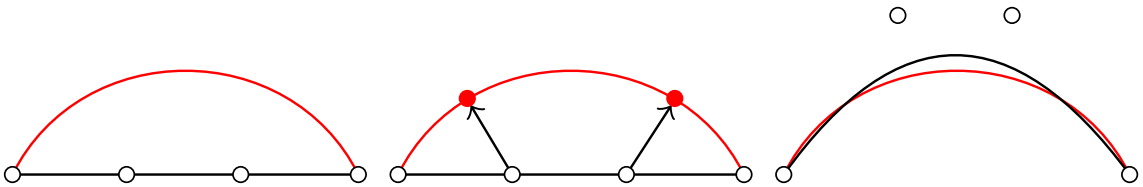
F.1 Methods to Curve Bézier Block Edges

Here, we present three different naive methods to compute the positions of the control points of a Bézier curve of arbitrary degree p to approximate a geometric curve. We are aware that different ways exist to process, and probably more accurately. However, we expect to show the impact of the way the control points are computed on the resulting block edge.

Method 1

The first method is, knowing we want to build a Bézier edge of degree p, we initialize the $p+1$ control points $P_0, ..., P_p$ in a uniform way on the linear edge (see black edge (▬) of Fig. F.21.a). For each control point P_i, we compute its projected position A_i (●) onto the geometric curve (▬) we want to approximate (see Fig. F.21.b). From this set of points $A_0, ..., A_p$, we solve a system to compute the positions of the control points $P_0, ..., P_p$ to interpolate the positions $A_0, ..., A_p$. To do so, we consider the control points $\{P_i\}_{i \in [\![0,p]\!]}$ set uniformly in the parametric space of the curve $[0, 1]$. To compute the positions of the final control points (see Fig. F.21.c), we simply invert the following system of $p+1$ equations:

$$A_i = \gamma\left(u = \frac{i}{p}\right), \ i \in [\![0, p]\!]. \tag{4.11}$$



(a) Control point initialization (b) Projected positions onto the curve (c) Final control point positions

Figure F.21: Naive approach to compute the positions of the control points (○) of a Bézier curve (▬) of order $p = 3$ in order to interpolate a geometric curve (▬). The control points of the Bézier curve are set uniformly on the linear edge (▬) (a). We compute the projected positions of the control points onto the geometrical curve (▬) (b). Finally, we compute the control points (c) to interpolate the positions projected onto the curve (●).

Method 2

The second method consists in an iterative process to compute the positions of the control points of the Bézier edge. Knowing we want to build a Bézier edge of degree p, we first compute successively a Bézier edge of degree $p = 2$, $p = 3$, etc. Until the Bézier edge of degree p.

In order to build the Bézier curve of degree $p{=}p_j$, we use the parametric space of the Bézier curve of degree $p_j - 1$ to initialize the control points $P_0, ..., P_{p_j}$. The control points are set uniformly in the parametric space of the curve (e.g., for a curve of degree p, the control points will be initialized on the Bézier curve of degree $p - 1$, at $u = \frac{i}{p}$, with $i \in [\![0, p]\!]$). Then, as for the precedent method, we compute a set of points $A_0, ..., A_{p_j}$, corresponding to the projected

positions of $P_0, ..., P_{p_j}$ on the geometric curve ( ). Then, we solve the system as before to compute the final positions of the control points to interpolate the positions $\{A_i\}_{i\in[\![0,p_j]\!]}$.

Due to the way the control points are computed with this method, building a Bézier Edge of degree $p=2$ should give the same result as with method 1.

Method 3

The third method presented here is almost exactly the same process as Method 2. The difference is when the Bézier edge of degree $p=2$ (see Fig. F.22) is built. We use the properties of the Bézier curves in order to compute the positions of the three control points P_0, P_1, P_2 to build the Bézier edge of degree $p=2$. As before, the control points P_0 and P_2 are taken as the two end points of the linear edge $p=1$. The third point, P_1, is set at the intersection of the two straight lines given by the two tangent to the geometric curve ( ) at the positions P_0 and P_2 (see Fig. F.22.b).

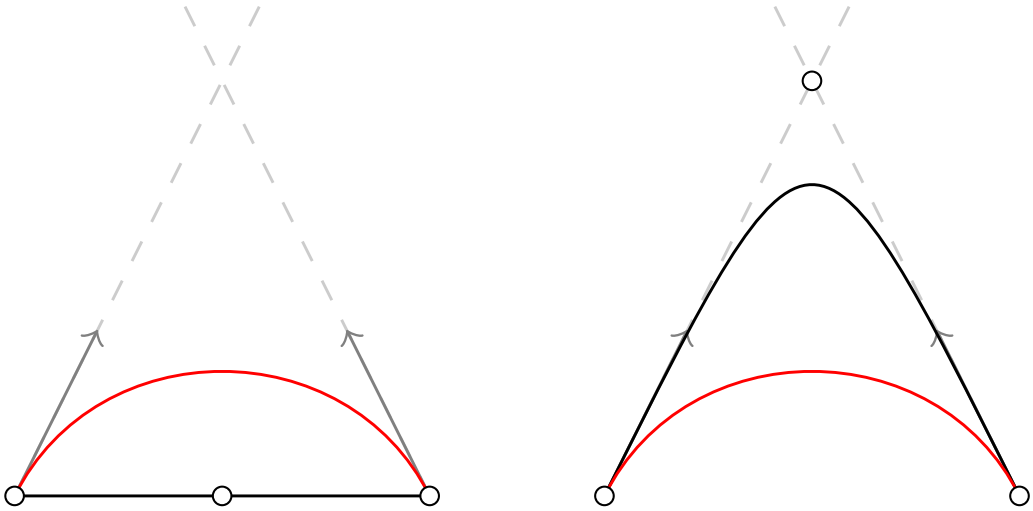


Figure F.22: Naive approach to compute the control points of an arbitrary Bézier curve of degree p to interpolate a geometrical curve.

From this point, building a Bézier edge of arbitrary degree p is the same iterative process as in method 2. However, due to the way the Bézier edge of degree $p=2$ is computed, the results will differ from the ones from method 2 for the other orders.

There is no reason in 3D that the tangent will intersect, but we can figure out the closest point to a line L_1 belonging to a second line L_2.

F.2 Methods Comparison on Curve Examples

Here, we take three different geometric curves built using analytical formulations as an example. The geometric curves represent a part of a 2D vehicle surface, while the Bézier curve is a block edge we want to align to this vehicle surface.

Curve 1

$$y = 0.4 \sin\left(x \frac{\pi}{2}\right), \tag{4.12}$$

with $x \in [0, 1]$.

If we take a look at the result of the first two methods on this geometric curve (see Fig. F.23), we can see that the results are quite similar, and the methods seem sufficient to capture the geometric curve.

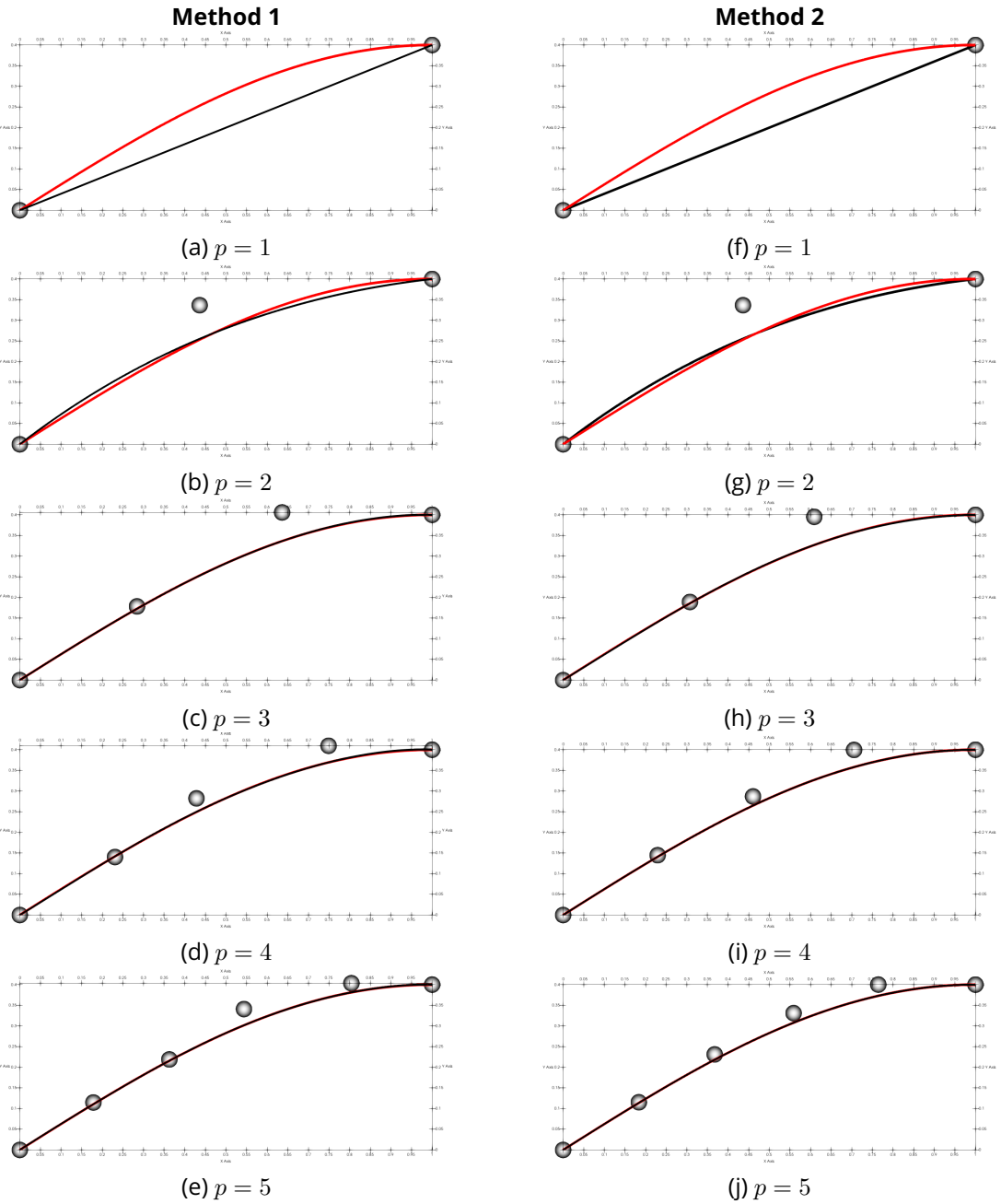


Figure F.23: Results of precedent methods to approximate the red geometric curve (▬) with the black curved Bézier block edge (▬) on first analytical curve.

Curve 2

$$\begin{cases} y = r(1-\cos(\theta)), x = r\sin(\theta), \theta = \frac{x\pi}{2} & \text{if } -0.1 \leq x \leq r \\ y = r & \text{if } r < x < 1 \end{cases} \tag{4.13}$$

The result obtained using the first approach illustrates that this naive method is not sufficient to capture the geometric curvature using a simple interpolated positions set. Even when increasing the degree, the curve is never captured. The second method proposed rectify the phenomena. However, still many control points are needed in order to capture the left part of the curve. The last method allows to capture the high curvature of the geometric curve with fewer control points. However, the Bézier curve oscillations (see Fig. F.24.q, and r) show the importance to keep control over the tangent at the extrimities (which is not the case here).

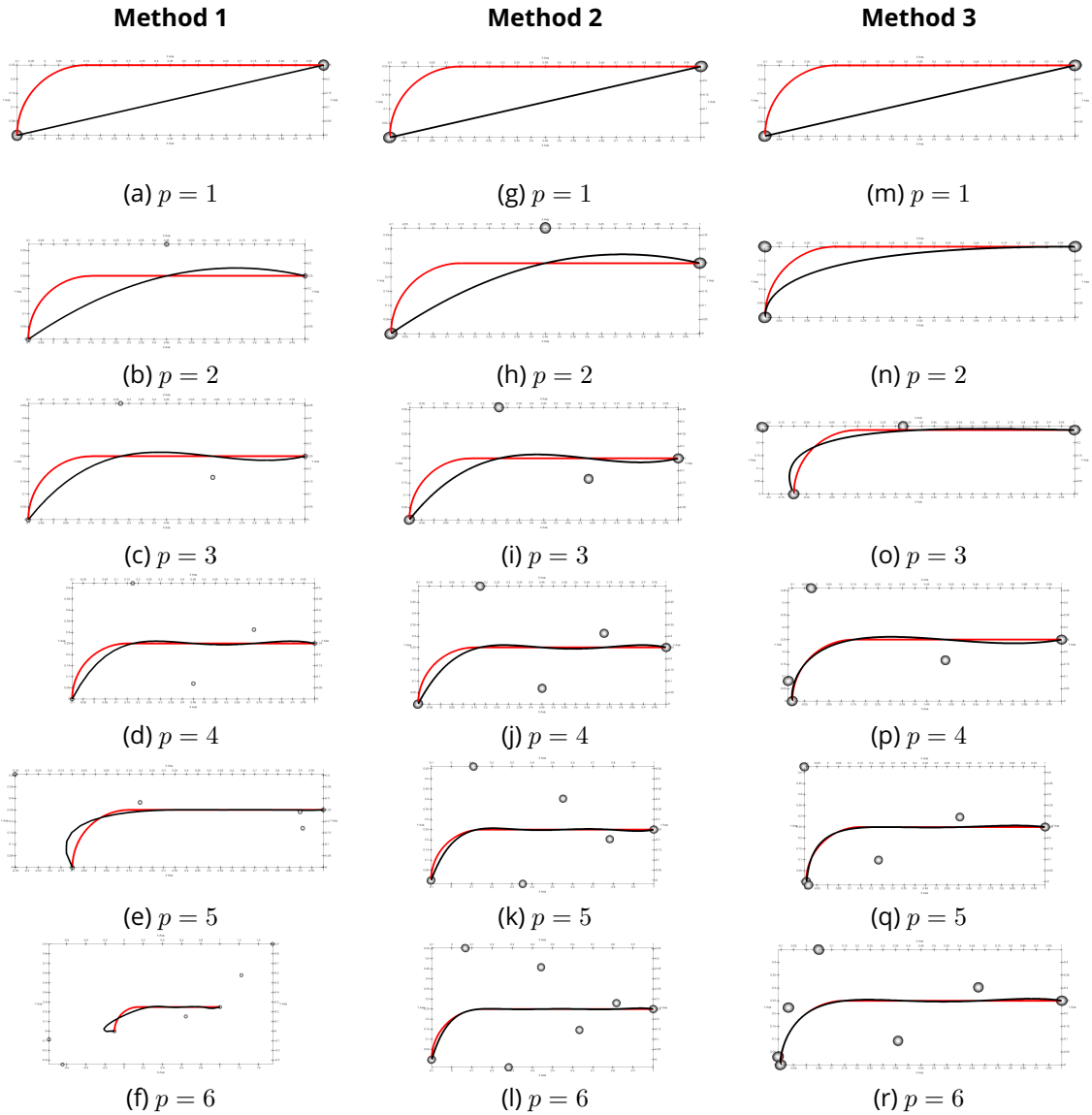


Figure F.24: Results of precedent methods to approximate the red geometric curve (▬) with the black curved Bézier block edge (▬) on second analytical curve.

Curve 3

The equation of the last curve we present here is given by

$$y = 0.4\sin(x\pi), \tag{4.14}$$

with $x \in [0, 1]$. The results on this curve (see Fig. F.25) illustrate well the problem of the naive projection method. If we take a look at the first method, for order $p=2$ (see Fig. F.25.b), we clearly see that the projected position of the middle control point on the geometric curve (▬) is not where expected at the top of the bump, but rather on the left part of the curve. This is because we project the control point on the closest point of the geometric curve.

This is not enough if we expect to correct this behavior using the second approach. Using the second method, for a degree $p=2$, the computation is the same as with the first method. The result of the Bézier curve (▬) of Figure F.25.b and F.25g are the same. In the case of the second method, we build the higher-order Bézier curve on the precedent ones. So the $p=3$ Bézier curve of Figure F.25.h is built starting from the one of Figure F.25.g. The two middle control points of the $p=3$ are placed on the $p=2$ one. They are placed uniformly in the parametrical space. Naturally, due to the shape of the $p=2$ Bézier curve (▬) of Figure F.25.g pulled on the left side, the density of the following control points will focus in this area. Even if we successfully capture the geometric curve shape (see Fig. F.25.j), all the control points are concentrated on the left part of the curve. Consequently, if we subdivide this Bézier curve, most of the final nodes will focus on the left part of the curve.

We correct this behavior using the last method presented in this part (see Fig. F.25.p). However, except in case $p = 2$, we do not control the tangent normal to the curve (see Fig. F.25.m,n,o). We probably want to keep control over that tangent; the question is how do we manage to choose the magnitude of the tangent vector in case $p > 2$ to approximate the curve most accurately.

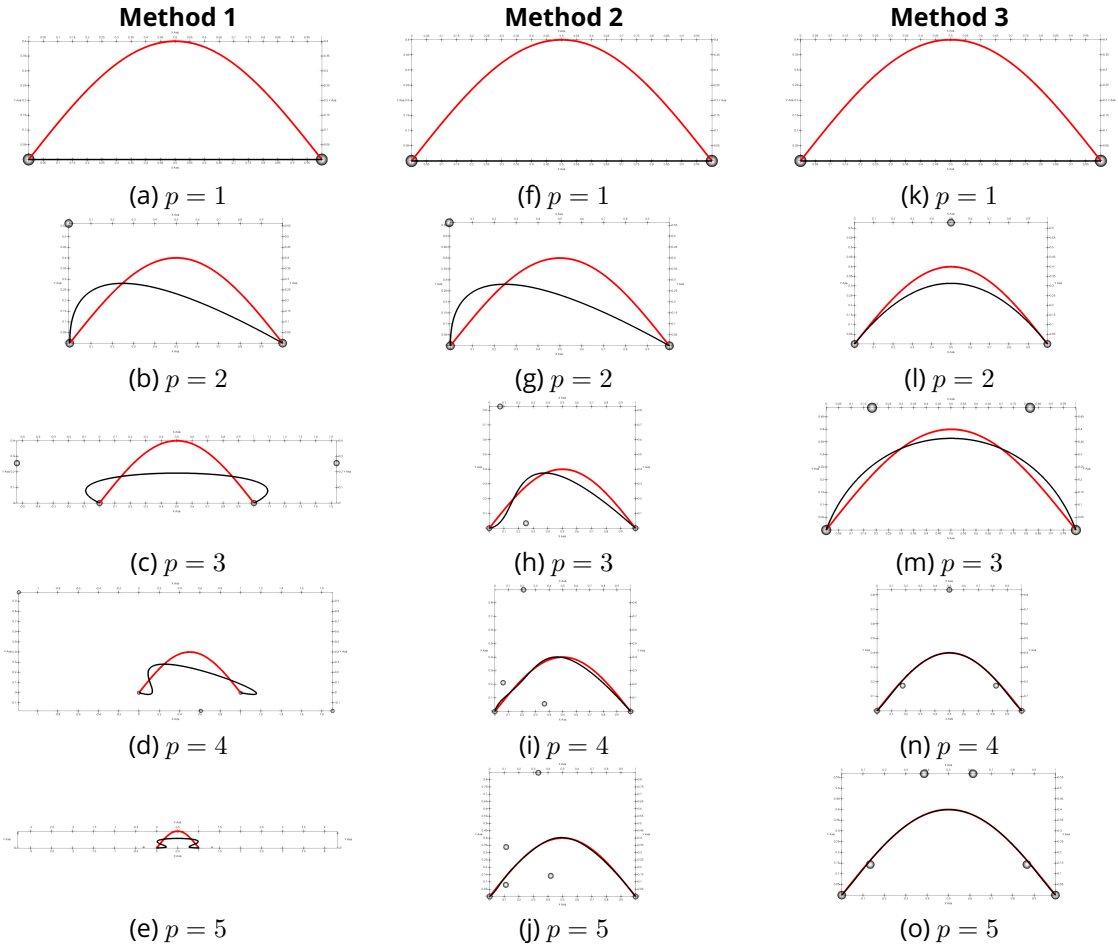
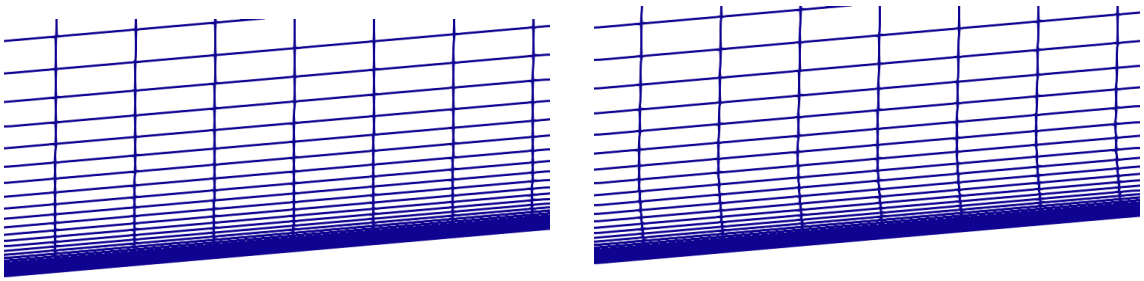


Figure F.25: Results of precedent methods to approximate red geometric curve (▬) with black curved Bézier block edge (▬) on third analytical curve.

G Line-Sweeping Smoothing



(a) Without boundary layer smoothing

(b) With boundary layer smoothing

Figure G.26: Comparison of the near wall mesh before and after smoothing.

Even if the block edges were placed in an orthogonal way, the computation remains local for each block corner. Consequently, there is no reason for the resulting mesh to be orthogonal to the wall (see Fig. G.26.a). At this stage, we perform a smoothing algorithm on the boundary layer blocks with an edge on $\partial\Omega_V$. This smoothing aims to enhance the orthogonality of the first cells in the block and is only available in this work in 2D. The smoothing algorithm [RBO22] is performed on each block. It is a modification of the Line-Sweeping method introduced by J. YAO [Yao13, YS16], specifically developed for structured meshes.

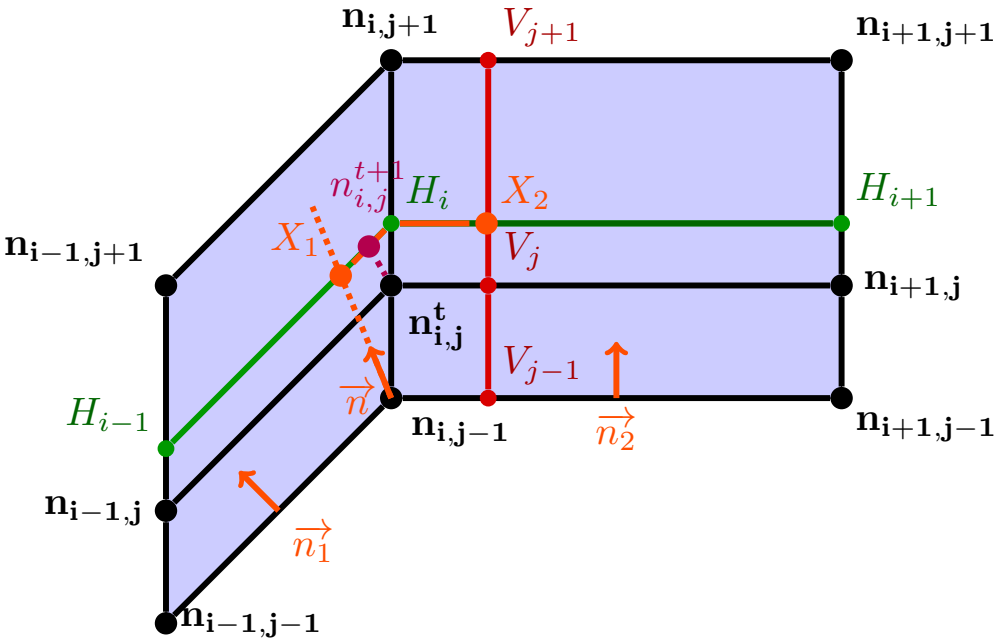


Figure G.27: Modified Line-Sweeping method on an internal node in a block.

The Line-Sweeping method is a geometric, local, iterative, and fully explicit method that aims to uniformize the cell sizes of a block. Let $\mathbf{B}$ be a block of size $N_x \times N_y$, and $n_{i,j}$ be a mesh node in $\mathbf{B}$ with $0 < i < N_x - 1$ and $0 < j < N_y - 1$[4]. To compute the new position at the iteration $t+1$ by the Line-Sweeping, we consider the stencil made of the six black nodes

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

on Figure G.27. 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 [RBO22]. Assuming the block edge on the wall is at the index $j = 0$. For each node $n_{i,j}$ as the one in Figure G.27, we compute the position X_2 with the Line-Sweeping method, and another orange point (●) X_1 is placed. Two vectors $\overrightarrow{n_1}$ and $\overrightarrow{n_2}$ (⟶) 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 $\overrightarrow{n} = \overrightarrow{n_1} + \overrightarrow{n_2}$ 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 G.26 illustrates how this smoothing stage improves the wall orthogonality.

H Shock Abacus

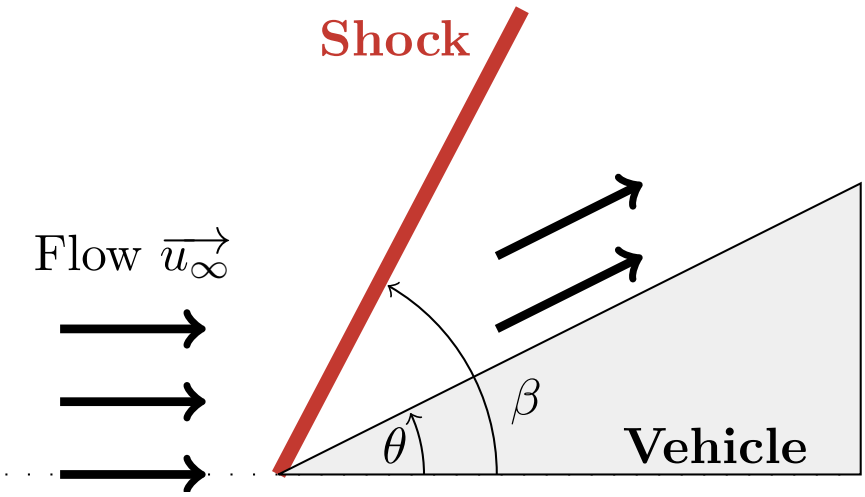


Figure H.28: Deflection angle θ and oblique shock ( ).

Figure H.28 represents an oblique shock ( ) that develops in front of the wedge object ( ). In this specific case, well-known tables [rs53] give the analytical solution of the shock angle β (see Fig. H.29), and the constant Mach number in areas upstream and downstream the shock wave (see Fig. H.30). The deflection angle θ is obtained through a combination of the flow Angle of Attack (equal to 0° in this case) and the wedge geometric feature.

In Figure H.29, we can find the shock position β value at the intersection between the abscissa deflection angle θ, and the isoline corresponding to the upsteam flow $\overrightarrow{u_\infty}$ Mach number. The β value is read on the ordinate axis. Figure H.30 gives the shock wave downstream Mach number in the specific case of a perfect gas. To know the expected downstream Mach number value, we look for the intersection between the abscissa deflection angle value θ and the isoline corresponding to our upstream Mach number.

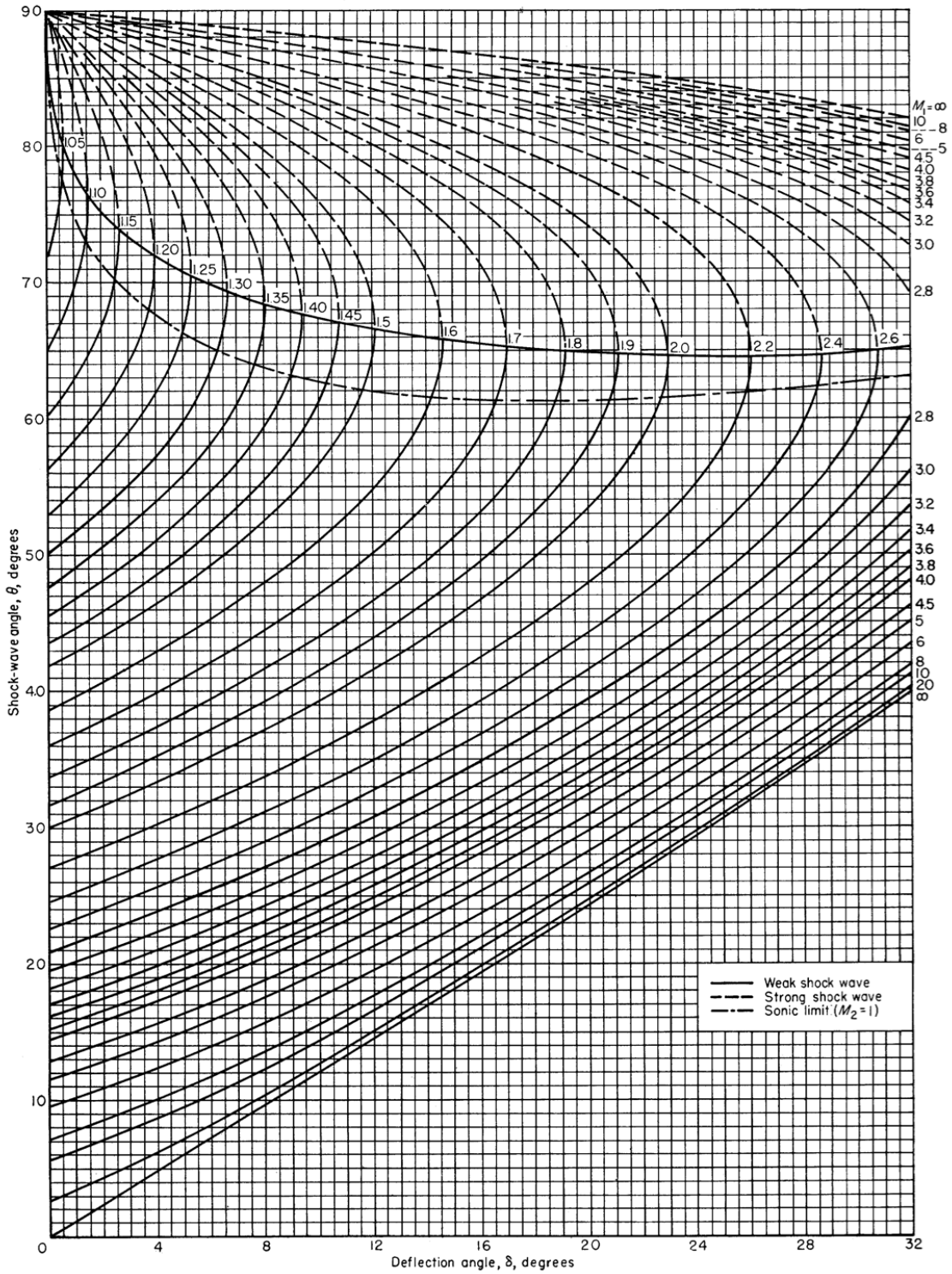


Figure H.29: Shock abacus [rs53].

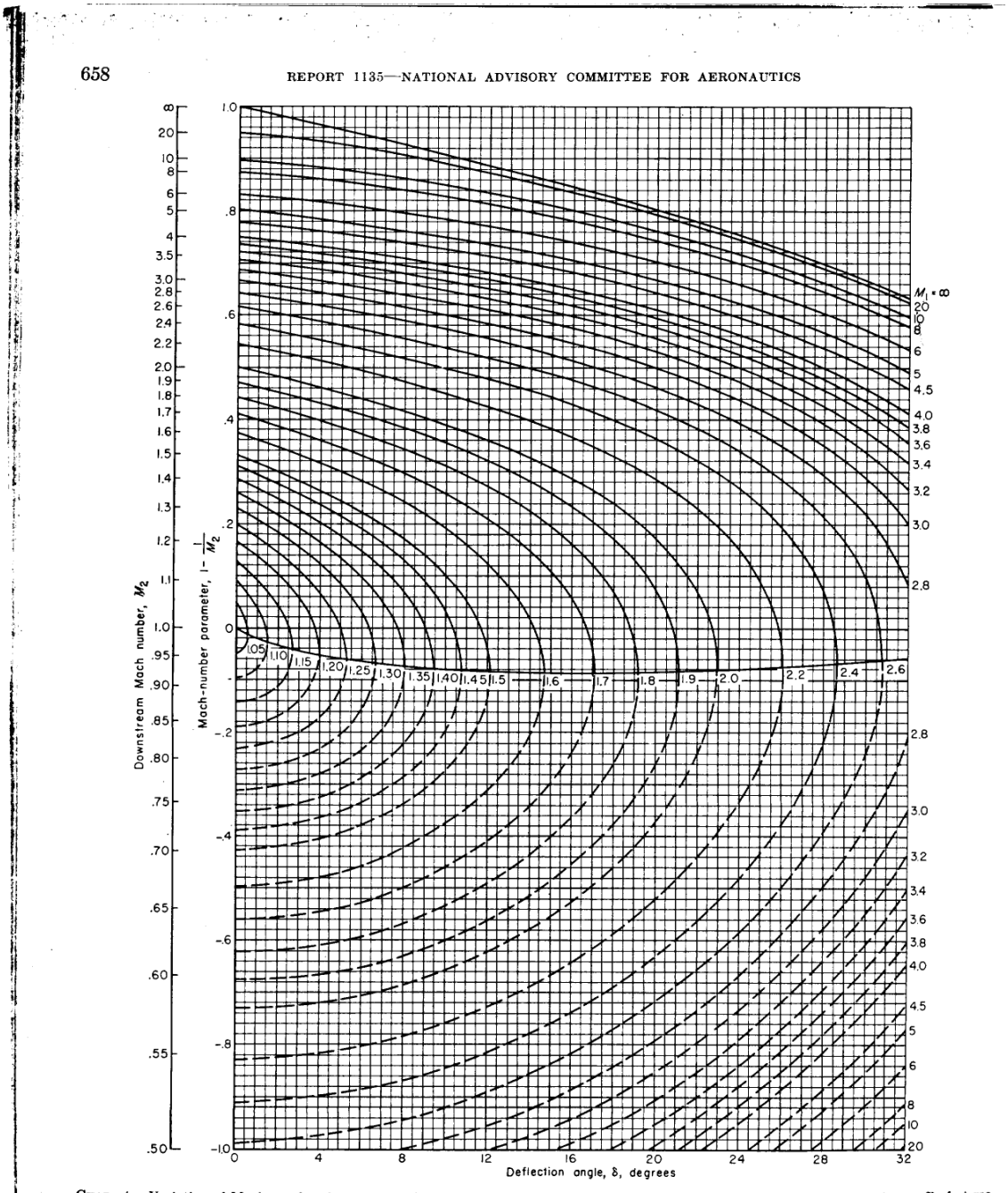


CHART 4.—Variation of Mach number downstream of a shock wave with flow-deflection angle for various upstream Mach numbers. Perfect gas, $\gamma=7/5$.

Figure H.30: Mach number abacus [rs53].

Bibliography

[AAB+21] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, et al. Mfem: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, 2021.

[AAB+24] Julian Andrej, Nabil Atallah, Jan-Phillip Bäcker, Jean-Sylvain Camier, Dylan Copeland, Veselin Dobrev, Yohann Dudouit, Tobias Duswald, Brendan Keith, Dohyun Kim, et al. High-performance finite elements with mfem. *The International Journal of High Performance Computing Applications*, page 10943420241261981, 2024.

[AADLC91] D Aymer, T Alziary, Luigi De Luca, and G Carlomagno. Experimental study of the flow around a double ellipsoid configuration. In *Hypersonic Flows for Reentry Problems: Volume II: Test Cases—Experiments and Computations Proceedings of a Workshop Held in Antibes, France, 22–25 January 1990*, pages 335–357, 1991.

[AL16] Frédéric Alauzet and Adrien Loseille. A decade of progress on anisotropic mesh adaptation for computational fluid dynamics. *Computer-Aided Design*, 72:13–39, 2016.

[Alt04] Stephen Alter. A structured grid quality measure for simulated hypersonic flows. In *42nd AIAA aerospace sciences meeting and exhibit*, page 612, 2004.

[And89] John David Anderson. *Hypersonic and high temperature gas dynamics*. Aiaa, 1989.

[ATS17] Zaib Ali, Paul G Tucker, and Shahrokh Shahpar. Optimal mesh topology generation for cfd. *Computer Methods in Applied Mechanics and Engineering*, 317:431–457, 2017.

[Bez86] Pierre Bezier. Courbes et surfaces, mathématiques et cao, 4. *Hermès, Paris*, 1986.

[BKMT23] Jorge-Luis Barrera, Tzanio Kolev, Ketan Mittal, and Vladimir Tomov. High-order mesh morphing for boundary and interface fitting to implicit geometries. *Computer-Aided Design*, 158:103499, 2023.

[Bla96] Ted Blacker. The cooper tool. *5th INTERNATIONAL MESHING ROUDTABLE, 1996*, 1996.

[BLP+13] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco

Tarini, and Denis Zorin. Quad-mesh generation and processing: A survey. *Computer Graphics Forum*, 32(6):51–76, 2013.

[BM93] Ted D Blacker and Ray J Meyers. Seams and wedges in plastering: a 3-d hexahedral mesh generation algorithm. *Engineering with computers*, 9:83–93, 1993.

[BM10] K. Beatty and N. Mukherjee. A transfinite meshing approach for body-in-white analyses. In *Proceedings of the 19th International Meshing Roundtable*. International Meshing Roundtable, 2010.

[BS91] Ted D Blacker and Michael B Stephenson. Paving: A new approach to automated quadrilateral mesh generation. *International journal for numerical methods in engineering*, 32(4):811–847, 1991.

[BSK21] Janis Born, Patrick Schmidt, and Leif Kobbelt. Layout embedding via combinatorial optimization. In *Computer graphics forum*, volume 40, pages 277–290. Wiley Online Library, 2021.

[cad] Cadence [Software]. https://www.pointwise.com/. Accessed: 2023-04-08.

[Cal22] Simon Calderan. *Génération interactive de maillages hexaédriques structurés par blocs*. PhD thesis, Université Paris-Saclay, 2022.

[Cam17] Marcel Campen. Partitioning surfaces into quadrilateral patches: A survey. In *Computer graphics forum*, volume 36, pages 567–588. Wiley Online Library, 2017.

[Can92] Scott Canann. Plastering-a new approach to automated, 3-d hexahedral mesh generation. In *33rd Structures, structural dynamics and materials conference*, page 2416, 1992.

[CDT16] John R Chawner, John Dannenhoffer, and Nigel J Taylor. Geometry, mesh generation, and the cfd 2030 vision. In *46th AIAA Fluid Dynamics Conference*, page 3485, 2016.

[Cha09] William M Chan. Overset grid technology development at nasa ames research center. *Computers & Fluids*, 38(3):496–503, 2009.

[Cou89] Jean Cousteix. *Turbulence et couche limite*. Editions Cépaduès, 1989.

[CWW17] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. The heat method for distance computation. *Communications of the ACM*, 60(11):90–99, 2017.

[DC59] Paul De Casteljau. Outillages méthodes calcul. *Andr e Citro en Automobiles SA, Paris*, 4:25, 1959.

[DGP12] Jean-Antoine Désidéri, Roland Glowinski, and Jacques Périaux. *Hypersonic Flows for Reentry Problems: Volume II: Test Cases—Experiments and Computations Proceedings of a Workshop Held in Antibes, France, 22–25 January 1990*. Springer Science & Business Media, 2012.

[DKK+19] Veselin Dobrev, Patrick Knupp, Tzanio Kolev, Ketan Mittal, and Vladimir Tomov. The target-matrix optimization paradigm for high-order meshes. *SIAM Journal on Scientific Computing*, 41(1):B50–B68, 2019.

[DWQ24] Siqiang Deng, Yibin Wang, and Ning Qin. Cross-field structured adaptive mesh using medial axis flow feature extraction. *AIAA Journal*, 62(1):247–262, 2024.

[EBSB19] Sébastien Esquieu, Elizabeth Benitez, Steven P Schneider, and Jean-Philippe Brazier. Flow and stability analysis of a hypersonic boundary layer over an axisymmetric cone cylinder flare configuration. In *AIAA Scitech 2019 Forum*, page 2115, 2019.

[EPC+16] Thomas D Economon, Francisco Palacios, Sean R Copeland, Trent W Lukaczyk, and Juan J Alonso. Su2: An open-source suite for multiphysics simulation and design. *AIAA Journal*, 54(3):828–846, 2016.

[esa] Europe space agency. https://www.esa.int/. Accessed: 2024-05-31.

[FA05] Pascal-Jean Frey and Frédéric Alauzet. Anisotropic mesh adaptation for cfd computations. *Computer methods in applied mechanics and engineering*, 194(48-49):5068–5082, 2005.

[FBL16] Xiao-Ming Fu, Chong-Yang Bai, and Yang Liu. Efficient volumetric polycube-map construction. *Computer Graphics Forum*, 35(7):97–106, 2016.

[FCK16] Nicolò Frapolli, Shyam S Chikatamarla, and Ilya V Karlin. Entropic lattice boltzmann model for gas dynamics: Theory, boundary conditions, and implementation. *Physical Review E*, 93(6):063302, 2016.

[Feu19] Rémi Feuillet. *Embedded and high-order meshes: two alternatives to linear body-fitted meshes*. PhD thesis, Université Paris-Saclay, 2019.

[FG99] Pascal Jean Frey and Paul-Louis George. *Maillages: applications aux éléments finis*. Hermès Science Publications Paris, France, 1999.

[FP16] Meire Fortunato and Per-Olof Persson. High-order unstructured curved mesh generation using the winslow equations. *Journal of Computational Physics*, 307:1–14, 2016.

[FXBH16] Xianzhong Fang, Weiwei Xu, Hujun Bao, and Jin Huang. All-hex meshing using closed-form induced polycube. *ACM Trans. Graph.*, 35(4):124:1–124:9, 2016.

[Gar02] Rao V Garimella. Mesh data structure selection for mesh generation and fea applications. *International journal for numerical methods in engineering*, 55(4):451–478, 2002.

[GB68] BJ Griffith and DE Boylan. Reynolds and mach number simulation of apollo and gemini re-entry and comparison with flight. *AGARD HYPERSONIC BOUNDARY LAYERS AND FLOW FIELDS MAY 1968 (SEE N69-10186 01-01)*, 1968.

[GB12] Paul-Louis George and Houman Borouchaki. *Sur les éléments finis quadrilatéraux de degré 1 et 2*. PhD thesis, INRIA, 2012.

[GJTP17] Xifeng Gao, Wenzel Jakob, Marco Tarini, and Daniele Panozzo. Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

[gmd] GMDS: a c++ library for writing meshing algorithms [Software]. https://github.com/LIHPC-Computational-Geometry/gmds.

[GMF11] J. Gould, D. Martineau, and R Fairey. Automated two-dimensional multiblock meshing using the medial object. In *Proceedings of the 20th International Meshing Roundtable*. Springer, 2011.

[gms] Gmsh[Software]. https://gmsh.info/.

[GO70] Nigel Gregory and CL O'reilly. Low-speed aerodynamic characteristics of naca 0012 aerofoil section, including the effects of upper-surface roughness simulating hoar frost. *ARC R & M*, 1970.

[GPRPS16] Abel Gargallo-Peiró, Xevi Roca, Jaume Peraire, and Josep Sarrate. A distortion measure to validate and generate curved high-order meshes on cad surfaces with independence of parameterization. *International Journal for Numerical Methods in Engineering*, 106(13):1100–1130, 2016.

[GR09] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International journal for numerical methods in engineering*, 79(11):1309–1331, 2009.

[GR24] Christophe Geuzaine and Jean-François Remacle. Geodesic meshing of closed surfaces. In *SIAM IMR24-SIAM International Meshing Roundtable Workshop 2024*, 2024.

[GSP19] Xifeng Gao, Hanxiao Shen, and Daniele Panozzo. Feature preserving octree-based hexahedral meshing. In *Computer graphics forum*, volume 38, pages 135–149. Wiley Online Library, 2019.

[GSZ11] James Gregson, Alla Sheffer, and Eugene Zhang. All-hex mesh generation via volumetric polycube deformation. *Computer Graphics Forum*, 30(5):1407–1416, 2011.

[HJS+14] Jin Huang, Tengfei Jiang, Zeyun Shi, Yiying Tong, Hujun Bao, and Mathieu Desbrun. ℓ1-based construction of polycube maps from complex shapes. *ACM Trans. Graph.*, 33(3):25:1–25:11, 2014.

[HQZ13] Kangkang Hu, Jin Qian, and Yongjie Zhang. Adaptive all-hexahedral mesh generation based on a hybrid octree and bubble packing. In *Proceedings of International Meshing Roundtable*, 2013.

[HZ16] Kangkang Hu and Yongjie Jessica Zhang. Centroidal voronoi tessellation based polycube construction for adaptive all-hexahedral mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 305:405 – 421, 2016.

[ICE] ICEM CFD [Software]. https://dl.cfdexperts.net/cfd_resources/Ansys_Documentation/ICEM%20CFD/Ansys_ICEM_CFD_Users_Manual.pdf. Accessed: 2024-06-17.

[ISS09] Yasushi Ito, Alan M Shih, and Bharat K Soni. Octree-based reasonable-quality

hexahedral mesh generation using a new set of refinement templates. *International Journal for Numerical Methods in Engineering*, 77(13):1809–1833, 2009.

[JAK15] Thomas J Juliano, David Adamczak, and Roger L Kimmel. Hifire-5 flight test results. *Journal of Spacecraft and Rockets*, 52(3):650–663, 2015.

[KET+06] Patrick Michael Knupp, CD Ernst, David C Thompson, CJ Stimpson, and Philippe Pierre Pebay. The verdict geometric quality library. Technical report, Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA, 2006.

[KKSW22] Steve L Karman, Kristen Karman-Shoemake, and Carolyn D Woeber. Mixed order mesh curving. In *Mesh Generation and Adaptation: Cutting-Edge Techniques*, pages 1–21. Springer, 2022.

[KLF16] Nicolas Kowalski, Franck Ledoux, and Pascal Frey. Smoothness driven frame field generation for hexahedral meshing. *Computer-Aided Design*, 72:65–77, 2016.

[Knu00] Patrick M Knupp. Achieving finite element mesh quality via optimization of the jacobian matrix norm and associated quantities. part ii—a framework for volume mesh optimization and the condition number of the jacobian matrix. *International Journal for numerical methods in engineering*, 48(8):1165–1185, 2000.

[Knu01] Patrick M Knupp. Algebraic mesh quality metrics. *SIAM journal on scientific computing*, 23(1):193–218, 2001.

[Knu07] Patrick Knupp. Remarks on mesh quality. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2007.

[Knu20] Patrick Knupp. Metric type in the target-matrix mesh optimization paradigm. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2020.

[Knu22] Patrick Knupp. Geometric parameters in the target matrix mesh optimization paradigm. *Partial Differential Equations in Applied Mathematics*, 5:100390, 2022.

[Kow13] Nicolas Kowalski. *Domain partitioning using frame fields: applications to quadrilateral and hexahedral meshing*. PhD thesis, Paris 6, 2013.

[Küc65] Dietrich Küchemann. Hypersonic aircraft and their aerodynamic problems. *Progress in Aerospace Sciences*, 6:271–353, 1965.

[LAA10] Adrien Loseille, Dervieux Alain, and Frédéric Alauzet. Fully anisotropic goal-oriented mesh adaptation for 3d steady euler equations. *Journal of computational physics*, 229(8):2866–2897, 2010.

[LBK16] Max Lyon, David Bommes, and Leif Kobbelt. Hexex: robust hexahedral mesh extraction. *ACM Trans. Graph.*, 35(4):123, 2016.

[LG97] Shang-Sheng Liu and Rajit Gadh. Automatic hexahedral mesh generation by recursive convex and swept volume decomposition. In *6th international meshing roundtable, Sandia National Laboratories*, pages 217–231. Citeseer, 1997.

[LGT01] Yong Lu, Rajit Gadh, and Timothy J Tautges. Feature based hex meshing methodology: feature recognition and volume decomposition. *Computer-Aided Design*, 33(3):221–232, 2001.

[LPC21] Marco Livesu, Luca Pitzalis, and Gianmarco Cherchi. Optimal dual schemes for adaptive grid based hexmeshing. *ACM Transactions on Graphics (TOG)*, 41(2):1–14, 2021.

[LR01] Hans Wolfgang Liepmann and Anatol Roshko. *Elements of gasdynamics*. Courier Corporation, 2001.

[LVS+13] Marco Livesu, Nicholas Vining, Alla Sheffer, James Gregson, and Riccardo Scateni. Polycut: Monotone graph-cuts for polycube base-complex construction. *ACM Trans. Graph.*, 32(6):171:1–171:12, 2013.

[LWB08] Franck Ledoux, Jean-Claude Weill, and Yves Bertrand. Gmds: A generic mesh data structure. *17th International Meshing Roundtable*, 2008.

[LZC+18] Heng Liu, Paul Zhang, Edward Chien, Justin Solomon, and David Bommes. Singularity-constrained octahedral fields for hexahedral meshing. *ACM Trans. Graph.*, 37(4):93, 2018.

[Mar09] Loïc Maréchal. Advances in octree-based all-hexahedral mesh generation: handling sharp features. In *Proceedings of the 18th international meshing roundtable*, pages 65–84. Springer, 2009.

[MBST96] Lai Mingwu, Steven E Benzley, Greg Sjaardema, and Tim Tautges. A multiple source and target sweeping method for generating all-hexahedral finite element meshes. In *Proceedings, 5th International Meshing Roundtable*, volume 96, pages 217–225. Citeseer, 1996.

[MCM+20] Julian Marcon, Giacomo Castiglioni, David Moxey, Spencer J Sherwin, and Joaquim Peiró. rp-adaptation for compressible flows. *International Journal for Numerical Methods in Engineering*, 121(23):5405–5425, 2020.

[MDK+24] Ketan Mittal, Veselin A Dobrev, Patrick Knupp, Tzanio Kolev, Franck Ledoux, Claire Roche, and Vladimir Z Tomov. Mixed-order meshes through rp-adaptivity for surface fitting to implicit geometries. In *Proceedings of the 2024 International Meshing Roundtable (IMR)*, pages 118–131. SIAM, 2024.

[Men92] Florian R Menter. Improved two-equation k-omega turbulence models for aerodynamic flows. Technical report, 1992.

[Men94] Florian R Menter. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA journal*, 32(8):1598–1605, 1994.

[mfe] MFEM: Modular finite element methods [Software]. https://mfem.org.

[MGSP15] D Moxey, MD Green, SJ Sherwin, and J Peiró. An isoparametric approach to high-order curvilinear boundary-layer meshing. *Computer Methods in Applied Mechanics and Engineering*, 283:636–650, 2015.

[Mit96] Scott A Mitchell. A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume. In *STACS 96: 13th Annual Symposium on Theoretical Aspects of Computer Science Grenoble, France, February 22–24, 1996 Proceedings 13*, pages 465–476. Springer, 1996.

[Mit21] S Mitchell. Incremental Interval Assignment by Integer Linear Algebra. *proc. of the International Meshing Roundtable*, October 2021.

[MJK64] John F McCarthy Jr and Toshi Kubota. A study of wakes behind a circular cylinder at m equal 5.7. *AIAA journal*, 2(4):629–636, 1964.

[MLP19] Claudio Mancinelli, Marco Livesu, and Enrico Puppo. A comparison of methods for gradient field estimation on simplicial meshes. *Computers & Graphics*, 80:37–50, 2019.

[NRP11a] M. Nieser, U. Reitebuch, and K. Polthier. Cubecover–parameterization of 3d volumes. *Computer Graphics Forum*, 30(5):1397–1406, 2011.

[NRP11b] Matthias Nieser, Ulrich Reitebuch, and Konrad Polthier. Cubecover–parameterization of 3d volumes. In *Computer graphics forum*, volume 30, pages 1397–1406. Wiley Online Library, 2011.

[OS00] Steven J Owen and Sunil Saigal. H-morph: an indirect approach to advancing front hex meshing. *International Journal for Numerical Methods in Engineering*, 49(1-2):289–312, 2000.

[OSCS99] Steven J Owen, Matthew L Staten, Scott A Canann, and Sunil Saigal. Q-morph: an indirect approach to advancing front quad meshing. *International journal for numerical methods in engineering*, 44(9):1317–1340, 1999.

[Owe98] Steven J Owen. A survey of unstructured mesh generation technology. *IMR*, 239(267):15, 1998.

[PA97] Mark A Price and Cecil G Armstrong. Hexahedral mesh generation by medial surface subdivision: Part ii. solids with flat and concave edges. *International Journal for Numerical Methods in Engineering*, 40(1):111–136, 1997.

[PAS95] Mark A Price, Cecil G Armstrong, and MA Sabin. Hexahedral mesh generation by medial surface subdivision: Part i. solids with convex edges. *International Journal for Numerical Methods in Engineering*, 38(19):3335–3359, 1995.

[PBS20] David Palmer, David Bommes, and Justin Solomon. Algebraic representations for volumetric frame fields. *ACM Transactions on Graphics (TOG)*, 39(2):1–17, 2020.

[PCS+22] Nico Pietroni, Marcel Campen, Alla Sheffer, Gianmarco Cherchi, David Bommes, Xifeng Gao, Riccardo Scateni, Franck Ledoux, Jean-François Remacle, and Marco Livesu. Hex-mesh generation and processing: A survey. *ACM Trans. Graph.*, jul 2022. Just Accepted.

[PLC+21] Luca Pitzalis, Marco Livesu, Gianmarco Cherchi, Enrico Gobbetti, and Riccardo Scateni. Generalized adaptive refinement for grid-based hexahedral meshing. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, 2021.

[Pro22] François Protais. *Maillage à dominante Polycube*. PhD thesis, Université de Lorraine, 2022.

[PSG16] Roman Poya, Ruben Sevilla, and Antonio J Gil. A unified approach for a posteriori high-order curved mesh generation using solid mechanics. *Computational Mechanics*, 58:457–490, 2016.

[QLYT21] Han Qi, Xinliang Li, Changping Yu, and Fulin Tong. Direct numerical simulation of hypersonic boundary layer transition over a lifting-body model hytrv. *Advances in Aerodynamics*, 3:1–21, 2021.

[Qua16] William Roshan Quadros. Laytracks3d: A new approach for meshing general solids using medial axis transform. *Computer-Aided Design*, 72:102–117, 2016.

[RBHL23] Claire Roche, Jérôme Breil, Thierry Hocquellet, and Franck Ledoux. Block-structured quad meshing for supersonic flow simulations. In *SIAM International Meshing Roundtable Workshop 2023 (SIAM IMR23)*, Amsterdam, The Netherlands, March 2023.

[RBO22] Claire Roche, Jerome Breil, and Marina Olazabal. Mesh regularization of ablating hypersonic vehicles. In *8th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2022)*, Oslo, Norway, June 2022.

[RG11] Eloi Ruiz-Gironés. *Automatic hexahedral meshing algorithms: from structured to unstructured meshes*. PhD thesis, Universitat Politècnica de Catalunya (UPC), 2011.

[RGR22] Eloi Ruiz-Gironés and Xevi Roca. Automatic penalty and degree continuation for parallel pre-conditioned mesh curving on virtual geometry. *Computer-Aided Design*, 146:103208, 2022.

[RGRS12] Eloi Ruiz-Gironés, Xevi Roca, and Josep Sarrate. The receding front method applied to hexahedral mesh generation of exterior domains. *Engineering with computers*, 28(4):391–408, 2012.

[Rob87] John Robinson. Cre method of element testing and the jacobian shape parameters. *Engineering Computations*, 4(2):113–118, 1987.

[rs53] Ames research staff. Raport 1135: Equations, tables, and charts for compressible flow. Technical report, Ames Aeronautical Laboratory, 1953.

[RSL16] Nicolas Ray, Dmitry Sokolov, and Bruno Lévy. Practical 3d frame field generation. *ACM Transactions on Graphics (TOG)*, 35(6):1–9, 2016.

[Rum21] Christopher Rumsey. 2DN00: 2D NACA 0012 Airfoil Validation Case. https://turbmodels.larc.nasa.gov/naca0012_val.html, 2021. [Online; accessed 29-January-2024].

[SBM21] Benjamin M Simmons, Pieter G Buning, and Patrick C Murphy. Full-envelope aero-propulsive model identification for lift+ cruise aircraft using computational experiments. In *AIAA Aviation 2021 Forum*, page 3170, 2021.

[Sca07] Leonardo C Scalabrin. *Numerical simulation of weakly ionized hypersonic flow over reentry capsules*. PhD thesis, Citeseer, 2007.

[SCB92] Micheal B Stephenson, Scott A Canann, and Ted D Blacker. Plastering: a new approach to automated, 3d hexahedral mesh generation. Technical report, Sandia National Labs., Albuquerque, NM (United States), 1992.

[Sch96] Robert Schneiders. A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with computers*, 12:168–177, 1996.

[Sch97] Robert Schneiders. An algorithm for the generation of hexahedral element meshes based on an octree technique. In *6th International Meshing Roundtable*, pages 195–196. Citeseer, 1997.

[SCO99] Matthew L Staten, Scott A Canann, and Steven J Owen. Bmsweep: locating interior nodes during sweeping. *Engineering with Computers*, 15:212–218, 1999.

[SD13] A Sobachkin and G Dumnov. Numerical basis of cad-embedded cfd. In *NAFEMS World Congress*, volume 19, 2013.

[Set99] James A Sethian. Fast marching methods. *SIAM review*, 41(2):199–235, 1999.

[She09] Jason F Shepherd. Conforming hexahedral mesh generation via geometric capture methods. In *Proceedings of the 18th international meshing roundtable*, pages 85–102. Springer, 2009.

[SKH+21] Ney R Secco, Gaetan KW Kenway, Ping He, Charles Mader, and Joaquim RRA Martins. Efficient mesh generation and deformation for aerodynamic shape optimization. *AIAA Journal*, 59(4):1151–1168, 2021.

[SKO+10] Matthew L Staten, Robert A Kerr, Steven J Owen, Ted D Blacker, Marco Stupazzini, and Kenji Shimada. Unconstrained plastering—hexahedral mesh generation via advancing-front geometry decomposition. *International journal for numerical methods in engineering*, 81(2):135–171, 2010.

[SKOB06] Matthew L Staten, Robert A Kerr, Steven J Owen, and Ted D Blacker. Unconstrained paving and plastering: Progress update. In *proceedings of the 15th International Meshing Roundtable*, pages 469–486. Springer, 2006.

[SLK04] Yi Su, KH Lee, and A Senthil Kumar. Automatic hexahedral mesh generation for multi-domain composite models using a hybrid projective grid-based method. *Computer-Aided Design*, 36(3):203–215, 2004.

[Smi99] Robert E Smith. Transfinite interpolation (tfi) generation systems. *Handbook of grid generation*, pages 3–1, 1999.

[SOB05] Matthew L Staten, Steven J Owen, and Ted D Blacker. Unconstrained paving & plastering: A new idea for all hexahedral mesh generation. In *proceedings of the 14th International Meshing Roundtable*, pages 399–416. Springer, 2005.

[SS96] Bih-Yaw Shih and Hiroshi Sakurai. Automated hexahedral mesh generation by swept volume decomposition and recomposition. In *5th International meshing roundtable*, volume 280. Citeseer, 1996.

[SSS08] Matthew L Staten, Jason F Shepherd, and Kenji Shimada. Mesh matching–creating conforming interfaces between hexahedral meshes. In *Proceedings of the 17th International Meshing Roundtable*, pages 467–484. Springer, 2008.

[SU2] SU2 [Software]. https://su2code.github.io/.

[TF89] LM Taylor and DP Flanagan. Pronto 3d: A three-dimensional transient solid dynamics program. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 1989.

[THCM04] Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. Polycube-maps. *ACM Trans. Graph.*, 23(3), 2004.

[Tho12] Hugh Thornburg. Overview of the pettt workshop on mesh quality/resolution, practice, current research, and future directions. In *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 606, 2012.

[TLR16] Thomas Toulorge, Jonathan Lambrechts, and Jean-François Remacle. Optimizing the geometrical accuracy of curvilinear meshes. *Journal of Computational Physics*, 310:361–380, 2016.

[TSW98] Joe F Thompson, Bharat K Soni, and Nigel P Weatherill. *Handbook of grid generation*. CRC press, 1998.

[WL16] Jean-Christophe Weill and Franck Ledoux. Towards an automatic and reliable hexahedral meshing, July, 2016. Tetrahedron V, Liège.

[XSHM13] Zhong Q Xie, Ruben Sevilla, Oubay Hassan, and Kenneth Morgan. The generation of arbitrary order curved meshes for 3d finite element analysis. *Computational Mechanics*, 51:361–374, 2013.

[Yao13] Jin Yao. A mesh relaxation study and other topics. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2013.

[YS16] Jin Yao and Douglas Stillman. An equal-space algorithm for block-mesh improvement. *Procedia engineering*, 163:199–211, 2016.

[ZB05] Yongjie Zhang and Chandrajit Bajaj. Adaptive and quality quadrilateral/hexahedral meshing from volumetric imaging data. In *Proceedings, 13th international meshing roundtable. Sandia National Laboratories*, pages 365–376, 2005.