Embedded and high-order meshes : two alternatives to linear body-fitted meshes

Rémi Feuillet

NNT : 2019SACLY010

THÈSE DE DOCTORAT

de

L'Université Paris-Saclay

École doctorale de mathématiques Hadamard (EDMH, ED 574)

Établissement d'inscription : École nationale supérieure de techniques avancées

Laboratoires d'accueil : Unité de mathématiques appliquées, ENSTA-CNRS-INRIA
Équipe-projet Gamma, INRIA

Spécialité de doctorat : Mathématiques appliquées

Rémi FEUILLET

Embedded and high-order meshes : two alternatives to linear body-fitted meshes

Date de soutenance : 10 Décembre 2019

Après avis des rapporteurs : Jean-François REMACLE (Université Catholique de Louvain)
Xevi ROCA (Barcelona Supercomputing Center)

Jury de soutenance :

Dr.	Frédéric ALAUZET	(INRIA Saclay-Île-de-France)	Co-directeur de thèse
Prof.	Patrick CIARLET	(ENSTA Paris)	Directeur de thèse
Dr.	Jean-François LAGÜE	(DISTENE)	Examinateur
Dr.	Franck LEDOUX	(CEA-DAM-Île-de-France)	Président du jury
Dr.	Adrien LOSEILLE	(INRIA Saclay-Île-de-France)	Co-directeur de thèse
Prof.	Jean-François REMACLE	(Université Catholique de Louvain)	Rapporteur
Dr.	Xevi ROCA	(Barcelona Supercomputing Center)	Rapporteur

Remerciements

Ha les remerciements ! Sans doute la page la plus lue [1] de toutes les thèses ... Il va donc s'agir de les faire correctement ! La thèse et plus particulièrement sa rédaction sont des phases assez uniques dans une vie. D'une manière assez curieuse, je me suis découvert une passion soudaine pour l'entretien de plantes vertes en voie de décès ; le changement de disposition de son bureau au travail ou de son salon à la maison ; ou encore la découverte et un intérêt insoupçonné pour de nouveaux groupes de musique « tapis dans l'ombre depuis des millénaires ». Mais une thèse, ce sont aussi des personnes que l'on rencontre, avec qui l'on travaille ou l'on vit et ce sont ces dernières que je vais tâcher de remercier comme il se doit !

Tout d'abord, je souhaiterais remercier les personnes qui ont encadré ma thèse. Je voudrais ainsi exprimer ma gratitude à Frédéric Alauzet et Adrien Loseille, mes encadrants à l'INRIA pour m'avoir proposé cette thèse. Je suis très reconnaissant envers Frédéric pour son encadrement, son expérience, sa disponibilité, sa vision très complète sur les sujets de recherche que l'on aborde mais aussi pour tous les moments extra-professionnels passés ensemble notamment en conférence et au Mississippi. Je ne saurais oublier Adrien avec qui l'on peut travailler, déconner ou travailler en déconnant. Je le remercie notamment pour m'avoir proposé des sujets de recherche assez originaux comme la visualisation. Je repense encore à la « demi-journée » de débogage d'un code *lawyered* qui dura finalement deux semaines et fut accompagnée de beaucoup de rires et de pétages de câbles ... Un grand merci aussi à Patrick Ciarlet notamment pour ses conseils qui m'ont amené à faire cette thèse, son apport initial indispensable au dossier de thèse et enfin pour sa disponibilité et sa bienveillance tout au long de cette thèse. Enfin, il me serait impossible de ne pas citer l'éminence grise qui se trouve derrière le projet Gamma de l'INRIA, projet qui m'a accueilli pour la durée de cette thèse. Je parle bien entendu de Paul Louis George, le « père du maillage en France ». [2] Je lui suis notamment reconnaissant d'avoir accepté de partager une partie de sa vaste connaissance du maillage (et notamment ses applications à la méthode des éléments finis, qui est de plus en plus utilisée) avec la personne ignare que je suis.

Ensuite, je suis très honoré que Xevi Roca et Jean-François Remacle aient accepté d'être rapporteurs de cette thèse. Thank you to both of you. It has always been and it will always be a pleasure to discuss with you. Je remercie également Jean-François Lagüe et Franck Ledoux d'avoir été examinateurs pour la soutenance de cette thèse.

J'ai aussi une pensée particulière pour les membres du projet Gamma de l'INRIA qui m'ont supporté pendant cette thèse. Je commence par les jeunes du projet, notamment Lucille, experte « qua qua », toujours de bonne humeur et bon public ; Matthieu, « Jean-Michel Capping » (déso), avec qui on peut toujours bien déconner ; Lucien, « Christian Ronald » (déso aussi), à qui je souhaite bien de la chance dans l'aventure du maillage d'ordre élevé et enfin Julien, grand passioné de l'aéronautique, à qui je suis reconnaissant d'avoir donné un aperçu des méthodes d'ordre élevé au béotien que je suis. Non dimentico gli italiani Francesco e Daniele. Je n'oublie pas non plus Loïc M., le « Q » façon Gamma, avec qui l'on peut discuter *hardware*, pyramides ou histoire ; ou Patrick avec qui les repas de midi ne sont jamais ennuyeux. Thank you also to Dave, it was really nice to you to

1. Certains diront que c'est même la seule.

2. Je ne fais que citer la quatrième de couverture des ouvrages d'une série à succès intitulée « Maillage, modélisation géométrique et simulation numérique ». Pour les ouvrages parus, voir [Borouchaki and George 2017a, George et al. 2018] ou [Borouchaki and George 2017b, George et al. 2019] pour les versions en langue de Shakespeare.

welcome us in Mississippi and I really enjoyed working with you. Je me dois pas de citer aussi les anciens du projet : Olivier (Ph.D.), le Clint Eastwood du NERF Gun ; Eléonore, toujours là pour papoter et à l'écoute quand on veut décompresser ; Loïc F. dont les coups de gueule contre son code furent mémorables ; Alexis avec qui René Coty a été mis à la mode dans le projet ou encore Nicolas R., cet espion venant du froid qui mettait un short dès qu'il faisait 15 degrés. Mention spéciale aussi aux encore plus anciens Nicolas B., Victorien et Géraldine. Pour leur travail de l'ombre, je remercie les diverses assistantes : Emmanuelle, Jessica, Agustina pour l'INRIA et Corinne pour l'ENSTA. Aussi, malgré mon peu de présence là bas, j'ai toujours apprécié l'accueil chaleureux que me réservait l'UMA de l'ENSTA. Mention particulière à Stéphanie, Axel, Sourour, Antoine, Sandrine et tous mes anciens professeurs.

Parce qu'une thèse c'est aussi ce qu'on fait à côté, je remercie mes amis. En premier lieu, je pense à mes colocs Laurent et PV ainsi que leurs moitiés respectives Valentine et Néné. Ces moments de partage culinaire (kebab inclus) ainsi que ces discussions interminables le soir à la maison m'ont permis de ne pas penser à ma thèse plus que nécessaire. Je n'oublie cependant pas que cette aventure coloc avait commencé par la fameuse pablocation avec Julien C. et Youssef. Je voudrais aussi mentionner mes amis de l'ENSTA qui comme moi ont eu l'idée saugrenue de faire une thèse. Hadrien, avec qui on peut discuter de n'importe quoi comme le foot, l'histoire ou la politique ; ~~Emilien F., qui ne manque pas de rappeler son existence dès qu'il est là, mais qui n'assume pas de faire des missiles dans sa thèse ;~~ Rémi, ce passionné du détail et des pièces de théâtre ; Benoît B. qui fait de l'automatique trop théorique pour moi ; Dara qui calcule des lambdas en utilisant un coq ; Léopold, ce jeune mathématicien en herbe qui code en Julia tout en portant un sarouel ou encore Julien G. qui ne se lassera jamais de boire du Tropico. Dédicace spéciale à ceux qui furent présents au fameux Week-End Grotte. Même s'il n'est pas docteur, je dresse une mention spéciale au sieur Florian E. avec qui je partage une passion certaine pour une carte de France de 1812. Je n'oublie pas Taha, mon vendeur de tapis favori, ami fidèle d'entre les fidèles. Parce que les Week-End ont été plus courts grâce à eux, je remercie la team « France Gall » composée de Marc, Julien J., Florian M. et Maxime. Enfin, je remercie tous les autres amis que j'ai rencontrés grâce à l'ENSTA (et la salle T pour beaucoup) et que je revois toujours avec plaisir : Adrien, Manuel, Benoît S., Théo, Emilien B., Nicolas F., Thomas, Guillaume, Florian R., Corentin, Pierre, Quentin, Basile, Servane, Delphine, Aurore, Elise, Katie, Juliette, Aurélie, Eugénie, Pauline et bien d'autres encore.

Enfin, et pas des moindres, je souhaiterais remercier ma famille qui a toujours été là pour moi tout au long de ces longues études, même si parfois c'était compliqué de voir où cela pouvait mener. Vous m'avez toujours soutenu et ce soutien m'a été plus que précieux.

Contents

Résumé

La simulation numérique de phénomènes physiques complexes requiert généralement l'utilisation d'un maillage. En mécanique des fluides numérique, cela consiste à représenter un objet dans un gros volume de contrôle. Cet objet étant celui dont l'on souhaite simuler le comportement. Usuellement, l'objet et la boîte englobante sont représentés par des maillages de surface linéaires et la zone intermédiaire est remplie par un maillage volumique. L'objectif de cette thèse est de s'intéresser à deux manières différentes de représenter cet objet. La première approche - dite immergée - consiste à mailler intégralement le volume de contrôle et ensuite à simuler le comportement autour de l'objet sans avoir à mailler explicitement dans le volume ladite géométrie. L'objet étant implicitement pris en compte par le schéma numérique. Le couplage de cette méthode avec de l'adaptation de maillage linéaire est notamment étudié. La deuxième approche - dite d'ordre élevé - consiste quant à elle à augmenter le degré polynomial du maillage de surface de l'objet. La première étape consiste donc à générer le maillage de surface de degré élevé et ensuite à propager l'information de degré élevé dans les éléments volumiques environnants si nécessaire. Dans ce cadre-là, il s'agit de s'assurer de la validité de telles modifications et de considérer l'extension des méthodes classiques de modification de maillages linéaires. Il convient également de développer de nouvelles méthodes de visualisation de maillages et de solutions, les méthodes classiques étant mises en défaut pour ce nouveau type d'entités.

Mots Clefs : Méthodes d'ordre élevé, Visualisation scientifique, Méthodes immergées, Mécanique des fluides numérique, Adaptation de maillage.

Abstract

The numerical simulation of complex physical phenomena usually requires a mesh. In Computational Fluid Dynamics, it consists in representing an object inside a huge control volume. This object is then the subject of some physical study. In general, this object and its bounding box are represented by linear surface meshes and the intermediary zone is filled by a volume mesh. The aim of this thesis is to have a look at two different approaches for representing the object. The first approach, called embedded method, consists in fully meshing the bounding box volume without explicitly meshing the object in it. In this case, the presence of the object is implicitly simulated by the CFD solver. The coupling of this method with linear mesh adaptation is in particular discussed. The second approach, called high-order method, consists on the contrary in increasing the polynomial order of the surface mesh of the object. The first step is therefore to generate a suitable high-order surface mesh and then to propagate the high-order information in the neighboring volume if necessary. In this context, it is mandatory to make sure that such modifications are valid and to consider the extension of classic linear mesh modification techniques. Also, a special care has to be done on the development of new mesh and solution visualization methods as the classic ones fail to properly render this new type of entities.

Keywords : High-order methods, Scientific visualization, Embedded methods, CFD, Mesh adaptation.

Introduction

Computers are commonly used to represent objects for various applications. However, while most of the studied objects can reasonably be assimilated as continuous entities, the representation of these objects on a computer has to be discrete due to the limited storage available on it. In this context, a discrete approximation of these objects is created. Usually, the nature of this discrete representation is done using a cloud of points. This might be sufficient for some applications, but this is definitely not the case in computer graphics or in numerical simulation. In these domains, the entities of the cloud of points are connected into simple shapes, like triangles, that the computer is able to process. The reunion of these shapes gives a discrete but continuous approximation of the studied object that is called a mesh. The nature of the mesh and its properties depend on the specific application. In numerical simulation, the mesh is part, as explained in [Loseille 2017], of the so-called *computational pipeline*. This pipeline is depicted in Figure 1.

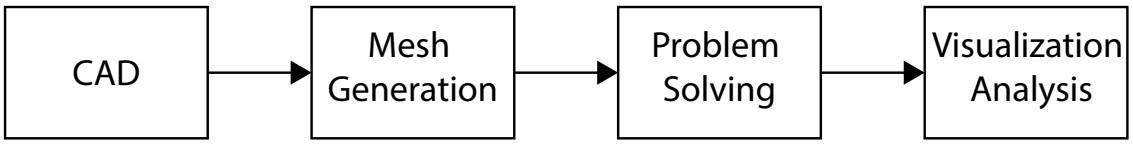

Figure 1 – Computational pipeline.

The main steps of this pipeline are described as follows:

- The geometry is usually defined using a CAD (Computer-Aided Design) model which gives a continuous representation of the geometry.
- From this CAD model, a surface mesh is deduced. Depending on the application, a volume mesh can be generated. For two-dimensional problems, the surface mesh is a set of edges and the volume mesh is composed of triangles and/or quadrilaterals. For the three-dimensional case, the surface mesh is made of triangles/quadrilaterals and the volume mesh has tetrahedra/prisms/pyramids/hexahedra.
- Using a mesh, a numerical scheme (Finite Element Method [Ciarlet 1978], Discontinuous Galerkin Method [Hesthaven and Warburton 2008], Boundary Element Method [Bonnet 1999], Finite Difference Method [Forsythe and Wasow 1960], Finite Volume Method [Van Leer 1979], Spectral Differences [Vanharen et al. 2017]) is applied to a given equation to simulate the chosen physics on the studied object. It consists in transforming the continuous equation into a discrete one by means of a mesh and to solve this discrete equation in order to deduce a numerical solution. Note that this part may be skipped for some applications.
- Finally, the obtained solution from the previous step as well as the mesh are analyzed and conclusions are taken from it.

This process is illustrated in Figures 2 and 3. In Figure 2, a CAD model of a supersonic aircraft is presented at the top. Using this CAD, a surface mesh is generated [Borouchaki et al. 2000] as it can be seen at the bottom. In Figure 3, the previously generated surface mesh is used as input for a volume meshing process. On the left, a tetrahedral volume mesh is generated while keeping the input triangular mesh intact [George et al. 2003] and in the middle, a full hexahedral volume mesh is generated as well as a quadrilateral surface mesh [Maréchal 2009]. Finally, on the right, a CFD

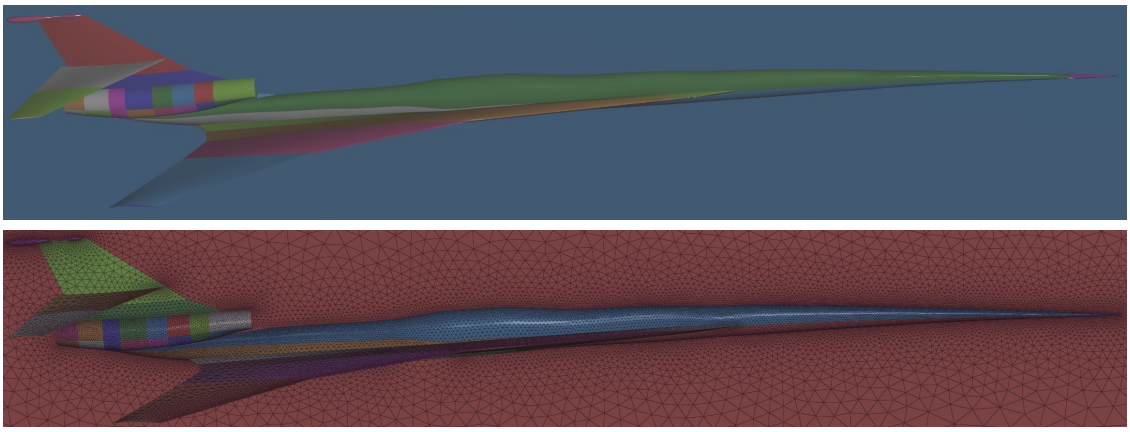

Figure 2 – CAD model (top) and surface mesh (bottom) generated from it.

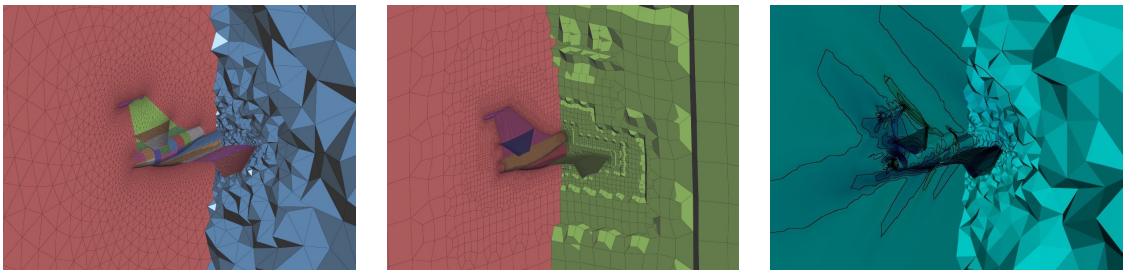

Figure 3 – Volume meshes generated from the surface mesh of Figure 2.
From left to right, tetrahedral-triangular mesh, hexahedral-quadrilateral mesh and CFD computation done on a tetrahedral-triangular mesh.

(Computational Fluid Dynamics) computation of a supersonic flow around the aircraft is performed using the tetrahedral mesh [Alauzet 2019]. To perform the visualization and the analysis of CAD models, meshes, and CFD solutions a visualization software is used [Loseille and Feuillet 2018].

It appears obvious that the mesh is the core component of this pipeline. The generation of these objects is a science in itself. Among the very large variety of methods that exist to generate surface or volume meshes, there is the Delaunay-based mesh generation [George et al. 1991a, Rebay 1993, Weatherill and Hassan 1994] (a brief introduction to the use of this method to generate simplicial (*e.g.* triangles/tetrahedra) meshes is done in Appendix A), the frontal mesh generation [Blacker and Stephenson 1991, Löhner and Parikh 1988], a combination of both of them [Marcum and Weatherill 1995, Frey et al. 1998, Borouchaki et al. 2000], the quadtree/octree-based mesh generation [Yerry and Shephard 1984, Shephard and Georges 1991, Schneiders and Bünten 1995, Maréchal 2001], medial axis mesh generation [Tam and Armstrong 1991, Price et al. 1995, Price and Armstrong 1997], ... A complete review of these methods can be found in [Frey and George 2008] or more recently in [Borouchaki and George 2017b] and [George et al. 2019]. Despite the huge variety of the proposed strategies, mesh generation still remains a tedious task and is not fail-safe. In particular, the automatic generation of all-hexahedral grids with the constraint of keeping intact an initial quadrilateral boundary mesh is still an open problem.

For its part, numerical simulation is a convenient way to perform experiments without the explicit use of a model and consequently enables to save money and time. In partic-

ular, the aeronautics industry quickly saw the interest of avoiding really expensive full- and model-size experiments by replacing it with numerical computations. It also offered the opportunity to simulate applications that were impossible to reproduce with surrogate experiments as it can be the case with military applications, catastrophic events simulations or difficult aeronautic configurations. Another advantage offered by the numerical simulation lies in the fact that it gives a global picture of the physical phenomenon where discrete measures only give a partial information about what really happens. Even with these advantages, the numerical simulation remained limited in the first place to the study of simple cases due to the huge computational costs involved in it. With the increase of the computational capabilities and the improvement of the numerical methods, it became possible to simulate more complex configurations on real-life geometrical cases. However, when dealing with such complex setups, the computational and modeling cost to get reliable results remains high. It is therefore of major interest to develop any strategy to improve these methods so that their costs are reduced.

Different strategies can be set to reduce the cost and/or improve the result of a numerical simulation:

- All the solvers and mesh generation tools were initially thought to work on sequential machines. With the hardware improvements, it is now possible to use the multithreading [Maréchal 2010, OpenMP 2018, Maréchal 2012, Khronos-Group 2017] or multi-processors [MPI-Forum 2015] capabilities of computers to accelerate the numerical computations by means of parallel computing. This approach has been successfully applied to numerical schemes [Law 1986], meshing problems [Chrisochoides and Nave 2003, Remacle et al. 2015, Loseille et al. 2017, Marot et al. 2019] or I/O issues [Maréchal 2018].
- Another idea consists in overcoming the meshing problem so that the geometries that have to be meshed in the end are easy. This is the principle of the **embedded or immersed method**. The studied object (in the example of Figure 2 it is the supersonic aircraft) is not explicitly meshed inside the volume, but implicitly taken into account by the CFD solver [Yang et al. 1997a, Löhner et al. 2004]. The advantage is that in this case the meshing process is not a problem anymore, all the type of meshes can be easily generated in an automatic fashion, and a solution is quickly obtained albeit some limitations on its accuracy. Nevertheless, this last problem can be solved to some extent.
- Historically, numerical simulation was the resolution of piecewise linear numerical solutions on linear meshes. However, it appeared that increasing the order of the polynomials used for the approximation of the solution yields a faster convergence even if it is greedier in memory. Moreover for the same accuracy, the used meshes are coarser than in the case of a linear approximation. Furthermore, it appeared that to get the full efficiency of the high-order methods, the used mesh should be high-order as well [Ciarlet and Raviart 1972, Lenoir 1986, Bassi and Rebay 1997]. In this context, the generation of **high-order meshes** has to be investigated as well as the generation and validation of high-order meshes [Dey et al. 1999, Sherwin and Peiró 2002, Persson and Peraire 2009, George and Borouchaki 2012, Johnen et al. 2013, Gargallo-Peiró et al. 2013, Xie et al. 2013]. As these objects are not linear anymore, special procedures have to be set-up. Also, the **visualization of both high-order meshes and solutions** has to be considered [Schroeder et al. 2006, Remacle et al. 2007, Peiró et al. 2015, Xu et al. 2017] as classic visualization techniques fail to render properly such objects.
- The local points density of the mesh, its quality and its adequation with the un-

derlying physics strongly impacts the quality of the numerical solution provided by the numerical scheme. Therefore, a judicious modification of the mesh features can improve the fidelity of the simulation. This can be performed by several ways.
A historical approach is based on empirical physical considerations. When solving viscous flows with the Navier-Stokes equations, the boundary layer regions that are near the surface must be resolved to capture the relevant physics. These regions are usually *a priori* known and often have very stringent numerical requirements due to the involvement of high-gradients and non-linear physics like turbulence. An approach to solve this problem is to design pseudo-structured meshes that are fitted to capture these physical features [Pirzadeh 1994, Marcum 1995, Löhner 2000, Aubry and Löhner 2009, Alauzet and Marcum 2015, Maréchal 2016]. These meshes are called **boundary-layer or viscous meshes**. An example of boundary layer mesh [Alauzet et al. 2017] around the aircraft presented above is given on Figure 4 on the left, whereas on the right, the CFD solution computed on it is shown. We indeed see that the boundary layer mesh is able to capture the features aligned with the boundary of the wing. Even if its effectiveness is proved, this method suffers from the lack of interaction with the physics occurring far from the aircraft. Also, this approach can lead to areas where the mesh is less refined than necessary and its use is limited to the study of viscous flow around the objects in the context of Navier-Stokes equations.

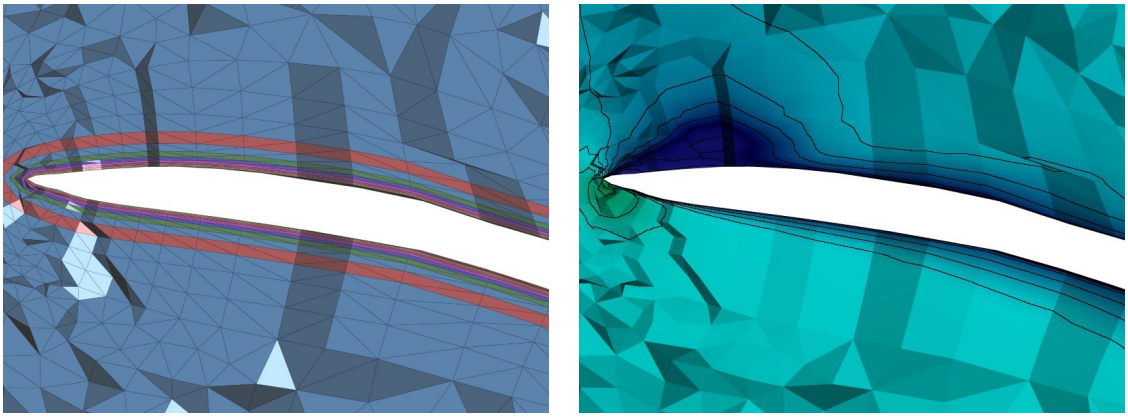

Figure 4 – Mesh (left) and CFD solution (right) resulting from a boundary layer meshing performed around a wing of the aircraft model presented in Figures 2 and 3.

In this context, another strategy can be used: the (anisotropic) **mesh adaptation** [George et al. 1991b, Castro-Díaz et al. 1997, Frey and Alauzet 2005, Gruau and Coupez 2005, Li et al. 2005, Alauzet and Loseille 2010]. It consists in modifying the local size and orientation of the discretization so that it increases the fidelity of the description. In a same manner as boundary layer meshing, this technique is useful to detect physical phenomena but in this case without *a priori* knowledge of the physical phenomenon. The idea of a mesh adaptation process is to generate meshes and solutions in an iterative way. A solution is computed on a mesh, an indicator that determines a target size map for each point of the mesh is deduced, a new mesh is generated from it, and the process can restart on this mesh. Using this, a finer mesh is generated in the regions of physical interest while a coarser mesh is generated in the regions of low interest. This strategy yields therefore to compute accurately complex phenom-

ena without the huge cost that would be expected without its use. Mesh adaptation has been widely applied in CFD to the case of linear solutions on linear meshes, but its extension to high-order methods has been studied [Mirebeau 2010, Mbinky 2013, Hecht and Kuate 2014, Coulaud and Loseille 2016] and its use in other physics as well [Vanharen et al. 2018]. Also, in the case of mesh adaptation for the RANS approximation of the Navier-Stokes equations, it has been shown that the boundary layer mesh was naturally recovered by the mesh adaptation process [Frazza 2018, Alauzet and Frazza 2019]. An example of adapted mesh [Loseille et al. 2018] of the aircraft presented above is given on Figure 5 on the left, whereas on the right, the CFD solution computed on it is shown. We indeed see that the adapted mesh is able to capture all the physical features induced by the presence of the aircraft as the mesh density fits to the studied physics and even in regions far from the plane. Moreover, we can see that the physical phenomenon is way more captured than in Figure 3.

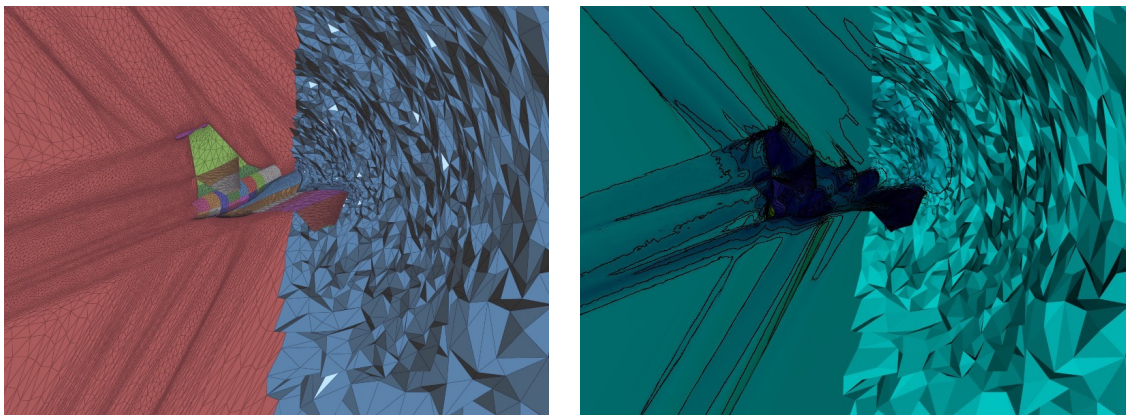

Figure 5 – Mesh (left) and solution (right) resulting from a mesh adaptation using CFD computations performed on the aircraft model presented in figures 2 and 3.

Of course, a coupling between some of the exposed strategies can be done and is even most of the time mandatory.
Facing all these challenges, the aim of this Ph.D. thesis is to tackle some of them.

Contributions

The objectives of this Ph.D. were multiple. On the one hand, the objective was to tackle the subject of the embedded boundary method and to integrate this approach in the already existing strategies of the team Gamma. On the other hand, the aim was to continue the work initiated in the team Gamma with [George and Borouchaki 2012, George et al. 2016] high-order meshes. This domain is relatively new and a lot of points have to be clarified about it. To this end, this thesis has brought the following contributions:

Mesh adaptation for the embedded boundary method. An implementation of the embedded boundary method for the Euler equations has been performed in the in-house solver `Wolf` for the 2D and 3D case. This approach is based on a cut-cell approach in the context of a vertex-centered Finite Volume solver. Then, the coupling of this approach with mesh adaptation has been studied.

Optimization of quadratic meshes and applications. The study and the development of strategies for dealing with mesh optimization in the case of meshes of degree two in

2D and 3D has been made. The developments have been done in the meshing modules of `Wolf` and several applications deriving from quadratic mesh optimization have been deduced: robust quadratic mesh generation, moving-mesh methods and boundary-layer mesh generation.

High-order error estimates for high-order parametric surface mesh generation. The generation of high-order meshes is mainly driven by the generation of suitable surface high-order meshes. Starting from the work of [Coulaud and Loseille 2016] performed in `pkmetrix`, a new set of anisotropic error estimates have been developed in `Feflo.a/AMG` to perform high-order parametric surface mesh generation.

Visualization of high-order meshes and solutions. Classic mesh visualization methods fail to properly render high-order meshes and solutions. In this context, starting from an already existing visualization kernel of linear and quadratic triangles in `OpenGL` 4.0, the development of the visualization capabilities for all the standard meshing elements from degree 1 to 4 has been done. Then, the rendering of high-order solutions on these elements has been implemented and has enabled an almost pixel exact rendering of them in real time. All these developments have been made in the new release of `Vizir`.

Scientific communications

Journal papers

- **Optimization of P^2 meshes and applications**, R. Feuillet, A. Loseille and F. Alauzet, Submitted to *Computer-Aided Design*.
- **On Pixel-exact rendering for high-order mesh and solution**, R. Feuillet and A. Loseille, Under preparation.

Peer-reviewed proceedings

- **P^2 mesh optimization operators**, R. Feuillet, A. Loseille and F. Alauzet, 27th International Meshing Roundtable, Albuquerque, NM, USA, 2018.
- **Anisotropic error estimate for high-order parametric surface mesh generation**, R. Feuillet, O. Coulaud and A. Loseille, 28th International Meshing Roundtable, Buffalo, NY, USA, 2019.

Proceedings

- **Vizir: High-order mesh and solution visualization using OpenGL 4.0 graphic pipeline**, A. Loseille and R. Feuillet, 56th AIAA Aerospace Sciences Meeting, AIAA Scitech, 2018.
- **Connectivity-change moving mesh methods for high-order meshes: Toward closed advancing-layer high-order boundary layer mesh generation**, R. Feuillet, A. Loseille, D. Marcum and F. Alauzet, 2018 Fluid Dynamics Conference, AIAA AVIATION Forum, 2018.
- **Mesh adaptation for fluid-structure interaction problems**, J. Vanharen, R. Feuillet, F. Alauzet, 2018 Fluid Dynamics Conference, AIAA AVIATION Forum, 2018.
- **A closed advancing-layer method for generating curved boundary layer mesh**, R. Feuillet, D. Marcum and F. Alauzet, AIAA AVIATION 2019 Forum, 2019.

Research reports

- **Mesh adaptation for the embedded boundary method in CFD** , R. Feuillet, A. Loseille and F. Alauzet, Research Report, INRIA, 2019.

Communications

- **Generation and motion of high-order meshes based on a high-order linear elasticity model**, R. Feuillet, A. Loseille and F. Alauzet, ECCM VI - ECCFD VII, Glasgow, UK, 2018.
- **High-order mesh operations based on a linear elasticity model**, R. Feuillet, A. Loseille and F. Alauzet, ICOSAHOM, London, UK, 2018.

Awards

- **Best Student Paper Award**, 28th International Meshing Roundtable, Buffalo, NY, USA, 2019.

Outline

This work is composed of two parts.

Part I deals with mesh adaptation for the embedded boundary method in CFD. In a first chapter, the used framework for the mesh adaptation for the Euler equations is recalled. Then, in a second chapter, the implementation of the embedded boundary method for the compressible Euler equations is detailed and its coupling with mesh adaptation is studied.

Part II is about high-order. In a first chapter, the framework used for high-order meshes in this work is presented. In particular, the analogy between high-order and Bézier elements is exposed. Then, a second chapter is devoted to the development of anisotropic error estimates suitable for high-order parametric surface mesh generation. Afterwards, a third chapter takes care of the generalization of the classic mesh optimization operators to meshes of degree two with some applications of it. Finally, the last chapter deals with the visualization of high-order meshes and solutions.

Résumé en français

Introduction

Les ordinateurs sont communément utilisés pour représenter des objets et ce, pour diverses applications. Pour de nombreuses modélisations physiques, l'objet étudié (avion, voiture, personne, ...) est assimilé à une entité continue. Cependant, il n'est pas possible de faire de même sur un ordinateur du fait du caractère fini des informations stockées. Dans cette optique, les objets sont approchés par des entités discrètes qui sont en général des ensembles de points. Pour de nombreuses applications, comme en infographie ou en calcul numérique, une table de connectivité est rajoutée, permettant ainsi de définir des formes géométriques simples, comme des triangles, qu'un ordinateur sait gérer. L'ensemble de ces formes élémentaires définit alors une approximation discrète mais continue de l'objet que l'on appelle un maillage. La nature du maillage ainsi que ses propriétés dépendent quant à elles de l'application souhaitée. En calcul numérique, par exemple, le maillage fait partie intégrante du processus de calcul [Loseille 2017]. L'ensemble des étapes de ce processus est présenté sur la Figure 6.

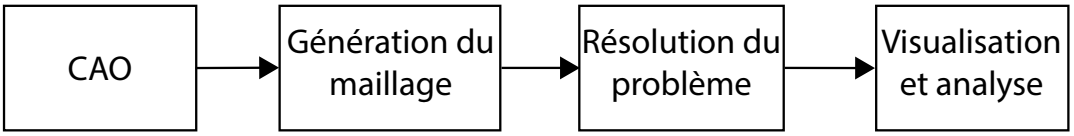

FIGURE 6 – Étapes du processus de calcul.

Les différentes étapes se décrivent de la manière suivante :

- La géométrie est définie de manière continue à l'aide d'un modèle de CAO (Conception Assistée par Ordinateur).
- À partir d'un modèle de CAO, un maillage de surface est créé et, en fonction des applications, un maillage de volume peut être généré. Dans le cadre d'une analyse bidimensionnelle, le maillage de surface est une liste d'arêtes et le maillage de volume est constitué de triangles et/ou de quadrilatères. Pour le cas tridimensionnel, le maillage de surface est composé de triangles et/ou de quadrilatères alors que le maillage de volume peut comporter des tétraèdres/prismes/pyramides/hexaèdres.
- Avec l'aide d'un maillage, un schéma numérique (Méthode des Éléments Finis [Ciarlet 1978], Méthode de type Galerkine Discontinue [Hesthaven and Warburton 2008], Méthode des Élements Finis de Frontière [Bonnet 1999], Méthode des Différences Finies [Forsythe and Wasow 1960], Méthode des Volumes Finis [Van Leer 1979], Méthode des Différences Spectrales [Vanharen 2017]) est appliqué à une équation donnée afin de simuler la physique qui lui est associée sur l'objet étudié. Cela consiste notamment à transformer une équation continue en un équivalent discret par l'intermédiaire du maillage. À partir de cela, une solution numérique discrète est déduite.
- Enfin, la solution obtenue précédemment ainsi que le maillage utilisé sont analysés et des conclusions en sont déduites.

Ces étapes sont illustrées avec les Figures 7 et 8. Sur la Figure 7, un modèle de CAO d'un avion supersonique est présenté en haut. À partir de ce modèle, un maillage de surface est généré [Borouchaki et al. 2000], en bas. Sur la Figure 8, le maillage de surface

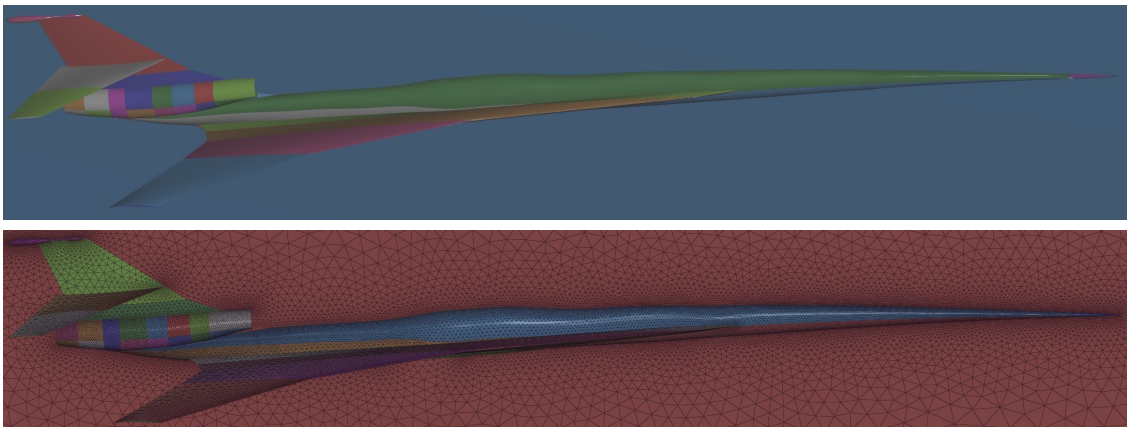

FIGURE 7 – Modèle de CAO (haut) et maillage de surface (bas) généré à partir de ce modèle.

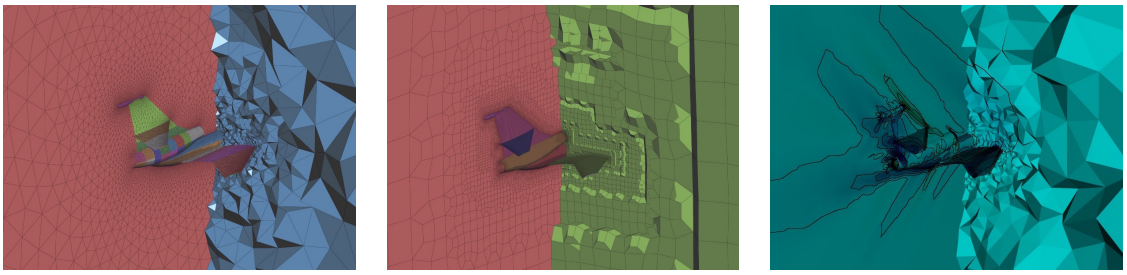

FIGURE 8 – Maillages volumiques générés à partir du maillage de surface de la Figure 7. De gauche à droite, maillage tétraèdrique/triangulaire, maillage hexaèdrique-quadrilatéral et simulation des équations de la mécanique des fluides sur un maillage tétraèdrique/triangulaire.

précédent est utilisé comme donnée d'entrée à deux mailleurs volumiques. À gauche, un maillage de volume tétraèdrique est généré tout en laissant le maillage triangulaire de surface intact [George et al. 2003] tandis qu'au milieu, un maillage de volume hexaédrique pur est généré avec un maillage de surface quadrilatéral associé [Maréchal 2009]. Finalement, à droite une simulation des équations de la mécanique des fluides est réalisée sur le maillage de tétraèdres [Alauzet 2019]. La visualisation et l'analyse de la CAO, du maillage et des solutions calculées sont quant à elles rendues possibles grâce un logiciel de visualisation [Loseille and Feuillet 2018].

Il apparaît indéniable que le maillage est un composant-clé de ce processus. L'étude et la génération de tels objets peuvent être considérées comme une science tellement le nombre d'approches disponibles est vaste. Notamment, pour générer des maillages de volume et/ou de surface, on compte les méthodes de génération de maillage basée sur une triangulation de Delaunay [George et al. 1991a, Rebay 1993, Weatherill and Hassan 1994], les méthodes frontales [Löhner and Parikh 1988, Blacker and Stephenson 1991], une combinaison de ces dernières [Marcum 1995, Frey et al. 1998, Borouchaki et al. 2000], les méthodes basées sur des arbres de type quadtree/octree [Yerry and Shephard 1984, Shephard and Georges 1991, Schneiders and Bünten 1995, Maréchal 2001] ou bien les méthodes utilisant l'axe médian [Tam and Armstrong 1991, Price et al. 1995, Price and Armstrong 1997]. Une revue complète de toutes les méthodes peut se trouver dans [Frey and George 1999] ou plus récemment dans [Borouchaki and George 2017a] et [George et al. 2018]. Malgré cette

diversité, la génération de maillage reste une tâche ardue et faillible. En particulier, la génération automatique de maillages hexaédriques purs avec la contrainte de laisser intact un maillage de surface quadrilatéral reste un problème ouvert.

De son côté, la simulation numérique est un moyen pratique de réaliser des expériences sans l'utilisation explicite d'une maquette, ce qui permet de faire des économies de temps et d'argent. Ceci a été rapidement constaté par l'industrie aéronautique qui y a vu l'intérêt d'éviter de faire de coûteuses expériences en taille réelle et de les remplacer par des simulations numériques. Cela permit ainsi de réaliser des expériences que des modèles de substitution ne peuvent reproduire comme des configurations aéronautiques très complexes ou des simulations de catastrophes naturelles. Malgré ces avantages, l'utilisation de la simulation numérique demeura limitée pendant plusieurs décennies, à cause notamment des énormes coûts de calcul. Grâce aux progrès des capacités de calcul et à l'amélioration des méthodes numériques employées, le calcul numérique, avec notamment l'utilisation de la méthode des éléments finis, est devenu de plus en plus utilisé. Il n'en reste pas moins que dans le cadre de simulations impliquant des phénomènes complexes, les temps de modélisation et de calcul restent très élevés et qu'il est toujours intéressant de les réduire.

À cet effet, diverses stratégies existent pour réduire le coût et/ou améliorer le résultat d'une simulation numérique :

- Tous les solveurs et les mailleurs ont été initialement pensés pour une utilisation sur des machines séquentielles (c'est-à-dire munies que d'une seule unité de calcul). Avec les progrès des architectures matérielles, il est tout à fait possible d'accélérer les calculs numériques à l'aide du calcul parallèle, que cela soit dans un contexte à mémoire distribuée [MPI-Forum 2015] ou partagée [Maréchal 2010, OpenMP 2018, Maréchal 2012, Khronos-Group 2017]. Cela a été réalisé avec succès pour des schémas numériques [Law 1986], des méthodes de maillage [Chrisochoides and Nave 2003, Remacle et al. 2015, Loseille et al. 2017, Marot et al. 2019] ou des problèmes de lecture/écriture [Maréchal 2018].
- Une autre approche consiste à éviter les difficultés inhérentes à un processus de maillage en s'arrangeant pour mailler seulement des objets simples. Cette approche dite **approche immergée** consiste à ne pas mailler explicitement l'objet étudié (par exemple, dans la Figure 7, l'objet étudié est l'avion) dans le maillage de volume environnant mais à le faire prendre implicitement en compte par le solveur lors de la résolution numérique des équations de la physique étudiée [Yang et al. 1997a, Löhner et al. 2004]. Le principal avantage est que dans ce cas, les objets à mailler sont élémentaires et peuvent être gérés par à peu près toutes les méthodes de maillage possibles et aussi une solution numérique peut être rapidement obtenue malgré quelques limitations quant à la précision. Ce dernier problème peut néanmoins être résolu dans une certaine mesure.
- Historiquement, la simulation numérique consistait essentiellement en la résolution d'une solution numérique linéaire par morceaux sur un maillage également linéaire par morceaux. Cependant, il est apparu assez vite que l'augmentation du degré polynomial de la solution permet une convergence plus rapide même si cela nécessite des ressources mémoires plus importantes. Il convient cependant de remarquer qu'augmenter l'ordre de la solution permet aussi d'utiliser un maillage de plus en plus grossier. Malgré tout, il fut mis en évidence que la pleine efficacité de ces méthodes n'est garantie que si le degré polynomial des éléments du maillage est aussi élevé [Ciarlet and Raviart 1972, Lenoir 1986, Bassi and Rebay 1997]. Face à ces observations, il apparaît alors primordial d'investiguer sur les méthodes de génération de **maillages d'ordre élevé** [Dey et al. 1999,

Sherwin and Peiró 2002, Persson and Peraire 2009, George and Borouchaki 2012, Johnen et al. 2013, Gargallo-Peiró et al. 2013, Xie et al. 2013]. Il convient notamment de s'intéresser à la génération mais surtout à la validité de tels objets. Ceci étant, il convient de remarquer que les objets considérés ne sont alors plus nécessairement linéaires, et que par conséquent, beaucoup d'approches classiques ne fonctionnent plus. C'est le cas notamment de la **visualisation de maillages et de solutions d'ordre élevé** associées [Schroeder et al. 2006, Remacle et al. 2007, Peiró et al. 2015, Xu et al. 2017].

- La densité locale de points du maillage, sa qualité et son adéquation avec la physique sous-jacente peuvent fortement impacter la qualité de la solution numérique fournie par le schéma numérique. Dans cette optique, il convient de s'intéresser à des modifications judicieuses du maillage afin d'améliorer la fidélité de la simulation. Cela peut se faire de diverses manières.
  Parmi elles, il y a une approche historique basée sur des considérations physiques empiriques. Lors de la résolution d'écoulements visqueux par les équations de Navier-Stokes, les régions de couche limite se trouvant près de la surface doivent être résolues à tout prix afin d'y capturer la physique pertinente qui y est présente. Ces zones sont *a priori* connues et demandent des analyses très fines à cause de la présence de gradients très élevés et d'une physique non-linéaire telle que la turbulence. Une solution à ce problème est alors de proposer des maillages pseudo-structurés qui permettent de capturer cette physique [Pirzadeh 1994, Marcum and Weatherill 1995, Löhner 2000, Aubry and Löhner 2009, Alauzet and Marcum 2015, Maréchal 2016]. De tels maillages sont appelés **maillages visqueux ou de couche limite**. Un exemple de maillage de couche limite [Alauzet et al. 2017] autour de l'avion étudié plus haut est donné sur la Figure 9 à gauche, tandis qu'à droite, le résultat d'une simulation des équations de Navier-Stokes est montré. On voit notamment que le maillage de couche limite est capable de capturer les phénomènes physiques alignés avec le bord de l'aile. Malgré son efficacité avérée, cette méthode souffre de son manque d'interaction avec la physique, ce qui a pour effet pervers de mailler finement à des endroits où en fait il ne se passe rien et au contraire de ne pas mailler assez finement là où il se passe quelque chose. De plus, son utilisation reste limitée aux équations de Navier-Stokes et à l'étude de phénomènes visqueux dans le voisinage direct des objets considérés.

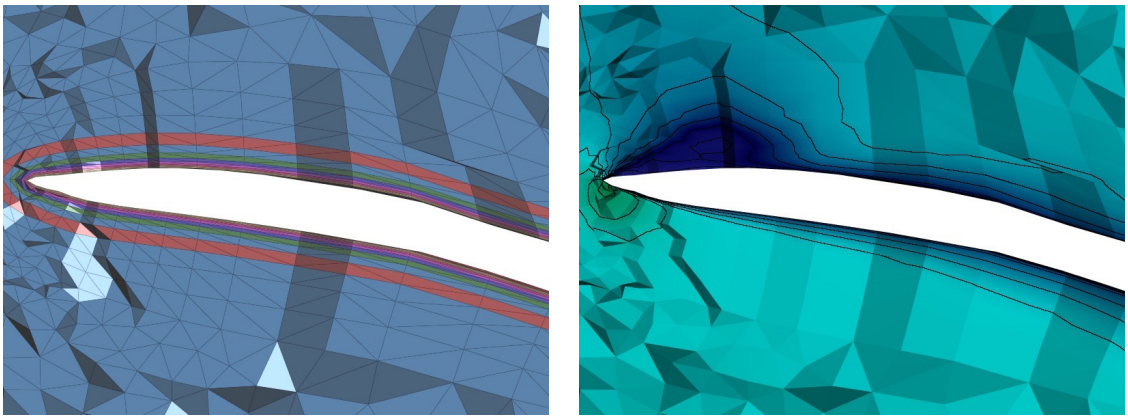

FIGURE 9 – Maillage (gauche) et solution des équations de la mécanique des fluides (droite) résultant d'un maillage de couche limite réalisé autour de l'aile de l'avion des Figures 7 et 8.

Dans ce cadre-là, une autre stratégie peut être utilisée : **l'adaptation de maillage** (anisotrope) [George et al. 1991b, Castro-Díaz et al. 1997, Frey and Alauzet 2005, Gruau and Coupez 2005, Li et al. 2005, Alauzet and Loseille 2010]. L'idée consiste à modifier localement la taille et l'orientation des éléments composant le maillage afin que la fidélité au phénomène physique soit améliorée. À l'instar du maillage de couche limite, cette technique permet de capturer des phénomènes physiques, mais avec dans ce cas aucune connaissance *a priori* de ces derniers. L'approche résulte d'un procédé qui génère maillages et solutions d'une manière itérative. Une solution est calculée sur un maillage. À partir de cela, un indicateur d'une carte de taille « optimale » est déduit. Cette carte de taille est alors utilisée pour générer un nouveau maillage et le procédé recommence. Ce faisant, un maillage plus fin est généré dans les régions contenant de la physique pertinente tandis que le maillage devient plus grossier dans d'autres régions ne présentant pas d'intérêt particulier. Cette stratégie permet alors de capturer de manière précise des phénomènes complexes sans le coût élevé auquel on pourrait naïvement s'attendre. L'adaptation de maillage est utilisée massivement en mécanique des fluides numérique ainsi que ses domaines connexes [Vanharen et al. 2018]. Le procédé est désormais bien établi pour ce qui est du cas des solutions linéaires sur des maillages linéaires mais son extension à l'ordre élevé est aussi considérée [Mirebeau 2010, Mbinky 2013, Hecht and Kuate 2014, Coulaud and Loseille 2016]. Il a aussi été également montré que dans le cadre des équations de Navier-Stokes, l'adaptation permet naturellement de trouver un maillage de couche limite [Frazza 2018, Alauzet and Frazza 2019]. Un exemple de maillage adapté [Loseille et al. 2018] est montré sur la Figure 10 dans le cas d'étude précédent de l'avion. À gauche, se trouve le maillage adapté, alors qu'à droite se trouve la solution numérique associée. On voit notamment que le maillage adapté est en mesure de capturer tous les phénomènes physiques induits par la présence de l'avion et ce, même dans des zones éloignées de l'appareil. On peut en particulier observer que le maillage colle à la physique et que la solution numérique est beaucoup plus propre que sur la Figure 8.

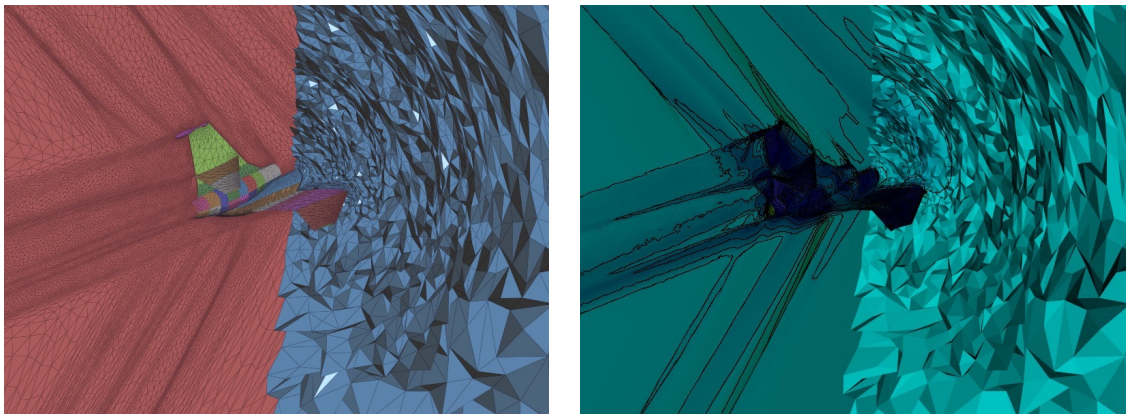

FIGURE 10 – Maillage (gauche) et solution (droite) résultant d'un processus d'adaptation de maillage appliqué aux équations de la mécanique des fluides dans le cas de l'avion des Figures 7 et 8.

Bien entendu, il est tout à fait possible, voire même recommandé, de coupler plusieurs de ces stratégies entre elles.
Face à ces multiples défis, l'objet de cette thèse est de s'intéresser à plusieurs d'entre eux. Cette thèse a été préparée au sein de l'équipe-projet GAMMA de l'INRIA et a donc

consisté à s'intéresser à deux des problématiques précédemment soulevées que le projet souhaitait examiner. D'une part, cela a consisté à s'intéresser à la méthode des frontières immergées et à intégrer le principe de cette approche dans les stratégies d'adaptation de maillage développées dans ce projet. D'autre part, l'objectif était de continuer le travail sur les maillages d'ordre élevé initié dans le projet avec [George and Borouchaki 2012, George et al. 2016]. Ce domaine de recherche est assez récent et il reste toujours un très grand nombre de choses à clarifier. À cet effet, un résumé des travaux faits pendant cette thèse est présenté ci-après.

Adaptation de maillage couplée à la méthode des frontières immergées en mécanique des fluides

Au sein du projet Gamma, un solveur volumes finis centré aux sommets nommé `Wolf` est développé. Ce solveur est notamment capable de résoudre les équations d'Euler et de Navier-Stokes, deux équations de la mécanique des fluides, et est ainsi utilisé comme solveur-test pour les techniques d'adaptation de maillages développées dans le projet. Le principe de l'adaptation de maillage usuellement employée est présenté dans le graphique de la Figure 11.

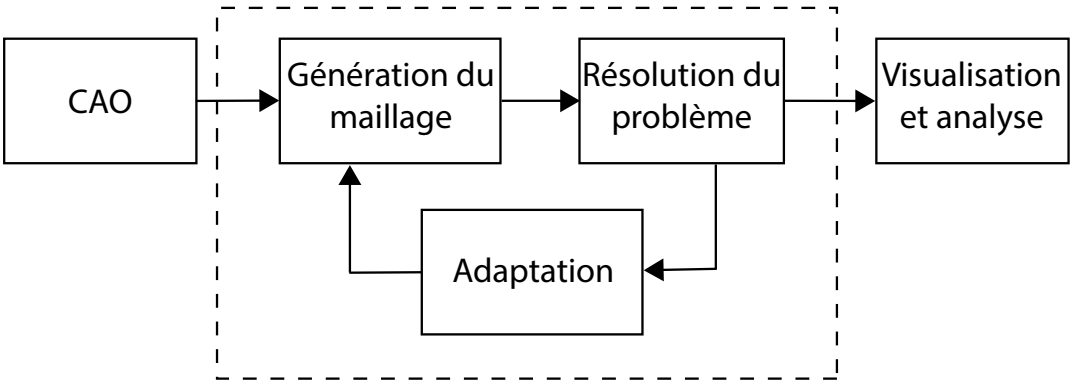

Figure 11 – Étapes du processus de calcul couplé avec l'adaptation de maillage.

L'idée principale de ce processus est d'obtenir un couple solution/maillage qui soit fidèle au phénomène que l'on souhaite reproduire. Dans cette optique, l'adaptation de maillage s'inscrit dans une démarche itérative. Au début, un maillage est généré sans connaissance *a priori* de la physique étudiée. À partir de ce maillage, le solveur est utilisé pour résoudre les équations de cette physique. La solution numérique calculée permet alors de définir une carte de taille (discrète) mieux adaptée à la solution. Cette carte (ou métrique) qui s'exprime par le biais de tenseurs de métrique peut être définie à l'aide de la dérivée seconde (matrice hessienne) de la solution et pour un nombre voulu de points dans le maillage. À partir de ces métriques, un nouveau maillage satisfaisant les tailles prescrites est généré. Le procédé peut alors recommencer sur ce nouveau maillage avec un nouveau calcul. Ce procédé est illustré sur les Figures 12 et 13. Un écoulement bidimensionnel supersonique (Mach 2) est simulé avec les équations d'Euler autour d'un profil d'aile de type NACA0012. La simulation sur un maillage non-adapté est donnée sur la Figure 12. On observe que l'onde de choc supersonique est bien identifiée en amont de l'aile mais aussi que l'onde de choc se dissipe au fur et à mesure que le maillage grossit en s'éloignant du profil. Au contraire, après quelques itérations d'un processus adaptatif, on obtient le résultat de la Figure 13. On observe alors que l'adaptation de maillage a parfaitement capturé les ondes de choc et que ces dernières ne se dissipent plus dès que l'on s'éloigne de l'aile.

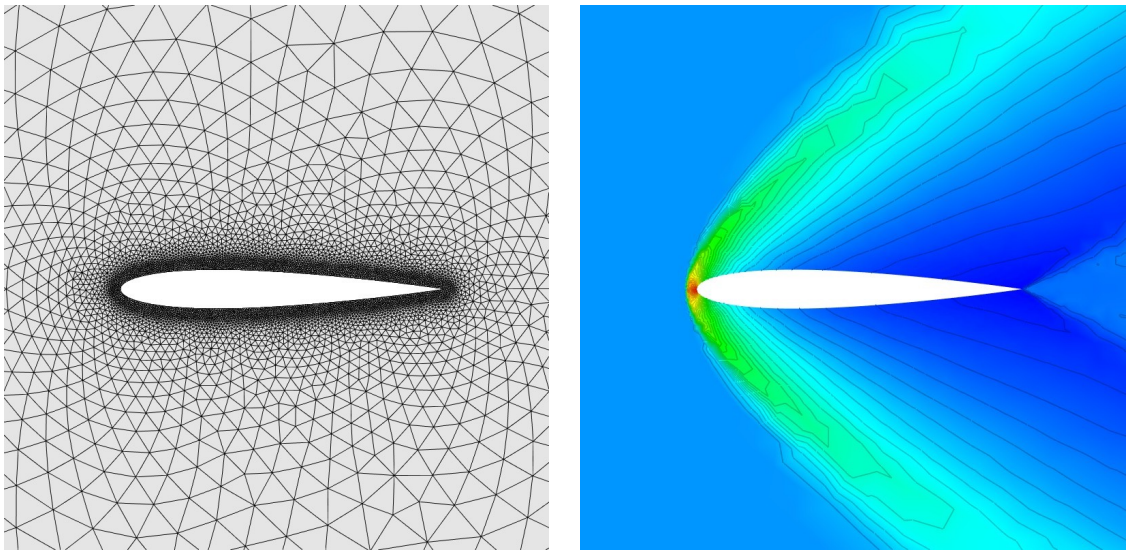

FIGURE 12 – Maillage non adapté (gauche) et solution (champ de densité) des équations d'Euler (droite) sur un maillage de NACA0012.

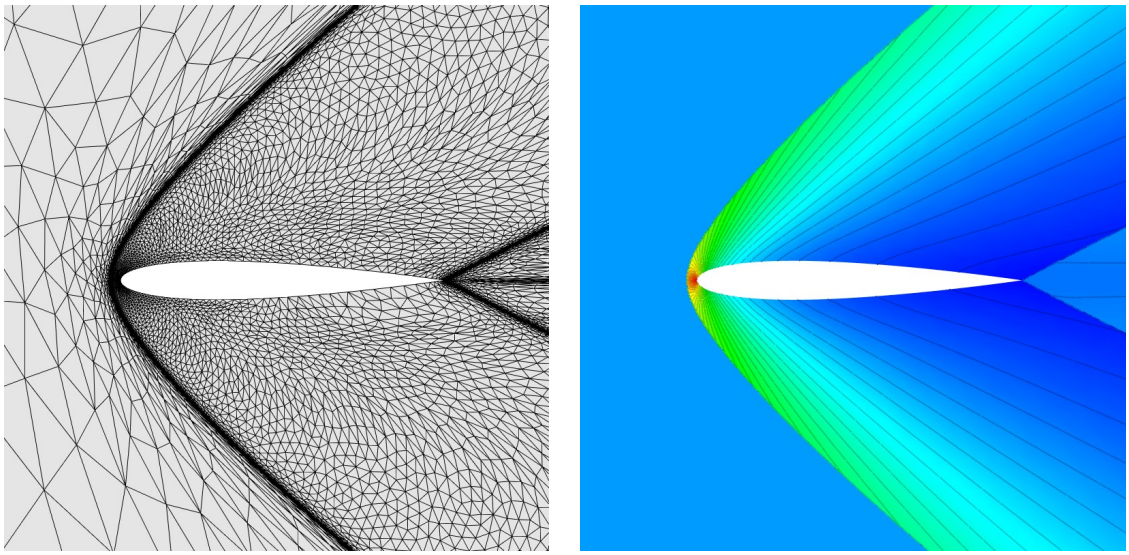

FIGURE 13 – Maillage adapté (gauche) et solution (champ de densité) des équations d'Euler (droite) sur un maillage de NACA0012.

Ce genre de stratégie est désormais mature et a fait ses preuves pour des simulations d'équations telles que les équations d'Euler que cela soit pour du 2D ou du 3D.
Dans notre cas, nous nous intéressons à une variante de son utilisation dans le cas particulier de la méthode des frontières immergées. Ici contrairement aux approches standards, le maillage de l'objet n'est pas explicitement représenté dans le maillage de calcul, mais est seulement pris en compte par le solveur. À cet effet, il a été rajouté dans `Wolf`, un module qui permet de gérer les maillages immergés qui sont soit représentés par une liste d'arêtes en 2D ou par une liste de triangles en 3D. Il est à noter que la géométrie considérée n'est pas forcément un maillage « conforme ». Dans une première phase, l'intersection de l'objet est calculée avec le maillage de calcul qui n'est pas nécessairement conforme avec l'objet. À partir de cela, il est déduit une modification locale du schéma volumes finis qui couplée avec la création d'une condition aux limites dédiée permet de simuler la présence de l'objet. Les résultats de cette approche peuvent être observés sur la Figure 14. On simule à nouveau

l'écoulement bidimensionnel supersonique (Mach 2) avec les équations d'Euler autour d'un profil d'aile immergé de type NACA0012. Une valeur constante qui correspond à un état de référence est imposée à l'intérieur de l'objet. On observe alors que la tendance globale est là, même si au niveau des détails une inévitable perte de précision est visible.

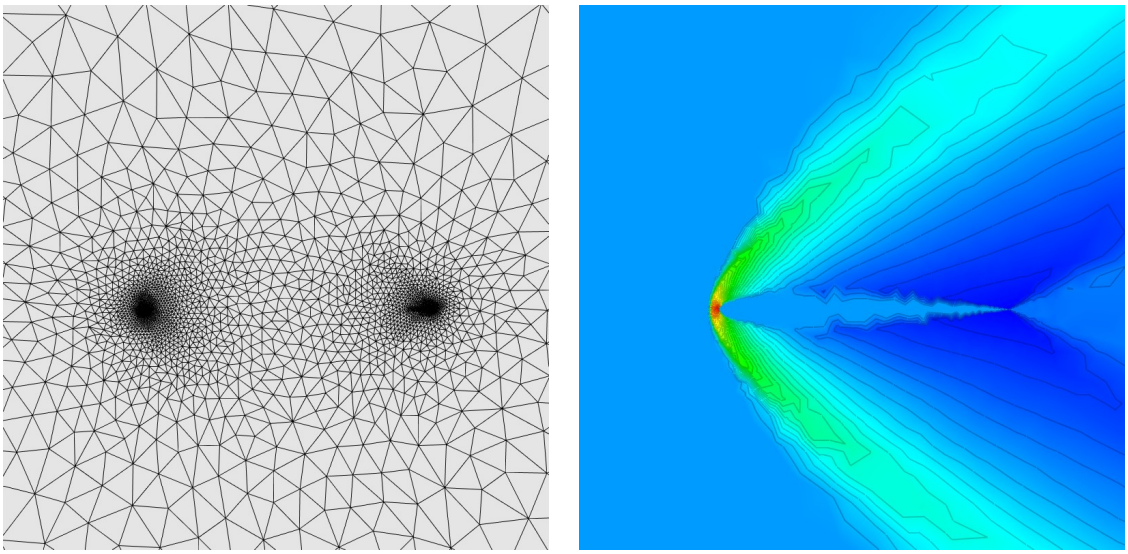

FIGURE 14 – Maillage non adapté (gauche) et solution (champ de densité) des équations d'Euler (droite) en immersion de frontières avec un maillage immergé de NACA0012.

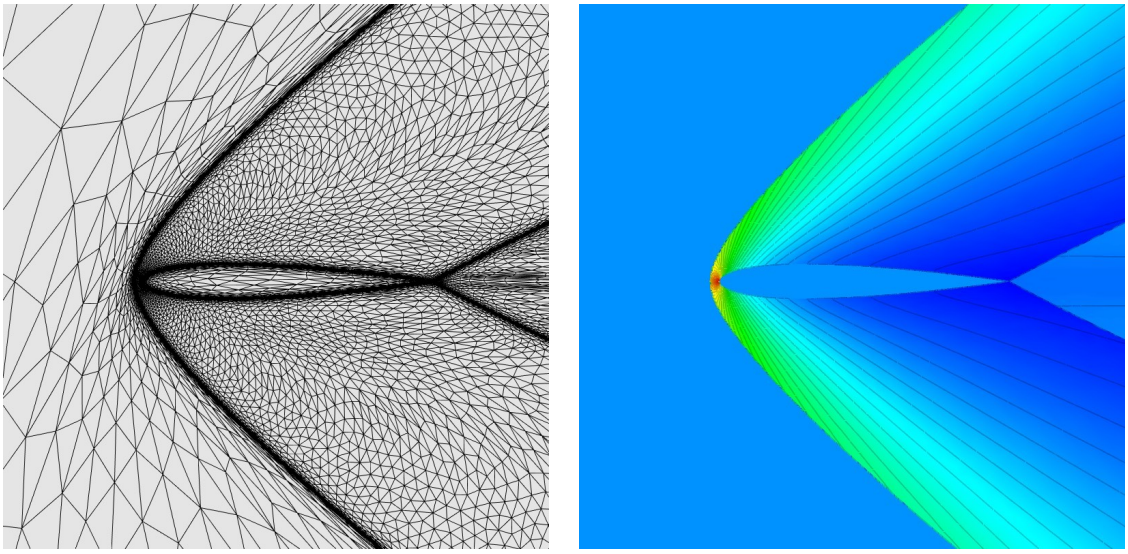

FIGURE 15 – Maillage adapté (gauche) et solution (champ de densité) des équations d'Euler (droite) en immersion de frontières avec un maillage immergé de NACA0012.

Une stratégie pour améliorer la précision des résultats tout en évitant de mailler l'objet consiste alors à faire de l'adaptation de maillage. En utilisant, la solution calculée sur le maillage de fond, l'imposition d'une valeur constante à l'intérieur de l'objet permet naturellement de différencier l'intérieur de l'extérieur tandis que les conditions aux limites permettent à la physique d'être là. Les résultats de l'adaptation de maillage sont visibles sur les images de la Figure 15, et on voit notamment que la physique tout comme l'objet sont capturés par l'adaptation de maillage. Ceci est notamment confirmé si l'on regarde le

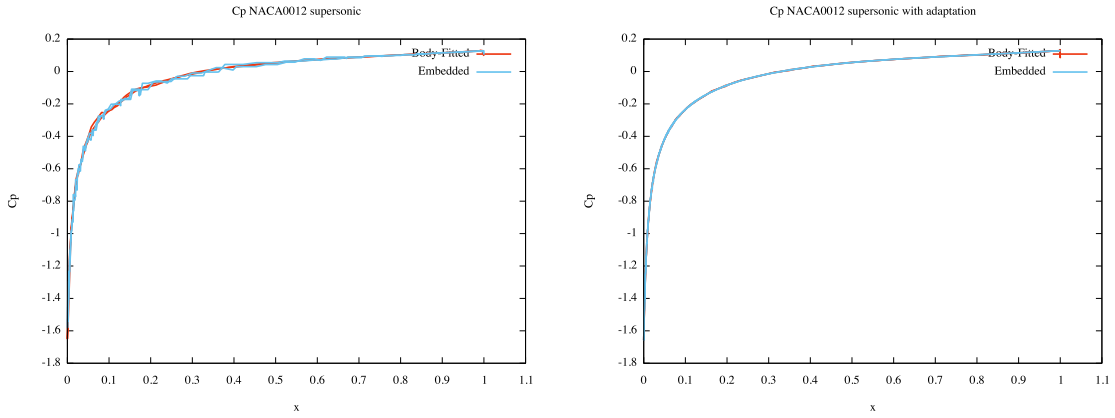

FIGURE 16 – Coefficient local de pression C_p autour d'un profil d'aile de type NACA0012 en régime supersonique pour les cas de géométrie immergée (*embedded*) et géométrie exacte (*body-fitted*) obtenues sans (gauche) et avec (droite) adaptation.

coefficient local de pression C_p autour du profil d'aile sur la Figure 16. On voit notamment que sans adaptation, les profils immergés (*embedded*) et exacts (*body-fitted*) ne sont pas identiques alors que l'adaptation de maillage permet de les avoir de manière quasi confondue. Ce faisant, on arrive alors à observer que l'adaptation de maillage permet de rendre l'approche immergée fiable et cela est vrai que ce soit en 2D ou en 3D.
Enfin, il est à noter que l'étude d'un objet mobile immergé a été considérée, mais l'analyse du schéma numérique proposé a montré que cette méthode n'est pas naturellement extensible aux objets mobiles. Cela en a donc considérablement réduit l'attrait dans la mesure où un de ses intérêts initiaux était de considérer de multiples objets immergés mobiles comme cela peut être observé lors d'une fragmentation.

Maillages d'ordre élevé

La génération et la validation de maillages d'ordre élevé est un problème complexe et loin d'être infaillible. Au delà du pur aspect algorithmique et mathématique, d'autres problèmes se posent avec ces objets dont notamment leur visualisation. En effet, il est compliqué de visualiser simplement de telles formes à l'aide d'un ordinateur. Il en va de même pour la solution numérique calculée dessus. Dans cette thèse, nous nous sommes donc penchés sur ces deux aspects.

Estimateurs d'erreur anisotropes pour la génération de maillages d'ordre élevé de surfaces paramétrées

La génération d'un maillage d'ordre élevé approprié à des méthodes numériques d'ordre élevé repose de manière primordiale sur la définition de la surface à l'ordre élevé. Alors que de nombreuses approches utilisent les méthodes de génération de maillage de surface linéaire avant de projeter les entités d'ordre élevé sur la frontière, nous souhaitons proposer une méthode de génération de maillage de surface qui soit directement adaptée à l'ordre élevé. En effet, l'utilisation d'une méthode linéaire peut conduire à générer des maillages qui ne correspondent pas aux contraintes de l'ordre élevé en termes de répartition des degrés de liberté. Dans le cas particulier des surfaces paramétrées, nous nous sommes donc intéressés à la généralisation d'une méthode de génération de maillage basée sur des quantités intrinsèques (c'est-à-dire indépendantes de la paramétrisation utilisée) que sont les rayons

de courbure principal et secondaire. Alors que ces quantités interviennent dans l'erreur entre une surface et son approximation par son plan tangent (c'est à dire son approximation locale linéaire), nous sommes allés plus loin dans cette analyse. Toujours en restant intrinsèque, nous avons cette fois-ci analysé l'erreur entre une surface et une approximation locale à l'ordre n où n est le degré polynomial du maillage de surface souhaité. À partir de cela, il a été possible d'extraire deux quantités intrinsèques qui peuvent s'interpréter comme l'équivalent des deux rayons de courbure au degré n. À partir de ces nouveaux estimateurs d'erreur, il devient alors possible d'effectuer la génération de maillages d'ordre élevé pour les surfaces paramétrées.

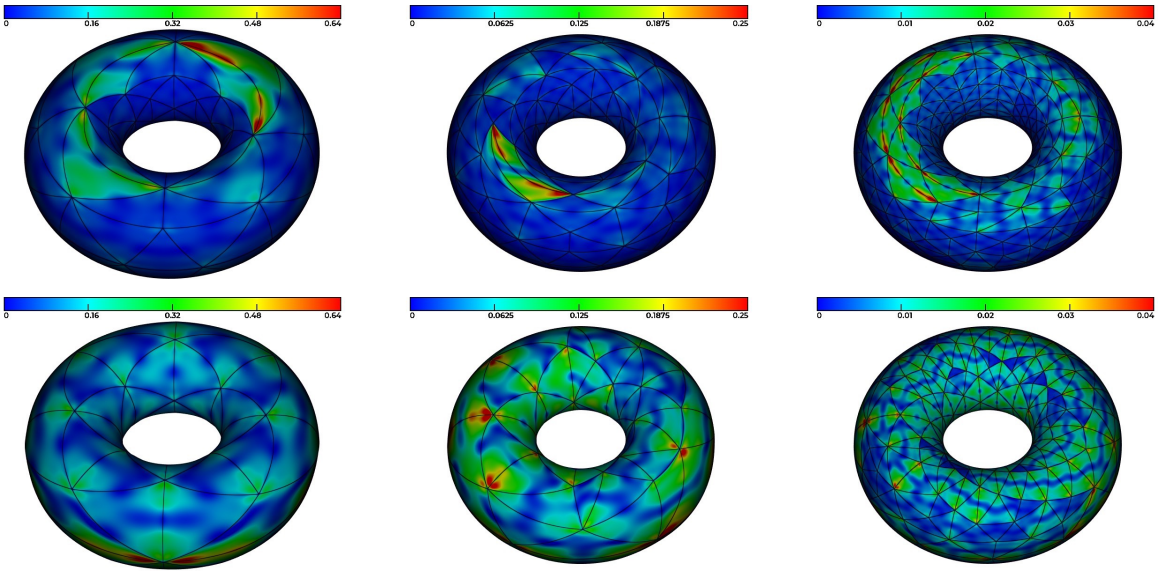

FIGURE 17 – Maillages quadratiques générés suivant une approche classique (haut) et une approche dédiée (bas) pour différents niveaux d'erreur : 0,5 (gauche) ; 0,2 (milieu) ; 0,03 (droite).

Erreur	DDL approche classique	DDL approche dédiée quadratique
0.5	347	156
0.2	687	242
0.03	1556	1036

TABLE 1 – Comparaison en termes de nombre de degrés de liberté requis pour obtenir une erreur donnée pour un maillage quadratique entre l'approche classique et l'approche dédiée.

Erreur	DDL approche classique	DDL approche dédiée cubique
0.1	255	238
0.002	6171	1848
0.0003	11329	5362

TABLE 2 – Comparaison en termes de nombre de degrés de liberté requis pour obtenir une erreur donnée pour un maillage cubique entre l'approche classique et l'approche dédiée.

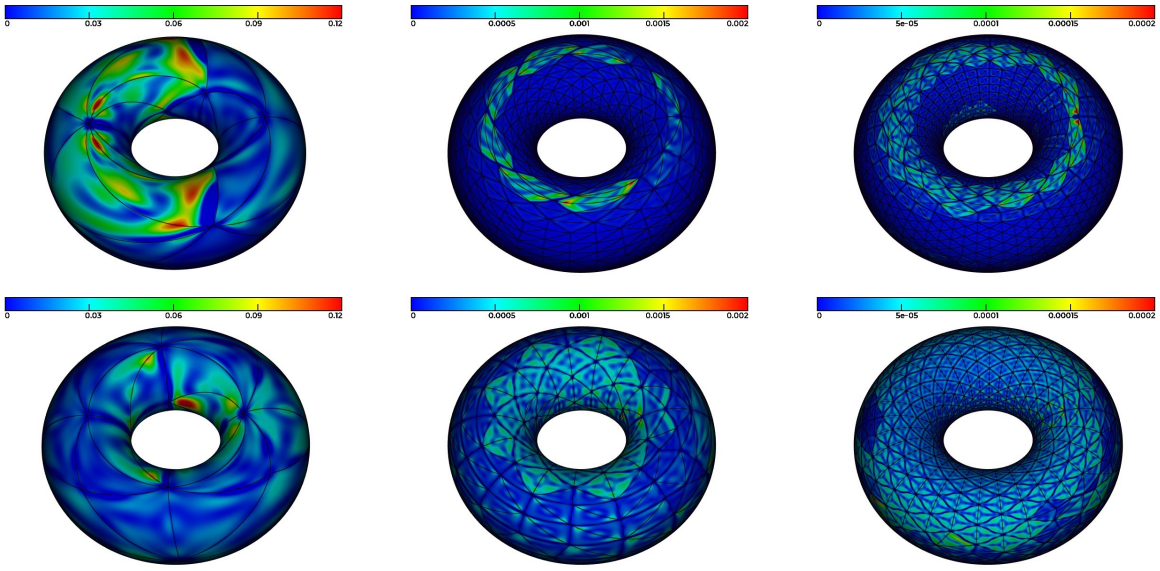

FIGURE 18 – Maillages cubiques générés suivant une approche classique (haut) et une approche dédiée (bas) pour différents niveaux d'erreur : 0,1 (gauche) ; 0,002 (milieu) ; 0,0003 (droite).

Pour illustrer cette approche, on se penche sur le cas académique du tore. Sur la Figure 17, sont présentés des maillages quadratiques et sur la Figure 18 des maillages cubiques. En haut des deux Figures, se trouvent des maillages d'ordre élevé générés suivant une approche classique, à savoir l'approche basée sur les rayons de courbure. Les nœuds d'ordre élevé sont ensuite projetés depuis le maillage droit sur la géométrie. En bas des mêmes Figures, se trouvent des maillages d'ordre élevé générés suivant une approche dédiée, c'est-à-dire une approche basée sur les rayons de courbure du degré voulu. Sur chacun de ces maillages, est représentée l'erreur ponctuelle du maillage, c'est-à-dire la distance ponctuelle entre le maillage et la géométrie réelle de l'objet. L'erreur du maillage de surface est alors définie comme la plus grande de ces erreurs ponctuelles. Pour comparer les deux méthodes, deux maillages avec la même erreur sont montrés et ce pour diverses erreurs. On observe alors que l'approche dédiée répartit mieux l'erreur, ce qui permet de générer moins d'éléments (voir Tables 1 et 2) qu'un maillage d'erreur équivalente généré suivant l'approche classique, cette dernière ne proposant pas une distribution de points pertinente.

Optimisation de maillage quadratique et applications

Une fois que le maillage de surface courbe est obtenu, il s'agit de s'occuper du maillage de volume. La stratégie communément admise pour générer un maillage d'ordre élevé consiste dans un premier temps à générer un maillage linéaire, puis à imposer le caractère courbe (*i.e.* la déformation induite par l'élévation de degré du maillage de surface) sur la frontière pour enfin propager cette information dans le volume, si nécessaire. Pour effectuer cette propagation, deux types d'approches sont envisageables : une approche globale (on résout le problème de déformation sur tout le maillage, et de manière simultanée) et une approche locale (on propage la déformation de voisin en voisin depuis le bord, tant que nécessaire). Dans l'optique d'une approche locale, nous nous intéressons dans le cas particulier des maillages quadratiques (de degré deux) à la généralisation de deux opérateurs classiques de modification locale de maillage linéaire : le bougé de point (*vertex smoothing*) et la bascule d'arête généralisée (*generalized edge swapping*). Cette généralisation a été effectuée en 2D ainsi qu'en 3D. Par exemple, la généralisation proposée pour le bougé de

point est donnée Figure 19. On voit notamment, que la généralisation de tels opérateurs au degré deux implique de prendre en compte la présence de degrés de liberté supplémentaires sur les arêtes. Il en va de même pour l'autre opérateur.

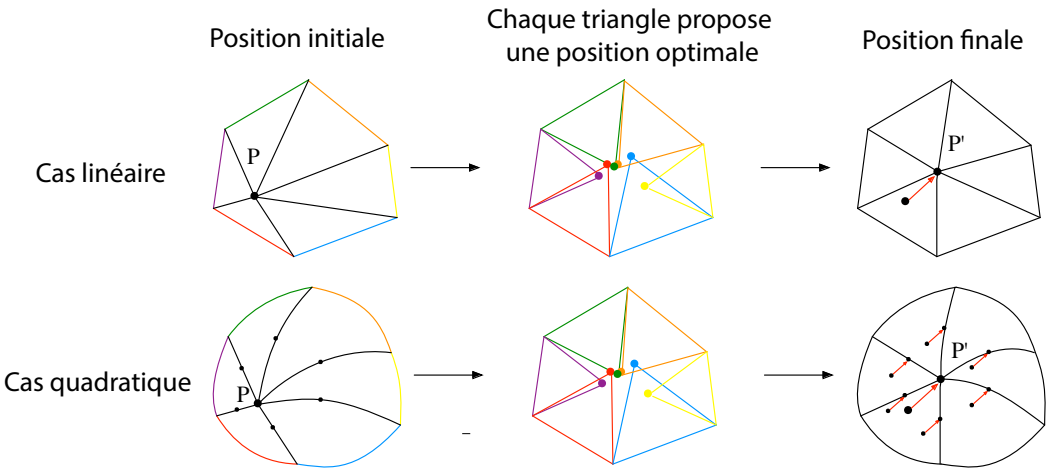

FIGURE 19 – Généralisation du bougé de point pour des maillages quadratiques.

Différentes applications découlant de l'usage de ces opérateurs peuvent alors être envisagées. La première application consiste à s'intéresser à la génération de maillages quadratiques. Dans un premier temps, à partir d'un maillage linéaire et étant donné une déformation de la frontière, un modèle de déformation globale est appliqué en utilisant une analogie avec la mécanique du solide. Cependant, cette approche manque de robustesse et l'usage d'opérateurs d'optimisation de maillage permet de pallier à ce manque. Ce procédé est illustré sur la Figure 20.

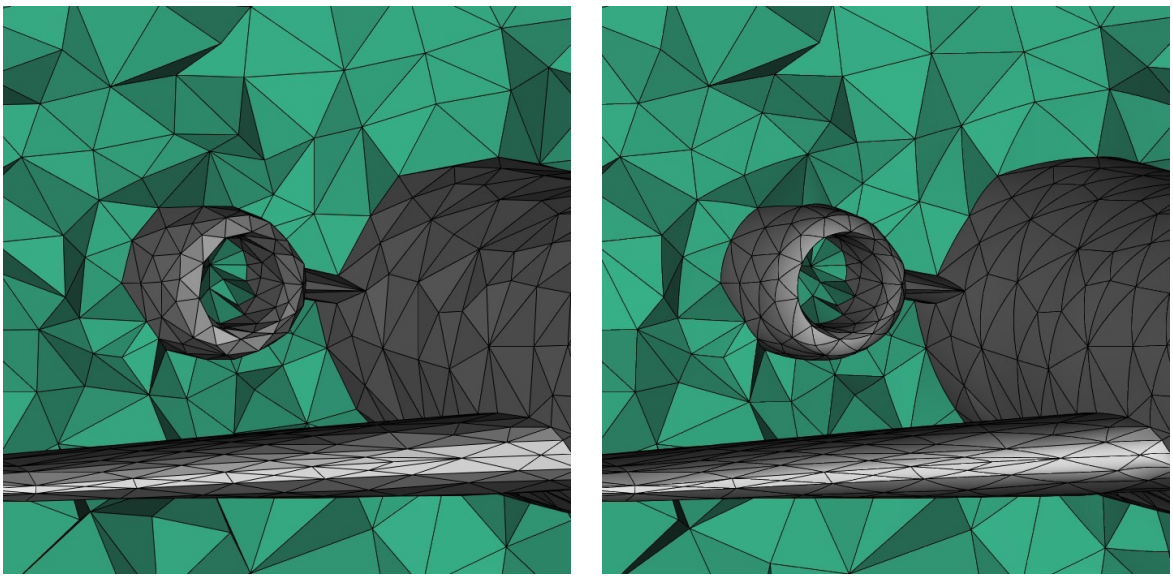

FIGURE 20 – Exemple de génération d'un maillage quadratique dans le cas d'un avion de type Falcon.

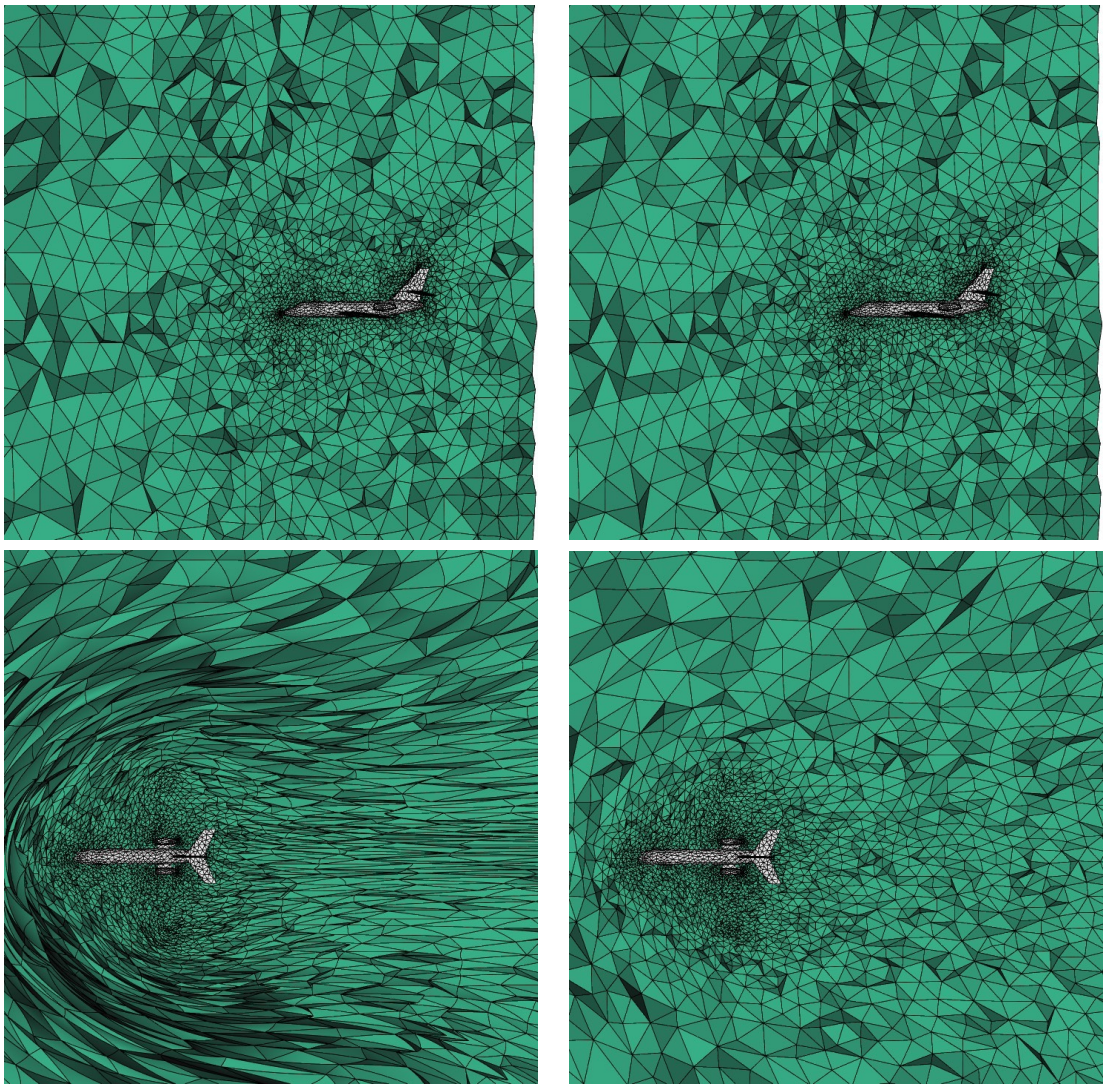

FIGURE 21 – Cas d'un avion mobile dans un maillage de tétraèdres de degré deux. En haut, la position initiale. En bas, la position finale. À gauche, le maillage obtenu en l'absence d'optimisation. À droite, le maillage obtenu avec l'usage de l'optimisation.

Une deuxième application consiste à se pencher sur le problème du maillage mobile. En effet, pour des besoins de simulation numérique, il est nécessaire de s'intéresser à la problématique d'un objet mobile au sein d'un maillage. Pour le cas linéaire, ce problème est traité à l'aide d'opérateurs d'optimisation locale. C'est donc tout naturellement que la version quadratique de ces opérateurs intervient dans le cas du maillage quadratique mobile. L'utilité de tels opérateurs est montrée sur la Figure 21 où l'absence de leur utilisation mène à un maillage totalement difforme qui n'est pas exploitable pour du calcul numérique alors que leur emploi évite justement de tels travers.
Finalement, une dernière application est la génération de maillage de couche limite. En effet, une stratégie standard pour générer de tels objets repose sur un algorithme de maillage mobile. En utilisant la stratégie de maillage mobile développée pour des maillages quadratiques, il devient possible de traiter la génération de maillages quadratiques de couche limite. Un exemple de tels maillages est donné sur la Figure 22. On observe notamment que la couche limite garde le caractère courbe de la frontière, ce qui est une propriété souhaitée vu que l'on veut obtenir un maillage structuré et aligné avec le bord.

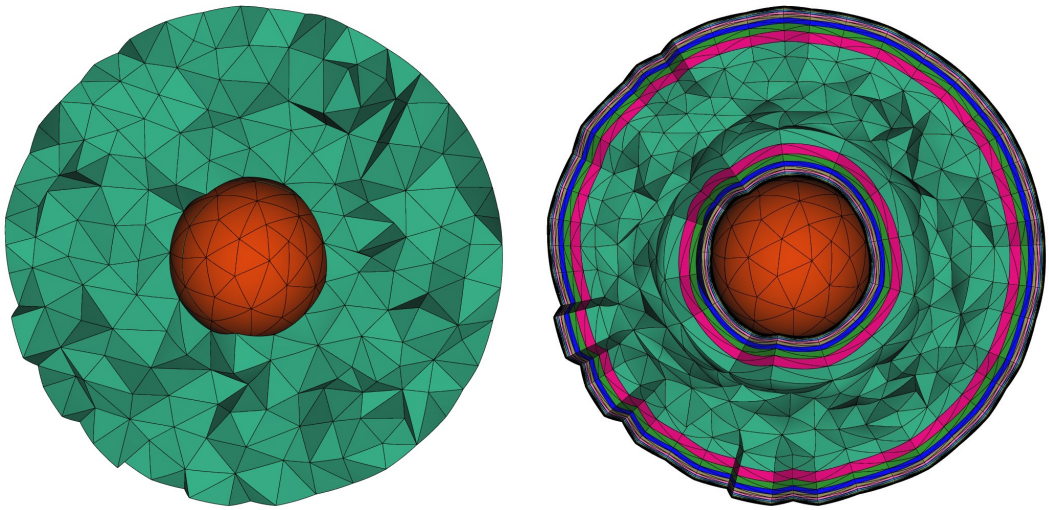

FIGURE 22 – Maillage quadratique autour d'une sphère. À gauche, maillage initial. À droite, maillage avec une couche limite quadratique.

Visualisation de maillages et de solutions d'ordre élevé

L'utilisation et le développement de techniques relatives aux maillages d'ordre élevé et aux solutions d'ordre élevé associées requièrent aussi de les visualiser de manière fidèle. Cependant au contraire des approches linéaires où la visualisation est assez naturelle pour les architectures matérielles, cela s'avère plus compliqué pour les entités d'ordre élevé. C'est à cette problématique que nous nous sommes intéressés. En prenant parti des dernières fonctionnalités de l'interface de programmation graphique `OpenGL` 4.0, nous proposons une nouvelle méthode permettant d'afficher de manière rapide une bonne représentation d'éléments de degré élevé ainsi qu'une approche permettant d'effectuer du rendu de solution d'ordre élevé de manière quasi exacte et ce, au pixel près. En effet, l'approche usuelle pour afficher des éléments d'ordre élevé repose sur une décomposition de ces éléments en un ensemble de sous-éléments linéaires, ces derniers pouvant être ensuite affichés par la carte graphique. La nouveauté réside dans la manière dont cette subdivision est faite. Alors que les approches standards génèrent la subdivision à l'aide des processeurs de calcul, nous prenons parti des opportunités offertes par l'`OpenGL` pour effectuer la subdivision à la volée sur la carte graphique. Cela permet notamment un rendu plus rapide (les communications entre un processeur et sa carte graphique étant lentes par rapport au temps d'éxécution d'un calcul) et moins gourmand en mémoire vive. Enfin, toujours grâce aux fonctionnalités de l'`OpenGL`, il est possible de contrôler ce qu'il se passe au niveau des pixels correspondants aux géométries qui sont visualisées. Ce faisant, il est possible d'avoir accès aux paramètres géométriques utiles pour définir une solution dessus. L'obtention de ces paramètres est d'autant précise que le rendu approché de la géométrie est proche de la vraie géométrie (autrement dit, il est exact pour des géométries linéaires). Ainsi, à l'aide de ces paramètres, il est possible d'avoir une valeur de la solution pour chaque pixel et d'obtenir ainsi un rendu de solution d'ordre élevé quasiment exact au pixel près (et avec la certitude d'être exact sur les géométries linéaires). Cette approche diffère des approches classiques qui elles se basent plutôt sur une subdivision des éléments, y compris linéaires, pour pouvoir afficher une solution d'ordre élevé dessus, le rendu de cette dernière étant alors l'union du rendu des solutions linéaires définies sur chaque sous-élément de cette subdivision. Un exemple de rendu d'éléments d'ordre élevé et de solutions d'ordre élevé est proposé Figure 23.

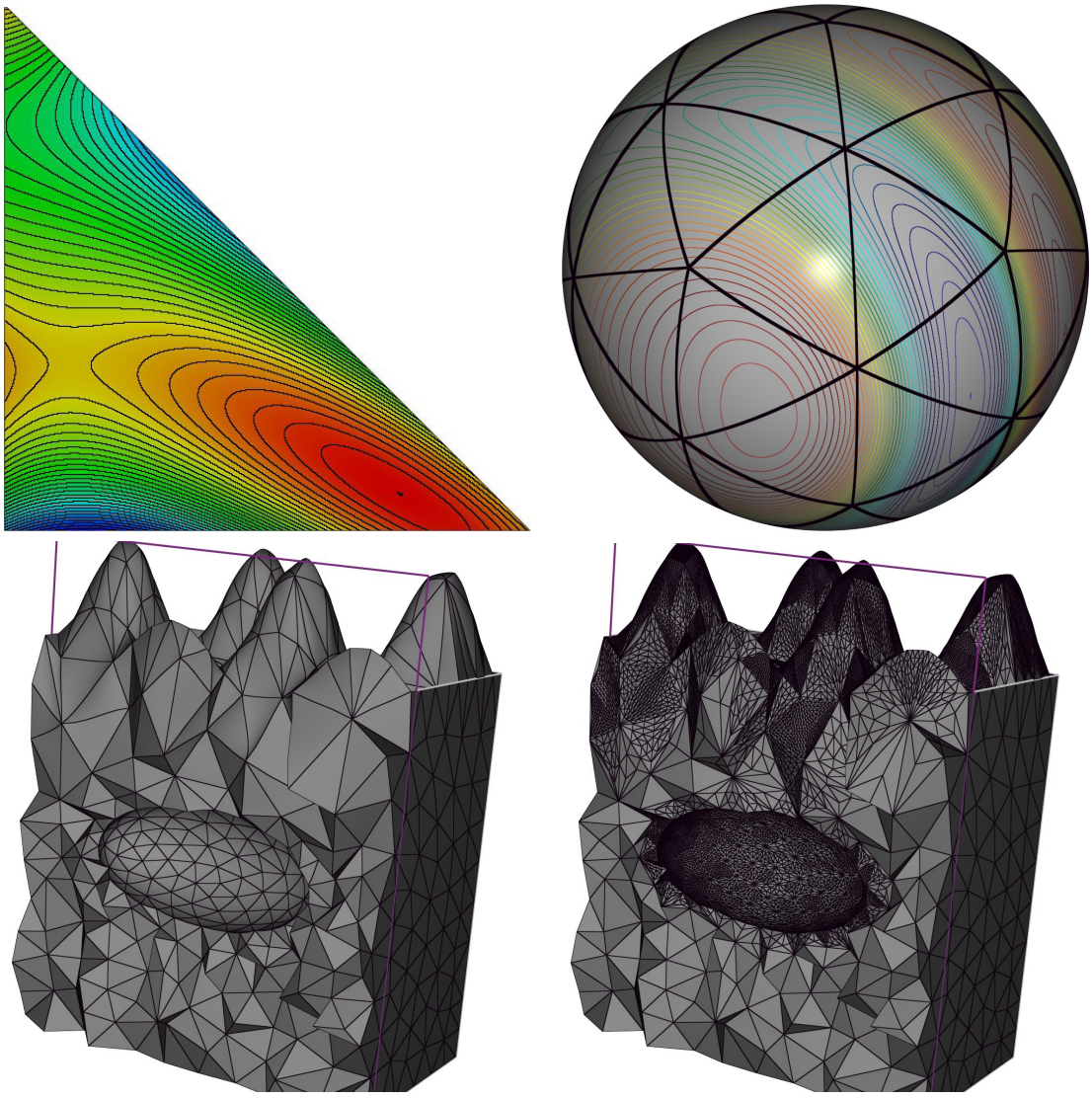

FIGURE 23 – En haut : À gauche : rendu pixel-exact avec isovaleurs d'une solution d'ordre élevé sur un triangle. À droite, isovaleurs d'une solution d'ordre élevé sur un maillage de surface en triangles d'ordre élevé d'une sphère. En bas : rendu d'un maillage cubique de tétraèdres. Maillage à gauche et subdivision à droite.

Part I

Mesh adaptation for the embedded boundary method in CFD

Introduction

Computational Fluid Dynamics (CFD) is the art of using numerical analysis to study the fluid mechanics and to numerically solve problems that involve fluid flows. The physical phenomena are various as steady or unsteady flow that are modeled by the Euler or Navier-Stokes equations with applications to Fluid-Structure Interaction or Shape Optimization. While most of CFD techniques use a discretization mesh (or grid) to solve these physics, the choice of the nature of the discretization mesh remains an open question in the community. Structured, unstructured, isotropic, anisotropic, simplicial, non-simplicial or hybrid, ... the diversity of the used meshes is as big as the number of numerical schemes applied on it. Finally, a last question remains: how do we represent the object ? Three approaches mainly exist: the first one known as the body-fitted approach consists in exactly representing the object in the mesh. The second approach is the Chimera (or overset) method [Benek et al. 1985], in which the studied body has it own body-fitted sub-mesh, and the sub-mesh overlaps the global computational mesh. Finally, the last one is the embedded (or immersed) approach [Peskin 1972] where the object is implicitly represented (using a level-set function or a (discrete) CAD geometry) in the computational mesh.

In this part, we consider the last approach. The main interest is its ability perform numerical simulations on very complex objects without requiring as input a conformal mesh (*e.g.* a water-tight and non self-intersecting mesh) of the geometry. Actually, it would be almost impossible to obtain this mesh without manual intervention. This specific type of simulation is of particular interest for simulating shape-varying geometries like veins or aortic valves [Quan et al. 2014a, Quan et al. 2014b], really complex geometries like helicopters or Formula One cars [Billon 2016], or finally multi components bodies like in fragmentation cases [Löhner et al. 2004, Löhner et al. 2007b, Löhner et al. 2007a, Puscas et al. 2015]. Another advantage relies in the simplicity of the generated volume mesh. As the constraint of the inner object is no longer there, the meshing of the volume part is not an issue and can be easily done, even in a structured fashion. However, following the law that there is constant balance between the work required in the solver and the mesh generation, it implies that the solver has more constraints to follow. Indeed, the solver has to be able to detect the boundary of the object, this boundary being represented by a set of edges or triangles, a CAD model or a level-set function, and to determine, if applicable, the inner and the outer part of the object. Then, the next step consists in applying the appropriate treatment to the elements of the mesh where it is intersected [Wang et al. 2011, Löhner 2008, Fidkowski and Darmofal 2007a]. This step is the most crucial part of the process to get proper boundary conditions. Note that this approach has been developed for both compressible and incompressible Euler/Navier-Stokes discretized with either the Finite Volume Method [Yang et al. 1997a, Yang et al. 1997b], the Finite Element Method [Löhner et al. 2004] or a Discontinuous Galerkin Method [Fidkowski and Darmofal 2007a].

In the case of simplicial (*e.g.* triangular and tetrahedral) meshing, the resolution of the flow solver is usually coupled with mesh adaptation. There exists mainly three type of adaptation: *r-adaptation* which consists in moving the vertices [Huang et al. 2010], *p-adaptation* which consists in locally enriching the functional space in the context of Finite Element Methods [Babuska et al. 1981] and finally *h-adaptation* which consists in locally

refining the discretization grid in a non-conformal [Berger and Oliger 1984] or conformal way [Löhner and Baum 1992]. The last approach appears to be interesting as it does not require to change the used solver (which, in our case, requires a conformal mesh) while highly refining the mesh if needed. In the context of body-fitted simulations, mesh adaptation has been successfully applied to various cases like Euler equations in the case of both steady [Vallet 1992, Loseille 2008] and unsteady flows [Alauzet 2003, Dobrzynski 2005] with the possibility of moving geometries [Olivier 2011, Barral 2015, Gauci 2018] and Fluid-Structure Interaction [Vanharen et al. 2018]. These methods have been extended for the steady case to the Navier-Stokes equations [Menier 2015, Frazza 2018] in the context of RANS simulations. Of course, other types of application can be done using mesh adaptation like acoustics [Chaillat et al. 2018], electromagnetics [Borouchaki et al. 2010], magnetohydrodynamics (MHD) [Amari et al. 2018] or solid mechanics [Borouchaki et al. 2005] among many others. The coupling of mesh adaptation with the embedded boundary method has also been tackled in the case of Navier-Stokes equations [Quan et al. 2014a, Quan et al. 2014b, Abgrall et al. 2014a] and more generally for non-simplicial meshes, the method is usually coupled with adaptive mesh refinement on cartesian grids [Vanella et al. 2010].

In this part, I will present the use of anisotropic mesh adaptation for the embedded boundary method applied to the Euler Equations. To this end, the first chapter deals with the basic principles used in mesh adaptation for the Euler equations and details the methodology used for the resolution of the Euler equations in `Wolf`. Then, the second chapter explains the approach used to implement the Embedded Boundary Method in the already existing resolution framework and shows some results that validate the implementation of the approach. Finally, the coupling between the method and mesh adaptation is performed.

Chapter 1

Mesh adaptation for the Euler equations

1.1 Introduction

The purpose of this chapter is to deal with mesh adaptation for Euler equations that govern adiabatic and inviscid flow. In the case of compressible flow, the Euler equations are conveniently solved by the Finite Volume Method. In the following, the implementation of the Finite Volume Method in the solver `Wolf` [Alauzet 2019] is presented. It consists in a vertex-centered finite volume formulation where the finite volume cells are implicitly built on unstructured meshes. Second-order space accuracy is achieved through a piecewise-linear extrapolation based on the Monotonic Upwind Scheme for Conservation Law (MUSCL) procedure with a particular edge-based formulation. Both explicit and implicit approaches are available for the time integration. During implicit simulations, a linearized system is solved at each solver iteration using a Lower-Upper Symmetric Gauss-Seidel (LU-SGS) approach.
The general idea of anisotropic mesh adaptation is to modify the discretization of the computational domain so that it minimizes the error induced by the discretization. Some regions of the mesh are refined while others are coarsened, and stretched mesh elements are generated to follow the natural anisotropy of the physical phenomena. In general, the approximation error $(u - u_h)$ is decomposed into two types of error: (i) the interpolation error $(u - \Pi_h u)$, (ii) the implicit error $(\Pi_h u - u_h)$, where u is the exact solution, u_h the numerical solution provided by the flow solver and $\Pi_h u$ the linear interpolate of u on the discretization. In this chapter, we focus on controlling the interpolation error. It is used as starting point for the construction of proper size maps for the mesh where these sizes are based on the hessian of the solution. Then, the mesh adaptation is performed using the anisotropic local remesher `Feflo.a/AMG` [Loseille and Löhner 2013].

1.2 The Finite Volume method for the Euler equations

Several methods exist to solve the Euler equations. In this section, we detail a vertex-centered Finite Volume Method. This type of numerical scheme is specifically tailored for the resolution of hyperbolic systems to whom the Euler equations are part of.

1.2.1 The Euler equations

The Euler equations in $\mathbb{R}^d_{(d=2,3)}$ read:

$$\begin{cases} \dfrac{\partial}{\partial t} W + \nabla \cdot (\mathbf{F}(W)) = 0, \quad \forall (\mathbf{x}, t) \in \Omega \times \mathbb{R}^+, \\ W(\mathbf{x}, 0) = W_0(\mathbf{x}), \\ W(\mathbf{x}, 0) = W_\infty(\mathbf{x}, t), \quad \forall (\mathbf{x}, t) \in \partial\Omega \times \mathbb{R}^+. \end{cases}$$

Ω is an open bounded domain $\mathbb{R}^d$ of boundary $\partial\Omega = \Gamma$, W is a vector function of $\mathbb{R}^{d+2}$ and $\mathbf{F}$ is an element $\mathbb{R}^{d+2} \times \mathbb{R}^{2d}$.
W is the state variable and is defined by:

$$W = \begin{pmatrix} \rho \\ \rho\mathbf{U} \\ \rho E \end{pmatrix},$$

where ρ denotes the density, $\mathbf{U} \in \mathbb{R}^d$ the fluid velocity and E the total energy.
$\mathbf{F}(W) = (F_i(W))_{1\leq i\leq d}$ with:

$$F_i(W) = \begin{pmatrix} \rho u_i \\ \rho u_i \mathbf{U} + p\delta_i \\ (\rho E + p)u_i \end{pmatrix},$$

where δ_i denotes the i^{th} column vector of the identity matrix I_d and p the pressure, also satisfying the perfect gas equation:

$$p = (\gamma - 1)(\rho T),$$

where γ is the ratio of the specific heats and is supposed constant. Note that following the International System of units, the pressure p is in Pascals (Pa), the velocity $\mathbf{u}$ in meters per second ($\mathrm{m.s}^{-1}$), the density ρ in kilograms per square ($d = 2$) or cube ($d = 3$) meters ($\mathrm{kg.m}^{-d}$) and the energy E is in Joules (J).

1.2.2 Spatial discretization

The spatial discretization of the Euler equations relies on the Finite Volume method (a more complete survey of this method can be found in [Menier 2015, Frazza 2018], notably for the case of RANS equations).
Let $\mathcal{H}$ be a mesh of domain Ω, the vertex-centered finite volume formulation consists in associating with each vertex P_i of the mesh a control volume or finite volume cell C_i (Figure 1.1).

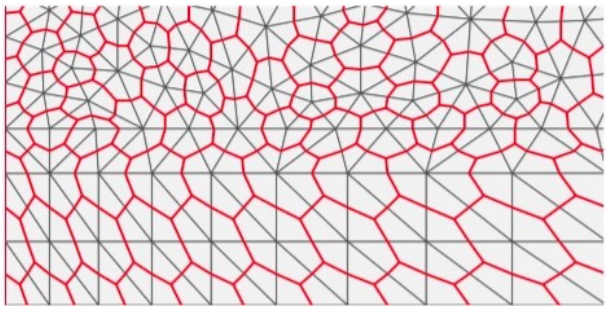

Figure 1.1 – Median Finite Volume cells.

Discretized domain Ω_h can be written as:

$$\Omega_h = \bigcup_{i=1}^{N_K} K_i = \bigcup_{i=1}^{N_V} C_i,$$

where N_K is the number of elements and N_V the number of vertices. Note that unlike a cell-centered approach, the finite volume cells are computed on the fly and never stored. The only stored data is the input simplicial mesh.
The integration of the Euler equation for each cell C_i (using Green Formula), gives:

$$|C_i|\frac{\mathrm{d}W_i}{\mathrm{d}t} + \mathbf{F}_i = 0,$$

where W_i is the mean value of W on cell C_i and $\mathbf{F}_i$ is the numerical convective flux *e.g.*:

$$\mathbf{F}_i = \int_{\partial C_i} \mathbf{F}(W_i).\mathbf{n}_i \mathrm{d}\gamma,$$

where $\mathbf{n}_i$ is the outer normal to the finite volume cell surface ∂C_i.

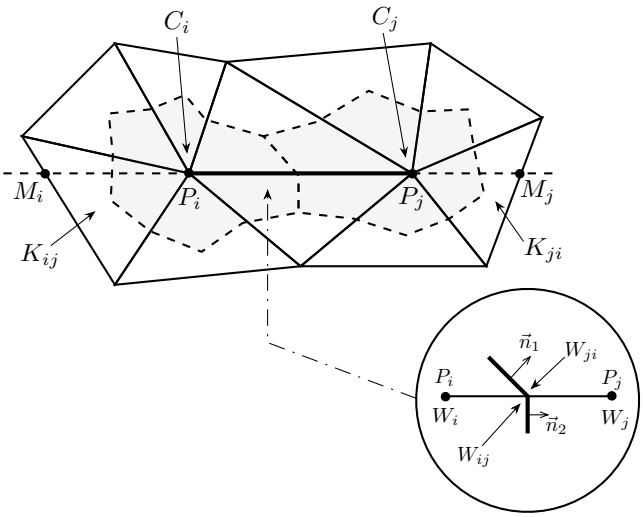

Figure 1.2 – Construction of a cell and interface between two cells.

Now, if we note $\partial C_{i,j} = \partial C_i \cup \partial C_j$ (cf Figure 1.2), the integration of the numerical flux writes:

$$\mathbf{F}_i = \sum_{P_j \in \nu(P_i)} F_{|\partial C_{i,j}} \cdot \int_{\partial C_{i,j}} \mathbf{n}_i \mathrm{d}\gamma, \tag{1.1}$$

where $\nu(P_i)$ is the set of all neighboring vertices linked by an edge to P_i and $F_{|\partial C_{i,j}}$ represents the constant value of $F(W)$ at the interface $\partial C_{i,j}$. It is then computed using a numerical flux function, denoted by Φ_{ij}:

$$\Phi_{ij} = \Phi_{ij}(W_i, W_j, \mathbf{n}_{ij}) = F_{|\partial C_{i,j}} \cdot \int_{\partial C_{i,j}} \mathbf{n}_i \mathrm{d}\gamma, \tag{1.2}$$

where $\mathbf{n}_{ij} = \int_{\partial C_{i,j}} \mathbf{n}_i \mathrm{d}\gamma$. The purpose of approximated Riemann solvers is to compute this term. We employ the HLLC approximate Riemann solver [Batten et al. 1997].

1.2.3 HLLC approximate Riemann solver

The concept behind the HLLC flow solver [Batten et al. 1997] is to locally consider an approximated Riemann problem with two intermediate states depending on the local left and right states, Figure 1.3 .

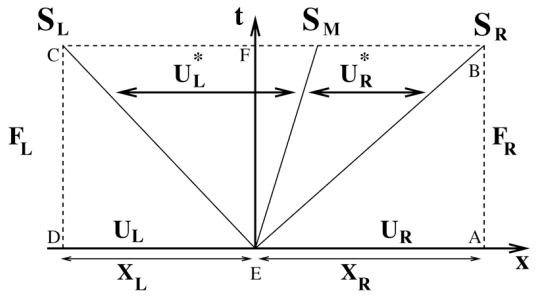

Figure 1.3 – Simplified Riemann problem used for the computation of HLLC flux.

The simplified solution to the Riemann problem consists of a contact wave with a velocity S_M and two acoustic waves, which may be either shocks or expansion fans. The acoustic waves have the smallest and the largest velocities (S_L and S_R respectively) of all the waves present in the exact solution. if $S_L > 0$, then the flow is supersonic from left to right and the upwind flux is simply defined from $F(W_l)$ where W_l is the state to the left of the discontinuity. Similarly, if $S_R < 0$, then the flow is supersonic from right to left and the flux is defined form $F(W_r)$ where W_r is the state to the right of the discontinuity. In the more difficult subsonic case when $S_L < 0 < S_R$, we have to compute $F(W_l^*)$ or $F(W_r^*)$. Consequently, the HLLC flux is given by :

$$\Phi_{lr}^{hllc}(W_l, W_r, \mathbf{n}_{lr}) = \begin{cases} F(W_l).\mathbf{n}_{lr} & \text{if} \quad S_L > 0, \\ F(W_l^*).\mathbf{n}_{lr} & \text{if} \quad S_L \leq 0 < S_M, \\ F(W_r^*).\mathbf{n}_{lr} & \text{if} \quad S_M \leq 0 \leq S_R, \\ F(W_r).\mathbf{n}_{lr} & \text{if} \quad S_R < 0. \end{cases}$$

W_l^* and W_r^* are evaluated as follows. We denote by $\eta = \mathbf{u}\cdot\mathbf{n}$. Assuming (for simplification) that $\eta^* = \eta_l^* = \eta_r^* = S_M$, the following evaluations are proposed (the subscript l or r are omitted for clarity):

$$W^* = \frac{1}{S - S_M} \begin{pmatrix} \rho(S-\eta) \\ \rho\mathbf{u}(S-\eta) + (p^* - p)\mathbf{n} \\ \rho E(S-\eta) + p^* S_M - p\eta \end{pmatrix}, \qquad \text{with} \quad p^* = \rho(S-\eta)(S_M - \eta) + p.$$

A key feature of this solver is in the definition of the three waves velocity. For the contact wave, we consider

$$\begin{aligned} S_M &= \frac{\rho_r\eta_r(S_R - \eta_r)\rho_l\eta_l(S_L - \eta_l) + p_l - p_r}{\rho_r(S_R - \eta_r) - \rho_l(S_L - \eta_l)}, \\ S_L &= \min(\eta_l - c_l, \tilde{\eta} - \tilde{c}), \\ S_R &= \min(\eta_r + c_r, \tilde{\eta} + \tilde{c}), \end{aligned}$$

where $\sim$ denotes Roe average values [Batten et al. 1997]. With such waves velocities, the approximate HLLC Riemann solver has the following properties : (i) it satisfies the entropy inequality, (ii) it resolves isolated contacts exactly, (iii) it resolves isolated shocks exactly and (iv) it preserves positivity.

1.2.4 Second order accurate version

The Finite Volume Method as is, is a first order scheme. The MUSCL type reconstruction scheme [Van Leer 1979] has been designed to increase the order of accuracy of the

scheme. An extension of this approach to unstructured meshes has then been proposed in [Debiez and Dervieux 2000].

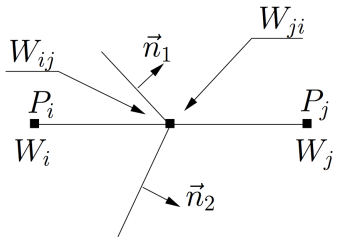

Figure 1.4 – Extrapolation used for the MUSCL scheme.

To this end, we use extrapolated values W_{ij} and W_{ji} (Fig.1.4) instead of W_i and W_j at the interface $\partial C_{i,j}$ to evaluate the flux. The numerical flux becomes:

$$\Phi_{ij} = \Phi_{ij}(W_{ij}, W_{ji}, \mathbf{n}_{ij}),$$

with :

$$W_{ij} = W_i + \frac{1}{2}(\nabla W)_{ij} \cdot \overrightarrow{P_iP_j} \quad \text{and} \quad W_{ji} = W_j + \frac{1}{2}(\nabla W)_{ji} \cdot \overrightarrow{P_jP_i}. \tag{1.3}$$

The computation of gradients $(\nabla W)_{ij}$ and $(\nabla W)_{ji}$ is obtained using a combination of centered, upwind and nodal gradients.
The centered gradient is implicitly given by the relation:

$$(\nabla W)^C_{ij} \cdot \overrightarrow{P_iP_j} = W_j - W_i,$$

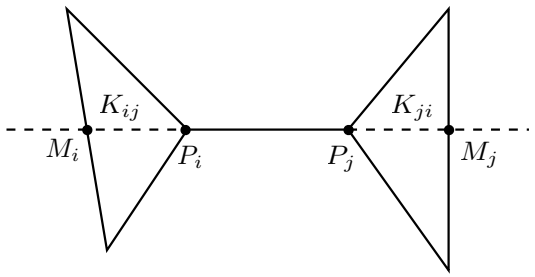

Figure 1.5 – Upwind and downwind elements used for the MUSCL scheme.

Upwind and downwind gradients are computed according to the definition of upwind (K_{ij}) and downwind elements(K_{ji}), (Fig.1.5). These elements are the unique simplices of the ball of P_i (resp. P_j) whose opposite facet to P_i (resp. P_j) is crossed by the line defined by the edge P_iP_j and that and do not contain P_iP_j.
Upwind and downwind gradients are then defines for vertices P_i and P_j as:

$$(\nabla W)^U_{ij} = (\nabla W)|_{K_{ij}} \quad \text{and} \quad (\nabla W)^D_{ij} = (\nabla W)|_{K_{ji}}$$

where $(\nabla W)|_K = \sum_{P\in K} W_P \nabla \phi_P|_K$ is the P^1-Galerkin gradient on the element K. Parameterized edge gradients are built by introducing the β-scheme:

$$(\nabla W)_{ij} \cdot \overrightarrow{P_iP_j} = (1-\beta)(\nabla W)^C_{ij} \cdot \overrightarrow{P_iP_j} + \beta(\nabla W)^U_{ij} \cdot \overrightarrow{P_iP_j}$$
$$(\nabla W)_{ji} \cdot \overrightarrow{P_jP_i} = (1-\beta)(\nabla W)^C_{ij} \cdot \overrightarrow{P_jP_i} + \beta(\nabla W)^D_{ij} \cdot \overrightarrow{P_jP_i}$$

where $\beta \in [0,1]$ is a parameter controlling the amount of upwinding. If $\beta = 0$, the scheme is centered and if $\beta = 1$ the scheme is fully upwind.
The most accurate β-scheme is obtained for $\beta = \frac{1}{3}$. On unstructured meshes, a second-order scheme with a fourth-order numerical dissipation is obtained. This scheme is denoted as the **V4-scheme**.
It is in fact possible to get an even less dissipative scheme. This scheme is denoted as the **V6-scheme**. It is a more complex linear combination of gradients using centered upwind and nodal gradients. The nodal P^1-Galerkin gradient of P_i is related to cell C_i and is computed by averaging the gradients of all the simplices containing the vertex P_i:

$$(\nabla W)_{P_i} = \frac{1}{(d+1)|C_i|} \sum_{K \in C_i} |K| (\nabla W)_K$$

A sixth-order dissipation scheme is then obtained by considering the following high-order gradient:

$$\begin{aligned}
(\nabla W)^{V6}_{ij} \cdot \overrightarrow{P_iP_j} = [(\nabla W)^{V4}_{ij} \; - \; & \frac{1}{30}\left((\nabla W)^{U}_{ij} - 2(\nabla W)^{C}_{ij} + (\nabla W)^{D}_{ij}\right) \\
& - \frac{2}{15}\left((\nabla W)_{M_i} - 2(\nabla W)_{P_i} + (\nabla W)_{P_j}\right)] \cdot \overrightarrow{P_iP_j} \\
(\nabla W)^{V6}_{ji} \cdot \overrightarrow{P_jP_i} = [(\nabla W)^{V4}_{ji} \; - \; & \frac{1}{30}\left((\nabla W)^{D}_{ij} - 2(\nabla W)^{C}_{ij} + (\nabla W)^{U}_{ij}\right) \\
& - \frac{2}{15}\left((\nabla W)_{M_j} - 2(\nabla W)_{P_j} + (\nabla W)_{P_i}\right)] \cdot \overrightarrow{P_jP_i}
\end{aligned} \tag{1.4}$$

where $(\nabla W)_{M_{i,j}}$ is the gradients at the points $M_{i,j}$, intersection of the line defined by (P_iP_j) and upwind/downwind elements (see Fig. 1.5). These gradients are computed by linear interpolation of the nodal gradients of the facets containing M_i and M_j.

Limiter function. MUSCL schemes are not monotone and can be a source of spurious oscillations. These oscillations can affect the accuracy of the final solution and its precision as it can crash the code by creating negative densities or pressures. One of the major concerns is to guarantee the LED (*Local Extremum Diminishing*) property of the scheme which ensures that the extrapolated values make sense. To this end, limiting functions are coupled with the previous high-order gradient evaluations. The gradient is substituted by a limited gradient denoted $(\nabla W)^{Lim}$. The choice of the limiting function is crucial as it directly affects the convergence and the accuracy of the solution. In this work, two limiters are used:

- Dervieux or Koren limiter [Koren 1993]: This is a three entries limiter,

  $$(\nabla W)^{Lim}_{ij}.\overrightarrow{P_iP_j} = Lim_{DE}((\nabla W)^{C}_{ij} \cdot \overrightarrow{P_iP_j} \; , \; (\nabla W)^{D}_{ij}.\overrightarrow{P_iP_j} \; , \; (\nabla W)^{HO}_{ij} \cdot \overrightarrow{P_iP_j}),$$

  with:

  $$\begin{aligned}
  &\text{if } uv \leq 0 \text{ then} \\
  &\quad Lim_{DE}(u,v,w) = 0 \\
  &\text{else} \\
  &\quad Lim_{DE}(u,v,w) = Sign(u) min(2|u|, 2|v|, |w|).
  \end{aligned}$$

  where $(\nabla W)^{HO}_{ij}$ is either $(\nabla W)^{V4}_{ij}$ or $(\nabla W)^{V6}_{ij}$.

- Piperno Limiter [Piperno and Depeyre 1998]: This limiter is expressed in a factorized form,

  $$(\nabla W)^{Lim}_{ij}.\overrightarrow{P_iP_j} = (\nabla W)^{C}_{ij}.\overrightarrow{P_iP_j} \; Lim_{PI}\left(\frac{(\nabla W)^{C}_{ij}.\overrightarrow{P_iP_j}}{(\nabla W)^{V4}_{ij}.\overrightarrow{P_iP_j}}\right),$$

with

$$Lim_{PI}(R) = (\frac{1}{3} + \frac{2}{3}R) \begin{cases} \frac{3\frac{1}{R^2} - 6\frac{1}{R} + 19}{\frac{1}{R^3} - 3\frac{1}{R} + 18} & \text{if } R < 1 \\ 1 + (\frac{3}{2}\frac{1}{R} + 1)(\frac{1}{R} - 1)^3 & \text{if } R \geq 1 \end{cases}$$

where

$$R = \frac{(\nabla W)_{ij}^{C}.\overrightarrow{P_iP_j}}{(\nabla W)_{ij}^{V4}.\overrightarrow{P_iP_j}}.$$

The Piperno limiter can be only used with the V4 high-order gradient.

1.2.5 Boundary conditions

Euler equations are coupled with some boundary conditions. Slip boundary conditions are imposed for bodies and Steger-Warming flux is used to set-up free-stream (external flow) conditions.

Slip condition (weak form)

This boundary condition consists in weakly imposing $\mathbf{U}.\mathbf{n} = 0$ on the boundary by imposing the following flux:

$$\Phi_{Slip} = F(W) = (0, p\mathbf{n}, 0)^T. \tag{1.5}$$

Riemann slip condition

For this boundary condition, we also weakly impose: $\mathbf{U}.\mathbf{n} = 0$. To this end, the flux Φ between a state W on the boundary and the mirror state $\overline{W}$ is computed:

$$W = \begin{pmatrix} \rho \\ \rho\mathbf{U} \\ \rho E \end{pmatrix} \quad \text{et} \quad \overline{W} = \begin{pmatrix} \rho \\ \rho\mathbf{U} - 2\rho(\mathbf{U}.\mathbf{n})\mathbf{n} \\ \rho E \end{pmatrix}, \tag{1.6}$$

If the slip condition holds, then $W = \overline{W}$ and $\Phi(W, \overline{W}) = \Phi_{Slip}$. Nevertheless, state W on the boundary does not satisfy the slip condition unless it is strongly imposed. For these reasons, the flux between these two states is computed which leads to compute this flux:

$$\Phi_{Slip\ Riemann} = (0, p^*\mathbf{n}, 0)^T, \tag{1.7}$$

where $p^* = p + \rho\mathbf{U}.\mathbf{n}\min(c, \tilde{c} + \mathbf{U}.\mathbf{n})$ in the case of a HLLC flux. This condition is numerically more stable but more dissipative (and consequently less accurate) than the previous one.

Free-stream condition

This condition imposes a free-stream uniform state W_∞ at the infinity:

$$W_\infty = \begin{pmatrix} \rho_\infty \\ (\rho\mathbf{U})_\infty \\ (\rho E)_\infty \end{pmatrix}.$$

This condition is imposed via the Steger-Warming flux which is completely upwind on solution W_i:

$$\Phi_\infty = A^+(W_i, \mathbf{n}_i)W_i + A^-(W_i, \mathbf{n}_i)W_\infty,$$

where A is the jacobian matrix of F, $A^+ = \frac{|A|+A}{2}$ and $A^- = \frac{|A|-A}{2}$.

1.2.6 Time integration

Once the spatial discretization is performed, the temporal part is discretized and an advance in time is performed. Usually, two types of discretization exist for this part. An implicit and an explicit one.

Explicit time integration

For the advance in time, it is convenient to rewrite the system under the form $\frac{\mathrm{d}W}{\mathrm{d}t} - L(W) = 0$ where:

$$\frac{\mathrm{d}W}{\mathrm{d}t} = L(W) = \frac{1}{|C_i|}\mathbf{R}_i^n,$$

and

$$\mathbf{R}_i^n = \sum_{j\in\nu(i)} \Phi_{ij}^{hllc}(W_i^n, W_j^n, \mathbf{n}_{ij}) + \sum_{f_{slip}|i\in f_{slip}} \Phi_{slip}(W_i^n, n_{f_{slip}}) + \sum_{f_\infty|i\in f_\infty} \Phi_\infty(W_i^n, n_{f_\infty}).$$

The discretization of this formula is done via the high-order multi-step Runge-Kutta scheme. The Runge-Kutta scheme of time-order 2 in with 2 steps is given by:

$$\begin{aligned} W^{(1)} &= W^n + \Delta t^n L(W^n), \\ W^{n+1} &= \frac{1}{2}W^n + \frac{1}{2}W^{(1)} + \frac{1}{2}\Delta t^n L(W^{(1)}). \end{aligned}$$

This scheme has a maximal CFL number of 1.
The Runge-Kutta scheme of time-order 2 in with 5 steps is given by:

$$\begin{aligned} W^{(1)} &= W^n + \frac{1}{4}\Delta t^n L(W^n), \\ W^{(k)} &= W^{(k-1)} + \frac{1}{4}\Delta t^n L(W^{(k-1)}), \quad \text{k=2,..,4}, \\ W^{n+1} &= \frac{1}{5}W^n + \frac{4}{5}W^{(4)} + \frac{1}{5}\Delta t^n L(W^{(4)}), \end{aligned}$$

and has a maximal CFL number of 4. It enables a gain of 60% in terms of CPU performance from the previous scheme.

Implicit time integration

For an implicit time integration, we prefer to consider the following system:

$$L(W) = \frac{1}{|C_i|}\mathbf{R}_i^{n+1}.$$

Now, if we approximate $L(W)$ by $\frac{\delta W_i^{n+1}}{\Delta t_n^i} = \frac{W_i^{n+1} - W_i^n}{\Delta t_n^i}$, we obtain after linearization of the RHS:

$$\left(\frac{|C_i|}{\Delta t_i^n} I_d - \frac{\partial \mathbf{R}_i^n}{\partial W_i}\right)\delta W_i - \sum_{j\in\nu(i)} \left(\frac{\partial \mathbf{R}_i^n}{\partial W_j}\right)\delta W_j = \mathbf{R}_i^n, \tag{1.8}$$

where $j \in \nu(i)$ is the set of points P_j connected via an edge to P_i. In th case of an HLLC solver, the linearization is given by:

$$\begin{aligned}\Phi_{ij}^{HLLC}(W_i^{n+1}, W_j^{n+1}, \mathbf{n}_{ij}) = \Phi_{ij}^{HLLC}(W_i^n, W_j^n, \mathbf{n}_{ij}) \\ + \frac{\partial \Phi_{ij}^{HLLC}(W_i^n, W_j^n, \mathbf{n}_{ij})}{\partial W_i} \delta W_i \Big\} (A) \\ + \frac{\partial \Phi_{ij}^{HLLC}(W_i^n, W_j^n, \mathbf{n}_{ij})}{\partial W_j} \delta W_j. \Big\} (B)\end{aligned} \tag{1.9}$$

Term (A) contributes to matrix $D(i,i)$ and term (B) contributes to upper matrix $U(i,j)$ (it is assumed that $i < j$). As $\Phi_{ji}^{HLLC} = -\Phi_{ij}^{HLLC}$, term $-(B)$ contributes to $D(j,j)$ and term $-(A)$ to lower matrix $L(j,i)$.
This linear system is solved at each flow solver iteration using an iterative Newton method. In practice, a maximal number k_{max} of iterations of the Newton method and a targeted order of magnitude by which the residual of the system must be decreased are provided. The iteration is stopped when this targeted residual is reached. To solve the non-linear system, we follow the approach based on Symmetric Gauss-Seidel (SGS) implicit solver. Implementation details can be found in [Menier 2015, Frazza 2018].
As implicit schemes are unconditionally stable, CFL number could be arbitrarily large thus allowing large time steps. This is true if the solution is almost converged near the steady-state solution but not at the beginning. Indeed, the solution may blow up (a breakdown of the iteration process) if the CFL is too large when the flow field is not established. Therefore, to converge the Newton method, we generally start with low CFL number and its value is increased while the solution is converged. This process is ruled by CFL laws, see details in [Menier 2015, Frazza 2018].

Time-step computation

The maximal time-step for this numerical scheme at t^n is:

$$\Delta t_i^n = CFL \times \frac{h_i}{c_i^n + ||\mathbf{U}_i^n||}, \tag{1.10}$$

where h_i is the smallest height of all the elements containing P_i, $\mathbf{U}_i$ the velocity and c_i is the speed of sound at P_i, $(c_i = \sqrt{\gamma \frac{P_i}{\rho_i}})$. The global time step is found via: $\Delta t = \min_i(\Delta t_i)$. In the case of steady flow, the local time step is used to gain CPU time up to a factor 10. Each vertex goes on with its own time step and its own CFL number.

1.2.7 Aerodynamic coefficients

In this section, we define all the aerodynamic coefficients that will be used for the analysis of the solution. In particular, we will define drag, lift and pressure coefficient C_p.

On the computation of reference data

The computation of these coefficients requires three parameters:

- S_{ref} the reference surface,
- L_{ref} the reference length,
- G_{ref} the reference (real) barycenter.

If they are not provided, we have to compute them. In this case, G_{ref} is simply the barycenter of the object. For the computation of L_{ref} and S_{ref}, we denote by S_{xy} the area

of the whole object projected in the plane xy and by E_w the wingspan. In this work, we consider:

$$S_{ref} = S_{xy} \quad \text{and} \quad L_{ref} = \frac{S_{xy}}{E_w} = \frac{S_{ref}}{E_w}\,.$$

Local aerodynamic coefficient

Analyzing the flow solution on the studied geometry leads us to the visualization of some data on the wetted surface of the body. The more common coefficient is C_p given by:

$$C_p(\mathbf{x}) = \frac{p(\mathbf{x}) - p_\infty}{\frac{1}{2}\rho_\infty \|\mathbf{u}_\infty\|^2},$$

where $\mathbf{u}_\infty$ the infinite inflow velocity, $\frac{1}{2}\rho_\infty\|\mathbf{u}_\infty\|^2$ is the infinite dynamic pressure.

Drag, lift, momentum

The pressure coefficient vector for inviscid flow is then evaluated by integration of the C_p on body S:

$$\begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix} = \frac{1}{|S_{Ref}|}\int_S C_p(\mathbf{x})\mathbf{n}(\mathbf{x})\,\mathrm{d}\mathbf{x} = \frac{1}{|S_{Ref}|}\int_S \frac{p(\mathbf{x}) - p_\infty}{\frac{1}{2}\rho_\infty\|\mathbf{u}_\infty\|^2}\mathbf{n}(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

with $\mathbf{n}$ the outward normalized normal and S_{Ref} the reference surface area of body S. Finally, the aerodynamic pressure coefficient vector for inviscid flow is obtained considering the orientation of the body in space:

$$\begin{pmatrix} \text{Drag} \\ \text{Slip} \\ \text{Lift} \end{pmatrix} = Rot(\theta,\alpha,\sigma)\begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix},$$

where $Rot(\theta,\alpha,\sigma)$ is the rotation matrix defined by the angle of attack α, the angle of side slip σ and the angle of roll θ. For instance, if the span is along y and the symmetry plane is xz, the rotation matrix is:

$$Rot(\theta,\alpha,\sigma) = R^x_\theta\, R^y_\alpha\, R^z_\sigma,$$

with:

$$R^x_\theta = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}, \quad R^y_\alpha = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix} \quad \text{and} \quad R^z_\sigma = \begin{pmatrix} \cos\sigma & \sin\sigma & 0 \\ -\sin\sigma & \cos\sigma & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

We deduce:

$$Rot(\theta,\alpha,\sigma) = \begin{pmatrix} \cos(\alpha)\cos(\sigma) & \cos(\alpha)\sin(\sigma) & -\sin(\alpha) \\ \sin(\alpha)\cos(\sigma)\sin(\theta) - \sin(\sigma)\cos(\theta) & \sin(\alpha)\sin(\sigma)\sin(\theta) + \cos(\sigma)\cos(\theta) & \cos(\alpha)\sin(\theta) \\ \sin(\alpha)\cos(\sigma)\cos(\theta) - \sin(\sigma)\sin(\theta) & \sin(\alpha)\sin(\sigma)\cos(\theta) - \cos(\sigma)\sin(\theta) & \cos(\alpha)\cos(\theta) \end{pmatrix}.$$

Similarly, the moment coefficient vector is given by:

$$\begin{pmatrix} C_l \\ C_m \\ C_n \end{pmatrix} = \frac{1}{|S_{Ref}||L_{Ref}|}\int_S C_p(\mathbf{x})\,(\mathbf{g}\mathbf{x}\times\mathbf{n}(\mathbf{x}))\,\mathrm{d}\mathbf{x},$$

where $\mathbf{g}$ is the body gravity center and L_{Ref} the reference length of body S. Then, the aerodynamic moment coefficient vector is obtained considering the orientation of the body in space:

$$\begin{pmatrix} \text{Roll} \\ \text{Pitch} \\ \text{Yaw} \end{pmatrix} = Rot(\theta, \alpha, \sigma) \begin{pmatrix} C_l \\ C_m \\ C_n \end{pmatrix}.$$

1.2.8 An example

Here, we show an example that will be used as a reference problem for the next section. A body-fitted circle is considered with a uniform subsonic flow (Mach 0.1). The expected result is a potential flow that is analytically well-known. To compute this flow, an implicit HLLC solver is used along with a V6 scheme and no limiter. Two different boundary condition are employed: the slip boundary condition defined in Eq. (1.5), that gives the expected solution (Figure 1.6 left) and a Riemann slip boundary condition defined in Eq. (1.7) (see Figure 1.6 right). The result is obtained after 500 iterations of an implicit advance in time and both of them got a residual of magnitude $1.e-6$. It can be noted that the solution obtained with Riemann slip boundary condition is less accurate as it adds numerical dissipation.

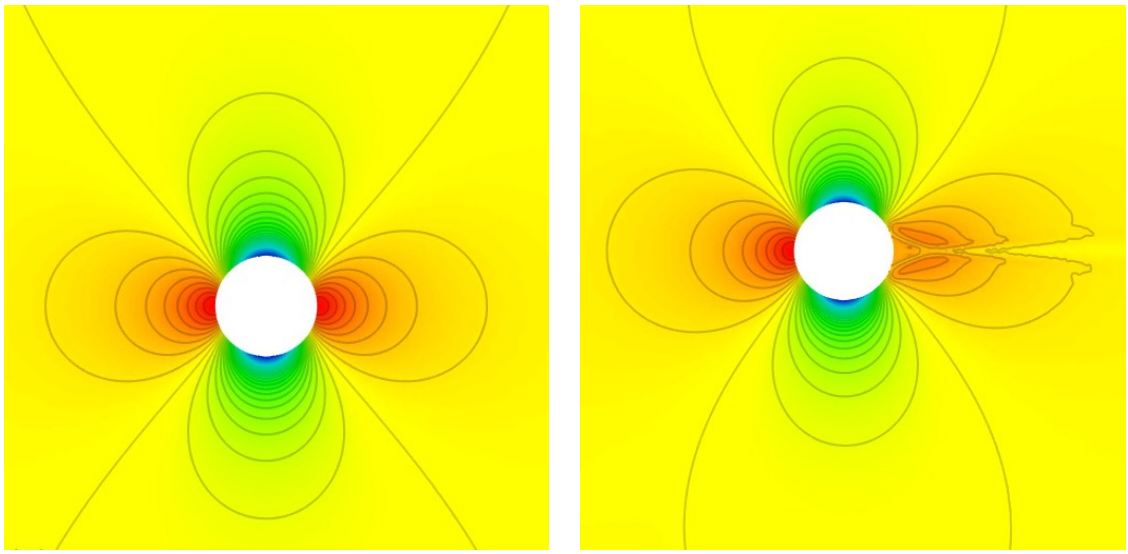

Figure 1.6 – Density field of a potential flow around a circle with slip boundary condition (left) and Riemann slip boundary condition (right).

1.3 Continuous mesh framework

In what follows, we introduce the concept of continuous mesh that will be used to derive optimal anisotropic interpolation error estimates.

1.3.1 Duality between discrete and continuous entities

In this section, some basic tools concerning the continuous mesh framework are recalled. More details about these techniques can be found in [Loseille 2008,

Loseille and Alauzet 2011a, Loseille and Alauzet 2011b, George et al. 2019] and the references therein. The main idea underlying the metric based mesh adaptation is to change the way distances are computed, via a continuous metric field $\mathbf{M} = (\mathcal{M}(x))_{x\in\Omega}$, where for all $x \in \Omega$, the matrix $\mathcal{M}(x)$ is a metric of $\mathbb{R}^d$, namely a symmetric definite positive matrix of $\mathbb{R}^d$. As it is explained below, the notion of unit mesh with respect to a metric field establishes a link between a metric field and a mesh, which reduces the problem of finding an optimal mesh to the problem of finding an optimal metric field. First of all, the scalar product induced by a constant metric $\mathcal{M}$ is given by:

$$\langle x,\, y\rangle_{\mathcal{M}} = x^T\, \mathcal{M}\, y, \text{ for all } x, y \in \mathbb{R}^d. \tag{1.11}$$

The following distance results from this scalar product

$$\|x\|_{\mathcal{M}} = \sqrt{x^T\, \mathcal{M}\, x}, \text{ for all } x \in \mathbb{R}^d. \tag{1.12}$$

Likewise, in the Banach space $\left(\mathbb{R}^d, \|.\|_{\mathcal{M}}\right)$, the distance between two points $a, b \in \mathbb{R}^d$ is given by:

$$\ell_{\mathcal{M}}\,(ab) = \sqrt{(ab)^T\, \mathcal{M}\, ab}, \tag{1.13}$$

where ab stands for the vector linking a to b. In the same way, if $\mathbf{M} = (\mathcal{M}(x))_{x\in\Omega}$ is a smooth metric field on Ω, the length of the segment between a and b is computed by using the integral formula:

$$\ell_{\mathcal{M}}\,(ab) = \int_0^1 \sqrt{(ab)^T\, \mathcal{M}\,(\gamma(t))\, ab}\, dt, \tag{1.14}$$

where $\gamma(t) = (1-t)a + tb$.
In a same manner, the volume of an element under a metric field is given by:

$$|\Omega|_{\mathcal{M}} = \int_\Omega \sqrt{\det \mathcal{M}(x))}\mathrm{d}\Omega.$$

As the next definition shows, there is a strong correspondence between continuous metric spaces and meshes, through the notion of unit mesh with respect to a metric field.

Definition 1.3.1. *An element K of a mesh $\mathcal{H}$ is said to be unit with respect to a continuous metric field $\mathbf{M} = (\mathcal{M}(x))_{x\in\Omega}$ if the lengths of its edges equal 1. That is to say, if $\{e_1, ...e_{m+1}\}$ are the edges of an element K of $\mathcal{H}$, then we have:*

$$\ell_{\mathbf{M}}(e_i) = 1, \quad \textit{for all} \quad i \in \{1, ..., m+1\}\,.$$

Definition 1.3.2. *A mesh $\mathcal{H}$ of a domain $\Omega \subset \mathbb{R}^d$ is said to be unit with respect to a continuous metric field $\mathbf{M} = (\mathcal{M}(x))_{x\in\Omega}$ if all its elements are unit.*

In particular, if K is a tetrahedron, then its volume is given by:

$$|K|_{\mathcal{M}} = \frac{\sqrt{2}}{12} \quad \text{and} \quad |K| = \frac{\sqrt{2}}{12}(\det(\mathcal{M}(x)))^{-\frac{1}{2}},$$

In fact, for a given element K such that $|K| \neq 0$, there exists a unique metric tensor $\mathcal{M}$ for which element K is unit with respect to $\mathcal{M}$. Consequently, the function *unit with respect to* defines classes of equivalences of discrete elements. A metric tensor $\mathcal{M}$ is therefore called a continuous element. Similarly, $\mathbf{M} = (\mathcal{M}(x))_{x\in\Omega}$, is called a continuous mesh. Using this analogy, a correspondence between discrete and continuous entities can be set [Loseille and Alauzet 2011a]. This is summarized in Table 1.1.

DISCRETE	CONTINUOUS
Element K	Metric tensor $\mathcal{M}$
Mesh $\mathcal{H}$ of Ω_h	Riemannian metric space $\mathbf{M}(\mathbf{x}) = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$
Number of elements N_e	Complexity $\mathcal{C}(\mathcal{M}) = 6\sqrt{2}\int_\Omega \sqrt{\det(\mathcal{M}(\mathbf{x}))}\mathrm{d}\Omega$
Linear interpolate $\Pi_h u$	Continuous linear interpolate $\Pi_{\mathcal{M}} u$

Table 1.1 – The continuous mesh model draws a duality between the discrete domain and the continuous domain.

1.3.2 Optimal control of the interpolation error in L^p norm

Mesh adaptation consists in finding the mesh $\mathcal{H}$ of a domain Ω that minimizes a given error for a given function u. For the sake of simplicity, we consider here the linear interpolation error $u - \Pi_h u$ controlled in L^p norm and that u is twice differentiable. The problem is thus stated in an *a priori* way:

$$\text{Find } \mathcal{H}_{opt} \text{ having } N \text{ vertices such that } \mathbf{E}_{L^p}(\mathcal{H}_{opt}) = \min_{\mathcal{H}} ||u - \Pi_h u||_{L^p(\Omega_h)},$$

with H_u the hessian of u. Such a formulation has several drawbacks, notably this is NP-complete. For this reason, its continuous counterpart is considered:

$$\text{Find } \mathbf{M}_{opt} \text{ having a complexity } \mathcal{N} \text{ such that } \mathbf{E}_{L^p}(\mathbf{M}_{opt}) = \min_{\mathbf{M}} ||u - \Pi_{\mathcal{M}} u||_{L^p(\Omega_h)},$$

where $\Pi_{\mathcal{M}} u$ is the continuous interpolate defined (in 3D) by:

$$\Pi_{\mathcal{M}} u(\mathbf{a}) = u(\mathbf{a}) + \nabla u(\mathbf{a}) + \frac{1}{20}\mathrm{trace}(\mathcal{M}(\mathbf{a})^{-\frac{1}{2}}|H_u(\mathbf{a})|\mathcal{M}(\mathbf{a})^{-\frac{1}{2}}).$$

The following problem is solved using a calculus of variations. Indeed, the previous problem can be rewritten into:

$$\begin{aligned}\text{Find } \mathbf{M}_{L^p} = \min_{\mathbf{M}} \mathbf{E}_{L^p}(\mathbf{M}) &= \left(\int_\Omega (u(\mathbf{x}) - \Pi_{\mathcal{M}} u\,(\mathbf{x}))^p \,\mathrm{d}\mathbf{x}\right)^{\frac{1}{p}} \\ &= \left(\int_\Omega \mathrm{trace}\left(\mathcal{M}(\mathbf{a})^{-\frac{1}{2}}|H_u(\mathbf{a})|\mathcal{M}(\mathbf{a})^{-\frac{1}{2}}\right)^p \mathrm{d}\mathbf{x}\right)^{\frac{1}{p}},\end{aligned}$$

under the constraint $\mathcal{C}(\mathbf{M}) = 6\sqrt{2}\int_\Omega \sqrt{\det(\mathcal{M}(\mathbf{x}))}\mathrm{d}\Omega = \mathcal{N}$. Using a calculus of variations, we prove that an unique solution exists on the space of continuous meshes. In [Loseille and Alauzet 2011b], this unique solution is given by:

$$\mathcal{M}_{L^p}(\mathbf{x}) = \mathcal{N}^{\frac{2}{3}}\left(\int_\Omega \det(|H_u(\tilde{\mathbf{x}})|)^{\frac{p}{2p+3}}\mathrm{d}\tilde{\mathbf{x}}\right)^{-\frac{2}{3}} \det(|H_u(\mathbf{x})|)^{-\frac{1}{2p+3}} H_u(\mathbf{x}). \tag{1.15}$$

1.4 Feature-based anisotropic mesh adaptation for steady flows

In the previous section, we have presented the continuous mesh model, on which is based a powerful framework to handle mathematically adapted meshes and we have also seen its application to the optimal control of the interpolation error. However, the assumption that was made is that the solution is analytically known. In this section, we tackle the subject in the case where the solution is discrete and only known at the vertices of the mesh. To this end, we present the classic mesh adaptation process for steady flows, which is a fixed-point algorithm where both solution and meshes are converged. The principle is to start from an initial coarse mesh and then to perform successive mesh adaptations. This way, the physics are progressively captured. At each stage of this fixed-point algorithm, the flow solver is called, an error estimate is then used to derive a metric. The latter is used as size map to perform mesh regeneration and adaptation. The considered estimate is a feature-based (or hessian-based) one.

1.4.1 Mesh adaptation algorithm for numerical simulations

Anisotropic mesh adaptation is a non-linear problem, therefore an iterative process is required to solve this problem. For steady simulations, an adaptive computation is carried out *via* a mesh adaptation loop inside which an algorithmic convergence of the mesh-solution couple is sought. This mesh adaptation loop is explained in **Algorithm 1** and in Figure 1.7 where $\mathcal{H}$, $\mathcal{S}$, $\mathcal{M}$ denote respectively meshes, solutions and metrics.

Algorithm 1: Mesh adaptation for Steady flows

init: Initial mesh and solution $(\mathcal{H}_0, \mathcal{S}_0)$ and target complexity $\mathcal{N}$

for $i = 0, n_{adap}$ **do**

1. (S_i) = Compute Solution with the flow solver from pair $(\mathcal{H}_i, \mathcal{S}_i^0)$;
   if $i = n_{adap}$ **then** break;
2. $(\mathcal{M}_{L^p,i})$ = Compute metric $\mathcal{M}_{L^p}$ according to selected error estimate from $(\mathcal{H}_i, \mathcal{S}_i)$;
3. $(\mathcal{H}_{i+1})$ = Generate a new adapted mesh from pair $(\mathcal{H}_i, \tilde{\mathcal{M}}_{L^p,i})$;
4. $(\mathcal{S}_{i+1}^0)$ = Interpolate new initial solution from $(\mathcal{H}_{i+1}, \mathcal{H}_i, \mathcal{S}_i)$

Note that this loop can be applied several times for a sequence of given mesh complexities, for instance $\mathcal{N}$, $2\mathcal{N}$, ...

1.4.2 Hessian-based anisotropic mesh adaptation

Unlike the case where u is known continuously, two major difficulties occur when applying mesh adaptation to numerical simulations:

- the continuous solution of the problem u is not known, only its numerical approximation u_h is available (it is provided by the flow solver),
- a control of the approximation error is expected, $u - u_h$ instead of $u - \Pi_h u$.

Control of the approximation error

Here, the interpolation theory is applied in the case when u_h is only known as a piecewise linear approximation of the solution. Indeed, in this case, the interpolation error

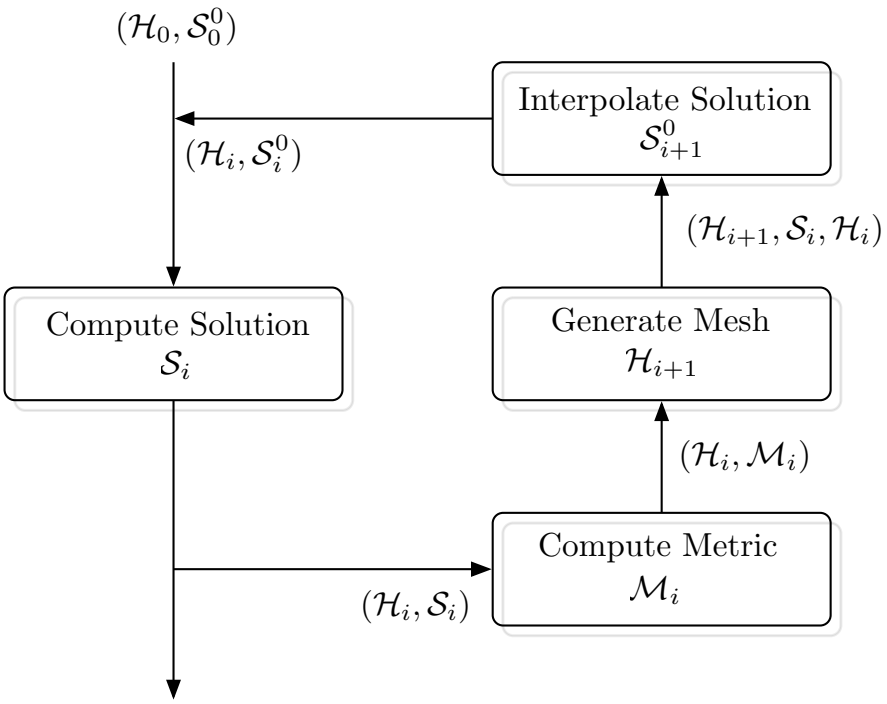

Figure 1.7 – Mesh adaptation algorithm.

estimates (1.15) cannot be readily applied to u nor u_h.
Let $\bar{V}_h^n$ be the space of piecewise polynomials of degree n (possibly discontinuous) and V_h^k be the space of continuous piecewise polynomials of degree n associated with a given mesh $\mathcal{H}$ of domain Ω_h. We denote by R_h a reconstruction operator applied to numerical approximation u_h. This reconstruction operator can be either a recovery process [Zienkiewicz and Zhu 1992a], a hierarchical basis [Bank and Smith 1993], or an operator connected to an *a posteriori* estimate [Huang et al. 2010]. We assume that the reconstruction $R_h u_h$ is more accurate than u_h for a given norm $\|.\|$ in the sense that:

$$\|u - R_h u_h\| \leq \alpha \|u - u_h\| \quad \text{where} \quad 0 \leq \alpha < 1 \,.$$

From the triangle inequality, we deduce:

$$\|u - u_h\| \;\leq\; \frac{1}{1-\alpha} \|R_h u_h - u_h\| \,.$$

R_h has the property: $\Pi_h R_h \phi_h = \phi_h \,,\ \forall \phi_h \in V_h^1$, the approximation error of the solution can be bounded by the interpolation error of reconstructed function $R_h u_h$:

$$\|u - u_h\| \leq \frac{1}{1-\alpha} \|R_h u_h - \Pi_h R_h u_h\| \,.$$

From the previous section, if $\mathcal{H}_{L^p}$ is an optimal mesh to control the interpolation error in L^p norm of $R_h u_h$, then the following upper bound of the approximation error can be exhibited:

$$\|u - u_h\|_{L^p(\Omega_h)} \leq \frac{3\, N^{-\frac{2}{3}}}{1-\alpha} \left(\int_\Omega \det \left(|H_{R_h u_h}(\mathbf{x})| \right)^{\frac{p}{2p+3}} \mathrm{d}\mathbf{x} \right)^{\frac{2p+3}{3p}} .$$

This upper bound is true for any $n \geq 1$ but is too large for $n > 1$. Though, the same analysis can be performed in general for derivatives of order $n+1$ and similar relations can be found in [Coulaud and Loseille 2016].

In the context of numerical simulations, u_h lies in V_h^1 and its derivatives ∇u_h in $\bar{V}_h^0$. We propose a reconstruction operator from V_h^1 into V_h^2 based on P^2 Lagrange finite element test functions. As approximate solution u_h is only known at mesh vertices, we need to reconstruct mid-edge values. To this end, we consider the L^2-projection operator $\mathcal{P} : \bar{V}_h^0 \to V_h^1$ defined by [Clément 1975]:

$$\nabla_R u_h = \mathcal{P}(\nabla u_h) = \sum_{\mathbf{p}_i \in \mathcal{H}} \nabla_R u_h(\mathbf{p}_i)\phi_i \quad \text{where} \quad \nabla_R u_h(\mathbf{p}_i) = \frac{\sum_{K_j \in S_i} |K_j| \nabla(u_{h|K_j})}{\sum_{K_j \in S_i} |K_j|},$$

where $\mathbf{p}_i$ denotes the i^{th} vertex of mesh $\mathcal{H}$, S_i is the stencil of $\mathbf{p}_i$, ϕ the basis function of V_h^1 and $|K_j|$ denotes the volume of element K_j. These nodal recovered gradients are used to evaluate mid-edge values. For edge $\mathbf{e} = \mathbf{pq}$, the mid-edge value $u_h(\mathbf{e})$ is given by:

$$u_h(\mathbf{e}) = \frac{u_h(\mathbf{p}) + u_h(\mathbf{q})}{2} + \frac{\nabla_R u_h(\mathbf{p}) - \nabla_R u_h(\mathbf{q})}{8} \cdot \mathbf{pq},$$

which corresponds to a cubic reconstruction. The reconstructed function $R_h u_h$ of V_h^2 writes:

$$R_h u_h = \sum_{\mathbf{p}_i} u_h(\mathbf{p}_i)\psi_{\mathbf{p}_i} + \sum_{\mathbf{e}_j} u_h(\mathbf{e}_j)\psi_{\mathbf{e}_j},$$

where $\psi_{\mathbf{p}} = \phi_{\mathbf{p}}(2\,\phi_{\mathbf{p}} - 1)$ and $\psi_{\mathbf{e}} = 4\,\phi_{\mathbf{p}}\,\phi_{\mathbf{q}}$ are the P^2 Lagrange test functions. This reconstructed function can be rewritten $R_h u_h = u_h + z_h$ and by definition verifies:

$$\Pi_h R_h u_h = u_h \quad \text{thus} \quad \Pi_h z_h = 0.$$

Therefore, we deduce:

$$\|R_h u_h - \Pi_h R_h u_h\| = \|u_h + z_h - u_h\| = \|z_h - \Pi_h z_h\|.$$

Finally, the approximation error can be estimated by evaluating the interpolation error of z_h:

$$\|u - u_h\| \;\leq\; \frac{1}{1-\alpha}\|z_h - \Pi_h z_h\| \;\leq\; \frac{3\,N^{-\frac{2}{3}}}{1-\alpha}\left(\int_\Omega \det\left(|H_{z_h}(\mathbf{x})|\right)^{\frac{p}{2p+3}} \mathrm{d}\mathbf{x}\right)^{\frac{2p+3}{3p}}.$$

Note that the Hessian of z_h lies in $\bar{V}_h^0$. If nodal values are needed to build $\mathcal{M}_{L^p}$, then the L^2-projection operator can be applied to these hessians [Clément 1975]. This recovery procedure is somehow similar to the ones of [Zienkiewicz and Zhu 1992a, Zienkiewicz and Zhu 1992b]. Other reconstruction operators can be applied such as the double L^2-projection, the least square method or eventually the Green formula based approach [Frey and Alauzet 2005].

1.5 Conclusion

In this chapter, we have recalled all the basic tools and algorithmic operations used to solve the Euler equations and then perform mesh adaptation on it. The purpose of the next chapter is to extend these tools to the case of the embedded boundary method applied to the Euler equations.

Chapter 2

Embedded boundary method and applications

2.1 Introduction

Several strategies are used in CFD to deal with embedded geometries. Once the boundary is detected and the inner part of the object is set, two approaches exist. The first approach consists in applying a so-called penalization method [Angot et al. 1999, Abgrall et al. 2014a] by weakly imposing a given value in the inner part of the geometry. This way, the presence of the object is simulated and the appropriate boundary condition is applied. These methods are specifically used for Navier-Stokes equations with Dirichlet boundary conditions (*e.g.* u=0 or u=v on the boundary). Another approach consists in imposing the value in a strong way and apply the wished boundary condition at the interface. This method is used in particular for the Euler equations. The interface is then created in a specific way. It can either be the surface induced by the resulting finite volume cells [Wang et al. 2011] or a result of a local modification of the scheme [Fidkowski and Darmofal 2007a, Fidkowski and Darmofal 2007b, Park and Darmofal 2010]. This is the last strategy, called cut-cell method, that is used in this work. Finally, it can be noted that the coupling of embedded methods with anisotropic mesh adaptation strategies has been made in several contexts. First, it is used in the finite element resolution of the incompressible Navier-stokes equations with objects represented by a level-set function [Quan et al. 2014a, Quan et al. 2014b], then this approach can be completed with the possibility of simulating fluid-structure interactions problems [Hachem et al. 2013] and finally it has been tackled in the context of the compressible Navier-Stokes equations with penalization [Abgrall et al. 2014a].
In this chapter, we present the principle of the embedded boundary method applied to the Euler equations. The processing of an embedded geometry is done in three steps: the intersection between the geometry and the background mesh is performed, then the dual mesh is modified by means of a cut-cell method and finally, modifications are brought to the numerical scheme with the implementation of a specific boundary condition that simulates the presence of the object. In particular, we show the implementation details needed in the vertex-centered finite volume CFD solver `Wolf` and highlight it with numerical examples that show that the method gives the expected results. Then, a coupling of this method with anisotropic mesh adaptation using `Feflo.a/AMG` is performed to improve the accuracy of the results.

2.2 Geometrical intersection

In this section, we detail how to deal with the embedded boundary from a geometrical point of view and in particular how the intersection between the embedded geometry and

the background mesh is performed. The detection of the boundary is crucial as it will be later useful for the modification of the dual scheme and the implementation of a boundary condition to model the presence of the boundary.

2.2.1 Intersection algorithm

The detection of the embedded boundary relies on an intersection algorithm. In 2D, the used process is an edge/edge intersection algorithm and in 3D it is a triangle/edge intersection algorithm.

2D case

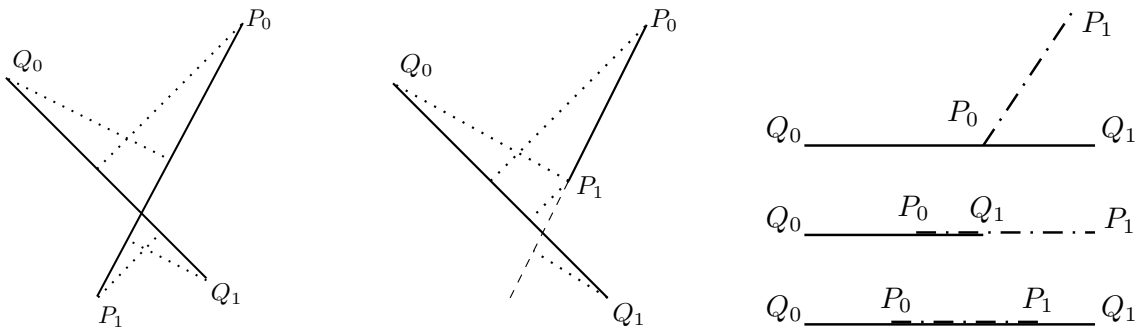

Figure 2.1 – Edge/edge intersection case: intersection (left), no intersection (middle) and degenerate cases (right).

The formula to compute an edge/edge intersection is the following [Alauzet and Mehrenberger 2010]:
Let $e_P = [P_0P_1]$ and $e_Q = [Q_0Q_1]$ be two edges of $\mathbb{R}^2$ and let us note $\mathbf{n}_{e_P}$ and $\mathbf{n}_{e_Q}$ their counterclockwise oriented normals.
If $P \in \mathbb{R}^2$, its *power* with respect to an edge $e_Q = \overrightarrow{Q_0Q_1}$ writes:

$$\mathcal{P}(P, e_Q) = \overrightarrow{Q_0P}.\frac{\mathbf{n}_{e_Q}}{||\mathbf{n}_{e_Q}||} = \overrightarrow{Q_1P}.\frac{\mathbf{n}_{e_Q}}{||\mathbf{n}_{e_Q}||}.$$

If the case is not degenerated ($\mathcal{P}(P_0, e_Q) \neq 0$ and $\mathcal{P}(P_1, e_Q) \neq 0$), then there is edge/edge intersection if and only if:

$$\mathcal{P}(P_0, e_Q)\mathcal{P}(P_1, e_Q) < 0 \quad \text{and} \quad \mathcal{P}(Q_0, e_P)\mathcal{P}(Q_1, e_P) < 0.$$

The intersection point X is then deduced:

$$X = P_0 + \frac{\mathcal{P}(P_0, e_Q)}{\mathcal{P}(P_0, e_Q) - \mathcal{P}(P_1, e_Q)}\overrightarrow{P_0P_1}.$$

The three degenerate cases are the following:

- A power equals 0, for instance $\mathcal{P}(P_0, e_Q) = 0$, then there is intersection if and only if $\mathcal{P}(Q_0, e_P)\mathcal{P}(Q_1, e_P) < 0$ and $X = P_0$.
- Two powers equal 0, one for each edge, for instance $\mathcal{P}(P_0, e_Q) = 0$ and $\mathcal{P}(Q_0, e_P) = 0$. In this case, there is intersection and $X = P_0 = Q_0$
- All the power equal 0, the edges are in fact aligned. There is intersection if and only if both edges overlap, which leads to the following two sub-cases:
  - ⋆ One intersection point that is the common point of both edges.
  - ⋆ Two intersection points that correspond to the extremities of one edge totally embedded in the other one or an extremity of each.

3D case

Like in the 2D case, degenerated cases appear. In this work, the strategy is to treat them in a first step as it is done in [Alauzet 2016].

Dealing with degenerated cases. We first introduce the 3D version of the *power*, of point P with respect to face $F = [P_0, P_1, P_2]$, whose outward normal is depicted by $\mathbf{n}_F$:

$$\mathcal{P}(P, F) = \overrightarrow{P_k P} \cdot \frac{\mathbf{n}_F}{\|\mathbf{n}_F\|} \text{ where } k \text{ is either } 0, 1 \text{ or } 2\,. \tag{2.1}$$

The distance of point P with respect to edge $e_i = \overrightarrow{P_0P_1}$ is:

$$\mathcal{P}(P, e_i) = \frac{\|\overrightarrow{P_0P_1} \times \overrightarrow{P_0P}\|}{\|\overrightarrow{P_0P_1}\|} = \frac{\|\overrightarrow{P_0P} \times \overrightarrow{P_1P}\|}{\|\overrightarrow{P_0P_1}\|}\,.$$

All possible vertex degenerated cases are checked according to $\mathcal{P}(P_j, F)$ values:

- is a vertex inside a face ?
- is a vertex on a edge ?
- are two vertices coinciding ?

Afterwards, edge/edge intersections - which are degenerated cases - are tested. Let P_0P_1 and Q_0Q_1 be the two considered lines. A necessary condition is that the two lines are coplanar. If it is the case, the intersection point $\mathbf{x}$ is evaluated following Hill's approach [Hill 1994]:

$$\begin{aligned} \mathbf{x} &= P_0 + \frac{\left(\overrightarrow{P_0Q_0} \times \overrightarrow{Q_0Q_1}\right) \cdot \left(\overrightarrow{P_0P_1} \times \overrightarrow{Q_0Q_1}\right)}{\|\overrightarrow{P_0P_1} \times \overrightarrow{Q_0Q_1}\|^2} \overrightarrow{P_0P_1} = P_0 + s\,\overrightarrow{P_0P_1}\,, \\ \text{or} \quad \mathbf{x} &= Q_0 + \frac{\left(\overrightarrow{P_0P_1} \times \overrightarrow{Q_0P_0}\right) \cdot \left(\overrightarrow{P_0P_1} \times \overrightarrow{Q_0Q_1}\right)}{\|\overrightarrow{P_0P_1} \times \overrightarrow{Q_0Q_1}\|^2} \overrightarrow{Q_0Q_1} = Q_0 + t\,\overrightarrow{Q_0Q_1}\,. \end{aligned}$$

Now, to check if the intersection point is an intersection between the two segments, we have to check that:

$$0 \leq s \leq 1 \quad \text{and} \quad 0 \leq t \leq 1\,.$$

Edge/face intersection. Dealing with degenerated cases first simplifies the following edge-face intersection procedure. Indeed, the computation of all the above geometric degeneracy treats in particular all the possible coplanar edge-face intersections as depicted in Figure 2.2.

Now, only non-coplanar edge-face intersection remains where the intersection point lies inside the face. The algorithm checks the 48 possible edge-face intersections. Let us denote the considered edge by $e = [P_0, P_1]$ and face by $F = [Q_0, Q_1, Q_2]$. As it is a non-coplanar case, the edge's vertex powers with respect to the face are not zero: $\mathcal{P}(P_0, F) \neq 0$ and $\mathcal{P}(P_1, F) \neq 0$. A necessary condition for the edge-face intersection is:

$$\mathcal{P}(P_0, F)\,\mathcal{P}(P_1, F) < 0\,.$$

If this condition is satisfied, the point of intersection between the edge and the plane defined by the face is computed:

$$\mathbf{x} = P_0 + \frac{\mathcal{P}(P_0, F)}{\mathcal{P}(P_0, F) - \mathcal{P}(P_1, F)} \overrightarrow{P_0P_1}\,.$$

Finally, to verify if the edge-face intersection effectively occurs, *i.e.,* point $\mathbf{x}$ lies inside face F, the signs of barycentrics of point $\mathbf{x}$ with respect to face F are analyzed. If there is intersection, point $\mathbf{x}$ is added to the intersection points list.

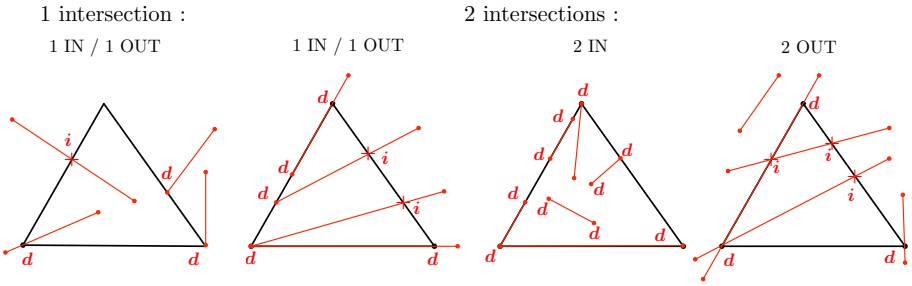

Figure 2.2 – *All possible intersection configurations for coplanar pair edge-face (all kinds of intersections are gathered by edge-face configurations). Four configurations can occur. One intersection with an edge's vertex in the face and the other vertex out. Two intersections with zero (resp. one or two) edge's vertex in the face and two (resp. one or zero) edge's vertices outside of the face. In the pictures, an intersection marked by "i" or "d" represents a pure or a degenerated intersection, respectively.*

2.2.2 Detection of the embedded boundary in 2D

Let us give a mesh $\mathcal{H}$ and the discretization of the embedded boundary I_h as a set of edges. Let us note $\mathcal{D}_h$, a sub-domain of $\mathcal{H}$ whose boundary is I_h and whose inner part is the approximation of the interior of the embedded object.
The first step is to determine which vertices of $\mathcal{H}$ are covered by $\mathcal{D}_h$ and to find the set of edges of $\mathcal{H}$ denoted as $\mathcal{E}_h$ that intersect I_h. The edges of $\mathcal{H}$ and I_h are respectively denoted by $e_{\mathcal{H}}$ and e_{I_h}. A core concept in all the following algorithms is the notion of mark whose principle is detailed here. This concept is the key to avoid algorithms with a "searching" of quadratic complexity.

Principle of the mark. When deploying an algorithm, it is common to create a list or stack of entities to be analyzed. To create this type of lists, a naive strategy would be to check if an entity is in the list before adding it. However, doing this creates an algorithm of quadratic complexity. A solution to overcome this issue is to create a flag table. For each of the entities is associated a flag. If, for instance, the considered object is added to the list its flag is activated. Thus, if its flag is already activated, this means that the element is already in the list and consequently and there is no need in adding it to the list. This way, the quadraticity of the algorithm vanishes. However, the flag table has to be reset before each of its use and this could be costly, in particular on large-size meshes. To this end, the mark is introduced. It consists in an integer value that is attached to the mesh and initialized to 0. The flag tables are then replaced by mark tables, which are also initialized to 0. Before each use of any of the mark tables, the mark is increased by 1 and then, the procedure is similar as with the flag table: if an object is added to the list, its mark (*i.e.* its corresponding value in the appropriate mark table) is set to the current mark. However, if its mark already equals the current mark, it is not added. Afterwards, the procedure is quite similar.

For instance, to detect the intersection between a triangle of $\mathcal{H}$ and e_{I_h}, the used procedure is as follows for a given edge, we choose one of the vertices of the edge, then we locate the triangle of $\mathcal{H}$ containing the vertex. Afterwards, all neighboring triangles of the previous triangles are checked in order to find if the embedded edge goes through this triangle or not *e.g* if at least one of the edges of the triangle is intersected. The mark is used to tag all the visited vertices and triangles along the path of the embedded edge. Note that the obtained set of tagged triangles is called a "pipe".

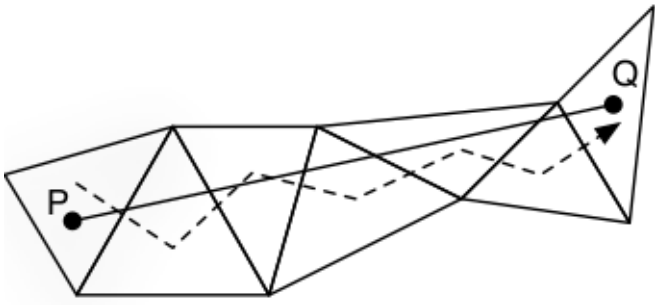

Figure 2.3 – Pipe of an embedded edge.

If for a triangle K_h, we note K_h^1, K_h^2, K_h^3 the three triangles respectively sharing the edge (e_1, e_2, e_3) with it, and if we note $\mathcal{K}_h$ the sort list of triangles to be analyzed for an intersection. Then, the intersection algorithm is given in **Algorithm 2**.

Algorithm 2: Two-dimensional geometry/mesh intersection algorithm

```
init: E_v = ∅
for e_{I_h} = [P, Q] ∈ I_h do
    mark = mark +1
    init: K_h | P ∈ K_h; i = 0; K_h = ∅; markTri(K_h) = mark
    while (i ≤ |K_h|) do
        if (i ≠ 0) then K_h = K_h[i];
        for j = 1..3 do
            if e_{I_h} ∩ e_j then
                E_v = E_v ∪ {e_j}
                if markTri(K_h^j) ≠ mark then
                    markTri(K_h^j) = mark ; K_h = K_h ∪ {K_h^j}
        i = i + 1
```

2.2.3 Generalization to 3D

The three-dimensional procedure is quite similar to the 2D algorithm. The embedded geometry is represented as a triangular surface mesh and the mesh $\mathcal{H}$ is a tetrahedral mesh. Since the embedded boundary method is studied for Finite Volumes purposes, we only need to handle the intersections of the embedded geometry discretization with the edges of $\mathcal{H}$. Basically, the idea is to locate the tetrahedra containing the 3 vertices of a given triangle of the embedded mesh and then to go through the neighboring tetrahedra containing the geometry thanks to an edge/triangle intersection algorithm. For the needs of the algorithm, let us note $(e_i)_{i\in[1,6]}$, the 6 edges of tetrahedron K_h and $(F_i)_{i\in[1,4]}$, the faces of K_h. Let us also note $\mathcal{K}_h^{e_i}$, the set of tetrahedra sharing the edge e_i with K_h and $K_h^{F_i}$ the tetrahedron sharing the face F_i with K_h. Finally, the triangles of the surface mesh I_h are noted T_{I_h}. Finally, we note $\mathcal{K}_h$ the sort list of triangles to be analyzed for an intersection. The 3D intersection algorithm is given in **Algorithm 3**.

Algorithm 3: Three-dimensional geometry/mesh intersection algorithm

init: $\mathcal{E}_v = \emptyset$
for $T_{I_h} = [P, Q, R] = [e^1_{T_{I_h}}, e^2_{T_{I_h}}, e^3_{T_{I_h}}] \in I_h$ **do**
  mark = mark +1
  init: $K_h \mid P \in K_h$; $i = 0$; $\mathcal{K}_h = \emptyset$; markTet($K_h$) = mark
  while $i \leq |\mathcal{K}_h|$ **do**
    if $i \neq 0$ **then** $K_h = \mathcal{K}_h[i]$;
    for $j = 1..6$ **do**
      if $T_{I_h} \cap e_j$ **then** $\mathcal{E}_v = \mathcal{E}_v \cup \{e_j\}$;
      for $K_h^{shell} \in \mathcal{K}_h^{e_i}$ **do**
        if markTet(K_h^{shell}) $\neq$ mark **then**
          mark(K_h^{shell}) = mark ; $\mathcal{K}_h = \mathcal{K}_h \cup \{K_h^{shell}\}$
    for ($j = 1..6$) and ($k = 1..3$) **do**
      if ($e^k_{T_{I_h}} \cap F_j$) and (markTet(K_h^j) $\neq$ mark) **then**
        markTet(K_h^j) = mark ; $\mathcal{K}_h = \mathcal{K}_h \cup \{K_h^j\}$
    $i = i + 1$

2.2.4 Detection of the covered vertices

Once the intersection is performed, the next step is to make sure if the intersection leads to an opened or closed embedded geometry. Indeed, if some vertices are covered by the geometry (*e.g.* it is a closed geometry), a constant solution is attached at this vertex and specific changes need to be done. The determination of the covered vertices is a consequence of the intersection algorithm: thanks to the mark and the set of connected neighbors to P_i noted $\nu(P_i)$, it is easy to determine them. In the following, the uncovered vertices will be marked and then the covered vertices are found. Let us note V_h as the set of the already checked vertices. Note this algorithm is independent of the dimension.

Algorithm 4: Algorithm for the detection of $\mathcal{D}_h$

init: $P \notin \mathcal{D}_h$; $V_h = \emptyset$; i = 0
mark = mark +1
while $i \leq |V_h|$ **do**
  for $Q \in \nu(P)$ **do**
    if $[P, Q] \in \mathcal{E}_v$ and markVer(Q) $\neq$ mark **then**
      markVer(Q) = mark
      $V_h = V_h \cup \{Q\}$
$\mathcal{D}_h = \mathcal{H} \setminus V_h$

2.3 Modification of the dual mesh via a cut-cell method

In this section, we detail the modification required to the finite volume dual mesh (see Section 1.2.2) when an embedded object is detected. Several strategies exist to deal with this problem. It is possible to directly apply slipping mirror boundary conditions, the obtained interface is the so-called *surrogate inter-*

face [Wang et al. 2011]. It is defined using the finite volume cells associated to the vertices that are not covered by the embedded geometry. Albeit its simplicity, this strategy is not really accurate. A possibility to remedy with that is to use a *cut-cell method* [Yang et al. 1997a, Yang et al. 1997b, Fidkowski and Darmofal 2007a, Fidkowski and Darmofal 2007b, Park and Darmofal 2010]. It consists in locally modifying the finite volume cells defined in Section 1.2.2 so that they match with the embedded interface.

2.3.1 Finite volume median cells

In the used solver, the finite volume cells are the median cells. In 2D, this method consists in building the cells triangle by triangle. Inside each of the triangles, the three medians that link the middle X_i of an edge e_i with the opposite vertex P_i are considered. Their intersection is the gravity center of the triangle X_G and using it with middle of the edges, we are able to divide a triangle into 3 quadrilaterals, one for each vertex of the triangle (see Figure 2.4 on the left). The 2D median finite volume cell of a point P consists in the gathering of all its associated quadrilaterals of the triangles containing P. In 3D, each tetrahedron is split into four hexahedra. To construct them for a point P_i of the tetrahedron, we use the middles $(X_j^i)_{j\in[1,3]}$ of the three incident edges to P_i the gravity centers $(XF_j^i)_{j\in[1,3]}$ of the 3 faces containing P_i, the gravity center X_G of the tetrahedron and the considered vertex P_i (see Figure 2.4 on the right). The 3D median finite volume cell of a point P consists then in the gathering of all its associated hexahedra of the tetrahedra containing P.

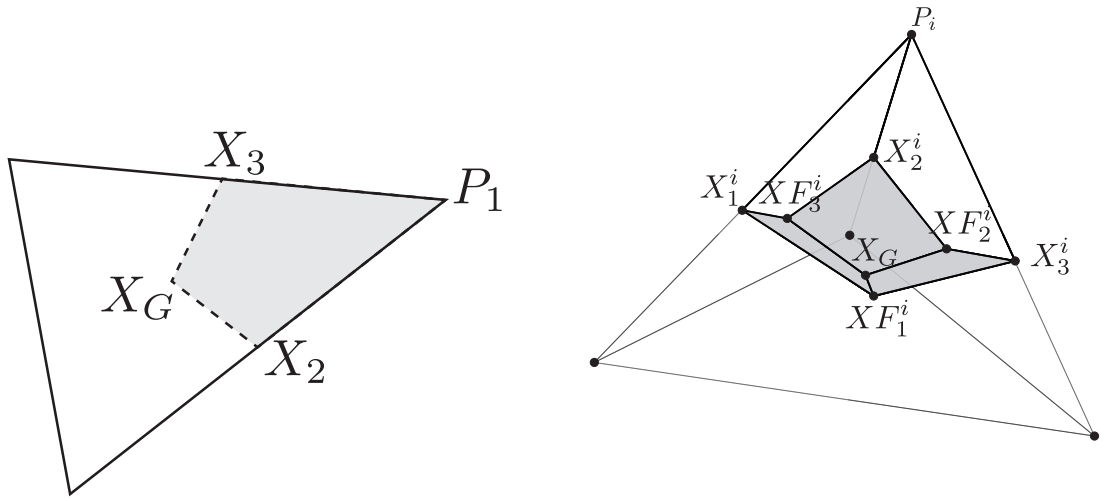

Figure 2.4 – Finite volume median cells.

2.3.2 Two-dimensional cut-cell

First, we tackle the two-dimensional case. Let us note K_h the triangles of $\mathcal{H}$ and let us reuse the notations of the previous section. In the most common case, two edges are intersected once. Let us assume it is e_1 and e_2. Let us note X_1, X_2, X_3 as the intersection points of the cell interfaces with the edges of the same number, and X_G the gathering point of the three cells of K_h. Let us finally note XI_1, XI_2, the intersection point of the

geometry with the edges e_1 and e_2. The principle of the cut-cell method is to modify the already existing finite volume-cells so that the intersection points are present in the dual mesh while keeping consistency with the classic strategy for the areas where no intersection occur. The process is explained in **Algorithm** 5 and an example of cut-cell is given in Figure 2.5. X_1, X_2 and X_G are modified so that the finite volume cells match with the embedded boundary while X_3 is kept intact to be consistent with a classic cut-cell in the neighboring triangle. Note that this algorithm is the same if only one (P_3, for instance) or two (P_1 and P_2 for instance) point are covered by the geometry as it does not need to know *a priori* which ones are covered.

Algorithm 5: Two-dimensional cut-cell algorithm

for $K_h \in \mathcal{H}$ **do**
  if $(e_1, e_2) \in \mathcal{E}_h \times \mathcal{E}_h$ **then**
    $X_1 = XI_1$; $X_2 = XI_2$; $X_3 = \frac{1}{2}(P_1 + P_2)$; $X_G = (X_3P_3) \cap (X_1X_2)$

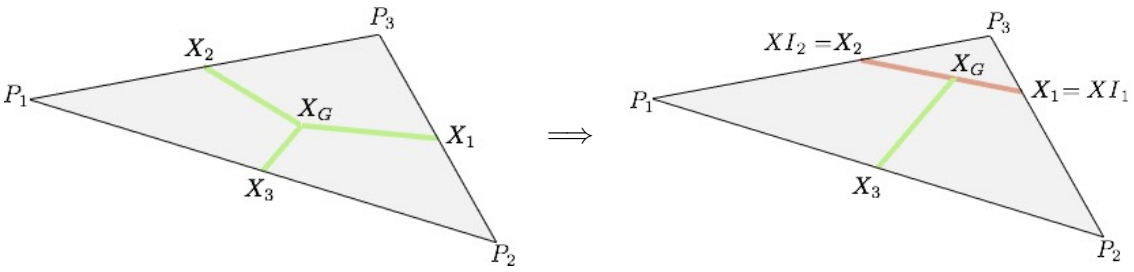

Figure 2.5 – Modification of the finite volume cells in 2D with a cut-cell method in the classic case. In green, the cell interfaces. In red, the embedded geometry interfaces.

Then several particular cases exist. The problem of the corners has to be tackled as well as the problem of the double intersection on an edge. In these cases, the cut-cell is also particular and explained in Figures 2.6 and 2.7.

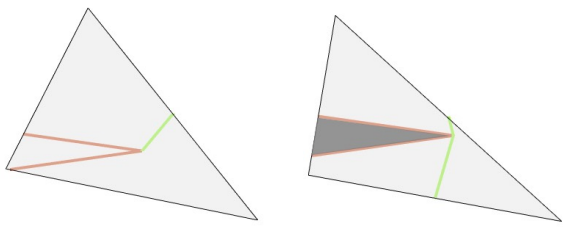

Figure 2.6 – Two particular cases of cut-cell in 2D dealing with the corners. In green, the cell interfaces. In red, the embedded geometry interfaces.

In Figure 2.6, the case of the corner is handled. On the left, there are intersections with two different edges. The procedure simply consists in setting the corner of the embedded surface as a new X_G. This procedure is the same if one (convex corner) or two (concave corner) points are covered by the geometry as it does not need to know *a priori* which ones are covered. On the right, there is a corner issued from a double intersections. If all the

three vertices of the triangle are not covered by the geometry, the finite volume cells are modified as depicted in Figure 2.6 (right) and some area of the triangle is erased. On the contrary, if the three vertices are covered, the corner is ignored.

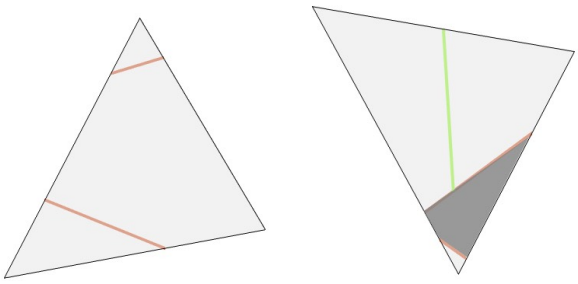

Figure 2.7 – Two particular cases of cut-cell in 2D dealing with the double intersections. In green, the cell interfaces. In red, the embedded geometry interfaces.

In Figure 2.7, the case of the double intersections is handled. On the left, there is one intersection on two different edges and a double intersection on the last one. The procedure simply consists setting the finite volume cells according to the cut part of the triangle a vertex belongs. This procedure is the same if one or two points are covered by the geometry as it does not need to know *a priori* which ones are covered. On the right, there are two double intersections. If all the three vertices of the triangle are not covered by the geometry, the finite volume cells are modified as depicted in Figure 2.7 (right) and some area of the triangle is erased. On the contrary, if the three vertices are covered, double intersections are ignored.

2.3.3 Three-dimensional cut-cell

In 3D, the idea behind the cut-cell is quite the same. First of all, a classical plane/tetrahedron intersection gives us two possibilities: a quadrilateral (this means that two points are covered by the geometry) or a triangle (this means that one or three points are covered). Like in 2D, the finite volume cells of the tetrahedron are locally modified to fit with the surface of intersection. For each face of the tetrahedron, there is either 2 or 0 intersections on the edges. Basically, for each face, it is possible to reproduce either the 2D classic cut-cell or the 2D classic median finite volume cells (in order to fit with the neighbors where no cut-cell has been done). Then, the gravity center of the tetrahedron is replaced by the gathering point of the four modified cells.
In the following algorithm, let us note $(e_i^j)_{i\in[1,3]}$, the three adjacent edges to a vertex P_j and X_i^j their middle. Let us also note $(F_i^j)_{i\in[1,3]}$ the three faces of the tetrahedron containing P_j so that F_1^j does not contain e_i^j and XF_i^j, their gravity center and let us note X_G the gravity center of the tetrahedron. Let us finally note XI_i^j, the intersection point (if any) of the geometry with the edges e_i^j. The principle of the algorithm is to set new values for X_G, XF_j^i and X_j^i in the tetrahedron so that the embedded boundary is well represented in the finite volume cells of the P^i. This process is summarized in **Algorithm** 6 and is illustrated in Figure 2.8. Like in the standard 2D case, this algorithm does not need to know *a priori* which vertices have been covered by the geometry.

In the same way as in 2D, some particular cases need to be treated. It might be interesting to take into account the corners, the required edges/ridges and the double intersections.

Algorithm 6: Three-dimensional cut-cell algorithm

for $K_h \in \mathcal{H}$ **do**
  for $P_j \in K_h$ **do**
    for $F_i^j \in K_h$ **do**
      if $F_i^j \cap I_h = \emptyset$ **then** $XF_i^j = bary(F_i^j)$;
      else
        $XF_i^j = X_G^{2D\ cut-cell}(F_i^j)$
    for $e_i^j \in K_h$ **do**
      if $e_i^j \cap I_h = \emptyset$ **then**
        $X_i^j = XI_i^j$
      else
        $X_i^j = middle(e_i^j)$
    $\text{Cell}(P_j) = [P_j, X_1^j, XF_1^j, X_2^j, X_3^j, XF_2^j, X_G, XF_3^j]$

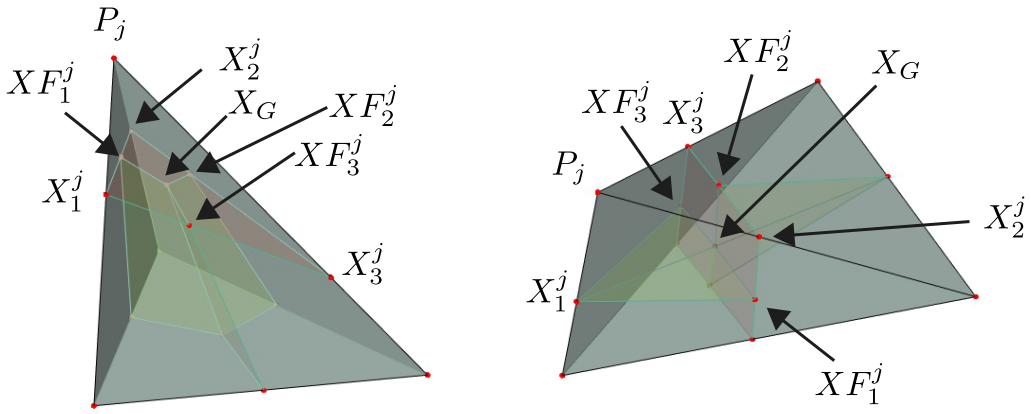

Figure 2.8 – Two classic cases of cut-cell in 3D.

These particular cases were not difficult to take into account in 2D, but it turns out to be pretty difficult to implement in 3D. That's why, it has been decided for the sake of simplicity to simplify these particular cases into a non intersection case or a classic intersection case. From a more general point of view, the question of taking into the singularities of the geometry is more a meshing-related problem than a solver-related. Thus, it should not be dealt with by a solver.

2.4 Modification of the numerical scheme

In this section, we give the modifications required to make the Embedded Boundary Method compatible with the numerical scheme of an already existing vertex-centered Finite-Volume solver. In particular, we show how the embedded intersection impacts the already existing scheme and then we explain how we modify the numerical flux to create an *embedded* boundary condition.

2.4.1 Impact on the vertices and edges covered by the geometry

As we are in the case of the Euler equations, we can directly enforce the solution vector W attached the vertices covered by the geometry. It is forced to reference value which is the value at the infinity in all our cases. This implies a change in the flux computation for an edge totally covered by the geometry (we recall that $\mathcal{D}_h$ is the set of vertices covered by the embedded geometry):

$$\Phi_{ij} = 0 \text{ if } (P_i, P_j) \in \mathcal{D}_h.$$

In the case of an implicit advance in time, a change in the differentiation of the flux is required for the edges where at least one vertex is covered:

$$\frac{\partial \Phi_{ij}(W_i^n, W_j^n, \mathbf{n}_{ij})}{\partial W_j} = 0 \text{ if } P_j \in \mathcal{D}_h.$$

In addition, the identity matrix is forced in the implicit linear system for the covered vertices. This means that the interactions of these points are canceled.

2.4.2 Impact on the gradient computation

The implementation of a finite volume scheme along with a MUSCL extrapolation relies on the computation of gradients. Nonetheless, gradient computation can be impacted by the embedded intersection. Gradient computation is based on some information provided by the upwind and downwind elements. If one of these elements is intersected by the embedded geometry, it is required to ignore the classical gradient computation and to compute a surrogate gradient as it is already the case with the boundary vertices. In the same way, nodal gradients, required for a V6-scheme, are modified as follows (d is the dimension):

$$(\nabla W)_{P_i} = \frac{1}{(d+1)|C_i|} \sum_{\substack{K \in C_i \\ K \notin \mathcal{D}_h}} |K| (\nabla W)_K.$$

2.4.3 Implementation of the embedded boundary condition

To simulate, the presence of the embedded boundary, we induce two types of slip conditions: a Riemann slip condition and a classic slip condition.

Embedded Riemann slip condition

The numerical flux is changed for the edges intersected by the geometry. In the first hand, a Riemann slip condition is weakly imposed. For an edge $e = [P_i, P_j]$ intersected by the embedded geometry, it writes [1]:

$$\Phi_{\textit{Embed Slip Riemann}} = \Phi_{ij}(W_i, \overline{W_i}, \mathbf{n}_{ij}) \text{ if } P_j \in \mathcal{D}_h \text{ and } P_i \notin \mathcal{D}_h \text{ (for instance)}, \tag{2.2}$$

The state $\overline{W}_i$ is the mirror state of W_i, associated to P_i. Thanks to this flux, a Riemann slip boundary condition is induced on the embedded surface.

1. Note that as the embedded interface Γ has become the interface of e by means of the cut-cell approach, we have $\mathbf{n}_\Gamma = \mathbf{n}_{ij}$. Consequently, the boundary is naturally recovered with the intersected edges.

Differentiation for implicit advance in time. In order to drastically reduce CPU time, the embedded boundary method has been extended to the implicit solver for the Euler equations. The jacobian of the flux for an intersected edge $e = [P_i, P_j]$ is modified as follows:

$$\frac{\partial \Phi_{ij}(W_i^n, W_j^n, \mathbf{n}_{ij})}{\partial W_i} = \frac{\partial \Phi_{ij}(W_i^n, \overline{W_i^n}, \mathbf{n}_{ij})}{\partial W_i} \text{ if } P_j \in \mathcal{D}_h \text{ and } P_i \notin \mathcal{D}_h \text{ (for instance).}$$

This way, a Riemann slipping boundary condition is induced.

A more accurate implementation

Mirror boundary condition is a first order approximation of the numerical flux done in a very critical area. To improve the accuracy, it seems natural to extrapolate the solution at the intersection between the embedded interface and the edge. To this end, a MUSCL type reconstruction method [Van Leer 1979, Debiez and Dervieux 2000] is used. To improve the accuracy, it seems natural to extrapolate. If we note $\overline{P}_i$ the symmetric point of P_i with respect to the intersection point XI on the edge P_iP_j (see Figure 2.9), we have for an intersected edge:

$$W_{XI} = W_i + \frac{1}{2}(\nabla W)_{ij}.\overrightarrow{P_i\overline{P}_i} = W_i + (\nabla W)_{ij}.\overrightarrow{P_iXI}, \tag{2.3}$$

and

$$\Phi_{Embed\ Slip\ Riemann} = \Phi_{ij}(W_{XI}, \overline{W_{XI}}, \mathbf{n}_{ij}).$$

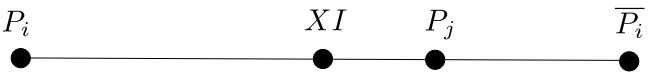

Figure 2.9 – Construction of the symmetric point used for the MUSCL extrapolation for an intersected edge.

The centered gradient computation is made by replacing W_j by $\overline{W_i}$ and P_j by $\overline{P_i}$ in every equation. Then, this second order extrapolation is locally coupled with a Dervieux or Piperno (if applicable) limiter. In Figure 2.10, we can see two simulations of the embedded boundary method in the case of a potential flow around a circle at Mach 0.1 on a mesh of 5 000 vertices. An HLLC solver is used along with a V6 scheme and no limiter. The implicit advance in time is used. This improved *embedded* boundary condition is applied around the detected circle and the result is shown after 500 iterations for two simulations, one without a MUSCL reconstruction on the boundary (left) and one with a MUSCL reconstruction on the boundary (right). A slight improvement of the results can be noticed when using the MUSCL, particularly with the contour lines.

Embedded slip condition (weak form)

In the previous section, the induced slip condition is a Riemann slip condition. This boundary condition is more stable but less accurate than a slip boundary condition (where $\mathbf{U}.\mathbf{n} = 0$ is enforced weakly), that is why another flux corresponding to a slip condition in weak form is proposed:

$$\Phi_{Embed\ Slip} = (0, p_{XI}\mathbf{n}, 0)^T,$$

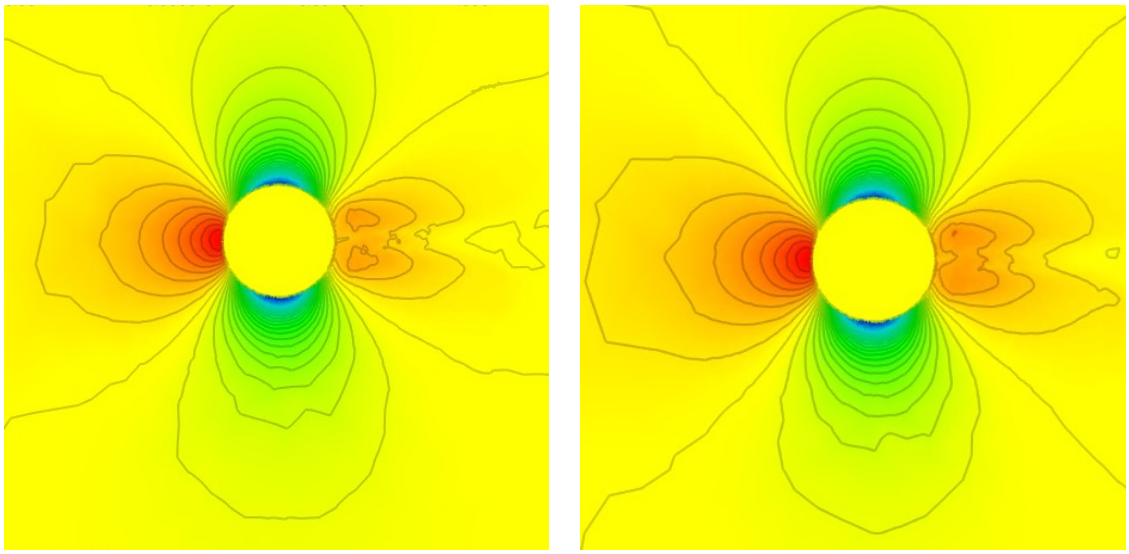

Figure 2.10 – Comparison of the solution without MUSCL on the boundary (left) and with MUSCL on the boundary (right).

where p_{XI} is the pressure related to W_{XI} computed thanks to Equation (2.3). For the implicit advance in time, the differentiation of the embedded Riemann slip condition is used.

2.5 Numerical results

In this section, we show the numerical results of flow simulations performed using the embedded boundary method.

2.5.1 On the geometrical accuracy of the embedded geometry

One of the major concerns of numerical computations is to give geometries that are consistent with the analytical definition of the initial geometry. In particular, the mesh size h_{I_h} from I_h needs to be locally consistent with the mesh size $h_{\mathcal{H}}$ from the computing mesh $\mathcal{H}$. Moreover, size h_{I_h} should be such that the curvature of the objects is preserved. For instance, a circle does not have any discontinuity on its outward normal whereas a regular polygon, one of its possible discretizations, has some. More precisely, if $h_{I_h} \leq h_{\mathcal{H}}$, there is not any effect of the discretization of the embedded boundary on the numerical solution (see Figure 2.11 bottom) but if $h_{I_h} \geq h_{\mathcal{H}}$, the geometric representation brings some undesired effects in the solution (see Figure 2.11 top). In the example Figure 2.11, the embedded geometry is seen by the flow solver as a piecewise linear wing whereas, the wing geometry is C^∞. Consequently, every discontinuity (in the tangent) creates vortices that are propagated in the domain. This is a pure artifact due to an inconsistent resolution of the true geometry.
As we will see in the following sections, mesh adaptation can highly increase the resolution of the mesh next to the boundary, therefore a reliable discretization of the boundary is needed for these simulations.

2.5.2 Steady flow simulations

To test the reliability of the code, several standard steady cases have been tested with this method. Each case compares a body-fitted simulation with its embedded counterpart.

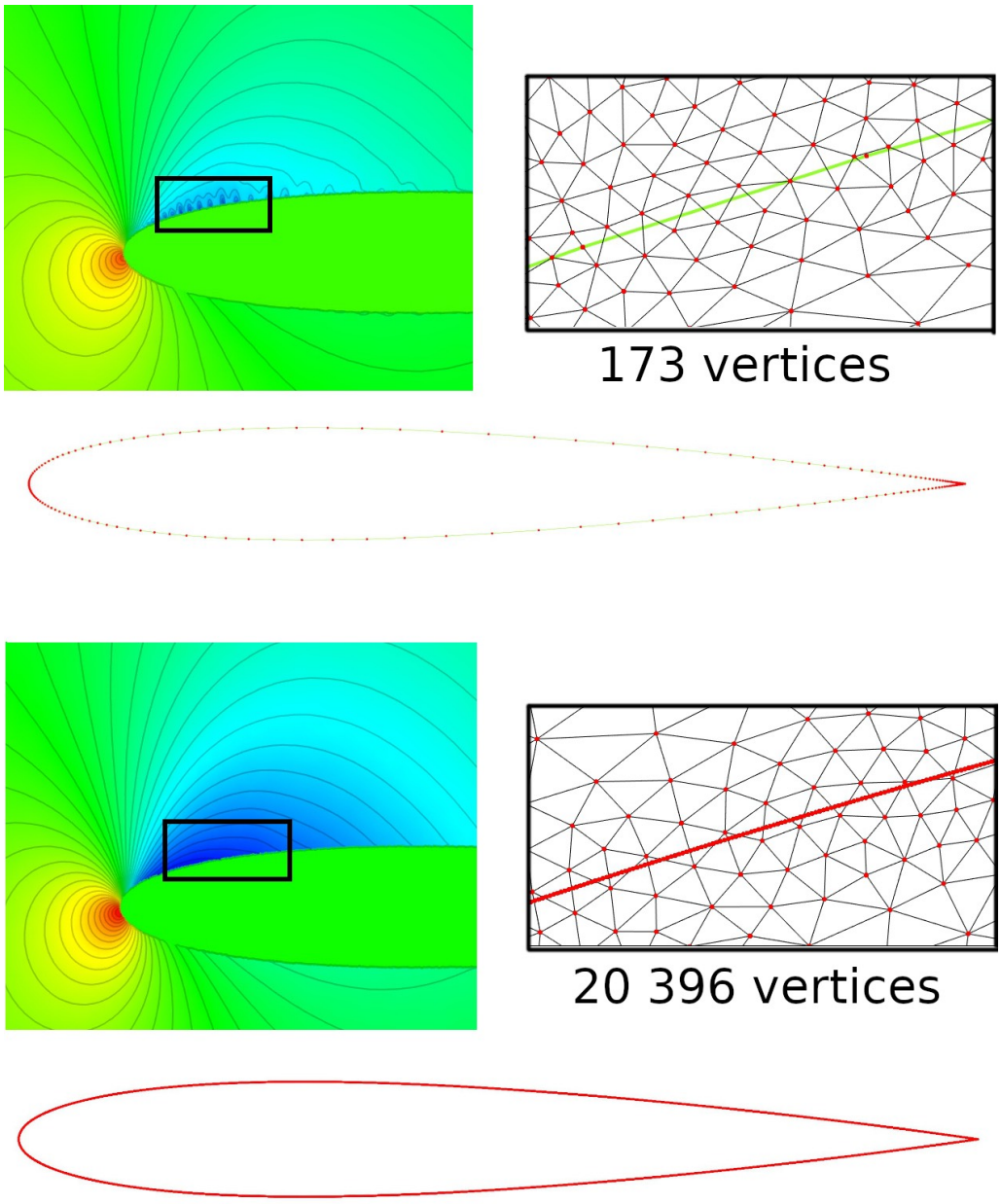

Figure 2.11 – On the left, computed solution after a mesh adaptation process with a coarse (top) and fine (bottom) representation of the geometry. On the right, zoom on the black square region. The extremities of the edges are the red points and the embedded boundary is in green. The whole embedded mesh along with the number of vertices are also given.

Circle

As a validation test case, the potential flow over a circle presented in Section 1.2.8 is used. To compute this flow, an HLLC solver is used along with a V6 scheme and no limiter. An implicit advance in time is used. The studied boundary condition on the circle is a slipping boundary condition. The results are shown after 1000 iterations. In Figure 2.13, two embedded cases are studied. The first one is a toy problem as it consists in explicitly representing the geometry in the background mesh. This is done so that we can make sure that the flow solver behaves as expected like in the

body-fitted case. The second one is a standard embedded case where the geometry is not explicitly defined in the mesh. The size of the meshes is of approximately 36 000 vertices and the embedded circle is defined with a uniform distribution of 189 vertices. The results show that the embedded boundary method degenerates into the body-fitted one when the edges of the embedded and the background mesh match. This is confirmed by the plot of C_p curves (Figure 2.12) where neither case can be distinguished. However, the other simulation shows the drawback of the embedded approach for meshes not fitted for the geometry. Both solution and C_p curves show that the behavior is the one expected but that a loss of accuracy in the vicinity of the geometry is present. It can be noted that the solution is highly impacted by the point distribution in the background mesh. Indeed, the induced intersected geometry is less well represented where the mesh is coarse which provides the loss of symmetry in the profile around the circle.

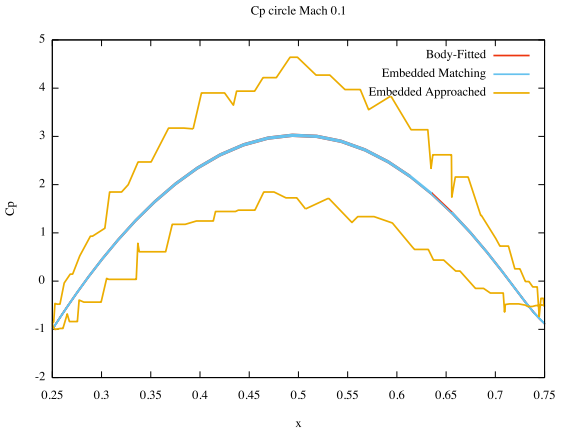

Figure 2.12 – Computed C_p of a potential flow around a circle. In red, the body-fitted case. In blue, the embedded matching case. In yellow, the standard embedded case.

NACA0012 airfoils

Using this method, flow simulations at three different regimes around NACA0012 airfoils have been done. The used meshes are shown in Figure 2.14 and the displayed field is the flow density. To compute these flows, an HLLC solver is again used along with a V4 scheme and a Piperno limiter. An implicit advance in time is used. The studied boundary condition on the airfoil is a slipping boundary condition. For all these simulations, the same meshes are used for all the regimes. A mesh of 4812 vertices for the embedded case and a mesh of 5041 vertices for the body-fitted case. The embedded geometry consists in a mesh of 20 396 vertices. For these meshes the treatment of the embedded geometry requires a CPU time during the preprocessing of the same order as one iteration of the flow solver.

Subsonic case. The first case is a subsonic flow (Mach 0.63) around NACA0012 airfoil with an angle of attack of 2 degrees. For this case, the solver is run until convergence is reached with a residual lower than 10^{-8}. For the body-fitted case, it is reached after 280 iterations whereas it is reached 1165 iterations for the embedded case. This phenomenon can be explained by a simple reason (among others): the minimal time step of the flow solver in the body-fitted case is of 0.00103 whereas it is of 0.00023 in the embedded case. Indeed, the cut-cell performed in the preprocessing step usually reduces the size of the finite volume cells. This reduction thus provokes an increase of the number of time steps.

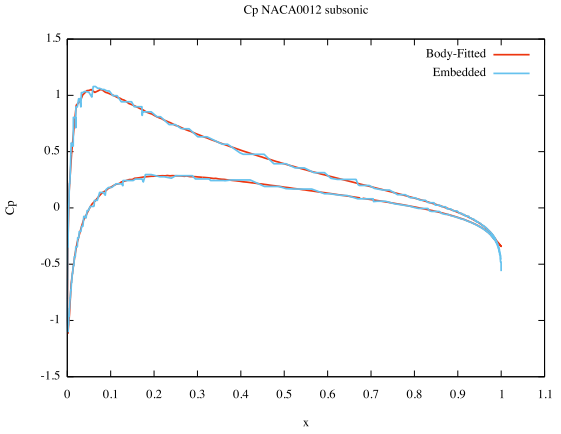

Figure 2.15 – Computed C_p of a subsonic flow around a NACA0012 airfoil.

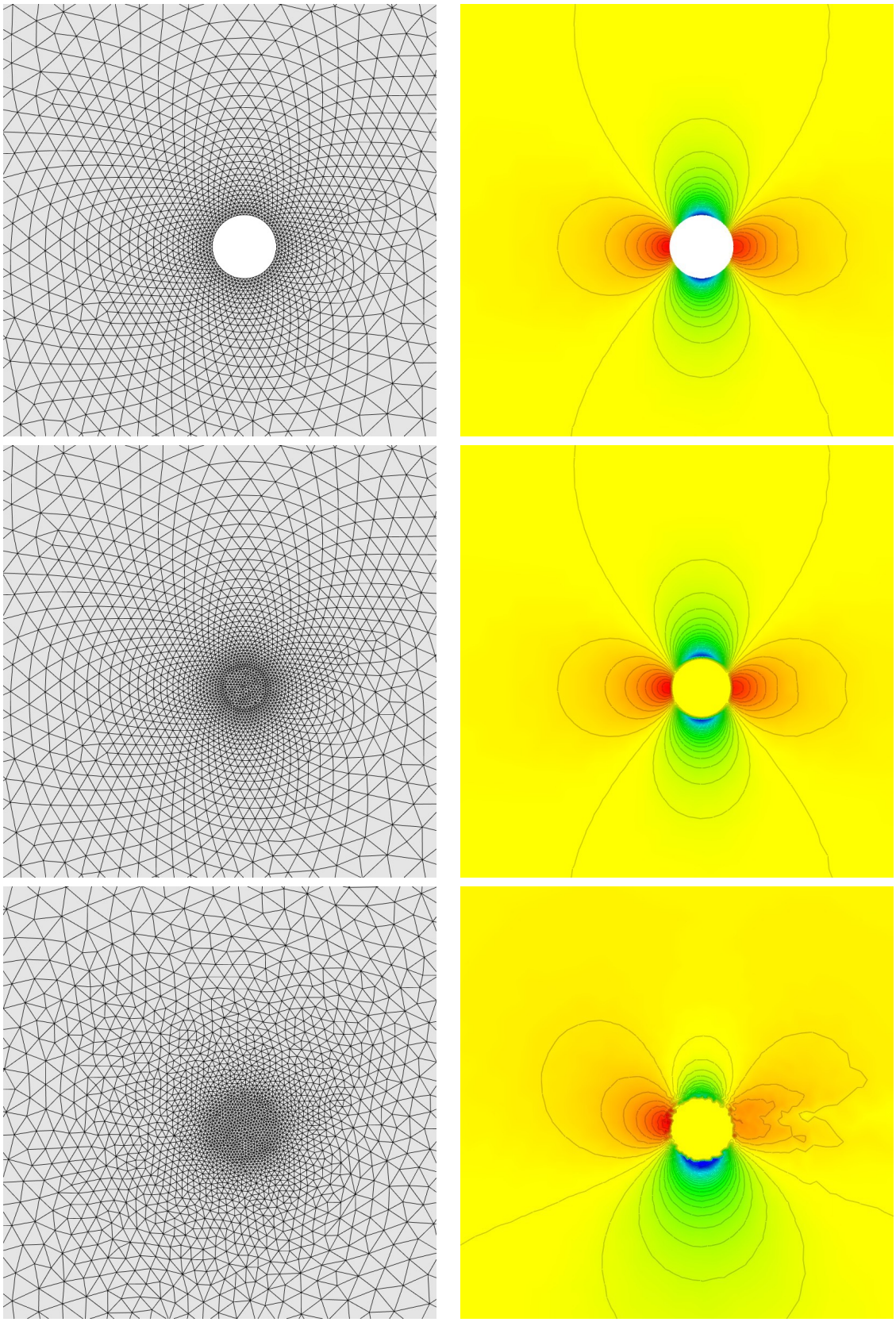

Figure 2.13 – Simulation of potential flow around a circle. Mesh (left) and density field (right) for three cases: body-fitted (top), embedded with exact representation of the circle (middle) and embedded without explicit representation of the circle (bottom).

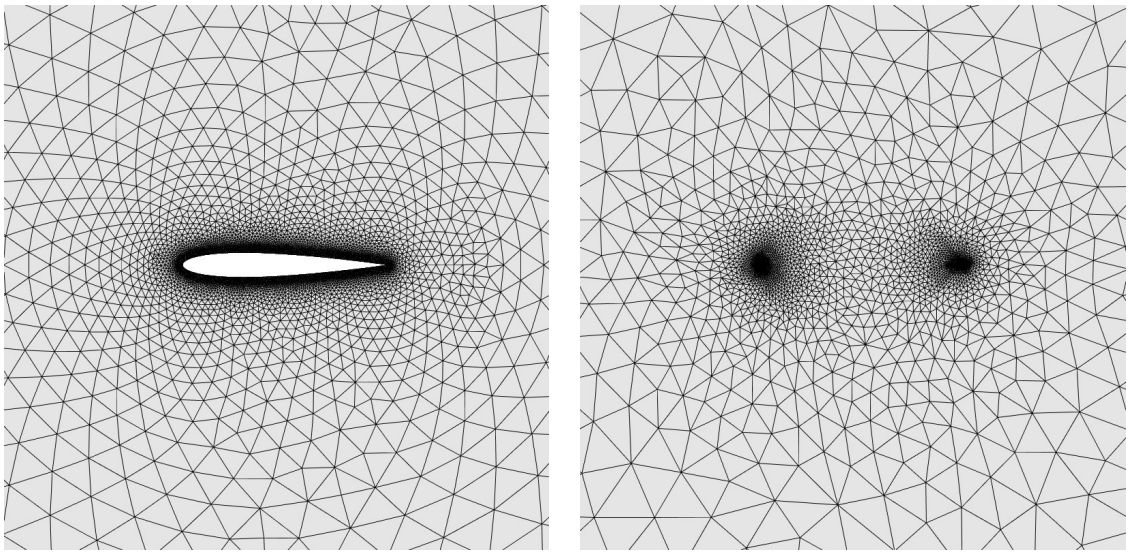

Figure 2.14 – Body-fitted (left) and embedded (right) meshes used for the simulations of the flow around a NACA0012 airfoil.

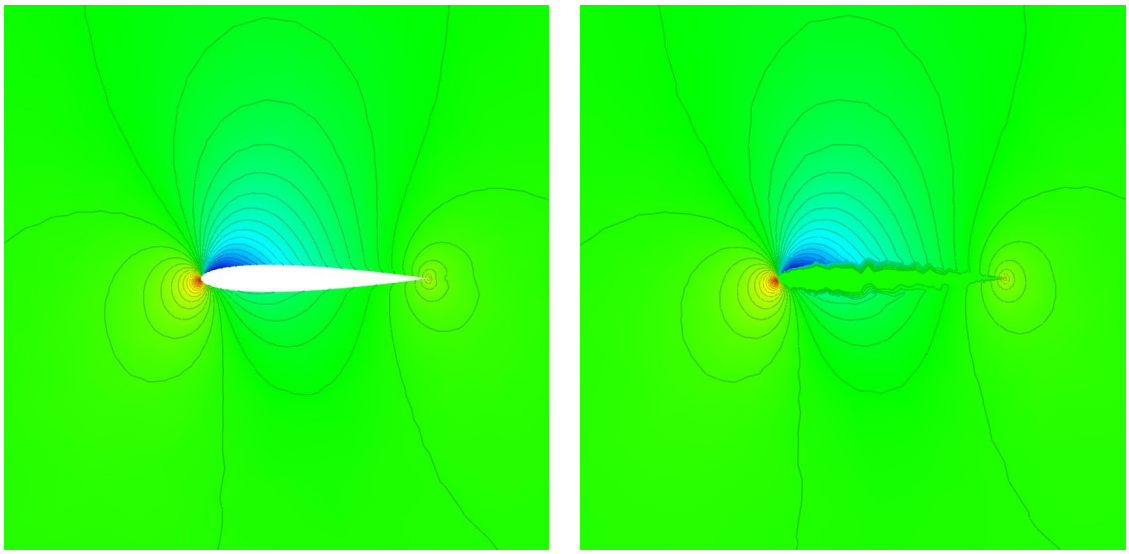

Figure 2.16 – Body-fitted (left) and embedded (right) simulation of a subsonic flow around a NACA0012 airfoil.

The obtained results are quite similar in trend (see Figure 2.16) and this can be confirmed by the C_p (Figure 2.15) curve around the airfoil even if we can see that the curve oscillates a lot in the embedded case. However, note that a relative error of 54% is obtained for drag computation which is not surprising as it is based on integration over the boundary of the object.

Transonic case. The second case is a transonic flow (Mach 0.85) around a NACA0012 airfoil with an angle of attack of 1.25 degrees. For this case, the solver is stopped after 500 iterations. The minimal time step of the flow solver in the body-fitted case is of 0.00012 whereas it is of 0.000029 in the embedded case. This is again a consequence of the cut-cell performed in the preprocessing step.

The obtained results are quite similar in trends (see Figure 2.18) and this can be confirmed by the C_p (Figure 2.17) curve around the airfoil. Again, an oscillation is seen with embedded case. Note that a relative error of 3% is obtained for the drag computation.

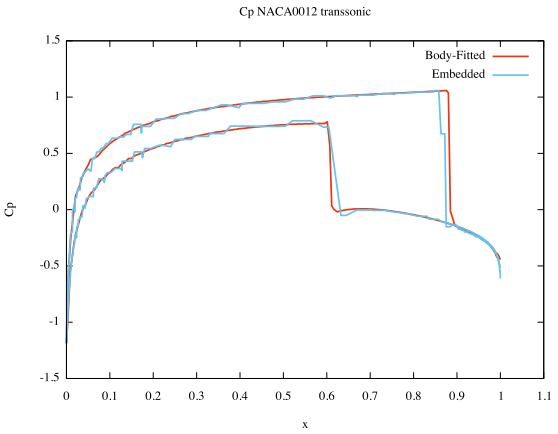

Figure 2.17 – Computed C_p of a transonic flow around a NACA0012 airfoil.

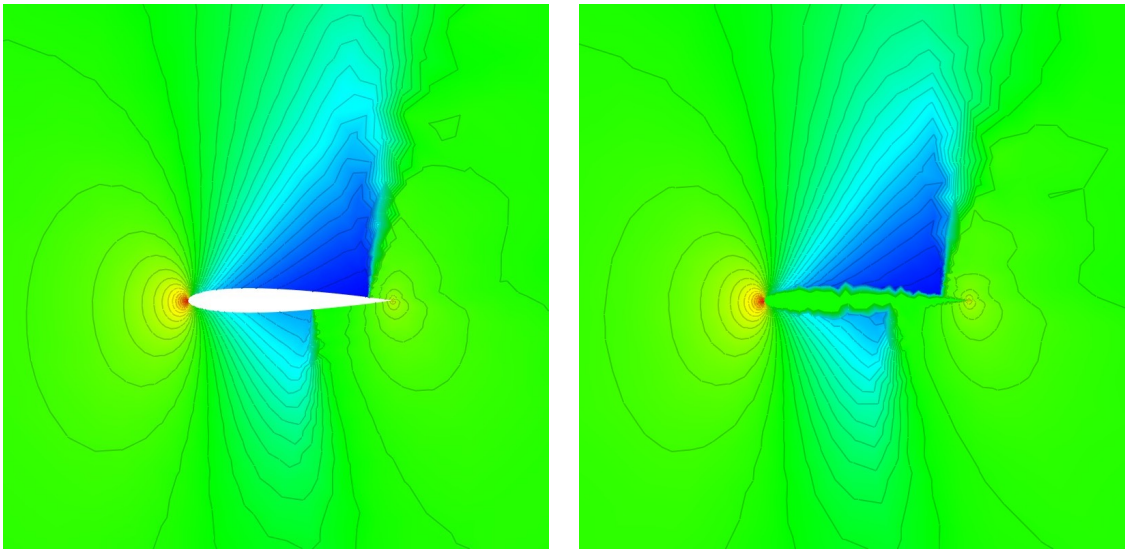

Figure 2.18 – Body-fitted (left) and embedded (right) simulation of a transonic flow around a NACA0012 airfoil.

Supersonic case. The last case is a supersonic flow (Mach 2) around a NACA0012 airfoil with an angle of attack of 0 degree. For this case, the solver is stopped after 500 iterations. The minimal time step of the flow solver in the body-fitted case is of 0.000185 whereas it is of 0.000037 in the embedded case. This is again consequence of the cut-cell performed in the preprocessing step.

The obtained results are quite similar in trends (see Figure 2.20) and this can be confirmed by the C_p (Figure 2.19) curve around the airfoil. Again, an oscillation is

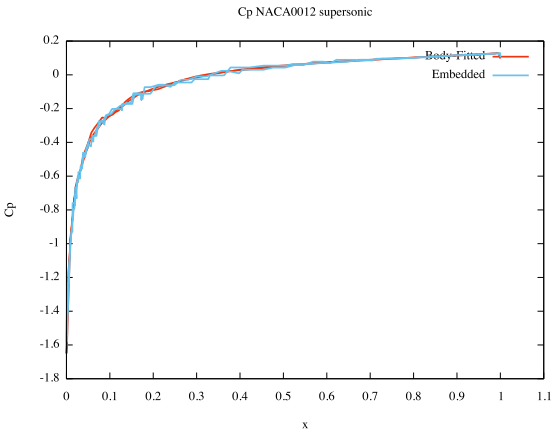

Figure 2.19 – Computed C_p of a supersonic flow around a NACA0012 airfoil.

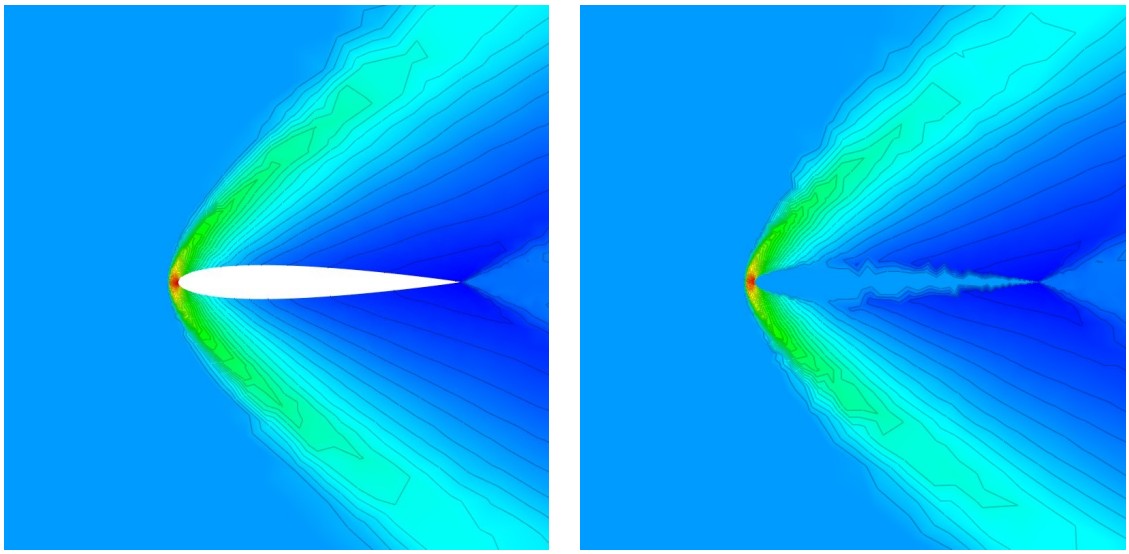

Figure 2.20 – Body-fitted (left) and embedded (right) simulation of a supersonic flow around a NACA0012 airfoil.

seen with embedded case as well as a greater dissipation of the supersonic shockwaves. Note that a relative error of 0.1% is obtained for the drag computation.

In all the cases, the global trend of solution and contour lines are quite similar even if a lot of spurious oscillations occur all along the C_p curve which leads to high differences in drag and lift computations. Also, other problems can be listed. First, the boundary of the airfoils is not well represented. Second, the point distribution of the mesh used for the embedded simulations is done to make sure something is captured by the tail of the airfoil in the first place, but it requires to know a bit the embedded object in the mesh generation process. One solution to overcome both of these issues is to use the iterative process of mesh adaptation. This will be done in one of the following sections.

Supersonic notional missile

In the three-dimensional case, we simulate a supersonic (Mach 1.6) flow around a notional missile (see Figure 2.21) with a zero incidence (angle of attack of 0 degrees).The main difficulty with this geometry is the very thin winglets that are challenging for the embedded method. The used meshes with the results are shown in Figures 2.22 and 2.23. The displayed solution field is the flow density. An implicit advance in time is used. To compute these flows, an HLLC solver is again used along with a V6 scheme and a Dervieux limiter. The studied boundary condition on the airfoil is a slipping boundary condition. Two meshes of 90 078 and 784 847 vertices for the embedded case and a mesh of 199 342

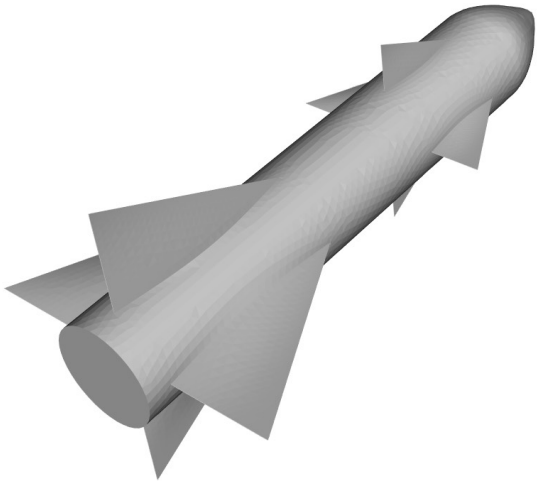

Figure 2.21 – Geometry of the studied notional missile.

vertices for the body-fitted case are considered. The embedded geometry is composed of 10 707 vertices in both cases. An analysis of the meshes shows a difference in the mesh density. This difference is explained because small sizes have to be satisfied near the missile in the body-fitted case whereas, the generated mesh in the embedded case does not have *a priori* knowledge of the location of the object. The analysis of the obtained solution shows that a trend of the global behavior is visible but the embedded approach does not capture the physics due to the lack of accuracy in the final representation and even if the used mesh is relatively fine (784 847 vertices). By analyzing the result of the detection by the CFD solver, we notice that in both cases the shape of the missile is totally lost, especially its winglets. The use of mesh adaptation seems inevitable to recover the geometry in the embedded case without huge costs in preprocessing. Note that like in 2D, the treatment of the embedded geometry with this mesh requires a CPU time of the same order as one iteration of the flow solver. Finally, the analysis of the time-steps in this case is not relevant as the size maps are not consistent between the embedded and the body-fitted case.

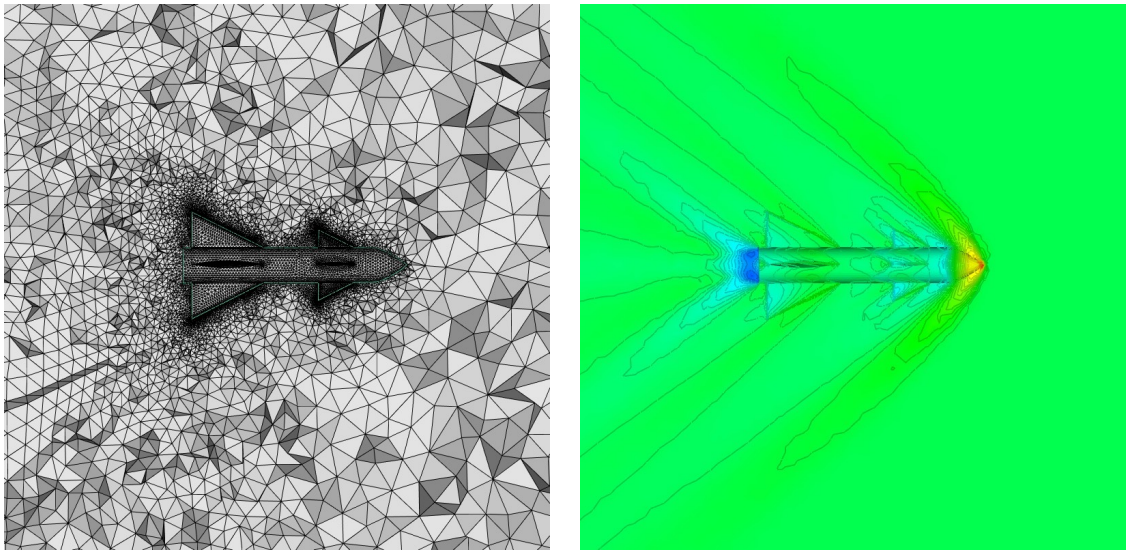

Figure 2.22 – Supersonic flow around a notional missile. Mesh (left) and density solution field (right) for the body-fitted simulation

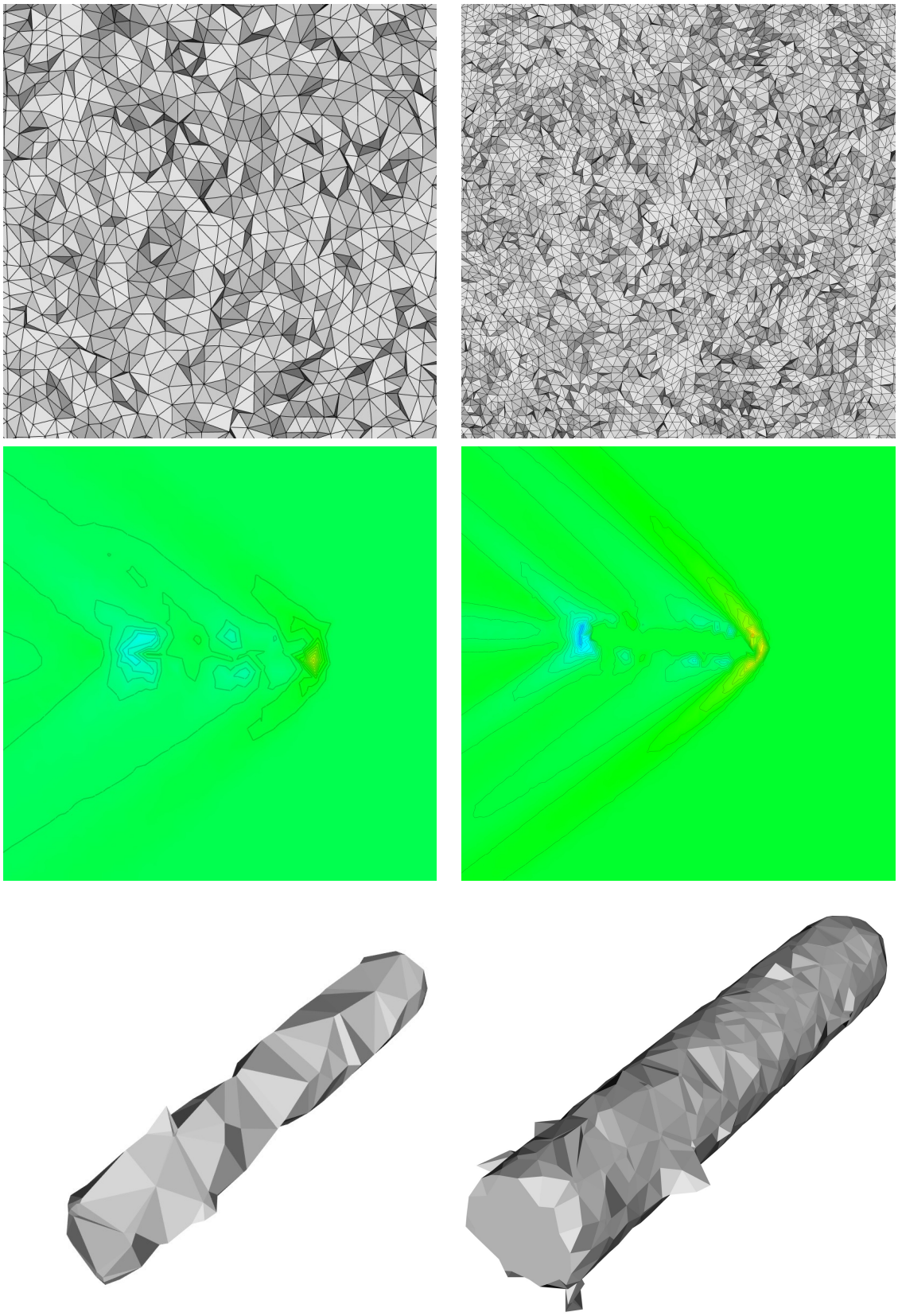

Figure 2.23 – Supersonic flow around a notional missile. On the left, embedded simulation on a coarse mesh of 90 078 vertices and on the right, simulation on a mesh twice finer of 784 847 vertices. Top, volume meshes used, middle, solutions computed on these meshes and bottom, the geometry as it is seen by the CFD solver in both cases.

2.5.3 Unsteady flow simulations with fixed geometry

Using the same geometrical approach as with steady flows, a blast simulation has been done as unsteady flow simulation. The considered domain is a square of size $[-10, 10] \times [-10, 10]$ with sixteen circle of radius 0.1. The blast is ignited like a Sod shock problem inside a circle of radius 0.5 located in the middle of the square: the initial (dimensionless) density is of 1 inside the circle and 0.125 outside and the initial (dimensionless) pressure is of 1 inside the circle and 0.1 outside. For the resolution of the unsteady Euler equations, an HLLC solver is used and is coupled with an explicit advance in time by means of a Runge-Kutta scheme of order 2. Wall boundary conditions are applied on the edge of the domain and slipping boundary condition is applied on the circles inside the domain. Finally, the total physical time of the blast is of 6s. The body-fitted mesh contains 4418 vertices and the mesh used for the embedded simulation is composed of 5050 vertices. Note that the mesh of the embedded case is non uniform and adapted to the position of the embedded circles. The results are shown in Figure 2.25 where the used meshes and two snapshots at $t = 1.5$ and $t = 4.0$ are given in both embedded and body-fitted cases. With only a dozen points covered by each embedded circle, the solver is able to detect them and apply the boundary condition properly like in the body-fitted case. A zoom around one of the circles is done in Figure 2.24. In the embedded case the CPU time required in preprocessing is approximately of the same time of one solver iteration and is clearly negligible beyond the total time required for the simulation. In the body-fitted case, the simulation requires 412 iterations with time steps that are approximately of order 10^{-2} and in the embedded case, the simulation requires 352 iterations with time steps that are approximately of order 10^{-2}. In other words, the embedded simulation has an equivalent behavior as its body-fitted counterpart.

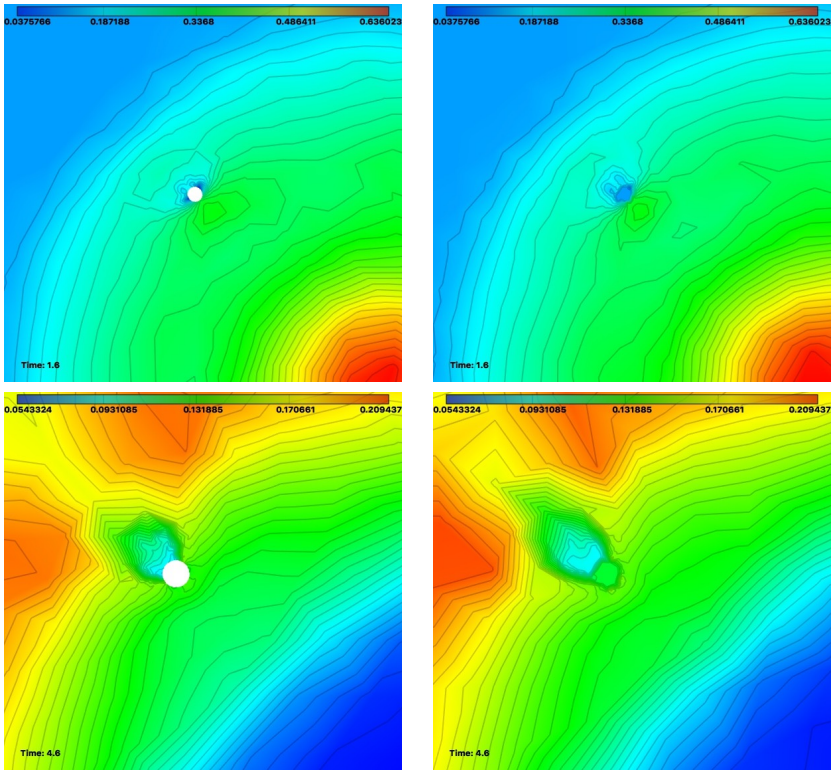

Figure 2.24 – Zoom of the density of field around one of the circles involved in the blast of Figure 2.25. Left, the body-fitted case and right, the embedded case. The solutions are at two different times: $t = 1.8$ and $t = 4.6$.

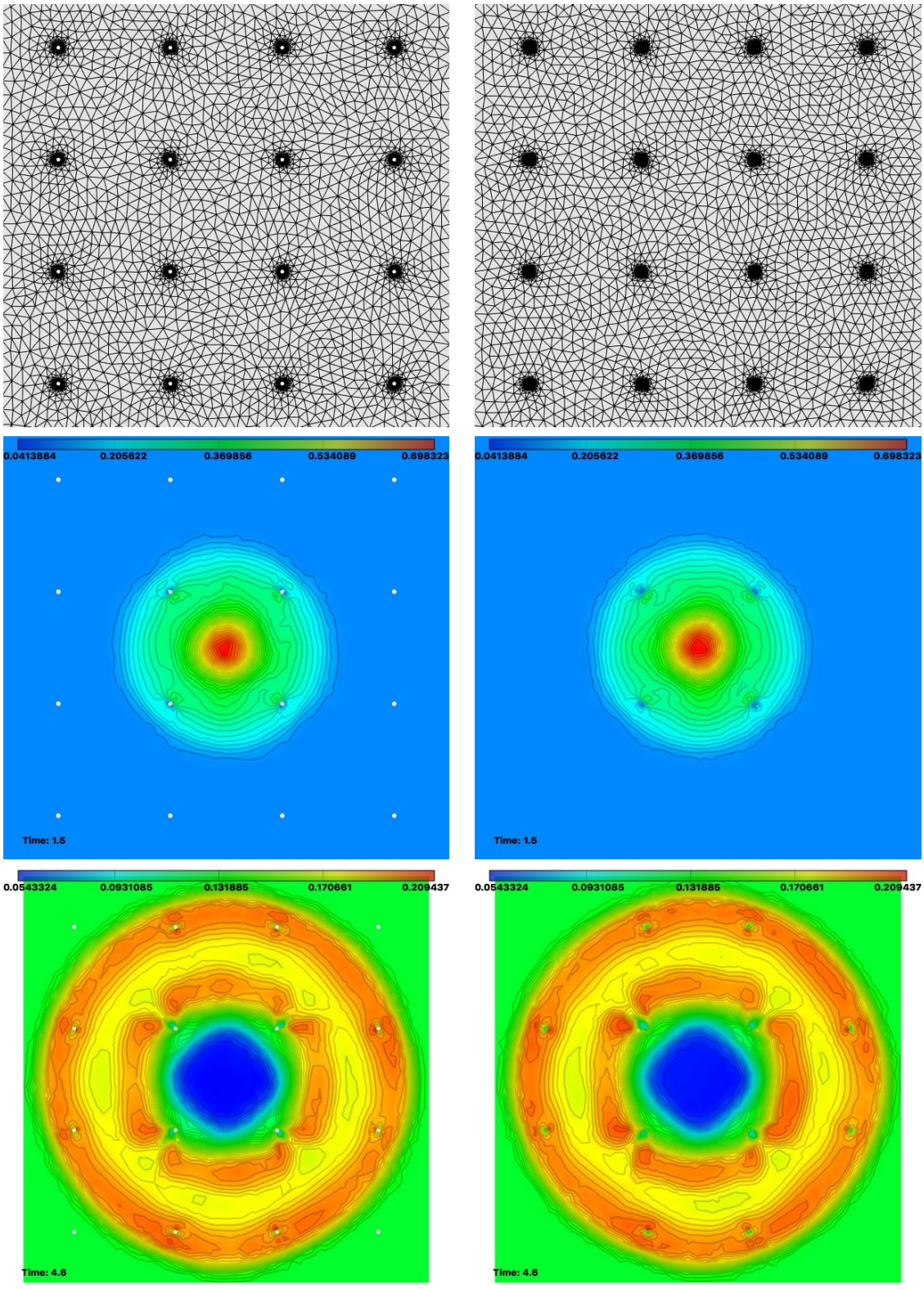

Figure 2.25 – Blast simulation with a fixed geometry. Left, the body-fitted case and right, the embedded case. Top, the considered meshes and thereafter, the associated solutions at two different times: $t = 1.8$ and $t = 4.6$ (density fields).

2.5.4 Mesh adaptation for steady flow simulations with embedded geometries

Steady flow simulations with embedded geometries captures naturally the embedded geometry in the mesh thanks to the boundary conditions. Indeed, if a vertex is detected as covered by the geometry, a constant value, the value at infinity, is assigned to this vertex. Most of the time, this creates a discontinuity (unless the flow is at rest) of the solution around the embedded geometry. Also in CFD, the computation of aerodynamic coefficients around geometries (drag, lift, ...) is of major interest as they quantify the quality of a design from an engineering point of view. Consequently, an accurate and reliable mesh that captures the geometry well is desirable. As the position of the geometry is not necessarily known in the first hand, mesh adaptation becomes highly interesting. The idea is to start from a uniform and/or coarse mesh and to apply the adaptation loop several times. Thanks to the gap naturally created around the geometry by the solver, the mesh is refined around the location of the embedded geometry. Once adaptation process is done, the geometric error due to embedding process is reduced, residual converges faster and solution looks really close to the body-fitted one.
On some of the previous cases, adaptation process has been performed. The general process is the following: for a given complexity, the flow solver is called 5 times and after each call, a feature-based metric using the sensor field is deduced (see Section 1.3). Using this and the input complexity, a new mesh is generated and the process continues. The method is applied in 2D on two different types of simulations around a NACA0012 airfoil : a supersonic and a subsonic flow as each of them highlights various phenomena involved in CFD. Then the three-dimensional test-case of the notional missile is performed in the supersonic case. In all cases, the sensor used for the mesh adaptation is the local mach number.

Subsonic and supersonic NACA0012

For the case of the NACA0012 airfoil, the considered complexities are 1 000, 2 000, 4 000 8 000, and 16 000. In these simulations, the studied flow and the solver characteristics are exactly the same as in Section 2.5.2. The results are given in Figures 2.26 and 2.27 where the last results at complexity 16 000 are shown as well as the comparison of both curves and the evolution of the computed drag all along the adaptation process.
In both cases, the observations are the same: the adaptation process captures the same physical features in the body-fitted case and in the embedded one. This is confirmed by the plot of C_p curves which shows no real difference between both curves. It is also corroborated by the plot of the computed drag all along the process which shows that both body-fitted and embedded case converge to the same result. Moreover, for the case of the embedded simulation, it is clear that the geometry is quickly and well captured in the adaptation process. This explains notably that for a given complexity, the computed drag seems more accurate than the body-fitted one as larger number of degrees of freedom lies on the boundary.

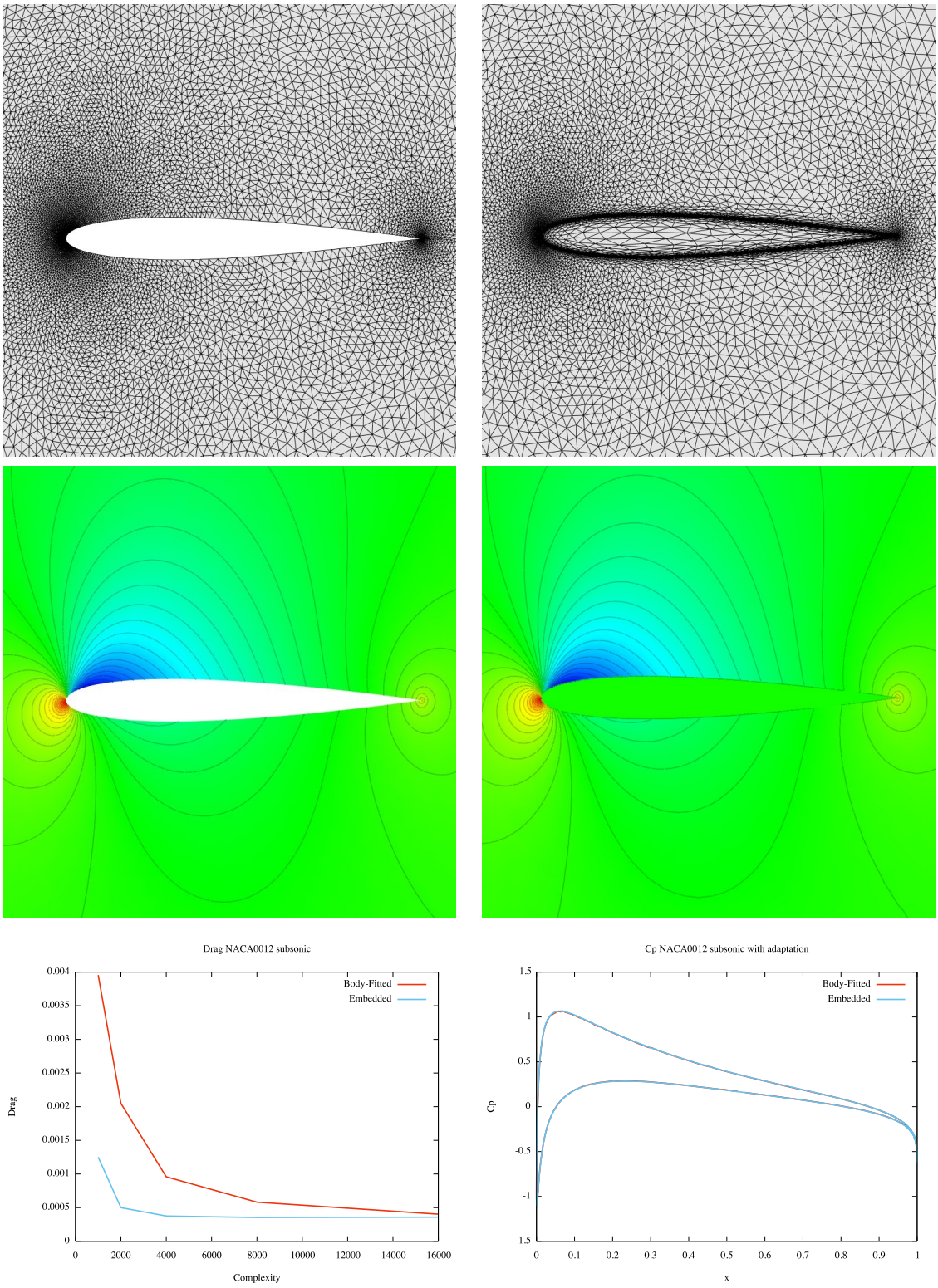

Figure 2.26 – Subsonic flow around NACA0012 airfoils. Left, the adapted body-fitted case. Right, the adapted embedded case. Top, the adapted meshes obtained at the last complexity. Middle, the associated solutions density fields. Bottom, the evolution of the drag all along the adaptation process (left) and the C_p curves at the last complexity (right).

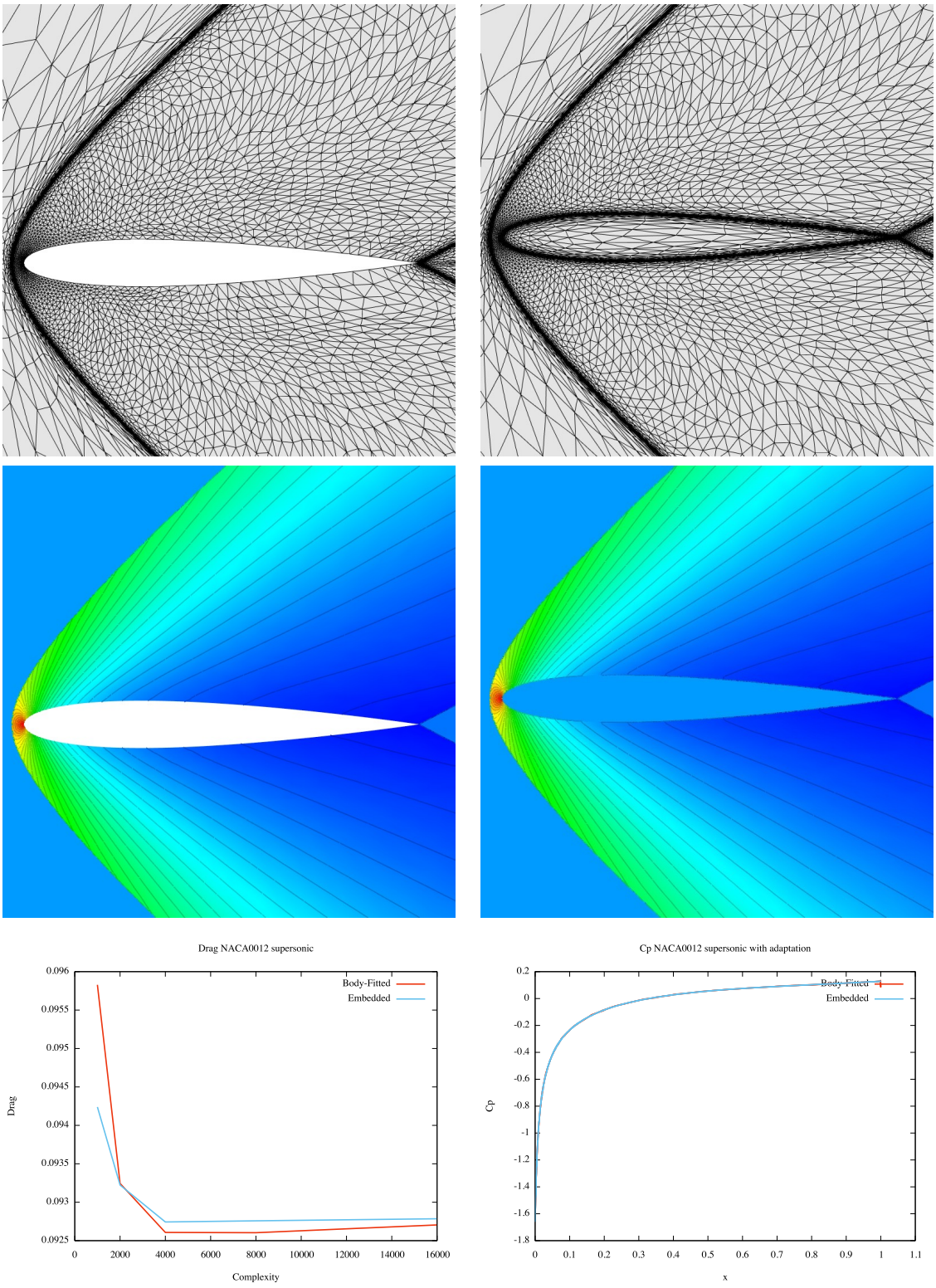

Figure 2.27 – Supersonic flow around NACA0012 airfoils. Left, the adapted body-fitted case. Right, the adapted embedded case. Top, the adapted meshes obtained at the last complexity. Middle, the associated solutions density fields. Bottom, the evolution of the drag all along the adaptation process (left) and the C_p curves at the last complexity (right).

Supersonic notional missile

For the case of the notional missile, the used complexities are 1 000, 2000, 4 000 8 000, 16 000, 32 000, 64 000 and 128 000. In this process, the studied flow and the solver characteristics are exactly the same as in Section 2.5.2. The numerical results are given in Figures 2.29 and 2.30 where the last results at complexity 128 000 are shown with two views. In both cases, the adaptation process captures the same physical features in the body-fitted case and in the embedded case. In the last case, the embedded geometry is well captured by the CFD solver as it is shown in Figures 2.31 and 2.32, where the detected geometry is shown after 5 iterations at each of the studied complexities. This process is able to recover even the small features of the object like the winglets as it is shown with the zoom provided.
Finally, a convergence study in L^2 norm (using the result at complexity 128 000 as reference) has been performed to compare the behavior of the numerical scheme when using the embedded approach with respect to the body-fitted counterpart. The results are given in Figure 2.28 and show that the approach damages the order of convergence of the method. This shows one of the limitations of the embedded approach. This can be explained by several factors: the local modification of the dual mesh, the lack of accuracy of the input geometry that is defined with a P^1 mesh, ...

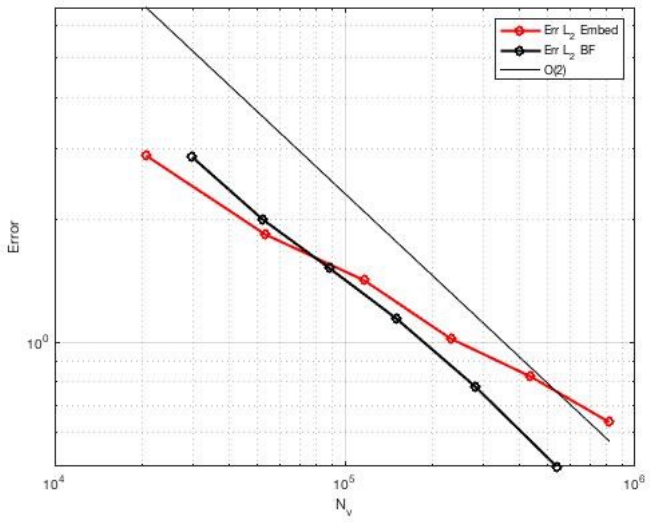

Figure 2.28 – Convergence study in L^2 norm of the mesh adaptation process in the case of the supersonic notional missile. Embedded (red line) and body-fitted (black line) approaches are shown.

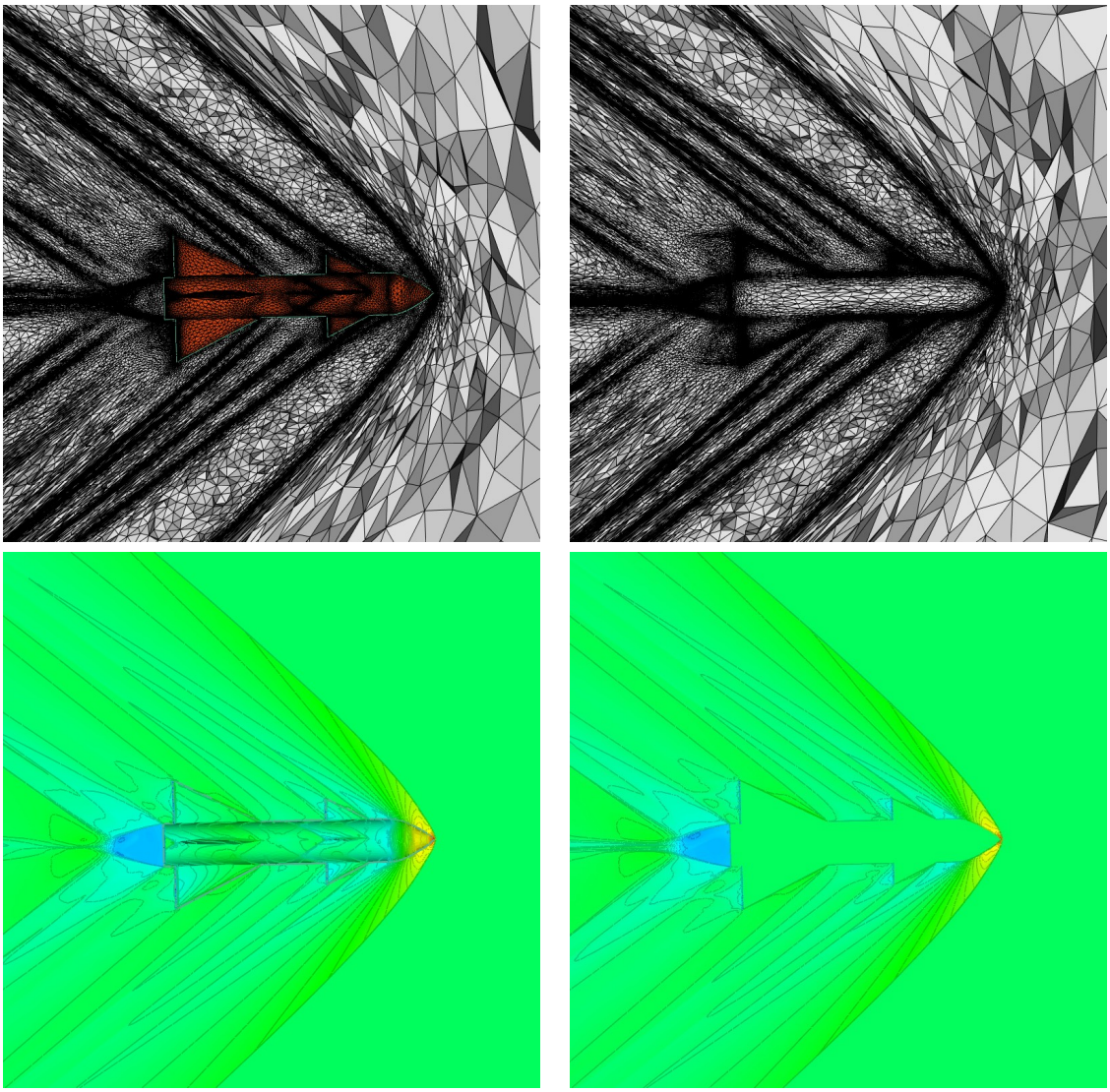

Figure 2.29 – Supersonic flow around a notional missile. Left, adapted body-fitted case. Right, adapted embedded case. View along $y = 0$ axis. Top, the adapted meshes. Bottom, the associated solutions.

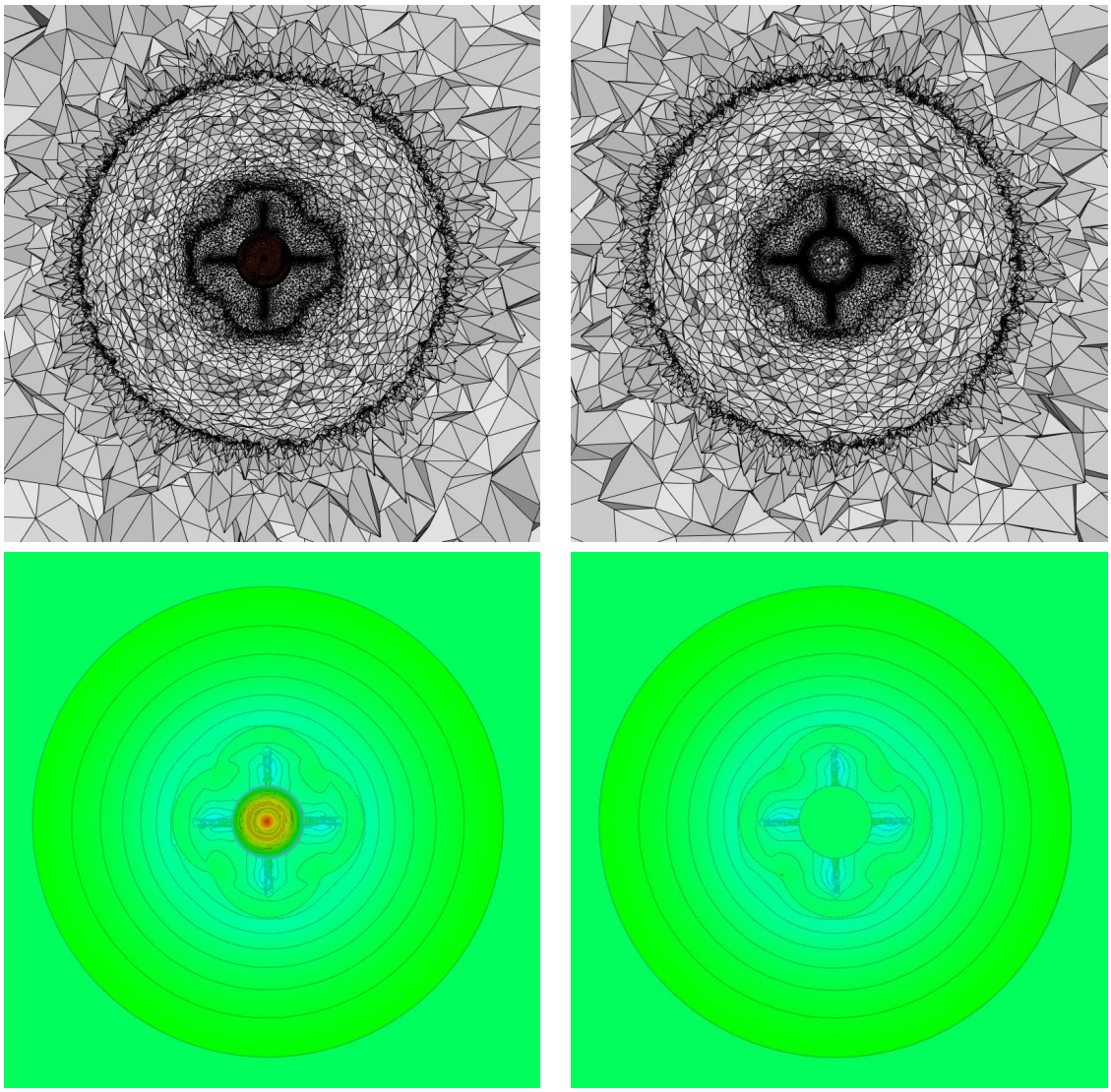

Figure 2.30 – Supersonic flow around a notional missile. Left, adapted body-fitted case. Right, adapted embedded case. View along $x = 0$ axis. Top, the adapted meshes. Bottom, the associated solutions.

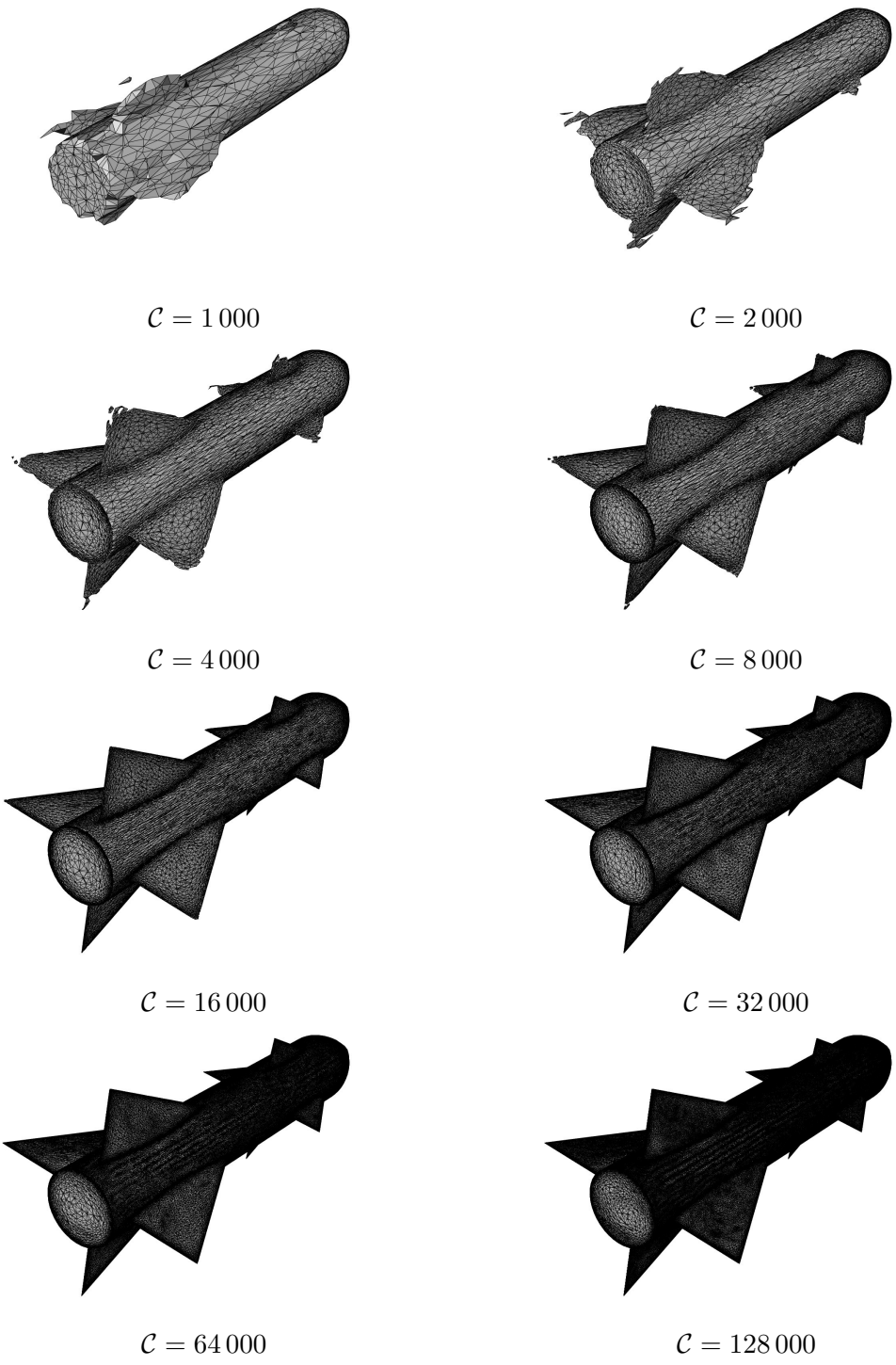

Figure 2.31 – Evolution of the detection of the embedded geometry by the CFD solver all along the mesh adaptation process. Each of the 8 cases shows the result obtained at the last iteration of the adaptation process for a given complexity.

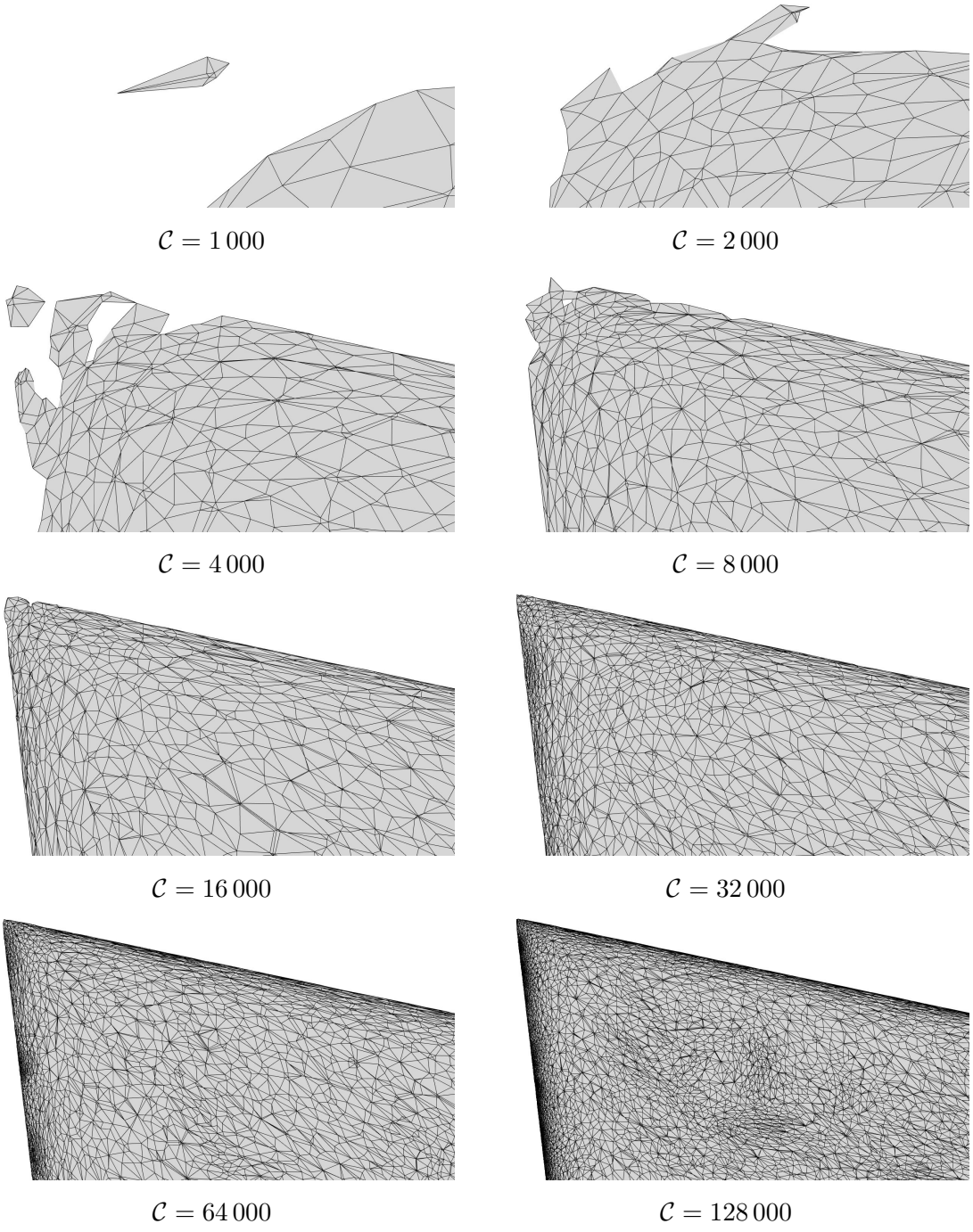

Figure 2.32 – Zoom around the top winglet of the embedded geometries detected by the solver in Figure 2.31. Each of the 8 cases shows the result obtained at the last iteration of the adaptation process for a given complexity.

2.5.5 Conclusion

In this chapter, we have shown the implementation of the embedded boundary method in the context of a vertex-centered finite volume method applied to the Euler equations. Albeit an inevitable loss of precision, this method works fine in 2D and 3D and is able to predict a phenomenon around a fixed object without the meshing cost of a potentially complex object. The implementation of this method is fairly easy and consists primarily in a preprocessing step and in the implementation of a specific boundary condition. In particular, this does not affect the global behavior of the CFD solver in terms of CPU load. The method works as is on both steady and unsteady simulations as long as the embedded object does not move and does not require the embedded mesh to be valid in the finite element sense even if the geometry needs to be water-tight in a way or another. The coupling of this method with mesh adaptation thus enables to retrieve the aerodynamic data of interest with a comparable accuracy than in the body-fitted case and without the requirement of a specific implementation during the mesh adaptation process.

Conclusion

In this part, a review of the standard and mature mesh adaptation process has been performed. Then, a study of the embedded boundary method applied to the Euler equations has been made. The implementation of this method has been validated and has been shown to give the expected results but with a loss of precision in terms of aerodynamic data. To this end, the coupling with mesh adaptation has been performed and has evidenced that its use enables to get back the lost accuracy in the first place. This work has been published as a research report [Feuillet et al. 2019b].
Some perspectives can be de drawn regarding this work. The first idea would be to create a level-set metric to improve the detection of the embedded geometry as it is performed in [Quan et al. 2014a, Hachem et al. 2013]. This way, it would improve the mesh adaptation to converge quicker to the geometrical shape, in particular for the 3D case. Then, facing the issues encountered when dealing with coarse linear discretizations of the embedded geometry, it would be interesting to directly consider the exact intersection of the mesh with the analytical model or with a high-order surrogate model. A study of such models is presented in Chapter 5. Finally, it would be a piece of interest to consider the generalization of this work to the Navier-Stokes equations, and in particular RANS equations.
Otherwise, it is important to point out that the interest of this work seems more limited than expected. Indeed, when the question of dealing with moving embedded geometries in 2D occurs, some issues appear:

- The implementation of the static case relies on local modifications of the Finite Volume Cell using a cut-cell approach so that the embedded boundary is well represented within the dual mesh. The underlying idea is that if a point is covered by the geometry then there is no sense in considering its nodal value and its associated Finite Volume cell thus has to be removed from the uncovered part of the mesh. However, by doing this with embedded moving geometries, it comes down to make locally move the cell interfaces from one iteration to another. To properly to take this into account, it requires the use of an ALE solver whose cost is way higher than a standard one.
- An attempted approach was thus to perform at the next iteration an exact cut-cell with the median cells of the back mesh (*e.g.* to consider the effective result of the intersection of the dual mesh with the back mesh), to perform the advance in time using a Runge-Kutta scheme and then to merge the cells so that we get back to the static case as in [Yang et al. 1997a, Yang et al. 1997b].
- However by doing this, it forces the CFL number to stay at really low values as it is mandatory for an existing cell at t^n and that vanishes at t^{n+1} to have a non-null area at the exact cut-cell step at t^{n+1}. In other words, it seems impossible to have a CFL number greater than 0.5.
  In every case, this drastically reduces the interest of considering moving objects as, based on this previous observation, it seems highly complicated to deal with both multi-step Runge-Kutta and implicit schemes. Finally, the use of unsteady mesh adaptation seems inappropriate as the smallest size of the mesh will rule the global time step using a CFL number that has to stay small.

Using these observations the development of the embedded approach on a vertex-centered Finite Volume scheme is not as advantageous as expected. One of the initial objectives of this study was indeed to use the embedded approach to simulate complex multi-bodies

moving problems as it occurs in fragmentation problems.

Part II

High-order

Introduction

For years, the resolution of numerical methods has consisted in solving Partial Derivative Equations by means of a piecewise linear representation of the physical phenomenon on linear meshes. This choice was merely driven by computational limitations. With the increase of the computational capabilities, it became possible to increase the polynomial order of the solution while keeping the mesh linear. This was motivated by the fact that even if the increase of the polynomial order requires more computational resources per iteration of the solver, it yields a faster convergence of the approximation error [2] [Vanharen 2017] and it enables to keep track of unsteady features for a longer time and with a coarser mesh than with a linear approximation of the solution. However, in [Ciarlet and Raviart 1972, Lenoir 1986], it was theoretically shown that for elliptic problems the optimal convergence rate for a high-order method was obtained with a curved boundary of the same order and in [Bassi and Rebay 1997], evidence was given that without a high-order representation of the boundary the studied physical phenomenon was not exactly solved using a high-order method. In [Zwanenburg and Nadarajah 2017], it was even highlighted that, in some cases, the order of the mesh should be of a higher degree than the one of the solver. In other words, if the used mesh is not a high-order mesh, then the obtained high-order solution will never reliably represent the physical phenomenon. [3] This issue is illustrated in Figure 3.0. On the left, the Euler equations are solved around a NACA0012 airfoil at the degree 2 on a linear mesh whereas on the right, they are solved on a quadratic (degree 2) mesh. The flow density is shown with it isolines. On the linear mesh, the CFD solver detects that the mesh is piecewise linear and resolves the Euler equations on the polygonal surface [4]. This is an issue as the geometry of the NACA0012 is not piecewise linear. On the contrary, the definition of the boundary at order 2 gives a more reliable representation of the boundary of the airfoil to CFD which takes into account the fact that the geometry is curved during the solving process.

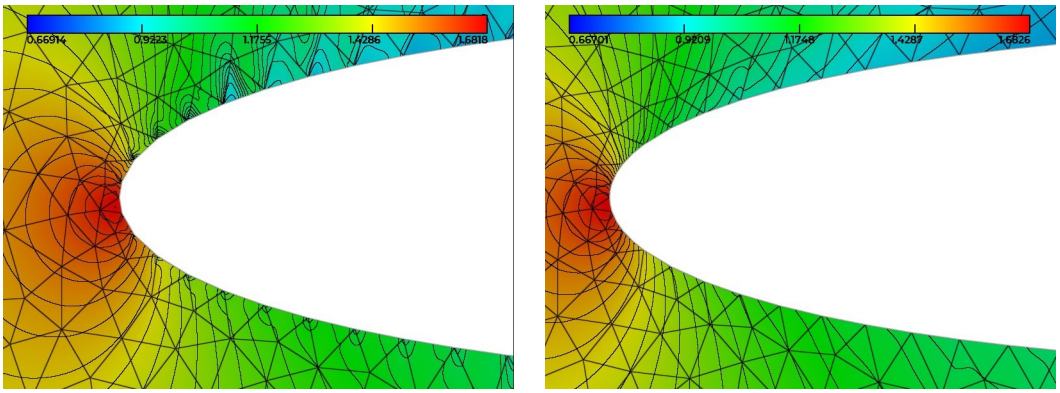

Figure 3.0 – Degree 2 resolution of the Euler equations on a NACA0012 airfoil with a linear mesh (left) and with a quadratic mesh (right). Courtesy of C. Peyret (ONERA).

Based on these issues, the development of high-order mesh generation procedures appears

2. The order of convergence is the degree of the polynomial approximation plus one.

3. Note that other strategies, like IGA [Hughes et al. 2005] or NEFEM [Sevilla et al. 2008], that are based on the NURBS, the functions defining the most common CAD systems, exist as an alternative/complement to high-order methods.

4. This is exactly the same issue that was pointed out in Section 2.5.1 of Chapter 2.

mandatory. To generate high-order meshes, several approaches exist. The first approach was tackled twenty years ago [Dey et al. 1999] for both surface and volume meshing. At this moment the idea was to use all the meshing tools to get a valid high-order mesh. The same problem was revisited a few years later in [Sherwin and Peiró 2002] for bio-medical applications. In these first approaches and in all the following, the underlying idea is to use a linear mesh and elevate it to the desired order. Some make use of a PDE or variational approach to do so [Abgrall et al. 2014b, Persson and Peraire 2009, Fortunato and Persson 2016, Moxey et al. 2016, Turner et al. 2016, Xie et al. 2013, Hartmann and Leicht 2016], others are based on optimization and smoothing operations and start from a linear mesh with a constrained high-order curved boundary in order to generate a suitable high-order mesh [Karman et al. 2016, Gargallo-Peiró et al. 2013, Toulorge et al. 2013]. Also, when dealing with Navier-Stokes equations, the question of generating curved boundary layer meshes (also called viscous meshes) appears. Most of the time, dedicated approaches are set-up to deal with this problem [Moxey et al. 2015b, Karman 2019]. In all these techniques, the key feature is to find the best deformation to be applied to the linear mesh and to optimize it. The prerequisite of these methods is that the initial boundary is curved and will be used as an input data. A natural question is consequently to study an optimal position of the high-order nodes on the curved boundary starting from an initial linear or high-order boundary mesh. This can be done in a coupled way with the volume [Ruiz-Gironès et al. 2016a, Toulorge et al. 2016] or in a preprocessing phase [Ruiz-Gironès et al. 2015, Ruiz-Gironès et al. 2016b]. In this process, the position of the nodes is set by projection onto the CAD geometry or by minimization of an error between the surface mesh and the CAD surface. Note that the vertices of the boundary mesh can move as well during the process. In the case of an initial linear boundary mesh with absence of a CAD geometry, some approaches based on normal reconstructions can be used to create a surrogate for the CAD model [Vlachos et al. 2001, Ims and Wang 2019]. Finally, a last question remains when dealing with such high-order meshes: Given a set of degrees of freedom, is the definition of these objects always valid ?. Until the work presented in [George and Borouchaki 2012, Johnen et al. 2013, George et al. 2016], no real approach was proposed to deal in a robust way with the validity of high-order elements. The novelty of these approaches was to see the geometrical elements and their Jacobian as Bézier entities. Based on the properties of the Bézier representation, the validity of the element is concluded in a robust sense, while the other methods were only using a sampling of the Jacobian to conclude about its sign without any warranty on the whole validity of the elements.

The use of both high-order methods and meshes also unveils a problem regarding the visualization techniques hitherto used. Indeed, the standard approaches are specifically tailored to display linear solution on linear meshes as it is what the hardware naturally process. A common strategy [Schroeder et al. 2006, Remacle et al. 2007, Maunoury et al. 2018] consists in performing a preprocessing step that creates both visualization mesh and solution fitted to the variations of the high-order solution and geometry so that it gives a proper rendering of a high-order solution while keeping standard rendering techniques. On the contrary, strategies specifically tailored for high-order finite elements can be set up [Peiró et al. 2015]. However, the cost of these methods in terms of CPU time and memory footprint is high.

In this part, we will mainly deal with high-order meshes. In Chapter 3, we set the basis of the framework used in this thesis to manipulate high-order meshes and solutions. In particular, the analogy between high-order and Bézier elements is highlighted. Afterwards, Chapter 4 is specifically devoted to the development of high-order error estimates suitable for parametric high-order surface mesh generation. Then, in Chapter 5, in the particular

case of quadratic meshes, we focus on the generalization of mesh optimization operators and their applications to curved mesh generation, moving-mesh methods and boundary layer mesh generation. Finally, Chapter 6 tackles the subject of the visualization of high-order meshes and solutions.

Chapter 3

High-order meshes: generalities

3.1 Introduction

In this chapter, we set the basis of the proposed framework to consider high-order meshes. This is an extension of the framework proposed in [George and Borouchaki 2012, George et al. 2016, Borouchaki and George 2017b, George et al. 2019] and also considered in [Johnen et al. 2013, Toulorge et al. 2013, Abgrall et al. 2014b]. The core concept of this framework is to draw a correspondence between high-order (Lagrange) and Bézier elements. To this end, both notions are detailed in Sections 3.2 and 3.3. Subdivision procedures for Bézier elements are exposed in order to determine the validity of their high-order element formulations. Once the parallel is drawn, some notions regarding high-order meshes are considered. In particular, in Section 3.4, the quality of a high-order mesh is discussed. After a review of the existing quality functions in the meshing community, two quality measures are proposed. The first one that is used in Chapter 5 for mesh optimization purposes and the second one that aims at being a successor of those used in P^1 mesh adaptation.

3.2 Bézier representation of the elements

This section deals with the Bézier representation of the elements. This notion was first introduced by Bézier and de Casteljau in the 1950s for CAD systems. More detailed information can be found in [Bézier 1986] and [de Casteljau 1959]. Here, the intent is to show some general results about Bézier elements that will be reused after. In particular, these results will be reused in the context of high-order finite elements.

3.2.1 A preliminary result

In this section, we detail a result that will be used in the next sections. In [Ciarlet 1978, Ciarlet Jr and Lunéville 2009], it is shown that for any simplex K of dimension d:

$$\int_K \prod_{i=0}^{d} \lambda_i^{\alpha_i} \mathrm{d}\Omega = \frac{\prod_{i=0}^{d} \alpha_i!}{(d + \sum_{i=0}^{d} \alpha_i)!} d!|K|, \tag{3.1}$$

where λ_i are the barycentric coordinates of the simplex and $\alpha_i \in \mathbb{N}$.
When $d = 1$, the simplex is an edge and if the edge is defined with $u \in [0, 1]$, $\lambda_0 = u$, $\lambda_1 = 1 - u$ and $|K| = 1$. This means that:

$$\int_0^1 u^i (1-u)^j \mathrm{d}u = \frac{i!j!}{(1+i+j)!}. \tag{3.2}$$

When $d = 2$, the simplex is a triangle and if we consider $\widehat{K} = \{0 \leq u, w, w \leq 1 \mid u+v+w = 1\}$, then $\lambda_0 = u$, $\lambda_1 = v$, $\lambda_2 = w$ and $|\widehat{K}| = \frac{1}{2}$. This means that:

$$\int_{\widehat{K}} u^i v^j w^k \mathrm{d}\sigma = \frac{i!j!k!}{(2+i+j+k)!}. \tag{3.3}$$

When $d = 3$, the simplex is a tetrahedron and if we consider $\widehat{K} = \{0 \leq u, w, w, t \leq 1 \mid u+v+w+t = 1\}$, then $\lambda_0 = u$, $\lambda_1 = v$, $\lambda_2 = w$, $\lambda_3 = t$ and $|\widehat{K}| = \frac{1}{6}$. This means that:

$$\int_{\widehat{K}} u^i v^j w^k t^l \mathrm{d}\Omega = \frac{i!j!k!l!}{(3+i+j+k+l)!}. \tag{3.4}$$

3.2.2 Bézier curve

Definition

For a given set $(P_i)_{0\leq i\leq n}$ of $n+1$ points, a Bézier curve of degree n is defined using the following algebraic relation:

$$\gamma(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} P_i, \quad \forall t \in [0,1],$$

with $\binom{n}{i} = \frac{n!}{i!(n-i)!}$. In this context, P_i are called control points and their dimensions are arbitrary. Note that $\gamma(0) = P_0$ and $\gamma(1) = P_n$. More precisely, a Bézier curve of degree n relies on the Bernstein polynomials of degree n. These polynomials are defined by:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad \forall t \in [0,1],$$

with $i \in [\![0, n]\!]$. This curve is C^∞. However, for a large n, the curve is sometimes oscillating (like any other polynomial curve). Also, these curves poorly represent commonly used curves like conics. To deal with theses issues, another definitions, other type of curves are considered like rational Bézier curves or composite curves (like B-Splines and NURBS) but the study of such curves is beyond the scope of this thesis.

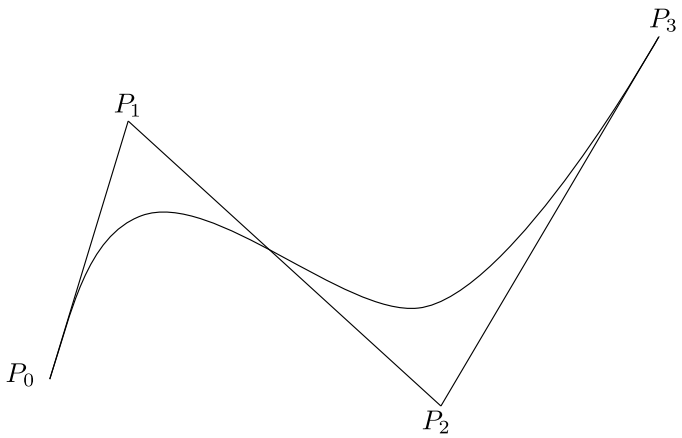

Figure 3.1 – A Bézier curve of degree 3.

About Bernstein polynomials

Bernstein polynomials do have several properties (a review of these properties is done in various references such as [Frey and George 2008]):

i) They are positive.

ii) They realize a partition of unity (Cauchy identity).

$$\forall t \in [0,1] \quad \sum_{i=0}^{n} B_i^n(t) = 1.$$

iii) Symmetry : $\forall t \in [0,1], \quad B_i^n(t) = B_{n-i}^n(1-t)$.

iv) They satisfy the following recursion between the degrees:

$$\begin{aligned} B_0^{n+1}(t) &= (1-t)B_0^n(t), \\ B_i^{n+1}(t) &= tB_{i-1}^n(t) + (1-t)B_i^n(t), \quad \forall i \in [\![1,n]\!], \\ B_{n+1}^{n+1}(t) &= tB_n^n(t). \end{aligned} \tag{3.5}$$

And conversely,

$$B_i^n(t) = \frac{i+1}{n+1}B_{i+1}^{n+1}(t) + \frac{n+1-i}{n+1}B_i^{n+1}(t).$$

v) Their derivatives are deduced using the following formula (for $n \geq 1$):

$$\begin{aligned} (B_0^n)'(t) &= -nB_0^{n-1}(t), \\ (B_i^n)'(t) &= n(B_{i-1}^{n-1}(t) - B_i^{n-1}(t)), \quad \forall i \in [\![1,n-1]\!], \\ (B_n^n)'(t) &= nB_{n-1}^{n-1}(t). \end{aligned}$$

vi)

$$\forall i \in [\![0,n]\!], \quad \int_0^1 B_i^n(t)\mathrm{d}t = \frac{1}{n+1}.$$

vii) They form a basis of all polynomials of degree n or less defined in $[0,1]$ (*e.g.* the edge finite element space of degree n). Indeed, every term of the polynomial basis $\{1, t, .., t^n\}$ can be expressed using these polynomials:

$$\forall k \in [\![0,n]\!] \quad t^k = \sum_{i=k}^{n} \frac{\binom{i}{k}}{\binom{n}{k}} B_i^n(t).$$

Proofs :

i) This is straightforward as $0 \leq t \leq 1$, *e.g.* $(1-t) \geq 0$.

ii) Using the binomial theorem, it comes:

$$\forall t \in [0,1] \quad \sum_{i=0}^{n} B_i^n(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} = (t+1-t)^n = 1.$$

iii) This is straightforward with the formula of Bernstein polynomials.

iv) For $i = 0$, $(1-t)B_0^n(t) = (1-t).(1-t)^n = (1-t)^{n+1} = B_0^{n+1}(t)$.
For $i = n+1$, $tB_n^n(t) = t.t^n = t^{n+1} = B_{n+1}^{n+1}(t)$.
And $\forall i \in [\![1,n]\!]$,

$$\begin{aligned} tB_{i-1}^n(t) + (1-t)B_i^n(t) &= t.\binom{n}{i-1} t^{i-1}(1-t)^{n-i+1} + (1-t).\binom{n}{i} t^i (1-t)^{n-i} \\ &= \left(\binom{n}{i-1} + \binom{n}{i} \right) t^i (1-t)^{n+1-i} = \binom{n+1}{i} t^i (1-t)^{n+1-i} \\ &= B_i^{n+1}(t). \end{aligned}$$

Finally,

$$\begin{aligned} B_i^n(t) &= (t+1-t)B_i^n(t) \\ &= tB_i^n(t) + (1-t)B_i^n(t) \\ &= \binom{n}{i} t^{i+1}(1-t)^{n-i} + \binom{n}{i} t^i (1-t)^{n+1-i} \end{aligned}$$

$$= \frac{i+1}{n+1}\binom{n+1}{i+1}t^{i+1}(1-t)^{n-i} + \frac{n+1-i}{n+1}\binom{n}{i}t^i(1-t)^{n+1-i}$$
$$= \frac{i+1}{n+1}B_{i+1}^{n+1}(t) + \frac{n+1-i}{n+1}B_i^{n+1}(t).$$

v) First, we have:

$$(B_0^n)'(t) = \frac{\mathrm{d}}{\mathrm{d}t}((1-t)^n) = -n(1-t)^{n-1} = -nB_0^{n-1}(t),$$
$$(B_n^n)'(t) = \frac{\mathrm{d}}{\mathrm{d}t}(t^n) = nt^{n-1} = nB_{n-1}^{n-1}(t),$$

and then, $\forall i \in [\![1, n-1]\!]$

$$\begin{aligned}(B_i^n)'(t) &= \frac{\mathrm{d}}{\mathrm{d}t}(\binom{n}{i}t^i(1-t)^{n-i}) \\ &= i\binom{n}{i}t^{i-1}(1-t)^{n-i} - (n-i)\binom{n}{i}t^i(1-t)^{n-i-1} \\ &= n\binom{n-1}{i-1}t^{i-1}(1-t)^{n-i} - n\binom{n-1}{i}t^i(1-t)^{n-i-1} \\ &= n(B_{i-1}^{n-1}(t) - B_i^{n-1}(t)).\end{aligned}$$

vi) Using the Equation (3.2) of Section 3.2.1, it comes:

$$\int_0^1 B_i^n(t)\mathrm{d}t = \int_0^1 \binom{n}{i}t^i(1-t)^{n-i}\mathrm{d}t = \binom{n}{i}\frac{i!(n-i)!}{(1+n)!} = \frac{1}{n+1}.$$

vii) Let us expand the expression $\forall k \in [\![0, n]\!]$:

$$\begin{aligned}\sum_{i=k}^{n}\frac{\binom{i}{k}}{\binom{n}{k}}B_i^n(t) &= \sum_{i=k}^{n}\frac{\binom{i}{k}\binom{n}{i}}{\binom{n}{k}}t^i(1-t)^{n-i} &&= \sum_{i=k}^{n}\binom{n-k}{i-k}t^i(1-t)^{n-i} \\ &= \sum_{i=0}^{n-k}\binom{n-k}{i}t^{i+k}(1-t)^{n-k-i} &&= t^k\sum_{i=0}^{n-k}\binom{n-k}{i}t^i(1-t)^{n-k-i} \\ &= t^k.\end{aligned}$$

De Casteljau's algorithm

A convenient way to evaluate a point corresponding to a parameter t on a Bézier curve is the de Casteljau's algorithm. This algorithm is based on the following recursion:

$$\begin{aligned}&D_i^0(t) = P_i, \quad \forall i \in [\![0, n]\!], \\ &D_i^r(t) = (1-t)D_i^{r-1}(t) + tD_{i+1}^{r-1}(t), \quad \forall i \in [\![0, n-r]\!], \quad \forall r \in [\![1, n]\!].\end{aligned} \tag{3.6}$$

This recursion ends when $r = n$ and $D_0^n(t) = \gamma(t)$. Indeed, it is shown that for $r \in [\![0, n]\!]$,

$$D_i^r(t) = \sum_{j=0}^{r} B_j^r(t)P_{j+i}.$$

Proof: (by induction)
For $r = 0$, we clearly have $D_i^0(t) = P_i, \quad \forall i \in [\![0, n]\!]$
For $r \in [\![1, n]\!]$ and $i \in [\![0, n-r]\!]$,

$$D_i^r(t) = (1-t)D_i^{r-1}(t) + tD_{i+1}^{r-1}(t)$$

$$
\begin{aligned}
&= \sum_{j=0}^{r-1}(1-t)B_j^{r-1}(t)P_{j+i} + \sum_{j=0}^{r-1} tB_j^{r-1}(t)P_{j+i+1} \\
&= \sum_{j=0}^{r-1}(1-t)B_j^{r-1}(t)P_{j+i} + \sum_{j=1}^{r} tB_{j-1}^{r-1}(t)P_{j+i} \\
&= \sum_{j=1}^{r-1}\left[(1-t)B_j^{r-1}(t) + tB_{j-1}^{r-1}(t)\right]P_{j+i} + (1-t)B_0^{r-1}(t)P_i + tB_r^{r-1}(t)P_{r+i}.
\end{aligned}
$$

By using the recursion Formula (3.5), we find:

$$D_i^r(t) = \sum_{j=0}^{r} B_j^r(t)P_{j+i}.$$

Evaluating $D_i^r(t)$ with $r = n$ (which forces $i = 0$) provides $\gamma(t)$. The main advantage of this algorithm, is that it does not require any knowledge of Bernstein polynomials and that it is numerically more stable than a direct evaluation of the monomials as it is only linear combinations. In practice, the algorithm is **Algorithm 7**.

Algorithm 7: De Casteljau's evaluation algorithm for a Bézier curve of degree n

Function *DeCasteljauCurv*
- **Input:** Degree $n \in \mathbb{N}$, Control Points P_i, Parameter $t \in [0,1]$
- **Output:** val : value on the Bézier curve corresponding to the parameter t
- **if** $n = 0$ **then**
  - val $= P_0$
- **else**
  - **for** $i \in \{0, n-1\}$ **do**
    - $P_i^* = (1-t)P_i + tP_{i+1}$
  - val = DeCasteljauCurv($n-1$,P_i^*,t)

Note this algorithm can also be done in a non-recursive manner [Borouchaki and George 2017b] by applying directly the formula of recursion (3.6).

Subdivision of a Bézier curve

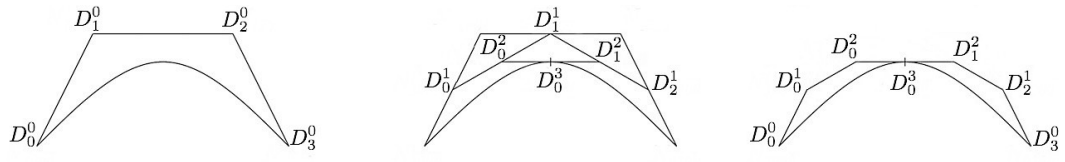

Figure 3.2 – Steps of the de Casteljau algorithm applied to a Bézier curve of degree 3 along with the induced subdivision in two.

As we will see in Section 3.3, the subdivision of a Bézier curve plays an important role to determine the validity of high-order elements. A subdivision of a Bézier curve is done using the intermediary points created by the de Casteljau's algorithm. Indeed, the

Algorithm 8: Subdivision algorithm for a Bézier curve of degree n

Function *SubdivisionCurv*
- **Input:** Degree $n \in \mathbb{N}$, Control Points P_i, Parameter $t \in [0,1]$
- **Output:** L : array containing the two sets of control points forming the subdivision
- **if** $n = 0$ **then**
  - L = [P_0,P_0]
- **else**
  - **for** $i \in \{0, n-1\}$ **do**
    - $P_i^* = (1-t)P_i + tP_{i+1}$
  - Lsub = SubdivisionCurv($n-1$,P_i^*,t)
  - L = [P_0; $Lsub$; P_n]

sequence of points $(D_0^r(t))_{0\leq r\leq n}$ and $(D_{n-r}^r(t))_{0\leq r\leq n}$ define control points for two Bézier curves which exactly represent the initial Bézier curve on respectively $[0,t]$ and $[t,1]$ (see Figure 3.2).

Proof:

We can parameterize the first curve with $u \in [0,1]$. The curve is written as:

$$\gamma_1(u) = \sum_{i=0}^{n} B_i^n(u) D_0^i(t).$$

If we expand γ_1, we have:

$$\begin{aligned}
\forall u \in [0,1], \quad \gamma_1(u) &= \sum_{i=0}^{n} B_i^n(u)\, D_0^i(t) = \sum_{i=0}^{n} B_i^n(u) \sum_{j=0}^{i} B_j^i(t) P_j \\
&= \sum_{j=0}^{n}\sum_{i=j}^{n} B_i^n(u)\, B_j^i(t) P_j = \sum_{j=0}^{n}\sum_{i=0}^{n-j} B_{n-i}^n(u)\, B_j^{n-i}(t) P_j \\
&= \sum_{j=0}^{n} P_j \sum_{i=0}^{n-j} \binom{n}{j}\binom{n-j}{i} (u)^{n-i} (1-u)^i t^j (1-t)^{n-i-j} \\
&= \sum_{j=0}^{n} \binom{n}{j} (ut)^j P_j \sum_{i=0}^{n-j} \binom{n-j}{i} (u(1-t))^{n-i-j} (1-u)^i \\
&= \sum_{j=0}^{n} \binom{n}{j} (ut)^j \left(u(1-t) + 1 - u\right)^{n-j} P_j.
\end{aligned}$$

This finally leads to:

$$\begin{aligned}
\forall u \in [0,1], \quad \gamma_1(u) &= \sum_{j=0}^{n} \binom{n}{j} (t)^j (1-u.t)^{n-j} P_j \\
&= \sum_{j=0}^{n} B_j^n(ut) P_j, \\
&= \gamma(ut).
\end{aligned}$$

The position in the initial curve is then deduced via $\tilde{t} = ut$ and it shows that it is the expression of the Bézier curve on $[0,t]$.

For symmetry reasons, the same reasoning applies to the second sub-curve, and it defines the initial curve on $[t, 1]$.
This property leads to the subdivision algorithm described in **Algorithm 8**.
The output of the algorithm is then an array containing the $2(n+1)$ control points defining the two sub Bézier curves. Like the de Casteljau's algorithm, this algorithm can also be done in a non-recursive manner [Borouchaki and George 2017b].

3.2.3 Bézier triangle

Definition

For a given set $(P_{ijk})_{0\leq i,j,k\leq n|i+j+k=n}$ of $\frac{(n+1)(n+2)}{2}$ points, a Bézier triangle (or Bézier triangular patch) of degree n is defined using the following algebraic relation:

$$\sigma(u,v,w) = \sum_{i+j+k=n} \frac{n!}{i!j!k!} u^i v^j w^k P_{ijk}, \quad \forall (u,v,w) \mid 0 \leq u,v,w \leq 1 \mid u+v+w=1.$$

(u, v, w) are called the barycentric coordinates and are linked to the reference coordinates $(\hat{x}, \hat{y}) \in [0, 1]^2$ via $(u, v, w) = (1 - \hat{x} - \hat{y}, \hat{x}, \hat{y})$. In this context, the P_{ijk} are called control points and their dimensions are arbitrary. Note that $\sigma(1, 0, 0) = P_{n00}$, $\sigma(0, 1, 0) = P_{0n0}$ and $\sigma(0, 0, 1) = P_{00n}$. Likewise, if (for instance) $w = 0$, then:

$$\begin{aligned}\sigma(u,v,0) &= \sum_{i+j+k=n} \frac{n!}{i!j!k!} u^i v^j 0^k P_{ijk} \quad \forall (u,v) \mid 0 \leq u,v \leq 1 \mid u+v=1. \\ &= \sum_{i=0}^{n} \frac{n!}{i!(n-i)} u^i (1-u)^{n-i} P_{i(n-i)0},\end{aligned}$$

which is the expression of a Bézier curve defined with $(P_{i(n-i)0})_{0\leq i\leq n}$. The same remark stands with $u = 0$ and $v = 0$.
Also, a Bézier triangle of degree n relies on the triangular Bernstein polynomials of degree n. Indeed, these polynomials are defined [Farin 1986] by:

$$B^n_{ijk}(u,v,w) = \frac{n!}{i!j!k!} u^i v^j w^k \quad \forall (u,v,w) \mid 0 \leq u,v,w \leq 1 \mid u+v+w=1,$$

with $0 \leq i, j, k \leq n$ and $i + j + k = n$. In a same manner as a Bézier curve, this surface is C^∞. However, for a large n, the surface is sometimes oscillating (like any other polynomial surface).

About triangular Bernstein polynomials

Like Bernstein polynomials, triangular Bernstein polynomials do have several properties:

i) They are positive.

ii) They realize a partition of unity (Cauchy identity):

$$\forall (u,v,w) \in [0,1] \text{ such that } u+v+w=1, \sum_{i+j+k=n} B^n_{ijk}(u,v,w) = 1.$$

iii) Symmetry:

$$\forall (u,v,w) \in [0,1] \text{ such that } u+v+w=1, \; B^n_{ijk}(u,v,w) = B^n_{ikj}(u,w,v),$$

and all the other possible combinations.

iv) They satisfy the following recursion between the degrees:

$$B^n_{ijk}(u,v,w) = uB^{n-1}_{(i-1)jk}(u,v,w) + vB^{n-1}_{i(j-1)k}(u,v,w) + wB^{n-1}_{ij(k-1)}(u,v,w), \quad (3.7)$$

with the convention $B^n_{ijk}(u,v,w) = 0$ if $\min(i,j,k,n) < 0$. And reciprocally:

$$B^n_{ijk}(u,v,w) = \frac{i+1}{n+1}B^{n+1}_{(i+1)jk}(u,v,w) + \frac{j+1}{n+1}B^{n+1}_{i(j+1)k}(u,v,w) + \frac{k+1}{n+1}B^{n+1}_{ij(k+1)}(u,v,w).$$

v) Their derivatives are deduced using the following formula:

$$\frac{\partial}{\partial u}B^n_{ijk}(u,v,w) = nB^{n-1}_{ijk}(u,v,w),$$

with the same convention as above. A similar expression goes for the other partials.

vi) Let us note $\widehat{K} = \{0 \le u,w,w \le 1 \mid u+v+w = 1\}$ then

$$\forall (i,j,k) \in [\![0,n]\!]^3 \text{ s.t. } i+j+k = n, \quad \int_{\widehat{K}} B^n_{ijk}(u,v,w)\mathrm{d}\sigma = \frac{1}{(n+1)(n+2)}.$$

vii) Using the reference coordinates, they form a basis of the triangular finite element space of degree n. Indeed, every term of the polynomial basis $\{\hat{x}^j\hat{y}^k\}_{0\le(j,k)\le 1 \mid j+k\le n}$ with $0 \le (\hat{x},\hat{y}) \le 1, \mid \hat{x}+\hat{y} \le 1$ is expressed using these polynomials:

$$\forall (j,k) \in [\![0,n]\!] \mid j+k \le n \quad \hat{x}^j\hat{y}^k = \sum_{i_1+j_1+k_1=n-j-k} \frac{\binom{j+j_1}{j}\binom{k+k_1}{k}}{\frac{n!}{(n-j-k)!j!k!}} B^n_{i_1(j+j_1)(k+k_1)}(1-\hat{x}-\hat{y},\hat{x},\hat{y}).$$

Proofs:

i) This is straightforward as u, v and w are by definition positive.

ii) If we expand the formula of the polynomials:

$$\begin{aligned}
&\forall (u,v,w) \in [0,1] \text{ s.t. } u+v+w = 1\\
\sum_{i+j+k=n} B^n_{ijk}(u,v,w) &= \sum_{i+j+k=n} \frac{n!}{i!j!k!}u^iv^jw^k\\
&= \sum_{i=0}^{n} \frac{n!}{i!(n-i)!}u^i \sum_{j=0}^{n-i}\binom{n-i}{k}v^jw^k \quad \text{(because } j+k = n-i)\\
&= \sum_{i=0}^{n}\binom{n}{i}u^i \sum_{j=0}^{n-i}\binom{n-i}{n-i-j}v^jw^{n-i-j} = \sum_{i=0}^{n}\binom{n}{i}u^i(v+w)^{n-i}\\
&= (u+v+w)^n = 1.
\end{aligned}$$

iii) This is straightforward with the formula of the polynomials.

iv) First:

$$\begin{aligned}
&uB^{n-1}_{(i-1)jk}(u,v,w) + vB^{n-1}_{i(j-1)k}(u,v,w) + wB^{n-1}_{ij(k-1)}(u,v,w)\\
&= \sum_{i+j+k=n} u^iv^jw^k\frac{n!}{i!j!k!}\left(\frac{i}{n}+\frac{j}{n}+\frac{k}{n}\right) = B^n_{ijk}(u,v,w).
\end{aligned}$$

Then:

$$\begin{aligned}
B^n_{ijk}(u,v,w) &= (u+v+w)B^n_{ijk}(u,v,w)\\
&= uB^n_{ijk}(u,v,w) + vB^n_{ijk}(u,v,w) + wB^n_{ijk}(u,v,w)\\
&= \frac{i+1}{n+1}B^{n+1}_{(i+1)jk}(u,v,w) + \frac{j+1}{n+1}B^{n+1}_{i(j+1)k}(u,v,w) + \frac{k+1}{n+1}B^{n+1}_{ij(k+1)}(u,v,w).
\end{aligned}$$

v) This result is straightforward.

vi) Indeed, if we apply the integration Formula (3.3) of Section 3.2.1, then:

$$\int_{\widehat{K}} B^n_{ijk}(u,v,w)\mathrm{d}\sigma = \frac{n!}{i!j!k!}\frac{i!j!k!}{(2+i+j+k)!} = \frac{n!}{(n+2)!} = \frac{1}{(n+1)(n+2)}.$$

vii) If we expand the formula, it comes:

$$\begin{aligned}
&\sum_{i_1+j_1+k_1=n-j-k} \frac{\binom{j+j_1}{j}\binom{k+k_1}{k}}{\frac{n!}{(n-j-k)!j!k!}} B^n_{i_1(j+j_1)(k+k_1)}(1-\hat{x}-\hat{y},\hat{x},\hat{y}) \\
&= \sum_{i_1+j_1+k_1=n-j-k} \frac{(n-j-k)!(j+j_1)!(k+k_1)!}{n!j_1!k_1!}\frac{n!}{i_1!(j+j_1)!(k+k_1)!}(1-\hat{x}-\hat{y})^{i_1}\hat{x}^{j+j_1}\hat{y}^{k+k_1} \\
&= \hat{x}^j\hat{y}^k \sum_{i_1+j_1+k_1=n-j-k} \frac{(n-j-k)!}{i_1!j_1!k_1!}(1-\hat{x}-\hat{y})^{i_1}\hat{x}^{j_1}\hat{y}^{k_1} \\
&= \hat{x}^j\hat{y}^k \sum_{i_1+j_1+k_1=n-j-k} B^{n-j-k}_{i_1j_1k_1}(1-\hat{x}-\hat{y},\hat{x},\hat{y}) = \hat{x}^j\hat{y}^k.
\end{aligned}$$

De Casteljau's algorithm

Algorithm 9: De Casteljau's evaluation algorithm for a Bézier triangle of degree n

Function *DeCasteljauTri*
- **Input:** Degree $n \in \mathbb{N}$, Control Points P_{ijk}, Parameters $(u,v,w) \in [0,1]^3$ with $u+v+w=1$
- **Output:** val : value on the Bézier triangle corresponding to the parameter (u,v,w)
- **if** $n=0$ **then**
  - val $= P_{000}$
- **else**
  - **for** $(i,j,k) \in \{0,n-1\}^3$ *such that* $i+j+k=n-1$ **do**
    - $P^*_{ijk} = uP_{(i+1)jk} + vP_{i(j+1)k} + wP_{ij(k+1)}$
  - val $=$ DeCasteljauTri$(n-1, P^*_{ijk}, (u,v,w))$

Like for Bézier curves, a convenient way to evaluate a point corresponding to a parameter (u,v,w) on a Bézier triangle is the de Casteljau's algorithm. This algorithm is based on the following recursion:

$$\begin{aligned}
&\forall (i,j,k) \in [\![0,n]\!] \text{ s.t. } i+j+k=n, \\
&D^0_{ijk}(u,v,w) = P_{ijk}, \\
&\forall r \in [\![1,n]\!], \quad \forall (i,j,k) \in [\![0,n-r]\!] \text{ s.t. } i+j+k=n, \\
&D^r_{ijk}(u,v,w) = uD^{r-1}_{(i+1)jk}(u,v,w) + vD^{r-1}_{i(j+1)k}(u,v,w) + wD^{r-1}_{ij(k+1)}(u,v,w).
\end{aligned} \tag{3.8}$$

This recursion ends when $r=n$ and $D^n_{000}(u,v,w) = \sigma(u,v,w)$. Indeed, it is shown that for $r \in [\![0,n]\!]$,

$$D^r_{ijk}(u,v,w) = \sum_{i_1+j_1+k_1=r} B^r_{i_1j_1k_1}(u,v,w)P_{(i+i_1)(j+j_1)(k+k_1)}.$$

The proof is based on the recursion formula (3.7) and uses the same principle as the proof of the de Casteljau's algorithm for the Bézier curve. Evaluating the above expression for

$r = n$ (and forcing $i = j = k = 0$) provides the expected result. Like for Bézier curves, this algorithm does not require any knowledge of Bernstein triangular polynomials and is also numerically more stable for the same reason as with curves. In practice, the algorithm is **Algorithm 9**.
Note this algorithm can also be done in a non-recursive manner [Borouchaki and George 2017b] by applying directly the formula of recursion (3.8).

Subdivision of a Bézier triangle

Subdivision in three sub-triangles

Algorithm 10: Subdivision in 3 algorithm for a Bézier triangle of degree n

Function *SubdivisionTri3*
- **Input:** Degree $n \in \mathbb{N}$, Control Points P_{ijk}, Parameters $(u, v, w) \in [0, 1]^3$ with $u + v + w = 1$
- **Output:** L : array containing the three set of control points forming the subdivision
- **if** $n = 0$ **then**
  - L[0][0][0][0] $= P_{000}$; L[1][0][0][0] $= P_{000}$; L[2][0][0][0] $= P_{000}$
- **else**
  - **for** $(i, j, k) \in \{0, n-1\}^3$ *such that* $i + j + k = n - 1$ **do**
    - $P^*_{ijk} = uP_{(i+1)jk} + vP_{i(j+1)k} + wP_{ij(k+1)}$
  - Lsub = SubdivisionTri3(n-1,P^*_{ijk},(u, v, w))
  - **for** $(i, j, k) \in \{0, n-1\}^3$ *such that* $i + j + k = n - 1$ **do**
    - L[0][i][j][k+1] = Lsub[0][i][j][k]
    - L[1][i][j+1][k] = Lsub[1][i][j][k]
    - L[2][i+1][j][k] = Lsub[2][i][j][k]
  - **for** $(i, j) \in \{0, n-1\}^2$ *such that* $i + j = n$ **do**
    - L[0][i][j][0] $= P_{ij0}$; L[1][i][0][j] $= P_{i0j}$; L[2][0][i][j] $= P_{0ij}$

As we will see in Section 3.3, the subdivision of a Bézier triangle plays an important role in determining the validity of high-order elements. Like for curves, a subdivision of a Bézier triangle is done using the intermediary points created by the de Casteljau's algorithm. Indeed, the three sequences of points $(D^k_{ij0}(u, v, w))_{i+j+k=n}$, $(D^j_{i0k}(u, v, w))_{i+j+k=n}$ and $(D^i_{0jk}(u, v, w))_{i+j+k=n}$ define control points for three Bézier triangles which exactly represent the initial Bézier triangle on respectively each of these barycentric sets: $\{(1, 0, 0); (0, 1, 0); (u, v, w)\}$, $\{(1, 0, 0); (u, v, w); (0, 0, 1)\}$ and $\{(u, v, w); (0, 1, 0); (0, 0, 1)\}$.

Proof :
Let us prove the result for the first sub-triangle.
If we parameterize the first sub-triangle with local barycentric coordinates (u', v', w') with $u' + v' + w' = 1$. The parameterization of the sub-triangle is therefore given by:

$$\sigma_1(u', v', w') = \sum_{i+j+k=n} B^n_{ijk}(u', v', w') D^k_{ij0}(u, v, w).$$

If we expand the expression, we can see that:

$$\sigma_1(u', v', w') = \sum_{i+j+k=n} B^n_{ijk}(u', v', w') \sum_{i_1+j_1+k_1=k} B^k_{i_1 j_1 k_1}(u, v, w) P_{(i+i_1)(j+j_1)k_1}$$

$$= \sum_{i+i_1+j+j_1+k_1=n} \frac{n!}{i!i_1!j!j_1!k_1!}(u')^i(v')^j(w')^{i_1+j_1+k_1}u^{i_1}v^{j_1}w^{k_1}P_{(i+i_1)(j+j_1)k_1}.$$

If we reindex, we can notice that:

$$\begin{aligned}\sigma_1(u',v',w') &= \sum_{i+j+k=n} P_{ijk}\frac{n!}{i!j!k!}(w'w)^k \sum_{i_1+i_2=i}\frac{i!}{i_1!i_2}(u')^{i_1}(w'u)^{i_2}\sum_{j_1+j_2=j}\frac{j!}{j_1!j_2!}(v')^{j_1}(w'v)^{j_2}\\ &= \sum_{i+j+k=n}\frac{n!}{i!j!k!}(u'+w'u)^i(v'+w'v)^j(w'w)^kP_{ijk}\\ &= \sigma(u'+w'u, v'+w'v, w'w),\end{aligned}$$

as $(u'+w'u, v'+w'v, w'w)$ defines the parameterization of the sub-triangle $\{(1,0,0);(0,1,0);(u,v,w)\}$, this achieves the proof.
The same goes for the two other sub-triangles.
The process is summed up by **Algorithm 10**. The output consists in a 4 entries vector where each of the entries is the control points of a sub triangle.
Like de Casteljau's algorithm, this algorithm can also be done in a non-recursive manner.

Subdivision in four sub-triangles
The issue with the previous subdivision is that it does give a subdivision that does not subdivide the element on its edges. However, we will need this property in Section 3.3. For this reason, we may want to perform an exact subdivision of a triangle of degree n in 4 using the three points defined via their barycentric coordinates given in the Figure 3.3 with $0 < \beta, \gamma, \alpha < 1$. Our wish is then to express the control points of these sub-triangles using the initial control points.

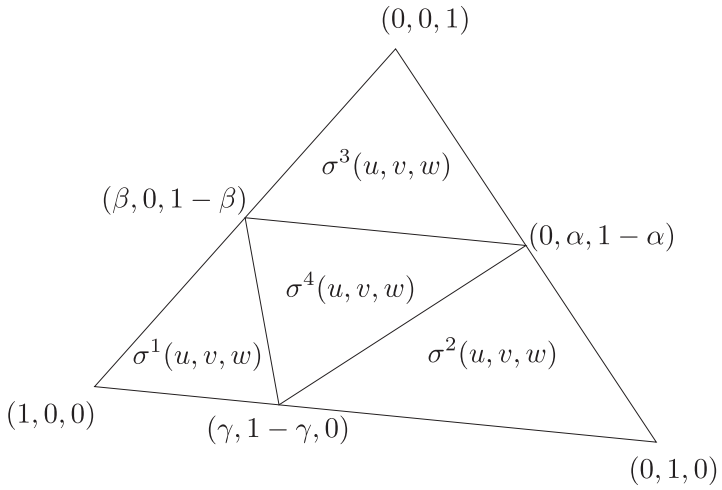

Figure 3.3 – Parts of the triangle where the sub-triangles are created. The points are defined by their barycentric coordinates.

As explained in Figure 3.3, a new parameterization for each sub-triangle is deduced. Let us have a look at the sub-triangle defined by $\{(1,0,0);(\gamma,1-\gamma,0);(\beta,0,1-\beta)\}$. It is parameterized with new barycentric coordinates $0 \leq u, v, w \leq 1$ as follows:

$$\sigma^1(u,v,w) = \sigma(u+\beta w+\gamma v, (1-\gamma)v, (1-\beta)w).$$

Let us expand this formula to obtain an expression of the Bézier parameterization of this sub-triangle.

$$\begin{aligned}\sigma^1(u,v,w) &= \sum_{i+j+k=n} B^n_{ijk}(u+\beta w+\gamma v,(1-\gamma)v,(1-\beta)w)P_{ijk}\\ &= \sum_{i+j+k=n} P_{ijk}\frac{n!}{i!j!k!}(u+\beta w+\gamma v)^i((1-\gamma)v)^j((1-\beta)w)^k\end{aligned}$$

If we notice that :

$$(u+\beta w+\gamma v)^i = \sum_{i_1+j_1+k_1=i} \frac{i!}{i_1!j_1!k_1!}u^{i_1}\beta^{k_1}w^{k_1}\gamma^{j_1}v^{j_1},$$

then we obtain:

$$\sigma^1(u,v,w) = \sum_{i+j+k=n} P_{ijk}\frac{n!}{i!j!k!}(1-\gamma)^j v^j(1-\beta)^k w^k\left(\sum_{i_1+j_1+k_1=i}\frac{i!}{i_1!j_1!k_1!}u^{i_1}\beta^{k_1}w^{k_1}\gamma^{j_1}v^{j_1}\right).$$

Now, if we concatenate all the combinations that define the two sums in one sum, we have:

$$\begin{aligned}\sigma^1(u,v,w) &= \sum_{i_1+j_1+k_1+j+k=n}\frac{n!}{i_1!j_1!k_1!j!k!}P_{(i_1+j_1+k_1)jk}(1-\gamma)^j v^j(1-\beta)^k w^k u^{i_1}\beta^{k_1}w^{k_1}\gamma^{j_1}v^{j_1}\\ &= \sum_{i_1+j_1+k_1+j+k=n}(\frac{n!}{i_1!(k+k_1)!(j+j_1)!}u^{i_1}v^{j+j_1}w^{k+k_1})\\ &\qquad(\frac{(k+k_1)!(j+j_1)!}{k!k_1!j!j_1!}\beta^{k_1}\gamma^{j_1}(1-\gamma)^j(1-\beta)^k)P_{(i_1+j_1+k_1)jk}\\ &= \sum_{i_1+J+K=n} B^n_{i_1JK}(u,v,w)\sum_{\substack{k+k_1=K\\ j+j_1=J}}(\frac{K!J!}{k!k_1!j!j_1!}\beta^{k_1}\gamma^{j_1}(1-\gamma)^j(1-\beta)^k)P_{(i_1+j_1+k_1)jk}\\ &= \sum_{I+J+K=n} B^n_{IJK}(u,v,w)P^1_{IJK}.\end{aligned}$$

The control points of the sub-triangle are therefore expressed as:

$$\begin{aligned}P^1_{IJK} &= \sum_{\substack{j+j_1=J\\ k+k_1=K}}(\frac{K!J!}{k!k_1!j!j_1!}\beta^{k_1}(1-\gamma)^j(1-\beta)^k\gamma^{j_1})P_{(I+j_1+k_1)jk}\\ &= \sum_{j+j_1=J} B^J_{j_1}(\gamma)\left(\sum_{k+k_1=K} B^K_{k_1}(\beta)P_{(I+j_1+k_1)jk}\right).\end{aligned}$$

Using the same derivations on sub-triangles defined with barycentrics $\{(0,0,1);(0,\alpha,1-\alpha);(\gamma,1-\gamma,0)\}$ and $\{(0,0,1);(\beta,0,1-\beta);(0,\alpha,1-\alpha)\}$, we have:

$$\begin{aligned}\sigma^2(u,v,w) =& \sigma(\gamma u, v+(1-\gamma)u+\alpha w,(1-\beta)w) &&= \sum_{I+J+K=n} B^n_{IJK}(u,v,w)P^2_{IJK},\\ \sigma^3(u,v,w) =& \sigma(\beta u,\alpha v, w+(1-\beta)u+(1-\alpha)v) &&= \sum_{I+J+K=n} B^n_{IJK}(u,v,w)P^3_{IJK},\end{aligned}$$

and the associated control points are:

$$P^2_{IJK} = \sum_{i+i_1=I} B^I_{i_1}(1-\gamma)\left(\sum_{k+k_1=K} B^K_{k_1}(\alpha)P_{i(i_1+J+k_1)k}\right),$$

$$P^3_{IJK} = \sum_{i+i_1=I} B^I_{i_1}(1-\beta)\left(\sum_{j+j_1=J} B^J_{j_1}(1-\alpha)P_{ij(i_1+j_1+K)}\right).$$

Finally, for the sub-triangle $\{(\gamma, 1-\gamma, 0); (\beta, 0, 1-\beta); (0, \alpha, 1-\alpha)\}$, its parameterization is:

$$\sigma^4(u,v,w) = \sigma(\beta v + \gamma w, \alpha u + (1-\gamma)w, (1-\alpha)u + (1-\beta)v).$$

Let us expand this formula to obtain the control points:

$$\begin{aligned}
\sigma^4(u,v,w) &= \sum_{i+j+k=n} P_{ijk} B^n_{ijk}(\beta v + \gamma w, \alpha u + (1-\gamma)w, (1-\alpha)u + (1-\beta)v) \\
&= \sum_{i+j+k=n} P_{ijk}\frac{n!}{i!j!k!}(\beta v + \gamma w)^i(\alpha u + (1-\gamma)w)^j((1-\alpha)u + (1-\beta)v)^k \\
&= \sum_{i+j+k=n} P_{ijk}\frac{n!}{i!j!k!}\left\{\sum_{i_1+i_2=i}\frac{i!}{i_1!i_2!}\beta^{i_1}\gamma^{i_2}v^{i_1}w^{i_2}\right\}\left\{\sum_{j_1+j_2=j}\frac{j!}{j_1!j_2!}\alpha^{j_1}(1-\gamma)^{j_2}u^{j_1}w^{j_2}\right\} \\
&\qquad \left\{\sum_{k_1+k_2=k}\frac{k!}{k_1!k_2!}(1-\alpha)^{k_1}(1-\beta)^{k_2}u^{k_1}v^{k_2}\right\} \\
&= \sum_{i_1+i_2+j_1+j_2+k_1+k_2=n} P_{(i_1+i_2)(j_1+j_2)(k_1+k_2)}\frac{n!}{i_1!i_2!j_1!j_2!k_1!k_2!} \\
&\qquad \beta^{i_1}\gamma^{i_2}\alpha^{j_1}(1-\gamma)^{j_2}(1-\alpha)^{k_1}(1-\beta)^{k_2}u^{j_1+k_1}v^{i_2+k_2}w^{i_2+j_2} \\
&= \sum_{I+J+K=n} B^n_{IJK}(u,v,w) \\
&\left(\sum_{\substack{j_1+k_1=I\\ i_1+k_2=J\\ i_2+j_2=K}} \frac{I!J!K!}{i_1!i_2!j_1!j_2!k_1k_2!}\beta^{i_1}\gamma^{i_2}\alpha^{j_1}(1-\gamma)^{j_2}(1-\alpha)^{k_1}(1-\beta)^{k_2}P_{(i_1+i_2)(j_1+j_2)(k_1+k_2)}\right) \\
&= \sum_{I+J+K=n} B^n_{IJK}(u,v,w)P^4_{IJK}.
\end{aligned}$$

The control points of the sub-triangle are therefore expressed:

$$\begin{aligned}
P^4_{IJK} &= \sum_{\substack{j_1+k_1=I\\ i_1+k_2=J\\ i_2+j_2=K}} \frac{I!J!K!}{i_1!i_2!j_1!j_2!k_1k_2!}\beta^{i_1}\gamma^{i_2}\alpha^{j_1}(1-\gamma)^{j_2}(1-\alpha)^{k_1}(1-\beta)^{k_2}P_{(i_1+i_2)(j_1+j_2)(k_1+k_2)}, \\
&= \sum_{j_1+k_1=I} B^I_{j_1}(\alpha)\sum_{i_1+k_2=J} B^J_{i_1}(\beta)\sum_{i_2+j_2=K} B^K_{i_2}(\gamma)P_{(i_1+i_2)(j_1+j_2)(k_1+k_2)}.
\end{aligned}$$

As each of these coefficient is expressed with the Bernstein polynomials, they can be seen as the evaluation of a tensor product of Bézier curves [1]. Consequently, they can be computed using a de Casteljau's algorithm (for the edges). For instance, to compute P^1_{IJK}, first for a fixed (j, j_1), the de Casteljau's algorithm is applied to the Bézier curve $\sum_{k+k_1=K} B^K_{k_1}(u)P_{(I+j_1+k_1)jk}$ in $u=\beta$, the result is noted R_{jj_1}, and then the de Casteljau's alorithm is applied to $\sum_{j+j_1=J} B^J_{j_1}(u)R_{jj_1}$ with $u=\gamma$, which provides P^1_{IJK}.
This computed subdivision has the advantage to be in agreement on the boundary with

1. More precisely, P^1_{IJK} can be seen as the evaluation of the function $f_{IJK}(u,v) = \sum_{j+j_1=J} B^J_{j_1}(u)\left(\sum_{k+k_1=K} B^K_{k_1}(v)P_{(I+j_1+k_1)jk}\right)$ at the point $(u,v)=(\gamma,\beta)$.

the subdivision of a Bézier curve explained in the previous section.
The implementation of the algorithm is straightforward as it consists in directly applying the formulas above. For the following sections, the induced algorithm will be named *SubdivisionTri4*.

3.2.4 Bézier quadrilateral

Definition

For a given set $(P_{ij})_{0\leq i\leq n, 0\leq j\leq m}$ of $(n+1)(m+1)$ points, a Bézier quadrilateral (or Bézier quadrilateral patch) of degree $n \times m$ (when $m = n$, we simply say it is of degree n) is defined using the following algebraic relation:

$$\sigma(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \binom{n}{i}\binom{m}{j} u^i(1-u)^{n-i}v^j(1-v)^{m-j}P_{ij} \quad \forall (u,v) \mid 0 \leq u,v \leq 1.$$

Here, (u,v) are exactly the reference coordinates $(\hat{x},\hat{y}) \in [0,1]^2$. In this context, P_{ij} are called control points and their dimensions are arbitrary. Note that $\sigma(0,0) = P_{00}$, $\sigma(0,1) = P_{0m}$, $\sigma(1,1) = P_{nm}$ and $\sigma(1,0) = P_{n0}$. Likewise, if (for instance) $v = 0$, then:

$$\sigma(u,0) = \sum_{i=0}^{n} \binom{n}{i} u^i(1-u)^{n-i}P_{i0} \quad \forall u \in [0,1].$$

which is the expression of a Bézier curve of degree n defined by $(P_{i0})_{1\leq i\leq n}$. The same kind of observation holds for $u = 1$, $v = 0$ and $v = 1$.
A Bézier quadrilateral is expressed using Bernstein polynomials:

$$\sigma(u,v) = \sum_{i=0}^{n} B_i^n(u) \sum_{j=0}^{m} B_i^m(v)P_{ij} \quad \forall (u,v) \mid 0 \leq u,v \leq 1.$$

In particular, we can see that a Bézier quadrilateral of degree $n \times m$ is interpreted as the tensor product of two Bézier curves of degree n and m. Henceforth, the observations about regularity and oscillation observed with curves hold for quadrilaterals.

De Casteljau's algorithm

Algorithm 11: De Casteljau's evaluation algorithm for a Bézier quadrilateral of degree $n \times m$

Function *DeCasteljauQua*
- **Input:** Degrees $(n,m) \in \mathbb{N}^2$, Control Points P_{ij}, Parameters $(u,v) \in [0,1]^2$
- **Output:** val : value on the Bézier quadrilateral corresponding to the parameter (u,v)
- **for** $j \in \{0,m\}$ **do**
  - **for** $i \in \{0,n\}$ **do**
    - $P_i^* = P_{ij}$
  - val_j = DeCasteljauCurv(n, P_i^*, u)
- val = DeCasteljauCurv(m, val_j, v)

As the Bézier quadrilateral is seen as a tensor product of two Bézier curves, the evaluation of a point of parameter (u,v) is done by applying twice the de Casteljau's algorithm for

the curves. This algorithm is based on the following recursion:

$$\begin{aligned}
&D_{ij}^{00}(u,v) = P_{ij}, \quad \forall i \in [\![0,n]\!] \quad \forall j \in [\![0,n]\!] \\
&D_{ij}^{r0}(u,v) = (1-u)D_{ij}^{(r-1)0}(u,v) + uD_{(i+1)j}^{(r-1)0}(u,v), \quad \forall i \in [\![0,n-r]\!] \quad \forall j \in [\![0,m]\!] \quad \forall r \in [\![1,n]\!], \\
&D_{0j}^{nr}(u,v) = (1-v)D_{0j}^{n(r-1)}(u,v) + vD_{0(j+1)}^{n(r-1)}(u,v), \quad \forall j \in [\![0,m-r]\!] \quad \forall r \in [\![1,m]\!].
\end{aligned} \tag{3.9}$$

Eventually, $D_{00}^{nm}(u,v) = \sigma(u,v)$. The proof of this result is straightforward as the result is already established for the curves.
Note that if $n = m$, a more simple algorithm can be used, but unlike the other algorithms it does not provide a subdivision [Borouchaki and George 2017b]. For this reason, we will not speak about it.
The evaluation algorithm is given in **Algorithm 11**.

Subdivision of a Bézier quadrilateral

As we will see in Section 3.3, the subdivision of a Bézier quadrilateral plays an important role in determining the validity of high-order elements. Using again the tensor product structure of a Bézier quadrilateral, a subdivision is deduced using the subdivision algorithm of a Bézier curve. A subdivision is proposed in **Algorithm 12**.

Algorithm 12: Subdivision algorithm for a Bézier Quadrilateral of degree $n \times m$

Function *SubdivisionQua*
- **Input:** Degree $(n,m) \in \mathbb{N}^2$, Control Points P_{ij}, Parameter $(u,v) \in [0,1]^2$
- **Output:** L : array containing the four set of control points forming the subdivision
- **for** $j \in \{0,m\}$ **do**
  - **for** $i \in \{0,n\}$ **do**
    - $P_i^* = P_{ij}$
  - L_j = SubdivisionCurv(n, P_i^*, u)
- **for** $i \in \{0,2n+1\}$ **do**
  - **for** $j \in \{0,m\}$ **do**
    - $P_j^* = L_j[i]$
  - L[i] = SubdivisionCurv(m, P_j^*, v)

3.2.5 Bézier tetrahedron

Definition

For a given set $(P_{ijkl})_{0\leq i,j,k,l\leq n | i+j+k+l=n}$ of $\frac{(n+1)(n+2)(n+3)}{6}$ points, a Bézier tetrahedron (or Bézier tetrahedral patch) of degree n is defined using the following algebraic relation:

$$\omega(u,v,w,t) = \sum_{i+j+k+l=n} \frac{n!}{i!j!k!l!} u^i v^j w^k t^l P_{ijkl}, \quad \forall (u,v,w,t) \mid 0 \leq u,v,w,t \leq 1 \mid u+v+w+t = 1.$$

(u,v,w,t) are called the barycentric coordinates and are linked to the reference coordinates $(\hat{x},\hat{y},\hat{z}) \in [0,1]^2$ via $(u,v,w,t) = (1-\hat{x}-\hat{y}-\hat{z},\hat{x},\hat{y},\hat{z})$. In this context, P_{ijk} are called control points and their dimensions are arbitrary. Note that if (for instance) $t = 0$, then:

$$\omega(u,v,w,0) = \sum_{i+j+k=n} \frac{n!}{i!j!k!} u^i v^j w^k P_{ijk0}, \quad \forall (u,v,w) \mid 0 \leq u,v,w \leq 1 \mid u+v+w = 1,$$

which is the expression of a Bézier triangle defined with $(P_{ijk0})_{0\leq i,j,k\leq n | i+j+k+l=n}$. The same remark stands with $u=0$, $v=0$ and $w=0$. Likewise, Bézier curves can be found when two coordinates are 0.
Also, a Bézier tetrahedron of degree n relies on the tetrahedral Bernstein polynomials of degree n. Indeed, these polynomials are defined [Borouchaki and George 2017b] by:

$$B^n_{ijkl}(u,v,w,t) = \frac{n!}{i!j!k!l!}u^i v^j w^k t^l, \quad \forall (u,v,w,t) \mid 0\leq u,v,w,t\leq 1 \mid u+v+w+t=1,$$

with $0\leq i,j,k,l\leq n$ and $i+j+k+l=n$.

About tetrahedral Bernstein polynomials

Like other Bernstein polynomials, tetrahedral Bernstein polynomials enjoy several properties:

i) They are positive.

ii) They realize a partition of unity (Cauchy identity):

$$\forall (u,v,w,t)\in[0,1] \text{ s.t. } u+v+w+t=1, \sum_{i+j+k+l=n} B^n_{ijkl}(u,v,w,t)=1.$$

iii) Symmetry:

$$\forall (u,v,w,t)\in[0,1] \text{ such that } u+v+w+t=1,\ B^n_{ijkl}(u,v,w,t)=B^n_{ilkj}(u,t,w,v),$$

and all the other possible combinations.

iv) They satisfy the following recursion between the degrees:

$$\begin{aligned} B^n_{ijkl}(u,v,w,t) &= uB^{n-1}_{(i-1)jkl}(u,v,w,t)+vB^{n-1}_{i(j-1)kl}(u,v,w,t)\\ &\quad + wB^{n-1}_{ij(k-1)l}(u,v,w,t)+tB^{n-1}_{ijk(l-1)}(u,v,w,t), \end{aligned} \tag{3.10}$$

with the convention $B^n_{ijkl}(u,v,w,t)=0$ if $\min(i,j,k,l,n)<0$. And reciprocally:

$$\begin{aligned} B^n_{ijkl}(u,v,w,t) &= \frac{i+1}{n+1}B^{n+1}_{(i+1)jkl}(u,v,w,t)+\frac{j+1}{n+1}B^{n+1}_{i(j+1)kl}(u,v,w,t)\\ &\quad + \frac{k+1}{n+1}B^{n+1}_{ij(k+1)l}(u,v,w,t)+\frac{l+1}{n+1}B^{n+1}_{ijk(l+1)}(u,v,w,t). \end{aligned}$$

v) Their derivatives are deduced using the following formula:

$$\frac{\partial}{\partial u}B^n_{ijkl}(u,v,w,t)=nB^{n-1}_{ijkl}(u,v,w,t),$$

with the same convention as above. A similar expression is found for the other partials.

vi) Let us note $\widehat{K}=\{0\leq u,w,w,t\leq 1 \mid u+v+w+t=1\}$ then:

$$\forall (i,j,k,l)\in[\![0,n]\!]^4 \text{ s.t. } i+j+k+l=n, \quad \int_{\widehat{K}} B^n_{ijkl}(u,v,w,t)\mathrm{d}\Omega = \frac{1}{(n+1)(n+2)}.$$

vii) Using the reference coordinates, they form a basis of the tetrahedral finite element space of degree n. Indeed, every term of the polynomial basis $\{\hat{x}^j\hat{y}^k\hat{z}^l\}_{0\leq(j,k,l)\leq 1 \mid j+k+l\leq n}$ with $0\leq(\hat{x},\hat{y},\hat{z})\leq 1 \mid \hat{x}+\hat{y}+\hat{z}\leq 1$ is expressed using these polynomials:

$$\forall (j,k,l)\in[\![0,n]\!] \mid j+k+l\leq n$$

$$\hat{x}^j\hat{y}^k\hat{z}^l = \sum_{i_1+j_1+k_1+l_1=n-j-k-l} \frac{\binom{j+j_1}{j}\binom{k+k_1}{k}\binom{l+l_1}{l}}{\frac{n!}{(n-j-k-l)!j!k!l!}} B^n_{i_1(j+j_1)(k+k_1)(l+l_1)}(1-\hat{x}-\hat{y}-\hat{z},\hat{x},\hat{y},\hat{z}).$$

Proofs:

i) This is straightforward as u,v, w and t are positive by definition.

ii) If we expand the formula, it comes:

$$\begin{aligned}
&\forall(u,v,w,t)\in[0,1] \text{ s.t. } u+v+w+t=1\\
\sum_{i+j+k+l=n} B^n_{ijkl}(u,v,w,t) &= \sum_{i+j+k+l=n} \frac{n!}{i!j!k!l!}u^i v^j w^k t^l\\
&= \sum_{i=0}^{n}\frac{n!}{i!(n-i)!}u^i \sum_{j=0}^{n-i}\frac{(n-i)!}{(n-i-j)!j!}v^j \sum_{k+l=n-i-j}\frac{(n-i-j)!}{k!l!}w^k t^l\\
&= \sum_{i=0}^{n}\binom{n}{i}u^i \sum_{j=0}^{n-i}\binom{n-i}{n-i-j}v^j (w+t)^{n-i-j}\\
&= \sum_{i=0}^{n}\binom{n}{i}u^i (v+w+t)^{n-i}\\
&= (u+v+w+t)^n = 1.
\end{aligned}$$

iii) This result is straightforward with the formula.

iv) First:

$$\begin{aligned}
&uB^{n-1}_{(i-1)jkl}(u,v,w,t) + vB^{n-1}_{i(j-1)kl}(u,v,w,t) +\\
&wB^{n-1}_{ij(k-1)l}(u,v,w,t) + tB^{n-1}_{ijk(l-1)}(u,v,w,t)\\
&= \sum_{i+j+k_l=n} u^i v^j w^k t^l \frac{n!}{i!j!k!l!}\left(\frac{i}{n}+\frac{j}{n}+\frac{k}{n}+\frac{l}{n}\right)\\
&= B^n_{ijkl}(u,v,w,t).
\end{aligned}$$

Then:

$$\begin{aligned}
B^n_{ijk}(u,v,w) &= (u+v+w+t)B^n_{ijkl}(u,v,w,t)\\
&= uB^n_{ijkl}(u,v,w,t) + vB^n_{ijkl}(u,v,w,t) + wB^n_{ijkl}(u,v,w,t)\\
&= \frac{i+1}{n+1}B^{n+1}_{(i+1)jkl}(u,v,w,t) + \frac{j+1}{n+1}B^{n+1}_{i(j+1)kl}(u,v,w,t)\\
&+ \frac{k+1}{n+1}B^{n+1}_{ij(k+1)l}(u,v,w,t) + \frac{l+1}{n+1}B^{n+1}_{ijk(l+1)}(u,v,w,t)
\end{aligned}$$

v) This result is straightforward.

vi) By using the integration result (3.4) of Section 3.2.1, it comes:

$$\int_{\widehat{K}} B^n_{ijkl}(u,v,w)\mathrm{d}\Omega = \frac{n!}{i!j!k!l!}\frac{i!j!k!l!}{(3+i+j+k+l)!} = \frac{n!}{(n+3)!} = \frac{1}{(n+1)(n+2)(n+3)}.$$

vii) By expanding the formula, it comes:

$$\begin{aligned}
&\sum_{i_1+j_1+k_1+l_1=n-j-k-l} \frac{\binom{j+j_1}{j}\binom{k+k_1}{k}\binom{l+l_1}{l}}{\frac{n!}{(n-j-k-l)!j!k!l!}} B^n_{i_1(j+j_1)(k+k_1)(l+l_1)}(1-\hat{x}-\hat{y}-\hat{z},\hat{x},\hat{y},\hat{z})\\
&= \sum_{i_1+j_1+k_1=n-j-k-l} \frac{(n-j-k-l)!(j+j_1)!(k+k_1)!(l+l_1)!n!}{n!j_1!k_1!l_1!i_1!(j+j_1)!(k+k_1)!(l+l_1)!}(1-\hat{x}-\hat{y}-\hat{z})^{i_1}\hat{x}^{j+j_1}\hat{y}^{k+k_1}\hat{z}^{l+l_1}\\
&= \hat{x}^j\hat{y}^k\hat{z}^l \sum_{i_1+j_1+k_1+l_1=n-j-k-l} \frac{(n-j-k-l)!}{i_1!j_1!k_1!}(1-\hat{x}-\hat{y})^{i_1}\hat{x}^{j_1}\hat{y}^{k_1}\\
&= \hat{x}^j\hat{y}^k\hat{z}^l \sum_{i_1+j_1+k_1+l_1=n-j-k-l} B^{n-j-k-l}_{i_1j_1k_1l_1}(1-\hat{x}-\hat{y}-\hat{z},\hat{x},\hat{y},\hat{z})\\
&= \hat{x}^j\hat{y}^k\hat{z}^l.
\end{aligned}$$

De Casteljau's algorithm

Like for Bézier triangles, a convenient way to evaluate a point corresponding to a parameter (u, v, w, t) on a Bézier tetrahedron is the de Casteljau's algorithm. This algorithm is based on the following recursion:

$$\begin{aligned}
&\forall (i,j,k,l) \in [\![0,n]\!] \text{ s.t. } i+j+k+l=n,\\
&D^0_{ijkl}(u,v,w,t) = P_{ijkl},\\
&\forall r \in [\![1,n]\!], \quad \forall (i,j,k,l) \in [\![0,n-r]\!] \text{ s.t. } i+j+k+l=n,\\
&D^r_{ijkl}(u,v,w,t) = uD^{r-1}_{(i+1)jkl}(u,v,w,t) + vD^{r-1}_{i(j+1)kl}(u,v,w,t)\\
&\qquad\qquad + wD^{r-1}_{ij(k+1)l}(u,v,w,t) + tD^{r-1}_{ijk(l+1)}(u,v,w,t).
\end{aligned} \tag{3.11}$$

This recursion ends when $r = n$ and $D^n_{000}(u, v, w, t) = \omega(u, v, w, t)$. Indeed, it is shown that for $r \in [\![0, n]\!]$,

$$D^r_{ijkl}(u,v,w,t) = \sum_{i_1+j_1+k_1+l_1=r} B^r_{i_1 j_1 k_1 l_1}(u,v,w,t) P_{(i+i_1)(j+j_1)(k+k_1)(l+l_1)}.$$

The proof is based on the recursion formula (3.10) and uses the same principles as the proof of the other de Casteljau's algorithms. Evaluating the expression above for $r = n$ (and thus forcing $i = j = k = l = 0$) provides the expected result. Like for other Bézier elements, this algorithm does not require any knowledge of Bernstein tetrahedral polynomials and is numerically more stable. In practice, the algorithm is **Algorithm 13**.

Algorithm 13: De Casteljau's evaluation algorithm for a Bézier tetrahedron of degree n

Function *DeCasteljauTet*
- **Input:** Degree $n \in \mathbb{N}$, Control Points P_{ijkl}, Parameters $(u, v, w, t) \in [0, 1]^4$ with $u + v + w + t = 1$
- **Output:** val : value on the Bézier tetrahedron corresponding to the parameter (u, v, w, t)
- **if** $n = 0$ **then**
  - val $= P_{0000}$
- **else**
  - **for** $(i, j, k, l) \in \{0, n-1\}^4$ *such that* $i + j + k + l = n - 1$ **do**
    - $P^*_{ijk} = uP_{(i+1)jkl} + vP_{i(j+1)kl} + wP_{ij(k+1)l} + tP_{ijk(l+1)}$
  - val $=$ DeCasteljauTet$(n-1, P^*_{ijk}, (u, v, w, t))$

Note this algorithm can also be done in a non-recursive manner [Borouchaki and George 2017b] by applying directly the formula of recursion (3.11).

Subdivision of a Bézier tetrahedron

Subdivision in four sub-tetrahedra

As we will see in Section 3.3, the subdivision of a Bézier tetrahedron plays an important role in determining the validity of a high-order tetrahedron. A subdivision of a Bézier tetrahedron is done using the intermediary points created by the de Casteljau's algorithm. Indeed, the four sequences of points $(D^l_{ijk0}(u,v,w,t))_{i+j+k+l=n}$, $(D^k_{ij0l}(u,v,w,t))_{i+j+k+l=n}$, $(D^j_{i0kl}(u,v,w,t))_{i+j+k+l=n}$ and $(D^i_{0jkl}(u,v,w,t))_{i+j+k+l=n}$ define control points for four

Bézier tetrahedra which exactly represent the initial Bézier tetrahedron on respectively each of these barycentric sets:

$$\{(1,0,0,0);(0,1,0,0);(0,0,1,0);(u,v,w,t)\},\{(1,0,0,0);(0,1,0,0);(u,v,w,t);(0,0,0,1)\},$$
$$\{(1,0,0,0);(u,v,w,t);(0,0,1,0);(0,0,0,1)\},\{(u,v,w,t);(0,1,0,0);(0,0,1,0);(0,0,0,1)\}.$$

The proof of a such subdivision is done in a same manner as with the triangles.
The process is summed up by**Algorithm 14**.

Algorithm 14: Subdivision in 4 algorithm for a Bézier tetrahedron of degree n

Function *SubdivisionTet4*
- **Input:** Degree $n \in \mathbb{N}$, Control Points P_{ijkl}, Parameters $(u,v,w,t) \in [0,1]^4$ with $u+v+w+t=1$
- **Output:** L : array containing the three set of control points forming the subdivision
- **if** $n=0$ **then**
  - L[0][0][0][0] = P_{0000}; L[1][0][0][0] = P_{0000}; L[2][0][0][0] = P_{0000}; L[3][0][0][0] = P_{0000}
- **else**
  - **for** $(i,j,k,l) \in \{0,n-1\}^4$ *such that* $i+j+k+l=n-1$ **do**
    - $P^*_{ijkl} = uP_{(i+1)jkl} + vP_{i(j+1)kl} + wP_{ij(k+1)l} + tP_{ijk(l+1)}$
  - Lsub = SubdivisionTet4(n-1,P^*_{ijkl},(u,v,w,t))
  - **for** $(i,j,k,l) \in \{0,n-1\}^4$ *such that* $i+j+k+l=n-1$ **do**
    - L[0][i][j][k+1][l] = Lsub[0][i][j][k][l]
    - L[1][i][j+1][k][l] = Lsub[1][i][j][k][l]
    - L[2][i+1][j][k][l] = Lsub[2][i][j][k][l]
    - L[3][i][j][k][l+1] = Lsub[3][i][j][k][l]
  - **for** $(i,j,k) \in \{0,n-1\}^3$ *such that* $i+j+k=n$ **do**
    - L[0][i][j][k][0] = P_{ijk0}; L[1][i][j][0][k] = P_{ij0k}; L[2][i][0][j][k] = P_{i0jk}; L[3][0][i][j][k] = P_{0ijk}

Like de Casteljau's algorithm, this algorithm can also be done in a non-recursive manner.

Subdivision in eight sub-tetrahedra
Like with triangles, the issue with the previous subdivision is that it does give a subdivision that does not subdivide the element on its edges nor on its triangular faces. This property is, however, mandatory to deal properly with the validity of a high-order tetrahedron. For this reason, we want to perform an exact subdivision of a tetrahedron of degree n in 8 [2] using the six points defined on each edge via their barycentric coordinates with $0 < \alpha,\beta,\gamma,\nu,\mu,\delta < 1$ respectively corresponding to edges $P_{0n00}P_{n000}$, $P_{00n0}P_{n000}$, $P_{00n0}P_{0n00}$, $P_{000n}P_{n000}$, $P_{000n}P_{0n00}$ and $P_{000n}P_{00n0}$. Our wish is then to express the control points of these sub-tetrahedra using the initial control points.
Let us define the first sub-tetrahedron defined with the barycentrics $\{(1,0,0,0);(\alpha,1-\alpha,0,0),(\beta,0,1-\beta,0),(\nu,0,0,1-\nu)\}$ as follows:

$$\omega^1(u,v,w,t) = \omega(u+\alpha v+\beta w+\nu t,(1-\alpha)v,(1-\beta)w,(1-\nu)t).$$

2. Note that three different valid subdivisions in 8 are possible. We will only consider one of them.

Let us expand this formula to obtain an expression of the Bézier parameterization of this sub-tetrahedron.

$$\begin{aligned}\omega^1(u,v,w,t) &= \sum_{i+j+k+l=n} B^n_{ijk}(u+\alpha v+\beta w+\nu t,(1-\alpha)v,(1-\beta)w,(1-\nu)t)P_{ijkl}\\ &= \sum_{i+j+k+l=n} P_{ijkl}\frac{n!}{i!j!k!l!}(u+\alpha v+\beta w+\nu t)^i((1-\alpha)v)^j((1-\beta)w)^k((1-\nu)t)^l\end{aligned}$$

If we notice that :

$$(u+\alpha v+\beta w+\nu t)^i = \sum_{i_1+j_1+k_1+l_1=i} \frac{i!}{i_1!j_1!k_1!l_1!}u^{i_1}\alpha^{j_1}v^{j_1}\beta^{k_1}w^{k_1}\nu^{l_1}t^{l_1},$$

then we obtain:

$$\begin{aligned}\omega^1(u,v,w,t) &= \sum_{i_1+j_1+k_1+l_1+j+k+l=n} \frac{n!}{i_1!j_1!k_1!l_1!j!k!l!}P_{(i_1+j_1+k_1+l_1)jkl}(1-\alpha)^j v^j(1-\beta)^k w^k(1-\beta)^l t^l\\ &\qquad u^{i_1}\alpha^{j_1}v^{j_1}\beta^{k_1}w^{k_1}\nu^{l_1}t^{l_1}\\ &= \sum_{i_1+j_1+k_1+l_1+j+k+l=n} (\frac{n!}{i_1!(k+k_1)!(l+k_1)!(j+j_1)!}u^{i_1}v^{j+j_1}w^{k+k_1}t^{l+l_1})\\ &\qquad (\frac{(k+k_1)!(j+j_1)!(l+l_1)!}{k!k_1!j!j_1!l!l_1!}\alpha^{j_1}\beta^{k_1}\nu^{l_1}(1-\alpha)^j(1-\beta)^k(1-\nu)^l)P_{(i_1+j_1+k_1+l_1)jkl}\\ &= \sum_{i_1+J+K+L=n} B^n_{i_1JKL}(u,v,w,t)\\ &\quad \sum_{\substack{k+k_1=K\\ j+j_1=J\\ k+k_1=J}} (\frac{K!J!L!}{k!k_1!j!j_1!l!l_1!}\beta^{k_1}\alpha^{j_1}\nu^{l_1}(1-\alpha)^j(1-\beta)^k(1-\beta)^l)P_{(i_1+j_1+k_1+l_1)jkl}\\ &= \sum_{I+J+K+L=n} B^n_{IJKL}(u,v,w,t)P^1_{IJKL}.\end{aligned}$$

The control solutions of the sub-tetrahedron are therefore expressed as:

$$\begin{aligned}P^1_{IJKL} &= \sum_{\substack{j+j_1=J\\ k+k_1=K\\ l+l_1=L}} (\frac{K!J!L!}{k!k_1!j!j_1!l!l_1!}\beta^{k_1}(1-\alpha)^j(1-\beta)^k(1-\nu)^l\alpha^{j_1}\nu^{l_1})P_{(I+j_1+k_1+l_1)jkl}\\ &= \sum_{j+j_1=J} B^J_{j_1}(\alpha)\left(\sum_{k+k_1=K} B^K_{k_1}(\beta)\left(\sum_{l+l_1=L} B^L_{l_1}(\nu)P_{(I+j_1+k_1+l_1)jkl}\right)\right).\end{aligned}$$

Likewise, for the three other sub-tetrahedra $\{(\alpha,1-\alpha,0,0),(0,1,0,0),(0,\gamma,1-\gamma,0),(\nu,0,0,1-\nu)\}$; $\{(\beta,0,1-\beta,0,0),(0,\gamma,1-\gamma,0),(0,0,1,0),(0,0,\delta,1-\delta)\}$ and $\{(\nu,0,0,1-\nu);(0,\mu,0,1-\mu),(0,0,\delta,1-\delta),(0,0,0,1)\}$, the expressions are :

$$\begin{aligned}\omega^2(u,v,w,t) &= \omega(\alpha u, v+(1-\alpha)u+\gamma w+\mu t,(1-\gamma)w,(1-\mu)t),\\ \omega^3(u,v,w,t) &= \omega(\beta u,\gamma v, w+(1-\beta)u+(1-\gamma)v+\delta t,(1-\delta)t),\\ \omega^4(u,v,w,t) &= \omega(\nu u,\mu v,\delta w, t+(1-\nu)u+(1-\mu)v+(1-\delta)w).\end{aligned}$$

Consequently the control coefficients are:

$$P^2_{IJKL} = \sum_{i+i_1=I} B^I_{i_1}(1-\alpha)\left(\sum_{k+k_1=K} B^K_{k_1}(\gamma)\left(\sum_{l+l_1=L} B^L_{l_1}(\mu)P_{i(i_1+J+k_1+l_1)kl}\right)\right),$$

$$P^3_{IJKL} = \sum_{i+i_1=I} B^I_{i_1}(1-\beta)\left(\sum_{j+j_1=J} B^J_{j_1}(1-\gamma)\left(\sum_{l+l_1=L} B^L_{l_1}(\delta)P_{ij(i_1+j_1+K+l_1)l}\right)\right),$$

$$P^4_{IJKL} = \sum_{i+i_1=I} B^I_{i_1}(1-\nu)\left(\sum_{j+j_1=J} B^J_{j_1}(1-\mu)\left(\sum_{k+k_1=K} B^K_{k_1}(1-\delta)P_{ijk(i_1+j_1+k_1+L)}\right)\right).$$

For the inner sub-tetrahedra defined by $\{(\alpha, 1-\alpha, 0, 0), (0, 0, \delta, 1-\delta), (\nu, 0, 0, 1-\nu), (0, \mu, 0, 1-\mu)\}$, its parameterization is given by:

$$\omega^5(u, v, w, t) = \omega(\alpha u + \mu w, (1-\alpha)u + \mu t, \delta v, (1-\delta)v + (1-\mu)t + (1-\nu)w).$$

If we expand it with Bernstein polynomials, we obtain:

$$\begin{aligned}\omega^5(u, v, w, t) = &\sum_{i+j+k+l=n} \frac{n!}{i!j!k!l!}\left(\sum_{i_1+i_2=i} \frac{i!}{i_1!i_2!}\alpha^{i_1}u^{i_1}\nu^{i_2}w^{i_2}\right)\left(\sum_{j_1+j_2=j} \frac{j!}{j_1!j_2!}(1-\alpha)^{j_1}u^{j_1}\mu^{j_2}t^{j_2}\right)\\ &\left(\sum_{l_1+l_2+l_3=l} \frac{l!}{l_1!l_2!l_3!}(1-\delta)^{l_1}v^{l_1}(1-\mu)^{l_2}t^{l_2})(1-\nu)^{l_3}w^{l_3}\right)P_{(i_1+i_2)(j_1+j_2)k(l_1+l_2+l_3)}\\ &= \sum_{I+J+K+L=n} B^n_{IJKL}(u, v, w, t)\\ &\left(\sum_{\substack{i_1+j_1=I\\ k+l_1=J\\ i_2+l_3=K\\ l_2+j_2=L}} \frac{I!j!K!L!}{i_1!j_1k!l_1!i_2!l_3!l_2!j_2!}\alpha^{i_1}\nu^{i_2}(1-\alpha)^{j_1}\mu^{j_2}\delta^k(1-\delta)^{l_1}(1-\mu)^{l_2}P_{(i_1+i_2)(j_1+j_2)k(l_1+l_2+l_3)}\right).\end{aligned}$$

Finally, its parameterization is rewritten as:

$$\omega^5(u, v, w, t) = \sum_{I+J+K+L=n} B^n_{IJKL}(u, v, w, t)P^5_{IJKL},$$

with :

$$\begin{aligned}P^5_{IJKL} &= \sum_{\substack{i_1+j_1=I\\ k+l_1=J\\ i_2+l_3=K\\ l_2+j_2=L}} \frac{I!j!K!L!}{i_1!j_1k!l_1!i_2!l_3!l_2!j_2!}\alpha^{i_1}\nu^{i_2}(1-\alpha)^{j_1}\mu^{j_2}\delta^k(1-\delta)^{l_1}(1-\mu)^{l_2}P_{(i_1+i_2)(j_1+j_2)k(l_1+l_2+l_3)}\\ &= \sum_{i_1+j_1=I} B^I_{i_1}(\alpha)\left(\sum_{k+l_1=J} B^J_k(\delta)\left(\sum_{i_2+l_3=K} B^K_{i_2}(\nu)\left(\sum_{j_2+l_2=L} B^L_{j_2}(\mu)P_{(i_1+i_2)(j_1+j_2)k(l_1+l_2+l_3)}\right)\right)\right).\end{aligned}$$

Likewise, if we consider the three sub-tetrahedra defined by $\{(\alpha, 1-\alpha, 0, 0), (0, 0, \delta, 1-\delta), (0, \mu, 0, 1-\mu), (0, \gamma, 1-\gamma, 0)\}$; $\{(\alpha, 1-\alpha, 0, 0), (0, 0, \delta, 1-\delta), (0, \gamma, 1-\gamma, 0), (\beta, 0, 1-\beta, 0)\}$ and $\{(\alpha, 1-\alpha, 0, 0), (0, 0, \delta, 1-\delta), (\beta, 0, 1-\beta, 0), (\nu, 0, 0, 1-\nu)\}$, their parameterization is given by:

$$\begin{aligned}\omega^6(u, v, w, t) &= \omega(\alpha u, (1-\alpha)u + \mu w + \gamma t, \delta v + (1-\gamma)t, (1-\mu)w + (1-\delta)v),\\ \omega^7(u, v, w, t) &= \omega(\alpha u + \beta t, (1-\alpha)u + \gamma w, (1-\gamma)w + \delta v + (1-\beta)t, (1-\delta)v),\\ \omega^8(u, v, w, t) &= \omega(\alpha u + \beta w + \nu t, (1-\alpha)u, (1-\beta)w + \delta v, (1-\delta)v + (1-\nu)t).\end{aligned}$$

Consequently the control coefficients are:

$$P^6_{IJKL} = \sum_{i+j_1=I} B^I_i(\alpha)\left(\sum_{k_1+l_2=J} B^J_{k_1}(\delta)\left(\sum_{j_2+l_1=K} B^K_{j_2}(\mu)\left(\sum_{j_3+k_2=L} B^L_{j_3}(\gamma)P_{i(j_1+j_2+j_3)(k_1+k_2)(l_1+l_2)}\right)\right)\right),$$

$$P^7_{IJKL} = \sum_{i_1+j_1=I} B^I_{i_1}(\alpha) \left(\sum_{k_2+l=J} B^J_{k_2}(\delta) \left(\sum_{j_2+k_1=K} B^K_{j_2}(\gamma) \left(\sum_{i_2+k_3=L} B^L_{i_2}(\beta) P_{(i_1+i_2)(j_1+j_2)(k_1+k_2+k_3)l} \right) \right) \right),$$

$$P^8_{IJKL} = \sum_{i_1+j=I} B^I_{i_1}(\alpha) \left(\sum_{k_2+l_1=J} B^J_{k_2}(\delta) \left(\sum_{i_2+k_1=K} B^K_{i_2}(\beta) \left(\sum_{i_3+l_2=L} B^L_{i_3}(\nu) P_{(i_1+i_2+i_3)j(k_1+k_2)(l_1+l_2)} \right) \right) \right).$$

As each of these coefficients is expressed with the Bernstein polynomials, they can be seen as the evaluation of a tensor product of Bézier curves. Consequently, they can be computed using a de Casteljau's algorithm (for the edges) in the same manner as in the case of the subdivision of a triangle in four.
This computed subdivision has the advantage to be in agreement on the boundary with the subdivision of a Bézier curve and the subdivision of a Bézier triangle in four explained previously.
The implementation into an algorithm is straightforward as it consists in directly applying the formulas above. For the following sections, the induced algorithm will be named *SubdivisionTet8*.

3.2.6 Bézier prism

Definition

For a given set $(P_{ijkl})_{0\leq l\leq m, 0\leq i,j,k\leq n | i+j+k=n}$ of $\frac{(n+1)(n+2)(m+1)}{2}$ points, a Bézier prism (or Bézier prismatic patch) of degree $n \times m$ (when $m = n$, we simply say it is of degree n) is defined using the following algebraic relation:

$$\omega(u,v,w,t) = \sum_{i+j+k=n} \sum_{l=0}^{m} \frac{n!}{i!j!k!} \binom{m}{l} u^i v^j w^k t^l (1-t)^{m-l} P_{ijkl}$$
$$\forall (u,v,w,t) \mid 0 \leq u,v,w,t \leq 1 \quad u+v+w = 1.$$

(u, v, w, t) are called the "barycentric" coordinates and are linked to the reference coordinates $(x, y, z) \in [0, 1]^3$ with the relationship $(u, v, w, t) = (1 - x - y, x, y, z)$. In this context, P_{ijkl} are called control points and their dimensions are arbitrary. Note that if (for instance) $t = 0$, then:

$$\omega(u,v,w,0) = \sum_{i+j+k=n} \frac{n!}{i!j!k!} u^i v^j w^k P_{ijk0}$$

which is the expression of a Bézier triangle of degree n defined with $(P_{ijk0})_{0\leq i,j,k\leq n | i+j+k=n}$. The same observation holds for $t = 1$. Also, if $w = 0$ (for instance) then :

$$\omega(u, v = 1-u, 0, t) = \sum_{i=0}^{n} \sum_{l=0}^{m} \binom{n}{i} \binom{m}{l} u^i (1-u)^{n-i} t^l (1-t)^{m-l} P_{i(n-i)0l}$$

which is the expression of a Bézier quadrilateral of degree $n \times m$ defined with $(P_{i(n-i)0l})_{0\leq i,j\leq n}$. The same holds with $u = 0$ and $v = 0$. Using both Bernstein and triangular Bernstein polynomials the expression of a Bézier prism is rewritten as:

$$\omega(u,v,w,t) = \sum_{i+j+k=n} B^n_{ijk}(u,v,w) \sum_{l=0}^{m} B^n_l(t) P_{ijkl},$$

Consequently a Bézier prism can be interpreted as the tensor product of a Bézier curve of degree m with a Bézier triangle of degree n.

Algorithm 15: De Casteljau's evaluation algorithm for a Bézier prism of degree $n \times m$

Function *DeCasteljauPri*
- **Input:** Degrees $(n, m) \in \mathbb{N}^2$, Control Points P_{ijkl}, Parameters $(u, v, w, t) \in [0, 1]^3$ with $u + v + w = 1$
- **Output:** val : value on the Bézier prism corresponding to the parameter (u, v, w, t)
- **for** $l \in \{0, m\}$ **do**
  - **for** $(i, j, k) \in \{0, n\}$ *such that* $i + j + k = n$ **do**
    - $P^*_{ijk} = P_{ijkl}$
  - val_l = DeCasteljauTri$(n, P^*_{ijk}, (u, v, w))$
- val = DeCasteljauCurv(m, val_l, t)

De Casteljau's algorithm

As the Bézier prism is seen as a tensor product of a Bézier curve with a Bézier triangle, the evaluation of a point of parameters (u, v, w, t) is done by applying both the de Casteljau's algorithm for the edges and the triangles. This algorithm is based on the following recursion:

$$
\begin{aligned}
&\forall k \in [\![0, m]\!] \quad \forall i \in [\![0, n]\!] \quad \forall j \in [\![0, n]\!] \quad \forall k \in [\![0, n]\!] | i + j + k = n \\
&\quad D^{00}_{ijkl}(u, v, w, t) = P_{ijkl}, \\
&\forall k \in [\![0, m]\!] \quad \forall r \in [\![1, n]\!], \quad \forall (i, j, k) \in [\![0, n - r]\!] \text{ s.t. } \; i + j + k = n, \\
&\quad D^{r0}_{ijkl}(u, v, w, t) = uD^{(r-1)0}_{(i+1)jkl}(u, v, w, t) + vD^{(r-1)0}_{i(j+1)kl}(u, v, w, t) + wD^{(r-1)0}_{ij(k+1)l}(u, v, w, t), \\
&\forall r \in [\![1, m]\!], \quad \forall k \in [\![0, m - r]\!] \\
&\quad D^{nr}_{000l}(u, v, w, t) = (1 - t)D^{n(r-1)}_{000l}(u, v, w, t) + tD^{n(r-1)}_{000(l+1)}(u, v, w, t).
\end{aligned}
$$

Eventually, $D^{nm}_{0000}(u, v, w, t) = \omega(u, v, w, t)$. The proof of this algorithm is straightforward as the results are already established for edges and triangles. The evaluation algorithm is given in **Algorithm 15**.

Algorithm 16: Subdivision algorithm for a Bézier prism of degree $n \times m$

Function *SubdivisionPri*
- **Input:** Degree $(n, m, p) \in \mathbb{N}^3$, Control Points P_{ijk}, Parameter $(u, v, w) \in [0, 1]^3$
- **Output:** L : array containing the four sets of control points forming the subdivision
- **for** $l \in \{0, m\}$ **do**
  - **for** $(i, j, k) \in \{0, n\}$ *such that* $i + j + k = n$ **do**
    - $P^*_{ijk} = P_{ijkl}$
  - L_l = SubdivisionTri$(n, P^*_{ijk}, (u, v, w))$
- **for** (i, j, k) *s.t* i+j+k=n **do**
  - **for** $l \in \{0, m\}$ **do**
    - $P^*_l = L_l[i][j][k]$
  - L[i][j][k] = SubdivisionCurv(m, P^*_k, w)

Subdivision of Bézier prism

As we will see in Section 3.3, the subdivision of a Bézier prism plays an important role to determine the validity of a high-order prism. Using again the tensor product structure of a Bézier prism, a subdivision is deduced using the subdivision algorithm of a Bézier curve and a Bézier triangle. A subdivision is proposed in **Algorithm 16**.
Note that the number of subdivisions in triangles can either be three or four, depending on the needs and this number does not change the process of the algorithm.

3.2.7 Bézier hexahedron

Definition

For a given set $(P_{ijk})_{0\leq i\leq n,0\leq j\leq m,0\leq k\leq p}$ of $(n+1)(m+1)(p+1)$ points, a Bézier hexahedron (or Bézier hexahedral patch) of degree $n\times m\times p$ (when $m=n=p$, we simply say it is of degree n) is defined using the following algebraic relation:

$$\begin{aligned}&\forall(u,v,w)\mid 0\leq u,v,w\leq 1,\\ \omega(u,v,w)=&\sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{p}\binom{n}{i}\binom{m}{j}\binom{p}{k}u^i(1-u)^{n-i}v^j(1-v)^{m-j}w^k(1-w)^{p-k}P_{ijk}.\end{aligned}$$

Here, (u,v,w) are the reference coordinates $(\hat{x},\hat{y},\hat{z})\in[0,1]^3$. In this context, P_{ijk} are called control points and their dimensions are arbitrary. Note that if (for instance) $w=0$, then:

$$\omega(u,v,0)=\sum_{i=0}^{n}\sum_{j=0}^{m}\binom{n}{i}\binom{m}{j}u^i(1-u)^{n-i}v^j(1-v)^{m-j}P_{ij0}$$

which is the expression of a Bézier quadrilateral of degree $n\times m$ defined with $(P_{ij0})_{0\leq i,j\leq n}$. The same remark stands with $w=1$, $u=\{0,1\}$ and $v=\{0,1\}$. Likewise, Bézier curves are found when two coordinates equal 0.
A Bézier hexahedron is expressed using Bernstein polynomials as follows:

$$\omega(u,v,w)=\sum_{i=0}^{n}B_i^n(u)\sum_{j=0}^{m}B_i^m(v)\sum_{k=0}^{p}B_k^p(v)P_{ijk}\quad\forall(u,v,w)\mid 0\leq u,v,w\leq 1.$$

In particular, we can see that a Bézier hexahedron of degree $n\times m\times p$ can be interpreted as the tensor product of three Bézier curves of degree n, m and p and consequently as the tensor product of a Bézier quadrilateral with a Bézier curve. Henceforth, the observations about regularity and oscillation observed with other elements still hold.

De Casteljau's algorithm

As the Bézier hexahedron is seen as a tensor product of three Bézier curves, the evaluation of a point of parameters (u,v,w) is done by applying thrice the de Casteljau's algorithm

Algorithm 17: De Casteljau's evaluation algorithm for a Bézier hexahedron of degree $n \times m \times p$

Function *DeCasteljauHex*
- **Input:** Degrees $(n, m, p) \in \mathbb{N}^3$, Control Points P_{ijk}, Parameters $(u, v, w) \in [0, 1]^3$
- **Output:** val : value on the Bézier hexahedron corresponding to the parameter (u, v, w)
- **for** $k \in \{0, p\}$ **do**
  - **for** $i \in \{0, n\}$ **do**
    - **for** $j \in \{0, m\}$ **do**
      - $P^*_{ij} = P_{ijk}$
  - $val_k = \text{DeCasteljauQua}((n, m), P^*_{ij}, (u, v))$
- val = $\text{DeCasteljauCurv}(p, val_k, w)$

Algorithm 18: Subdivision algorithm for a Bézier Hexahedron of degree $n \times m \times p$

Function *SubdivisionHex*
- **Input:** Degree $(n, m, p) \in \mathbb{N}^3$, Control Points P_{ijk}, Parameter $(u, v, w) \in [0, 1]^3$
- **Output:** L : array containing the four set of control points forming the subdivision
- **for** $k \in \{0, p\}$ **do**
  - **for** $i \in \{0, n\}$ **do**
    - **for** $j \in \{0, m\}$ **do**
      - $P^*_{ij} = P_{ijk}$
  - $L_k = \text{SubdivisionQua}((n, m), P^*_{ij}, (u, v))$
- **for** $i \in \{0, 2n + 1\}$ **do**
  - **for** $j \in \{0, 2m + 1\}$ **do**
    - **for** $k \in \{0, p\}$ **do**
      - $P^*_k = L_k[i][j]$
    - L[i][j] = $\text{SubdivisionCurv}(p, P^*_k, w)$

for the edges. This algorithm is based on the following recursion:

$$
\begin{aligned}
&\forall i \in [\![0, n]\!] \quad \forall j \in [\![0, m]\!] \quad \forall k \in [\![0, p]\!], \quad D^{000}_{ijk}(u, v, w) = P_{ijk}, \\
&\forall i \in [\![0, n - r]\!] \quad \forall (j, k) \in [\![0, m]\!] \times [\![0, p]\!] \quad \forall r \in [\![1, n]\!], \\
&\quad D^{r00}_{ijk}(u, v, w) = (1 - u) D^{(r-1)00}_{ij}(u, v, w) + u D^{(r-1)00}_{(i+1)j}(u, v, w). \\
&\forall j \in [\![0, m - r]\!] \quad \forall k \in [\![0, p]\!] \quad \forall r \in [\![1, m]\!], \\
&\quad D^{nr0}_{0jk}(u, v, w) = (1 - v) D^{n(r-1)0}_{0jk}(u, v, w) + v D^{n(r-1)0}_{0(j+1)k}(u, v, w). \\
&\forall k \in [\![0, p - r]\!] \quad \forall r \in [\![1, m]\!], \\
&\quad D^{nmr}_{00k}(u, v, w) = (1 - w) D^{nm(r-1)}_{00k}(u, v, w) + w D^{nm(r-1)}_{00(k+1)}(u, v, w).
\end{aligned}
\tag{3.12}
$$

Eventually, $D^{nmp}_{000}(u, v, w) = \omega(u, v, w)$. The proof of this result is straightforward as the result is already established for the curves.
Note that in practice, the two first step of the recursion will be reused from the algorithm for the quadrilaterals. Also, if $n = m = p$, a more simple algorithm can be used, but unlike

the other algorithms it does not provide a subdivision [Borouchaki and George 2017b]. For this reason, we will not speak about it.
The evaluation algorithm is given in **Algorithm 17**.

Subdivision of Bézier hexahedron

As we will see in Section 3.3, the subdivision of a Bézier hexahedron plays an important role to determine the validity of high-order elements. Using again the tensor product structure of a Bézier hexahedron, a subdivision is deduced using the subdivision algorithm of a Bézier curve and a Bézier quadrilateral. A subdivision is proposed in **Algorithm 18**.

3.2.8 Bézier pyramid

Definition

For a given set $(P_{ijk})_{0\leq i,j\leq n-k\leq n}$ of $\frac{(2n+3)(n+2)(n+1)}{6}$ points, a Bézier pyramid (or Bézier pyramidal patch) of degree n is defined using the following algebraic relation [Johnen and Geuzaine 2015, Chan and Warburton 2016]:

$$\begin{aligned}
&\forall (u,v,w) \mid 0 \leq u,v \leq 1-w \leq 1,\\
\omega(u,v,w) = &\sum_{k=0}^{n}\sum_{i=0}^{n-k}\sum_{j=0}^{n-k}\binom{n}{k}\binom{n-k}{i}\binom{n-k}{j}\\
&w^k(1-w)^{n-k}(\frac{u}{1-w})^i(\frac{1-u-w}{1-w})^{n-k-i}(\frac{v}{1-w})^j(\frac{1-v-w}{1-w})^{n-k-j}P_{ijk}.
\end{aligned}$$

(u,v,w) are exactly the reference coordinates $(\hat{x},\hat{y},\hat{z}) \in [0,1]^3$. In this context, P_{ijkl} are called control points and their dimensions are arbitrary. Note that if $w=0$, then:

$$\omega(u,v,0) = \sum_{i=0}^{n}\sum_{j=0}^{n}\binom{n}{i}\binom{n}{j}u^i(1-u)^{n-i}v^j(1-v)^{n-j}P_{ij0},$$

which is the expression of a Bézier quadrilateral of degree n defined with $(P_{ij0})_{0\leq i,j\leq n}$. Also, if $u=0$ (for instance) then :

$$\begin{aligned}
\omega(0,v,w) &= \sum_{k=0}^{n}\sum_{j=0}^{n-k}\frac{n!}{k!(k-i)!i!}w^k(1-w)^{n-k}(\frac{v}{1-w})^j(\frac{1-v-w}{1-w})^{n-k-j}P_{0jk}\\
&= \sum_{k=0}^{n}\sum_{j=0}^{n-k}\frac{n!}{k!(n-k-j)!j!}w^kv^j(1-v-w)^{n-k-j}P_{0jk},
\end{aligned}$$

which is the expression of a Bézier triangle of degree n defined with $(P_{0jk})_{0\leq j+k\leq n}$. The same holds with $u=1$, $v=0$ and $v=1$. Using Bernstein polynomials the expression of a Bézier pyramid is rewritten as:

$$\omega(u,v,w) = \sum_{k=0}^{n}\sum_{i=0}^{n-k}\sum_{j=0}^{n-k}B_k^n(w)B_i^{n-k}(\frac{u}{1-w})B_j^{n-k}(\frac{v}{1-w})P_{ijk}, \quad \forall (u,v,w) \mid 0\leq u,v\leq 1-w\leq 1.$$

De Casteljau's algorithm

Using its expression as a product of Bernstein polynomials, a de Casteljau's algorithm is applied to the Bézier pyramid in order to get its value at a point of parameters (u,v,w). The algorithm is based on the following recursion:

$$\forall k \in [\![0,n]\!] \quad \forall (i,j) \in [\![0,n-k]\!]^2, \quad D_{ijk}^{000}(u,v,w) = P_{ijk},$$

Algorithm 19: De Casteljau's evaluation algorithm for a Bézier pyramid of degree n

Function *DeCasteljauPyr*
- **Input:** Degrees $n \in \mathbb{N}$, Control Points P_{ijk}, Parameters $(u, v, w) \in [0,1]^3$ with $u, v \leq 1 - w$
- **Output:** val : value on the Bézier pyramid corresponding to the parameter (u, v, w)
- **for** $k \in \{0, n\}$ **do**
  - **for** $(i, j) \in \{0, n-k\}$ **do**
    - $P^*_{ij} = P_{ijk}$
  - $val_k = \text{DeCasteljauQua}(n-k, P^*_{ij}, (\frac{u}{1-w}, \frac{v}{1-w}))$
- val = $\text{DeCasteljauCurv}(n, val_k, w)$

$$
\begin{aligned}
&\forall k \in [\![0,n]\!] \quad \forall r \in [\![1,n-k]\!] \quad \forall i \in [\![0,n-k-r]\!] \quad \forall j \in [\![0,n-k]\!],\\
&\quad D^{r00}_{ijk}(u,v,w) = \frac{1-u-w}{1-w} D^{(r-1)00}_{ijk} + \frac{u}{1-w} D^{(r-1)00}_{(i+1)jk},\\
&\forall k \in [\![0,n]\!] \quad \forall r \in [\![1,n-k]\!] \quad \forall j \in [\![0,n-k-r]\!],\\
&\quad D^{(n-k)r0}_{0jk}(u,v,w) = \frac{1-v-w}{1-w} D^{(n-k)(r-1)0}_{0jk} + \frac{v}{1-w} D^{(n-k)(r-1)0}_{0(j+1)k},\\
&\forall r \in [\![0,n]\!] \quad \forall k \in [\![1,n-r]\!],\\
&\quad D^{(n-k)(n-k)r}_{00k} = (1-w) D^{(n-k)(n-k)(r-1)}_{00k} + w D^{(n-k-1)(n-k-1)(r-1)}_{00(k+1)}.
\end{aligned}
$$

Eventually, $D^{nnn}_{000} = \omega(u,v,w)$. Indeed, lines two and three of the recursion will build an exact evaluation of the Bézier quadrilaterals parameterized by $\sum_{i=0}^{n-k}\sum_{j=0}^{n-k} B_i^{n-k}(\frac{u}{1-w}) B_j^{n-k}(\frac{v}{1-w}) P_{ijk}$ for $k \in \{0, n\}$. And the last line will use these evaluation points as control points for a de Casteljau's algorithm for a Bézier curve, which gives the expected result.
The process can be summed up by the **Algorithm 19**.
Finally, the subdivision of Bézier pyramids is not tackled in this section as it will be not used in this work. Nevertheless, note that given its particular definition, a Bézier pyramid cannot easily be subdivided into only Bézier pyramids.

Transformation into a Bézier hexahedron

In a way, the Bézier pyramid is seen as a degenerate Bézier hexahedron. Indeed, if we set the following change of variables $a = \frac{u}{1-w}, b = \frac{v}{1-w}, c = w$, this gives a parameterization which is close to the one of a Bézier hexahedron. In fact, it is possible to rewrite a Bézier pyramid of degree n as a Bézier hexahedron of degree n. The underlying idea is pretty simple, as a Bézier pyramid can be seen as $n+1$ levels of Bézier quadrilaterals of various degrees lower than n. If we apply a degree elevation (which is possible accordingly the properties of the Bernstein polynomials) to each of the layers up to degree n, we finish obtaining $n+1$ layers of Bézier quadrilaterals of degree n which is in fact a Bézier hexahedron of degree n. Indeed, using Eq (iv), we see that a Bézier quadrilateral of degree n is written as:

$$\sigma(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{n} B_i^n(u) B_j^n(v) P_{ij}$$

$$= \sum_{i=0}^{n}\sum_{j=0}^{n}\left(\frac{i+1}{n+1}B_{i+1}^{n+1}(u) + \frac{n+1-i}{n+1}B_i^{n+1}(u)\right)\left(\frac{j+1}{n+1}B_{j+1}^{n+1}(v) + \frac{n+1-j}{n+1}B_j^{n+1}(v)\right)P_{ij}$$
$$= \sum_{i=0}^{n+1}\sum_{j=0}^{n+1} B_i^{n+1}(u)B_j^{n+1}(v)P_{ij}^*,$$

with:

$$P_{ij}^* = \frac{j}{n+1}\left(\frac{i}{n+1}P_{(i-1)(j-1)} + \frac{n+1-i}{n+1}P_{i(j-1)}\right) + \frac{n+1-j}{n+1}\left(\frac{i}{n+1}P_{(i-1)j} + \frac{n+1-i}{n+1}P_{ij}\right),$$

and the convention $P_{ij} = 0$ if $\min(i,j) < 0$ or $\max(i,j) > n$. If we note this formula as an algorithm *QuadDegElevation*(n, P_{ij}), with n the input degree, the algorithm for the transformation of a pyramid into an hexahedron is given in **Algorithm 20**. This algorithm will be used in Section 3.3 to deal with the validity of a Bézier pyramid.

Algorithm 20: Algorithm to transform a Bézier pyramid of degree n into a Bézier hexahedron of degree n

Function *Pyr2Hex*
- **Input:** Degrees $n \in \mathbb{N}$, Control Points P_{ijk}
- **Output:** P_{ijk}^*, control points of the corresponding hexahedron
- **for** $k \in \{0, n\}$ **do**
  - **for** $(i,j) \in \{0, n-k\}$ **do**
    - $P_{ij}^{**} = P_{ijk}$
  - **for** $l \in \{n-k, n-1\}$ **do**
    - $P_{ij}^{**} = QuadDegElevation(l, P_{ij}^{**})$
  - **for** $(i,j) \in \{0, n-k\}$ **do**
    - $P_{ijk}^* = P_{ij}^{**}$

3.3 High-order elements: definition and validity

The aim of the following section is to exhibit some definitions and properties of the high-order finite elements. In particular, we will have a look at the most common type of finite elements.

In general, a finite element is defined (see [Ciarlet 1978], for instance) by the triplet $\{K, \Sigma_K, V_K\}$ where K denotes the geometric element (curve, triangle, etc), Σ_K the list of nodes of K, and V_K, the space of the *shape functions*, here it will be the Lagrange polynomial functions (or interpolants). To properly define the geometry and these functions, a reference space (that can also be seen as a parameter space) is defined where all coordinates are between 0 and 1. In this space, the reference element is denoted $\widehat{K}$ and has a fixed (and sometimes uniform) distribution of nodes. The element K, also called physical element, is thus the image of $\widehat{K}$ via a mapping, denoted F_K. More specifically, for each point M of K, there is a point $\widehat{M}$ of $\widehat{K}$ such that $M = F_K(\widehat{M})$. The exact expression is defined by:

$$M = \sum_{i=1}^{N} \phi_i(\widehat{M})A_i,$$

where N is the number of nodes, $A_i = F_K(\widehat{A}_i)$ with $\widehat{A}_i$ the nodes of the reference element which map to A_i the nodes of the physical element, and ϕ_i are the Lagrange polynomial

functions and each of them is defined such that:

$$\phi_i(\widehat{A}_j) = \delta_{ij} \quad \text{and} \quad \sum_{i=1}^{N} \phi_i = 1.$$

Every element has its own shape functions and its own reference element. In the following sections, we will study elements defined on seven different reference elements and then we will define their associated shape functions so that it defines a (complete) finite element of order n.

3.3.1 High-order curve

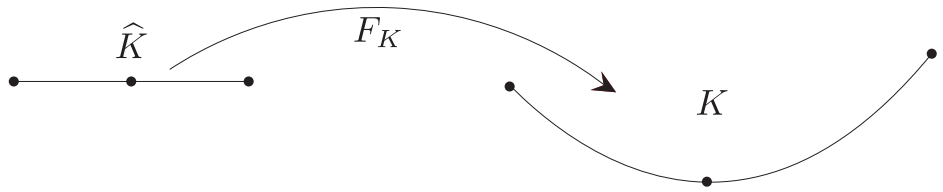

Figure 3.4 – Mapping F_K from $\widehat{K}$ to K for a P^2 curve.

Definition

A finite-element curve is defined using the following reference element:

$$\widehat{K} = \{0 \leq (u, v) \leq 1 \mid u + v = 1\} = \{\widehat{x} \mid 0 \leq \widehat{x} \leq 1\}.$$

The first definition (simplicial case) is a definition of the parameters space as a simplex of dimension 1 while the second one (tensorial case) defines the parameters as the interval $[0, 1]$. $\widehat{x}$ and (u, v) are related via $u = 1 - \widehat{x},\ v = \widehat{x}$.
For each of the definition, the following (Lagrange) shape functions of degree n are introduced (where power s stands for simplicial):

$$\phi_i^s(u) = \frac{1}{i!} \prod_{l=0}^{i-1} (nu - l) \quad \text{for} \quad 1 \leq i \leq n \quad \text{and} \quad \phi_0(u) = 1,$$
$$\phi_{ij}^s(u, v) = \phi_i^s(u)\phi_j^s(v) \quad \text{for} \quad i + j = n.$$

and (where power t stands for tensorial)

$$\phi_i^t(\widehat{x}) = \frac{(-1)^i}{i!(n-i)!} \prod_{\substack{l=0 \\ l \neq i}}^{k} (l - n\widehat{x}).$$

The link between both formulas is:

$$\phi_i^t(\widehat{x}) = \phi_{(n-i)i}^s(1 - \widehat{x}, \widehat{x}).$$

The polynomial space of shape functions of degree n is consequently defined as:

$$V_K = \{u^i v^j \mid i + j = n\} = \{\widehat{x}^i \mid 0 \leq i \leq n\}.$$

In particular, we can see that the space of degree n contains all the space of lower degree. Also, if we note $u_k^n = \frac{k}{n}$, $\widehat{x}_l^n = v_l^n = \frac{l}{n} = 1 - u_k^n$ with $k + l = n$, then we have:

$$\phi_{ij}^s(u_k^n, v_l^n) = \delta_{ik}\delta_{jl} \quad \text{or equivalently} \quad \phi_i^t(\widehat{x}_l^n) = \delta_{il}.$$

Concretely this means that these shape functions are the Lagrange polynomials associated with the nodes of parameters (u_i^n, v_j^n). Here, for the sake of simplicity, the choice has been made to use a uniform distribution of nodes on the reference element. But from a numerical point of view, a more clever choice can be made on the distribution of nodes to define the same finite element space [Warburton 2006].
A (finite element) curve of degree n, or P^n-curve, is consequently defined by a set of $n+1$ points $(A_i)_{0\leq i\leq n}$ of arbitrary dimension using the following relation:

$$M = \gamma(\widehat{x}) = \sum_{i=0}^{n} \phi_i^t(\widehat{x}) A_i.$$

Of course, the same definition can be done with ϕ^s. An example of P^2-curve is presented in Figure 3.4.

Transformation into a Bézier curve

As the Bernstein polynomials of degree n span the finite element space of curves of degree n (see Section 3.2.2), a curve of degree n can be rewritten as a Bézier curve of degree n. In practice, the transformation is done by solving the following system:

$$\sum_{i=0}^{n} B_i^n(u_i^n) P_i = A_i, \ \forall i \in [\![0, n]\!],$$

where we recall that the P_i are the control points of the Bézier curve.
In practice, the system is only solved for $i \in \{1, .., n-1\}$ as $P_0 = A_0$ and $P_n = A_n$. In P^2, for instance, it gives us the following result:

$$P_1 = \frac{4A_1 - A_0 - A_2}{2}.$$

3.3.2 High-order triangle

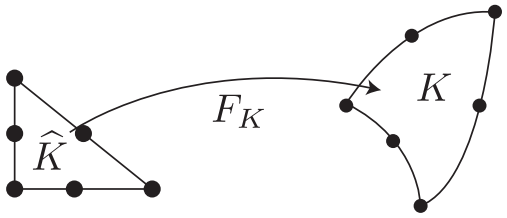

Figure 3.5 – Mapping F_K from $\widehat{K}$ to K for a P^2 triangle.

Definition

A finite-element triangle is defined using the following reference element:

$$\widehat{K} = \{0 \leq (u, v, w) \leq 1 \mid u + v + w = 1\} = \{0 \leq (\widehat{x}, \widehat{y}) \leq 1 \mid 0 \leq \widehat{x} + \widehat{y} \leq 1\}.$$

The first definition uses the fact that the triangle is a simplex of dimension two while the second one defines the element as the first half of an unit square. The link between the two expressions is done via $u = 1 - \widehat{x} - \widehat{y}$, $v = \widehat{x}$ and $w = \widehat{y}$. Using its definition as a simplex, the following shape functions of degree n are introduced:

$$\phi_{ijk}(u, v, w) = \phi_i^s(u)\phi_j^s(v)\phi_k^s(w) \quad \text{for} \quad i + j + k = n,$$

where ϕ_i^s is the simplicial shape function defined for the high-order curve. The polynomials space spanned by the shape functions is therefore:

$$V_K = \{u^i v^j w^k \mid i + j + k = n\} = \{\widehat{x}^i \widehat{y}^j \mid 0 \leq i + j \leq n\}.$$

For the same reason as curves, the shape functions are Lagrange polynomials corresponding to the nodes $(u_i^n, v_j^n, w_k^n) = (\frac{i}{n}, \frac{j}{n}, \frac{k}{n})$.
A (finite element) triangle of degree n, or P^n-triangle, is consequently defined as:

$$M = \sigma(u, v, w) = \sum_{i+j+k=n} \phi_{ijk}(u, v, w) A_{ijk}.$$

where A_{ijk} are the nodes whose dimensions are arbitrary. An example of P^2-triangle is presented in Figure 3.5.

Transformation into a Bézier triangle

As the triangular Bernstein polynomials of degree n span the finite element space of triangles of degree n (see Section 3.2.3), a triangle of degree n can be rewritten as a Bézier triangle of degree n. In practice, the transformation is done by solving the following system:

$$\sum_{i+j+k=n} B_{ijk}^n(u_i^n, v_j^n, w_k^n) P_{ijk} = A_{ijk}, \ \forall (i, j, k) \in [\![0, n]\!] \quad \text{such that} \quad i + j + k = n,$$

where we recall that P_{ijk} are the control points of the Bézier triangle.
In practice, as $B_{ijk}^n(u_i^n, v_j^n, 0) = 0$ if $k > 0$ (for instance), the edges of the triangle can be seen as finite element curves, they are transformed into Bézier curves, and then using this result, the previous system is solved for all the internal points of the triangle (*e.g.* the points that are not on one of the edges of the triangle).

Validity of a high-order triangle in 2D

To deal with the validity of a high-order curve, we need to have a transformation between two spaces of equal dimension. Therefore, we will consider in this section that the $(A_i)_{0 \leq i \leq n}$ are vectors of $\mathbb{R}^2$ (and consequently the P_i).
The validity of an high-order triangle means that the associated mapping F_K is a diffeomorphism. It can be ensured if the minimum of the determinant $\mathcal{J}_K$ of the Jacobian matrix of the mapping F_K is strictly positive everywhere inside the element [George and Borouchaki 2012]. The Jacobian $\mathcal{J}_K$ writes as polynomial of degree $2(n-1)$. When the element is of degree 1 (*e.g.* straight-sided), it simply means that the oriented area is strictly positive. As the Jacobian is a polynomial defined on the triangular finite element space, it can also be expressed in the triangular Bernstein polynomial basis

(Theorem of Panzoult, see [George and Borouchaki 2012]) as follows [3]:

$$\mathcal{J}_K(u,v,w) = \det(\frac{\partial F_K}{\partial \widehat{x}}, \frac{\partial F_K}{\partial \widehat{y}}) = \det(\frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u}, \frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial u})$$
$$= \det(\frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial w}, \frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial u}).$$

Now, as we have:

$$\frac{\partial F_K}{\partial u} = n \sum_{i+j+k=n-1} B_{ijk}^{n-1}(u,v,w) P_{(i+1)jk},$$

and the same formula for the other variables. The Jacobian is rewritten as:

$$\mathcal{J}_K(u,v,w)$$
$$= n^2 \det(\sum_{i+j+k=n-1} B_{ijk}^{n-1}(u,v,w)\overrightarrow{P_{ij(k+1)}P_{i(j+1)k}}, \sum_{i+j+k=n-1} B_{ijk}^{n-1}(u,v,w)\overrightarrow{P_{(i+1)jk}P_{ij(k+1)}})$$
$$= n^2 \det(\sum_{i+j+k=n-1} B_{ijk}^{n-1}(u,v,w)\overrightarrow{P_{(i+1)jk}P_{i(j+1)k}}, \sum_{i+j+k=n-1} B_{ijk}^{n-1}(u,v,w)\overrightarrow{P_{(i+1)jk}P_{ij(k+1)}}).$$

Finally, by applying the multiplicative properties of the Bernstein polynomials, it comes that:

$$\mathcal{J}_K(u,v,w) = \sum_{i+j+k=2(n-1)} B_{ijk}^{2(n-1)}(u,v,w) N_{ijk},$$

where:

$$N_{ijk} = n^2 \sum_{\substack{i_1+i_2=i \\ j_1+j_2=j \\ k_1+k_2=k}} \frac{i!j!k!((n-1)!)^2}{(2(n-1))!i_1!i_2!j_1!j_2!k_1!k_2!} \det(\overrightarrow{P_{(i_1+1)j_1k_1}P_{i_1(j_1+1)k_1}}, \overrightarrow{P_{(i_2+1)j_2k_2}P_{i_2j_2(k_2+1)}}),$$

with $i_1+j_1+k_1 = i_2+j_2+k_2 = n-1$. N_{ijk} will be called a control coefficient and their expression as determinants give them a geometrical meaning. In P^2 (see Figure 3.6), a corner and an edge control coefficients [Borouchaki and George 2017b] are for example:

$$N_{200} = 4\det(\overrightarrow{P_{200}P_{110}}, \overrightarrow{P_{200}P_{101}}), \tag{3.13}$$
$$N_{110} = 2\det(\overrightarrow{P_{200}P_{110}}, \overrightarrow{P_{110}P_{011}}) + 2\det(\overrightarrow{P_{110}P_{020}}, \overrightarrow{P_{200}P_{101}}). \tag{3.14}$$

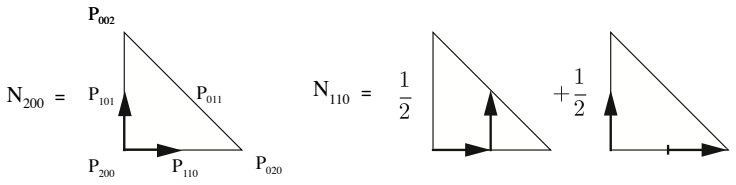

Figure 3.6 – Vectors involved in the determinant for the computation of a corner (left) and an edge (right) control coefficients of a P^2 triangle.

3. It can be noted that F_K is dependent on the choice of the half-square for reference element and that consequently four transformations are possible. However, the Jacobian of these four expressions is the same which is why we choose arbitrary one transformation for our analysis. This observation holds for the tetrahedron and the prism.

These control coefficients also appear in the computation of the area of a P^n-triangle. Indeed, using the fact (see Section 3.2.3 and the property vi) of the triangular Bernstein polynomials) that $\int_{\widehat{K}} B^n_{ijk}(u,v,w)\mathrm{d}K = \frac{1}{(n+1)(n+2)}$, we have:

$$\begin{aligned}|K| &= \int_{\widehat{K}} \mathcal{J}_K(u,v,w)\mathrm{d}K \\ &= \sum_{i+j+k=2(n-1)} N_{ijk} \int_{\widehat{K}} B^{2(n-1)}_{ijk}(u,v,w)\mathrm{d}K \\ &= \frac{2}{(2(n-1)+1)(2(n-1)+2)} \sum_{i+j+k=2(n-1)} \frac{N_{ijk}}{2}.\end{aligned}$$

In other words, the area of a triangle of degree n is the average of all the control coefficients multiplied by the area of the reference element that is $\frac{1}{2}$.
The triangular Bernstein polynomials are always positive on the reference element and realize a partition of unity. Consequently, a sufficient condition to prove that $\mathcal{J}_K$ is strictly positive everywhere is to ensure that all N_{ijk} are strictly positive, but this is too strong a condition if $n > 1$. On the contrary, if a N_{ijk} is negative in a corner, it means that $\mathcal{J}_K$ is negative somewhere inside the element as N_{ijk} is the exact value of the Jacobian in this corner [George and Borouchaki 2012]. However, if a control coefficient lying on an edge or on a face is negative, it does not mean that $\mathcal{J}_K$ is negative somewhere inside the element. We cannot conclude on the positiveness of the Jacobian without any further analysis. In this case, a few iterations of a subdivision algorithm [George and Borouchaki 2012, Johnen et al. 2013] are required to have more accurate bounds of the Jacobian. The subdivision techniques that are used are the ones that are presented in Sections 3.2.2 and 3.2.3 and the used strategy is the following: at the beginning, the value at the corners is checked, if the values are positive, the value of the control coefficient lying on the edges are examined. If a negative value is detected, the subdivision algorithm for the curve is applied to conclude on the validity of the jacobian on the edge. If the validity remains, the inner control coefficients are considered. If one of them is negative, a subdivision of the triangle is performed to conclude. In order to keep consistency with the analysis done on the edges, and also because it is mandatory to make the process work, a subdivision in four of the triangle is used. Similarly to curves, the triangles are subdivided as many times as necessary. For numerical reasons, a limit of the number of nested subdivisions is set. If at this limit no conclusion has been possible, the element is declared invalid. This way a hierarchical method to deal with the validity of a P^n-triangle in 2D is set.

3.3.3 High-order quadrilateral

Definition

A finite-element quadrilateral is defined using the following reference element:

$$\widehat{K} = \{0 \leq (\widehat{x},\widehat{y}) \leq 1\} = [0,1]^2.$$

The reference element defines the element as a tensor product of two reference edges. Based on this definition, the following shape functions are introduced:

$$\phi_{ij}(\widehat{x},\widehat{y}) = \phi^t_i(\widehat{x})\phi^t_j(\widehat{y}),$$

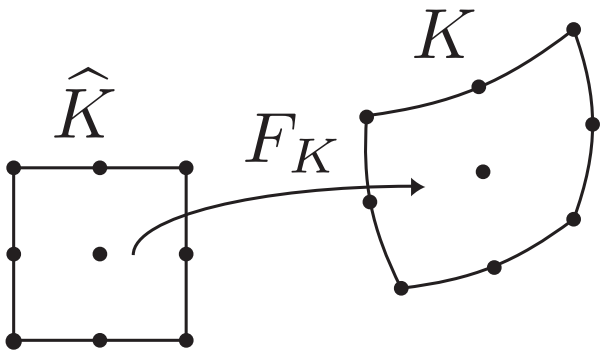

Figure 3.7 – Mapping F_K from $\widehat{K}$ to K for a Q^2 quadrilateral.

where ϕ_i^t is the tensorial shape function defined for the high-order curve of degree n.[4] The polynomial space spanned by the shape functions is therefore:

$$V_K = \{\widehat{x}^i \widehat{y}^j \mid 0 \leq (i,j) \leq n\}.$$

Naturally, the shape functions are Lagrange polynomials corresponding to the nodes $(x_i^n, y_j^n) = (\frac{i}{n}, \frac{j}{n})$.
A (finite element) quadrilateral of degree n, or Q^n-quadrilateral, is consequently defined as:

$$M = \sigma(\widehat{x}, \widehat{y}) = \sum_{i=0}^{n} \sum_{j=0}^{n} \phi_{ij}(\widehat{x}, \widehat{y}) A_{ij},$$

where A_{ij} are the nodes whose dimension is arbitrary. An example of Q^2-quadrilateral is presented in Figure 3.7.

About the decomposition into triangles: When dealing with high-order quadrilaterals, a natural question is their decomposition into high-order triangles. If we analyze the finite element spaces spanned by the quadrilaterals we can see that the monomial basis of the quadrilateral finite element space of degree n is always contained in a triangular finite element space of degree greater or equal than $2n$ [Moxey et al. 2015a]. This is due to, among others, the presence of the monomial term $\{\widehat{x}^n \widehat{y}^n\}$ in the quadrilateral finite element basis of degree n. Consequently, any valid quadrilateral of degree n can be safely divided into two valid triangles if and only if their degree is greater or equal than $2n$.

Transformation into a Bézier quadrilateral

As Bernstein polynomials of degree n span the finite element space of the edges of degree n, the product of two Bernstein polynomials of degree n spans the finite element space of the quadrilaterals of degree n. Therefore, a high-order quadrilateral can be written as a Bézier quadrilateral. In practice, the transformation is done by solving the following system:

$$\sum_{i=0}^{n} \sum_{j=0}^{n} B_i^n(x_i^n) B_j^n(y_j^n) P_{ij} = A_{ij}, \ \forall (i,j) \in [\![0,n]\!],$$

where we recall that the P_{ij} are the control points of the Bézier quadrilateral.
In practice, as $B_i^n(0) = 0$ if $i > 0$ and $B_i^n(1) = 0$ if $i < n$, the edges of the quadrilateral can be seen as finite element curves, they are transformed into Bézier curves, and then using

4. In theory, it is possible to define different degrees in each direction, however, apart from the case of structured lattices, this case is not possible in practice.

this result, the previous system is solved for all the internal points of the quadrilateral (*e.g.* the points that are not on one of the edges of the quadrilateral).

Validity of a high-order quadrilateral in 2D

For the same reasons as with the triangle, $A_{ij} \in \mathbb{R}^2$. Now the Jacobian is a polynomial function of degree $2n-1$, and can be written in Bézier form. Using the same kind of derivations as for the triangles and the curves. We obtain that the Jacobian $\mathcal{J}_K$ is written as:

$$\mathcal{J}_K(\widehat{x},\widehat{y}) = \sum_{i=0}^{2n-1}\sum_{j=0}^{2n-1} B_i^{2n-1}(\widehat{x})B_j^{2n-1}(\widehat{y})N_{ij},$$

where N_{ij} are defined as:

$$N_{ij} = n^2 \sum_{i_1+i_2=i}\sum_{j_1+j_2=j} \frac{\binom{n-1}{i_1}\binom{n}{i_2}}{\binom{2n-1}{i}}\frac{\binom{n}{j_1}\binom{n-1}{j_2}}{\binom{2n-1}{j}} \det(\overrightarrow{P_{i_1j_1}P_{(i_1+1)j_1}}, \overrightarrow{P_{i_2j_2}P_{i_2(j_2+1)}}),$$

with $(i_1, j_2) \in [\![0, n-1]\!]$ and $(i_2, j_1) \in [\![0, n]\!]$. Once again, these control coefficients have a geometrical meaning if we have a look at their expression as determinants. Also, using that $\int_0^1 B_i^{2n-1}(\widehat{x})\mathrm{d}\widehat{x} = \frac{1}{2n}$ ()see Section 3.2.2 and the property vi) of the Bernstein polynomials), we can see that:

$$|K| = \frac{1}{(2n)^2}\sum_{i=0}^{2n-1}\sum_{j=0}^{2n-1} N_{ij}.$$

In other words, the area of a Q^n-quadrilateral is the average of all its control coefficients. The validity of a quadrilateral is ensured if its Jacobian is strictly positive everywhere. In Q^1, this means that the quadrilateral has to be convex. It can be ensured if all its control coefficients are positive. But for $n > 1$, the opposite is not necessarily true. Again, a subdivision procedure can be performed to conclude on the validity of a quadrilateral. The subdivision techniques that are used are the ones that are presented in Sections 3.2.2 and 3.2.4 and the used strategy is the same as the one used with the triangles: at the beginning, the value at the corners is checked, if the values are positive, the value of the control coefficient lying on the edges are examined. If a negative value is detected, the subdivision algorithm for the curve is applied to conclude on the validity of the Jacobian on the edge. If the validity remains, the inner control coefficients are considered. If one of them is negative, a subdivision of the quadrilateral is performed to conclude. Similarly to curves, the quadrilaterals are subdivided as many times as necessary. For numerical reasons, a limit of the number of nested subdivisions is set. If at this limit no conclusion has been possible, the element is declared invalid. This way a hierarchical method to deal with the validity of a Q^n-quadrilateral in 2D is set.

3.3.4 High-order tetrahedron

Definition

A finite-element tetrahedron is defined using the following reference element:

$$\widehat{K} = \{0 \le (u,v,w,t) \le 1 \mid u+v+w+t = 1\} = \{0 \le (\widehat{x},\widehat{y}),\widehat{z} \le 1 \mid 0 \le \widehat{x}+\widehat{y}+\widehat{z} \le 1\}.$$

The first definition uses the fact that the tetrahedron is a simplex of dimension three while the second one defines the element as a portion of an unit cube. The link between the two

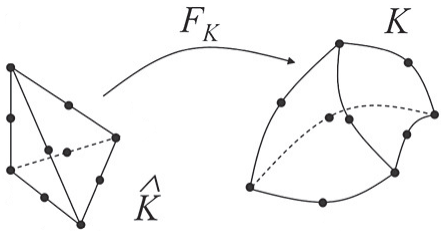

Figure 3.8 – Mapping F_K from $\widehat{K}$ to K for a P^2 tetrahedron.

expressions can be done via $u = 1 - \widehat{x} - \widehat{y} - \widehat{z}$, $v = \widehat{x}$, $w = \widehat{y}$ and $t = \widehat{z}$. Using its definition as a simplex, the following shape functions of degree n are introduced:

$$\phi_{ijkl}(u, v, w, t) = \phi_i^s(u)\phi_j^s(v)\phi_k^s(w)\phi_l^s(t) \quad \text{for} \quad i + j + k + l = n,$$

where ϕ_i^s is the simplicial shape function defined for the high-order curve. The polynomial space spanned by the shape functions is therefore:

$$V_K = \{u^i v^j w^k t^l \mid i + j + k + l = n\} = \{\widehat{x}^i \widehat{y}^j \widehat{z}^k \mid 0 \leq i + j + k \leq n\}.$$

Like for triangles, the shape functions are Lagrange polynomials corresponding to the nodes $(u_i^n, v_j^n, w_k^n, t_l^n) = (\frac{i}{n}, \frac{j}{n}, \frac{k}{n}, \frac{l}{n})$.
A (finite element) tetrahedron of degree n, or P^n-tetrahedron, is consequently defined as:

$$M = \omega(u, v, w, t) = \sum_{i+j+k+l=n} \phi_{ijkl}(u, v, w, t) A_{ijkl},$$

where A_{ijkl} are the nodes whose dimensions are arbitrary. An example of P^2-tetrahedron is presented in Figure 3.8.

Transformation into a Bézier tetrahedron

As the tetrahedral Bernstein polynomials of degree n span the finite element space of tetrahedra of degree n (see Section 3.2.5), a tetrahedron of degree n can be rewritten as a Bézier tetrahedron of degree n. In practice, the transformation is done by solving the following system:

$$\sum_{i+j+k+l=n} B_{ijkl}^n(u_i^n, v_j^n, w_k^n, t_l^n) P_{ijkl} = A_{ijkl}, \ \forall (i, j, k, l) \in [\![0, n]\!] \quad \text{such that} \quad i + j + k + l = n,$$

where we recall that P_{ijkl} are the control points of the Bézier tetrahedron.
In practice, as $B_{ijkl}^n(u_i^n, v_j^n, w_k^n, 0) = 0$ if $l > 0$ (for instance), the faces (resp. the edges) of the tetrahedron can be seen as finite element triangles (resp. curves), they are transformed into Bézier triangles (resp. curves), and then using this result, the previous system is solved for all the internal points of the tetrahedron (*e.g.* the points that are not on one of the edges and faces of the tetrahedron).

Validity of a high-order tetrahedron in 3D

In this section, $A_{ijk} \in \mathbb{R}^3$. Now the Jacobian is a polynomial function of degree $3(n-1)$, and can be written in Bézier form. Using the same kind of derivations as for the triangles and the curves, we obtain that the Jacobian $\mathcal{J}_K$ writes as follows:

$$\mathcal{J}_K(u, v, w, t) = \sum_{i+j+k+l=3(n-1)} B_{ijkl}^{3(n-1)}(u, v, w, t) N_{ijkl},$$

where N_{ijkl} are defined as:

$$N_{ijkl} = n^3 \sum_{\substack{i_1+i_2+i_3=i \\ j_1+j_2+j_3=j \\ k_1+k_2+k_3=k \\ l_1+l_2+l_3=l}} \frac{((n-1)!)^3 i!j!k!l!}{(3(n-1))\prod_{\alpha=1}^{3}(i_\alpha!j_\alpha!k_\alpha!l_\alpha!)}$$
$$\det(\overrightarrow{P_{(i_1+1)j_1k_1l_1}P_{i_1(j_1+1)k_1l_1}}, \overrightarrow{P_{(i_2+1)j_2k_2l_2}P_{i_2j_2(k_2+1)l_2}}, \overrightarrow{P_{(i_3+1)j_3k_3l_3}P_{i_3j_3k_3(l_3+1)}}),$$

with $i_1 + j_1 + k_1 + l_1 = i_2 + j_2 + k_2 + l_2 = i_3 + j_3 + k_3 + l_3 = n - 1$. Once again, these control coefficients have a geometrical meaning if we have a look at their expression as determinants.
These control coefficients also appear in the computation of the volume of a P^n-tetrahedron. Indeed, using the fact (see section 3.2.5) that $\int_{\widehat{K}} B^n_{ijk}(u,v,w)\mathrm{d}K = \frac{1}{(n+1)(n+2)(n+3)}$, we have:

$$\begin{aligned}
|K| &= \int_{\widehat{K}} \mathcal{J}_K(u,v,w,t)\mathrm{d}K \\
&= \sum_{i+j+k=3(n-1)} N_{ijkl} \int_{\widehat{K}} B^{3(n-1)}_{ijk}(u,v,w,t)\mathrm{d}K \\
&= \frac{6}{(3(n-1)+1)(3(n-1)+2)(3(n-1)+3)} \sum_{i+j+k=3(n-1)} \frac{N_{ijkl}}{6}.
\end{aligned}$$

In other words, the volume of a tetrahedron of degree n is the average of all the control coefficients multiplied by the area of the reference element that is $\frac{1}{6}$.
The tetrahedral Bernstein polynomials are always positive on the reference element and realize a partition of unity. Consequently, a sufficient condition to prove that $\mathcal{J}_K$ is strictly positive everywhere is to ensure that all N_{ijkl} are strictly positive, but this is too strong a condition if $n > 1$. On the contrary, if a N_{ijkl} is negative in a corner, it means that $\mathcal{J}_K$ is negative somewhere inside the element as N_{ijkl} is the exact value of the Jacobian in this corner [George and Borouchaki 2012]. Again, a subdivision procedure can be performed to conclude on the validity of a tetrahedron. The subdivision techniques that are used are the ones that are presented in Sections 3.2.2, 3.2.3 and 3.2.5 and the used strategy is the same as used with the triangles: at the beginning, the value at the corners is checked, if the values are positive, the value of the control coefficient lying on the edges are examined. If a negative value is detected, the subdivision algorithm for the curve is applied to conclude on the validity of the Jacobian on the edge. If the validity remains, the faces' inner control coefficients are considered. If one of them is negative, a subdivision of the triangular face is performed to conclude. In order to keep consistency with the analysis done on the edges, and also because it is mandatory to make the process work, a subdivision in four of the face is used. If the validity remains, the volume's inner control coefficients are considered. If one of them is negative, a subdivision of the tetrahedron is performed to conclude. In order to keep consistency with the analysis done on both edges and faces, a subdivision in eight of the volume is used. Similarly to other elements, the tetrahedra are subdivided as many times as necessary. For numerical reasons, a limit of the number of nested subdivisions is set. If at this limit no conclusion has been possible, the element is declared invalid. This way a hierarchical method to deal with the validity of a P^n-tetrahedron in 3D is set.

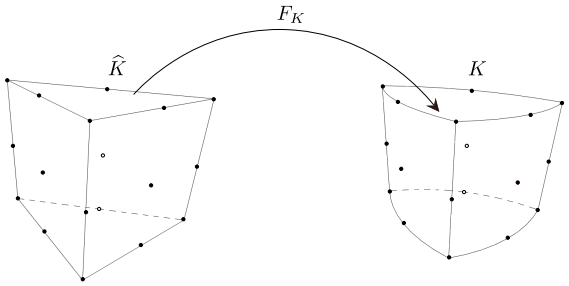

Figure 3.9 – Mapping F_K from $\widehat{K}$ to K for a P^2 prism.

3.3.5 High-order prism

Definition

A finite-element prism is defined using the following reference element:

$$\widehat{K} = \{0 \leq (u,v,w,t) \leq 1 \mid u+v+w=1\} = \{0 \leq (\widehat{x},\widehat{y},\widehat{z}) \leq 1 \mid 0 \leq \widehat{x}+\widehat{y} \leq 1\}.$$

This reference element is a trade-off between simplicial and tensorial elements and has in the same time quadrilateral and triangular faces. This element is a so-called hybrid element. The first definition represents the prism as an extrusion of triangle, while the second one represents it as the half of a unit cube. The equivalence between both definitions is ensured via the relation: $u = 1-\widehat{x}-\widehat{y}$, $v=\widehat{x}$, $w=\widehat{y}$ and $t=\widehat{z}$. Using this definition, the following shape functions are introduced:

$$\phi_{ijkl}(u,v,w,t) = \phi_i^s(u)\phi_j^s(v)\phi_k^s(w)\phi_l^t(t) \quad \text{for} \quad i+j+k=n,$$

where ϕ_i^s (resp. ϕ_i^t) is the barycentric (resp. tensorial) shape function defined for the high-order curve of degree n.[5] The polynomial space spanned by the shape functions is therefore:

$$V_K = \{u^i v^j w^k t^l \mid i+j+k=n \quad 0 \leq l \leq n\} = \{\widehat{x}^i\widehat{y}^j\widehat{z}^k \mid 0 \leq i+j,k \leq n\}.$$

Naturally, the shape functions are Lagrange polynomials corresponding to the nodes $(u_i^n, v_j^n, w_k^n, t_l^n) = (\frac{i}{n}, \frac{j}{n}, \frac{k}{n}, \frac{l}{n})$.
A (finite element) prism of degree n, or P^n-prism, is consequently defined as:

$$M = \omega(u,v,w,t) = \sum_{i+j+k=n} \sum_{l=0}^{n} \phi_{ijkl}(u,v,w,t) A_{ijkl}.$$

where A_{ijkl} are the nodes whose dimensions are arbitrary. An example of P^2-prism is presented in Figure 3.9.

About the decomposition into tetrahedra: When dealing with high-order prisms, a natural question is their decomposition into high-order tetrahedra. If we analyze the finite element spaces spanned by the prisms, we can see that the monomial basis of the prismatic finite element space of degree n is always contained in a tetrahedral finite element space of degree greater or equal than $2n$ [Moxey et al. 2015a]. This is due to, among others, the presence of the monomial term $\{\widehat{x}^n\widehat{z}^n\}$ or $\{\widehat{y}^n\widehat{z}^n\}$ in the prismatic finite element basis of degree n. Consequently, any valid prism of degree n can be safely divided into three valid tetrahedra if and only if their degree is greater or equal than $2n$.

5. In theory, it is possible to define different degrees for the tensorial and simplicial part, however, apart from the case of structured lattices, this case is not possible in practice.

Transformation into a Bézier prism

As a prism is a tensor product of a triangle of degree n and an edge of degree n, it can be written into a Bézier form as both triangular and standard Bernstein polynomials span their associated finite element space. In practice, the transformation is done by solving the following system:

$$\sum_{i+j+k=n} \sum_{l=0}^{n} B_{ijk}^n(u_i^n, v_j^n, w_k^n) B_l^n(t_l^n) P_{ijkl} = A_{ijkl}, \ \forall (i,j,k,l) \in [\![0,n]\!],$$

where we recall that P_{ijkl} are the control points of the Bézier prism.
In practice, as $B_{ijk}^n(u,v,0) = 0$ if $k > 0$ (for instance), $B_i^n(0) = 0$ if $i > 0$ and $B_i^n(1) = 0$ if $i < n$, the faces (resp. edges) of the prism can be seen as finite element triangles/quadrilaterals (resp. curves), they are transformed into Bézier triangles/quadrilaterals (resp. curves), and then using this result, the previous system is solved for all the internal points of the prism (*e.g.* the points that are not on one of the edges/faces of the prism).

Validity of a high-order prism in 3D

For the same reasons as with the tetrahedron, $A_{ijkl} \in \mathbb{R}^3$. Now the Jacobian is a polynomial function of degree $3n-2$ in (u,v,w) and $3n-1$ in t, and can be written in Bézier form as Bézier prism of degree $3n-2 \times 3n-1$. The Jacobian $\mathcal{J}_K$ can be written as:

$$\mathcal{J}_K(u,v,w,t) = \sum_{i+j+k=3n-2} \sum_{l=0}^{3n-1} B_{ijk}^{3n-2}(u,v,w) B_l^{3n-1}(t) N_{ijkl},$$

where N_{ijkl} is defined as:

$$N_{ijkl} = n^3 \sum_{\substack{i_1+i_2+i_3=i \\ j_1+j_2+j_3=j \\ k_1+k_2+k_3=k \\ l_1+l_2+l_3=l}} \frac{((n-1)!)^2 n! i! j! k!}{(3n-2)! \prod_{\alpha=1}^{3} i_\alpha! j_\alpha! k_\alpha!} \frac{\binom{n}{l_1}\binom{n}{l_2}\binom{n-1}{l_3}}{\binom{3n-1}{l}} \times$$
$$\det(\overrightarrow{P_{(i_1+1)j_1k_1l_1}P_{i_1(j_1+1)k_1l_1}}, \overrightarrow{P_{(i_2+1)j_2k_2l_2}P_{i_2j_2(k_2+1)l_2}}, \overrightarrow{P_{i_3j_3k_3l_3}P_{i_3j_3k_3(l_3+1)}}),$$

with $i_1+j_1+k_1 = i_2+j_2+k_2 = i_3+j_3+k_3 = n$, $(l_1,l_2) \in [\![0,n]\!]$ and $l_3 \in [\![0,n-1]\!]$. Once again, these control coefficients have a geometrical meaning if we have a look at their expression as determinants. Also, using the integration results obtained for the quadrilateral and triangle, we can see that:

$$|K| = \frac{2}{(3n)(((3n-2)+1)((3n-2)+2))} \sum_{i+j+k=3n-2} \sum_{l=0}^{3n-1} \frac{N_{ijkl}}{2},$$

which means that the volume of a P^n-prism is the average of all the control coefficients multiplied by the volume of the reference prism which is $\frac{1}{2}$.
The validity of a prism is ensured if its Jacobian is strictly positive everywhere. It can be ensured if all its control coefficients are positive. But for $n > 1$, the opposite is not necessarily true. In P^1, this is already not that straightforward as the Jacobian is of degree 1×2: all its decompositions in tetrahedra have to be checked to ensure its validity. This becomes more complex for higher degrees. Again, a subdivision procedure can be performed to conclude on the validity of a prism. The subdivision techniques that are used are the ones that are presented in Sections 3.2.2, 3.2.3, 3.2.4 and 3.2.6 and the used strategy is still the same: at the beginning, the value at the corners is checked, if the values are positive,

the value of the control coefficient lying on the edges are examined. If a negative value is detected, the subdivision algorithm for the curve is applied to conclude on the validity of the Jacobian on the edge. If the validity remains, the faces' inner control coefficients are considered. If one of them is negative, a subdivision of the quadrilateral (note that the quadrilateral has different degrees depending on the direction) face or a subdivision in four (for consistency purposes with the edges) of the triangular face is performed to conclude. If the validity remains, the volume's inner control coefficients are considered. If one of them is negative, a subdivision of the prism is performed to conclude. Similarly to other elements, the prisms are subdivided as many times as necessary. For numerical reasons, a limit of the nested subdivisions is set. If at this limit no conclusion has been possible, the element is declared invalid. This way a hierarchical method to deal with the validity of a P^n-prism in 3D is set.

3.3.6 High-order hexahedron

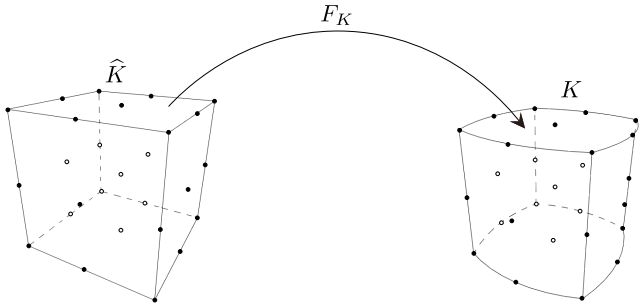

Figure 3.10 – Mapping F_K from $\widehat{K}$ to K for a Q^2 hexahedron.

Definition

A finite-element hexahedron is defined using the following reference element:

$$\widehat{K} = \{0 \leq (\widehat{x}, \widehat{y}), \widehat{z} \leq 1\} = [0,1]^3.$$

The reference element defines the element as a tensor product of three reference edges. Based on this definition, the following shape functions are introduced:

$$\phi_{ij}(\widehat{x}, \widehat{y}, \widehat{z}) = \phi_i^t(\widehat{x})\phi_j^t(\widehat{y})\phi_k^t(\widehat{z}),$$

where ϕ_i^t is the tensorial shape function defined for the high-order curve of degree n.[6] The polynomials space spanned by the shape functions is therefore:

$$V_K = \{\widehat{x}^i\widehat{y}^j\widehat{y}^k \mid 0 \leq (i,j,k) \leq n\}.$$

Naturally, the shape functions are Lagrange polynomials corresponding to the nodes $(x_i^n, y_j^n, z_k^n) = (\frac{i}{n}, \frac{j}{n}, \frac{k}{n})$.
A (finite element) hexahedron of degree n, or Q^n-hexahedron, is consequently defined as:

$$M = \omega(\widehat{x}, \widehat{y}, \widehat{z}) = \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=0}^{n} \phi_{ijk}(\widehat{x}, \widehat{y}, \widehat{z})A_{ijk}.$$

where A_{ijk} are the nodes whose dimensions are arbitrary. An example of Q^2-hexahedron is presented in Figure 3.10.

6. In theory, it is possible to define different degrees in each direction, however, apart from the case of structured lattices, this case is not possible in practice.

About the decomposition into tetrahedra or prisms: When dealing with high-order hexahedra, a natural question is their decomposition into high-order tetrahedra or prisms. If we analyze the finite element spaces spanned by the hexahedra, we can see that the monomial basis of the hexahedral finite element space of degree n is always contained in a tetrahedral finite element space if its degree is greater or equal than $3n$ and is always contained in a prismatic finite element space if its degree is greater or equal than $2n$ [Moxey et al. 2015a]. This is due to, among others, the presence of the monomial term $\{\widehat{x}^n\widehat{y}^n\widehat{z}^n\}$ in the hexahedral finite element basis of degree n. Consequently, any valid hexahedron of degree n can be safely divided into two valid prisms if and only if their degree is greater or equal than $2n$ and can be safely divided into six valid tetrahedra if and only if their degree is greater or equal than $3n$.

Transformation into a Bézier hexahedron

As Bernstein polynomials of degree n span the finite element space of the edges of degree n, the product of three Bernstein polynomials of degree n spans the finite element space of the hexahedra of degree n. Therefore, a high-order hexahedron can be written as a Bézier hexahedron. In practice, the transformation is done by solving the following system:

$$\sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=0}^{n} B_i^n(x_i^n)B_j^n(y_j^n)B_k^n(z_k^n)P_{ijk} = A_{ijk},\ \forall(i,j,k)\in[\![0,n]\!],$$

where we recall that P_{ijk} are the control points of the Bézier hexahedron.
In practice, as $B_i^n(0) = 0$ if $i > 0$ and $B_i^n(1) = 0$ if $i < n$, the faces (resp. edges) of the hexahedron can be seen as finite element quadrilaterals (resp. curves), they are transformed into Bézier quadrilaterals (resp. curves), and then using this result, the previous system is solved for all the internal points of the hexahedron (*e.g.* the points that are not on one of the edges/faces of the hexahedron).

Validity of a high-order hexahedron in 3D

For the same reasons as with the tetrahedron, $A_{ijk} \in \mathbb{R}^3$. Now the Jacobian is a polynomial function of degree $3n-1$, and can be written in Bézier form. Using the same kind of derivations as for quads and curves. We obtain that the Jacobian $\mathcal{J}_K$ can be written as:

$$\mathcal{J}_K(\widehat{x},\widehat{y},\widehat{z}) = \sum_{i=0}^{3n-1}\sum_{j=0}^{3n-1}\sum_{k=0}^{3n-1} B_i^{3n-1}(\widehat{x})B_j^{3n-1}(\widehat{y})B_k^{3n-1}(\widehat{z})N_{ijk},$$

where N_{ijk} are defined as:

$$N_{ijk} = n^3 \sum_{\substack{i_1+i_2+i_3=i\\ j_1+j_2+j_3=j\\ k_1+k_2+k_3=k}} \frac{\binom{n-1}{i_1}\binom{n}{i_2}\binom{n}{i_3}}{\binom{3n-1}{i}}\frac{\binom{n}{j_1}\binom{n-1}{j_2}\binom{n}{j_3}}{\binom{3n-1}{j}}\frac{\binom{n}{k_1}\binom{n}{k_2}\binom{n-1}{k_3}}{\binom{3n-1}{k}}\times$$
$$\det(\overrightarrow{P_{i_1j_1k_1}P_{(i_1+1)j_1k_1}}, \overrightarrow{P_{i_2j_2k_2}P_{i_2(j_2+1)k_2}}, \overrightarrow{P_{i_3j_3k_3}P_{i_3j_3(k_3+1)}}),$$

with $(i_1,j_2,k_3)\in[\![0,n-1]\!]$ and $(i_2,i_3,j_1,j_3,k_1,k_2)\in[\![0,n]\!]$. Once again, these control coefficients have a geometrical meaning if we have a look at their expression as determinants. Also, using the same integration result as with the quadrilateral, we can see that:

$$|K| = \frac{1}{(3n)^3}\sum_{i=0}^{3n-1}\sum_{j=0}^{3n-1}\sum_{k=0}^{3n-1} N_{ijk},$$

which means that the volume of a Q^n-hexahedron is the average of all the control coefficients.
The validity of a hexahedron is ensured if its Jacobian is strictly positive everywhere. It can be ensured if all its control coefficients are positive. But for $n > 1$, the opposite is not necessarily true. In Q^1, this is already not that straightforward as the Jacobian is of degree 2: already 27 coefficients containing 64 volumes (this number can be reduced to 20 volumes [Johnen et al. 2017]) of tetrahedra have to be checked to ensure its validity. This becomes more complex for higher degrees. Again, a subdivision procedure can be performed to conclude on the validity of a hexahedron. The subdivision techniques that are used are the ones that are presented in Sections 3.2.2, 3.2.4 and 3.2.7 and the used strategy is the same as used with the tetrahedra: at the beginning, the value at the corners is checked, if the values are positive, the value of the control coefficient lying on the edges are examined. If a negative value is detected, the subdivision algorithm for the curve is applied to conclude on the validity of the Jacobian on the edge. If the validity remains, the faces' inner control coefficients are considered. If one of them is negative, a subdivision of the quadrilateral face is performed to conclude. If the validity remains, the volume's inner control coefficients are considered. If one of them is negative, a subdivision of the hexahedron is performed to conclude. Similarly to other elements, the hexahedra are subdivided as many times as necessary. For numerical reasons, a limit of the number of nested subdivisions is set. If at this limit no conclusion has been possible, the element is declared invalid. This way a hierarchical method to deal with the validity of a Q^n-hexahedron in 3D is set.

3.3.7 High-order pyramid

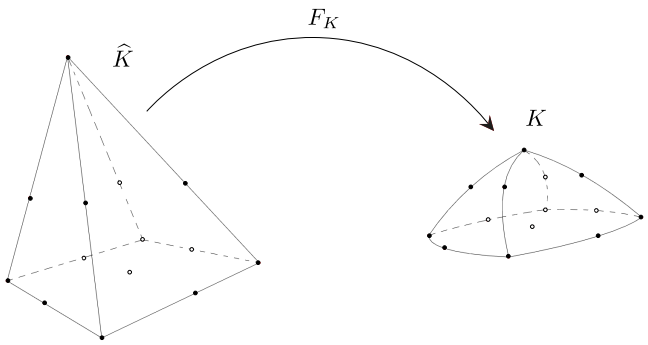

Figure 3.11 – Mapping F_K from $\widehat{K}$ to K for a P^2 pyramid.

Definition

A finite-element pyramid is defined using the following reference element:

$$\widehat{K} = \{0 \leq (\widehat{x}, \widehat{y}, \widehat{z}) \leq 1 \mid 0 \leq \widehat{x} + \widehat{y} \leq 1 - \widehat{z} \leq 1\}.$$

This element is even more special than the prism, it is neither barycentric nor tensorial. However, it has both quadrilateral and triangular faces and this is for this reason that it is called hybrid element. The main issue with the pyramid is that we are not able to define explicit shape functions so that we find a finite element triangle/quad on a triangle/quad face. However, it is possible to define a finite element space for pyramids, but it is a non

polynomial space. The space that will be used is introduced in [Bergot et al. 2010]:

$$V_K = \left\{ \widehat{x}^i \widehat{y}^j \widehat{z}^k \mid i+j+k \leq n \right\} \bigcup \left\{ \widehat{x}^i \widehat{y}^j \left(\frac{\widehat{x}\widehat{y}}{1-\widehat{z}} \right)^{n-l} \mid i+j \leq l \leq n-1 \right\}.$$

This space can be rewritten into a simpler form [Johnen and Geuzaine 2015]:

$$V_K = \left\{ \left(\frac{\widehat{x}}{1-\widehat{z}} \right)^i \left(\frac{\widehat{y}}{1-\widehat{z}} \right)^j (1-\widehat{z})^k \mid i+j \leq k \leq n \right\}.$$

Even if shape functions cannot be properly set, a (finite element) pyramid of order n or P^n-pyramid can be defined using a discrete set of points A_{ijk} (of arbitrary dimension) with $0 \leq i,j \leq n-k \leq n$ which corresponds to the point of coordinates $(x_i^n, y_j^n, z_k^n) = (\frac{i}{n}, \frac{j}{n}, \frac{k}{n})$ in the reference pyramid.

Transformation into a Bézier pyramid

Even if the finite element space is not polynomial, it is possible to define a Bernstein basis to the pyramidal finite element space [Chan and Warburton 2016]. The idea is merely to see a pyramid as a degenerate hexahedron. In all cases, the Bézier pyramid can be deduced solving the following system:

$$\sum_{k=0}^{n} \sum_{i=0}^{n-k} \sum_{j=0}^{n-k} B_i^{n-k} \left(\frac{x_i^n}{1-z_k^n} \right) B_j^{n-k} \left(\frac{y_j^n}{1-z_k^n} \right) B_k^n(z_k^n) P_{ijk} = A_{ijk}, \; 0 \leq (i,j) \leq n-k \leq n,$$

where P_{ijk} are the Bézier pyramid control points.
Now, as we recall that the extremities of a Bézier pyramid are Bézier triangles or quadrilaterals, the faces (resp. edges) of the prism can be seen as finite element triangles/quadrilaterals (resp. curves), they are transformed into Bézier triangles/quadrilaterals (resp. curves), and then using this result, the previous system is solved for all the internal points of the pyramid (*e.g.* the points that are not on one of the edges/faces of the pyramid). An example of P^2-pyramid is presented in Figure 3.11.

Validity of a high-order pyramid in 3D

Now, we will consider that $A_{ijk} \in \mathbb{R}^3$. Once again, the pyramid is ill-defined for the standard validity analysis [Johnen and Geuzaine 2015]. The proposed strategy is to perform a change of variable on its expression as a Bézier pyramid. Indeed, if we set $a = \frac{u}{1-w}, b = \frac{v}{1-w}, c = w$, we have $(a,b,c) \in [0,1]^3$ and we therefore defined the pyramid with an hexahedral reference element. Now, if we apply the transformation explained in Section 3.2.8, the pyramid of degree n is transformed into a degenerate hexahedron of degree n. The next step is therefore to check the validity of the obtained hexahedron with one change: we should never check the validity of the pyramid at the apex (*e.g.* the point shared by the four triangular faces) as it is a degenerated point for the transformation. Using this trick, we are able to figure out the validity of a P^n-pyramid.

3.3.8 About the cost of validity checks for a given mesh

From a more general point of view, the validation of high-order finite-element meshes for complex three-dimensional cases is really expensive. Indeed, Tables 3.1 and 3.2 show the number of determinants that need to be checked to deal with the validity of high-order

tetrahedra and hexahedra, the most common elements in any three-dimensional finite-element mesh. Whereas the cost of a high-order mesh from a solver point of view would be to sample the value of the jacobian at the Gauss (integration) points (*e.g.* the number of nodes) of the geometry, we realize that the effective number of coefficients to be checked for the validity of such meshes is way higher. Indeed, we need to make sure that the mesh is valid everywhere no matter what the order is, considering that the order of the solver is not *a priori* known. In particular this shows that the generation of valid 3D meshes for degrees higher than three is really expensive.

Tetrahedron	P^1	P^2	P^3	P^4	P^5
# nodes	4	10	20	35	56
Degree of the Jacobian	1	3	6	9	12
# control coef.	1	20	84	220	455
# of volume check	1	64	1 000	**8 000**	**42 875**
Ratio/(# of nodes)	-	6.4	50	**228.6**	**765.6**

Table 3.1 – Statistics about the validity check of a high-order tetrahedron.

Hexahedron	Q^1	Q^2	Q^3	Q^4	Q^5
# nodes	8	27	64	125	216
Degree of the Jacobian	2	5	8	11	14
# control coef.	27	216	729	1 728	3 375
# of volume check	64	5 832	110 592	**1 000 000**	**5 832 000**
Ratio/(# of nodes)	8	216	1 728	**8 000**	**27 000**

Table 3.2 – Statistics about the validity check of a high-order hexahedron.

3.4 Quality measures for high-order elements

3.4.1 A review of some quality measures

The definition of a quality measure for the elements of a mesh is crucial as it directly impacts the properties of the mesh and the behavior of the solution computed on it. In general, a quality measure is defined as a function that varies between 1 and the infinity (or if we consider the inverse, between 0 and 1) and that gives its optimal value for 1. Usually, a quality measure is defined for each element of the mesh. The most common sense is to define quality measures that are directly related to the computation that is performed on the mesh. However, it is not that easy as the ideal (or optimal) mesh can be drastically different from one computation to another.

In a simplicial (and linear) mesh generation context, the common idea is to generate a mesh which is ideal for isotropic (or equilateral) elements [George and Borouchaki 1998]. Even with this assertion, it is still difficult to define properly what a good element is. A comprehensive and detailed study of all the existing quality measures has been made in [Dompierre et al. 2005] and shows that it is indeed not that easy to define only one. The most general idea is to use geometrical quantities of the considered element such as

edge length, face area, element volume or the radii of the inscribed and circumscribed circles or spheres to define an algebraic measure that satisfies the properties previously mentioned. In [Knupp 2001], the studied quality measures (that are called metrics in this paper) are defined using the jacobian of the transformation from the ideal element and propose this way an extension of this quality measure to non-simplicial (degree 1) elements. This extension is not straightforward as the jacobian of the transformation is not constant anymore over the element. Therefore, the proposed solution is to compute the jacobian at each of the corners of the element and either to take the worst (*e.g.* smallest) or the average value of these.

Of course, this does not solve the problem of the best mesh for a given physics and a given problem. One option to overcome this problem is to use riemannian metrics tensors in the definition of the quality measure [Brière de L'isle and George 1995]. The proposed metric tensors are generally based on error estimates issued from the computed solution [Loseille and Alauzet 2011b]. Using these tensors, the ideal element may not be an equilateral one if the used metric is not isotropic. Again, a complete survey of these anisotropic quality measures has been done in [Dompierre et al. 2005].

In the last decade, an important effort has been made in the meshing community to extend these definitions to high-order meshes, mainly in the isotropic case. A complete survey of the quality measures for curved meshes can be seen in [Johnen et al. 2018]. More concretely, in several curved mesh generation works [Dey et al. 1999, Sherwin and Peiró 2002, Persson and Peraire 2009, Fortunato and Persson 2016, Turner et al. 2016], the idea is simply to state that the initial P^1-mesh is of high quality and that the only thing that needs to be checked is the jacobian based distortion. The scaled jacobian is introduced to answer to this question. This measure is more or less the ratio between the smallest and the largest value of the jacobian. However, this measure can fail in detecting highly curved and distorted elements as it is possible to generate such elements with a constant jacobian [Toulorge et al. 2013] and the prerequisite of an ideal straight mesh is also a condition that may be too strong, especially when the mesh is coarse. For this reason, it appears mandatory to define quality measures specifically dedicated to high-order elements. As all high-order elements are defined by a mapping from a reference element and can be easily linked to an ideal element via the chain rule, it appears natural to consider the jacobian-based quality measures developed in [Knupp 2001]. This is the approach that was followed by [Roca et al. 2012, Gargallo-Peiró et al. 2015] for volume simplicial mesh curving. An extension to non-simplicial elements and surface elements was proposed in respectively [Gargallo-Peiró 2014] and [Gargallo-Peiró et al. 2013, Ruiz-Gironès et al. 2016a]. The main idea is to compute the so-called *distortion measure* that is a generalization of the one developed in [Knupp 2001]. As the jacobian matrix is not constant over the element, this gives a pointwise distortion measure. By integrating it over the element, it gives then an elementwise measure. Finally, normalizing it by the measure of the ideal element (in our case, the equilateral one) provides the wished elementwise quality measure. The motivation of the integration is to define a smooth function that will then be usable as a cost function for high-order mesh untangling. Another approach to get an elementwise measure starting from a pointwise one can consist in taking the worst pointwise measure over the element as the elementwise one. This is the idea developed in [Johnen et al. 2018] on several quality measures including the previous one. The use of both Bernstein basis and subdivision procedures on Bézier elements allows to get sharp bounds on the quality measure of the worst case. Finally, another quality measure for simplicial elements based mainly on geometrical quantities was set in [George et al. 2019]. Further explanations about this quality function are given in the next section.

Eventually, the question of dealing with anisotropy for curved meshes appears.

This subject is relatively new and has been tackled in [Aparicio-Estrems et al. 2019, Zhang et al. 2019]. The idea is mainly to generalize the work of the previous section with a metric tensor.

In the following sections, we detail the computation of two quality measures that are of special interest for this work. In particular, the next one will be used in the context of high-order mesh optimization while the second one is a generalization of the one used in a anisotropic mesh adaptation and could be used as the starting point for curvilinear mesh adaptation.

3.4.2 An isotropic quality measure for simplicial elements

This quality measure for simplicial elements [7] is proposed in [George et al. 2019]:

$$Q = \underbrace{\alpha \frac{hS_k}{V_k}}_{①} \underbrace{\frac{\max(V_1, V_k)}{\min(V_1, V_k)}}_{②} \underbrace{\left(\frac{N_{max}}{N_{min}}\right)^{1/d}}_{③} \in [1, \infty), \tag{3.15}$$

with:

- d the dimension, S_k the exterior surface of the polyhedron (in 2D, the half perimeter of the polygon) defined by nodes and vertices $(A_i)_{1 \leq i \leq n}$
- V_k the volume (resp. surface) of the high-order simplex defined previously
- h the element's largest edge P^k-length (e.g the length of the union of straight-sided lines defined by the nodes)
- V_1 the volume/surface of the equivalent P^1 element, e.g the element defined by the vertices
- N_{min} (resp. N_{max}) the smallest (resp. largest) control coefficient of the Jacobian of the element
- α is a normalization factor, dependent of the dimension such that $Q = 1$ for a regular simplex, $\alpha = \frac{\sqrt{3}}{6}$ in 2D and $\alpha = \frac{\sqrt{3}}{36}$ in 3D.

This quality function is actually a product of 3 terms. ① is only a generalization of a standard P^1 quality function and measures the gap to the regular element. ② measures the distance between the volume of the curved element and the volume of the straight-sided element and ensures the function to be greater than 1. And, finally ③ gives a measure of the distortion of the element, it also detects if the element is invalid or almost invalid by taking an infinite value. Note that if the element is straight-sided, the standard P^1 quality function [Frey and George 2008] is recovered: $Q = Q_{P^1} = \alpha \frac{hS_1}{V_1} = \alpha \frac{h}{\rho}$ where ρ is the inradius of the straight element. Based on these definitions, this elementwise quality measure is between 1 and the infinity. The closer the element quality to 1, the better the quality. Note this quality measure is only well-defined for planar elements in 2D and volume elements in 3D.

3.4.3 A more general quality measure

In this section, we aim at generalizing a quality function that is widely used in (linear and simplicial) mesh adaptation.

7. By simplicial elements, we mean elements that are defined with a simplicial reference element.

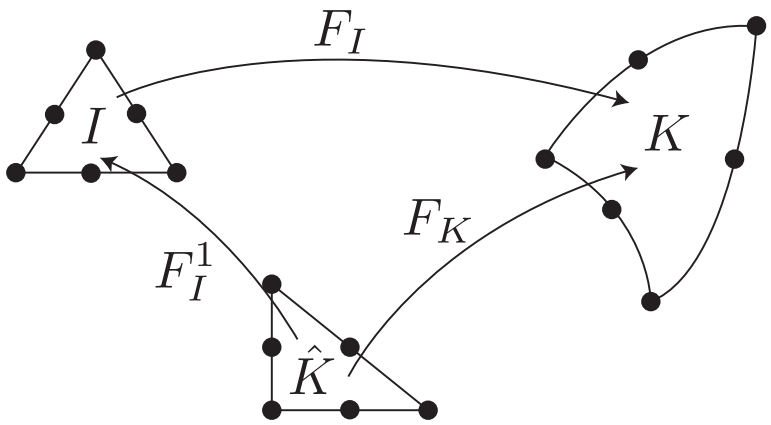

Figure 3.12 – Mappings between three elements among the reference ($\hat{K}$), the ideal (I) and the physical element (K).

The standard quality function

In [Dompierre et al. 2005, Loseille and Alauzet 2011b], the used quality function for simplices consists in a ratio of edges length against a volume and is the following:

$$Q_A = \alpha \frac{\left(\sum_{i=0}^{3(d-1)} \ell_{\mathcal{M}}(e_i)^2\right)^{\frac{d}{2}}}{|K|_{\mathcal{M}}} \in [1, \infty),$$

where K is the studied element, e_i its edges, d the dimension, the subscript A stands for anisotropic and α a normalization factor such that $Q_A = 1$ for the regular element corresponding the metric field $(\mathcal{M}(x))_{x\in\Omega}$. In 2D, $\alpha = \frac{\sqrt{3}}{12}$ and in 3D, $\alpha = \frac{\sqrt{3}}{216}$. $\ell_{\mathcal{M}}$ and $|.|_{\mathcal{M}}$ denote respectively the riemannian edge length and the riemannian area/volume and are defined as:

$$\ell_{\mathcal{M}}(e_i) = \int_0^1 \sqrt{(e_i)^T \mathcal{M}(\gamma(t))\, e_i}\, dt,$$
$$|K|_{\mathcal{M}} = \int_K \sqrt{\det \mathcal{M}(x)} \mathrm{d}\Omega,$$

where $\gamma(t) = (1-t)a + tb$ if we note $e_i = ab$. In the isotropic case (*e.g.* $\mathcal{M} = \beta Id$), this formula degenerates into:

$$Q_I = \alpha \frac{\left(\sum_{i=0}^{3(d-1)} \ell(e_i)^2\right)^{\frac{d}{2}}}{|K|},$$

where the subscript I stands for isotropic, ℓ and $|.|$ are simply the corresponding euclidean versions. For a regular tetrahedron, we have $Q_I = 1$.

The distortion measure

Now, if we consider the jacobian-based shape distortion measure η considered in [Knupp 2001, Roca et al. 2012], we have:

$$\eta(\mathbf{x}_I) = \frac{1}{d} \frac{||\nabla F_I(\mathbf{x}_I)||_F^2}{(\mathcal{J}_I(\mathbf{x}_I))^{\frac{2}{d}}}, \quad \forall \mathbf{x}_I \in I, \tag{3.16}$$

where I is the ideal element, $||\nabla F_I(\mathbf{x}_I)||_F$ is the Frobenius norm [8] of the jacobian matrix of the mapping F_I from the ideal element I to the current element K and $\mathcal{J}_I(\mathbf{x}_I)$ its determinant. The ideal element (see Figure 3.12) is the element of desired shape. In the following, this will be the equilateral triangle or regular tetrahedron. Consequently, the mapping from the reference element $\hat{K}$ to the ideal element I is just an affine transformation and $\nabla F_I = \nabla F_K (\nabla F_I^1)^{-1}$ where ∇F_I^1 is a constant matrix corresponding to the jacobian of the transformation between the reference and the ideal element. Now as we will always assume that the ideal element is regular, we will use $\hat{\mathbf{x}} \in \hat{K}$ as variable for η rather than $\mathbf{x}_I \in I$.

By construction, we always have $\eta(\hat{\mathbf{x}}) \geq 1$. Indeed, if we note $A(\hat{\mathbf{x}}) = (\nabla F_I(\hat{\mathbf{x}}))^T \nabla F_I(\hat{\mathbf{x}})$, we can rewrite η and we have as a consequence of the arithmetic-geometric inequality:

$$\eta(\hat{\mathbf{x}}) = \frac{1}{d} \frac{\mathrm{tr}(A(\hat{\mathbf{x}}))}{(\det(A(\hat{\mathbf{x}})))^{\frac{1}{d}}} \geq 1, \quad \forall \hat{x} \in \hat{K}. \tag{3.17}$$

Now let us have a look on how η is in the case of linear triangles and tetrahedra. Note that in both cases, it will be independent of $\hat{\mathbf{x}}$.

P^1-triangle. In this study, the ideal triangle is the equilateral triangle defined by the vertices $(0,0)$, $(1,0)$ and $(0.5, \frac{\sqrt{3}}{2})$ while the reference element is defined by $(0,0)$, $(1,0)$ and $(0,1)$ [9]. The inverse of the jacobian of F_I can be expressed as:

$$(\nabla F_I^1)^{-1} = \begin{pmatrix} 1 & -\frac{1}{\sqrt{3}} \\ 0 & \frac{2}{\sqrt{3}} \end{pmatrix},$$

and the jacobian of F_K as:

$$\nabla F_K = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix},$$

where $P_{100} = (x_0, y_0)$, $P_{010} = (x_1, y_1)$ and $P_{001} = (x_2, y_2)$ are the nodes of the physical triangle (in Bézier notations). This yields to:

$$\nabla F_I = \begin{pmatrix} x_1 - x_0 & \frac{2x_2 - x_1 - x_0}{\sqrt{3}} \\ y_1 - y_0 & \frac{2y_2 - y_1 - y_0}{\sqrt{3}} \end{pmatrix}.$$

We can deduce that:

$$||\nabla F_I||_F^2 = ||\overrightarrow{P_{100}P_{010}}||^2 + \frac{1}{3}||\overrightarrow{P_{010}P_{001}} + \overrightarrow{P_{100}P_{001}}||^2,$$

where $||.||$ denotes the euclidean norm. Now, if we recall the parallelogram law which states that for any vectors (x, y) of an euclidean vector space:

$$||x + y||^2 = 2(||x||^2 + ||y||^2) - (||x - y||^2 + ||x - z||^2).$$

Hence by application of it to the second term, it comes:

$$||\nabla F_I||_F^2 = \frac{2}{3}(||\overrightarrow{P_{100}P_{010}}||^2 + ||\overrightarrow{P_{010}P_{001}}||^2 + ||\overrightarrow{P_{100}P_{001}}||^2) = \frac{2}{3}\sum_{i=1}^{3} \ell(e_i)^2.$$

8. We recall that the Frobenius norm of a Matrix A is $||A||_F = tr(A^T A) = \sum a_{ij}^2$.

9. Of course another choice of both half-square reference element and equilateral ideal element can be done, but it can be shown that it has no influence on the final distortion measure [Aparicio-Estrems et al. 2019].

The computation of the jacobian is done via the chain rule:

$$\mathcal{J}_I(\hat{\mathbf{x}}) = \mathcal{J}_K(\hat{\mathbf{x}}) \det((\nabla F_I^1)^{-1}) = 2|K| \times \frac{2\sqrt{3}}{3}.$$

The distortion measure for a triangle is therefore exactly the previous 2D isotropic quality function:

$$\eta(\hat{\mathbf{x}}) = \frac{\sqrt{3}}{12} \frac{\sum_{i=1}^{3} \ell(e_i)^2}{|K|} = Q_I^{2D}.$$

This approach can be generalized to surface triangles (*viz.* triangles defined in a 3D space). In this case, the jacobian $\nabla F_I(\hat{\mathbf{x}})$ is however not squared. To overcome this issue, Expression (3.17) of the distortion can be used. The numerator is the same. For the denominator, the obtained matrix $A(\hat{\mathbf{x}})$ is the following:

$$A(\hat{\mathbf{x}}) = \begin{pmatrix} ||\overrightarrow{P_{100}P_{010}}||^2 & \left\langle \overrightarrow{P_{100}P_{010}}, \frac{\overrightarrow{P_{010}P_{001}} + \overrightarrow{P_{100}P_{001}}}{\sqrt{3}} \right\rangle \\ \left\langle \overrightarrow{P_{100}P_{010}}, \frac{\overrightarrow{P_{010}P_{001}} + \overrightarrow{P_{100}P_{001}}}{\sqrt{3}} \right\rangle & \frac{1}{3}||\overrightarrow{P_{010}P_{001}} + \overrightarrow{P_{100}P_{001}}||^2 \end{pmatrix},$$

so its determinant is:

$$\begin{aligned} \det(A) &= \frac{1}{3}||\overrightarrow{P_{010}P_{001}} + \overrightarrow{P_{100}P_{001}}||^2 ||\overrightarrow{P_{100}P_{010}}||^2 - \frac{1}{3}\left(\left\langle \overrightarrow{P_{100}P_{010}}, \overrightarrow{P_{010}P_{001}} + \overrightarrow{P_{100}P_{001}} \right\rangle\right)^2 \\ &= \frac{1}{3}||(\overrightarrow{P_{010}P_{001}} + \overrightarrow{P_{100}P_{001}}) \times \overrightarrow{P_{100}P_{010}}||^2 \\ &= \frac{4}{3}||(\overrightarrow{P_{010}P_{001}}) \times \overrightarrow{P_{100}P_{010}}||^2 \\ &= \frac{4}{3}(4|K|)^2, \end{aligned}$$

where $|K|$ is by definition the non-oriented surface area of K. Finally, we have the denominator equal to:

$$\sqrt{\det(A)} = 2|K| \times \frac{2\sqrt{3}}{3},$$

which is a direct extension of the $2D$ result.

P^1-tetrahedron. In this study, the ideal tetrahedron is the regular tetrahedron defined by the vertices $(0,0,0)$, $(1,0,0)$, $(0.5, \frac{\sqrt{3}}{2}, 0)$ and $(0.5, \frac{1}{2\sqrt{3}}, \sqrt{\frac{2}{3}})$ while the reference element is defined by $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, $(0,0,1)$. From this we deduce:

$$(\nabla F_I^1)^{-1} = \begin{pmatrix} 1 & -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{6}} \\ 0 & \frac{2}{\sqrt{3}} & -\frac{1}{\sqrt{6}} \\ 0 & 0 & \sqrt{\frac{3}{2}} \end{pmatrix} \quad \text{and} \quad \nabla F_K = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 \end{pmatrix},$$

where $P_{1000} = (x_0, y_0, z_0)$, $P_{0100} = (x_1, y_1, z_1)$, $P_{0010} = (x_2, y_2, z_2)$ and $P_{0001} = (x_3, y_3, z_3)$ are the nodes of the physical tetrahedron (in Bézier notations). This yields to:

$$\nabla F_I = \begin{pmatrix} x_1 - x_0 & \frac{2x_2 - x_1 - x_0}{\sqrt{3}} & \frac{3x_3 - x_2 - x_1 - x_0}{\sqrt{6}} \\ y_1 - y_0 & \frac{2y_2 - y_1 - y_0}{\sqrt{3}} & \frac{3y_3 - y_2 - y_1 - y_0}{\sqrt{6}} \\ z_1 - z_0 & \frac{2z_2 - z_1 - z_0}{\sqrt{3}} & \frac{3z_3 - z_2 - z_1 - z_0}{\sqrt{6}} \end{pmatrix}.$$

From this, we can deduce that:

$$||\nabla F_I||_F^2 = ||\overrightarrow{P_{1000}P_{0100}}||^2 + \frac{1}{3}||\overrightarrow{P_{0100}P_{0010}} + \overrightarrow{P_{1000}P_{0010}}||^2$$

$$+\frac{1}{6}||\overrightarrow{P_{0010}P_{0001}}+\overrightarrow{P_{0100}P_{0001}}+\overrightarrow{P_{1000}P_{0001}}||^2,$$

and by using the parallelogram law:

$$||\nabla F_I||_F^2=\frac{2}{3}(||\overrightarrow{P_{1000}P_{0100}}||^2+||\overrightarrow{P_{0100}P_{0010}}||^2+||\overrightarrow{P_{1000}P_{0010}}||^2)$$
$$+\frac{1}{6}||\overrightarrow{P_{0010}P_{0001}}+\overrightarrow{P_{0100}P_{0001}}+\overrightarrow{P_{1000}P_{0001}}||^2.$$

Now, if we use the fact for any vectors (x,y,z) of an euclidean vector space:

$$||x+y+z||^2=3(||x||^2+||y||^2+||z||^2)-(||x-y||^2+||x-z||^2+||y-z||^2),$$

then we obtain:

$$\begin{aligned}||\nabla F_I||_F^2&=\frac{1}{2}(||\overrightarrow{P_{1000}P_{0100}}||^2+||\overrightarrow{P_{0100}P_{0010}}||^2+||\overrightarrow{P_{1000}P_{0010}}||^2\\&\quad+||\overrightarrow{P_{0010}P_{0001}}||^2+||\overrightarrow{P_{0100}P_{0001}}||^2+||\overrightarrow{P_{1000}P_{0001}}||^2)\\&=\frac{1}{2}\sum_{i=1}^{6}\ell(e_i)^2.\end{aligned}$$

Again, the computation of the jacobian is done via the chain rule:

$$\mathcal{J}_I(\hat{\mathbf{x}})=\mathcal{J}_K(\hat{\mathbf{x}})\det((\nabla F_I^1)^{-1})=6|K|\times\sqrt{2}.$$

This leads us to the following expression of the distortion measure:

$$\eta(\hat{\mathbf{x}})=\frac{1}{6(6\sqrt{2})^{\frac{2}{3}}}\frac{\sum_{i=1}^{6}\ell(e_i)^2}{|K|^{\frac{2}{3}}}.$$

and we can see that it is linked to the previous 3D isotropic quality via:

$$\eta(\hat{\mathbf{x}})^{\frac{3}{2}}=\frac{\sqrt{3}}{216}\frac{(\sum_{i=1}^{6}\ell(e_i)^2)^{\frac{3}{2}}}{|K|}=Q_I^{3D}.$$

In both cases, we note that the isotropic quality measure is the distortion measure at a power $\frac{d}{2}$ [10].

Note that it is possible to consider other algebraic measures and to link them to classic geometrical quality measures. Indeed, in [George et al. 2019] it is shown that, for simplices, the algebraic measure $\frac{d^2}{||\nabla F_I(\mathbf{x}_I)||_F^2||\nabla F_I^{-1}(\mathbf{x}_I)||_F^2}$ has as upper bound the quality measure $\alpha\frac{h}{\rho}$ (that is recalled in section 3.4.2). However, when this measure is considered for high-order elements [Johnen 2016], it is said to be very complex and extremely expensive in terms of computational costs [Johnen et al. 2018].

Extension to non-simplicial and high-order elements

As the distortion measure is purely based on algebraic quantities that do not depend on the geometrical element, it appears natural to use it as a starting point for defining a quality function for non-simplicial and high-order elements that will be a generalization of the usual ones.

10. Where we recall that d is the dimension.

However, it appears that with these elements ∇F_I is not constant anymore but is a polynomial function of the (parametric) coordinates on the reference element $\hat{K}$. From this, it follows that both η and Q are pointwise functions. This is an issue as we wish to get an elementwise function. One option is to integrate this function over the reference element. The advantage of this option is that it provides a smooth function that is suitable for optimization procedures. Another approach [Johnen et al. 2018] is to get an upper bound of this pointwise measure and to use it as an elementwise measure. The advantage of this procedure is that it highlights the worst case possible in the element which is something of interest when dealing with mesh optimization and adaptation. The procedure we will use is the latter one. In the most general case, the pointwise quality measure is written as:

$$Q(\hat{\mathbf{x}}) = \alpha \frac{(Q_N(\hat{\mathbf{x}}))^{\frac{d}{2}}}{Q_D(\hat{\mathbf{x}})} \sim \frac{(\text{edges length})^{\frac{d}{2}}}{\text{measure of dimension d}},$$

where $\alpha \in \mathbb{R}$ is a normalization factor and both Q_N and Q_D are polynomial functions [11] with N and D subscripts that stand for numerator and denominator.
The objective of this section is to get an upper bound of $Q(\hat{\mathbf{x}})$ that is to say, the worst case over the element and to set it as quality function $Q(K)$. To this end, let us analyze both Q_N and Q_D.
Q_D is an homogeneous polynomial of degree n_D defining m_D degrees of freedom and Q_N is a sum of a few homogeneous polynomials. If we note m_{pol} the number of different homogeneous polynomials [12] of degree $n_{N,j}$ that are defining $m_{N,j}$ degrees of freedom, we have:

$$Q_N(\hat{\mathbf{x}}) = \sum_{j=1}^{m_{pol}} \sum_{i=0}^{m_{N,j}} B_i^{n_{N,j}}(\hat{\mathbf{x}}) Q_{N,i,j}, \quad \forall \hat{\mathbf{x}} \in \hat{K},$$
$$Q_D(\hat{\mathbf{x}}) = \sum_{i=0}^{m_D} B_i^{n_D}(\hat{x}) Q_{D,i}, \quad \forall \hat{\mathbf{x}} \in \hat{K},$$

where B_i^n is the appropriate Bernstein polynomial for this given element, degree and number of degrees of freedom. Using the properties of the Bernstein polynomials, it comes that:

$$Q_N(\hat{\mathbf{x}}) \leq \sum_{j=1}^{m_{pol}} \max_i Q_{N,i,j}, \quad \forall \hat{\mathbf{x}} \in \hat{K},$$
$$Q_D(\hat{\mathbf{x}}) \geq \min_i Q_{D,i}, \quad \forall \hat{\mathbf{x}} \in \hat{K}.$$

From this we deduce that [13]:

$$\forall \hat{\mathbf{x}} \in \hat{K} \quad Q(\hat{\mathbf{x}}) \leq \alpha \frac{\left(\sum_{j=1}^{m_{pol}} \max_i Q_{N,i,j} \right)^{\frac{d}{2}}}{\min_i Q_{D,i}} = Q(K).$$

11. The case of pyramids will be tackled later.
12. 1 for simplices, 2 for a quadrilateral and prism, 3 for an hexahedron.
13. For the following, we assume that both Q_N and Q_D are positive for $\hat{\mathbf{x}} \in \hat{K}$. Discussion will be made later when it is not the case.

This gives us the wished upper bound on the pointwise quality. It can be noted that this might be a too large upper bound. To overcome this issue, a few iterations of a subdivision algorithm can be performed to get a sharper upper bound to this quality. The detail of such procedures is explained in Section 3.2. For surface elements, the procedure is quite similar, apart from the fact that the polynomial $Q_D(\hat{\mathbf{x}})$ is replaced by $\sqrt{Q_D^{surf}(\hat{\mathbf{x}})}$.
Now, let us explicit both Q_N and Q_D with geometrical quantities for all the elements:

P^n-triangle. The reference and the ideal element are the same as in P^1. Now, the jacobian matrix of the mapping F_K is (using the notations of section 3.3.2):

$$\nabla F_K = \begin{pmatrix} \frac{\partial F_K}{\partial \hat{x}} & \frac{\partial F_K}{\partial \hat{y}} \end{pmatrix} = \begin{pmatrix} \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u} & \frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial u} \end{pmatrix},$$

and from this, it follows that:

$$\nabla F_I = \begin{pmatrix} \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u} & \frac{1}{\sqrt{3}}\left(2\frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u}\right) \end{pmatrix}.$$

It ensues from the P^1-case that:

$$||\nabla F_I||_F^2 = \frac{2}{3}\left(\left\|\frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial v}\right\|^2 + \left\|\frac{\partial F_K}{\partial u} - \frac{\partial F_K}{\partial w}\right\|^2 + \left\|\frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u}\right\|^2\right).$$

If we expand the first term (for instance), we have:

$$\begin{aligned}
&\left\|\frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial v}\right\|^2 = \left\| n \sum_{i+j+k=n-1} B_{ijk}^{n-1}(u,v,w)\overrightarrow{P_{ij(k+1)}P_{i(j+1)k}}\right\|^2 \\
&= \sum_{i+j+k=2(n-1)} B_{ijk}^{2(n-1)}(u,v,w) \\
&\left(\sum_{\substack{i_1+i_2=i \\ j_1+j_2=j \\ k_1+k_2=k}} \frac{n^2 i!j!k!((n-1)!)^2}{(2(n-1))!i_1!i_2!j_1!j_2!k_1!k_2!}\left\langle \overrightarrow{P_{i_1j_1(k_1+1)}P_{i_1(j_1+1)k_1}}, \overrightarrow{P_{i_2j_2(k_2+1)}P_{i_2(j_2+1)k_2}}\right\rangle\right).
\end{aligned}$$

By analogy with the other terms, we can deduce that:

$$||\nabla F_I||_F^2 = \frac{2}{3}\sum_{i+j+k=2(n-1)} B_{ijk}^{2(n-1)}(u,v,w)M_{ijk},$$

where M_{ijk} are the control coefficients defined by:

$$\begin{aligned}
M_{ijk} = \sum_{\substack{i_1+i_2=i \\ j_1+j_2=j \\ k_1+k_2=k}} \frac{n^2 i!j!k!((n-1)!)^2}{(2(n-1))!i_1!i_2!j_1!j_2!k_1!k_2!}\Big(&\left\langle \overrightarrow{P_{i_1j_1(k_1+1)}P_{i_1(j_1+1)k_1}}, \overrightarrow{P_{i_2j_2(k_2+1)}P_{i_2(j_2+1)k_2}}\right\rangle \\
&+ \left\langle \overrightarrow{P_{(i_1+1)j_1k_1}P_{i_1j_1(k_1+1)}}, \overrightarrow{P_{(i_2+1)j_2k_2}P_{i_2j_2(k_2+1)}}\right\rangle \\
&+ \left\langle \overrightarrow{P_{i_1(j_1+1)k_1}P_{(i_1+1)j_1k_1}}, \overrightarrow{P_{i_2(j_2+1)k_2}P_{(i_2+1)j_2k_2}}\right\rangle\Big).
\end{aligned}$$

where we recall that $i_1 + j_1 + k_1 = n - 1$ and $i_2 + j_2 + k_2 = n - 1$. These coefficients can be geometrically interpreted as the scalar products between all the control edges defined for a constant parameter u, v and w.
The computation of the jacobian is done via the chain rule:

$$\mathcal{J}_I(\hat{\mathbf{x}}) = \mathcal{J}_K(\hat{\mathbf{x}})\det((\nabla F_I^1)^{-1}) = \frac{2\sqrt{3}}{3}\sum_{i+j+k=2(n-1)} B_{ijk}^{2(n-1)}(u,v,w)N_{ijk}.$$

where the N_{ijk}s are defined in Section 3.3.2. From this, we can deduce both Q_D and Q_N. If the triangle is a surface triangle, there is still a way to compute a quality function by using the form (3.17) of the distortion measure. The numerator stays the same while the denominator has a slight variation as $\mathcal{J}_I(\hat{\mathbf{x}})$ is replaced by $\sqrt{\det(A(\hat{\mathbf{x}})}$. By analogy with P^1-case and the development of section 3.3.2, we have:

$$\begin{aligned}\det(A(\hat{\mathbf{x}})) &= \frac{4}{3}\left\|\left(\frac{\partial F_K}{\partial v}-\frac{\partial F_K}{\partial u}\right)\times\left(\frac{\partial F_K}{\partial w}-\frac{\partial F_K}{\partial u}\right)\right\|^2 \\ &= \frac{4}{3}\left\|\sum_{i+j+k=2(n-1)} B_{ijk}^{2(n-1)}(u,v,w)\overrightarrow{C_{ijk}}\right\|^2,\end{aligned}$$

where :

$$\overrightarrow{C_{ijk}} = n^2 \sum_{\substack{i_1+i_2=i\\ j_1+j_2=j\\ k_1+k_2=k}} \frac{i!j!k!((n-1)!)^2}{(2(n-1))!i_1!i_2!j_1!j_2!k_1!k_2!}\left(\overrightarrow{P_{(i_1+1)j_1k_1}P_{i_1(j_1+1)k_1}}\times\overrightarrow{P_{(i_2+1)j_2k_2}P_{i_2j_2(k_2+1)}}\right).$$

where we recall that $i_1+j_1+k_1=n-1$ and $i_2+j_2+k_2=n-1$. This finally leads to:

$$\det(A(\hat{\mathbf{x}})) = \frac{4}{3}\left(\sum_{i+j+k=4(n-1)} B_{ijk}^{4(n-1)}(u,v,w)A_{ijk}\right),$$

with:

$$A_{ijk} = \sum_{\substack{i_1+i_2=i\\ j_1+j_2=j\\ k_1+k_2=k}} \frac{i!j!k!(2(n-1)!)^2}{(4(n-1))!i_1!i_2!j_1!j_2!k_1!k_2!}\left\langle\overrightarrow{C_{i_1j_1k_1}},\overrightarrow{C_{i_2j_2k_2}}\right\rangle.$$

where we recall that $i_1+j_1+k_1=2(n-1)$ and $i_2+j_2+k_2=2(n-1)$. Using this polynomial expansion, we are able to get Q_D^{surf} and then to compute, like for planar P^n-triangles, an upper bound for the quality function for surface P^n-triangles.

Q^n-quadrilateral. In the case of the quadrilateral of degree n, the reference and ideal element are identical, that is to say the unit square $[0,1]^2$. In this context, $F_I = F_K$. The jacobian matrix is (using the notations of section 3.3.3):

$$\nabla F_I = \nabla F_K = \begin{pmatrix} \frac{\partial F_K}{\partial \hat{x}} & \frac{\partial F_K}{\partial \hat{y}} \end{pmatrix}.$$

From this, it comes that:

$$||\nabla F_I||_F^2 = \left\|\frac{\partial F_K}{\partial \hat{x}}\right\|^2 + \left\|\frac{\partial F_K}{\partial \hat{y}}\right\|^2.$$

The first term can be expanded:

$$\begin{aligned}\left\|\frac{\partial F_K}{\partial \hat{x}}\right\|^2 &= \left\|n\sum_{i=0}^{n-1}\sum_{j=0}^{n} B_i^{n-1}(\hat{x})B_j^n(\hat{y})\overrightarrow{P_{ij}P_{(i+1)j}}\right\|^2 \\ &= \sum_{i=0}^{2(n-1)}\sum_{j=0}^{2n} B_i^{2(n-1)}(\hat{x})B_j^{2n}(\hat{y})M_{ij}^1.\end{aligned}$$

with:

$$M_{ij}^1 = n^2 \sum_{i_1+i_2=i}\sum_{j_1+j_2=j} \frac{\binom{n-1}{i_1}\binom{n-1}{i_2}}{\binom{2(n-1)}{i}}\frac{\binom{n}{j_1}\binom{n}{j_2}}{\binom{2n}{j}}\left\langle\overrightarrow{P_{i_1j_1}P_{(i_1+1)j_1}},\overrightarrow{P_{i_2j_2}P_{(i_2+1)j_2}}\right\rangle.,$$

where $(i_1, i_2) \in [\![0, n-1]\!]$ and $(j_1, j_2) \in [\![0, n]\!]$. By analogy, we have:

$$\left\| \frac{\partial F_K}{\partial \hat{y}} \right\|^2 = \sum_{i=0}^{2n} \sum_{j=0}^{2(n-1)} B_i^{2n}(\hat{x}) B_j^{2(n-1)}(\hat{y}) M_{ij}^2.$$

with:

$$M_{ij}^2 = n^2 \sum_{i_1+i_2=i} \sum_{j_1+j_2=j} \frac{\binom{n}{i_1}\binom{n}{i_2}}{\binom{2n}{i}} \frac{\binom{n-1}{j_1}\binom{n-1}{j_2}}{\binom{2(n-1)}{j}} \left\langle \overrightarrow{P_{i_1 j_1} P_{i_1 (j_1+1)}}, \overrightarrow{P_{i_2 j_2} P_{i_2 (j_2+1)}} \right\rangle.$$

where $(i_1, i_2) \in [\![0, n]\!]$ and $(j_1, j_2) \in [\![0, n-1]\!]$. This finally yields that:

$$||\nabla F_I||_F^2 = \sum_{i=0}^{2(n-1)} \sum_{j=0}^{2n} B_i^{2(n-1)}(\hat{x}) B_j^{2n}(\hat{y}) M_{ij}^1 + \sum_{i=0}^{2n} \sum_{j=0}^{2(n-1)} B_i^{2n}(\hat{x}) B_j^{2(n-1)}(\hat{y}) M_{ij}^2.$$

In other words, this is polynomials with two different degrees, one for each tensor product term. The same geometrical interpretation as with triangles holds.
The computation of the jacobian is explained in section 3.3.3 and it can be expressed as:

$$\mathcal{J}_I(\hat{\mathbf{x}}) = \mathcal{J}_K(\hat{\mathbf{x}}) = \sum_{i=0}^{2n-1} \sum_{j=0}^{2n-1} B_i^{2n-1}(\widehat{x}) B_j^{2n-1}(\widehat{y}) N_{ij},$$

where N_{ij} is detailed in Section 3.3.3. From this, we are able to define both Q_N and Q_D and then to define an upper bound to the quality function for planar Q^n-quadrilaterals. When the quadrilateral is not planar, the distortion Formula (3.17) is used. Again, the numerator does not change. For the denominator, we have by analogy with the results for the triangle and the results of Section 3.3.3:

$$\det(A(\hat{\mathbf{x}})) = \left\| \sum_{i=0}^{2n-1} \sum_{j=0}^{2n-1} B_i^{2n-1}(\hat{x}) B_j^{2n-1}(\hat{y}) \overrightarrow{C_{ij}} \right\|^2,$$

with:

$$\overrightarrow{C_{ij}} = n^2 \sum_{i_1+i_2=i} \sum_{j_1+j_2=j} \frac{\binom{n-1}{i_1}\binom{n}{i_2}}{\binom{2n-1}{i}} \frac{\binom{n}{j_1}\binom{n-1}{j_2}}{\binom{2n-1}{j}} \left(\overrightarrow{P_{i_1 j_1} P_{(i_1+1) j_1}} \times \overrightarrow{P_{i_2 j_2} P_{i_2 (j_2+1)}} \right).$$

where $(i_1, j_2) \in [\![0, n-1]\!]$ and $(i_2, j_1) \in [\![0, n]\!]$. This finally gives:

$$\det(A(\hat{\mathbf{x}})) = \sum_{i=0}^{2(2n-1)} \sum_{j=0}^{2(2n-1)} B_i^{2(2n-1)}(\hat{x}) B_j^{2(2n-1)}(\hat{y}) A_{ij},$$

with:

$$A_{ij} = \sum_{i_1+i_2=i} \sum_{j_1+j_2=j} \frac{\binom{2n-1}{i_1}\binom{2n-1}{i_2}}{\binom{2(2n-1)}{i}} \frac{\binom{2n-1}{j_1}\binom{2n-1}{j_2}}{\binom{2(2n-1)}{j}} \left\langle \overrightarrow{C_{i_1 j_1}}, \overrightarrow{C_{i_2 j_2}} \right\rangle.$$

where $(i_1, i_2, j_1, j_2) \in [\![0, 2n-1]\!]$. Using this polynomial expansion, we are able to get Q_D^{surf} and then to compute, like for planar Q^n-quadrilaterals, an upper bound for the quality function for surface Q^n-quadrilaterals.

P^n-tetrahedron The reference and the ideal element are the same as in P^1. Now, the jacobian matrix of the mapping F_K is (using the notations of Section 3.3.4):

$$\nabla F_K = \left(\begin{array}{ccc} \frac{\partial F_K}{\partial \hat{x}} & \frac{\partial F_K}{\partial \hat{y}} & \frac{\partial F_K}{\partial \hat{z}} \end{array} \right) = \left(\begin{array}{ccc} \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u} & \frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial u} & \frac{\partial F_K}{\partial t} - \frac{\partial F_K}{\partial u} \end{array} \right),$$

and from this, it follows that:

$$\nabla F_I = \left(\begin{array}{ccc} \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u} & \frac{1}{\sqrt{3}} \left(2\frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u} \right) & \frac{1}{\sqrt{6}} \left(3\frac{\partial F_K}{\partial t} - \frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u} \right) \end{array} \right).$$

It ensues from the P^1-case that:

$$\begin{aligned} ||\nabla F_I||_F^2 = \frac{1}{2} \Bigg(& \left\| \frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial v} \right\|^2 + \left\| \frac{\partial F_K}{\partial u} - \frac{\partial F_K}{\partial w} \right\|^2 + \left\| \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u} \right\|^2 \\ & + \left\| \frac{\partial F_K}{\partial t} - \frac{\partial F_K}{\partial u} \right\|^2 + \left\| \frac{\partial F_K}{\partial t} - \frac{\partial F_K}{\partial v} \right\|^2 + \left\| \frac{\partial F_K}{\partial t} - \frac{\partial F_K}{\partial w} \right\|^2 \Bigg). \end{aligned}$$

Following the triangular case, the expansion of the first term gives:

$$\begin{aligned} & \left\| \frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial v} \right\|^2 = \sum_{i+j+k+l=2(n-1)} B_{ijkl}^{2(n-1)}(u,v,w) \\ & \left(\sum_{\substack{i_1+i_2=i \\ j_1+j_2=j \\ k_1+k_2=k \\ l_1+l_2=l}} \frac{n^2 i!j!k!l!((n-1)!)^2}{(2(n-1))!i_1!i_2!j_1!j_2!k_1!k_2!l_1!l_2!} \left\langle \overrightarrow{P_{i_1 j_1 (k_1+1) l_1} P_{i_1 (j_1+1) k_1 l_1}}, \overrightarrow{P_{i_2 j_2 (k_2+1) l_2} P_{i_2 (j_2+1) k_2 l_2}} \right\rangle \right). \end{aligned}$$

By analogy for the other terms, it comes that:

$$||\nabla F_I||^2 = \sum_{i+j+k+l=2(n-1)} B_{ijkl}^{2(n-1)}(u,v,w) M_{ijkl},$$

with:

$$\begin{aligned} M_{ijkl} = & \sum_{\substack{i_1+i_2=i \\ j_1+j_2=j \\ k_1+k_2=k \\ l_1+l_2=l}} \frac{n^2 i!j!k!l!((n-1)!)^2}{(2(n-1))!i_1!i_2!j_1!j_2!k_1!k_2!l_1!l_2!} \\ & \Bigg(\left\langle \overrightarrow{P_{i_1 j_1 (k_1+1) l_1} P_{i_1 (j_1+1) k_1 l_1}}, \overrightarrow{P_{i_2 j_2 (k_2+1) l_2} P_{i_2 (j_2+1) k_2 l_2}} \right\rangle \\ & + \left\langle \overrightarrow{P_{(i_1+1) j_1 k_1 l_1} P_{i_1 j_1 (k_1+1) l_1}}, \overrightarrow{P_{(i_2+1) j_2 k_2 l_2} P_{i_2 j_2 (k_2+1) l_2}} \right\rangle \\ & + \left\langle \overrightarrow{P_{i_1 (j_1+1) k_1 l_1} P_{(i_1+1) j_1 k_1 l_1}}, \overrightarrow{P_{i_2 (j_2+1) k_2 l_2} P_{(i_2+1) j_2 k_2 l_2}} \right\rangle \\ & + \left\langle \overrightarrow{P_{i_1 j_1 k_1 (l_1+1)} P_{(i_1+1) j_1 k_1 l_1}}, \overrightarrow{P_{i_2 j_2 k_2 (l_2+1)} P_{(i_2+1) j_2 k_2 l_2}} \right\rangle \\ & + \left\langle \overrightarrow{P_{i_1 j_1 k_1 (l_1+1)} P_{i_1 (j_1+1) k_1 l_1}}, \overrightarrow{P_{i_2 j_2 k_2 (l_2+1)} P_{i_2 (j_2+1) k_2 l_2}} \right\rangle \\ & + \left\langle \overrightarrow{P_{i_1 j_1 k_1 (l_1+1)} P_{i_1 j_1 (k_1+1) l_1}}, \overrightarrow{P_{i_2 j_2 k_2 (l_2+1)} P_{i_2 j_2 (k_2+1) l_2}} \right\rangle \Bigg). \end{aligned}$$

where we recall that $i_1 + j_1 + k_1 + l_1 = i_2 + j_2 + k_2 + l_2 = n - 1$.
These coefficients can be geometrically interpreted as the scalar products between all the

control edges defined for two constant parameters.
The computation of the jacobian is done via the chain rule:

$$\mathcal{J}_I(\hat{\mathbf{x}}) = \mathcal{J}_K(\hat{\mathbf{x}}) \det((\nabla F_I^1)^{-1}) = \sqrt{2} \sum_{i+j+k+l=3(n-1)} B_{ijkl}^{3(n-1)}(u,v,w) N_{ijkl}.$$

where N_{ijkl} are defined in Section 3.3.4. From this, we can deduce both Q_N and Q_D.

P^n-prism The ideal prism is the extrusion of the ideal triangle with unit squares as quad faces and the reference element is an extrusion of the triangular reference element. From this, it comes that the inverse of the jacobian of the ideal mapping is:

$$(\nabla F_I^1)^{-1} = \begin{pmatrix} 1 & -\frac{1}{\sqrt{3}} & 0 \\ 0 & \frac{2}{\sqrt{3}} & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

and the jacobian of the mapping is (using the notation of Section 3.3.5):

$$\nabla F_K = \begin{pmatrix} \frac{\partial F_K}{\partial \hat{x}} & \frac{\partial F_K}{\partial \hat{y}} & \frac{\partial F_K}{\partial \hat{z}} \end{pmatrix} = \begin{pmatrix} \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u} & \frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial u} & \frac{\partial F_K}{\partial t} \end{pmatrix}.$$

This gives the expression of the jacobian as:

$$\nabla F_I = \begin{pmatrix} \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u} & \frac{1}{\sqrt{3}}\left(2\frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u}\right) & \frac{\partial F_K}{\partial t} \end{pmatrix},$$

From this, it comes using the result for the triangles:

$$||\nabla F_I||_F^2 = \frac{2}{3}\left(\left\|\frac{\partial F_K}{\partial w} - \frac{\partial F_K}{\partial v}\right\|^2 + \left\|\frac{\partial F_K}{\partial u} - \frac{\partial F_K}{\partial w}\right\|^2 + \left\|\frac{\partial F_K}{\partial v} - \frac{\partial F_K}{\partial u}\right\|^2\right) + \left\|\frac{\partial F_K}{\partial t}\right\|^2.$$

Reusing the results issued from both triangle and quadrilateral sections, it comes that:

$$\begin{aligned} ||\nabla F_I||_F^2 =& \frac{2}{3}\left(\sum_{i+j+k=2(n-1)} \sum_{l=0}^{2n} B_{ijk}^{2(n-1)}(u,v,w) B_l^{2n}(t) M_{ijkl}^1\right) \\ &+ \sum_{i+j+k=2n} \sum_{l=0}^{2(n-1)} B_{ijk}^{2n}(u,v,w) B_l^{2(n-1)}(t) M_{ijkl}^2, \end{aligned}$$

where M_{ijkl}^1 and M_{ijkl}^2 are defined as:

$$\begin{aligned} M_{ijkl}^1 = & \sum_{\substack{i_1+i_2=i \\ j_1+j_2=j \\ k_1+k_2=k \\ l_1+l_2=l}} \frac{n^2 i!j!k!((n-1)!)^2}{(2(n-1))!i_1!i_2!j_1!j_2!k_1!k_2!} \frac{\binom{n}{l_1}\binom{n}{l_2}}{\binom{2n}{l}} \\ & \left(\left\langle \overrightarrow{P_{i_1 j_1 (k_1+1) l_1} P_{i_1 (j_1+1) k_1 l_1}}, \overrightarrow{P_{i_2 j_2 (k_2+1) l_2} P_{i_2 (j_2+1) k_2 l_2}} \right\rangle \right. \\ & + \left\langle \overrightarrow{P_{(i_1+1) j_1 k_1 l_1} P_{i_1 j_1 (k_1+1) l_1}}, \overrightarrow{P_{(i_2+1) j_2 k_2 l_2} P_{i_2 j_2 (k_2+1) l_2}} \right\rangle \\ & \left. + \left\langle \overrightarrow{P_{i_1 (j_1+1) k_1 l_1} P_{(i_1+1) j_1 k_1 l_1}}, \overrightarrow{P_{i_2 (j_2+1) k_2 l_2} P_{(i_2+1) j_2 k_2 l_2}} \right\rangle \right), \end{aligned}$$

where we recall that $i_1+j_1+k_1 = i_2+j_2+k_2 = n-1$, $(l_1,l_2) \in [\![0,n]\!]$ and :

$$M^2_{ijkl} = \sum_{\substack{i_1+i_2=i\\ j_1+j_2=j\\ k_1+k_2=k\\ l_1+l_2=l}} \frac{n^2 i!j!k!(n!)^2}{(2n)!i_1!i_2!j_1!j_2!k_1!k_2!} \frac{\binom{n-1}{l_1}\binom{n-1}{l_2}}{\binom{2(n-1)}{l}} \left(\left\langle \overrightarrow{P_{i_1j_1k_1l_1}P_{i_1j_1k_1(l_1+1)}}, \overrightarrow{P_{i_2j_2k_2l_2}P_{i_2j_2k_2(l_2+1)}} \right\rangle\right).$$

where we recall that $i_1+j_1+k_1 = i_2+j_2+k_2 = n$, $(l_1,l_2) \in [\![0,n-1]\!]$. The computation of the jacobian is done via the chain rule:

$$\mathcal{J}_I(\hat{\mathbf{x}}) = \mathcal{J}_K(\hat{\mathbf{x}})(\mathcal{J}^1_I)^{-1} = \frac{2\sqrt{3}}{3} \sum_{i+j+k=3n-2} \sum_{l=0}^{3n-1} B^{3n-2}_{ijk}(u,v,w) B^{3n-1}_l(t) N_{ijkl},$$

where N_{ijkl} is detailed in Section 3.3.5. From this, we are able to define both Q_N and Q_D and then to define an upper bound to the quality function for P^n-prisms.

Q^n-hexahedron. In the case of the hexahedron of degree n, the reference and ideal element are identical, that is to say the unit cube $[0,1]^3$. The developments are a straightforward extension of the quadrilateral results, they are (using the notations of section 3.3.6):

$$\nabla F_I = \nabla F_K = \begin{pmatrix} \frac{\partial F_K}{\partial \hat{x}} & \frac{\partial F_K}{\partial \hat{y}} & \frac{\partial F_K}{\partial \hat{z}} \end{pmatrix},$$

and

$$\begin{aligned} ||\nabla F_I||^2_F = &\sum_{i=0}^{2(n-1)} \sum_{j=0}^{2n} \sum_{k=0}^{2n} B^{2(n-1)}_i(\hat{x}) B^{2n}_j(\hat{y}) B^{2n}_k(\hat{z}) M^1_{ijk} \\ &+ \sum_{i=0}^{2n} \sum_{j=0}^{2(n-1)} \sum_{k=0}^{2n} B^{2n}_i(\hat{x}) B^{2(n-1)}_j(\hat{y}) B^{2n}_k(\hat{z}) M^2_{ijk} \\ &+ \sum_{i=0}^{2n} \sum_{j=0}^{2n} \sum_{k=0}^{2(n-1)} B^{2n}_i(\hat{x}) B^{2n}_j(\hat{y}) B^{2(n-1)}_k(\hat{z}) M^3_{ijk}. \end{aligned}$$

with:

$$M^1_{ijk} = n^2 \sum_{i_1+i_2=i} \sum_{j_1+j_2=j} \sum_{k_1+k_2=k} \frac{\binom{n-1}{i_1}\binom{n-1}{i_2}}{\binom{2(n-1)}{i}} \frac{\binom{n}{j_1}\binom{n}{j_2}}{\binom{2n}{j}} \frac{\binom{n}{k_1}\binom{n}{k_2}}{\binom{2n}{k}} \left\langle \overrightarrow{P_{i_1j_1k_1}P_{(i_1+1)j_1k_1}}, \overrightarrow{P_{i_2j_2k_2}P_{(i_2+1)j_2k_2}} \right\rangle,$$

$$M^2_{ijk} = n^2 \sum_{i_1+i_2=i} \sum_{j_1+j_2=j} \sum_{k_1+k_2=k} \frac{\binom{n}{i_1}\binom{n}{i_2}}{\binom{2n}{i}} \frac{\binom{n-1}{j_1}\binom{n-1}{j_2}}{\binom{2(n-1)}{j}} \frac{\binom{n}{k_1}\binom{n}{k_2}}{\binom{2n}{k}} \left\langle \overrightarrow{P_{i_1j_1k_1}P_{i_1(j_1+1)k_1}}, \overrightarrow{P_{i_2j_2k_2}P_{i_2(j_2+1)k_2}} \right\rangle,$$

$$M^3_{ijk} = n^2 \sum_{i_1+i_2=i} \sum_{j_1+j_2=j} \sum_{k_1+k_2=k} \frac{\binom{n}{i_1}\binom{n}{i_2}}{\binom{2n}{i}} \frac{\binom{n}{j_1}\binom{n}{j_2}}{\binom{2n}{j}} \frac{\binom{n-1}{k_1}\binom{n-1}{k_2}}{\binom{2(n-1)}{k}} \left\langle \overrightarrow{P_{i_1j_1k_1}P_{i_1j_1(k_1+1)}}, \overrightarrow{P_{i_2j_2k_2}P_{i_2j_2(k_2+1)}} \right\rangle,$$

where for each line:

$$\begin{aligned} (i_1,i_2) \in [\![0,n-1]\!] \quad &\text{and} \quad (j_1,j_2,k_1,k_2) \in [\![0,n]\!], \\ (j_1,j_2) \in [\![0,n-1]\!] \quad &\text{and} \quad (i_1,i_2,k_1,k_2) \in [\![0,n]\!], \\ (k_1,k_2) \in [\![0,n-1]\!] \quad &\text{and} \quad (j_1,j_2,k_1,k_2) \in [\![0,n]\!]. \end{aligned}$$

The computation of the jacobian is explained in section 3.3.6 and it is expressed as follows:

$$\mathcal{J}_I(\hat{\mathbf{x}}) = \mathcal{J}_K(\hat{\mathbf{x}}) = \sum_{i=0}^{3n-1}\sum_{j=0}^{3n-1}\sum_{k=0}^{3n-1} B_i^{3n-1}(\widehat{x})B_j^{3n-1}(\widehat{y})B_j^{3n-1}(\widehat{z})N_{ijk},$$

where N_{ij} is detailed in Section 3.3.6. From this, we are able to define both Q_N and Q_D and then to define an upper bound to the quality function for Q^n-hexahedra.

Not all elements were considered in the previous analysis. Indeed, the (P^n-)pyramid is missing. Several problems arise when considering the quality for this element. The first one is that the used mapping to define the pyramid is not polynomial and consequently the previous analysis does not hold to get an upper bound to the quality. When dealing with the validity of a high-order pyramid, the used trick is to consider the pyramid as a degenerate hexahedron and then to apply the analysis dedicated to hexahedra to conclude on the validity. This approximation is done with the price of not checking the validity at the apex on the pyramid. It appears inevitable that doing this to compute the quality will lead to very bad qualities if not infinite. Another issue, from a more general point of view is the lack of definition for an ideal pyramid. Unlike the other elements, it is impossible to have at the same time, the same angles and the same unit length in a pyramid. This property avoids a proper definition of what an ideal pyramid would be. For these reasons, the quality of the pyramid is not tackled in this work.
Finally, it can be noted that the assumption of positive polynomials on the reference element has been made so that an upper bound to the quality would be obtained. It can be noted that this is not always true as the inverse of the jacobian cannot have an upper bound as it is possible for the jacobian to have both positive and negative values.[14] The solution to overcome the first issue is the one proposed in Section 3.3 that consists in refining the lower bound so that we can obtain something which is positive. If it is not possible, this means that we are in the second case and that the element is invalid. By convention, an invalid element will be set to an infinite quality.

Extension to anisotropy. As in P^1, this function can be extended to the anisotropic case. However, dealing with the anisotropic case is not easy, Indeed, the anisotropy can be interpreted as a deformation of the ideal space. In high-order, the ideal element is used as starting point for defining the distortion and the quality measure. The strategy used in [Aparicio-Estrems et al. 2019] is to take this into account in the framework. In a full continuous framework, this idea makes sense. Now, if we do this, the considered quantities, will not be polynomials anymore which is an issue as our analysis is primarily based on this fact. A solution to deal with this is to interpret differently the anisotropy. If we consider it as a different way to compute volumes and lengths, the natural idea is to replace all the geometrical quantities by their riemannian counterpart. This is already the strategy used to define the anisotropic quality function in P^1 for simplices. The only difference is the fact that riemannian scalar products, cross products and determinants will also be considered. And this way, upper bounds on the quality will be deduced with respectively the max and the min of the riemannian counterparts of the coefficients $Q_{N,i,j}$ and $Q_{D,i}$.

14. Note that if we use the inverse of the distortion measure as quality measure, the problem does not stand as negatives values of the jacobian are accepted with this definition.

3.5 Conclusion

In this chapter, we have presented the framework used to deal with high-order meshes. In particular, the duality between Bézier and high-order (Lagrange) elements has been shown. Several properties regarding these elements have been discussed like the validity or the quality. To deal properly with the validity, subdivision procedure of these elements have been set up. Also, dedicated definitions of the quality of high-order elements have been given and two quality functions have been proposed. The first one that is an isotropic one which is used in Chapter 5 and the second one is thought as a generalization of the ones use in P^1 mesh adaptation. The use of the second one has not been made yet as the curvilinear mesh adaptation is beyond the scope of this thesis. However, the study of this notion is clearly a perspective to the work of this thesis which is why this study has been done. Also, this new quality function is clearly more general than the first one and could be used for the study of Chapter 5 as well.

Chapter 4

Anisotropic error estimate for high-order parametric surface mesh generation

4.1 Introduction

In industrial applications, the definition of the computational domain (or of a design) is provided by a continuous description composed by a collection of patches using a CAD (Computer Aided Design) system. If several continuous representations of a patch exist *via* an implicit equation or a solid model, we focus on the boundary representation (BREP). In this description, the topology and the geometry are defined conjointly. For the topological part, a hierarchical description is used from top level topological objects to lower level objects, we have:

$$\text{model} \rightarrow \text{bodies} \rightarrow \text{faces} \rightarrow \text{loops} \rightarrow \text{edges} \rightarrow \text{nodes}.$$

Each entity of upper level is described by a list of entities of lower level. This is represented in Figure 4.1 for the Onera M6 model, where a face, a loop and corresponding edges are depicted. Note that most of the time, only the topology of a face is provided, the topology between all the faces (patches) needs to be recovered. This piece of information is needed to have a valid watertight surface mesh on output for the whole computational domain. This step makes the surface mesh generation of equal difficulty as volume mesh generation and has been shown to be not trivial [Alleaume 2009].

For node, edge, and face, a geometry representation is also associated to the entity. For node, it is generally the position in space, while for edge and face a parametric representation is used. It consists in defining a mapping from a bounded domain of $\mathbb{R}^2$ onto $\mathbb{R}^3$ such that $(x, y, z) = \sigma(u, v)$ where (u, v) are the parameters (Figure 4.2).

Generally, σ is a NURBS function (Non-uniform rational B-spline) as it is a common tool in geometry modeling and CAD systems [Piegl and Tiller 1997]. From a conceptual point of view, meshing a parametric surface consists in meshing a 2D domain in the parametric space.

The linear case has been widely studied for years [de Cougny and Shephard 1996,

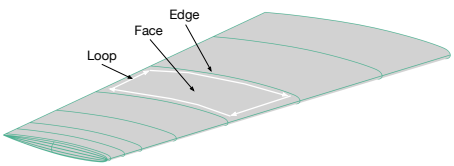

Figure 4.1 – Topology hierarchy (Face, Loop, Edge) of the continuous representation of model using the Boundary REPresentation (BREP).

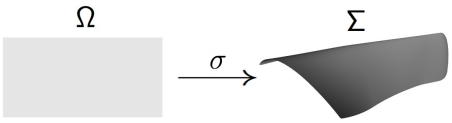

Figure 4.2 – Mapping between the parametric and the physical spaces.

Tristano et al. 1998, Miranda and Martha 2002, Wang et al. 2006, Laug 2010, de Siqueira et al. 2010, de Siqueira et al. 2014, Aubry et al. 2015] and some approaches consist of meshing the 2D domain according to a curvature-based metric [Borouchaki and George 1996]. The use of this metric enables independence to the used parameters space as the curvature is an intrinsic data. The generation of high-order meshes is on the contrary relatively new. The most common idea is to generate a linear mesh and then to project the high-order nodes on the geometry. However, the position of the nodes may not be suitable for a high-order representation of the boundary. For this reason, optimization procedures are applied to the mesh to improve its shape [Toulorge et al. 2016, Ruiz-Gironès et al. 2015, Ruiz-Gironès et al. 2016a, Ruiz-Gironès et al. 2016b, Turner 2018]. The procedure can be done by solving an optimization problem or performing a spring analogy. This can also be performed directly in the parameters space [Gargallo-Peiró et al. 2016] by minimizing a *distortion measure*. In all these cases, it is a *r-adaptation* which is performed. The intent of the present work is to provide tools for high-order parametric surface mesh generation while ensuring independence with respect to any parameterization. The use of the high-order estimates developed in [Coulaud and Loseille 2016] will provide a node distribution which will be specifically tailored for the high-order with a given threshold. Note that high-order surface meshes have a large set of applications. It can be naturally used as an input for the generation of 3D curved meshes. In our case, high-order surface mesh is used advantageously as a surrogate CAD (geometry) model. Indeed, it provides fast forward and inverse evaluation as required in classic linear mesh adaptation.

The first Section briefly recalls the framework used for the high-order error estimates and how a metric can be deduced from it. The second Section shows how the standard parametric surface mesh generation can be extended to high-order using this framework. Finally examples are shown to highlight the process.

4.2 High-order metric based mesh adaptation

In this section a brief presentation of the work done in [Coulaud and Loseille 2016] is performed. It consists in a generalization of the metric-based mesh adaptation presented in Section 1.3. Notably, the idea is to extend to the higher-order the notion of interpolation-based error estimates and then to deduce a multi-scale metric field controlling the error of degree n in L^p norm. From this work, we will only keep the local analysis of the error model.

4.2.1 Error model

Let u be a smooth solution on the domain Ω, $\mathcal{H}$ be a mesh of Ω and n be an arbitrary positive integer. In what follows, $\Pi_n u$ denotes the projection of u onto the finite elements space $\mathbb{P}^n(\mathcal{H})$, whose functions are polynomials of degree n on each element K of $\mathcal{H}$. For all $x_0 \in \Omega$, it is well known that there exists a positive constant C such that, for all $x_0 \in \Omega$

and $x \in \mathbb{R}^d$,

$$|u(x) - \Pi_n u(x)| \leq C \left| d^{(n+1)}u(x_0)(x - x_0) \right| + \mathcal{O}\left(\|x - x_0\|^{n+2} \right), \tag{4.1}$$

where $d^{n+1}u(x_0)$ is the differential form of u of order $n+1$ at x_0, $|.|$ is the absolute value function and $\|.\|$ denotes the Euclidean norm of $\mathbb{R}^d$. Note $d^{n+1}u(x_0)$ is an homogeneous polynomial of order $n+1$ [1]. In the high-order case, the main idea is to replace the right-hand side of Relation (4.1) by a term governed by a metric field $\mathbf{Q} = (\mathcal{Q}(x))_{x\in\Omega}$, which approximates the $n+1$ differential form of u. More precisely, we are looking for $\mathbf{Q}$ such that for all $x_0 \in \Omega, x \in \mathbb{R}^d$

$$\left| d^{n+1}u(x_0)(x - x_0) \right| \leq \left((x - x_0)^T \, \mathcal{Q}(x_0) \, (x - x_0) \right)^{\frac{n+1}{2}}. \tag{4.2}$$

The main issue is to find the metric field $\mathbf{Q}$ such that Inequality (4.2) is as optimal as possible. To solve this problem, let us rewrite it into another form. From a geometrical point of view, the local problem is equivalent to find the largest ellipse in 2D (or the largest ellipsoid in 3D) included into the domain surrounded by the level set of level 1 of $d^{(n+1)}u(x_0)$ (see Figure 4.3).

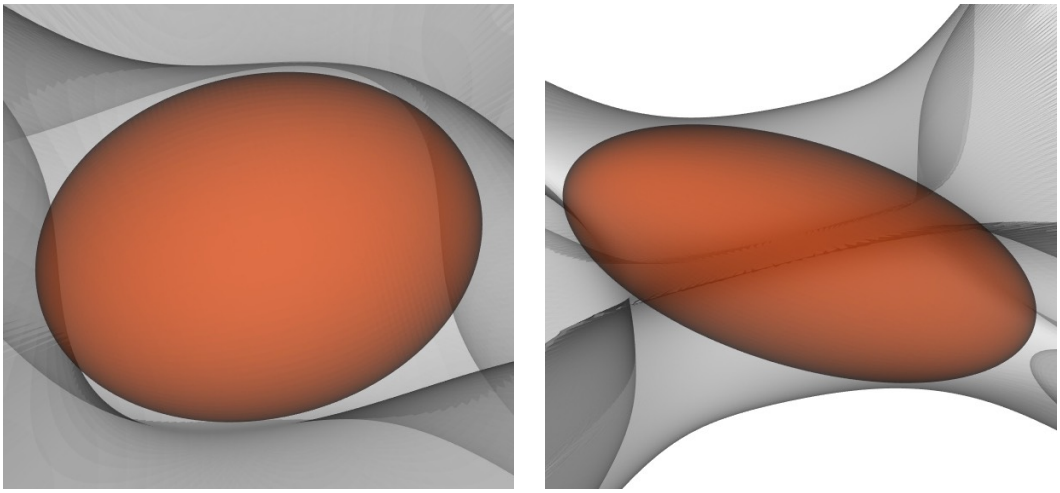

Figure 4.3 – Example of embedded ellipsoids (in red) into polynomial level curves (in grey) verifying the geometric inequality.

Indeed, let us consider the level-set of level 1 of $d^{(n+1)}u(x_0)$ *e.g.* the set of $y \in \mathbb{R}^d$ such that $|d^{(n+1)}u(x_0)(y)| = 1$. Now, let us assume that there exists a symmetric matrix $\mathcal{Q}$ such that for every point y of the level set, inequality $y^T \, \mathcal{Q} \, y \geq 1$ holds. If we note $\mathcal{B}_\mathcal{Q} = \left\{ y \in \mathbb{R}^d : y^T \, \mathcal{Q} \, y \leq 1 \right\}$ the unit ball of metric $\mathcal{Q}$, (this is an ellipse in 2D and an ellipsoid in 3D), then we have a metric $\mathcal{Q}$ whose unit ball is contained in the level set curve. Let us prove that if this result is true for any y, then Inequality (4.2) is verified for any $x \in \mathbb{R}^d$.
Let us consider $x \in \mathbb{R}^d$ and let us set $y = \dfrac{x}{\left| d^{(n+1)}u(x_0)(x) \right|^{\frac{1}{n+1}}}$. Since $d^{(n+1)}u(x_0)$ is a homogeneous polynomial of degree $n+1$:

$$|d^{(n+1)}u(x_0)(y)| = 1.$$

1. A polynomial p is said to be homogeneous of order n, if for any $\lambda \in \mathbb{R}$, $p(\lambda x) = \lambda^n p(x)$.

Now, if we assume that $1 \leq y^T \mathcal{Q}\, y$, then it comes:

$$1 \leq \frac{x^T \mathcal{Q}\, x}{\left|d^{(n+1)}u(x_0)(x)\right|^{\frac{2}{n+1}}},$$

and consequently

$$\left|d^{(n+1)}u(x_0)(x)\right| \leq \left(x^T \mathcal{Q}\, x\right)^{\frac{n+1}{2}}, \quad \text{for all } x \in \mathbb{R}^d,$$

which achieves the proof. The purpose of the next section is to solve this minimization problem.

4.2.2 Log-simplex method

The high-order mesh adaptation method which we use in this chapter is the log-simplex method. It can be seen as an extension of the method described in [Loseille and Alauzet 2011a, Loseille and Alauzet 2011b] to the high-order interpolants. The main difference between the P^1 and the P^n adaptation method relies on the fact that $\mathbf{Q}$ is directly given by the Hessian matrix of u when dealing with P^1 adaptation, whereas it is mandatory to find a suitable metric field satisfying Inequality (4.2) for the P^n adaptation. The log-simplex algorithm is a way to compute such a metric field. It is based on a sequence of linear problems written in terms of the logarithm matrix $\mathcal{L} = \log(\mathcal{Q})$. In this section, the highlights of this method are recalled.
Given a homogeneous polynomial p of degree $n+1$ on $\mathbb{R}^d$ which stands for $d^{(n+1)}u(x_0)$, a set of points $\{x_1, ..., x_m\}$ of $\mathbb{R}^d$ such that $p(x_i) = 1$ for all $i \in \{1, ..., m\}$ is considered. The optimization problem that the log-simplex method solves is the following.

Find a metric $\mathcal{Q}$ such that

$$\begin{aligned} &\det(\mathcal{Q}) \quad \text{is minimal,} \\ &x_i^T \mathcal{Q}\, x_i \geq 1, \quad \text{for all} \quad i \in \{1, ..., m\}. \end{aligned} \tag{4.3}$$

The first line of Problem (4.3) translates the fact that we are looking for the metric with the largest area (or volume in 3D). Since the cost function of this problem is nonlinear, one rewrites it as a problem in $\mathcal{L} = \log(\mathcal{Q})$. Notice that $\mathcal{L}$ is not a metric but only a symmetric matrix. This formulation also enables the discrete counterpart of the problem to be well-posed. Indeed, in [Coulaud and Loseille 2016], it is shown that the discrete form of Equation (4.3) is ill-posed. For $\det(\mathcal{Q}) = \exp(\text{trace}(\mathcal{L}))$, a linear cost function is recovered by replacing $\mathcal{Q}$ by $\mathcal{L}$ in Equation (4.3). On the contrary, the constraints which are linear on $\mathcal{Q}$ become nonlinear when writing them in terms of $\mathcal{L}$. This can lead to really expensive computations. To avoid this problem, the convexity property of the exponential is used and replaces these constraints by approximated linear ones. More precisely, through the classic convexity inequality, if $x \in \mathbb{R}^d$ satisfies $x^T \mathcal{L}\, x \geq -\|x\|^2 \log\left(\|x\|^2\right)$, it ensures that $x^T \mathcal{Q}\, x \geq 1$. By this way, the following linear optimization problem is obtained.

Find a symmetric matrix $\mathcal{L}$ such that

$$\begin{aligned} &\text{trace}(\mathcal{L}) \quad \text{is minimal,} \\ &x_i^T \mathcal{L}\, x_i \geq -\|x_i\|^2 \log\left(\|x_i\|^2\right), \quad \forall i \in \{1, ..., m\}. \end{aligned} \tag{4.4}$$

Since this problem is linear in $\mathcal{L}$, it can be solved by a simplex method (see for instance [Dantzig and Thapa 2003]). Unfortunately, in most of the cases, solving Problem (4.4) once does not provide accurate metrics, in the sense that the unit ball of the obtain metric $\mathcal{Q} = \exp(\mathcal{L})$ can be far from the level set of p (see Figure 4.4).

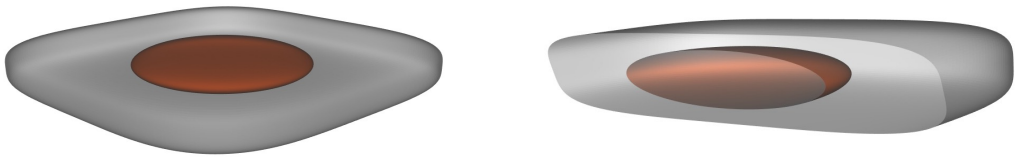

Figure 4.4 – Illustration of the log approximation for the constraints for an error level 1 (in grey). The optimal metric (in red) is far from the boundary of the error due to the convexity approximation.

This issue is dealt by an iterative process. More precisely, once we have computed the solution of Problem (4.4) and recovered $\mathcal{Q} = \exp(\mathcal{L})$, we apply the mapping $x \to \mathcal{Q}^{\frac{1}{2}}x$ by replacing p by $q = p \circ \mathcal{Q}^{-\frac{1}{2}}$. Then, we take a new set of points $\{x_1, ..., x_m\}$ such that $q(x_i) = 1$, for all $i \in \{1, ..., m\}$ and solve again Problem (4.4). The next theorem ensures that if this iterative process converges to a metric $\mathcal{Q}$, then this metric approximates well the level set of level 1 of p.

Theorem 4.2.1. *Let p be a homogeneous polynomial and $\{z_1, \dots, z_m\}$ be a set of points of $\mathbb{R}^d$ such that $||z_i|| = 1$ for all $i = 1, \dots, m$. Let $(\mathcal{Q}_j)_{j\in\mathbb{N}}$ be the sequence of metrics of $\mathbb{R}^d$ defined by:*

- $\mathcal{Q}_0 = I_d$
- $\mathcal{Q}_j = \mathcal{Q}_{j-1}^{\frac{1}{2}} exp(\mathcal{L}_j)\mathcal{Q}_{j-1}^{\frac{1}{2}}$ *where $\mathcal{L}_j$ is an optimal solution of the log-simplex (4.4) with the constraint points $\left\{y_1^j, \dots, y_m^j\right\}$ defined by:*

$$y_i = \frac{z_i}{\left|p\left(\mathcal{Q}^{-\frac{1}{2}}z_i\right)\right|^{\frac{1}{n}}}, \text{ for all } i = 1, \dots, m.$$

If the sequence $(\mathcal{Q}_j)_{j\in\mathbb{N}}$ converges to a metric $\mathcal{Q}$ then the sequence $(y_i^j)_{i\in\mathbb{N}}$ converges to $y_i \in \mathbb{R}^d$, $(\mathcal{L}_j)_{j\in\mathbb{N}}$ converges to $\mathcal{L} = 0$. Furthermore, we have:

$$(y_i)^T \mathcal{L} y_i \geq -||y_i||^2 \log(||y_i||^2) \equiv (x_i)^T \mathcal{Q} x_i \geq 1,$$

where, for all $i \in \{1, \dots, m\}$, $x_i = \mathcal{Q}^{-\frac{1}{2}}y_i$ is a point of the level-set of level 1 of p.

The proof of Theorem 4.2.1 is done in [Coulaud and Loseille 2016] and the final implementation of the log-simplex method is explained in **Algorithm** 21.
In order to implement numerically the log-simplex method, notice that this algorithm must contain a polynomial reduction so that the possible infinite branches in the level set of $d^{(n+1)}u$ disappear.
The objective of the following section is to find the function u on which the high-order error model should be applied and then to deduce a metric on which mesh adaptation will be performed in the parametric space.

Algorithm 21: Log-simplex method

input : A mesh $\mathcal{H}$ of Ω
$d^{(n+1)}u(x)$, for all $x \in \mathcal{H}$

output: $\mathbf{Q} = (\mathcal{Q}(x))_{x\in\mathcal{H}}$

foreach $x \in \mathcal{H}$ **do**
  repeat
    choose a set of points $\{x_1, ..., x_m\}$ on the level set of p of level 1
    perform the log-simplex algorithm and obtain a metric $\mathcal{Q}$
    replace p by $p \circ \mathcal{Q}^{-\frac{1}{2}}$
  until *convergence*;

4.3 High-order surface mesh generation

4.3.1 Metrics for linear surface mesh generation

In the case of parametric surface meshing, the whole problem is to find a suitable metric to drive the mesh adaptation process in the parametric space. Note that it is also mandatory for the process to be intrinsic. First let us have a look at the case of meshing a curve.

Metrics for curves

When dealing with meshing of parametric curves, it is frequent to perform a local analysis on it. To do so, let us have a look at a Taylor expansion of a parametric curve $t \to \gamma(t) \in \mathbb{R}^3$, that we will assume smooth enough, in the vicinity of t_0 [2]:

$$\gamma(t) = \gamma(t_0) + \gamma'(t_0)(t - t_0) + \frac{\gamma''(t_0)}{2}(t - t_0)^2 + \mathcal{O}((t - t_0)^3).$$

Now if we consider a change of variable with s being the curvilinear abscissa such that $s(t_0) = 0$ and $\frac{ds}{dt} = ||\gamma'(t)||$ then the Taylor expansion becomes:

$$\gamma(s) = \gamma(0) + s\mathbf{T} + \frac{1}{2}\kappa(t_0)s^2\mathbf{N} + \mathcal{O}(s^3),$$

where $\mathbf{T} = \frac{\gamma'(t)}{||\gamma'(t)||}$ and $(\kappa, \mathbf{N})$ are such that $\frac{d\mathbf{T}}{ds} = \kappa\mathbf{N}$. $(\kappa, \mathbf{T}, \mathbf{N})$ are intrinsic data [Do Carmo 2016]. κ is called the *curvature* and can be computed with:

$$\kappa(t) = \frac{||\gamma'(t) \times \gamma''(t)||}{||\gamma'(t)||^3}.$$

By setting $\mathbf{B} = \mathbf{T} \times \mathbf{N}$, $(\mathbf{T}, \mathbf{N}, \mathbf{B})$ defines an orthornormal basis, of so-called Frénet frame. Note that if we denote (x, y, z) the components of $\gamma(t)$ in the Frénet frame defined in t_0 with $\gamma(t_0) = (x_0, y_0, z_0)$ in this frame, we have:

$$\begin{cases} y = y_0 + \frac{1}{2}\kappa(t_0)(x - x_0)^2 + \mathcal{O}(|x - x_0|^3), \\ z = z_0 + \mathcal{O}(|x - x_0|^3). \end{cases} \tag{4.5}$$

2. In the case of NURBS, it is easy to split the NURBS according their smooth parts [Piegl and Tiller 1997].

Based on the Frénet frame, and on the curvature, a 3D metric tensor can be deduced via:

$$\mathcal{M}_1 = \left(\mathbf{T}^T\mathbf{N}^T\mathbf{B}^T\right)\begin{pmatrix} \frac{1}{(2\sqrt{\epsilon(2-\epsilon)}\rho(t))^2} & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{pmatrix}\begin{pmatrix} \mathbf{T} \\ \mathbf{N} \\ \mathbf{B} \end{pmatrix},$$

where $\lambda \in \mathbb{R}$ is an arbitrary constant[3], $\rho(t) = \frac{1}{\kappa(t)}$ is the radius of curvature, and $2\sqrt{\epsilon(2-\epsilon)}$ is a scaling coefficient which guarantees, for a second order approximation of the curve, to maintain a deviation gap between the mesh elements and the curve geometry of ϵ [Frey 2000]. As the metric relies on only intrinsic data, it is independent of the parameterization.
The metric is mapped back to the parametric space via the following formula:

$$\widetilde{\mathcal{M}}_1 = \gamma'(t)^T\mathcal{M}_1\gamma'(t).$$

In this case, the formula simplifies to $\widetilde{\mathcal{M}}_1 = \frac{1}{h_1^2} = \frac{||\gamma'(t)||^2}{(2\sqrt{\epsilon(2-\epsilon)}\rho(t))^2}$.
Once the metrics for curves are set, the next step is to define metrics for the surfaces.

Metrics for surfaces

The meshing process of parametric surfaces is a bit more complex. It relies on some differential geometry notions [Do Carmo 2016]. For this purpose, let us consider a parametric surface $(u,v) \to \sigma(u,v) \in \mathbb{R}^3$ that we will assume smooth enough. In this case, the *first fundamental form* $I(du,dv)$ is defined as follows:

$$I(du,dv) = (du\ dv)\begin{pmatrix} ||\sigma_u||^2 & (\sigma_u,\sigma_v) \\ (\sigma_u,\sigma_v) & ||\sigma_v||^2 \end{pmatrix}\begin{pmatrix} du \\ dv \end{pmatrix},$$

where (du,dv) is an elementary displacement, and σ_u (resp. σ_v) the partial derivative of σ w.r.t. u (resp. v). The first fundamental form explains how the three-dimensional distances are perceived in the two-dimensional space. In particular, it provides a two-dimensional riemannian structure to the surface with a metric tensor defined as:

$$\mathcal{M}_I = \begin{pmatrix} ||\sigma_u||^2 & (\sigma_u,\sigma_v) \\ (\sigma_u,\sigma_v) & ||\sigma_v||^2 \end{pmatrix}.$$

In the same framework, we define the *second fundamental form* $II(du,dv)$ by:

$$II(du,dv) = (du\ dv)\begin{pmatrix} (\sigma_{uu},\mathbf{N}) & (\sigma_{uv},\mathbf{N}) \\ (\sigma_{uv},\mathbf{N}) & (\sigma_{vv},\mathbf{N}) \end{pmatrix}\begin{pmatrix} du \\ dv \end{pmatrix},$$

where $(\sigma_{uu},\sigma_{uv},\sigma_{vv})$ are the second derivatives of σ w.r.t. (u,v) and $\mathbf{N} = \frac{\sigma_u\times\sigma_v}{||\sigma_u\times\sigma_v||}$ is the normal vector to the surface. The second fundamental form expresses the gap of a surface to its tangent plane at the second order.
Based on these two quadratic forms and their matrices, we are able, for a given point of the surface, to define the principal curvatures $(\kappa_i)_{i=1,2}$ and principal directions $(\mathbf{V}_i)_{i=1,2}$ (in 3D) as solution of the generalized eigenvalue problem:

$$\begin{cases} \mathcal{M}_{II}\mathbf{v}_i = \kappa_i\mathcal{M}_I\mathbf{v}_i, \\ \mathbf{V}_i = \dfrac{\sigma_u v_{i,1} + \sigma_v v_{i,2}}{||\sigma_u v_{i,1} + \sigma_v v_{i,2}||}, \end{cases} i = 1,2\ ,$$

3. It sets the size in the normal plane to the curve

with $\mathcal{M}_{II}$ is the symmetric matrix associated to the second fundamental form. These quantities are independent of the parameterization and when $\kappa_1 \neq \kappa_2$, $(\mathbf{V}_1, \mathbf{V}_2)$ forms an orthonormal basis of the tangent plane. If we complete the basis with $\mathbf{N}$, they form a local basis $(\mathbf{V}_1, \mathbf{V}_2, \mathbf{N})$ of $\mathbb{R}^3$. Note that if we denote (x, y, z) the components of $\sigma(u, v)$ in this local basis defined in (u_0, v_0) with $\sigma(u_0, v_0) = (x_0, y_0, z_0)$ in this basis, we have:

$$z = z_0 + \frac{1}{2}(\kappa_1(u_0, v_0)(x - x_0)^2 + \kappa_2(u_0, v_0)(y - y_0)^2) + \mathcal{O}(\|(x - x_0, y - y_0)\|^3). \quad (4.6)$$

Now, using this basis and both curvatures, we define the following 3D metric tensor:

$$\mathcal{M}_2 = \left(\mathbf{V}_1^T \mathbf{V}_2^T \mathbf{N}^T\right) \begin{pmatrix} \frac{1}{(c_1\rho_1(u,v))^2} & 0 & 0 \\ 0 & \frac{1}{(c_2\rho_2(u,v))^2} & 0 \\ 0 & 0 & \lambda \end{pmatrix} \begin{pmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{N} \end{pmatrix},$$

where $\lambda \in \mathbb{R}$ is an arbitrary constant, $\rho_i(u, v) = \frac{1}{\kappa_i(u,v)}$ for $i = 1, 2$ are the radii of curvature, with the convention $\rho_1(u, v) \leq \rho_2(u, v)$ and $c_1$and c_2 are scaling coefficients. For the direction of greater curvature (*e.g.* the direction given by $\mathbf{V}_1$), we want to control the deviation under a threshold ϵ which comes down to set c_1 to the value of $2\sqrt{\epsilon(2-\epsilon)}$. Now, as we want the same threshold in all the directions in the tangent plane, the coefficient c_2 is set to $2\sqrt{\epsilon\frac{\rho_1}{\rho_2}(2 - \epsilon\frac{\rho_1}{\rho_2})}$ [George et al. 2019]. Indeed, in Figure 4.5, we see that for a given ρ the choice of the length $2\sqrt{\epsilon(2-\epsilon)}\rho$ ensures a deviation below $\epsilon\rho$. Consequently, if we want the process to be driven by the greater curvature κ_1 with a constant threshold, whatever the direction in the tangent plane, the value ϵ has to be set to $\epsilon\frac{\rho_1}{\rho_2}$ in the other direction of curvature. By doing so, a constant gap of $\epsilon\rho_1$ is ensured for every possible direction.

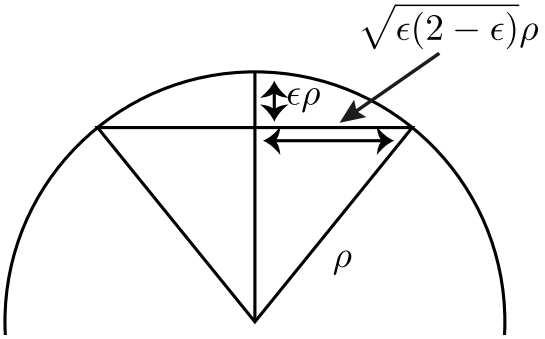

Figure 4.5 – Derivation of the length for the control of the deviation of a curve with respect to its tangent plane.

Similarly to curves, the metric relies only on intrinsic data and is therefore independent of the parameterization.
Now, the metric $\mathcal{M}_2$ is mapped back to the parametric space by applying the first fundamental form:

$$\widetilde{\mathcal{M}_2} = \left(\sigma_u^T \ \sigma_v^T\right) \mathcal{M}_2 \begin{pmatrix} \sigma_u \\ \sigma_v \end{pmatrix}.$$

This is the metric that will be used as an anisotropic metric for the mesh adaptation in the parametric space.

4.3.2 Computation of higher order metrics

The scope of this section is to extend the previous framework to higher order elements. In particular, we seek for a parameterization-independent Taylor expansion similar to (4.5) and to (4.6) with terms of degree greater than 2. As previously, we consider the case of curves first.

Case of the curves

As mentioned previously, the metric should rely on intrinsic data to guarantee independence to the parameterization. A way to do so is to have a look at Formula (4.5). This formula gives a Taylor expansion of the gap of a curve to the straight edge at the order two and shows that it is driven by the curvature. Moreover, this expansion is done within the physical coordinates which naturally gives independence with respect to any parameterization. A natural idea is therefore to extend the previous Taylor expansion to get higher order terms and to deduce metrics that will control high-order terms.
To do so, let us write $\gamma(t)$ in the Frénet frame $(\mathbf{T}_0, \mathbf{N}_0, \mathbf{B}_0)$ associated to t_0, a regular point of γ. If we note $X = x - x_0, Y = y - y_0, Z = z - z_0$, we have:

$$\begin{cases} X = (\gamma(t) - \gamma(t_0), \mathbf{T}_0), \\ Y = (\gamma(t) - \gamma(t_0), \mathbf{N}_0), \\ Z = (\gamma(t) - \gamma(t_0), \mathbf{B}_0). \end{cases}$$

Let us note $\phi(t) = (\gamma(t) - \gamma(t_0), \mathbf{T}_0)$. If t_0 is a regular point of γ then $\gamma'(t_0) \neq 0$ and therefore $\phi'(t_0) \neq 0$. Using the inversion function theorem, there exists a function ψ such that $\psi(X) = \psi(\phi(t)) = t - t_0$ in the vicinity of t_0. Moreover, if ϕ is C^{n+1} then ψ is C^{n+1} and $\psi'(X) = \frac{1}{\phi'(t)}$ with $X = \phi(t)$.
It is therefore possible to get a Taylor expansion of $t - t_0$ with respect to X up to order n. To do so, let us compute the higher order derivatives of ψ at t_0. As γ (and consequently ϕ) is an analytical function issued from (CAD) model, all its derivatives can be computed using the implementation details of [Piegl and Tiller 1997]. The derivatives of ψ are then deduced using the following result [Faà di Bruno 1857]:

Theorem 4.3.1 (Faà di Bruno's Formula)**.** *Let us consider $f, g : \mathbb{R} \to \mathbb{R}$ of class C^{n+1} with $n + 1 \geq k$, then $\forall t \in \mathbb{R}$, we have:*

$$\frac{\mathrm{d}^k}{\mathrm{d}t^k}(g(f(t))) = \sum_E \frac{k!}{m_1!..m_k!} g^{(m_1+\ldots+m_k)}(f(t)) \prod_{j=1}^{k} \left(\frac{f^{(j)}(t)}{j!} \right)^{m_j},$$

where $E = \{(m_1, .., m_k) \in \mathbb{N}^k | \sum_{j=1}^k j.m_j = k\}$ [4].

In our case, we set $g = \psi$ and $f = \phi$ and it comes:

$$\frac{\mathrm{d}^k}{\mathrm{d}t^k}(\psi(\phi(t))) = \frac{\mathrm{d}^k}{\mathrm{d}t^k}(t - t_0) = 0 \text{ for } n + 1 \geq k \geq 2.$$

Now, if we apply the Faà di Bruno's Formula to $\psi \circ \phi$, the only term of E that makes appear $\psi^{(k)}(X)$ is $(k, 0, \ldots, 0)$. This brings us to that for $n + 1 \geq k \geq 2$:

$$(\phi'(t))^k \psi^{(k)}(X) = F(\psi'(X), .., \psi^{(k-1)}(X), \phi'(t), .., \phi^{(k)}(t)),$$

4. Note that in practice, the set E can be precomputed once for all for the range of values of k that are used.

where F is a function that can be directly deduced from Theorem 4.3.1.
This result means that as long as $\phi'(t) \neq 0$, then $\psi^{(k)}(X)$ can be recursively computed given its previous derivatives and the derivatives of ϕ.
Thanks to this, we can now write:

$$t = t_0 + \sum_{k=1}^{n+1} a_k X^k + \mathcal{O}(|X|^{n+2}),$$

where the a_k have been computed with the derivatives of ψ. Now if we recall that γ is C^{n+1}, we also have:

$$\begin{cases} Y = \sum_{k=1}^{n+1} b_k (t - t_0)^n + \mathcal{O}(|t - t_0|^{n+2}), \\ Z = \sum_{k=1}^{n+1} c_k (t - t_0)^n + \mathcal{O}(|t - t_0|^{n+2}). \end{cases}$$

By composition of both Taylor expansion, we then obtain a Taylor expansion of Y and Z in X at the order $n + 1 \geq 2$, which is independent of the parameterization. This is a generalization of Formula (4.5).
In other words, we have an intrinsic information of the gap of a curve to the straight edge up to the order $n + 1$.
A compact form of this estimates writes:

$$\begin{pmatrix} Y \\ Z \end{pmatrix} = F_n(X) + A_{n+1} X^{n+1} + \mathcal{O}(|X|^{n+2}),$$

where $F_n(X)$ is a polynomial of degree n in X and $A_{n+1} \in \mathbb{R}^2$. For an approximation of a curve at the degree n, the leading term of the error is therefore $A_{n+1}X^{n+1}$[Loseille and Alauzet 2011a, Loseille and Alauzet 2011b, Coulaud and Loseille 2016]. Thus, if we control $||A_{n+1}X^{n+1}||$, the error of approximation will be controlled. Now, if we note that:

$$||A_{n+1}X^{n+1}|| = (||A_{n+1}||^{\frac{2}{n+1}} X^2)^{\frac{n+1}{2}},$$

we can then set $\kappa_{n+1} = 2||A_{n+1}||^{\frac{2}{n+1}}$ and reuse the metrics used for the linear case with $\rho = \frac{1}{\kappa_{n+1}}$ for radii of curvature and $\epsilon_{n+1} = \epsilon^{\frac{2}{n+1}}$ for threshold. This way, the classic formula is found for $n = 1$ and a generalization is proposed for $n \geq 2$. Note that in some configurations, the found size for the order n can be significantly lower than the found size for the order $n + 1$. In this case the size given by order $n + 1$ is preferred.
Now, we consider to the case of surfaces.

Case of the surface

Like for curves, the idea is to start from an intrinsic representation of the surface. For this purpose, let us have a look at Formula (4.6). This formula gives a Taylor expansion of the gap of a surface to its tangent plane in physical coordinates which is an intrinsic representation. As previously, our idea is to extend this Taylor expansion to higher-order terms.
To do so, let us note $(\mathbf{V}_{1,0}, \mathbf{V}_{2,0}, \mathbf{N}_0)$ the local basis defined for a regular point (u_0, v_0) of the surface. If we note $X = x - x_0, Y = y - y_0, Z = z - z_0$, we have:

$$\begin{cases} X = (\sigma(u, v) - \sigma(u_0, v_0), \mathbf{V}_{1,0}), \\ Y = (\sigma(u, v) - \sigma(u_0, v_0), \mathbf{V}_{2,0}), \\ Z = (\sigma(u, v) - \sigma(u_0, v_0), \mathbf{N}_0). \end{cases}$$

Now, if we define:

$$\Phi(u,v) = \begin{pmatrix} (\sigma(u,v) - \sigma(u_0,v_0), \mathbf{V}_{1,0}) \\ (\sigma(u,v) - \sigma(u_0,v_0), \mathbf{V}_{2,0}) \end{pmatrix},$$

and if (u_0, v_0) is a regular point of σ, the jacobian matrix J_Φ of Φ is invertible in (u_0, v_0). Consequently, the inverse function theorem can be applied and there exists a function Ψ such that $\Psi(X,Y) = \Psi(\Phi(u,v)) = (u - u_0, v - v_0)$ in the vicinity of (u_0, v_0). Moreover, if Φ is C^{n+1} then Ψ is C^{n+1} and $J_\Psi(X,Y) = J_\Phi(u,v)^{-1}$ with $(X,Y) = \Phi(u,v)$.
With this statement, we know that we can have a Taylor expansion of $(u - u_0, v - v_0)$ with respect to (X,Y) up to order $n+1$. For this purpose, let us compute the higher-order derivatives of Ψ. As σ (and therefore Φ) is an analytical function issued from a CAD model, all its derivatives can be computed using the recipes in [Piegl and Tiller 1997]. The derivatives of Ψ can be then deduced using the following result [Encinas and Masque 2003]:

Theorem 4.3.2 (2D Faà di Bruno's Formula)**.** *Let us consider $f, g : \mathbb{R}^2 \to \mathbb{R}^2$ of class C^{n+1} with $n+1 \geq |\alpha|$, then $\forall x \in \mathbb{R}^2$, we have:*

$$\frac{\partial^{|\alpha|}}{\partial x^\alpha}(g(f(x))) = \sum_{|\sigma|=1}^{k} \alpha! \frac{\partial^{|\sigma|}}{\partial x^\sigma}(g(f(x))) \sum_{E_\sigma} \prod_{i=1}^{2} \prod_{A_\alpha} \frac{1}{e_{i\alpha^i}!} \left(\frac{1}{\alpha^i!} \frac{\partial^{|\alpha^i|}}{\partial x^{\alpha^i}}(f_i(x)) \right)^{e_{i\alpha^i}},$$

where $f(x) = (f_1(x), f_2(x))$, $\alpha = (\alpha_1, \alpha_2) \in \mathbb{N}^2$, $k = |\alpha| = \alpha_1 + \alpha_2$, $\sigma = (\sigma_1, \sigma_2) \in \mathbb{N}^2$

$$E_\sigma = \left\{ (e_{1\alpha^1}, e_{2\alpha^2}) \in \mathbb{N}^2, 1 \leq |\alpha^i| \leq k, \left(\sum_{|\alpha^i|=1}^{k} e_{i\alpha^i} = \sigma_i, \right)_{i=1,2} \right\}$$

and[5]

$$A_\alpha = \left\{ (\alpha^1, \alpha^2) : 1 \leq |\alpha^i| \leq k, i = 1,2, \sum_{i=1}^{2} \sum_{|\alpha^i|=1}^{k} e_{i\alpha^i}.\alpha^i = \alpha \right\}.$$

In our case, we set $g = \Psi$ and $f = \Phi$ and it comes that $\frac{\partial^{|\alpha|}}{\partial x^\alpha}(g(f(x))) = 0$ for $n+1 \geq |\alpha| \geq 2$.
If we consider all the $k+1$ possible values of α such that $|\alpha| = k$, then we have a system of equations:

$$A \left(\frac{\partial^{|\alpha|}\Phi}{\partial x^\alpha} \right)_{|\alpha|=k} \times \left(\frac{\partial^{|\alpha|}\Psi}{\partial x^\alpha} \right)_{|\alpha|=k} = F \left(\left(\frac{\partial^{|\alpha|}\Psi}{\partial x^\alpha} \right)_{|\alpha|<k}, \left(\frac{\partial^{|\alpha|}\Phi}{\partial x^\alpha} \right)_{|\alpha|\leq k} \right),$$

where A is $(k+1) \times (k+1)$ matrix, $\left(\frac{\partial^{|\alpha|}\Psi}{\partial x^\alpha} \right)_{|\alpha|=k}$ is a vector of size $k+1$ containing the $k+1$ derivatives of Ψ of order k and F is vector function of size $k+1$ that can be deduced from Theorem 4.3.2. Moreover, it is shown in [Encinas and Masque 2003] that $|A| = |J_\Phi|^k$, which proves that the system has always a solution if the considered point is regular.
This way, a recursive method to compute all the derivative of Ψ in (u_0, v_0) is set and the computation of the Taylor expansion is therefore possible:

$$\begin{pmatrix} u - u_0 \\ v - v_0 \end{pmatrix} = \sum_{k=1}^{n+1} \sum_{i+j=k} A_{ij}^k X^i Y^j + \mathcal{O}(||(X,Y)||^{n+2}),$$

5. Note that in practice, the sets E_σ and A_α can be precomputed once for all for the range of values of α and σ that are used.

where $A_{ij}^k \in \mathbb{R}^2$ and is defined thanks to the partial derivatives of Ψ.
But, as σ is C^{n+1}, we also have:

$$Z = \sum_{k=1}^{n+1} \sum_{i+j=k} c_{ij}^k (u-u_0)^i (v-v_0)^j + \mathcal{O}(||((u-u_0),(v-v_0))||^{n+2}).$$

By composition of both Taylor expansions, we then obtain a Taylor expansion of Z in (X,Y) at the order $n+1 \geq 2$ which is independent of the parameterization and a generalization of the Formula (4.6).
The gap to the tangent plane is thus expressed up to the order $n+1$:

$$Z = F_n(X,Y) + R_{n+1}(X,Y) + \mathcal{O}(||(X,Y)||^{n+2}), \tag{4.7}$$

where F_n is a polynomial of degree n and R_{n+1} is an homogeneous polynomial of degree $n+1$. For an approximation of the surface at the degree n, the leading term of the error is $R_{n+1}(X,Y)$ [Loseille and Alauzet 2011a, Loseille and Alauzet 2011b, Coulaud and Loseille 2016]. So, if we want to control the P^n approximation, we need to control $|R_{n+1}(X,Y)|$.
By applying the log-simplex algorithm explained in the previous section, we are able to find a metric that satisfies a similar inequality as (4.2), that is to say, we are able to compute a matrix Q_{n+1} such that:

$$|R_{n+1}(X,Y)| \leq \left(\frac{1}{2} (X\ Y) Q_{n+1} \begin{pmatrix} X \\ Y \end{pmatrix} \right)^{\frac{n+1}{2}},$$

where Q_{n+1} is the optimal symmetric matrix (in a sense explained in the first section) that verifies this inequality.
If we note $(\kappa_{i,n+1}, \mathbf{v}_{i,n+1})_{i=1,2}$, the eigenvalues and eigenvectors of Q_{n+1}, we can then reuse the metrics used for the linear meshing with $\rho_i = \frac{1}{\kappa_{i,n+1}}$ the radii of curvature, $\mathbf{V}_{i,n+1} = (\mathbf{V}_{1,0}\mathbf{V}_{2,0})\,\mathbf{v}_{i,n+1}$ the principal directions in the tangent plane and $\epsilon_{n+1} = \epsilon^{\frac{2}{n+1}}$ the threshold. This way, the classic formula is found for $n=1$ and a generalization is proposed for $n \geq 2$. Like for curves, in some configurations, found sizes for the order n can be significantly lower than found sizes for the order $n+1$. In this case, sizes given by order $n+1$ are preferred.

4.3.3 Meshing process

The mesh generation process is based on the classical metric-based concept, where a metric field, as $\mathcal{M}_2$ or $\tilde{\mathcal{M}}_2$, is used to drive the orientation and sizing of the elements. In the context of parametric surface mesh generation, several approaches are typically devised to generate a final 3D surface mesh. Full 2D methods are a convenient way to avoid 3D surface meshing and inverse projection to the geometry. However, a special care is needed to handle degenerated points, periodicity, highly non-uniform (even discontinuous) parameterization or degenerated edges. Approaches that mix 2D and 3D methods tend to reduce the impact of the parametric space to the final mesh.

In this chapter, we consider a rather classical approach. The core steps of the procedure are decomposed as follows:

1. For each Edge
   1.1 Generate a 3D adaptive mesh using $\mathcal{M}_1$
2. For each Face
   2.1 Generate a fast (u,v)-aligned tessellation,

2.2 Compute high-order metric $\widetilde{\mathcal{M}_2}$ on the tessellation,
2.3 Project 3D Edges of Loops as parametric curves and generate a 2D (u, v) mesh forming the boundary of the patch,
2.4 Recycle points from the tessellation : insert points from the tessellation into the current mesh [6],
2.4 Project to 3D, convert $\widetilde{\mathcal{M}_2}$ to $\mathcal{M}_2$ to adapt the mesh (with explicit projection to the geometry),
2.5 Generate high-order mesh.

The `EGADS` API [Haimes and Drela 2012] is used to perform the CAD linking and the planar mesh generation process is performed using a Delaunay triangulation-based algorithm [George and Borouchaki 1998].

4.4 Examples

We illustrate our approach on several examples. To compute error indicators, the computation of an exact Hausdorff distance would be too expensive. Therefore, a normalized and an absolute error in distance are computed. For each triangle of the generated surface mesh, a sampling of the underlying parametric space is performed and the distance between a point of the curved mesh and its surface counterpart is computed:

$$\delta(\tilde{u}, \tilde{v}, \tilde{w}) = ||\sigma(\sum_{i=0}^{N} \phi_i(\tilde{u}, \tilde{v}, \tilde{w})(u_i, v_i)) - \sum_{i=0}^{N} \phi_i(\tilde{u}, \tilde{v}, \tilde{w})\sigma(u_i, v_i)||,$$

where (u_i, v_i) is the set of points of the geometry used to define the high-order triangle ϕ_i the associated shape functions and $(\tilde{u}, \tilde{v}, \tilde{w})$ are the barycentric coordinates of the triangle. To get the normalized error in distance, the absolute distance is normalized by multiplying it by the local curvature of greater value.
The worst of all these errors then gives the overall errors of the meshing process.
Note that the relative error can be higher than the absolute one. This process can be explained for several reasons. Firstly, the absolute error, is dependent on the system of units. In the case of the shuttle, the average order of magnitude for the bounding box is $10^2 - 10^3$ while it is of $10^0 - 10^1$ in the case of the torus. Secondly, the point of worst value that determines the error is not necessarily the same for both errors. The worst value of the absolute error can occur in high stretching zones where the curvature is high, which emphasizes the value. Thus, it provides levels of error of greater magnitude.
Note that all the figures are obtained using `Vizir` [Loseille and Feuillet 2018].

Sphere example. Despite its simplicity, the unit sphere example is used to illustrate that our error estimate is independent to the parameterization of the model. A classic boundary representation of a sphere is to consider a surface of revolution, where the two pole maps to two degenerated edges. In the vicinity of these points, depending on the underlying CAD kernel, the definition of normals or more generally the definition of the derivatives of u, v are either undefined or unstable.

We illustrate the obtained Taylor expansion on one point of the sphere using the inversion formula given by Formula (4.7). Note that the sphere is parameterized as a circle

6. Points in the tessellation are generally closer when length are computed in $\tilde{\mathcal{M}}_2$. So the recycling consists in keeping points of the tessellation that are at a length greater than one.

of revolution. far from the pole, we have:

$$\begin{aligned} Z &= -0.5Y^2 - 0.5X^2 - 0.125Y^4 - 0.25X^2Y^2 - 0.125X^4 \\ &\quad -0.0625Y^6 - 0.1875X^2Y^4 - 0.1875X^4Y^2 - 0.0625X^6 \\ &\quad +\mathcal{O}(||(X,Y)||^7). \end{aligned}$$

In the vicinity of the two poles, the expansion becomes:

$$\begin{aligned} Z &= -0.5Y^2 - 0.5X^2 - 0.125Y^4 - 0.25X^2Y^2 - 0.125X^4 \\ &\quad -1.80444e^{-9}XY^4 + 1.9886e^{-9}X^3Y^2 - 0.0625Y^6 - 0.187501X^2Y^4 \\ &\quad -0.187499X^4X^2 - 0.0625X^6 + \mathcal{O}(||(X,Y)||^7). \end{aligned}$$

We observe that a numerical noise appears while an almost perfect expansion is obtained as for regular points. As expected, the second order terms reveal half of the principal curvatures. This numerical noise is a consequence of the manipulation of the huge values at stake in the vicinity of the apex. As the value of the derivative of the parameters is going to 0, the normalization can lead to arbitrary huge values. In the end, everything is simplified but the numerical computations are impacted by this.

Torus. The torus example is based on a NURBS representation is based on two NURBS of degree 5 both defined by 6 control points and 12 knots. P^1, P^2 and P^3 meshes are generated for this case. The pointwise normalized and absolute error in distance are reported in Table 4.1. The meshes and pointwise errors are reported in Figure 4.6. In particular, we see that the error on the annulus of the torus is not significant anymore when the degree is higher than 1 as an high-order expansion is more able to capture this feature.

Order	DOF	Normalized	Absolute
P^1	2787	0.0371489	0.275027
P^2	666	0.0287625	0.18465
P^3	252	0.0137922	0.0883003

Table 4.1 – Deviation to the geometry for torus geometry for P^1 to P^3 meshes.

Also, to show the efficiency of the method on this case, a study has been made between a method where the P^1 metric is used and a method where a P^n metric is used to generate a mesh of degree n. For various absolute error levels, P^1-metric based and P^n-metric based meshes are generated and comparison are done with regards to the number of degrees of freedom. In Table 4.2 (resp 4.3), a comparison between a P^1-metric and a P^2-metric (resp. P^3-metric) is done. It shows in particular, that for a given error, the number of requested degrees of freedom is higher using the P^1-error than the P^2 (resp. P^3) one. If we have a look at the meshes with the pointwise error in Figure 4.7 (resp 4.8), we explain it by the fact that the high-order metric is able to detect high-order features that the P^1-metric does not detect. In particular, we see that the error due to the presence of an annulus on the torus is mostly canceled. And from a more global point a view, the high-order metric provides a mesh with a more balanced distribution of the error with a lower number of degrees of freedom.

Shuttle. The shuttle geometry is based on two NURBS of degree 3 defined by 8 (resp. 13) control points and 12 (resp. 17) knots with strong variation in the parametric space. The P^1, P^2 and P^3 meshes are depicted in Figures 4.9. The error to the geometry are reported

Error	P^1-metric #DOF	P^2-metric #DOF
0.5	347	156
0.2	687	242
0.03	1556	1036

Table 4.2 – Comparison between the number of needed degrees of freedom for the generation of a P^2 mesh to get a given error between a P^1-metric and a P^2-metric.

Error	P^1-metric #DOF	P^3-metric #DOF
0.1	255	238
0.002	6171	1848
0.0002	11329	5362

Table 4.3 – Comparison between the number of needed degrees of freedom for the generation of a P^2 mesh to get a given error between a P^1-metric and a P^3-metric.

in Table 4.4. In particular, we see that the error in high stretching areas is significantly reduced by the increase of the order.

Order	DOF	Normalized	Absolute
P^1	1647	254.551	197.814
P^2	1690	61.6727	58.3507
P^3	1791	7.02513	22.217

Table 4.4 – Deviation to the geometry for the shuttle geometry.

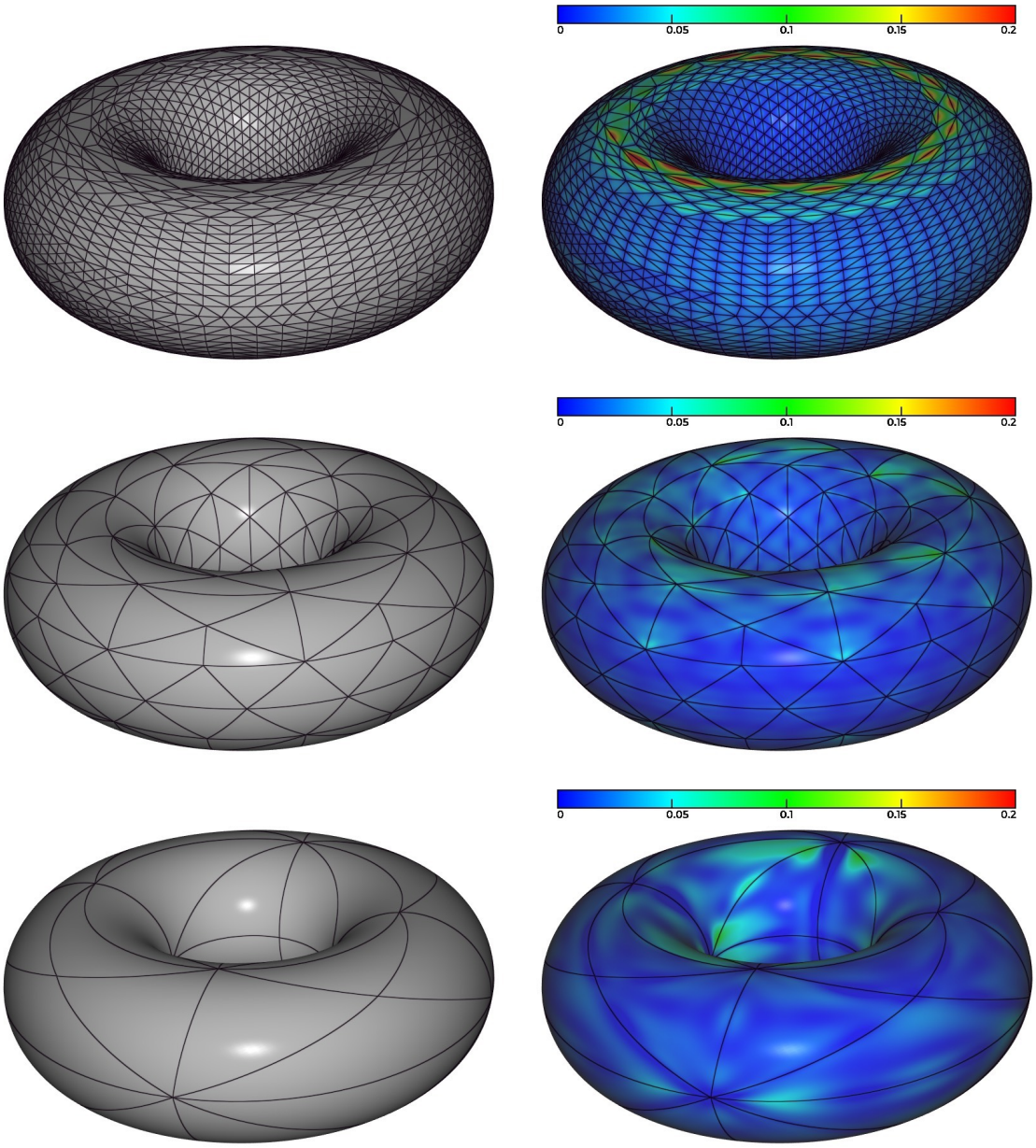

Figure 4.6 – P^1 (top), P^2 (middle) and P^3 (bottom) meshes (left) and point-wise distance (right) to the torus geometry.

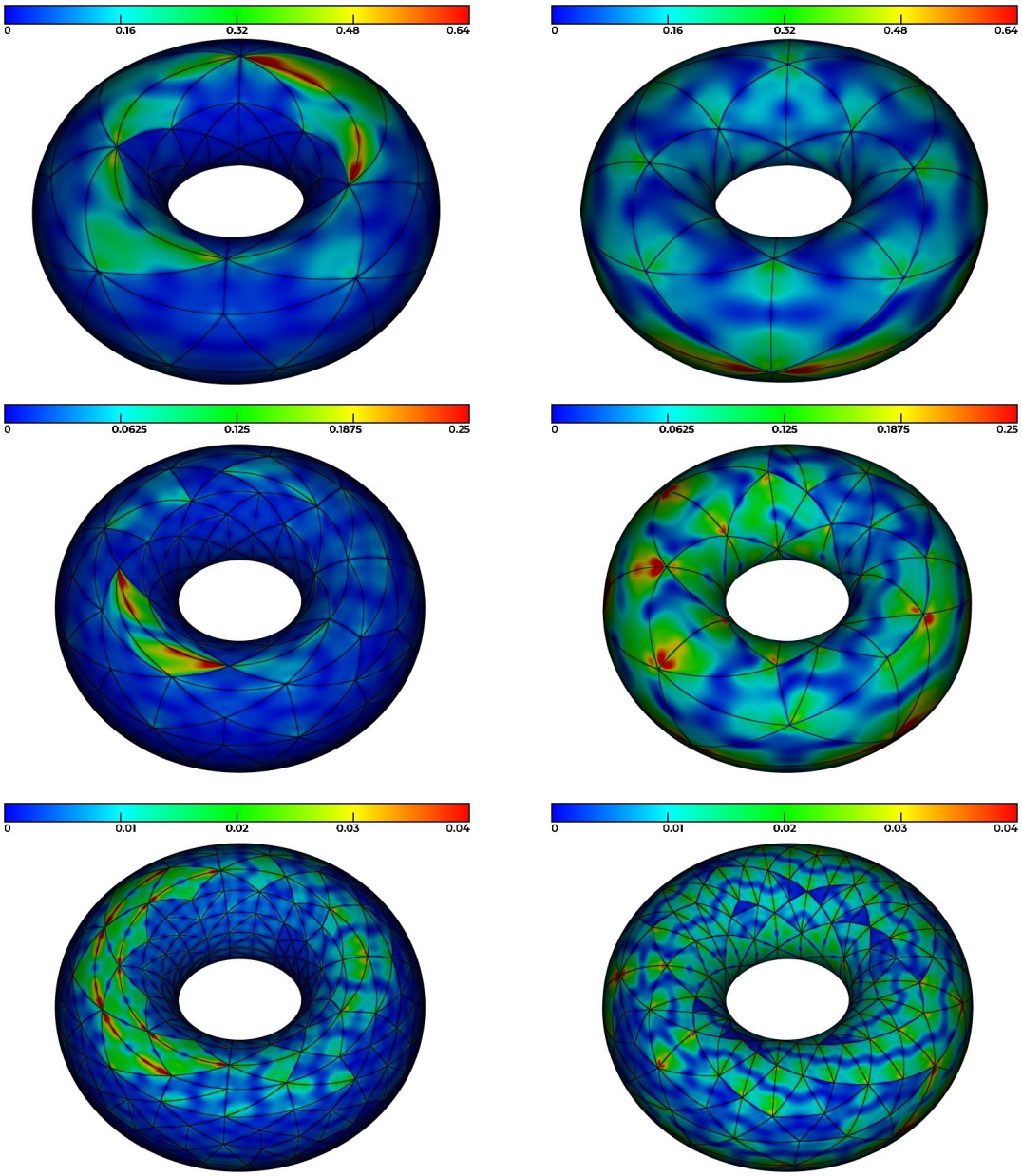

Figure 4.7 – P^2 meshes generated with a P^1-metric (left) and P^2-metric (right) for various error levels: 0.5 (top), 0.2 (middle), 0.03 (bottom)

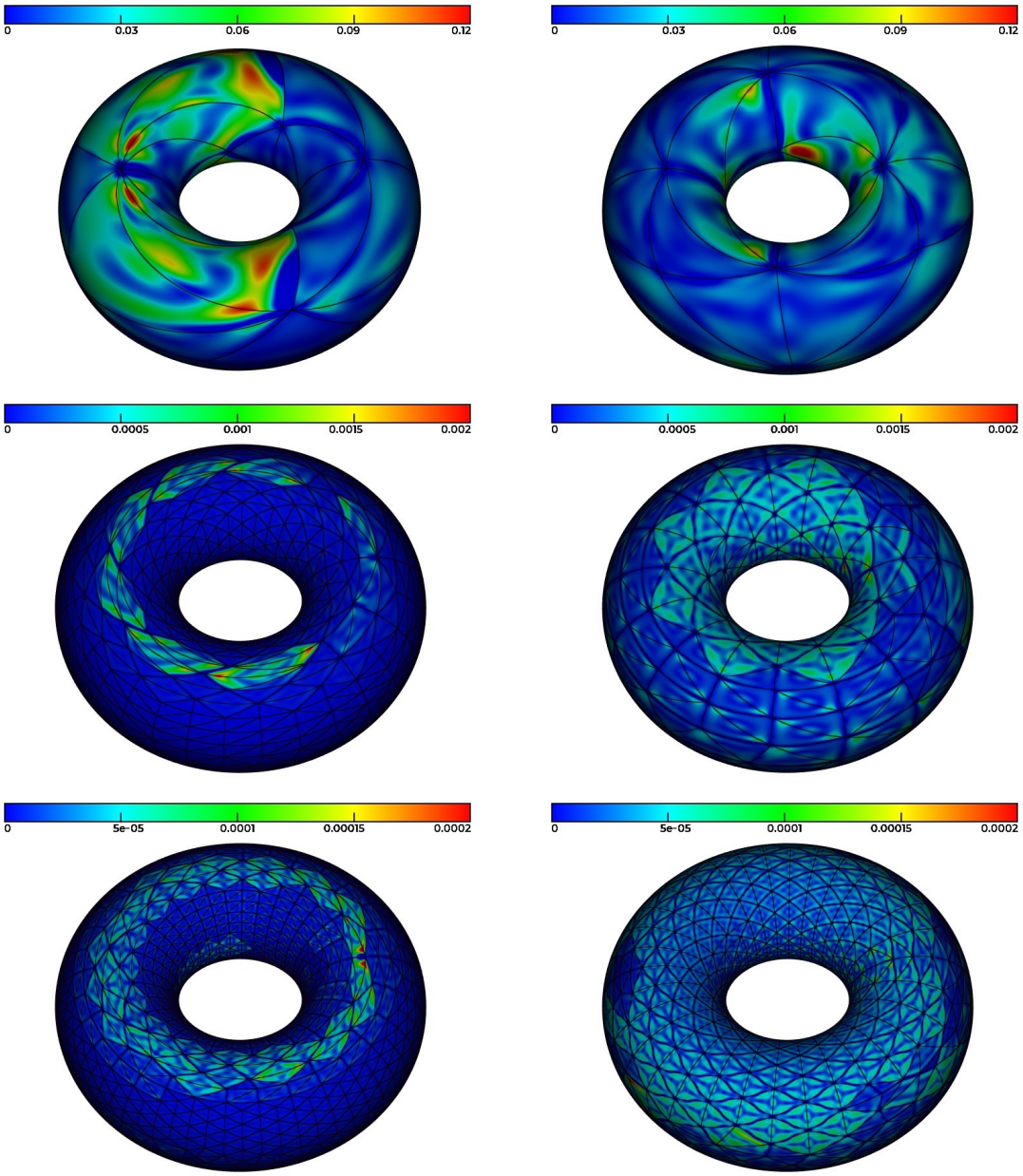

Figure 4.8 – P^3 meshes generated with a P^1-metric (left) and P^3-metric (right) for various error levels: 0.1 (top), 0.002 (middle), 0.0003 (bottom)

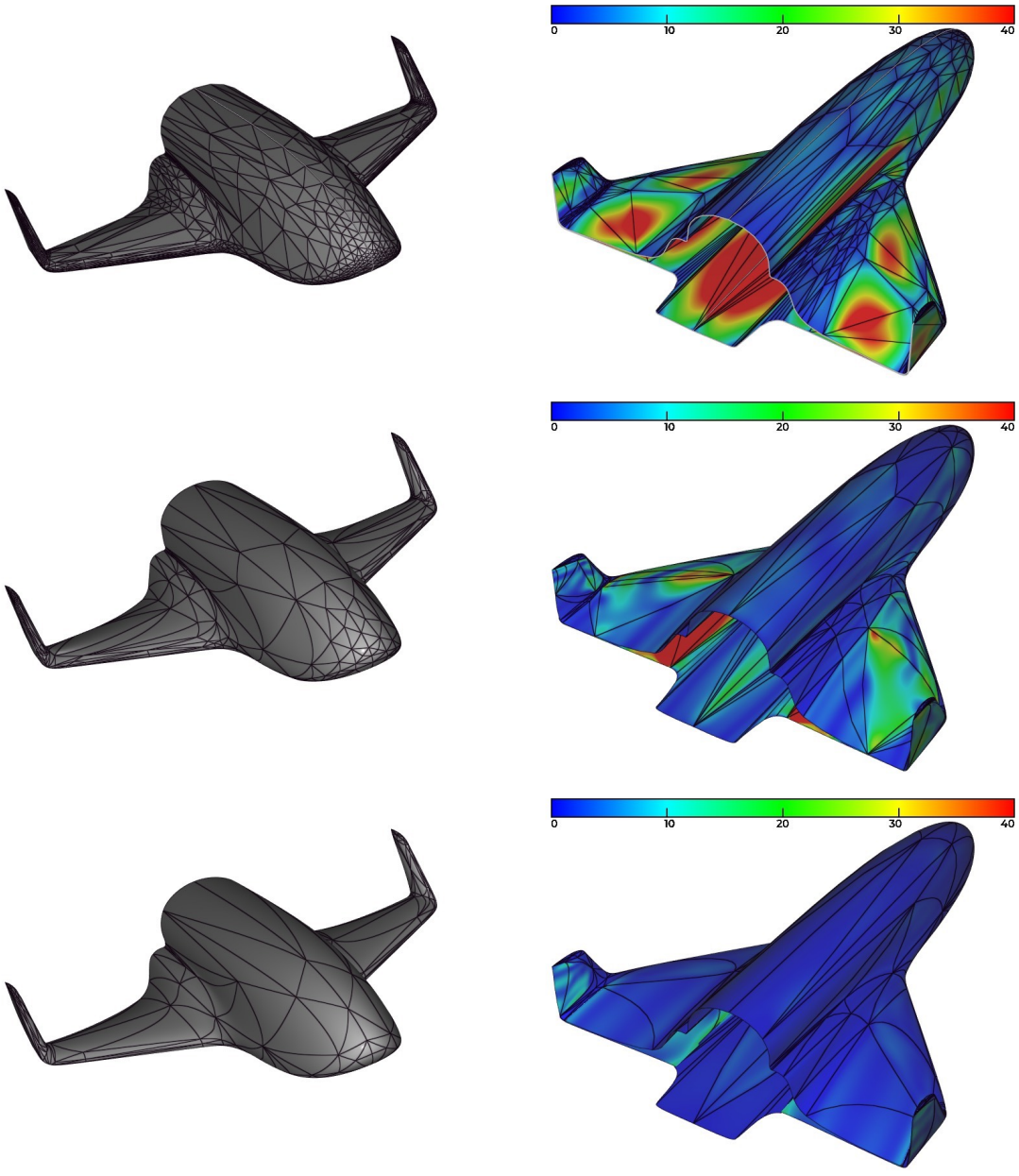

Figure 4.9 – P^1 (top), P^2 (middle) and P^3 (bottom) meshes (left) and point-wise distance (right) to the shuttle geometry.

4.5 Conclusion

Generating high-order curved surface meshes from a geometry require the derivation of intrinsic quantities of the surface in order to guarantee the approximation. In this chapter, we have used a Taylor expansion coupled with an inversion formula to derive a local approximation of the underlying surface in the Frénet frame. To extend the notion of principal curvature, a log-simplex approach is used to approximate optimally the variation of the polynomial by a quadratic function. The eigenvalues and eigenvectors can be viewed as "high-order" curvatures. As a metric field is naturally derived, these estimates can be used directly in any adaptive anisotropic mesh generation process. All these developments have been made in the `CADSurf` module of `Feflo.a/AMG` using some linkings with `pkmetrix` and this work has been accepted for publication at a conference [Feuillet et al. 2019a].
Natural perspectives are possible for this work. Notably, this work could be extended to the implicit surface meshing. Indeed, once the implicit function theorem is applied, it is possible to apply the previous framework in order to get a Taylor expansion at any point of the surface and consequently deduce a metric. The only limitation to this meshing method would be to find the set of points where the surface is evaluated, which is the same problem as in linear meshing of implicit surfaces.
Also, the parametric mesh generation process could be improved by performing metric-based curvilinear mesh adaptation to optimize the position of the high-order nodes. This problem will be the subject of future work.

Chapter 5

Optimization of P^2 meshes and applications

5.1 Introduction

Mesh optimization techniques are commonly used in the meshing community. The underlying idea consists in improving the mesh with regards to a given quality criterion, the choice of this criterion being driven by the application. For instance, it can be used to provide, after optimization, a mesh for better finite element computations or a rendering mesh suitable for visualization purposes. In our context, the used criteria are those that can be used for optimization as part of a mesh generation process [Brière de L'isle and George 1995] or for moving-mesh techniques [Dobrzynski and Frey 2008, Alauzet 2014, Dapogny et al. 2014]. These optimization procedures use in particular two local operators that are known as (generalized) edge/-face swapping and (Laplacian) vertex smoothing [1].
In this chapter, P^2 mesh quality-based optimization operators are presented. They are a generalization of the classical P^1 operators and rely on the resolution of a local optimization problem. These two operators are applied to improve P^2 mesh generation in a process starting from a P^1 mesh. It also enables high-order connectivity-change moving mesh methods.
To this end, Section 5.2 deals with P^2 mesh optimization. Then, three types of applications are shown. The first one presented in Section 5.3 is an improvement of a P^2 mesh generation technique that curves a P^1 mesh thanks to a high-order linear elasticity solver. The second one presented in Section 5.4 describes a P^2 connectivity-change moving mesh method also using a high-order linear elasticity solver. The last one presented in Section 5.5 describes a P^2-boundary layer mesh generation process.

5.2 P^2 mesh optimization

In the same way as we want to have an optimal P^1 mesh in terms of quality, we want to have an optimal P^n mesh. Several optimization techniques exist to correct an invalid P^n mesh [Toulorge et al. 2013, Karman et al. 2016] and to optimize the geometrical accuracy [Toulorge et al. 2016]. The idea here is to extend two classic mesh quality-based optimization operators used in [Alauzet 2014] to P^2 meshes to increase its quality. The quality function is the one presented in Section 3.4.2.

5.2.1 P^2 swap operator

The swap operator (see Figure 5.1) locally changes the connectivity of the mesh in order to improve its quality. In 2D, it consists in flipping an edge shared by two triangles to form two new triangles with the same four vertices (see Figure 5.1 left).

1. Additional operators exist, point insertion and edge collapse, but this consists more in local remeshing than in mesh optimization.

In 3D, two types of swap exist: face and edge swapping. The face swapping is the extension of the 2D edge swapping, it consists in replacing the common face of two neighboring tetrahedra by the edge linking the opposite vertices to the common face of each tetrahedron, also called $2 \rightarrow 3$. The edge swapping is a bit different. First, the shell of the edge to delete (e.g. the set of elements containing this edge) is constructed. From a shell of size N, a non-planar pseudo-polygon formed by N vertices is obtained. The swap consists in deleting the edge, generating a triangulation of the polygon and creating two tetrahedra for each triangle of the triangulation thanks to the two extremities of the former edge. These swaps are designated as $N \rightarrow M$ with $N \geq 3$, where N is the initial number of tetrahedra and $M = 2(N-2)$ is the final number of tetrahedra. Figure 5.1 shows the case of three tetrahedra around an edge.

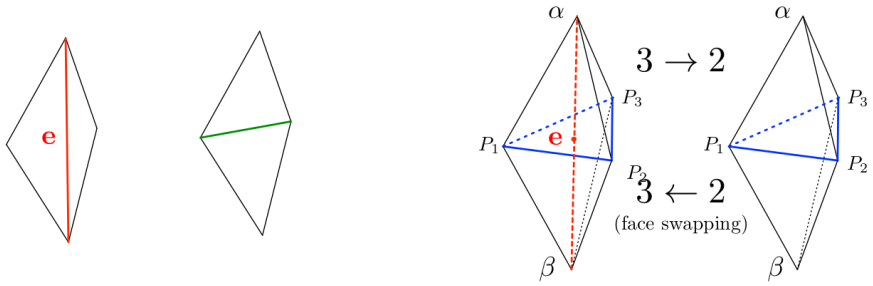

Figure 5.1 – Left, the swap operation in 2D. Right, edge swap $3 \rightarrow 2$ and face swap $2 \rightarrow 3$. For all these pictures, shells are in *black*, old edges are in *red*, new edges are in *green*.

For each possible swapped configuration, if the worst quality of all the elements the shell is improved, the configuration is kept and will be in the new mesh unless another swapped configuration of the shell provides a better quality improvement. When it comes to connectivity-change moving-mesh algorithms in 3D, a small local degradation of the shell's worst quality has been observed as an efficient way to improve the global quality of the mesh [Barral 2015].
To generalize to P^2 meshes, the inner nodes of the edges of the shell have to be taken into account. For instance, for the P^2 case in 2D, there is one node on the swapped edge and if we want the swap to be performed, we need first to find an optimal position for the node in the swapped configuration and then to check if this configuration improves the quality function (see Figure 5.2). The key feature is therefore to find a functional whose optimum will give the optimal position for the node in the swapped configuration in terms of quality.

In this context, the idea is to find a simple and smooth functional that will be easy to optimize. The quality function (3.15) is not a good candidate as it is not smooth. We propose to consider the following functional (inspired by the work of [Toulorge et al. 2013]):

$$f(\mathbf{X}) = \sum_{K \in \mathcal{S}(e)} \sum_{i+j+k+l=d(n-1)} \omega_{ijkl} \left(\frac{N_{ijkl}(\mathbf{X})}{d! V_1} - 1 \right)^2,$$

where d is the dimension, n the degree (in the following $n = 2$), $\mathbf{X} = [\mathbf{x_1}, ..., \mathbf{x_N}]$ is a *set* of N coordinates to be optimized, $\mathbf{X} \rightarrow N_{ijkl}(\mathbf{X})$ is a function that depends, in the worst case on two variables of $\mathbf{X}$, $\mathcal{S}(e)$ is the shell of the initial edge e (e.g. the set of elements K containing e), ω_{ijkl} is a weight function that measures the importance of each control coefficient, and V_1 is the volume of the straight-sided element deduced from K. Note that $\mathbf{X}$ can either be empty (in which case no optimization is required), or contains more than

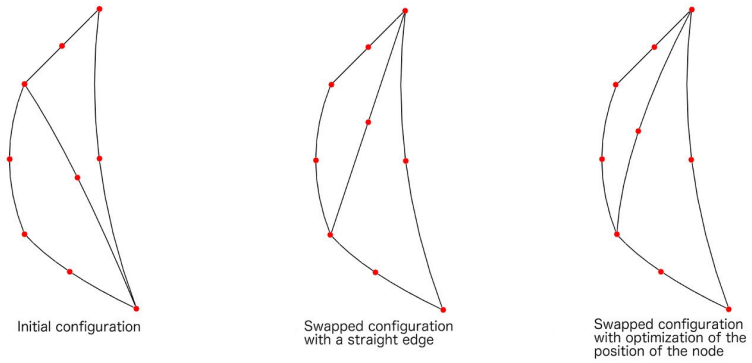

Figure 5.2 – Three steps of P^2 swap in 2D.

one node's coordinates as it represents the coordinate of the created edges of the shell.
In 2D, $\mathbf{X}$ is always a singleton and is simply noted $\mathbf{x}$, weights ω_{ijk} $(l = 0)$ are equal to 2 for the corner coefficients and equal to 1 elsewhere. In this case, f has the following properties : it is a positive define quadratic form as $\mathbf{x} \to N_{ijk}(\mathbf{x})$ is linear in $\mathbf{x}$, which means that the functional has a unique minimum. Also, on every regular swap configuration, the minimum of f in $\mathbf{x}$ is the same as the minimum of the worst quality of the swapped shell in $\mathbf{x}$. Using the result of the optimization problem in the swap configuration gives therefore a very good approximation of the best potential configuration. Since the best swap configuration is found, we are able to conclude if this swap will improve the quality or not.
In 3D, weights ω_{ijkl} are equal to 4 for a corner control coefficient, equal to 2 for an edge control coefficient and equal to 1 otherwise. In this work, we consider the swaps $2 \to 3$, $3 \to 2$ and $4 \to 4$. This means that the functional of the problem is in the worst case quadratic since the control coefficients have a linear dependence with respect to a given control point. Also, the problem appears numerically to be definite positive on a regular configuration. Note that these swaps represent $\sim 95\%$ of the swaps performed during a P^1 mesh optimization. For the other swaps $(5 \to 6,\ 6 \to 8)$, the problem begins to be more costly in terms of CPU.
The resolution of these optimization problems is performed thanks to a L-BFGS algorithm [Liu and Nocedal 1989]. Like several optimization methods, this method requires the computation of the gradient of f with respect to the coordinates. The computation of the gradient of f mainly relies on the computation of the gradient of the N_{ijkl}. The computation of this gradient is done in two steps. First, the gradient with respect to the control points is computed. As the control coefficient are determinants based on linear combinations of the control points, it is always possible to rewrite them so that a control point appears at most once in the determinant. In this case, the computation of the gradient is straightforward. Then, by composition of the gradient (the correspondence between Lagrange points and control points is linear), we have:

$$\nabla_{A_m} N_{ijkl} = ((M_{L2B})^T \nabla_C N_{ijkl})_m,$$

where A_m is m^{th} index of the node of coordinates x in K, $\nabla_C N_{ijkl}$ is the gradient of N_{ijkl} with respect to the control points and M_{L2B} is the matrix which converts Lagrange points into control points.
Finally, note that even if the quality function is not used for the optimization problem, it is mandatory to still use it for the decision process so that it degenerates into classic swap operator when the elements are straight-sided.

5.2.2 P^2 mesh smoothing

Mesh smoothing is a technique that consists in relocating some points inside the mesh to improve the quality of the elements. In P^1, the idea is to relocate each vertex P_i inside its ball of elements (see Figure 5.3). For each element K_j in the ball of P_i, the opposite face to P_i denoted by F_j gives an optimal position P_j^{opt}. Then, the vertex is relocated considering a weighted average of the proposed positions. If the proposed new location of the vertex does not improve the ball configuration in term of quality, then relaxation is performed to check if an improved configuration exists between the original location and the new one. The optimal configuration is computed in 3D as follows:

$$P_j^{opt} = G_j + \sqrt{\frac{2}{3}} h_j \frac{\mathbf{n}_j}{||\mathbf{n}_j||},$$

where G_j is the gravity center of F_j, h_j is the average length of the edges of F_j, and $\mathbf{n}_j$ is the outward normal to F_j. The proposed position is then computed with:

$$P_i^{opt} = \frac{\sum_{K_j \in Ball(P_i)} \max(Q(K_j), Q_{\max}) P_j^{opt}}{\sum_{K_j \in Ball(P_i)} \max(Q(K_j), Q_{\max})},$$

where $Q_{\max}$ is a user parameter. Here $Q_{\max} = 10$. Note that if every computed configuration decreases the quality, the smoothing is not performed.

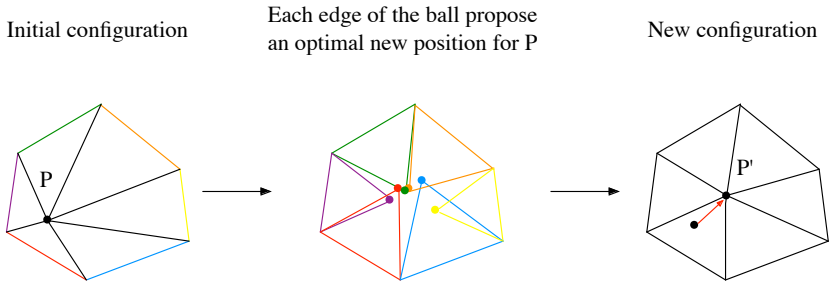

Figure 5.3 – Laplacian smoothing in two dimensions. Each element of the ball of considered vertex P_i suggests an optimal position for P_i. The resulting new optimal position for P_i is computed as a weighted average of all these proposed locations.

To extend it to P^2 meshes, the edges' node needs to be taken into account. The idea here, is to perform two independent smoothing operations:

- A vertex smoothing

The vertex smoothing is simply a generalization of the P^1 smoothing. The optimal position of the vertex is computed in the same way as in P^1, and the vertex is located exactly in the same way as before. In order to be consistent with the P^1 vertex smoothing and to keep in the ball straight edges that are initially straight, the displacement of all the inner nodes of the ball cavity is set to half of the value of the displacement of the central vertex (see Figure 5.4). In the exact same way as in P^1 if the final configuration does not improve the quality in P^2, relaxation is performed between the original location and the new one.

- A node smoothing

The optimization of the node position follows the same algorithm as for the P^2 swap operator to find its optimal position. For this purpose, functional f can be reused to find the optimal node position (see Figure 5.5). In this case, there is always only one node

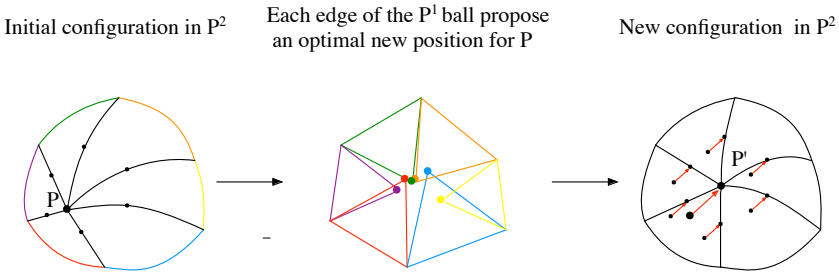

Figure 5.4 – P^2 Laplacian smoothing in two dimensions. Each element of the P^1 ball of considered vertex P suggests an optimal position for P. The resulting new optimal position for P is computed as a weighted average of all these proposed locations. The new position of the nodes of the internal edges of the P^2 ball is then deduced by proportionality.

coordinates to optimize and consequently the optimization problem is quadratic. In the exact same way as in P^1, if the final configuration does not improve the quality, relaxation is performed between the original location and the new one and the best case of all the studied configurations is kept.

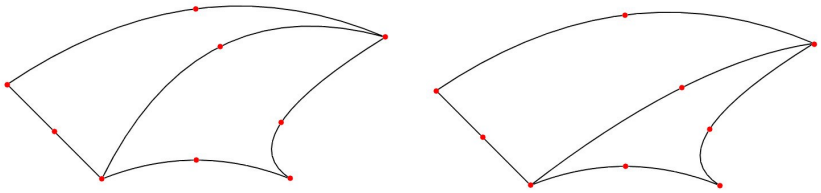

Figure 5.5 – P^2 node smoothing in two dimensions. The optimal position of the node of the central edge is computed solving an optimization problem. Left, the initial configuration, right, the optimized configuration.

5.3 P^2 simplicial mesh generation by curving an initial P^1 mesh

5.3.1 Mesh curving algorithm

Most of the techniques to generate an high-order mesh is to start from a P^1 mesh and then to curve it, in order to obtain a P^n mesh. The main reason to use a post-treatment is that all existing P^1 mesh generation algorithms can be reused. To curve meshes, used models are numerous : PDE or variational models [Persson and Peraire 2009, Xie et al. 2013, Abgrall et al. 2014b, Fortunato and Persson 2016, Moxey et al. 2016, Turner et al. 2016, Hartmann and Leicht 2016], smoothing and/or optimization procedures [Dey et al. 1999, Sherwin and Peiró 2002, Toulorge et al. 2013, Karman et al. 2016, Ruiz-Gironès et al. 2016a], ... Our choice here is to do a trade-off between both approaches: we use the linear elasticity equation as a model for the motion of the vertices to generate a P^k mesh from a P^1 mesh and perform mesh optimization in a second step to optimize the result. For this purpose, let us consider the linear elasticity equation with Dirichlet

boundary conditions:

$$\nabla \cdot (\sigma(\mathcal{E})) = 0\,, \quad \text{with} \quad \mathcal{E} = \frac{\nabla\xi + (\nabla\xi)^T}{2}\,, \tag{5.1}$$

where σ, and $\mathcal{E}$ are respectively the Cauchy stress and strain tensors, ξ is the Lagrangian displacement. The Cauchy stress tensor follows the Hooke's law for isotropic homogeneous medium.

Here, Dirichlet boundary conditions represent the gap between the P^k-nodes of the initial straight boundary elements and their position on the *real* boundary. For mesh boundary vertices, the gap is equal to 0.

To compute the gap, two different techniques are used:

- First case, the boundary is analytically known. In this case, the gap is only the difference between a node on the straight-sided boundary element and its projection on the analytical surface (see Figure 5.6).
- Second case, the boundary is only known via its P^1 discretization. In this case, a cubic reconstruction of the surface is performed to replace the analytical function by a discrete definition. The gap is then computed in the same way as in the first case.

The cubic reconstruction [Vlachos et al. 2001] (see Figure 5.6) relies on the Bézier representation of a curve. Let us give two normals at two vertices A and B, the idea is to find the two Bézier control points P and Q by choosing:

- P as the orthogonal projection of the point X such that $X = \frac{1}{3}A + \frac{2}{3}B$ on the tangent vector/plane associated with the first point.
- Q as the orthogonal projection of the point Y such that $Y = \frac{2}{3}A + \frac{1}{3}B$ on the tangent vector/plane associated with the second point.

The normal at a vertex is computed as the weighted sum of the normals of all the boundary elements containing this vertex. Note that in the case of corners and ridges a special procedure is required. This way a P^3 curve is obtained. For the computation of the inner Bézier control points of the P^3 triangle, a *Serendipity* model [George et al. 2014] is used. The central Bézier control point is computed as a weighted sum of the vertices and the edges control points of the triangle ABC:

$$P^{central} = -\frac{1}{6}(A + B + C) + \frac{1}{4}\sum_{i=1}^{3}(P_1^{edge_i} + P_2^{edge_i}).$$

The gap is then computed by using the projection of the P^n nodes on this P^3 reconstruction. Note that other techniques exist to generate a high-order mesh from a linear mesh, see for example [Ims and Wang 2019] or [Piegl and Tiller 1997] for a more global approach.

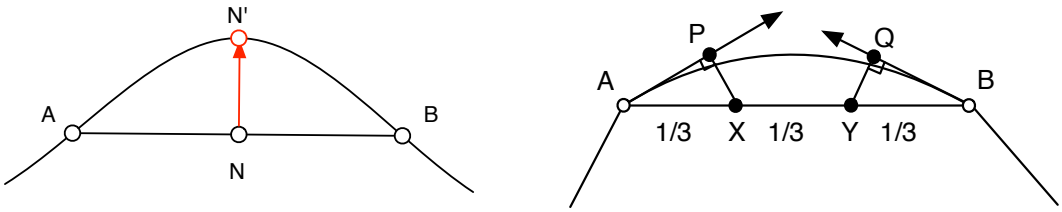

Figure 5.6 – Left, method to compute the gap between nodes on the straight line and real boundary. Right, the cubic reconstruction technique.

Once Dirichlet boundary conditions are set, the high-order finite element linear elasticity code is called. As the high-order element K is straight (we are still on the initial linear mesh), it is possible to compute analytically the linear system integral coefficients (it has a lower CPU cost than a quadrature) by remembering that the shape functions and their derivatives are only polynomial combinations of the barycentric functions and their (constant) derivatives. The analytical formula is provided for all cases (see Appendix B for more precisions). The FE system is then solved by a Conjugate Gradient algorithm coupled with a LU-SGS preconditioner. The elasticity problem using the high-order FEM provides the new position of the internal vertices and nodes. It is then used to generate the high-order mesh by moving the vertices and nodes of the initial straight-sided mesh with the associated values in the elasticity solution vector. If some elements remain invalid after optimization, it is always due to non-suitable boundary displacements. In this case, the FEM solution is proportionally reduced in the vicinity (boundary included) of the invalid element until global validity is obtained. The process is summarized in **Algorithm 22**.

Algorithm 22: Mesh curving algorithm

1. Generate a P^1 mesh.
2. Perform P^1 mesh optimization preprocessing: generalized swapping and vertex smoothing.
3. Perform cubic reconstruction of the boundary or use its analytical representation to set Dirichlet boundary conditions for the linear elasticity equation.
4. Solve linear elasticity equation on the P^1 mesh with the FEM at the order of the wanted mesh.
5. Generate the P^n mesh by moving the P^1 mesh with the solution of the linear elasticity.
6. Perform P^2 mesh optimization post-processing: generalized swapping and node/vertex smoothing.
7. Check validity of P^n elements and locally relax the FEM solution if necessary or desired until it is valid.

The major fact with this method is that the deformed mesh is only made of isotropic or almost isotropic elements. The studied analogy states that the mesh can be assimilated to continuous and isotropic media. Consequently, the use of the elasticity problem on isotropic meshes is efficient and always provides a valid mesh. Optimization in the preprocessing makes elements more isotropic and therefore helps curvature process to be more robust whereas optimization in the postprocessing improves the quality of the mesh and untangle invalid elements if any. Results for P^2 meshes are presented hereafter (the visualization of the meshes is performed using the tools developed in Chapter 6).

5.3.2 NASA Rotor 37

This example is a NASA Rotor 37 used for turbomachinery applications. Two different meshes are considered (see Figure 5.7).

- A coarse mesh (see Figure 5.7, middle) with an initial number of 2 434 vertices, 13 326 nodes and 10 970 tetrahedra. The initial mesh average quality is 6.16 with a

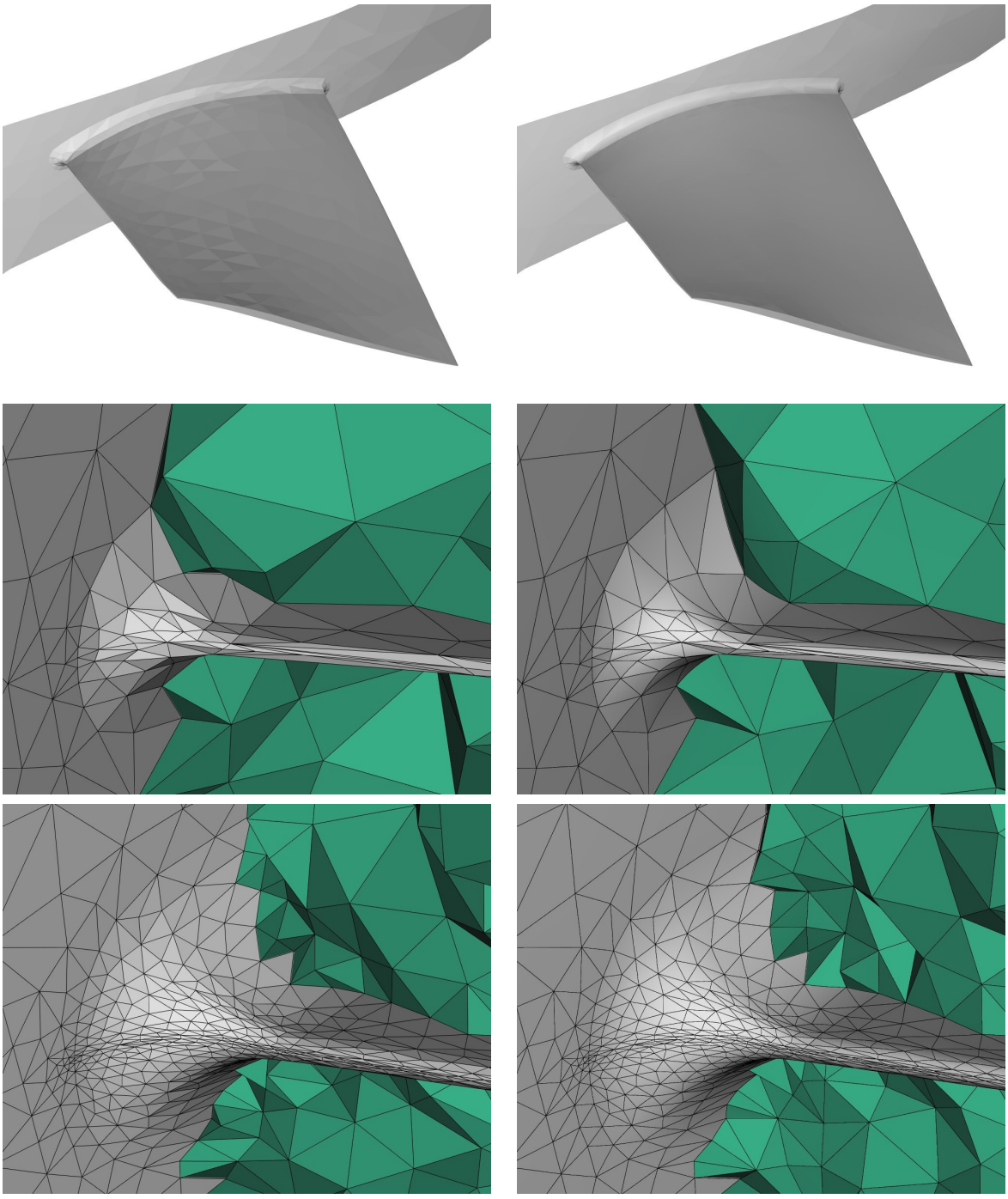

Figure 5.7 – NASA Rotor 37 turbomachinery meshes. From top to bottom, surface, coarse and fine meshes. Left, P^1 mesh and right, P^2 mesh. P^2 meshes are generated with Algorithm 22 using a cubic reconstruction.

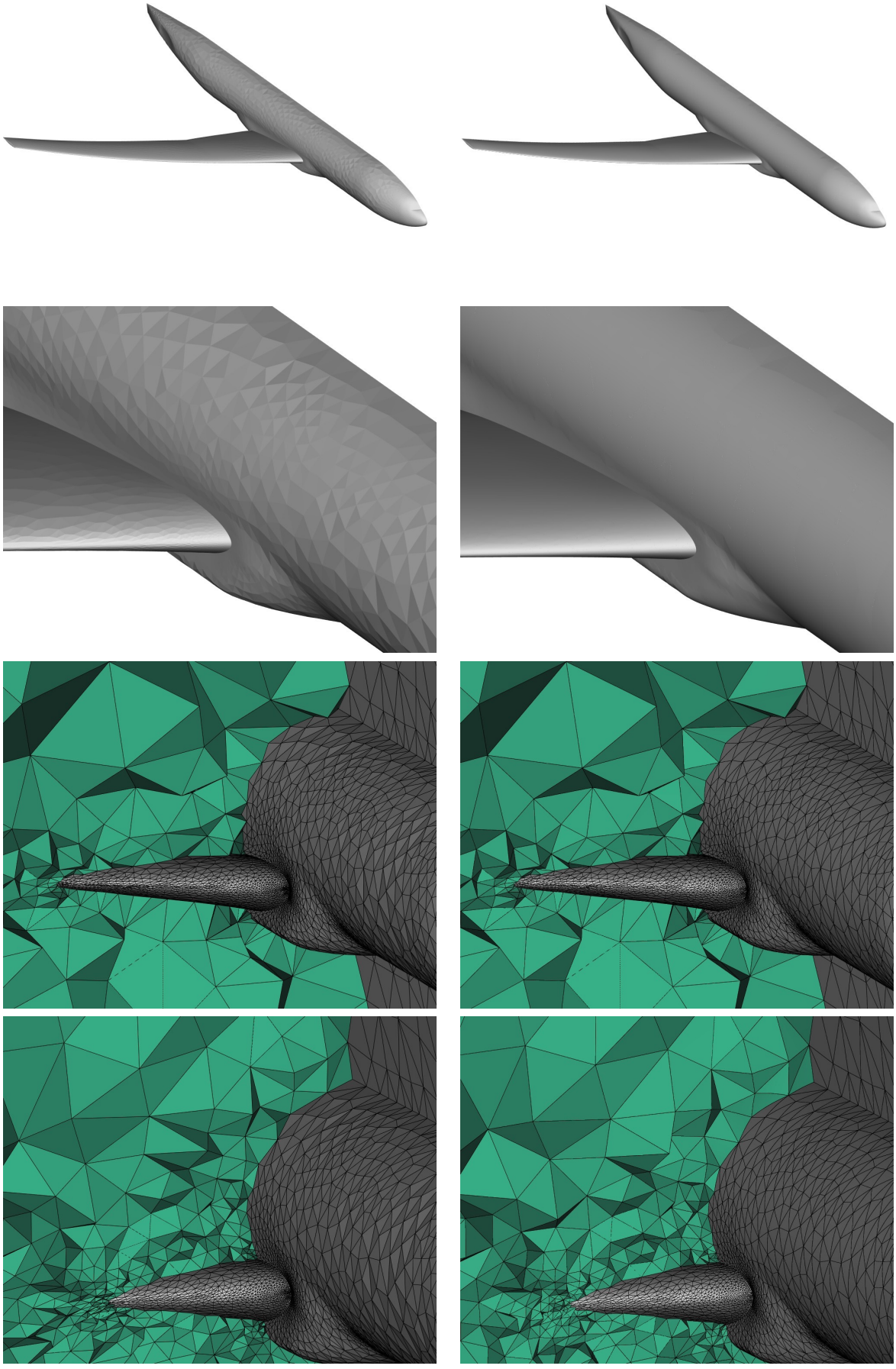

Figure 5.8 – NASA Common Research Model aircraft meshes. From top to bottom, surface, coarse and fine meshes. Left, P^1 mesh and right, P^2 mesh. P^2 meshes are generated with Algorithm 22 using a cubic reconstruction.

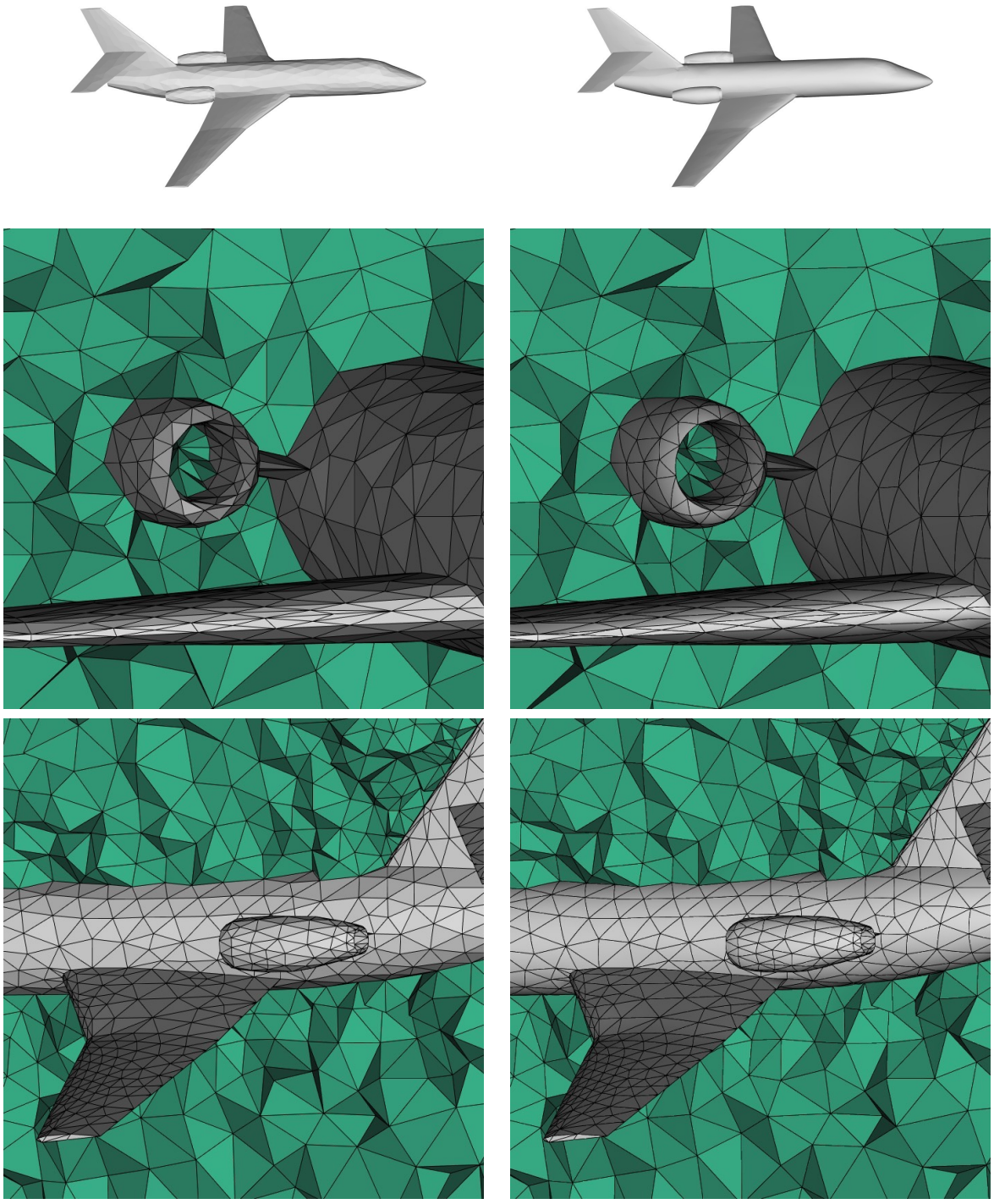

Figure 5.9 – Falcon business jet meshes. From top to bottom, surface and volume meshes. Left, P^1 mesh and right, P^2 mesh. P^2 meshes are generated with Algorithm 22 using a cubic reconstruction.

worst quality of 1 682 due to the presence of sharp anisotropic trailing and leading edges on the input rotor surface mesh. Curving the mesh without optimization provides an average quality of 5.3 with a worst quality of 1 729. Optimization in post and preprocessing improve average quality to 2.5 and worst quality to 1 346. Note that the number of nodes and tetrahedra is changed to respectively 11 598 and 10 862.

- A finer mesh (see Figure 5.7, bottom) with an initial number of 22 145 vertices, 137 393 nodes and 106 562 tetrahedra. The initial mesh average quality is 2.14 with a worst quality of 111. Note that it is better than with the coarse mesh as the mesh is finer and therefore deals better with trailing and leading edges. Curving the mesh without optimization provides an average quality of 2.15 with a worst quality of 366. Optimization in post and preprocessing improve average quality to 1.7 and worst quality to 27.5. Note that the number of nodes and tetrahedra is changed to respectively 136 924 and 106 093.

For both cases, we clearly observe the benefits of the optimization that improves the average and the worst quality of the final mesh. Note that the worst quality of the final P^2 might be greater than the one of the initial P^1 mesh. This a consequence of the curving process that decreases the quality of straight elements by curving them. We can also observe that the curvature is not propagated far into the volume. The curvature disappears after 2 or 3 layers of elements (see Figure 5.7, middle and bottom right). This is an illustration of St Venant's principle which states that the elasticity solution can be divided into transmissive effects and local disturbances.

5.3.3 NASA Common research model aircraft

This example is the NASA Common Research Model, an aircraft model that is commonly used in both experimental and numerical simulations in fluid dynamics (see Figure 5.8). Again, two meshes are considered:

- A coarse mesh (see Figure 5.8, line 3) with an initial number of 32 479 vertices, 173 468 nodes and 118 012 tetrahedra. The initial mesh average quality is 2.49 with a worst quality of 271 due to the presence of sharp anisotropic trailing and leading edges on the input wing surface mesh. Curving the mesh without optimization provides an invalid configuration with 10 invalid elements. Optimization in post and preprocessing improve average quality to 2.13 and worst quality to 271. Note that the number of nodes and tetrahedra is changed to respectively 173 744 and 118 288.
- A finer mesh (see Figure 5.8, line 4) with an initial number of 101 422 vertices, 660 071 nodes and 535 672 tetrahedra. The initial mesh average quality is 2.28 with a worst quality of 1 314. Curving the mesh without optimization provides an invalid configuration with 4 invalid elements. Optimization in post and preprocessing improve average quality to 1.4 and worst quality to 1 777. Note that the number of nodes and tetrahedra is changed to respectively 659 545 and 535 146.

In both cases, optimization ensures a valid curved mesh at the end that would not have been obtained without it. In the same way as in the previous example, the curvature is not propagated a lot in the volume as it is not visible after 2 or 3 layers (see Figure 5.8, line 3 and 4 right).

5.3.4 Dassault Falcon aircraft

This example is a notional Dassault Falcon aircraft, a business jet that is used by private persons, companies and governments (see Figure 5.9). For this example we consider a mesh (see Figure 5.9, line 2) with an initial number of 89 078 vertices, 599 539 nodes and 498 181 tetrahedra. The initial mesh average quality is 1.44 with a worst quality of 21. Optimization in post and preprocessing improve average quality to 1.42 and worst quality to 30.66. Note that the number of nodes and tetrahedra is changed to respectively 599 287 and 497 929. In the same way as in the previous examples, the curvature is not propagated a lot in the volume as it is not visible after 2 or 3 layers (see Figure 5.9, line 2 and 3 right).

5.4 A moving mesh technique for P^2 elements

5.4.1 Initial algorithm

In this section, a connectivity-change moving-mesh method inspired from [Alauzet 2014] for P^2 meshes is presented. In this case, the initial mesh is a P^2-mesh whose boundary has an initial displacement. Using a linear elasticity analogy, the resolution of the elasticity equation with high-order finite elements gives us a displacement for all the vertices and nodes in the volume. Then the mesh is moved to the new position. Note that unlike in the mesh curving case, the mesh has potentially curved elements. The use of quadrature formula is consequently required for the FE resolution on the curved elements (see Appendix B). The motion of the vertices can also be enhanced by using a local stiffness factor technique [Alauzet 2014]. This technique locally multiplies the tensor σ of linear elasticity equation by a factor proportional to $\mathcal{J}_K(\mathbf{x})^{-\chi}$. χ determines the degree by which smaller elements are rendered stiffer than larger ones. We use $\chi = 1$. Afterwards, connectivity changes are performed on the mesh to improve the *quality* of the elements. It is an efficient way to get rid of any shearing that occurs in the mesh. The high-order moving-mesh algorithm is summarized in Algorithm 23.

Algorithm 23: High-order moving mesh algorithm

1. Mesh deformation algorithm.
   (a) Compute body displacement from body translation and rotation data.
   (b) Solve linear elasticity equation with the FEM at the order of the mesh.
   (c) Perform high-order mesh optimization.
   (d) Check validity of the obtained displacement and restart it with a smaller body displacement if necessary/desired until the obtained displacement is valid.
2. Move the mesh.

5.4.2 Moving sphere

The first studied case is a moving sphere of radius 0.6 inside a large control volume (see Figure 5.10, left). At each iteration, the sphere is displaced of 0.08 in x direction. The initial P^2-mesh (see Figure 5.11, line 1) has an average quality of 1.27 and a worst quality of 2.5. We analyzed the quality of the mesh at three different positions of the moving sphere (see Figure 5.10, right) with and without connectivity-change: the initial position,

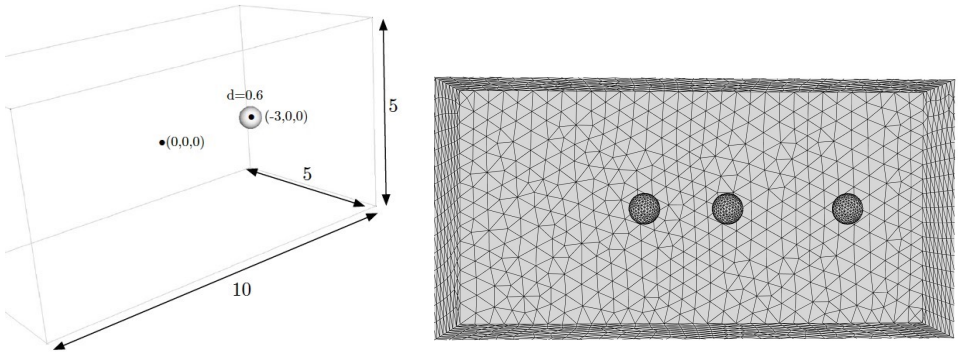

Figure 5.10 – Case of the moving sphere inside a P^2 mesh. Left, description of the geometrical domain. Right, illustration of the three considered positions of the moving sphere where the mesh quality is analyzed. The sphere is moving from right to left.

the position after 30 iterations (displacement of 4 radii) and the position after 50 iterations (displacement of 6.7 radii).

When no connectivity-change is done after each iteration, shearing appears in the mesh which constraints the displacement. The high-order linear elasticity resolution gives to the elements a curvature that fits to the displacement of the sphere in order to move as far as possible the object when the mesh connectivity is fixed. Indeed, in front of the sphere, the deformed elements fit to the shape of the sphere, whereas in the wake, the curvature of elements is made so that the shearing is reduced. This is a good point but the mesh quality decreases drastically (see Fig 5.11, line 2 and 3 left): after 30 iterations, average quality is 1.64 and worst quality is 5.7 and after 50 iterations, 84 elements are invalid. On the contrary, when mesh quality-based optimization operators are considered with the moving mesh algorithm at each iteration, the average and the worst quality do not change a lot (see Fig 5.11, line 2 and 3 right): after 30 iterations, average quality is 1.43 and worst quality is 3.1 and after 50 iterations, average quality is 1.48 and worst quality is 3.5. To get an idea of what happens at every iteration, the evolution of the worst quality of the mesh all along the moving mesh process is shown in Fig 5.12.

5.4.3 Moving and rotating falcon

The second studied case is a moving falcon aircraft (see Figure 5.9 for the geometry) of size whose bounding box is $x : [0,\ 18]\ y : [-9,\ 9]\ z : [-1,\ 4.75]$. It is also inside a large control volume ($x : [-129.6,\ 30]\ y : [-50,\ 50]\ z : [-50,\ 50]$). The imposed motion is both a translation and a rotation. More precisely, this a translation of -1.5 in x coupled with a rotation of 1.8 degrees around Ox axis that is done at each iteration. The initial mesh has an average quality of 1.42 and a worst quality of 30.58. Again, three positions are considered to analyze the quality of the P^2-mesh: the initial position, the position after 25 iterations and the position after 50 iterations.

The observations are quite similar to the previous case but the importance of mesh optimization is more striking. Indeed, due to the presence of the rotation, the mesh becomes quickly invalid (see Figure 5.14). Otherwise, we can still say that the use of the linear elasticity allows the motion of the mesh to be good in the vicinity of the falcon, as it can be seen on the line 2 and 3 of Figure 5.13, on the left, where the mesh is still valid even if no optimization is used. However, the absence of both smoothing and swapping causes a lot of shearing in the front and in the wake of the aircraft. Also, the use of mesh

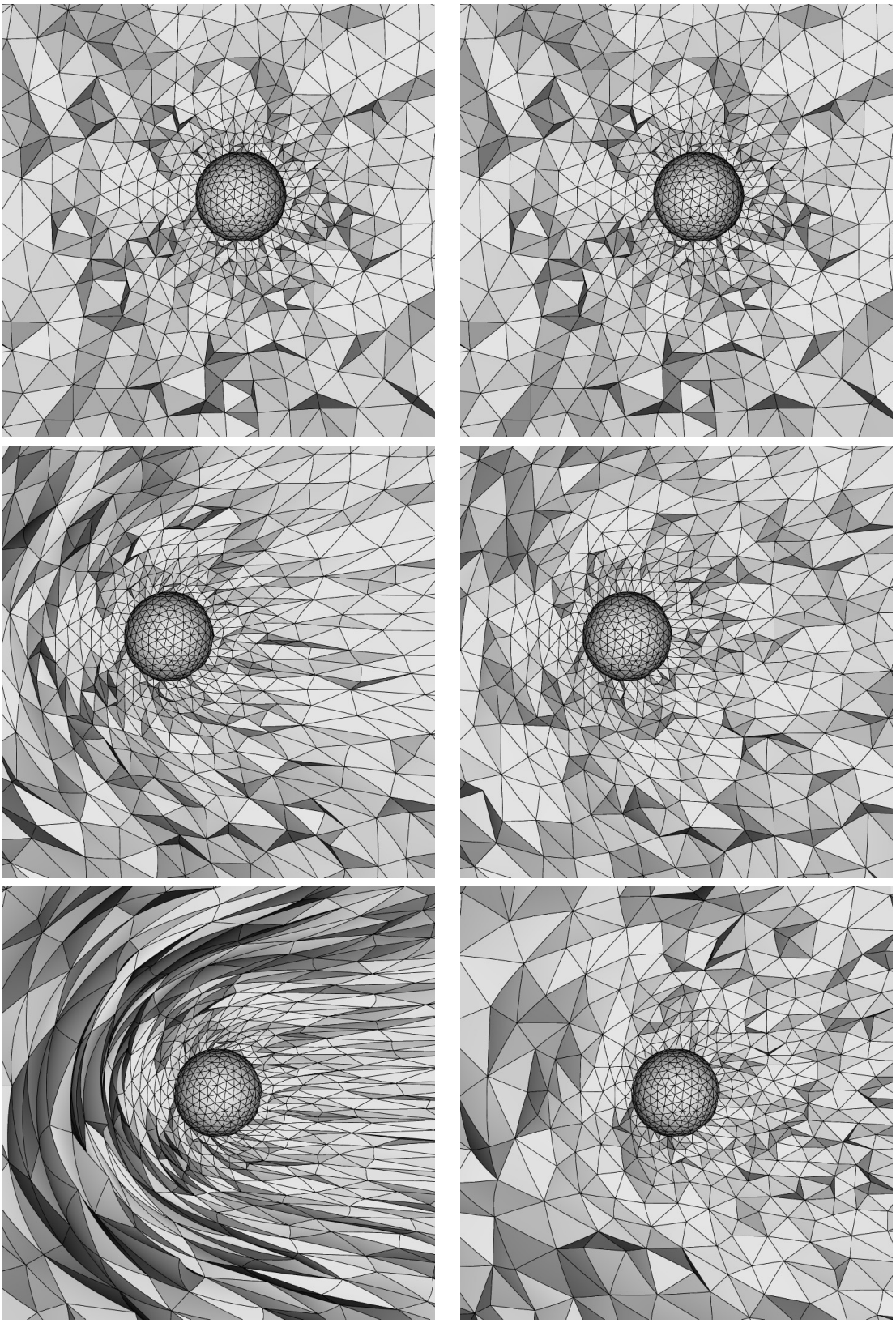

Figure 5.11 – Case of the moving sphere inside a P^2 mesh. From top to bottom, zoom in the vicinity of the sphere at respectively 0 radii, 4 radii and 6.7 radii of displacement. Left, without mesh optimization operators. Right, with mesh optimization operators.

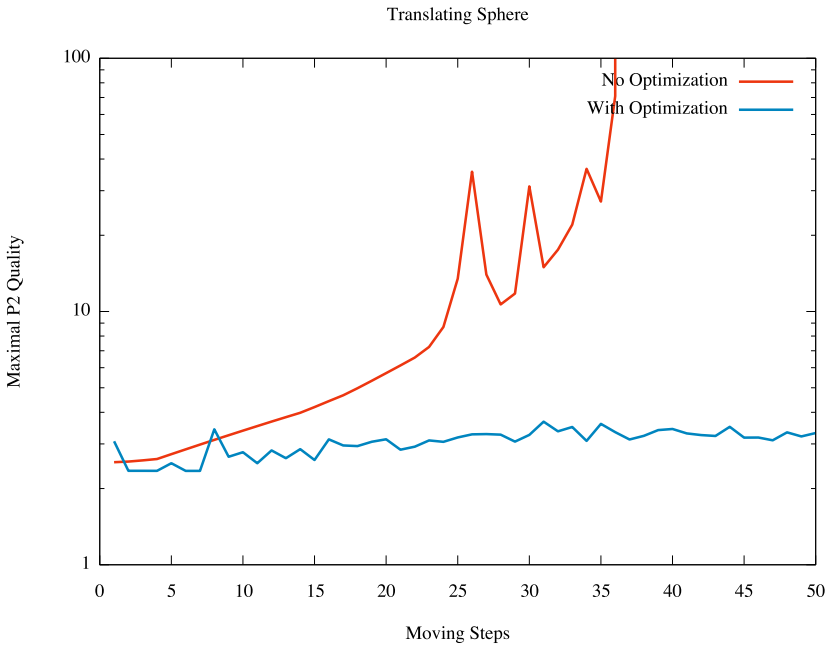

Figure 5.12 – Evolution of the worst P^2 quality of the mesh of the moving sphere all along the moving mesh process with and without optimization.

optimization allows the worst and average quality to remain relatively constant. After 25 iterations (line 2, right of Figure 5.13), the average quality is of 1.48 and the worst quality is of 26.57 and after 50 iterations (line 3, right of Figure 5.13) the average quality is of 1.51 and the worst quality is of 32.97. Finally, the global evolution of the quality all along the process is shown in Figure 5.14.

5.4.4 Improved algorithm

Algorithm 24: Improved High-order Moving Mesh Algorithm with Curved Trajectories

While $(t < T^{end})$

1. Compute body displacement from current body translation, rotation, speed and acceleration data for $[t, t + \Delta t/2]$ and then $[t, t + \Delta t]$. Solve elasticity and obtain $\mathbf{d}(t + \Delta t/2)$ and $[\mathbf{d}(t + \Delta t)$
2. Deduce inner vertex speed and acceleration for both displacement $\mathbf{d}(t + \Delta t)$ and $\mathbf{d}(t + \Delta t/2)$
3. If predicted mesh motion is invalid then $\Delta t = \Delta t/2^n$ and goto 1.
   Else $T^{els} = t + \Delta t$
4. While $(t < T^{els})$
   (a) Get moving mesh step δt thanks to CFL^{geom} and speed data.
   (b) Perform high-order mesh optimization
   (c) Move the mesh and update vertex speed and acceleration
   (d) $t = t + \delta t$

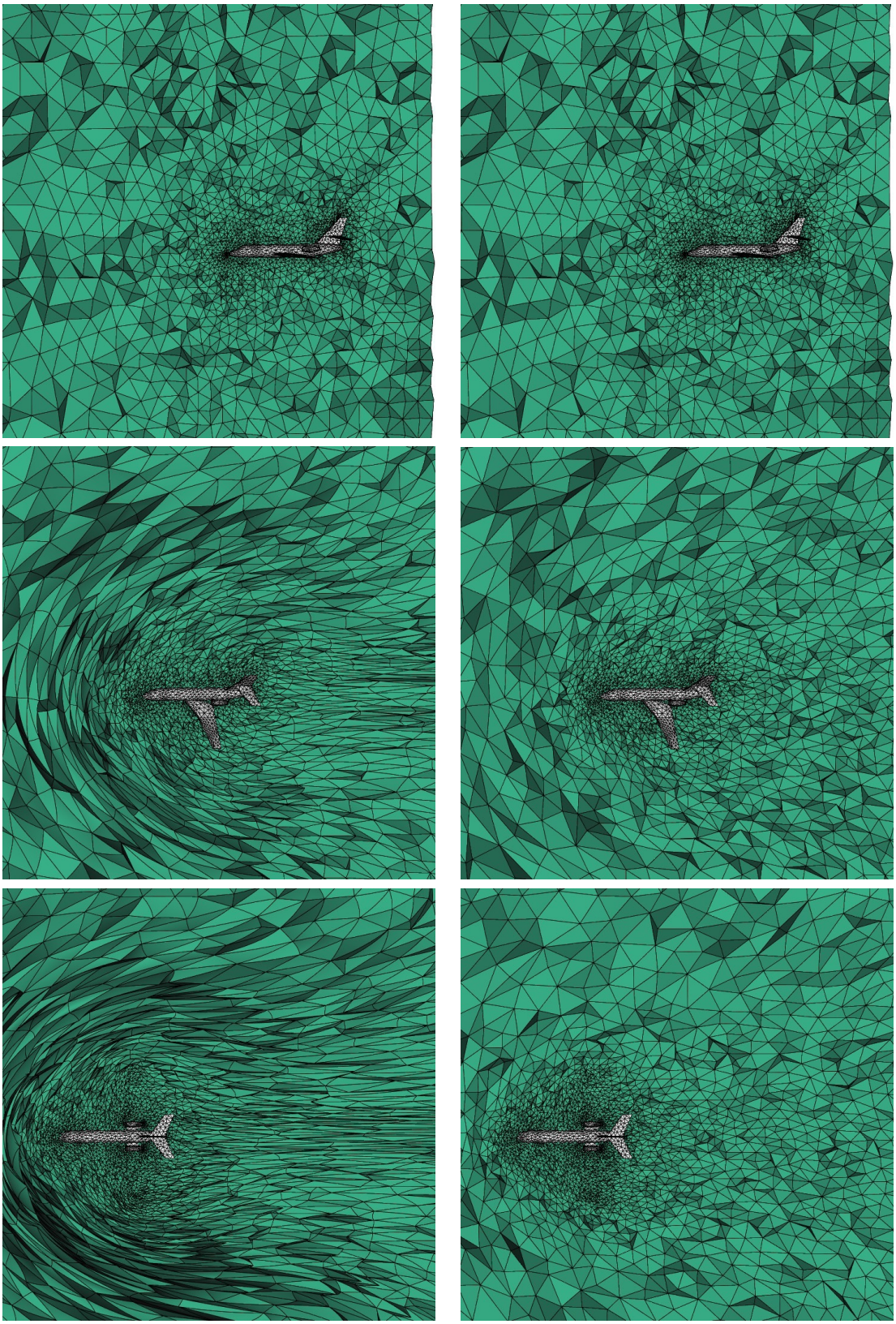

Figure 5.13 – Case of the translating-rotating falcon aircraft inside a P^2 mesh. From top to bottom, zoom in the vicinity of the falcon jet after 0, 25 and 50 iterations of moving-mesh process. Left, without mesh optimization operators. Right, with mesh optimization operators.

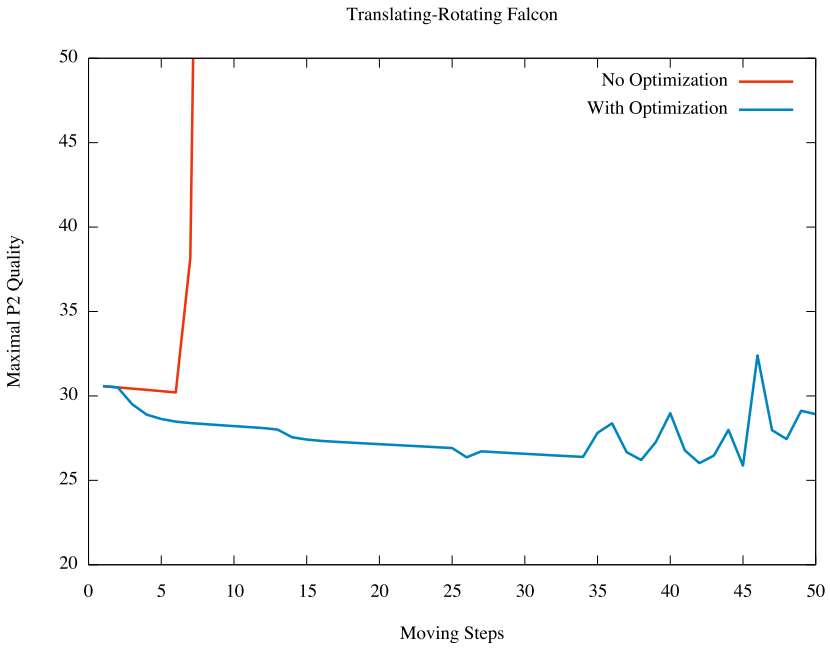

Figure 5.14 – Evolution of the worst P^2 quality of the mesh of the translating-rotating falcon aircraft all along the moving mesh process with and without optimization.

The initial **Algorithm** 23 can be improved using the same idea as in [Alauzet 2014], that is to say to compute a speed and an acceleration and therefore deduce a quadratic trajectory for all the degrees of freedom. This notably reduces the number of linear elasticity resolution, as the trajectory is performed by interlacing extrapolations based on speed and acceleration and linear elasticity resolutions. The process is summarized in **Algorithm** 24.

5.4.5 Ejection door

This improved algorithm is used to perform a more complex test case than the two previous ones. The problem is the door ejection of an over-pressurized aircraft cabin. This is a usual industrial benchmark for aircraft designers whose aim is to evaluate when the door hinge will yield under cabin pressure. Indeed, the trend for new aircrafts is to lower the cabin altitude (*e.g.*, higher cabin pressure). The difficulty is that the geometry is anisotropic which forces to deal with low quality elements.
The door movement is shown in Figure 5.16 and corresponds to physical time of 1.6s. For this test case, a total of 811 linear elasticity resolution have been performed and 5 519 moving steps have been done. The worst detected quality all along the process is of 20.097 and a total number of 631 764 swaps have been done. In Figure 5.15, the evolution of the average and worst quality is shown. It can be seen that the average quality is maintained relatively constant all along the process while the worst quality is slightly decreased all along the process. The latter is due to the initial configuration where badly shaped elements are located between the door and the body of the aircraft.

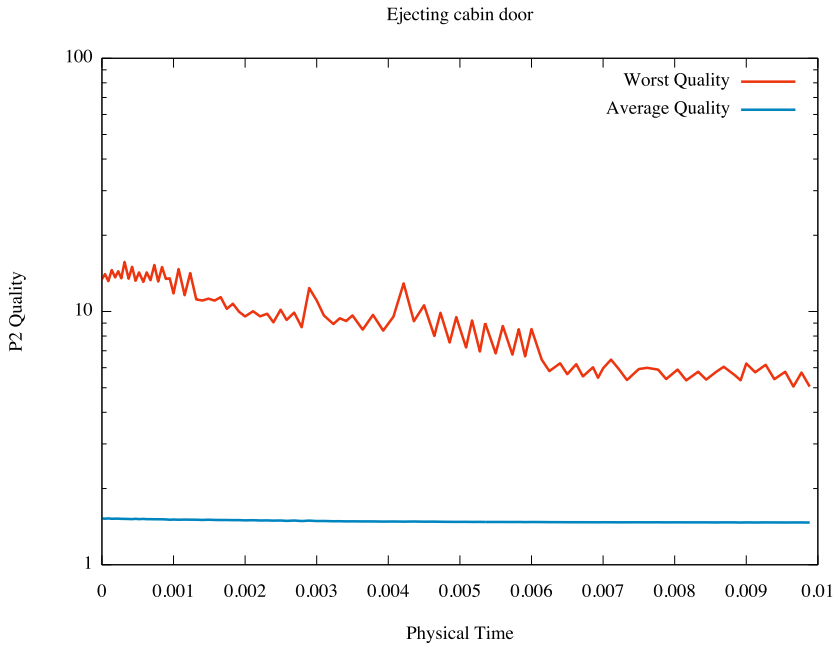

Figure 5.15 – Evolution of the worst P^2 quality of the mesh of the door-ejection case all along the moving mesh process with and without optimization.

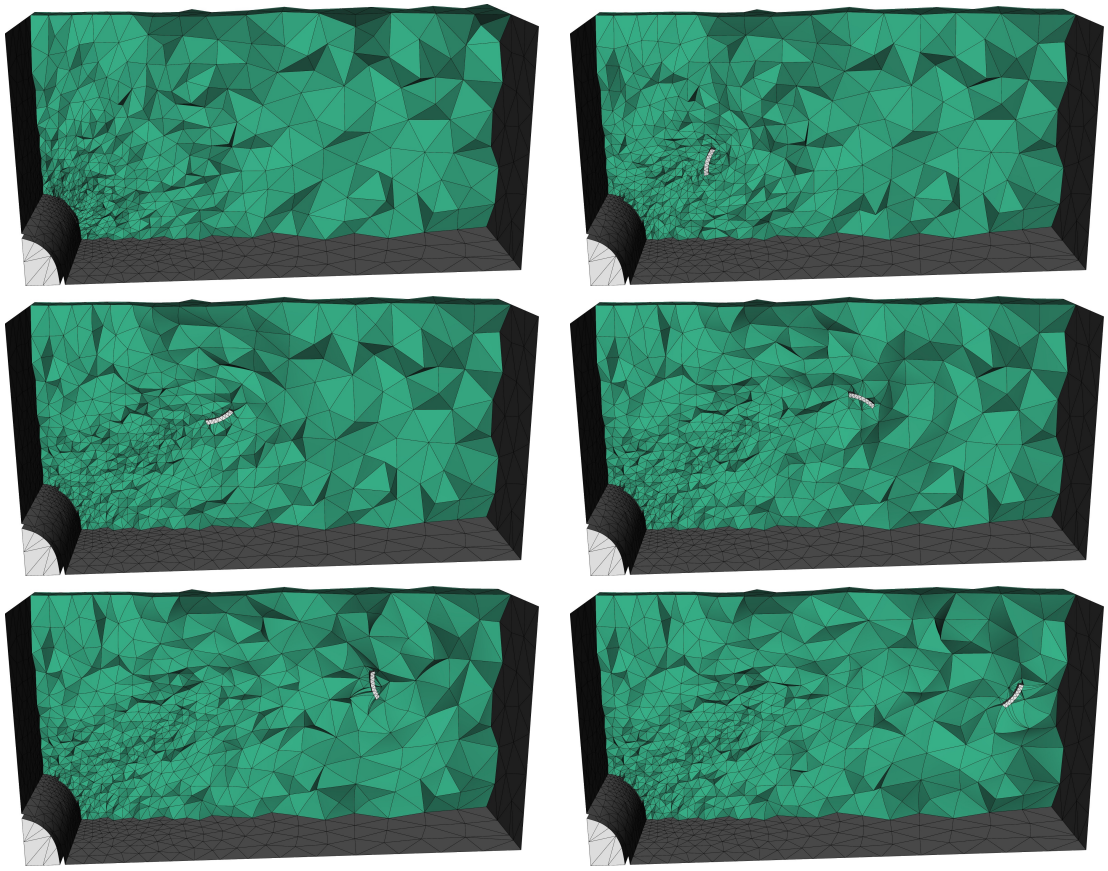

Figure 5.16 – Ejected cabin door test case. Snapshots of the displacement at time 0, 0.32, 0.64, 0.96, 1.28 and 1.6 from left to right and top to bottom

5.5 Boundary layer mesh generation by P^2 closed advancing-layer method

A closed advancing-layer boundary layer (BL) mesh generation approach is used for the present work. It is based on the existing advancing-layer method described in [Alauzet and Marcum 2015, Alauzet et al. 2017] as it has proven to be a robust procedure that consistently produces high-quality linear BL meshes for complex configurations. This approach starts with a fully resolved volume mesh and inserts the BL mesh one or more layers at a time using a mesh deformation process. This section proposes its extension for P^2 meshes. It relies on the P^2 connectivity-change moving mesh method presented in Section 5.4 to inflate the boundary layer (BL) inside the volume mesh [Feuillet et al. 2018]. This moving mesh strategy has proved to be very efficient to handle large geometry displacement and shear motion inside the mesh while preserving the mesh quality [Barral and Alauzet 2014, Barral et al. 2014, Barral et al. 2015]. The main advantages of this strategy are the following

- it is robust because it starts from an already existing high-order mesh and it always produces a valid result
- it automatically and naturally handles BL front collision as BL is inflated inside an existing volume mesh
- it produces automatically high-quality meshes with smooth size blending between colliding BL fronts and in concave regions.

Figures 5.17, 5.18 and 5.19 present the resulting P^1 mesh obtained with the closed advancing-layer BL mesh generation method on the 3rd AIAA CFD High Lift Prediction Workshop geometry. The surface mesh has been provided by Boeing and the initial volume mesh has been generated using `AFLR` [Marcum 1998]. The BL parameters follows the meshing guidelines: the initial BL spacing is set to 0.00117, the 5 first layers keep a constant size, the initial growth rate is 1, the growth rate acceleration is 1.025 and the maximal growth rate is 1.2. A maximum of 54 layers have been generated leading to a final mesh composed of $25,633,441$ vertices, $152,036,662$ tetrahedra and $981,238$ triangles. The first stops occur at layer 22 and less than one percent of the BL vertices are stopped until layer 36. In Figure 5.18, we observe that all the BL stops occur in the regions between the slat and the wing, between the flap and the wing and at the wing-body junction. The BL growths up to layer 54 elsewhere. This case also points out the capability of this method to maintain quality elements in BL collision regions. For example the regions between the wing and both slat and flap are shown in Figure 5.19. We notice a very smooth size variation between the BL mesh and the deformed volume mesh.

The following sections present the main points of the P^2 BL mesh generation closed advancing-layer method.

5.5.1 Overall closed advancing-layer P^2 boundary layer mesh generation

The overall process to generate P^2 meshes with boundary layers follows the following stages:

1. Generate an initial P^1 surface mesh for the given CAD geometry
2. Generate an initial (Euler) P^1 volume mesh using an advancing front [Marcum 2000] or a Delaunay-based [George et al. 1991a] mesh generation method. This mesh has no boundary layer mesh.
3. Generate a (Euler) P^2 surface and volume mesh using the method of Section 5.3. This mesh has no boundary layer mesh.

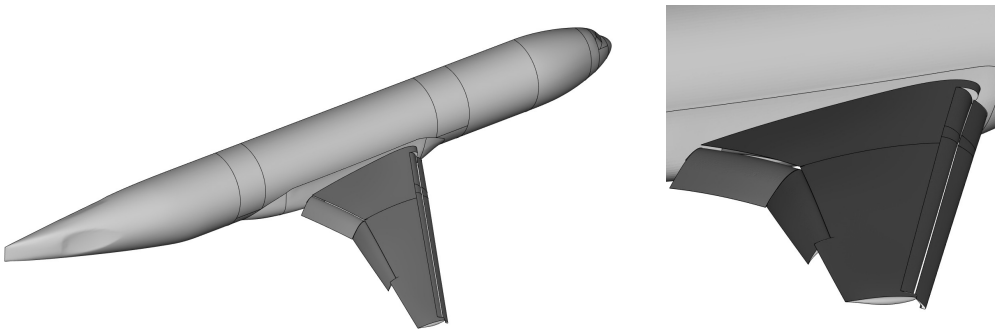

Figure 5.17 – 3rd AIAA CFD High Lift Prediction Workshop geometry.

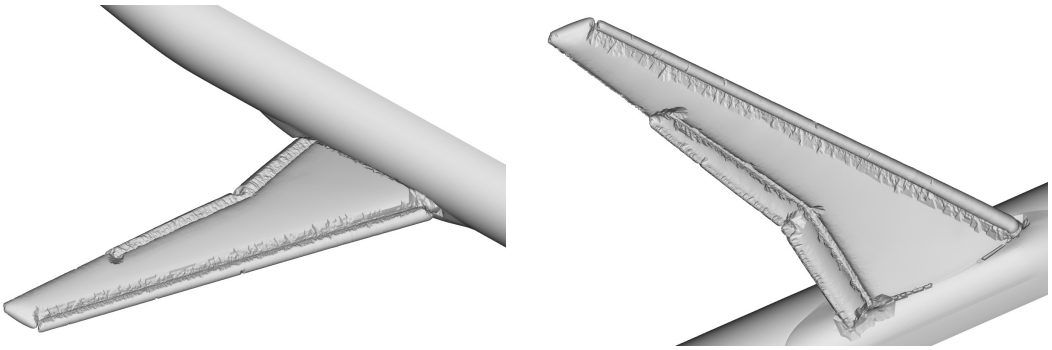

Figure 5.18 – 3rd AIAA CFD High Lift Prediction Workshop final BL outer skin mesh. Left, view of the top of the wing and, right, view of the bottom of the wing.

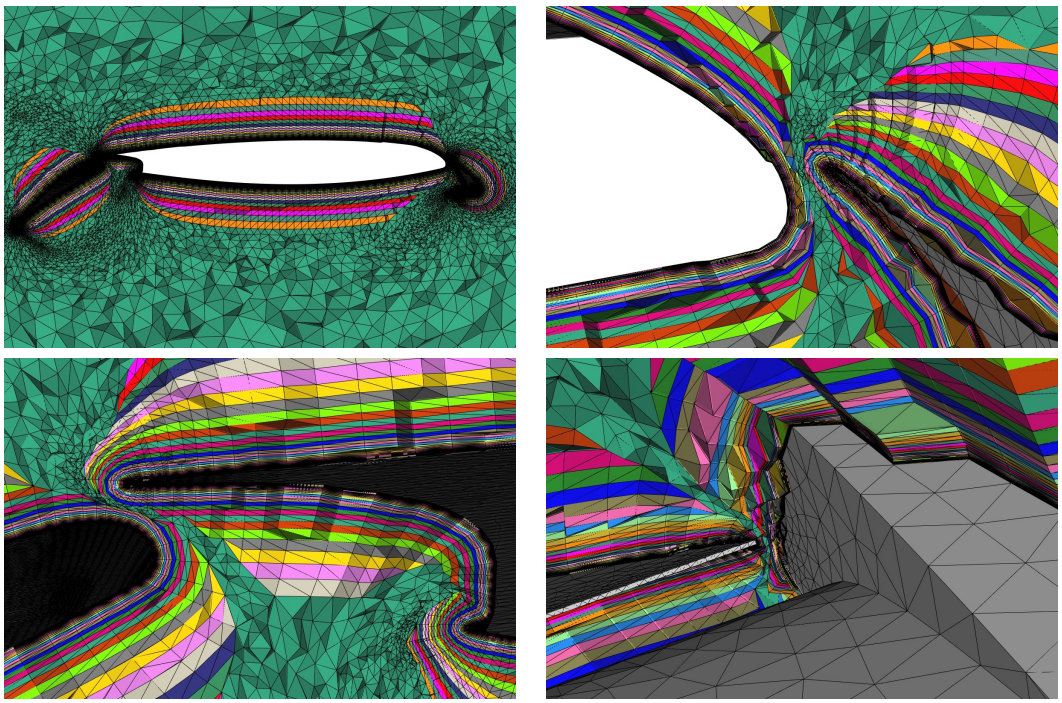

Figure 5.19 – 3rd AIAA CFD High Lift Prediction Workshop final BL mesh. Top left, a cut plane through the volume showing the BL mesh around the multi-elements wing. Top right, a close-up view in the gap between the slat and the wing. Bottom left, a close-up view in the gap between the flap and the wing. Bottom right, a close-up view on the wing in the region where the slat ends.

4. Generate directly the P^2 boundary layer mesh inside the given P^2 Euler mesh by inflating the BL mesh inside the domain using the connectivity-change high-order moving mesh method presented in Section 5.4.

The last step of the overall process is detailed in the following discussion.

5.5.2 P^2 closed advancing-layer method

The P^2 advancing-layer method is an extension of the existing linear method [Alauzet and Marcum 2015, Alauzet et al. 2017]. This algorithm is based on that described in [Marcum 1998]. In it, each BL vertex and BL node (high-order case only) is advanced one layer at a time. The primary steps in the advancing-normal/layer procedure are presented in **Algorithm** 25.

Algorithm 25: Closed advancing-layer algorithm

1. Determine a normal vector at each active BL point from the geometry of the inflated boundary surface. For high-order elements this includes the element-edge and element-face nodes along with the vertices.
2. Generate proposed BL points one layer at a time. New points are created along the normal vector with the normal spacing determined using geometric growth from the boundary surface.
3. Proposed points are rejected if any of the elements that include it are invalid or fail a quality check.
4. Check element aspect ratio. As the mesh advances and the normal spacing increases the element aspect ratio will eventually be isotropic. Boundary-layer advancement is terminated locally when the aspect ratio on the next layer would be greater than a preset factor (≤ 1).
5. Send displacement vectors for the proposed BL points to the outer region mesh deformation process (see Algorithm 26). Note that this algorithm may reject one or more of the proposed BL points.
6. Boundary-layer advancement is terminated locally if the proposed BL point was rejected by any of the preceding steps.
7. If the proposed BL points are accepted then the BL interface is advanced.
8. Continue if there are still active BL points and the number of layers generated does not exceed a preset maximum.

With P^2 elements, the treatment of vertices and element-edge (or mid-side) points require special procedures. Two steps are required to advance one layer of elements. First the mid-side points that are advanced at a half-step in the normal direction must be inserted and then the vertices and mid-side points that are advanced a full-step. There is also the issue of whether to create the element edge parallel to the BL normal vector as a straight or curved edge. We use a straight edge as it provides better quality. Also, the rejection of any vertex or element-edge point implies the rejection of all associated element-edge points and elements. This includes rejection of BL points advanced a full-step forcing the rejection of previously accepted BL points advanced only a half-step. The impact of this issue and need for two steps can be minimized by using only a full-step advancement and then optimizing the location of the half-step points. In this case, the rejection of any full-step BL points automatically rejects the associated half-step ones. And, if the location of

the half-step points can not be optimized, then they force rejection of associated full-step points.

5.5.3 BL P^2 mesh deformation method

The preceding advancing-layer BL method provides a displacement vector for the mesh deformation process at each vertex and node on the BL interface surface. Given this displacement vector the volume mesh is deformed using the high-order connectivity change moving mesh algorithm to insert the layer. The overall algorithm is presented in Algorithm 26. More details can be found in [Alauzet and Marcum 2015, Alauzet et al. 2017] for the linear case. The cost of the mesh deformation can be reduced seeing that it is not efficient to solve the mesh deformation at each layer insertion. Especially, at the beginning when the boundary layer inflation is quite small. In our approach, the mesh deformation is solved each 10 layers or when the sum of the layer normal sizes, since the last mesh deformation solution, is larger than twice of the surface mean size. If the mesh deformation is avoided, then inner vertices displacements are updated using the boundary layer growth rate:

$$\mathbf{d}^{i_{lay}+1} = \beta^{i_{lay}}\, \mathbf{d}^{i_{lay}} \quad \text{where } \beta^{i_{lay}} \text{ is the growth rate at the } i\text{th layer.}$$

A good restriction to be imposed on the mesh movement to limit the apparition of flat or inverted elements is that vertices and nodes cannot cross too many elements on a single move between two mesh optimizations. Therefore, a geometric parameter CFL^{geom} is introduced to control the number of stages used to perform the mesh displacement. If CFL^{geom} is greater than one, the mesh is authorized to cross more than one element in a single move. In practice, CFL^{geom} is usually set to 1. The moving geometric time step is given by:

$$\delta t = CFL^{geom} \max_{P_i} \frac{h(\mathbf{x}_i)}{\mathbf{v}(\mathbf{x}_i)}\,, \tag{5.2}$$

where $h(\mathbf{x}_i)$ is the smallest height of all the elements in the ball of vertex P_i and $\mathbf{v}(\mathbf{x}_i)$ is the velocity of vertex P_i.

5.5.4 Managing the boundary layer progression

Checking mesh validity

At the end of each moving step, we check if the mesh is still valid and meets all quality requirements. The validity requires all mesh elements to have a positive volume after the mesh deformation. Quality requirements are thresholds that are set depending on the initial mesh or actual mesh quality. They govern when the BL region will stop inflating: lower bound stops the BL sooner while larger bound allows the BL to propagate farther. For isotropic unstructured meshes, quality functions cannot directly be used to stop the BL progression because the BL inflation may lead to anisotropic elements. These anisotropic elements are considered bad for the quality function but, in fact, they are good for the transition from the BL to the outer region. To solve this issue, quality criteria check for an element are only activated if the height of the element is below one fourth of the current layer normal size. In that case, we compare its height h and its quality Q of with respect to its initial height h_{ini} and initial quality Q_{ini} (*i.e.,* evaluated before inflating the BL) weighted by a prescribed coefficient, both quality criteria are violated if:

$$h \leq \alpha h_{ini} \quad \text{and} \quad Q \geq \beta Q_{ini} \quad \text{with} \quad \alpha < 1 \quad \text{and} \quad \beta > 1.$$

Algorithm 26: P^2 closed advancing-layer BL mesh generation

For $i_{lay} = 1, .., n_{lay}$

1. Create layer i_{lay}: For each active point propose its optimal position using the advancing-layer method given in Algorithm 25
2. If (mesh deformation criteria) Then
   $\mathbf{d}_{|\partial\Omega_h}$ = Get boundary vertex and node displacement from inflating boundary layer
   $\mathbf{d}$ = Get inner vertex and node displacement by solving the elasticity system $\left(\mathbf{d}_{|\partial\Omega_h}\right)$
   Else
   $\mathbf{d} = \beta\ \mathbf{d}$ increment vertex and node displacement by the growth rate
   EndIf
3. Set $t = 0$, $T = 1$ and vertex speed $\mathbf{v} = \mathbf{d}$
4. While $(t < T)$
   (a) δt = Get moving mesh time step $\left(\mathcal{H}^k, \mathbf{v}, CFL^{geom}\right)$
   (b) $\mathcal{H}^k$ = Connectivity optimization $\left(\mathcal{H}^k, Q^{swap}_{target}\right)$
   (c) $\mathbf{v}^{opt}$ = Vertex smoothing $\left(\mathcal{H}^k, Q^{smoothing}_{target}, Q_{max}\right)$
   (d) $\mathbf{v}^{opt}$ = Node smoothing $\left(\mathcal{H}^k, Q^{smoothing}_{target}, Q_{max}\right)$
   (e) $\mathcal{H}^{k+1}$ = Move the mesh $\left(\mathcal{H}^k, \delta t, \mathbf{v}, \mathbf{v}^{opt}\right)$
   (f) Check mesh validity:
   If (element quality threshold is exceeded) Then
   Cancel element's vertices and nodes displacements
   Freeze element's vertices and nodes : $\mathbf{v} = 0$
   EndIf
   (g) $t = t + \delta t$
   EndWhile
5. Move back vertices and nodes that have moved less than a threshold percentage of the layer size

EndFor

In this work, we use $\alpha = 0.25$ and $\beta = 3$. This criterion ensures a smooth transition between colliding layers.

If the validity or the quality requirement is not reached for an element at the end of a moving step, then displacements of the element's vertices are canceled ($\mathbf{v} = 0$) and the element's vertices positions are reset to their initial positions at the beginning of this moving step. This canceling action changes the neighboring elements quality, thus this procedure is iterated until no more vertices are canceled. At worst, the displacement of all the vertices is canceled and the mesh is returned to the previous stage position where it was valid. In the case of P^2 meshes, this action can lead to the suppression of several points. Indeed, if an edge corresponding to a node displacement is totally collapsed, all it is inner nodes (*e.g* the mid-layer nodes) should be deleted as well.

Checking boundary layer validity

As some vertices may be frozen at each moving step, some faces may have only a part of its three vertices continuing growing. In some tricky situation, this may lead to

an invalid element in the boundary layer which has not been previously detected by the closed advancing-layer algorithm. Indeed, the decomposition of the prism into tetrahedra changed. Thus, if a vertex or a node of a BL face has been frozen, then the BL elements validity should be checked. If an invalid element is going to be created in the BL, then all the vertices and nodes of that face are marked frozen.

Finalizing the BL position

At the end of the mesh movement, the final position at the current layer of each BL vertex and node is compared with its targeted final position. Vertices and nodes are categorized into two groups:

- If the total displacement is greater than 60% of the expected displacement: this BL vertex or node remains active for progressing
- If the total displacement is less or equal to 60% of the expected displacement: this BL vertex or node advancement is terminated and we seek for an optimal final position.

The threshold 60% has been chosen because we limit the maximum number of moving step for each layer to five, and it is appropriate in that case.

If a BL vertex or node advancement is terminated, then we search for an optimal final position. First, we try to move it back to its position at the end of the previous layer, *i.e.,* its initial position at the current layer. Most of the time, this move back is possible. However, again in complicated situations, moving back the vertex or the node may significantly degrade the mesh quality or possibly create invalid elements. In that case, we analyze the mesh configuration when the vertex or the node is moved by 0%, 40% and 60% of its expected displacement, and the best one is selected. In P^2, the position of the mid-layer nodes is adjusted accordingly to the shrink of the expected displacement. Also, when a high-order boundary node has an expected displacement which is reduced, the displacement of the neighboring vertices is reduced as well so that it does not distort that much the inflation of the boundary mesh.

5.5.5 Numerical Examples

The first results obtained on simple geometries are presented to assess the proposed strategy.

Spheres

The first example is a domain represented by two spheres, the inner sphere is of radius 1 and the outer one of radius 3, see Figure 5.20 (left). The initial domain is composed of 12 517 vertices and nodes, 1 290 P^2-triangles and 8 394 P^2-tetrahedra. We are growing BL meshes on each sphere to check the behavior of the moving mesh method when two BL meshes are colliding. The parameters are: the initial BL spacing is set to 0.0001, the growth rate is 1.2, and 21 layers are generated leading to a final mesh composed of 123 629 vertices and nodes, 89 634 P^2-tetrahedra and 1 290 P^2-triangles. This viscous P^2-mesh is displayed in Figure 5.20 (right). Close-up views of the boundary layer are shown in Figure 5.21. We observe that the proposed methodology is able to manage nicely the BL mesh collision with a nice transition between the BL mesh region and the outer mesh region. The curvature of the domain is properly accounted in the BL mesh and the elements of the initial mesh are curved to fit the shape of the BL.

Cylinder

The second example is a cylinder of height 2 and radius 0.2 included in a spherical domain of radius 10, see Figure 5.22 (left). The initial domain is composed of 113 695 vertices and nodes, 9 598 P^2-triangles and 77 894 P^2-tetrahedra. The parameters are: the initial BL spacing is set to 0.0005, the growth rate is 1.12, and 20 layers are generated leading to a final mesh composed of 900 895 vertices and nodes, 653 452 P^2-tetrahedra and 9 598 P^2-triangles. This viscous P^2-mesh is displayed in Figure 5.22 (right). Close-up views of the boundary layer are shown in Figure 5.23. This simple test case is interesting because it has a curved part and a straight part. We note that on the straight part, the BL mesher behaves like in the P^1-case which is consistent. It also means that no artificial curvature is created. In the curved part, the curvature is transferred properly through the layers. The elements on the last layer seem to be straight but they are a little bit curved, see Figure 5.23 (right). We also observe that the ridge of the cylinder is correctly managed by the process, so does the normal smoothing, see Figure 5.23 (left).

Rocket

The third example is a little bit more complex. It is a notional rocket model. This geometry is interesting because it has at the same time spherical like curvature on the nose, cylindrical like curvature on the body, multiple circular convex and concave ridges, and a sharp circular trailing edge, see Figure 5.24. The rocket length is 7 and it has a maximal diameter of 2.5. The initial domain is composed of 227 258 vertices and nodes, 10 850 P^2-triangles and 162 275 P^2-tetrahedra. The parameters are: the initial BL spacing is set to 0.00004, the growth rate is 1.15, and 20 layers are generated leading to a final mesh composed of 1 075 018 vertices and nodes, 781 632 P^2-tetrahedra and 10 850 P^2-triangles. Figure 5.25 displays close-up views fo the boundary layer mesh at geometric singularities. The top left pictures shows the management of the circular trailing edges, and the other pictures the management of circular convex and concave ridges. We note again that the whole algorithm is behaving properly on these geometric singularities.

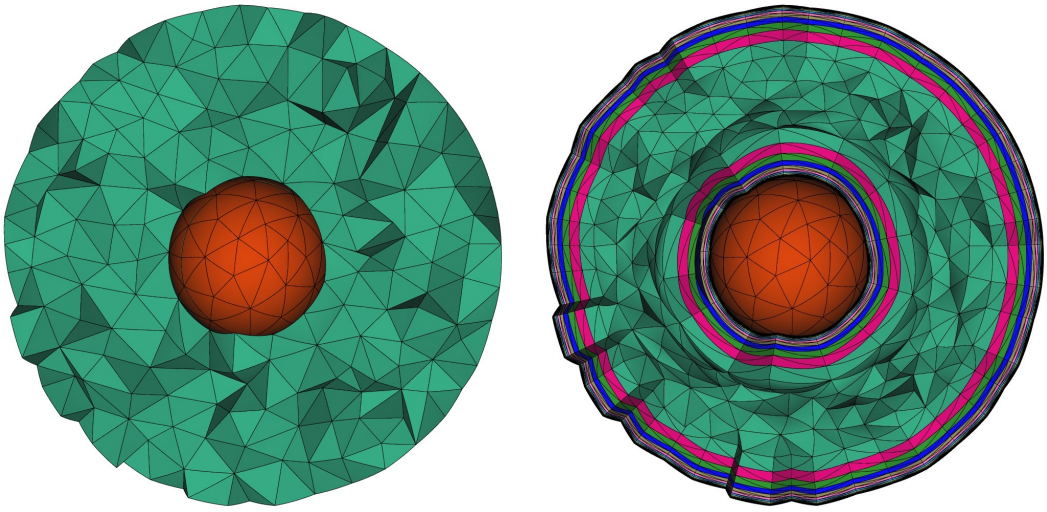

Figure 5.20 – Spheres case. Left, initial P^2-mesh and, right, final P^2-mesh after the insertion of the boundary layer.

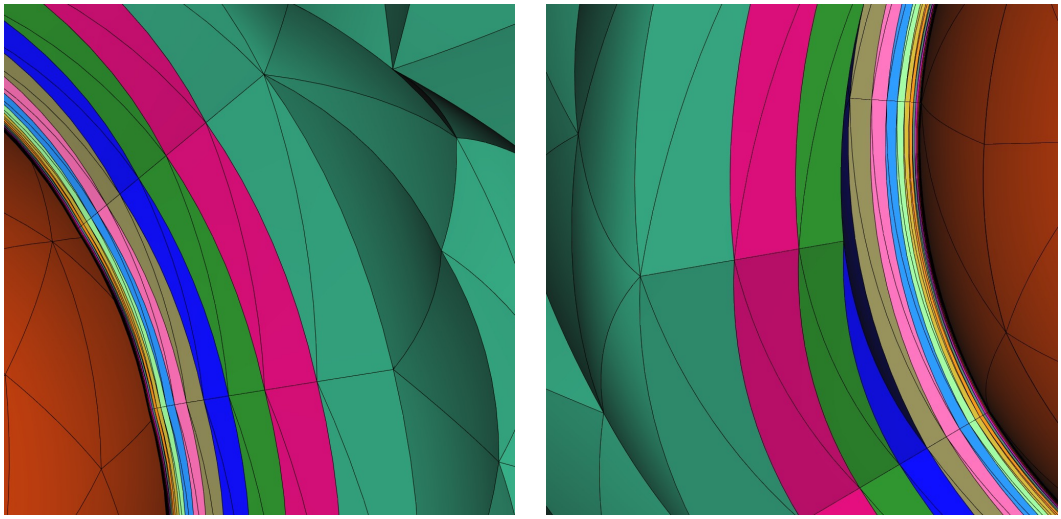

Figure 5.21 – Spheres case. Close-up views in the boundary layer region.

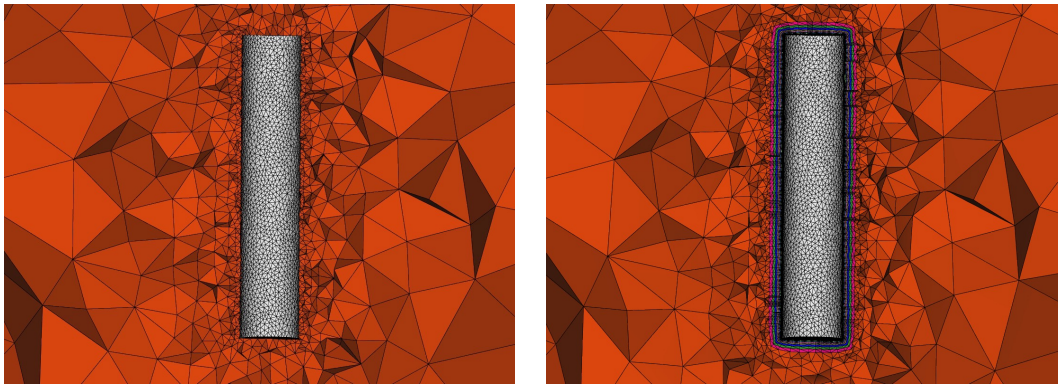

Figure 5.22 – Cylinder case. Left, initial P^2-mesh and, right, final P^2-mesh after the insertion of the boundary layer.

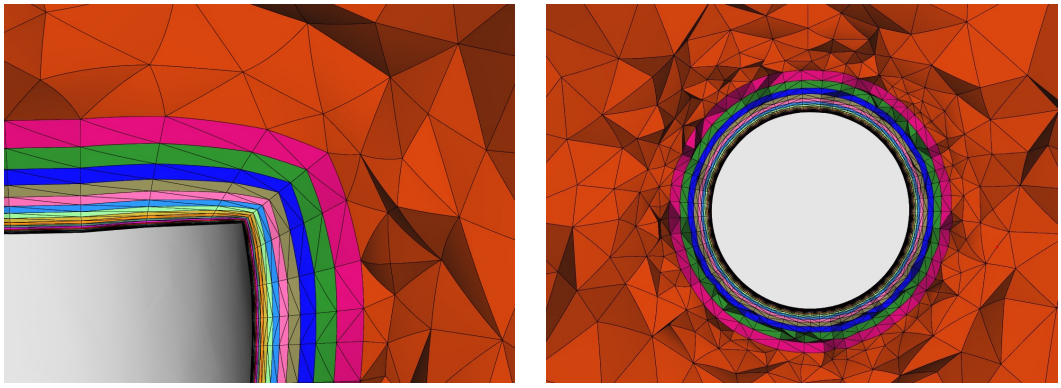

Figure 5.23 – Cylinder case. Close-up views in the boundary layer region.

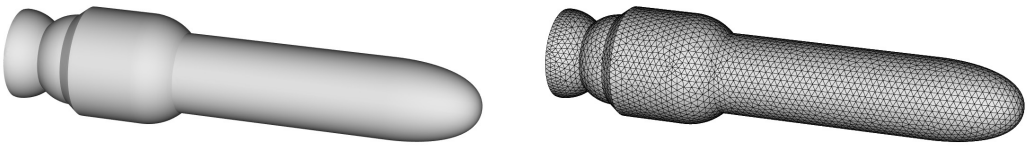

Figure 5.24 – Rocket case. Left, rocket geometry, right, initial P^2-surface mesh.

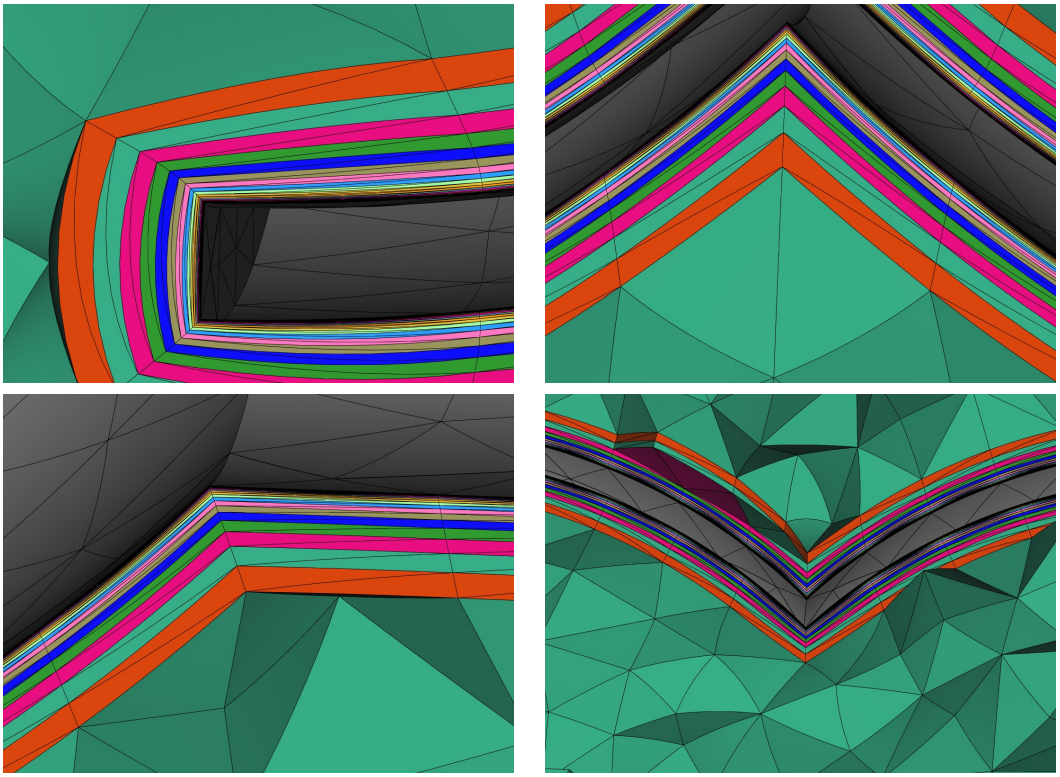

Figure 5.25 – Rocket case. Close-up views in the boundary layer region.

5.6 Conclusion

In this chapter, P^2 mesh quality-based optimization operators have been presented. These operators ensure a valid P^2 mesh generation starting from a P^1 mesh and enable to deal with connectivity-change P^2 moving-mesh methods. Note that all these developments are suitable for isotropic meshes but do not work that well on anisotropic meshes and do not work as is with boundary layer meshes. The moving-mesh method gives similar results in term of quality as in P^1 which is promising for future research. Then, an application as a closed advancing-layer method for generating P^2 boundary layer meshes has been presented. This method combines the high-order connectivity-change moving mesh method that works on the isotropic part of the mesh and a high-order closed advancing layer algorithm which proposes the degrees of freedom for the boundary layer mesh. The obtained results are promising and take into account high-order features. Future work will consider more realistic geometries. Note that all these developments have been made in several `Wolf` modules: `PkCurved` fo high-order mesh generation, `Spyder` for P^2 optimization, `MovMsh` for P^2 moving-mesh methods and `Bloom` for P^2 boundary-layer mesh generation and have been subject to several publications [Feuillet et al. 2018, Feuillet et al. 2019e, Feuillet et al. 2019d, Feuillet et al. 2019c]. Some perspectives can be driven regarding this work. In mesh optimization, isotropic degree two meshes were only the first step, further developments will be to generalize it to any higher-order meshes and to deal with anisotropy using metric fields. An extension to P^2-surface mesh optimization operators will also be considered. When it comes to the applications, several modifications can be done. First, the backtracking algorithm used for the boundary-layer mesh generation is at the time made in a P^1 sense, *e.g.* the nodes are moving back according to the straight edge but not according to the curved edge (note the most of the time the considered edges are straight). This should help to keep the validity in corner areas for instance. Then, the mesh deformation algorithm is based on a linear elasticity analogy and it does work fine for linear meshes. However, it has some drawbacks with higher-order meshes especially for the motion of the high-order nodes that curves high-order elements where it is not necessary. To cope with that, some alternatives are possible: we can decide to perform a P^1 linear elasticity motion and then perform P^2 node smoothing to enforce the curvature where needed. It could also be possible to change the deformation model and use some models like IDW [Barral et al. 2014]. Also, the development of a P^2-surface mesh optimization operators will allow the use of boundary-layer mesh on imprint surfaces. Finally, the whole process could be applied to generate hybrid boundary layer meshes including prisms, pyramids or hexahedra. To this end, the high-order validity process of such elements is already established as explained in Chapter 3.

Chapter 6

Visualization of high-order meshes and solutions

6.1 Introduction and state of the art

Classic visualization software like ParaView [KitWare Inc.], TecPlot [TecPlot Inc.], FieldView [Intelligent Light], Ensight [Ansys Inc.], Medit [Frey 2001], Vizir (OpenGL legacy based version) [Loseille et al. 2016], Gmsh [Geuzaine and Remacle 2009], ... historically rely on the display of linear triangles with linear solutions on it. More precisely, each element of the mesh is divided into a set of elementary triangles. At each vertex of the elementary triangle is attached a value and an associated color. The value and the color inside the triangle is then deduced by a linear interpolation inside the triangle. With the increase of high-order methods and high-order meshes, these softwares adapted their technology by using subdivision methods. If a mesh has high-order elements, these elements are subdivided into a set of linear triangles in order to approximate the shape of the high-order element [Vlachos et al. 2001]. Likewise, if a mesh has a high-order solution on it (Figure 6.1, first line), each element is subdivided into smaller linear triangles in order to approximate the rendering of the high-order solution on it. The subdivision process can be really expensive if it is done in a naive way (Figure 6.1, second line). For this reason, mesh adaptation procedures [Remacle et al. 2007, Maunoury et al. 2018, Maunoury 2019] are used (Figure 6.1, third line) to efficiently render high-order solutions and high-order elements using the standard linear rendering approaches. Even when optimized these approaches do have a huge RAM memory footprint as the subdivision is done on CPU in a preprocessing step. Also the adaptive subdivision process can be dependent on the palette (*i.e.* the range of values where the solution is studied) as the color only vary when the associated value is in this range. In this case, a change of palette inevitably imposes a new adaptation process. Finally, the use of a non conforming mesh adaptation can lead to a discontinuous rendering for a continuous solution (Figure 6.1, third line).
Other approaches are specifically devoted to high-order solutions and are based on ray casting [Nelson et al. 2011, Nelson et al. 2012, Peiró et al. 2015]. The idea is for a given pixel, to find exactly its color. To do so, for each pixel, rays are cast from the position of the screen in the physical space and their intersection with the scene determines the color for the pixel. If high-order features are taken into account, it determines the color exactly for this pixel. However, this method is based on two non-linear problems: the root-finding problem and the inversion of the geometrical mapping. These problems are really costly and do not compete with the interactivity of the standard linear rendering methods even when these are called with a subdivision process unless they are done conjointly on the GPU. However, synchronization between GPU and `OpenGL` buffer are non-trivial combination.
The proposed method intends to be a good compromise between both methods. It does guarantee pixel-exact rendering on linear elements (Figure 6.1, first line) without extra subdivision or ray casting and it keeps the interactivity of a classical method. Moreover,

the subdivision of the curved entities is done on the fly on GPU which leaves the RAM memory footprint at the size of the loaded mesh.

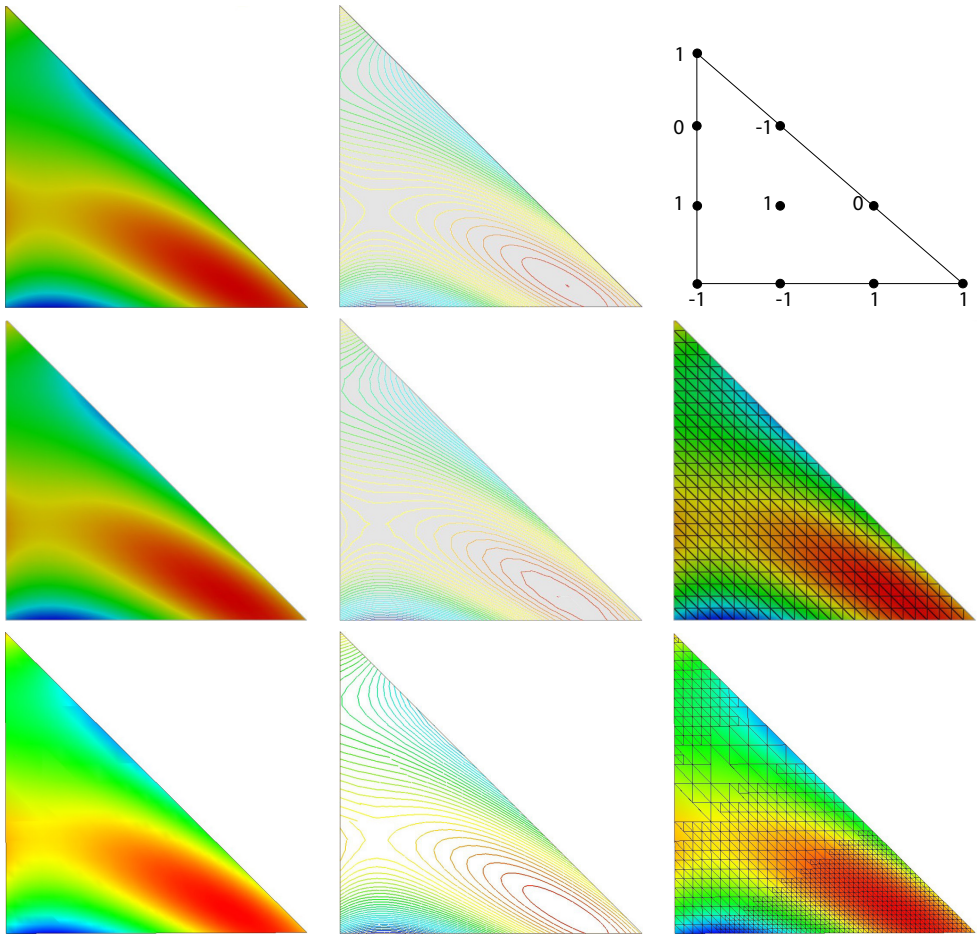

Figure 6.1 – From top to bottom, rendering of a P^3-solution: First Line: Definition of the solution on a triangle and its exact representation with its isovalues. Second line: representation of the solution linearly subdivided on a regular grid of 625 triangles with its isovalues. Third line: representation of the solution linearly subdivided on an adaptive grid of 1858 triangles (generated by Gmsh [Geuzaine and Remacle 2009]) with its isovalues.

6.2 Presentation of the `OpenGL` 4.0 graphic pipeline

In this section, we present the `OpenGL` 4.0 rendering pipeline. We emphasize how we use the flexibility for high-order elements and solution rendering. `OpenGL` is a graphic API widely used for graphic rendering. We interest ourselves to its release 4.0 or upper. The customizable part of its graphic-fixed pipeline is based on the use of shaders. They are GLSL (a C-like programming language) source code files that replace parts of the standard `OpenGL` pipeline. The main pros of redefining all the shader stages are:

- The memory footprint in RAM is limited to the size of the mesh,
- Addition (subdivided) entities are created on the graphic cards directly on the fly,
- Solutions are computed in the fragment shader so that new shape functions and interpolation schemes can be interchanged.

The OpenGL 4.0 pipeline can be customized with up to five different shader stages (see Figure 6.7). More precisely, the description of the shaders is the following:

- The **Vertex Shader** (VS), that is corresponding to the source code files xxx.vs. Its input variables are vertex attributes. This shader can also send other information down the pipeline using shader output variables. For instance, the vertex shader can compute its color and give it to the pipeline. The data corresponding to the vertex position must be transformed into clip coordinates and assigned to the output variable `gl_Position`. The vertex shader is executed (possibly in parallel) once for each vertex.
- The **Fragment Shader** (FS), that is corresponding to the source code files xxx.fs. Its input variables come from the graphic pipeline and is a transformation of other shader's outputs. The fragment shader determines the appropriate color for the pixel and sends it to the frame buffer using output variables. Also, among built-in variables, we can pass through our own variables which means that for a pixel, we can deduce (x, y, z), (u, v), or primitive ids. Using these variables, it is possible to display anything which is a function of these variables (see Figure 6.2). In our context, the Fragment Shader is used to perform the wireframe rendering as well as high-order solution and isoline rendering (see Section 6.4). For the storage of raw data (like high-order solutions), textures are used. The fragment shader is executed (possibly in parallel) once for each fragment (pixel) of the polygonal object being rendered.

These two shaders can be enough to define a customization of the graphic pipeline. In this case, between the two shaders, the vertices are assembled into primitives, clipping takes place and the viewport transformation is applied. The principle of the involved shaders is summed up in Section 6.3. In our case, we use this customization to display dots.

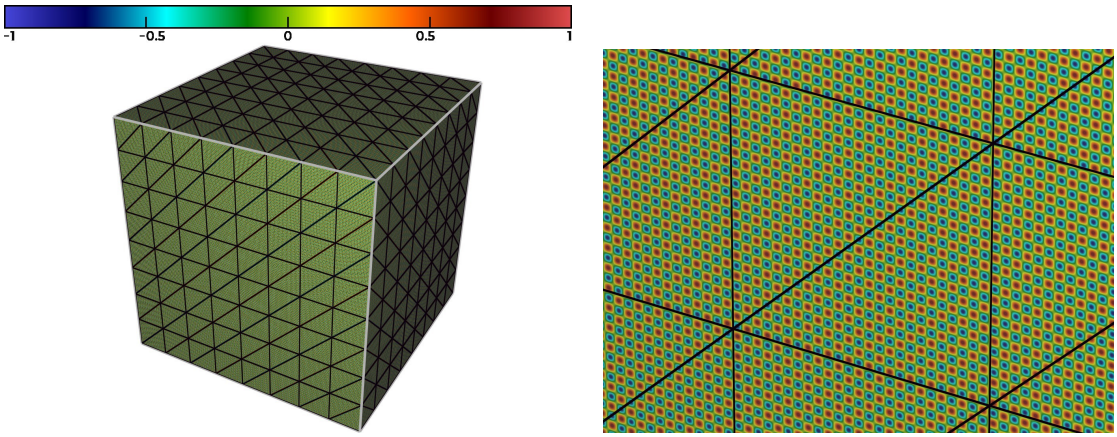

Figure 6.2 – Example of an exact rendering of the function $sin(100\pi x) + sin(100\pi y) + sin(100\pi z)$ (with its isolines on the right) on a coarse mesh of the unit cube. This is performed by using the Fragment Shader.

However, the OpenGL pipeline can be even more customized thanks to the following shaders:

- The **Geometry Shader** (GS), that is corresponding to the source code files xxx.gs. This shader cannot be used without the two previous shaders and is executed once for each primitive. It has access to all the input data of the vertices of the primitive that can be provided either by the vertex shader or by the tessellation evaluation shader (see below). As a consequence all variables are arrays. The GS can receive

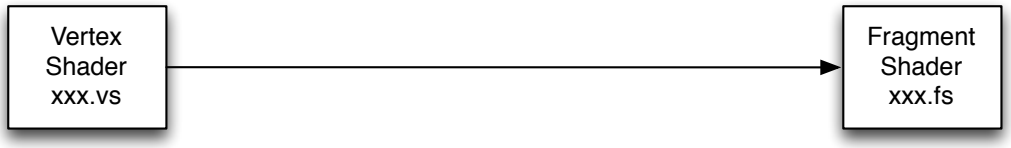

Figure 6.3 – Shaders used for a simple customization of the `OpenGL` graphic pipeline.

some primitives and can emit other primitives as long as only one type of primitive is in input or output. For instance, it can receive vertices and output triangles. The GS functionality is centered around two primitives: `EmitVertex` and `EndPrimitive`. To each vertex of the primitive, all useful data are linked with it and then the vertex is emitted thanks to the first primitive. Once all vertices of the primitive are processed, the second primitive is called. Also, every data attached to each vertex is by default linearly interpolated inside the primitive and sent to the fragment shader. In our context, the GS is used to define the flat shading (see Figure 6.4, left) and the shrinking of the linear elements (see Figure 6.4, middle). For all the elements, when a clipping plane is used, the operation is performed at this stage (see Figure 6.4, right) and finally, the GS is used to propagate the physical and barycentric coordinates ((x, y, z) and (u, v)) as well as normals. More details are given in Section 6.3.

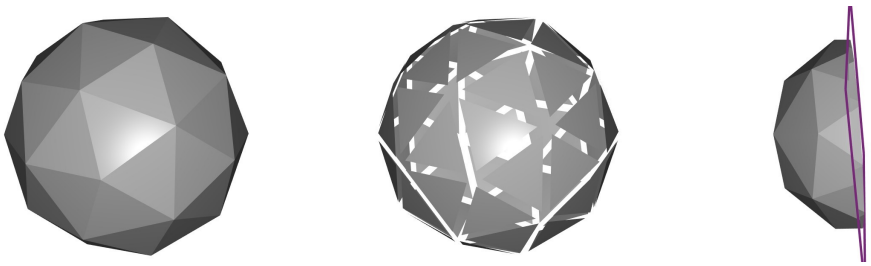

Figure 6.4 – Example of three different operations performed with the Geometry Shader.

The principle of the involved shaders is summed up in Figure 6.5. In our case, we use this customization to display simple and linear geometries such as edges and triangles of degree 1.

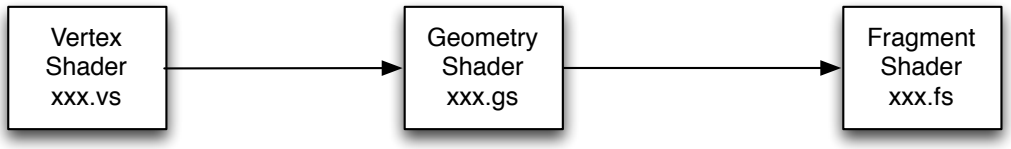

Figure 6.5 – Shaders used for the display of simple geometries using the `OpenGL` graphic pipeline.

To display more complex geometries such as high-order elements, a subdivision of

the element into linear is sometimes required. This is performed with the pipeline as it is shown thereafter.

- The **Tessellation Control Shader** (TCS) and the **Tessellation Evaluation Shader** (TES), that are respectively corresponding to the source code files xxx.tcs and xxx.tes. These shaders cannot be used without the three previous shaders. As soon as tessellation shaders are used, the only used primitives are patches. A patch primitive is a part of the geometry defined by the programmer. The number of vertices by patch is configurable as well as the interpretation of the geometry. For instance, the patch can be used as a set of control points that defined an interpolated curve or surface (Bézier curve for instance). More precisely, the TCS sets up the Tessellation Primitive Generator (TPG) by defining how the primitives should be generated by it and in particular in how many sub-entities the element should be divided (e.g. tessellated). The TCS is executed once for each entity and it can compute additional information and give it to the TES. In our context, the level of discretization (e.g. the granularity of the tessellation) given by the TCS is controlled with basic geometrical error estimates so that straight elements are not subdivided more than necessary. More details are given in Section 6.3.2. The TPG computes in the parameter space, the discretization of parameterized entities like quadrilaterals, triangles and isolines. Once the discretization is done, the entities are sent to the TES. The TES is executed once for each parameter space vertex. For a given vertex, it determines the position of the vertices in the physical space thanks to the coordinates of the parameter space and also other vertex-related data. All the data related to each sub-entity are then sent to the GS that processes them like any other entity. The principle of the involved shaders is summed up in Figure 6.6. For our purpose, we use this customization to display non-linear geometries such as quadrilaterals and any other high-order element.

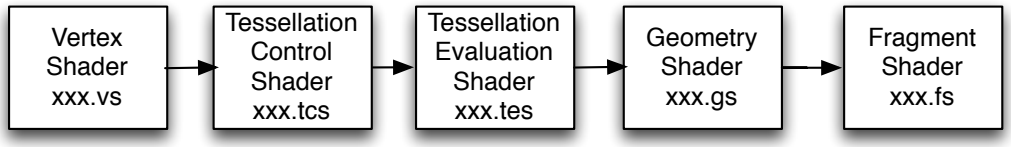

Figure 6.6 – Shaders used for the display of non-linear geometries using the `OpenGL` graphic pipeline.

The general principle of all the combinations of shaders is summarized in Figure 6.7.

6.3 High-order elements visualization

In this section, we show how we take advantage in practice of the `OpenGL` graphic pipeline to display geometrical finite elements of any kind. In particular, we describe some implementation details.

6.3.1 Visualization of surface elements

For simple shapes like P^1-triangles and edges, no tessellation is needed as these elements are planar. In this case, only three shaders are used: the Vertex Shader, the Geometry

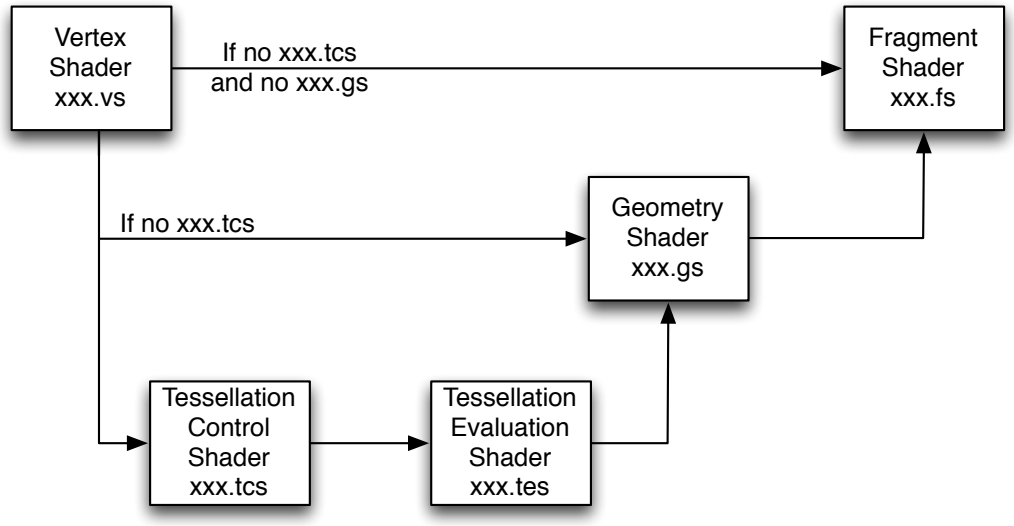

Figure 6.7 – Summary of all the possible combinations of shaders while using the `OpenGL` graphic pipeline.

Shader (which outputs triangles or lines) and the Fragment Shader. For more complex and potentially non-planar geometries (Q^1-quadrilateral, or any higher-order element), the two tessellation shaders are added to the three others. To process any geometry, it is first transformed into its Bézier form and then the obtained control points are sent to the graphic pipeline. On output, a reliable representation of the geometry is displayed on the screen. For the sake of clarity, let us have a look on the case of a P^3-triangle (see Figure 6.8). A P^3 triangle is generally defined by its 10 Lagrange points (here in barycentric notations): $(M_{ijk})_{i+j+k=3}$ (see Figure 6.8, left). The first step is to compute the control points [1]. It is done in a hierarchical fashion: the points lying on the edges of the surface first and the points lying on the inside of the surface. For the first edge, we have:

$$\begin{cases} P_{300} = M_{300}, \\ P_{120} = \dfrac{18M_{210} - 9M_{120} - 5M_{300} + 2M_{030}}{6}, \\ P_{210} = \dfrac{18M_{120} - 9M_{210} - 5M_{030} + 2M_{300}}{6}, \\ P_{030} = M_{030}, \end{cases}$$

and the same goes for the two other edges. The inner surface control point can be deduced:

$$P_{111} = \frac{9M_{111}}{2} - \frac{P_{300} + P_{030} + P_{003}}{6} - \frac{\sum_{j=1}^{2}\sum_{i=1}^{2-j} P_{ij0} + P_{i0j} + P_{0ij}}{2}.$$

Control points are preferred to Lagrange points as their use minimizes the number of operations which is a great property when it comes to GPU programming. For every P^3 triangle, its 10 control points are sent to the graphic pipeline.

The first step is the processing by the Vertex Shader. In this shader, the coordinates of each vertex are scaled and multiplied by the viewport and model matrices so that they are transformed into clip coordinates. As the element is curved, the next shader is the

1. This computation is a consequence of the inversion of the system performed in Chapter 3.

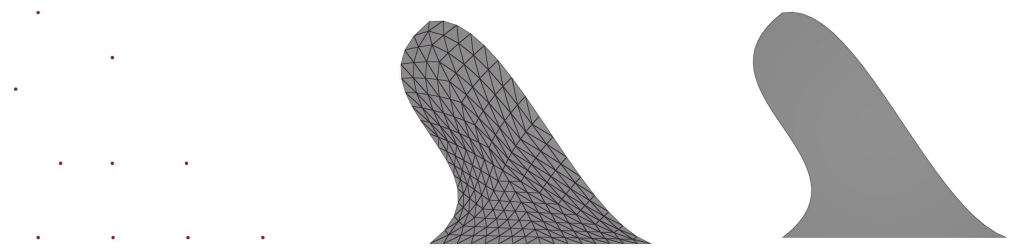

Figure 6.8 – Different steps involved in the drawing of P^3-Triangle. Left, initial distribution of points. Middle, created tessellation and right, final geometry rendering.

Tessellation Control Shader (for linear elements, it goes directly to the Geometry shader). In this shader, the number of subdivisions along each edge of the geometry and the number of subdivisions inside the surface are set. Thanks to these parameters, a tessellation (*e.g.* a subdivision in triangles or lines) of the parametric space is generated by the Tessellation Primitive Generator. The Tessellation Evaluation Shader is then executed for each point of the tessellation. In the case of a triangular element, these points are characterized by a triplet (u, v, w) with $0 \leq u, v, w \leq 1$ and $u+v+w = 1$ which is the corresponding sampling in the parameters space. Thanks to this triplet, the coordinates of the corresponding point in the physical space are computed (see Figure 6.8 middle and left):

$$M(u, v, w) = \sum_{i+j+k=3} \frac{3!}{i!j!k!} u^i v^j w^k P_{ijk}.$$

In a same manner, the geometric normal is computed. This way, an approximation made of triangles of the curved geometry is performed on the fly on the GPU. Each sub-triangular element is then sent to the Geometry Shader. In this shader, all useful data related to each vertex defining the (sub) elements are attached: parametric coordinates, physical coordinates, distance to the clipping plane (if any). Once all these data are set, they are linearly interpolated to define them inside the primitive and sent to the Fragment Shader. The Fragment Shader handles all the entities for each fragment/pixel related to the primitive previously processed. It does give a wanted background color to the pixel and apply a shading to it (toon, diffuse or Phong model) using the normals provided by the previous shader. It is also able to perform wireframe, numerical solution and isoline rendering (see Figure 6.8, right and Figure 6.9). More details will be given in Section 6.4.3. By performing all these previous steps, it is possible to display any type of surface element. We note that the display of the tessellation grid (like in Figure 6.8 middle) can detect if an high-order element is valid or not. Indeed, if the grid is not properly mapped because sub-triangles overlap each other, this means that the mapping is not invertible as the physical position of the points of the tessellation grid is set by the mapping.

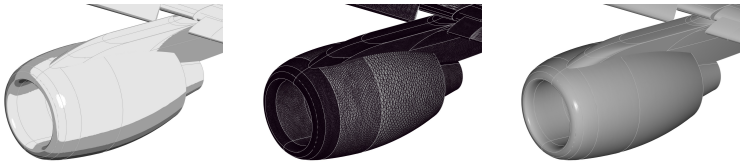

Figure 6.9 – Example of three different displays (toon, wire and Phong model) on an engine of the NASA Common Research Model.

6.3.2 Control of the granularity of the tessellation

In the case of curved elements a tessellation helps to properly display the geometric features but it is not always mandatory. First, let us have a closer look at what happens in the Tessellation Control Shader (see Figure 6.10):

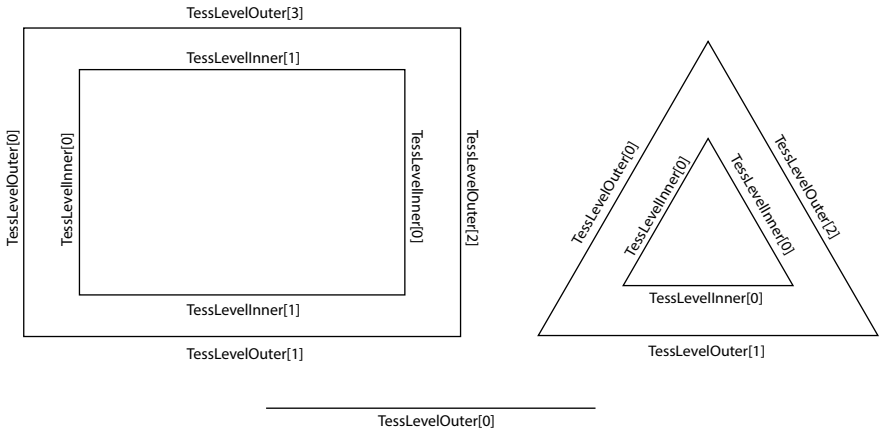

Figure 6.10 – Variables controlling the subdivision for each type of element in the Tessellation Control Shader. Top: Quadrilateral (left) and Triangle (right). Bottom: Edge.

The subdivision of an element is controlled through two types of integer variables `TessLevelOuter` and `TessLevelInner`. `TessLevelOuter[i]` is the number of subdivisions along the $i-th$ line of the element, and `TessLevelInner` is related to the sub element inside (if any) and sets the number of subdivisions for the second layer of triangles along the axis with the associated edge. An uniform discretization is then set in the inner element. The idea is then to set up simple but fast error estimates regarding the geometrical approximation of a high-order element by its equivalent planar element. Based on these error estimates, the level of tessellation of these elements is automatically set. In order to have a conforming tessellation of the global mesh, the discretization of the lines/edges is computed independently of the inner tessellation.
On an edge, if a control point lying on an edge is denoted P_i^{edge} and its equivalent control point on the straight edge is denoted $P_i^{straight}$, the following point-wise error estimate ϵ is considered:

$$\epsilon(P_i^{edge}) = \frac{||P_i^{edge} - P_i^{straight}||}{l_{edge}},$$

where l_{edge} is the length of the edge defined by the extremities [2]. This error estimate is computed for every control point lying on a given edge. The error estimate ϵ_{edge} associated to the edge is then the largest of its control points error estimates. The associated `TessLevelOuter` can then be set using the following formula:

$$\texttt{TessLevelOuter} = 1 + [5t\epsilon_{edge}],$$

where t is a user's integer parameter defining the level of discretization of the line. This

2. If the extremities are the same (case of a looping edge), l_{edge} is replaced by the length of the polygonal curve defined by the control points of the edge.

means that a line is subdivided $N+1$ times if

$$N.l_{edge} \leq 5t.\max_i(||P_i^{edge} - P_i^{straight}||) < (N+1)l_{edge}.$$

In particular, if the element is straight, the number of subdivisions is always equal to 1. For instance in the P^3-triangle of Figure 6.11, the bottom edge is not divided as it is a straight edge, and the right edge is less divided than the left edge because the first one is closer to the straight edge than the second one. In Figure 6.13 (right), it shows that the edges are discretized independently of the characteristics of the surrounding elements.

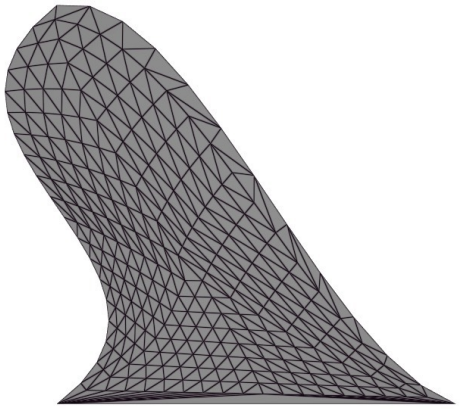

Figure 6.11 – P^3-Triangle of Figure 6.8 discretized with an adaptive tessellation.

For the number of subdivisions inside the face, the idea is equivalent: if a control point lying on the inner part of a face (if any) is denoted P_i^{face} and its equivalent control point on the straight face is denoted $P_i^{straight}$, the following point-wise error estimate ϵ is considered:

$$\epsilon(P_i^{face}) = \frac{||P_i^{face} - P_i^{straight}||}{l_{edge}^{max}},$$

where l_{edge}^{max} is the largest edge length of the straight element. The error estimate ϵ_{face} associated to the inner part of face is then the largest of its control coefficients error estimates. When dealing with quadrilaterals of degree n, another error estimate ϵ_{quad} is considered:

$$\epsilon_{quad} = \frac{|\det(P_{0n} - P_{00}, P_{nn} - P_{00}, P_{n0} - P_{00})|}{||P_{0n} - P_{00}||\ ||P_{nn} - P_{00}||\ ||P_{n0} - P_{00}||}.$$

This error estimate has the ability to detect if a quadrilateral is non-planar which is possible even if $n = 1$ (see Figure 6.12, if the quadrilateral is not tessellated, its true shape is not accurately represented by its decomposition in two triangles [3]). Based on these estimates, the `TessLevelInner` is then set:

$$\texttt{TessLevelInner} = 1 + [5t \max(\max_i(\epsilon_{edge[i]}), \epsilon_{face}, \epsilon_{quad})], \tag{6.1}$$

using the same t as with `TessLevelOuter`.

3. A commonly used trick is to perform shading by using the normal at each of the 4 vertices and linearly interpolate them inside each triangle. The shading will give the feeling that the quadrilateral is non-planar and not divided in 2 triangles. But this trick does not work if we look at the shape of the quad with certain incidences like in Figure 6.12 (top left).

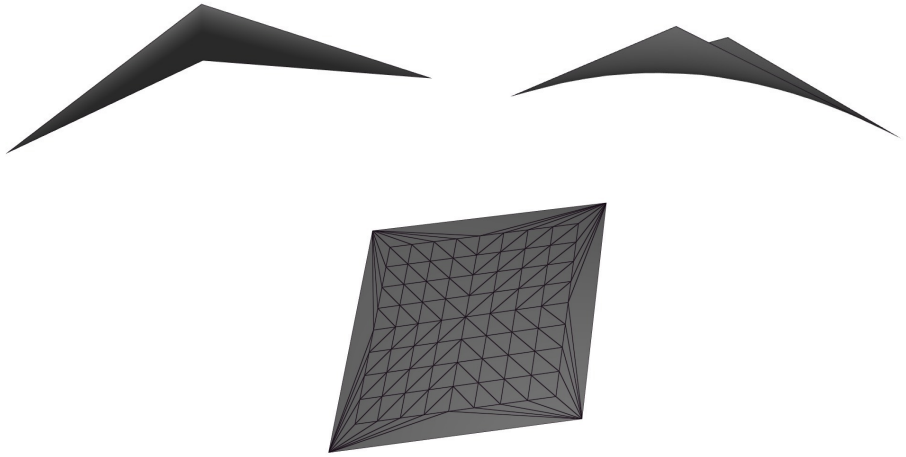

Figure 6.12 – Various renderings of a non-planar Q^1-quadrilateral. Top left: rendering with two triangles. Top right: rendering with a tessellation. Bottom: Used tessellation.

Notice that if the elements are straight, all the ϵ are equal to 0 and consequently `TessLevelOuter` = `TessLevelInner` = 1. A triangle and an edge are then not divided and a quadrilateral is divided in two triangles, which is the classical way to visualize these elements in `OpenGL` Legacy.

6.3.3 Volume elements visualization

The rendering of the volume elements is done in an indirect way. Indeed, only a surface deduced from the elements is rendered. The idea is to render the external surface of the elements. This surface is either be external part of a volume provided without surface or the external surface of elements intersected by a cut plane. In the last case, the display of the cut plane is carried out with clipping so that the external surface is half visible at the same time (see Figure 6.13). In this case, the rendering process is exactly the same as a surface rendering with quads, triangles and edges.

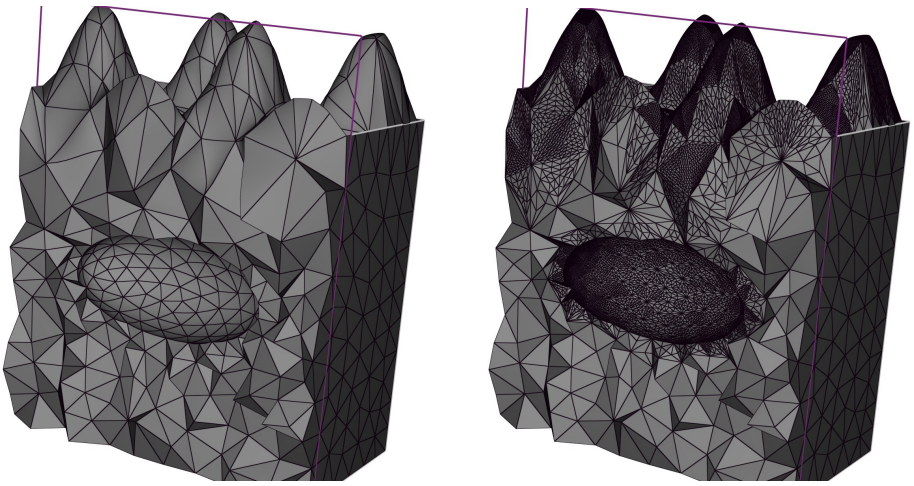

Figure 6.13 – Rendering of a mesh composed of P^3-tetrahedra with cut plane and clipping. The right picture shows the adaptive tessellation used to render the P^3 surface triangles of the P^3-tetrahedra.

6.4 High-order solutions visualization

In this section, we highlight the interesting properties offered by the `OpenGL` graphic pipeline and how we can use them to display high-order solutions in an almost pixel-exact fashion.

6.4.1 Almost pixel-exact solution rendering

To display high-order (scalar) solutions using the `OpenGL` graphic pipeline, the Fragment Shader is used. As seen in the previous section, when considered, the Fragment shader is attached to a pixel representing a fragment of a given primitive associated to an element. In particular, it receives the interpolation of the various data attached to the each one of the vertices defining the primitive. It does interpolate normals, parametric and physical coordinates. Thus, if a given element is defined by a linear mapping, it has access to the exact set of physical coordinates associated to a given set of parametric coordinates. When the mapping is not linear, it does give an approximation, depending on the level of the tessellation, of the set of physical coordinates associated to a given set of parametric coordinates. Since the parametric coordinates are available in the Fragment Shader, the only thing left to display high-order solutions is to provide the value of the solution at each degree of freedom of each element. A convenient way to transmit these data is to use textures that are meant to store raw data of large size. From a practical point of view, it is the control solutions (*e.g.* the coefficients of the solution in the Bernstein basis) that are stored in the texture.
Once parametric coordinates and control solutions are obtained, the solution is computed using its exact polynomial expression. The effective computation of this solution is based on a de Casteljau's algorithm which evaluates with a minimal number of operations a polynomial of any degree using its control coefficients. The value is then known for each fragment. The main pro is that if the mapping is linear then the computed solution is pixel-exact (Figure 6.14), otherwise, it is almost pixel-exact.

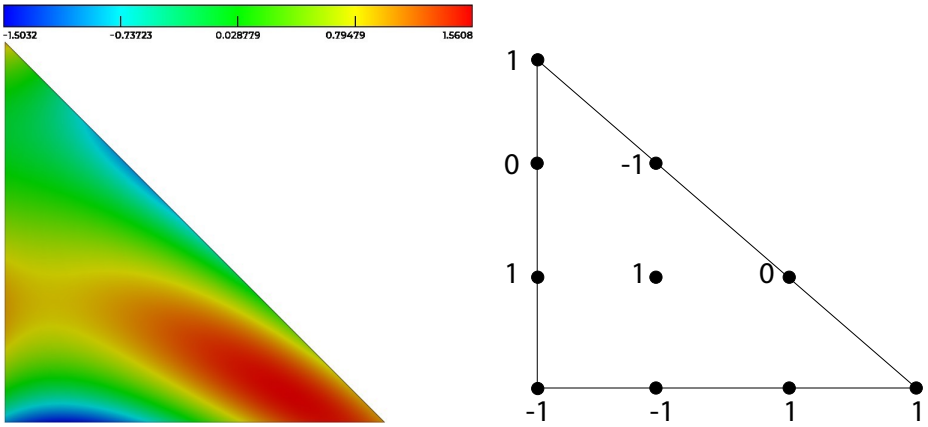

Figure 6.14 – Pixel-Exact rendering without subdivision of a P^3-solution on a triangle.

Furthermore, a special treatment is performed for the display of a solution at planar quadrilaterals. Indeed, the subdivision in 2 triangles is enough for the visualization of the shape of the quadrilateral as the two triangles span the same shape as the considered quadrilateral (Figure 6.15, first line). However, it is not enough for the rendering of a solution (Figure 6.15, third line right and left). In fact, by using the Geometry Shader, the

parametric coordinates are interpolated inside each of the sub-triangle in a linear way, but not in a bilinear way. Consequently, if the quadrilateral is not a parallelogram, the solution will not be reliably rendered. For this reason, another error estimate is considered:

$$\epsilon_{quad}^{sol} = \frac{||P_{00} - P_{n0} - P_{0n} + P_{nn}||}{\max(||P_{0n} - P_{00}||, ||P_{nn} - P_{0n}||)}.$$

This error estimate has the ability to detect if the mapping associated to a Q^1 quadrilateral has quadratic terms or not. When a solution is displayed on a quad, ϵ_{quad} is replaced by $\max(\epsilon_{quad}, \epsilon_{quad}^{sol})$ in Equation (6.1). Note this error estimate is not used when the mesh is displayed without solution rendering (Figure 6.15).

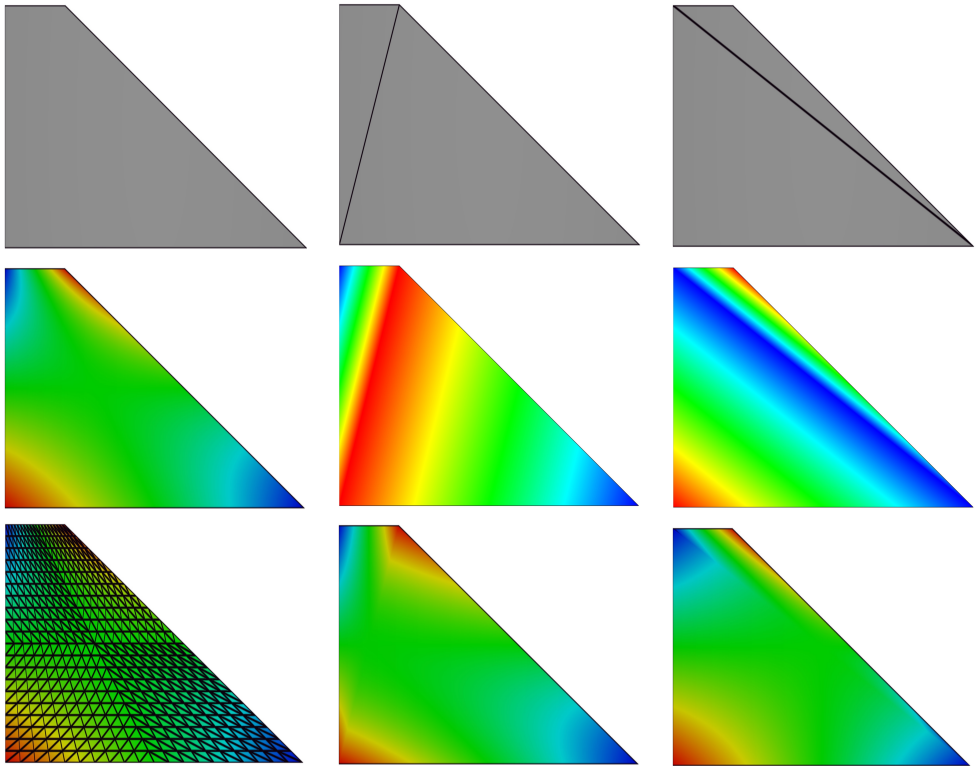

Figure 6.15 – Left, from top to bottom: rendering of a solution on a non-linear Q^1-quadrilateral with the used tessellation. Middle and right, from top to bottom: rendering of the solution on quad for each of its two decomposition in triangles; linear rendering done using the decomposition; bilinear rendering with the Fragment Shader but with a wrong approximation of the parametric coordinates due to the decomposition in only two triangles.

Finally, to display a numerical solution, it is mandatory to have a palette and its colormap (e.g. a range of values for the solution and its associated color in RGB). In a same manner, these two arrays are sent to the Fragment Shader using textures. The next step is therefore to transform our computed solution into a RGB vector. First, we check the values of the bounds of the palette and if the solution is not inside the bounds, it is set to the bound of closest value. The RGB vector is then deduced using the colormap array.

6.4.2 Computation of proper bounds for a high-order solution

When displaying a numerical (scalar) solution, a palette is always used to determine the color associated to a solution value. It can be set by the user, but in some cases, the user is not aware in advance of the variations of its numerical solution and in particular of its range of values. Classic visualization methods use the sampling of the solution on the visualization grid to get it. However in our case, the solution evaluation is performed in the Fragment Shader and no easy callback of the highest and lowest computed value is easily possible.
The idea here is therefore to find proper bounds for a high-order solution (in a preprocessing step) and then to use them as a default palette. If the solution is linear (or constant), it is straightforward, as the extrema on each element lies on the vertices. In this case, the bounds of the solution are the largest and the smallest values of the solution at the vertices. For other polynomial solutions, it is not necessarily the case and a refinement procedure is required. Now, if we recall that a high-order solution of degree n on a given element can be rewritten into a Bézier form (see Section 3.3), we can write f as:

$$f(\mathbf{U}) = \sum_{i=0}^{N} B_i^n(\mathbf{U}) F_i,$$

where $\mathbf{U}$ is a vector containing the coordinates of the reference element, $B_i^n(\mathbf{U})$ is the appropriate Bernstein polynomial for this given element and N is the number of nodes.
If we recall, that whatever its type is, these polynomials form a partition of unity (see Section 3.2) and are positive, we have that:

$$\forall \mathbf{U}, \quad F_{min} = \min_i(F_i) \leq f(\mathbf{U}) \leq \max_i(F_i) = F_{max}.$$

However, (see Section 3.3) these bounds are not exact bounds. In order to get a closer idea of the bound values, subdivision strategies are developed. The used subdivisions are the ones developed in Section 3.2. Thanks to these subdivisions, we are able to compute more accurate bounds for the high-order solution. This is explained in **Algorithm** 27:

Algorithm 27: Computation of accurate bounds for a high-order solution on a mesh

Initialization: Set initial bounds $[lb, ub]$ using the largest and smallest value of the solution f at Lagrange points of the mesh.

- For each element e of the mesh
  - ⋆ $depth = 0$
  - ⋆ Compute extremal control solutions of the element F^e_{min} and F^e_{max}
  - ⋆ (1) If ($lb > F^e_{min}$ or $ub < F^e_{max}$) and ($depth < depth_{max}$)
    - — Subdivide solution n sub-solutions.
    - — For each sub-solution i
      - — Update lb and ub with the value at the vertices for each sub-element.
      - — $depth = depth + 1$
      - — if ($lb > F^{e_i}_{min}$ or $ub < F^{e_i}_{max}$) re execute the block starting from (1)

By using this recursive algorithm, accurate bounds for any high-order solution are obtained. Note that all these developments are independent of the degree of the geometrical element where the solution is defined. In Figure 6.16, the application of the algorithm

changes the initial bounds from $[-1,\ 1]$ (deduced from the value at Lagrange points) to $[-1.5,\ 1.5]$ which significantly changes the color of the rendered solution. This is particularly interesting to detect oscillations in high-order solution fields. Let us imagine, for instance, a P^n projection of an Heaviside function.

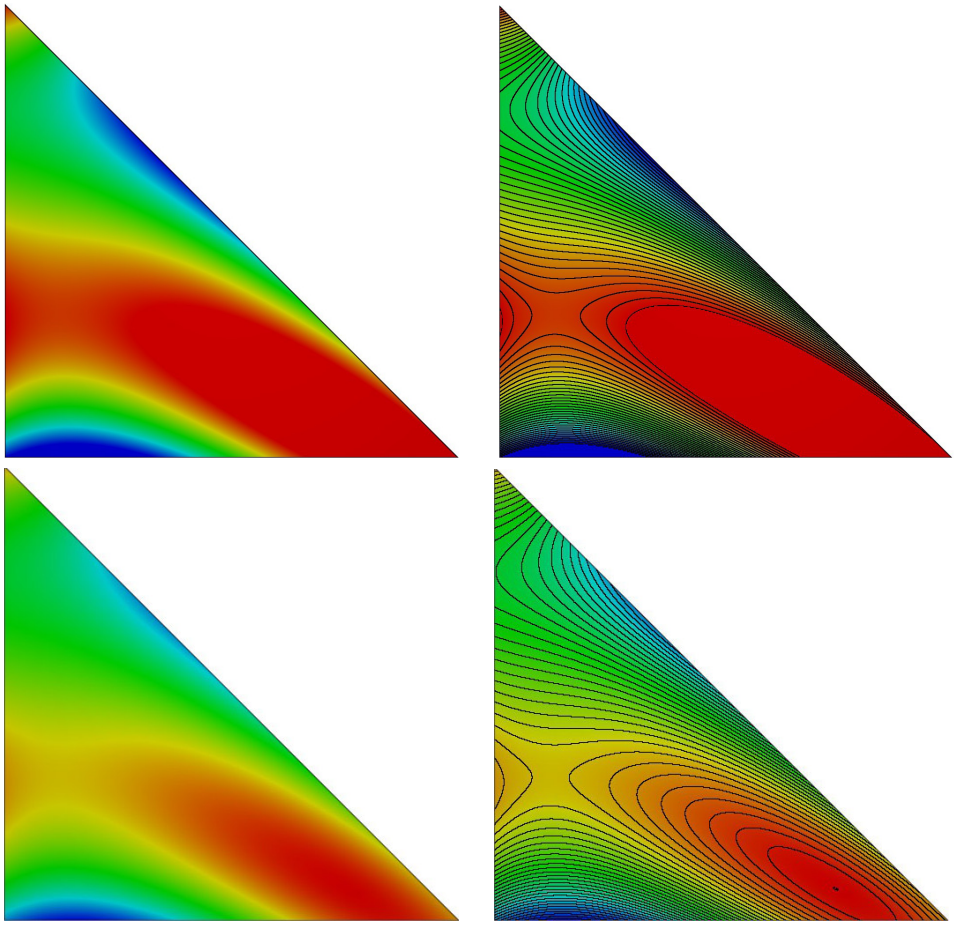

Figure 6.16 – Rendering of P^3-solution of Figure 6.14 with its isovalues. Top: rendering with using the largest and smallest values of the solution at the Lagrange points for the palette (*i.e.* bounds are $[-1,\ 1]$). Bottom: rendering with a refinement procedure to compute the bounds of the solution for the palette (*i.e.* bounds are $[-1.5,\ 1.5]$).

6.4.3 Isoline and wireframe rendering

The display of isolines is a common feature used when displaying a numerical (scalar) solution. Usually, classic visualization software use techniques like *marching cubes* to plot isolines. As considered solutions are in the end linear, it just consists in plotting an edge. In our context, these methods do not apply as our solution is mainly not linear. Our wish is also to have a rendering of isolines which thickness is independent of the considered element and its degree. Also, it should be independent of the zooming level. The proposed solution takes advantage of the built-in functions proposed in the Fragment Shader and uses them to display isolines. The Fragment Shader has the following function `fwidth` which takes in input a scalar variable and outputs an approximate value of the norm of its spatial gradient with respect to the coordinates in the window space. In other words, it gives the variation of your numerical solution on the screen of your computer and it is

automatically recomputed at each rendering.
To understand how to plot an isoline with a wanted thickness, let us have an analysis of the behavior of the numerical solution in the vicinity of an isoline curve. For this purpose, let us consider the numerical solution as a smooth function f, that we will assume at least to be C^2 and let us define our isoline as $I_0(f) = \{x \in \mathbb{R}^2 | f(x) = f_0\}$. Let us also consider a point $p \in I_0(f)$ such that $||\nabla f(p)|| \neq 0$. A Taylor expansion at the first order in the neighborhood of p with respect to the coordinates of the screen gives us:

$$f(q) = f_0 + (\nabla f(p))^T (q - p) + \mathcal{O}(||q - p||^2),$$

where q is the neighborhood of p. By neglecting terms of order 2, we can write that:

$$|f(q) - f_0| = |(\nabla f(p))^T (q - p)|.$$

As f is C^2, we have by continuity:

$$\nabla f(q) = \nabla f(p) + \mathcal{O}(||q - p||).$$

By neglecting terms of order 1, we can write that:

$$\nabla f(q) = \nabla f(p).$$

Now, if we assume that:

$$|f(q) - f_0| < \epsilon ||\nabla f(q)||, \tag{6.2}$$

and if we decompose $q - p$ as:

$$q - p = \alpha t + \beta n, \tag{6.3}$$

where t is a unit vector tangent to $I_0(f)$ and n a unit vector normal to $I_0(f)$ (e.g. aligned with $\nabla f(p)$). We then have at first order:

$$|(\nabla f(p))^T (q - p)| = |\beta| . ||\nabla f(p)||,$$

which leads to ($||\nabla f(q)|| = ||\nabla f(p)||$ and $||\nabla f(p)|| \neq 0$):

$$|\beta| < \epsilon.$$

Consequently, we notice that Inequality (6.2) gives a way to control the normal component of the distance between the two points p and q.
Under regularities assumptions ($||\nabla f(p)|| \neq 0$), it can be shown that this property is true and that q and p can be chosen such that $\alpha = 0$ in Equation (6.3) [Taubin 1994].

In practice, for a given f, its two closest isovalues f_0 and f_1 are computed. And for each of these isovalues, Inequality (6.2) is tested. If it is true, the color associated to the fragment shader is set to the wanted color for the isoline in RGB. The main advantage of this method is that it does not ask any preprocessing to display isovalues. The result of isoline rendering is immediate when asked by the user and gives lines of thickness ϵ in pixels (Figure 6.17 shows that the thickness of the isolines remains the same whatever the level of zoom is). Notice that when f is constant or the considered isoline is an extremum or containing a singular point of f ($||\nabla f(p)|| = 0$), the theory cannot apply. However, in this case, Inequality (6.2) is always false and therefore no isoline is plotted there.

Using the same approach, a way to properly render the wireframes (*e.g.* the edges of a geometrical element) is deduced. Indeed, apart from the linear case, where the rendering

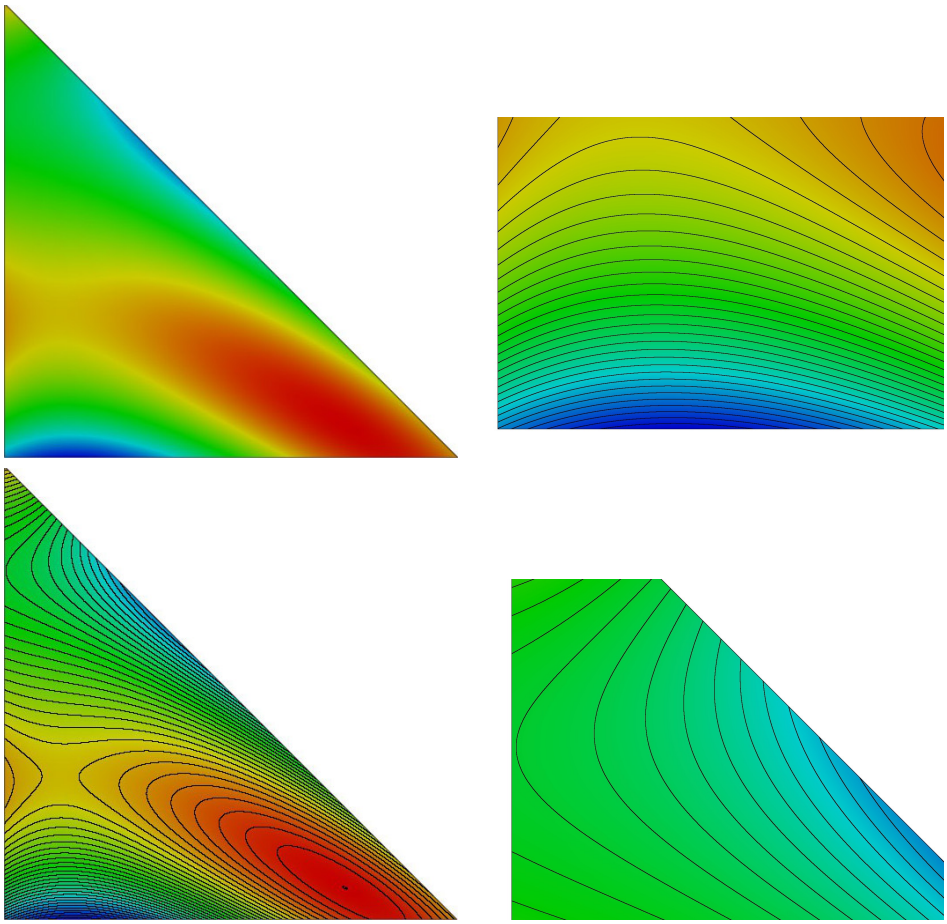

Figure 6.17 – Left: Rendering of P^3-solution of Figure 6.14 with its isovalues. Right : zoom in various parts.

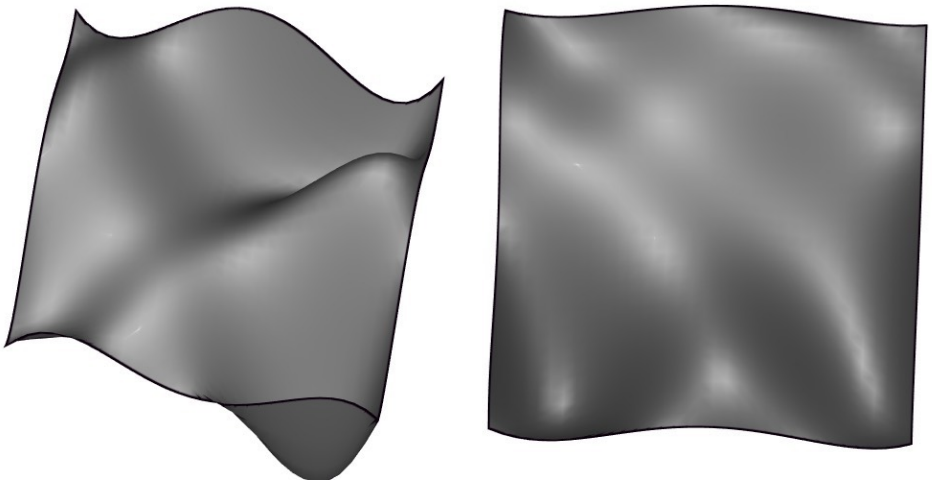

Figure 6.18 – Wireframe rendering performed on a non-planar Q^4-quadrilateral.

of such wireframes is straightforward with basic geometry [NVIDIA], it becomes harder to perform a proper rendering of wireframe with non-linear elements. Several problems apply in this case: the correspondence between the parameters and the physical coordinates is not linear, but worst of it, the displayed geometry is piecewise linear, as it is a tessellated

geometry. The idea is therefore to apply the previous technique by considering each of the parametric coordinates as a numerical solution. Indeed, a parametric coordinate is defined between 0 and 1 and reaches these values on the boundary of the element. Plotting the isolines associated to 0 and 1 gives thus a way to plot the extremities of any element of any degree with a constant and zoom-independent thickness (Figure 6.18). This way anti-aliasing is automatically featured. We also avoid an extra cost which is caused by the classic rendering techniques which plots a set of straight edges on the boundary of the elements. This may be time consuming when the mesh is of large size where the proposed approach gives an immediate rendering.

6.5 Numerical examples

In this section, we show some possible uses of the software `Vizir` issued from this research work.

6.5.1 High-order mesh adaptation example

This example is taken from [Coulaud and Loseille 2016]. It shows high-order mesh adaptation. The underlying idea is to develop high-order error estimates so that it minimizes the error induced by the high-order differential of a given numerical solution. The adaptation is performed with respect to a multiscale function. This function oscillates at high frequency and contains small details, it is given by :

$$f_r(x,y,z) = 8\,xyz\,\sin(5\,\pi\,xyz)^4 + \frac{1}{10}\left(1-(\sin(5\,\pi\,xyz)^4\right)^8\,\cos(100\,\pi\,xyz).$$

The high frequency variations of the function are depicted in Figures 6.19 and 6.20. The final mesh for P^1 contains more that 2 000 000 vertices, 564 774 triangles and 14 621 126 tetrahedra while the equivalent mesh for P^4 in terms of degrees of freedom contains 39 612 vertices, 19 710 triangles and 208 197 tetrahedra.
In all the figures, we assess the interest of mesh adaptation for high-order functions. In particular, we show no loss of accuracy in the solution rendering while the mesh (and the number of used degrees of freedom) is coarsened and the order of the solution increased. Note that, as all the meshes are linear, the rendering is pixel-exact.

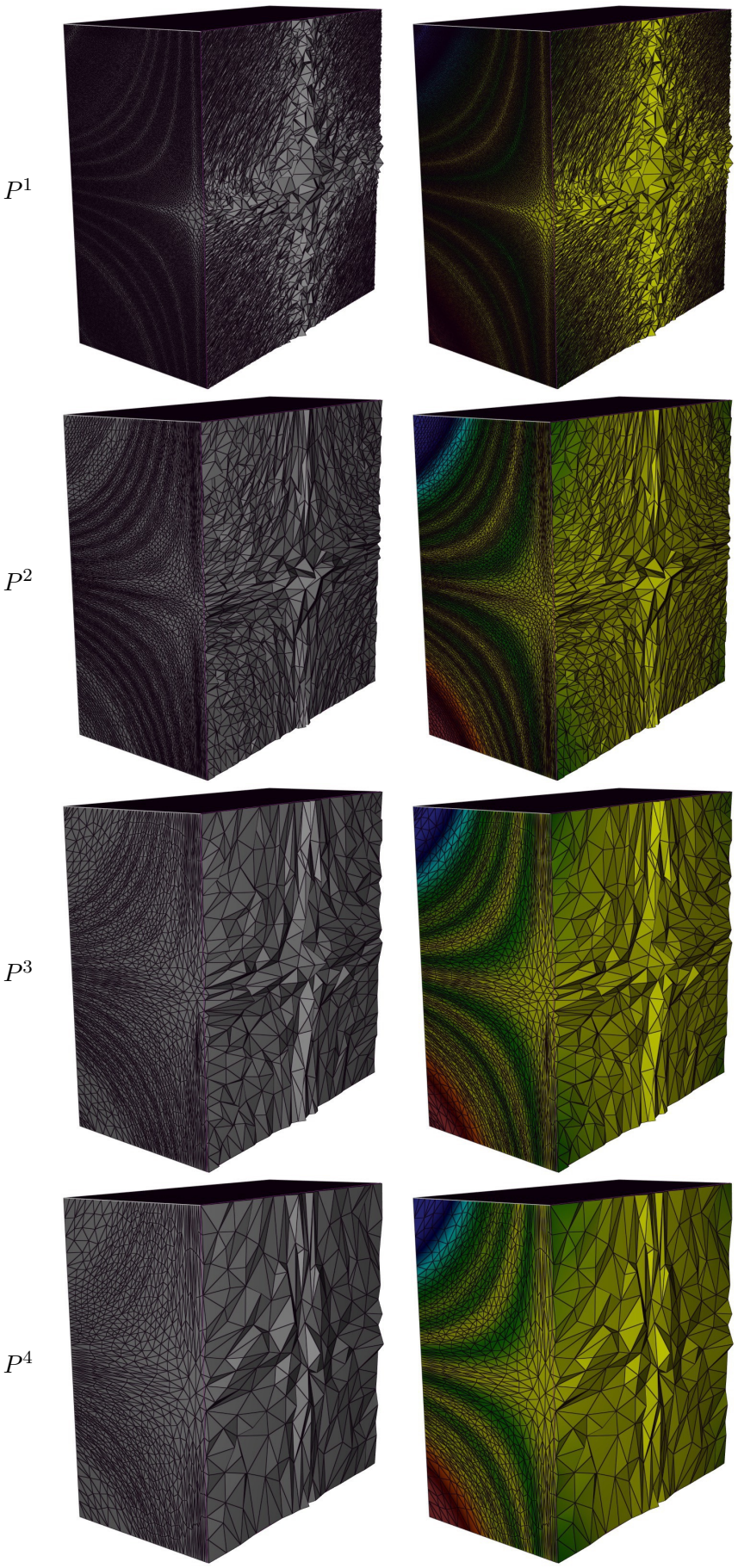

Figure 6.19 – From top to bottom, P^1 to P^4 adapted meshes (left) with the corresponding solution on it (right).

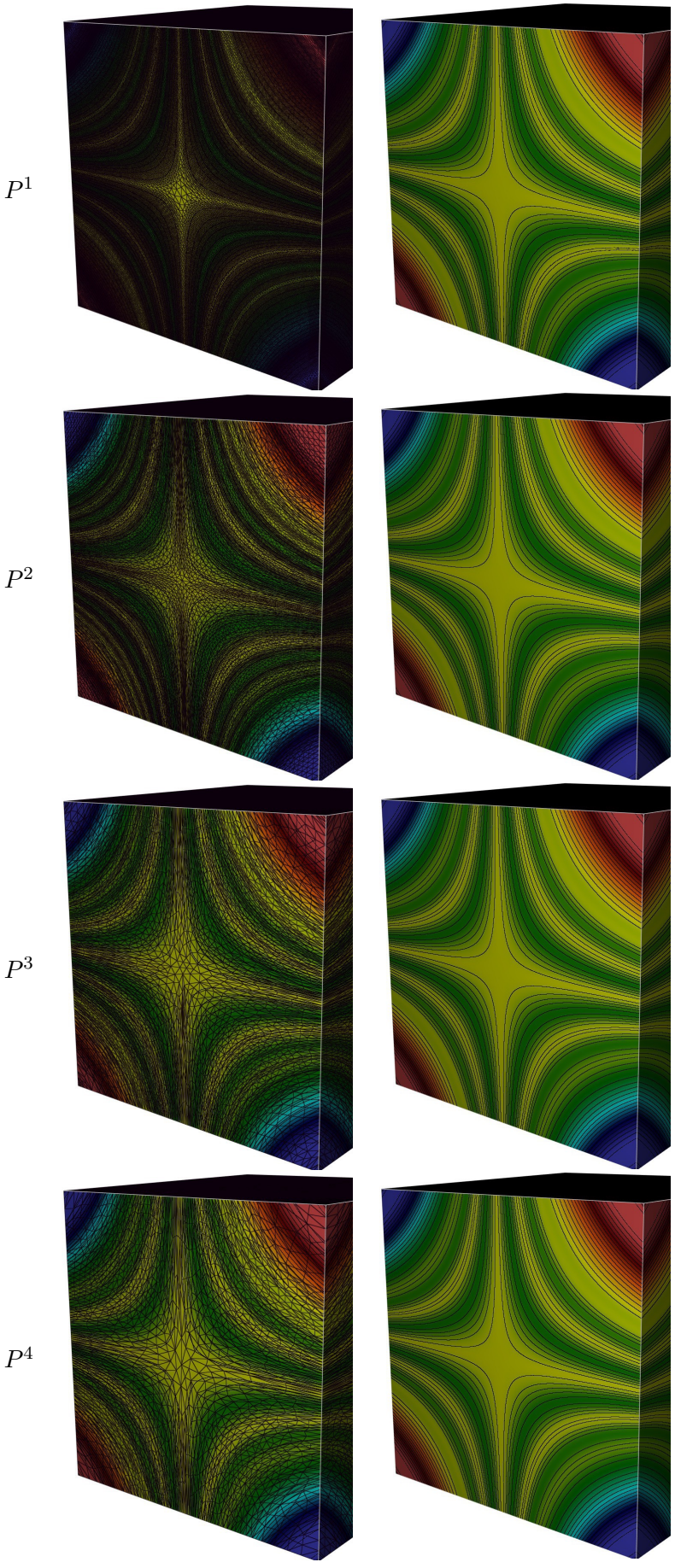

Figure 6.20 – From top to bottom, P^1 to P^4 adapted meshes with the solution (left) and isovalues (right).

6.5.2 High-order boundary element solution

This example is taken from [Chaillat et al. 2018]. It shows a P^3-geometry of an unarmed F15 aircraft. The considered problem is the scattering of plane waves using adaptive Boundary Element Method (BEM). In Figure 6.21, the resolution of a P^3 BEM on the aircraft is shown. In Figure 6.22, the used P^3-mesh is shown with various tessellation levels. We clearly observe the diffraction phenomenon in a high-order fashion as it takes into account the high-order features of the geometry.

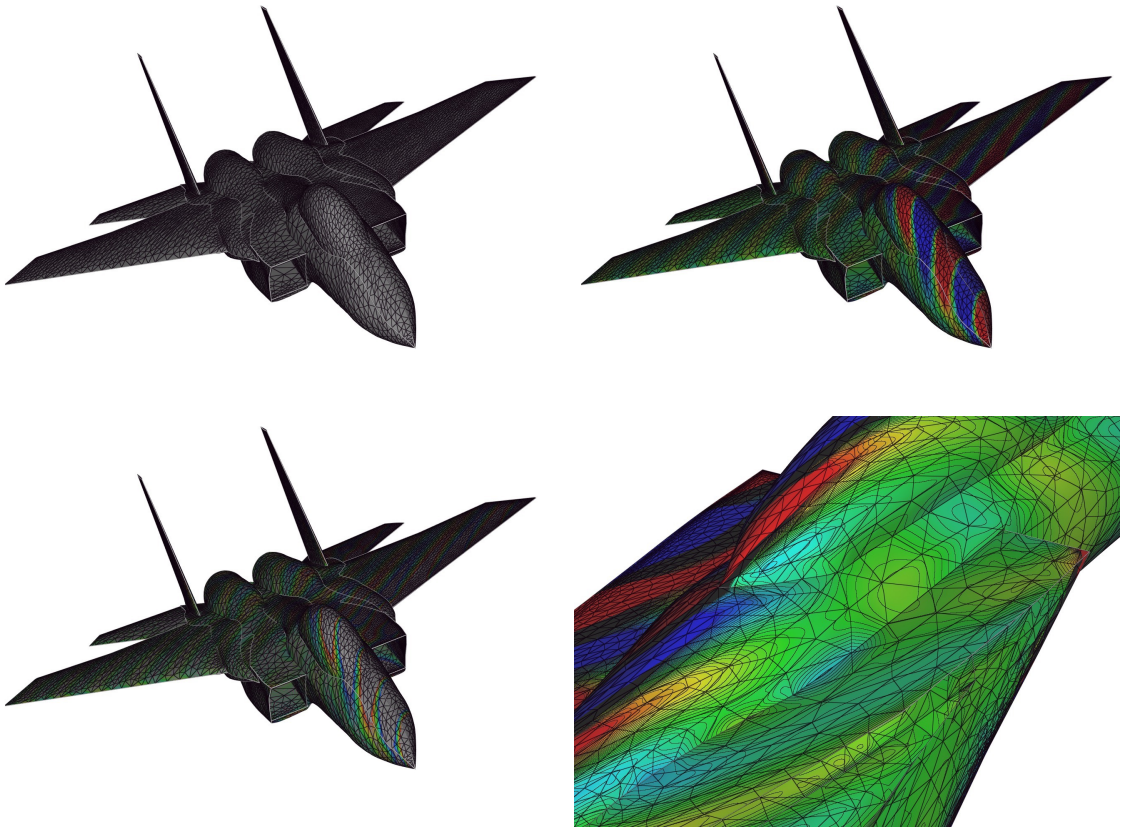

Figure 6.21 – High-order BEM resolution of the scattering of plane waves on an aircraft. Top, meshes. Middle, solution, Bottom, isolines. Bottom right, zoom around the cockpit. Courtesy of Stéphanie Chaillat (ENSTA).

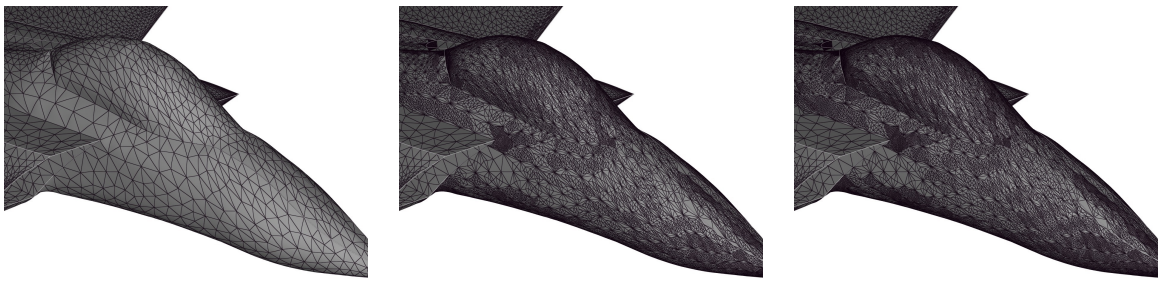

Figure 6.22 – Left, zoom on a curved part of the P^3-mesh used for the BEM resolution. Middle and right, various tessellation levels used for the display of the geometry. Courtesy of Stéphanie Chaillat (ENSTA).

6.5.3 Wave propagation with Perfectly Matched Layers (PML)

This example is based on the numerical simulation explained in [Baffet et al. 2019]. It is a wave propagation inside the cube $[-1,\ 1]^3$ with a PML boundary condition on the left ($y = -1$) and on the right ($y = 1$), and a Dirichlet boundary condition at the top ($z = 1$) and at the bottom ($z = -1$). It is solved on an hexahedral mesh with a Q^6 solution using Gauss-Lobatto quadrature points. The resolution is done with spectral finite elements and a Leap-Frog scheme for time integration. In Figures 6.24 and 6.23 , we show the mesh and the solution at the following time step on the volume elements lying along $x = 0$, $y = 0$ and $z = 0$. We clearly observe the absorbing boundary condition at the boundary $x = 1$ and the reflecting waves at the boundary $y = 1$. For this case, the hexahedral rendering is pixel-exact, as the geometrical elements are truly linear.

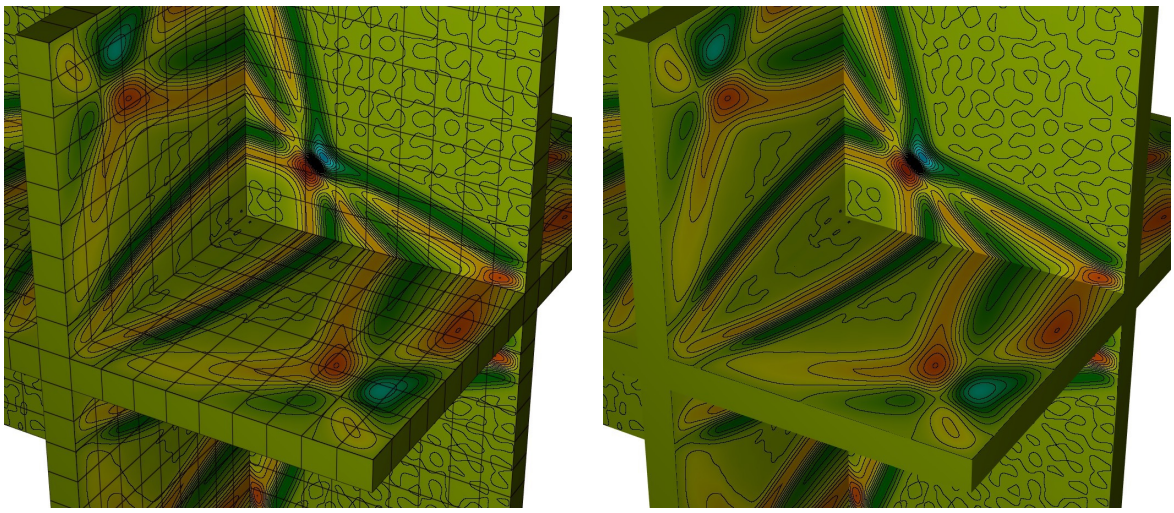

Figure 6.23 – Zoom in the middle of the cube with the solution of Figure 6.24 with its isolines. Courtesy of Sébastien Impériale (INRIA).

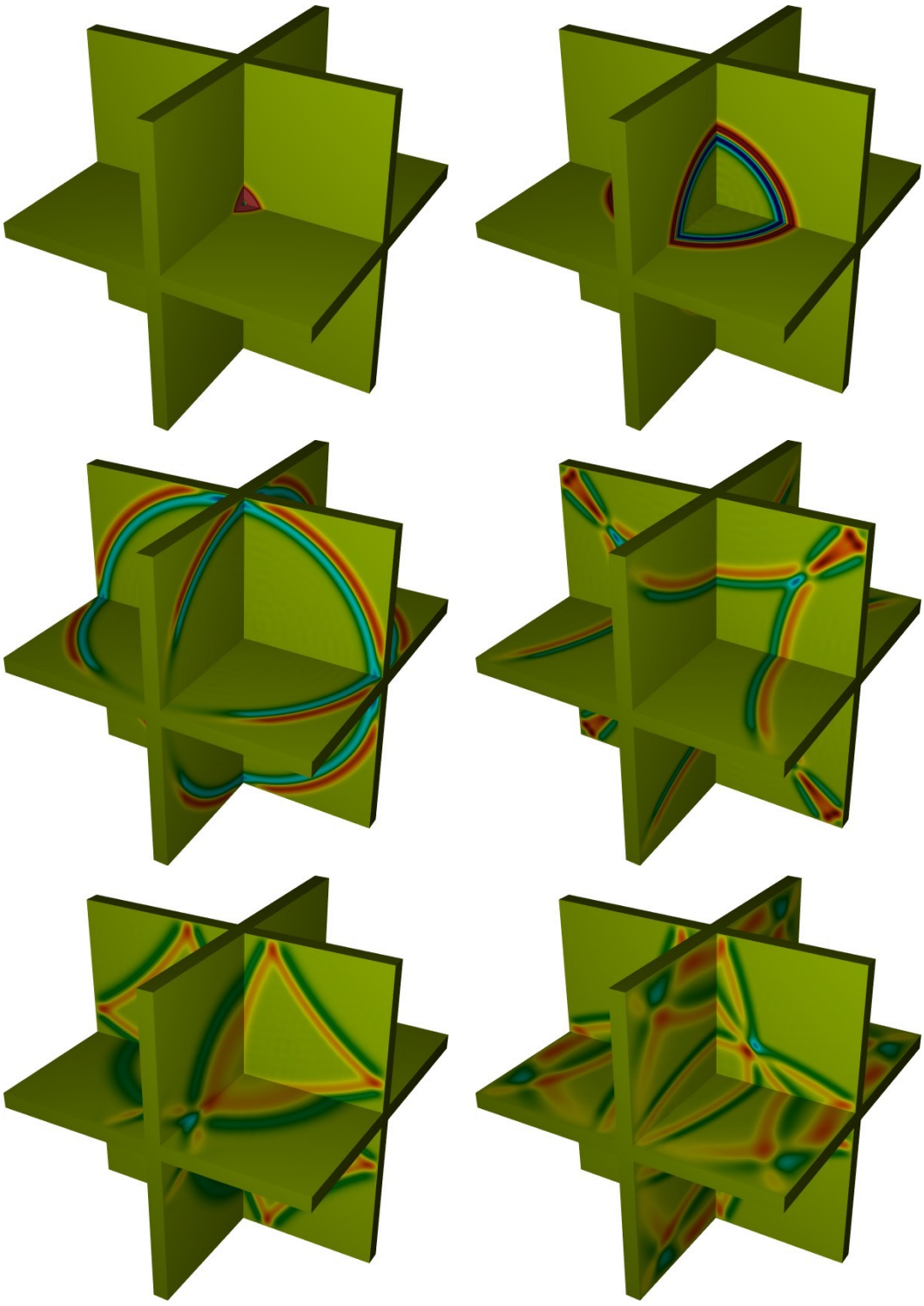

Figure 6.24 – Q^6 solution of a wave propagation problem on an hexahedral mesh. Courtesy of Sébastien Impériale (INRIA).

6.5.4 Computational AeroAcoustics

This example is from [Peyret 2017]. It is based on the resolution of two equations. First, Euler equations solved with Discontinuous Galerkin Method provides the mean flow. Then, linearized Euler equations solved with Discontinuous Galerkin Method provide the acoustic field. In Figure 6.25, the method is shown on an engine containing a central body. The domain of computation is bounded with the 4 boundaries: inlet, wall, outlet and wall. The mean flow is computed on Q^2 mesh with Q^{10} solution (top) and the propagation of a small perturbation is computed over the mean flow using a Q^8 solution (bottom). We clearly observe the variations of the mean flow as well as the small perturbations induces by it.

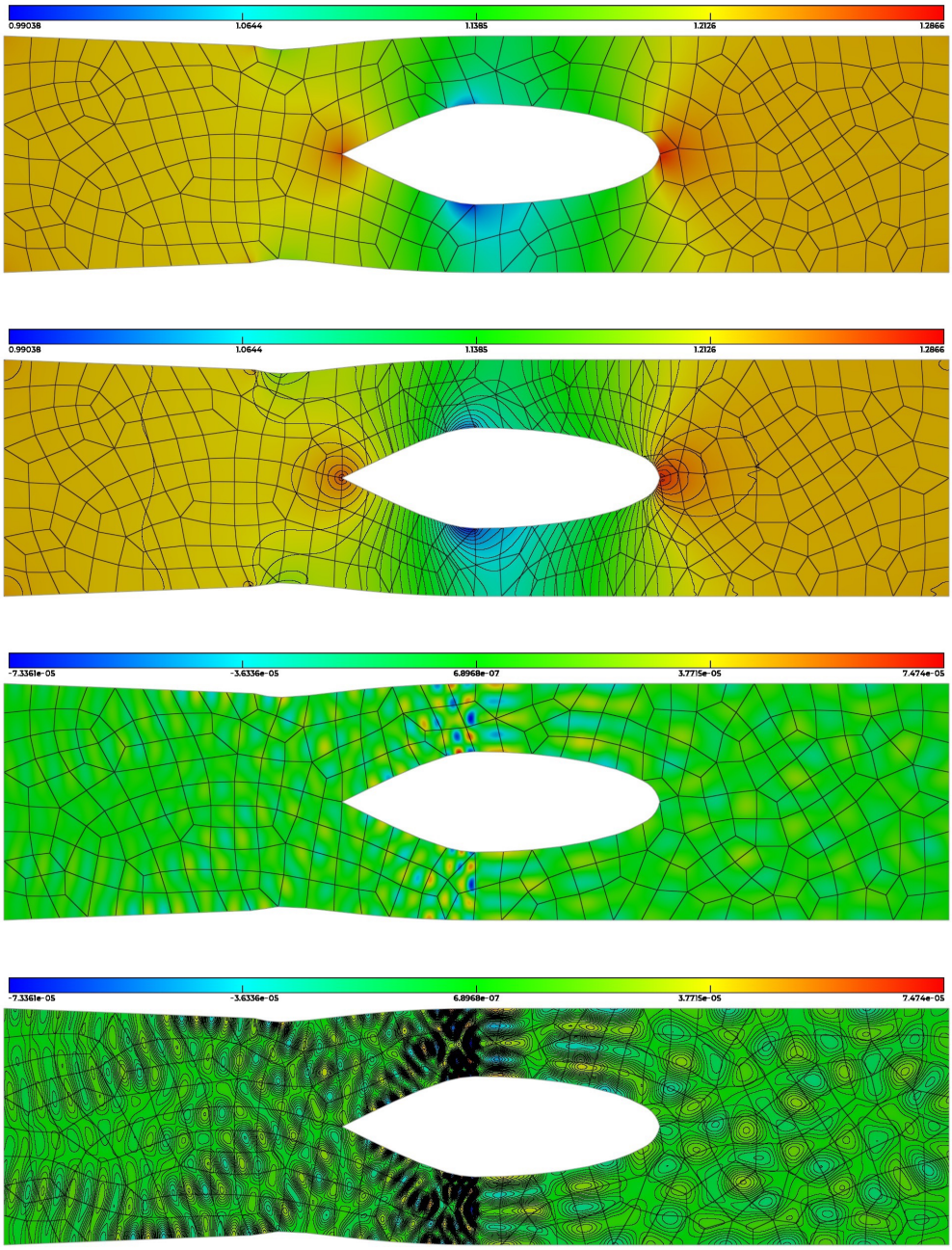

Figure 6.25 – Top, mean flow around an engine using a Q^2 mesh with Q^{10} solution. Bottom, perturbation over the mean flow using a Q^8 solution. Courtesy of Christophe Peyret (ONERA).

6.5.5 Waveguide solution

This example is taken from [Viquerat 2019]. This shows a procedure for the generation of accurate polychromatic sources in waveguides which takes into account mode dispersion. This method is implemented using a Discontinuous Galerkin Time-Domain (DGTD) solver. In Figure 6.26, E_x field during the conversion of a polychromatic TE_{01} mode to a TE_{02} mode is shown, with a zoom in Figure 6.27. The displayed mesh is P^2 and the solution is P^2 as well.

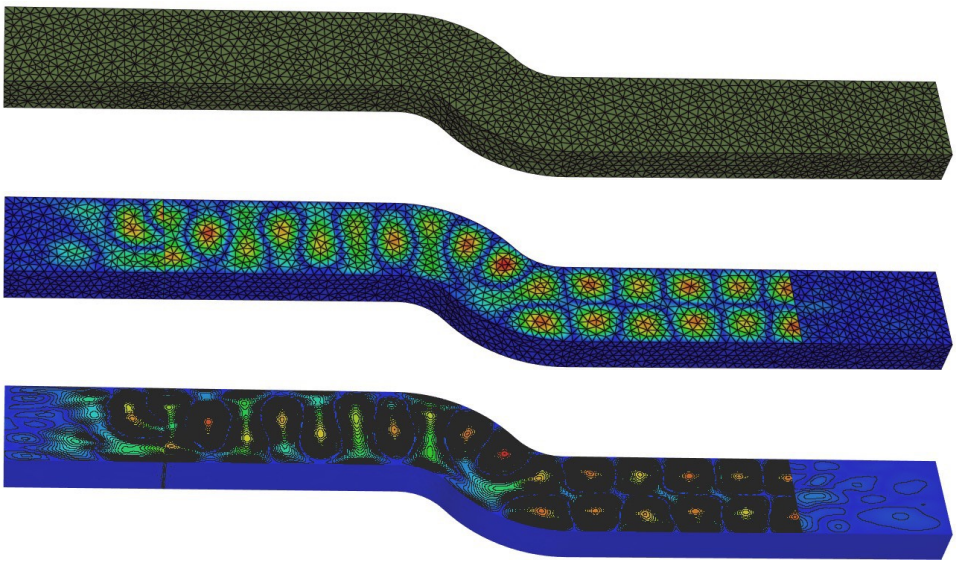

Figure 6.26 – From top to bottom, meshes, solution and solution with isolines of a conversion of a polychromatic TE_{01} mode to a TE_{02} mode. Courtesy of Alexis Gobé and Jonathan Viquerat (INRIA).

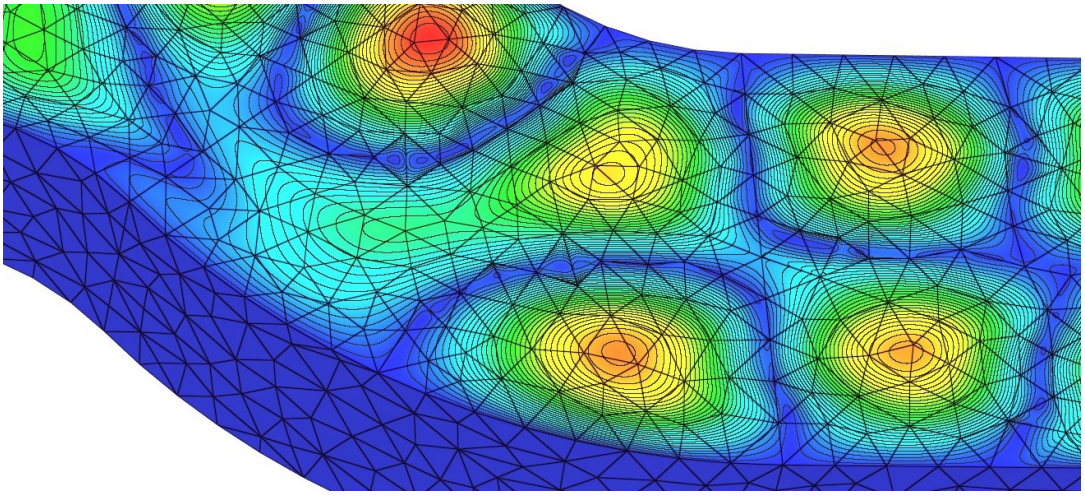

Figure 6.27 – Zoom around the curved part of the mesh with the waveguide solution of Figure 6.26. Courtesy of Alexis Gobé and Jonathan Viquerat (INRIA).

6.5.6 CAD visualization

Using a linking with the `EGADS` CAD API [Haimes and Drela 2012], the rendering of CAD is possible. The process to display a CAD is the following:

1. Opening of the CAD geometry with `EGADS`,
2. For each face of the CAD model, generation of a fast conforming triangular tessellation using `EGADS` functions,
3. Display of the geometry as a triangular mesh but using the real geometrical normals as vertex attribute. It gives a smooth representation of the geometry and the visualization mesh is mostly not visible.

An example of a rendering of Dassault Falcon CAD geometry defined with 32 patches is given in Figure 6.28.

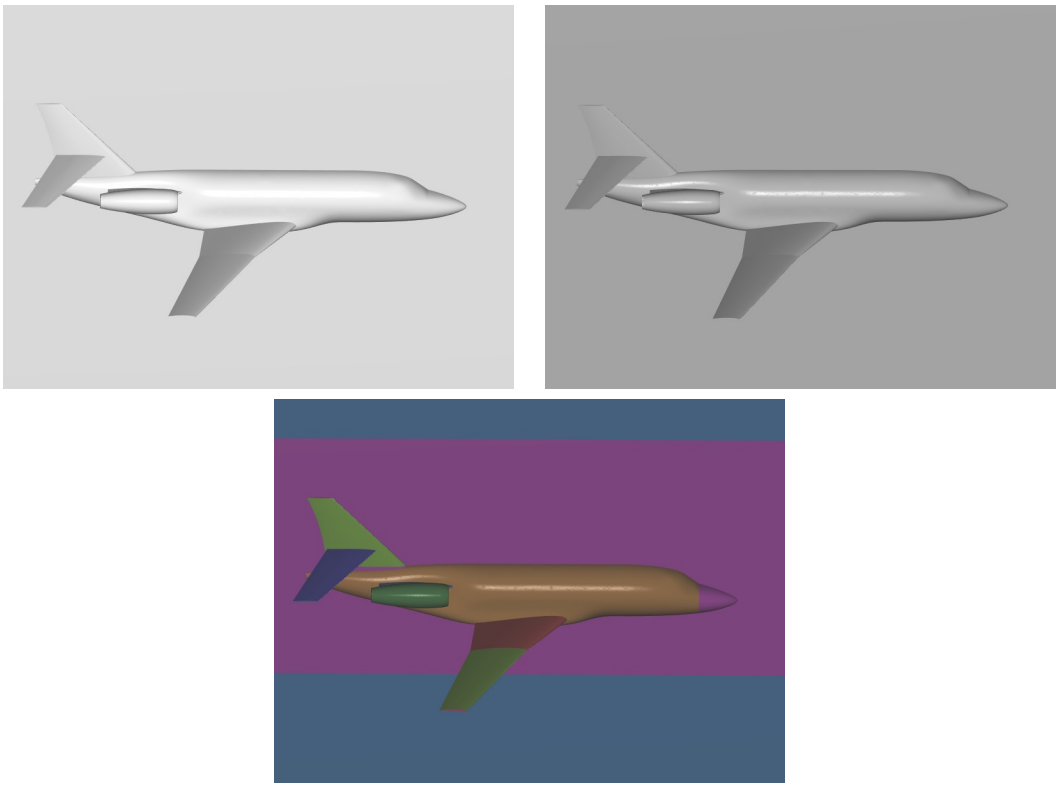

Figure 6.28 – Various renderings of a CAD model of a Falcon aircraft.

6.6 Conclusion

In this chapter, we have presented novel approaches to high-order mesh and solution visualization. This method is based on `OpenGL` 4.0 and enables a full GPU-based rendering of high-order entities and a reliable rendering of high-order solutions on it. The latter is even pixel exact when it is defined on linear elements. The `OpenGL` framework and its use in our case has been exposed. Geometrical properties of the high-order elements have been discussed in order to optimize their discretization by using the GPU. Also, numerical properties of the high-order solutions have been studied in order to get appropriate bounds for the solution and therefore set a pertinent colorbar for the user. A method to plot isolines and perform wireframe rendering on any kind of elements has been developed. This method is based on implicit curves rasterization methods and does not need a marching square method. Finally several examples illustrate the ability of the technology to handle high-order meshes and solutions in various applications. All these developments are part of a the new generation of `Vizir` software that is developed in the Gamma team and have been subject of a publication [Loseille and Feuillet 2018]. The software is at the time able to render any geometry from degree 1 to 4 and any polynomial solution from degree 0 to 10. Perspectives for this work are about the volume rendering. Indeed, further developments to perform the capping (e.g the computation of the exact intersection of volume mesh with a cut plane) and in particular with elements of order greater than one. Also, some developments are needed for the proper rendering of isosurfaces. For the latter, classic rendering techniques are currently used.

Conclusion

In Part II of this thesis, we have investigated the case of high-order meshes. The scope of Chapter 3 was to introduce the framework used in this thesis to deal with high-order meshes. The cornerstone is to consider any high-order element as a Bézier element. Using this analogy, a lot of properties regarding high-order meshes can be derived. In particular, subdivision techniques can be easily set up and from this, robust techniques for the validity of high-order elements can be set. All these results have been established for the standard finite elements (edge, triangle, quadrilateral, tetrahedron, prism, pyramid and hexahedron) and successfully implemented. Then a last part of this chapter was devoted to the quality functions used with high-order meshes and again, using the analogy between Bézier and high-order elements, geometrical quantities have been derived from algebraic entities to provide an extension of standard quality measures defined with elements of degree one. Chapter 4 tackled the subject of high-order surface mesh generation with the development of devoted error estimates. Starting from a well-established parametric surface mesh generation technique, a generalization of this technique has been proposed. A specific effort has been made in keeping the independence with respect to the parameterization as it is a fundamental prerequisite for a suitable surface mesh generation process. Some examples were shown to prove the interest of the approach. The development of high-order surface mesh generation tools is essential as it is commonly the first step in a high-order mesh generation process. To this end, Chapter 5 was dealing with the development of mesh optimization operators for the specific case of meshes of degree 2. A generalization of two classic mesh optimization operators (generalized edge sapping and vertex smoothing) has been proposed to P^2 meshes and some interesting applications stemmed from that. First, these operators enable robustness to a linear-elasticity based mesh curving process. Then, the extension of mesh optimization to degree 2 naturally gives the possibility to extend moving-mesh methods as well. This generalization has been successfully done and has opened a last opportunity: the generation of P^2 boundary layer meshes. Indeed, an already existing approach was based on a moving mesh approach and the generalization of the moving mesh method gives naturally an extension of the boundary layer mesh generation process. Examples are given all along the chapter to highlight the different applications. Finally, Chapter 6 studied the visualization issues related to high-order meshes and also their associated high-order solutions. As these elements are not linear anymore, the proper visualization of these entities is not a straightforward extension of the linear case. Using the customization possibilities offered by the `OpenGL` 4.0 graphic pipeline, a technique for the visualization of high-order meshes and solutions has been proposed. The intended approach gives a fast and CPU-light rendering of high-order curved elements and gives almost pixel-exact rendering of high-order solutions. This rendering is even pixel-exact for any high-order solution when the considered geometrical elements are linear.

Some perspectives can be drawn regarding this work. All the previous developments for high-order methods are almost exclusively done on isotropic meshes and no metric field is considered. A natural extension is therefore to deal with curvilinear mesh adaptation. Also, all the developments in mesh optimization are done in the particular case of meshes of degree two. A natural perspective would be thus to extend this approach to higher degrees. Finally, the visualization of high-order solutions through cut surfaces should be considered as it is a commonly used feature when analyzing a numerical solution. This is a challenge as the result of an intersection between a plane and a high-order mesh is difficult.

Conclusion and Perspectives

In this thesis, we have studied two alternatives to the classic and linear representation of body-fitted boundary meshes : the embedded and the high-order approach. These two alternatives stem from two different points of view that exist in the scientific computing community. The first point of view which consists in putting most of the constraints on the solver rather on the mesher, and the other one that tends to make it easier for the solver but at the cost of giving higher constraints on the mesh.

The principle of the embedded boundary method consists in simplifying the meshing process by not requiring the explicit presence in the computational mesh of the studied object. Its presence is in fact simulated later by the solver. In this thesis, we analyzed in Chapters 1 and 2 the implementation of this approach in the solver `Wolf` for the particular case of the compressible Euler equations. As `Wolf` is vertex-centered finite volume solver based on an HLLC approximate Riemann solver and a MUSCL-like gradient reconstruction, a specific implementation had to be done. The first part consisted in detecting the geometry so that the simulation of its presence could be done. As the embedded geometry was represented as a (not necessarily conform) surface mesh (*e.g.* edges in 2D and triangles in 3D), the detection was made through a mesh-mesh intersection procedure. After that, specific boundary conditions were set-up to simulate the slipping boundary condition induced by the presence of the object. These boundary conditions were coupled with a local modification of the finite volume cells by means of a cut-cell method and specific implementations were also required regarding both explicit and implicit advance in time. This implementation was done in 2D and 3D and has been validated using comparisons with their body-fitted counterparts. However, the obtained results suffered from a lack of precision, due to inappropriate sizes of the mesh that intersected the embedded geometry. For this reason, a coupling with mesh adaptation has been studied. This strategy was well established in the body-fitted case and its use for the specific case of the embedded boundary method did not require major changes in the already existing coupling approaches. The obtained results were satisfactory as it was able to recover both the embedded geometry and the appropriate physical field. Note that these results were obtained at the condition of a reliably represented (discretized) embedded geometry. Otherwise, solution artifacts appear. Nonetheless, the embedded boundary method appeared to be less interesting than expected. Indeed, the characteristics (vertex centered finite volume method) of the flow solver proved to be incompatible with the embedded boundary method in the context of embedded moving objects. This last remark drastically reduced the advantages of such a method.

For their part, high-order meshes consist in increasing the polynomial order of the elements defining the mesh, this feature being mandatory for a good behavior of a high-order flow solver. Usually, this high-order information is defined on the boundary in a first step and then propagated into the volume in a second step. However, this class of objects is not that easy to manipulate as they are not polynomial anymore.
In this thesis, we analyzed these objects. The first step was to explain in Chapter 3 a framework in order to properly manipulate these entities. The chosen framework was the one used in [George and Borouchaki 2012, Johnen et al. 2013, Abgrall et al. 2014b, George et al. 2016]. The core concept was to notice that high-order elements are also Bézier elements. Based on this analogy, several interesting properties can be derived for

high-order elements. In particular, subdivision properties of Bézier elements as well as bounding properties of Bézier functions appeared to be useful for dealing with the validity of high-order elements. Also, a discussion about the quality of high-order elements was made with the spirit of this framework. The next step was then to explore the generation of high-order meshes. As most of time the high-order induced curvature of a high-order mesh is on the boundary, it appeared natural to tackle the problem of high-order surface mesh generation in Chapter 4. Starting from a classic parametric surface mesh generation technique [Laug 2010], the idea was to provide a high-order extension to the intrinsic quantities used in the linear process. These quantities are the two radii of curvature and intervene in the expression of the gap of a surface with its tangent plane. By going further in the involved intrinsic Taylor expansion, it became possible to derive two intrinsic quantities by means of a log-simplex approach [Coulaud and Loseille 2016]. These two quantities were then interpreted as the high-order counterpart of the radii and used for a high-order parametric surface mesh generation process. The use of these high-order error estimates proved to give a more suitable point distribution for the studied degree to the high-order surface mesh than a standard linear approach.
As curving the surface mesh is not enough to generate a high-order mesh, it also appeared mandatory to understand how to deal with a high-order volume mesh. To this end, a generalization of two classic mesh optimization operators to the particular case of degree two meshes was proposed in Chapter 5. It consisted in extending the concept of vertex smoothing and generalized edge swapping. This extension was not necessarily that straightforward as a special treatment had to be done for the additional degrees of freedom brought by the degree two. With this generalization, several applications were studied. First, these operators were proved to be useful in a linear elasticity-based mesh curving as they gave a complete robustness to this curving process. Then, in the case of moving-mesh methods where the use of mesh optimization operators is common, their generalization naturally implied the extension of moving-mesh methods to degree two meshes. Finally, the generation of degree two boundary layers meshes was performed. As an already existing method was based on a moving-mesh algorithm, the possibility of doing so for the degree two appeared natural. In each of these applications, the efficiency of the proposed methods was shown on several examples.
The manipulation of high-order meshes along with high-order solutions also needs the development of dedicated visualization techniques as standard approaches fail to render them properly given the non-linearity involved with these entities. Using the interesting features available in the `OpenGL` 4.0 framework, the principle of a high-order mesh and solution visualization process was detailed in Chapter 6. With the help of several built-in functionalities of the API, we were able to tailor a process to render high-order entities with no extra cost on the CPU as all the required operations to properly display them was processed on GPU. Regarding high-order solutions, a novel approach was designed. It offered the possibility to render a high-order solution in an almost pixel-exact fashion. More precisely, the proposed rendering was pixel-exact for any high-order solution defined on a linear element and was almost pixel-exact otherwise.

Perspectives

This thesis brought some contribution to the problems aforementioned and a lot work remains to be done.

Concerning the embedded boundary method, only the particular case of the Euler equations was studied but an extension of this approach to the case of the Navier-Stokes

equations appear natural. To do so, a computation of a distance function to the geometry should be defined on the whole computational mesh and the specific treatment required by a Dirichlet boundary condition should be considered. Also, even if the mesh adaptation helps to recover almost entirely the feature of the embedded geometry, the process sometimes fails to do it perfectly. A solution to help to do so is to compute a level-set metric based on the distance between a point of the mesh and the embedded geometry [Quan et al. 2014a, Billon 2016] and to intersect it with the metric induced by the CFD solution. Finally, to avoid some problems related to the discretization of the embedded geometry, it might be beneficial to consider the case of embedded CAD and/or high-order surrogate models.

The developments performed in mesh optimization and its applications were only made in the particular case of degree two meshes. It appears mandatory to generalize it to higher degrees. This task should be more challenging as new degrees of freedom appear inside surface elements (degree ≥ 3) and volume elements (degree ≥ 4) that will surely require a specific treatment. But it is necessary as underlying applications are commonly needed with meshes of degree greater than two.

The new contributions in high-order parametric surface mesh generation were mainly done with the point of view of error estimates. However, a lot of work remains on the exploitation of the results and in particular on a more clever positioning of the high-order degrees of freedom on the high-order surface mesh. Also, an extension of the proposed framework to other type of surfaces like implicit surfaces should be considered.

The concepts exposed in high-order mesh and solution visualization open the gate to other perspectives. In particular, some interesting features regarding the visualization of solution should be analyzed using this framework, such as the rendering of isosurfaces and the rendering of a solution through a cut surface of a high-order mesh. For the case of isosurfaces, a devoted technique should be developed as all the almost pixel-exact approaches work only in the case of surface or 2D solutions. Concerning the extraction of cut surfaces, the biggest challenge is to compute in an effective fashion the intersection between a plane and high-order mesh as this result is not trivial for meshes of degree higher than one.

In the context of meshes of degree one, mesh adaptation is widely used to accelerate the convergence and improve the accuracy of numerical computations. It seems interesting and indispensable to consider it for meshes of higher degree. This subject is relatively new but some work is starting to appear on it [Aparicio-Estrems et al. 2019, Zhang et al. 2019]. The use of this technique would in particular help to improve the positioning of the high-order nodes in the case of high-order parametric mesh generation exposed before. Also, a generalization of the metric-orthogonal and metric-aligned approaches [Loseille 2014, Marcum and Alauzet 2014] to the high-order should be investigated as well.

Finally, from a more general point of view the 4D (or 3D + t) anisotropic (curvilinear or not) mesh adaptation is an interesting field to investigate regarding the meshing challenges. Indeed, some strategies exist regarding the adaptation of phenomena involving a time dependence [Barral et al. 2017], but they do not really consider the temporal variable as a real dimension. Some recent works tackled this problem [Caplan et al. 2019, Belda-Ferrín et al. 2018] but a lot of interesting research still has to be done with it.

Appendix A

Delaunay-based mesh generation

In this appendix, we recall the basic Delaunay-based mesh generation algorithms [George et al. 1991a, George and Borouchaki 1998] that are used in `GHS3D` and `Feflo.a/AMG-CADSurf` for respectively the 3D and the 2D/3D algorithm.

A.1 Delaunay triangulation

In this section, we detail what a Delaunay triangulation is and how it can be constructed. The first step consists in properly defining the used tools.

A.1.1 Definitions

Even if the term triangulation stands in the first place for triangles, it appears that this notion can be generalized beyond the two-dimensional case. In this context, the notion of simplex is introduced and from this the concept of triangulations, and in particular Delaunay ones, is detailed.

Simplex

Let d be the spatial dimension, and $\mathbb{R}^d$ being this space, and let $\mathcal{S}$ be a set $(A_i)_{1\leq i\leq m}$ of points in $\mathbb{R}^d$ then:

$$\sum_{i=1}^{m} \lambda_i A_i,$$

represents a linear combination of points in $\mathcal{S}$. These combinations of m members of $\mathcal{S}$, for $\sum_{i=1}^{m} \lambda_i = 1$, define a subspace of $\mathbb{R}^d$ denoted as the affine hull of the A_i's. If, for all i, $\lambda_i \geq 0$ such combinations are called convex. The convex hull of a finite number of points is a called a polytope and a k-polytope if the affine hull is of dimension k. The convex hull of $k+1$ points with $k \leq d$ not in an affine space of dimension $k-1$ is a particular polytope called simplex or k-simplex. A 2-simplex is simply a triangle, while a 3-simplex is a tetrahedron.

Triangulations

Let $\mathcal{S}$ be a set of points in $\mathbb{R}^d$ with $d = 2, 3$. The convex hull of $\mathcal{S}$, denoted $Conv(\mathcal{S})$, defines a domain $\Omega \in \mathbb{R}^d$. If K is a simplex, then:

Definition A.1.1 (Simplicial covering)**.** *$\mathcal{T}_r$ is a simplicial covering of Ω if the following conditions hold:*

- *(H0) The vertices of the elements in $\mathcal{T}_r$ is exactly $\mathcal{S}$.*
- *(H1) $\Omega = \cup_{K\in\mathcal{T}_r} K$.*
- *(H2) Every element K in $\mathcal{T}_r$ is non empty.*
- *(H3) The intersection of the interior of any two elements is an empty set.*

Now, we want to have something conformal, and we will be referred to as a triangulation:

Definition A.1.2 (Triangulation). *$\mathcal{T}_r$ is a conformal triangulation of Ω if $\mathcal{T}_r$ is a covering following Definition A.1.1 and if, in addition, the following condition holds:*

- *(H4) The intersection of any two elements $\mathcal{T}_r$ is either:*
  - ⋆ *the empty set,*
  - ⋆ *a vertex,*
  - ⋆ *an edge,*
  - ⋆ *a face (d=3).*

Among the different possible types of triangulations, the Delaunay is the one of interest. If we recall that $\mathcal{S}$ is a set (or a cloud) of points and that Ω is $Conv(\mathcal{S})$, then:

Definition A.1.3 (Delaunay triangulation). *$\mathcal{T}_r$ is a Delaunay triangulation of Ω if the open (balls) discs circumscribed associated with its elements are empty.*

This criterion, referred to as the *empty sphere criterion* [Delaunay 1934] means that the open balls associated with the elements do not contain any vertices. This criterion is a characterization of the Delaunay triangulation. The next step consists in constructing this type of triangulations.

A.1.2 Construction

Delaunay lemma

In [Delaunay 1934], the following result is considered as a key-issue for most of the Delaunay triangulation algorithms.

Lemma A.1.1. *Let $\mathcal{T}$ be a given arbitrary triangulation of the convex hull of a set of points of $\mathcal{S}$. If for every pairs of adjacent simplices in $\mathcal{T}$, the empty sphere criterion holds, then this criterion holds globally and $\mathcal{T}$ is a Delaunay triangulation.*

Various proofs of this lemma can be found in [George and Borouchaki 1998, Borouchaki and George 2017b] for instance.

Some topological tools

The algorithm of a Delaunay triangulation relies on several topological tools that we need to define.

Definition A.1.4 (Ball). *Let P be an element vertex, the ball associated with P is the set of elements including P as a a vertex.*

Note that the size of a ball is arbitrary.

Definition A.1.5 (Cavity). *Under some assumptions detailed later, the cavity associated with a given point P is the set of elements in $\mathcal{T}_r$, whose circumdisc/circumsphere enclose the point*

The key feature of the Delaunay algorithm relies in the definition of this set.

Incremental method

The Delaunay triangulation construction can be completed by using an incremental method also known as the Bowyer-Watson algorithm [Bowyer 1981, Watson 1981].

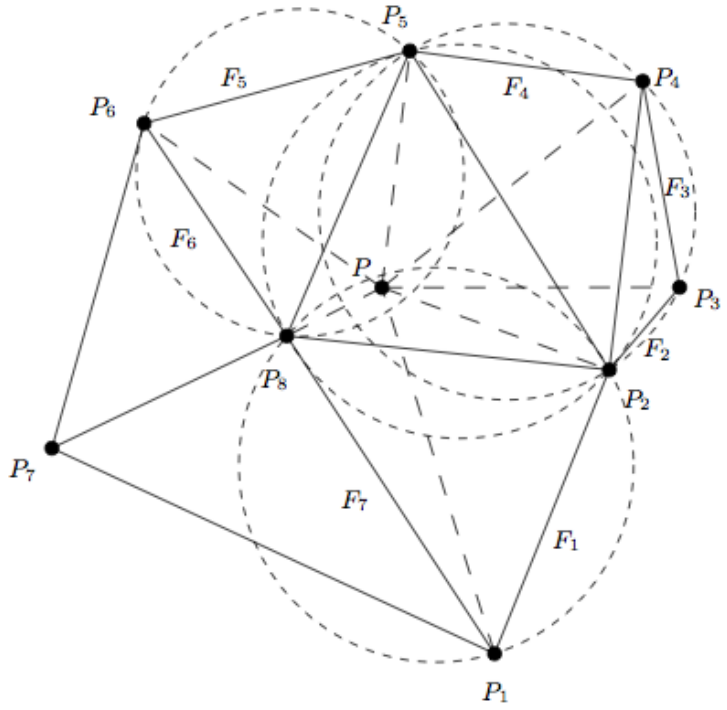

Figure A.1 – Insertion of the point P. The point P is initially located in the triangle $(P_2P_5P_8)$. The cavity is formed by the triangles traced in unbroken lines (except those sharing vertex P_7). The external faces of the cavity are denoted by F_1,F_2,...,F_7. The ball composed of the elements traced in dotted lines, is obtained by joining the point P to the faces (edges in 2D) F_i.

General methodology. Given $\mathcal{T}_i$ the Delaunay triangulation of the convex hull of the first i points in $\mathcal{S}$, we consider the $i+1^{th}$ point P of this set. The goal of the incremental method is to obtain $\mathcal{T}_{i+1}$, the Delaunay triangulation that contains P, starting from $\mathcal{T}_i$. To this end, the Delaunay kernel procedure is considered:

Delaunay kernel. This is the kernel defined by the procedure:

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}_P + \mathcal{B}_P,$$

where $\mathcal{B}_P$ is the ball of P. $\mathcal{B}_P$ is in fact the set of points formed by joining P with the external edges/faces of $\mathcal{C}_P$ the cavity of P. As we will always ensure that the triangulation $\mathcal{T}_i$ encloses P, we define th cavity as the set of elements in $\mathcal{T}_i$ whose circumdiscs/circumball contains P. The validty of this method is ensured by the following result:

Theorem A.1.1. *Given $\mathcal{T}_i$ a Delaunay triangulation of the convex hull of the first i points of a set $\mathcal{S}$, then $\mathcal{T}_{i+1}$ is a Delaunay triangulation of this hull that the $i+1$-th point P as a vertex.*

Reduced incremental method. To ensure the considered point P to be always in the considered triangulation $\mathcal{T}_i$, a box enclosing the points of $\mathcal{S}$ is created. Using this box, it is then possible to obtain a triangulation of a hull of $\mathcal{S}$. To define this box, several methods are used. A solution is to construct one simplex enclosing $\mathcal{S}$ or to construct an elementary enclosing polygon (like a square or a cube) that will be easily triangulated. Then the Delaunay kernel can be used.

Cavity correction. Theoretically, the obtained triangulation is valid. However, numerical issues in the computation of both distance and radius can lead to a wrong result. More precisely, a non-star shaped cavity is obtained, thus leading to the formation of elements with negative volume. The computed cavity becomes incorrect and the algorithm no longer works. One approach to solve this problem is to correct the cavity so that the obtained ball is valid. More precisely, for each face of the obtained cavity, the area with the considered point is computed. If the area is not satisfactory (negative or too small), the element generating this face is removed and the process is reexecuted as many as necessary. Also, if the cavity contains contains a vertex inside, an element containing this vertex is removed and the process is reexecuted as many as necessary. It can be noted that this part of the process may not be strictly Delaunay.

A.2 Mesh generation

In this section, we deal with the specific problem of mesh generation and we highlight the link of this notion with the one of constrained triangulations.

A.2.1 Definition of a mesh

Let Ω be a closed bounded domain in $\mathbb{R}^2/\mathbb{R}^3$, the question is how to construct a conforming triangulation of this domain. Such a triangulation will be referred to as a mesh of Ω and will be denoted by $\mathcal{H}$. Thus,

Definition A.2.1. *$\mathcal{H}$ is a mesh of Ω if:*

- *(H1) $\Omega = \cup_{K\in\mathcal{H}} K$.*
- *(H2) Every element $K \in \mathcal{H}$ is non empty.*
- *(H3) the intersection of the interior of any two elemnts is empty.*
- *(H4) The intersection of ant two elements in $\mathcal{H}$ is either:*
  - ⋆ *the empty set,*
  - ⋆ *a vertex,*
  - ⋆ *an edge,*
  - ⋆ *a face (d=3).*

The fundamental difference between a triangulation and a mesh is that a triangulation is a covering of the convex hull of a set of points (the latter being given) while a mesh is a covering of a given domain defined, in general, by a discretization of its boundary. From this, two problems arise:

- the enforcement, in some sense, of the boundary of the domain meaning that the triangulation is a constrained triangulation,
- the necessity of constructing the set of points which will form the vertices of the mesh since usually only the boundary points of the given boundary discretization are, in general, given as input.

A.2.2 Constrained triangulation

In this context, we are interested to the triangulations that must satisfy external constraints like a list of edges or faces. To satisfy such constraints, the method consists in enforcing the entities of the current triangulation [1] so that the constrained entities appear. In a mesh generation context, this type of triangulation appears when the considered object

1. From this point, the triangulation is usually not Delaunay anymore.

is meshed as a cloud of points inside a box. In most of the cases, the obtained triangulation does not respect the topology of the input object that must be enforced.

Two dimensional case

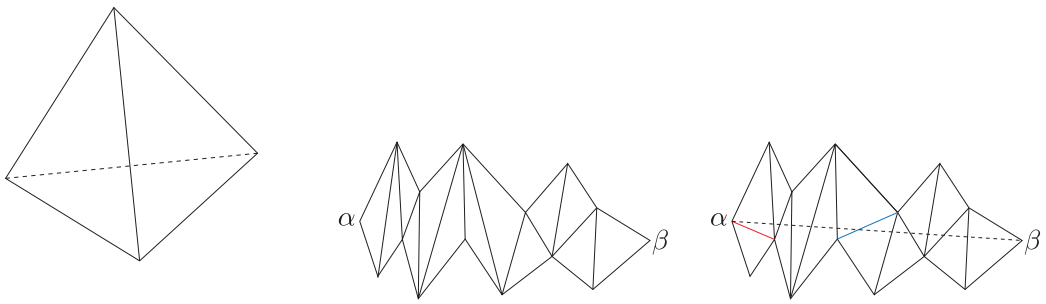

Figure A.2 – Boundary recovering operation for an edge (α, β). The edge is enforced using swaps whose principle is depicted on the left. In the middle, initial pipe. On the right, a first swap (in red) that enables to diminish the size of the problem. In blue, a swap without apparent effect.

In this case, the constrained entities are the edges. Let us introduce the following topological object:

Definition A.2.2 (Pipe)**.** *Let s be a segment enclosed in $\mathcal{T}_r$, we assume that the endpoints of s are two vertices in the triangulation. Then, in two dimensions, the pipe associated with s is the set of elements having at least one edge intersected by s.*

Then, the following result holds in 2D:

Theorem A.2.1. *There is a triangulation without internal vertex covering any domain with a non-intersecting polygonal boundary.*

Based on this theorem, we know that it is possible to enforce a boundary. The studied method is based on edge swapping. The idea is to consider an edge in the list of the constrained entities and to apply to it edge swapping. The edge swapping consist in considering an edge and the two triangles sharing this edge (*e.g.* its shell) and replacing it by the other diagonal of the quadrilateral formed by this two triangles. It can be shown that applying randomly swaps in the pipe eventually forces the constrained edge in it. The process is explained in figure A.2.

Three dimensional approach

The three dimensional case is way more complicated and does not have any theoretical proof on the success of the procedure. In fact, it is possible to create objects (like Schönhardt polyhedron) whose boundary cannot be recovered without the creation of supplementary points called *Steiner points*. However, some procedures can be used. In 3D, we consider the following objects:

Definition A.2.3 (General pipe)**.** *Let s be a segment enclosed in $\mathcal{T}_r$, we assume that the endpoints of s are two vertices in the triangulation. Then, in three dimensions, the pipe associated with s is the set of elements having at least one face or one edge intersected by s.*

Definition A.2.4 (Shell)**.** *Let a be an edge in $\mathcal{T}_r$, the shell associated with a is the set of elements sharing this edge.*

Then, the following operations are considered:

- (OP1) Create a point along an edge and remesh the corresponding shell.
- (OP2) Create a point on a face and remesh the shell formed by two elements sharing this face.
- (OP3) Create a point inside a tetrahedron and subdivide it into four elements.
- (OP4) Create two points on two faces of a tetrahedron and remesh into five tetrahedra.
- (OP5) Remove the edge defining a shell (3D edge swapping).
- (OP6) Removed the face common to two tetrahedra (3D face swapping).
- (OP7) Relocate a point (vertex smoothing).

Using these operators can lead to the creation of Steiner points but it is the price to pay to get the boundary recovery in the end. In every cases, these operators are applied to enforce the edges and the enforce the faces. Of course, the number of created Steiner points should be minimal. The boundary recovery step is the more tedious step in 3D volume meshing and is what makes the value of a 3D meshing software. The common used procedures are quite complex and can be found in references like [George and Borouchaki 1998, Frey and George 2008, Borouchaki and George 2017b] and others.

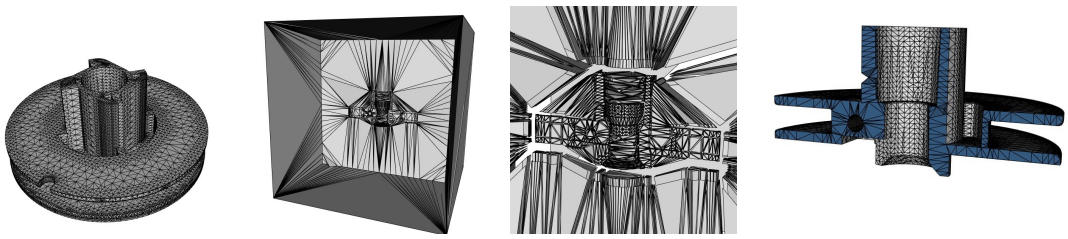

Figure A.3 – Constrained Delaunay method. From left to right, initial surface mesh, volume mesh after insertion of the surface points, volume mesh after boundary recovery and final mesh by removing element connected to the initial mesh of the surrounding box.

A.2.3 Field points generation

Once the boundary is recovered, the external part of the mesh, *e.g.* the elements between the boundary and the box as well as the elements inside the inner objects of the mesh, are removed. The procedure is based on a tag of the mesh by connected components. Then, the next step of the mesh generation process occurs: the creation of the internal points with an appropriate mesh size. As no mesh size is *a priori* known, the idea is to compute the average mesh size h of every surface mesh point and a prescribed dilution factor $\alpha \geq 1$. Then using this, all the edges of the mesh are analyzed. If the edge has a conforming size, then the edge is kept. Otherwise, one or two points (one from each extremity) are added at the wished size. Reconnection is performed and a new size is set according to the dilution factor. The process is done as long as necessary. When all edges can no longer be split, we say that the mesh is saturated.

A.2.4 Mesh optimization

The final step is to optimize the mesh. Indeed, the local size may be in agreement with the dilution factor and the boundary prescription but the resulting mesh may be of poor quality with regard to the finite element computations. For this reason, mesh optimization procedures are carried out [Brière de L'isle and George 1995]. The subject is already tackled in Section 5.2 and consists in two types of operations (that are also depicted previously as (OP5), (OP6) and (OP7)): the generalized edge swapping and the node smoothing. Once the process is applied to the mesh, it is ready to be used for numerical simulations.

Appendix B

Finite Element resolution of the Linear Elasticity Equation

In this appendix, we aim at describing the Finite Element solver as well as the underlying theory which are part `Wolf-Elast` and its depending modules.

B.1 Problem formulation

We solve the 3D *dynamic linear elasticity equation*[1] with $T > 0$ on a domain Ω of boundary $\partial\Omega = \Gamma_D \cup \Gamma_N$:

$$\begin{cases} \operatorname{div} \mathcal{S}(\mathcal{E}(\mathbf{x},t)) + \rho \mathbf{f}(\mathbf{x},t) = \rho \dfrac{\partial^2 \boldsymbol{\xi}(\mathbf{x},t)}{\partial t^2} & \forall (\mathbf{x},t) \in \Omega \times [0,T), \\ \boldsymbol{\xi}_{|\partial\Omega} = \boldsymbol{\xi}_D(\mathbf{x},t) & \forall (\mathbf{x},t) \in \Gamma_D \times [0,T), \\ \mathcal{S}(\mathcal{E}(\mathbf{x},t))_{|\partial\Omega}.\mathbf{n}_{\partial\Omega} = \mathbf{g}(\mathbf{x},t) & \forall (\mathbf{x},t) \in \Gamma_N \times [0,T), \\ \boldsymbol{\xi}(\mathbf{x},0) = \boldsymbol{\xi}_0(x) & \forall \mathbf{x} \in \Omega, \\ \dfrac{\partial \boldsymbol{\xi}(\mathbf{x},0)}{\partial t} = \boldsymbol{v}_0(x) & \forall \mathbf{x} \in \Omega, \end{cases} \tag{B.1}$$

with $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3)^T$ the Lagrangian displacement of the vertices. $\mathcal{S}$ and $\mathcal{E}$ are respectively the constraint and the deformation tensors, ρ is the material density that we will assume constant in both time and space and $\mathbf{f}$ are volume forces containing, among others, gravity. $\boldsymbol{\xi}_D$ and $\mathbf{g}$ are the spatial boundary conditions that respectively represent the displacement (Dirichlet Boundary Conditions) and the constraints (Neumann Boundary Conditions) applied on a given boundary. In Fluid-Structure Interaction problems [Vanharen et al. 2018], we usually have $\mathbf{g} = -p\mathbf{n}$ where p is the fluid pressure. $\boldsymbol{\xi}_0$ and $\boldsymbol{v}_0$ are the temporal boundary conditions. $\boldsymbol{\xi}_0$ can be seen as a solution of the static problem with the boundary conditions $\boldsymbol{\xi}_D(\mathbf{x},0)$ and $\mathbf{g}(\mathbf{x},\mathbf{0})$. The deformation tensor is defined by the compatibility relation:

$$\mathcal{E} = \frac{\nabla \boldsymbol{\xi} + \nabla \boldsymbol{\xi}^T}{2},$$

The constraint tensor follows the linear elasticity behavior law:

$$\mathcal{S}(\mathcal{E}) = \lambda \operatorname{trace}(\mathcal{E}) I_3 + 2\,\mu\,\mathcal{E}, \tag{B.2}$$

where λ and μ are the Lamé coefficients. Physically, it is very difficult to give a meaning to λ and μ. Fortunately, the Lamé's coefficients can be expressed as a function of the Young's modulus E and the Poisson ratio ν:

$$\mathrm{E} = \frac{\mu\,(3\lambda + 2\mu)}{\lambda + \mu} \qquad \text{and} \qquad \nu = \frac{\lambda}{2\,(\lambda + \mu)},$$

1. When the equation is static, $\frac{\partial^2 \boldsymbol{\xi}}{\partial t^2} = 0$, $\frac{\partial \boldsymbol{\xi}}{\partial t} = 0$, the initial conditions at $t = 0$ are not specified and Neumann boundary conditions cannot be used alone.

$$\text{or} \qquad \lambda = \frac{\mathrm{E}\,\nu}{(1+\nu)(1-2\nu)} \qquad\qquad \text{and} \qquad \mu = \frac{\mathrm{E}}{2(1+\nu)}\,.$$

We have the following properties for the above coefficients:

$$\lambda + \frac{2}{3}\mu \geq 0 \quad \text{and} \quad \mu > 0,$$
$$\mathrm{E} > 0 \quad \text{and} \quad -1 < \nu < \frac{1}{2}\,.$$

Physically, the Young's modulus is a measure of the stiffness of an isotropic elastic material. It is the ratio between the stress and the stretch. The higher the Young's modulus, the stiffer the material. For instance, the rubber has a small strain between 0.01 and 0.1 GPa (10^9 Pascal) and for the aluminum and the steel it is 69 and 200 GPa. For the diamond the value is 1220 GPa.
As regards the Poisson ratio, it is the ratio, when a sample object is stretched, of the contraction or transverse strain (perpendicular to the applied load), to the extension or axial strain (in the direction of the applied load). When a sample cube of a material is stretched in one direction, it tends to contract (or occasionally, expand) in the other two directions perpendicular to the direction of stretch. Conversely, when a sample of material is compressed in one direction, it tends to expand (or rarely, contract) in the other two directions. This phenomenon is called the Poisson effect. Poisson's ratio ν is a measure of the Poisson effect. A perfectly incompressible material deformed elastically at small strains will have a Poisson's ratio of exactly 0.5, for instance rubber. Most steels when used within their design limits (before yield) exhibit values of about 0.3, increasing to 0.5 for post-yield deformation (which occurs largely at constant volume.)

B.2 Variational Form

We note:

$$\begin{aligned}
&H^1(\Omega) && = \{u \in L^2(\Omega) \text{ such that } \nabla_{\boldsymbol{x}} u \in L^2(\Omega) \text{ in the distribution sense }\},\\
&V_{\boldsymbol{\xi}_D} && = \{u \in H^1(\Omega) \text{ such that } u = \boldsymbol{\xi}_D \text{ on } \Gamma_D\},\\
&C^0(0,T;(V_{\boldsymbol{\xi}_D})^d) && = \{u(\mathbf{x},t) \text{ with } (\mathbf{x},t) \in \Omega \times [0,T) \text{ such that}\\
&&& \quad u(\mathbf{x},.) \in (V_{\boldsymbol{\xi}_D})^d \text{ and } u(.,t) \in C^0([0,T))]\},\\
&C^1(0,T;(L^2(\Omega))^d) && = \{u(\mathbf{x},t) \text{ with } (\mathbf{x},t) \in \Omega \times [0,T) \text{ such that}\\
&&& \quad u(\mathbf{x},.) \in (L^2(\Omega))^d \text{ and } u(.,t) \in C^1([0,T))]\},\\
&V && = \{u \in H^1(\Omega) \text{ such that } u = 0 \text{ on } \Gamma_D\}.
\end{aligned}$$

Using these functional spaces, the variational form of the elasticity problem can be written as:

$$\begin{aligned}
&\text{Find } \boldsymbol{\xi} \in C^0(0,T;(V_{\boldsymbol{\xi}_D})^d) \cap C^1(0,T;(L^2(\Omega))^d)\,,\\
&\int_\Omega \rho \frac{\partial^2 \boldsymbol{\xi}}{\partial t^2} \boldsymbol{v}\,\mathrm{d}\Omega + \int_\Omega \mathcal{S}(\mathcal{E}) : \nabla \boldsymbol{v}\,\mathrm{d}\Omega = \int_\Omega \rho \mathbf{f} \boldsymbol{v}\,\mathrm{d}\Omega + \int_{\Gamma_N} \mathbf{g}\boldsymbol{v}\,\mathrm{d}\Gamma\,, \quad \forall \boldsymbol{v} \in (V)^d,\\
&\boldsymbol{\xi}(\mathbf{x},0) = \boldsymbol{\xi}_0(x)\,, \frac{\partial \boldsymbol{\xi}(\mathbf{x},0)}{\partial t} = \boldsymbol{v}_0(x), \qquad \forall \mathbf{x} \in \Omega.
\end{aligned} \tag{B.3}$$

where the operator ":" is understood as $A : B = \sum_{i,j} a_{ij}\, b_{ij}$. Under regularity assumptions on $\boldsymbol{\xi}_0$, $\boldsymbol{v}_0$, $\boldsymbol{\xi}_D$ and $\mathbf{g}$, it can be shown that the problem has a unique solution in space and time [Bécache et al. 2010]. The resolution of this equation is done in two steps. The spatial part is solved using the Finite Element Method with isoparametric P^n-Lagrange finite elements and then the temporal part is solved using the Finite Difference Method with a Newmark scheme.

B.3 FEM discretization

In this section, we will only consider the spatial part of the equation (*e.g.* we consider the equation for a fixed t). In the following sections, $\frac{\partial^2 \boldsymbol{\xi}}{\partial t^2}$ will be noted as $\ddot{\boldsymbol{\xi}}$.
In the following, we choose a polynomial approximation space:

$$V_h = \{u \in V \cap C^0(\Omega) \text{ such that } u_{|_K} \in P^n \, \forall K \in \mathcal{H}\},$$

where P^n is the set of polynomials of degree n and we set $s = \lambda + 2\mu$.

B.4 2D case

Let us first deal with the two dimensional case. Under matrix form, we have:

$$\mathcal{S}(\mathcal{E}) = \begin{pmatrix} \lambda(\frac{\partial \xi_1}{\partial x} + \frac{\partial \xi_2}{\partial y}) + 2\mu \frac{\partial \xi_1}{\partial x} & \mu(\frac{\partial \xi_1}{\partial y} + \frac{\partial \xi_2}{\partial x}) \\ \mu(\frac{\partial \xi_1}{\partial y} + \frac{\partial \xi_2}{\partial x}) & \lambda(\frac{\partial \xi_1}{\partial x} + \frac{\partial \xi_2}{\partial y}) + 2\mu \frac{\partial \xi_2}{\partial y} \end{pmatrix}.$$

The discrete variational formulation reads:

$$\text{Find } \boldsymbol{\xi}_h = (\xi_1, \xi_2)^T \in V_h^2 \text{ such as, } \forall \boldsymbol{v}_h = (v_1, v_2)^T \in V_h^2:$$

$$\begin{aligned} &\int_{\Omega_h} \left(s\frac{\partial \xi_1}{\partial x} + \lambda \frac{\partial \xi_2}{\partial y} \right) \frac{\partial v_1}{\partial x} + \mu \left(\frac{\partial \xi_1}{\partial y} + \frac{\partial \xi_2}{\partial x} \right) \frac{\partial v_1}{\partial y} \\ &+ \mu \left(\frac{\partial \xi_1}{\partial y} + \frac{\partial \xi_2}{\partial x} \right) \frac{\partial v_2}{\partial x} + \left(\lambda \frac{\partial \xi_1}{\partial x} + s\frac{\partial \xi_2}{\partial y} \right) \frac{\partial v_2}{\partial y} \\ &= \int_{\Omega_h} \rho(-\ddot{\boldsymbol{\xi}}_h + \mathbf{f}_h)\boldsymbol{v}_h + \int_{\Gamma_{N,h}} \mathbf{g}_h \boldsymbol{v}_h \, , \end{aligned}$$

From now on, we note $\xi_{J,k}$ the unknown value of the k^{th} component of displacement $\boldsymbol{\xi}_h$ at vertex P_J. We then decompose each component of the displacement $\boldsymbol{\xi}_h$ on the finite element basis:

$$\boldsymbol{\xi}_h(\mathbf{x}) = \left(\sum_J \xi_{J,1}\phi_J(\mathbf{x}) \, , \, \sum_J \xi_{J,2}\phi_J(\mathbf{x}) \right)^T \quad \text{and} \quad \frac{\partial \boldsymbol{\xi}_h}{\partial x_k}(\mathbf{x}) = \left(\sum_J \xi_{J,1} \frac{\partial \phi_J}{\partial x_k}(\mathbf{x}) \, , \, \sum_J \xi_{J,2} \frac{\partial \phi_J}{\partial x_k}(\mathbf{x}) \right)^T ,$$

The same goes for $\ddot{\boldsymbol{\xi}}_h$, $\mathbf{f}_h$ and $\mathbf{g}_h$.
We obtain the following discrete approximate problem:

$$\text{Find } \Xi = \left\{ (\xi_{I,1}, \xi_{I,2})^T \right\}_{I=1\ldots N} \text{ such that, } \forall \boldsymbol{v}_h = (v_1, v_2)^T \in V_h^2:$$

$$\begin{aligned} & \sum_J \xi_{J,1} \int_{\Omega_h} \left(s\frac{\partial \phi_J}{\partial x}\frac{\partial v_1}{\partial x} + \mu \frac{\partial \phi_J}{\partial y}\frac{\partial v_1}{\partial y} + \mu \frac{\partial \phi_J}{\partial y}\frac{\partial v_2}{\partial x} + \lambda \frac{\partial \phi_J}{\partial x}\frac{\partial v_2}{\partial y} \right) \\ + & \sum_J \xi_{J,2} \int_{\Omega_h} \left(\lambda \frac{\partial \phi_J}{\partial y}\frac{\partial v_1}{\partial x} + \mu \frac{\partial \phi_J}{\partial x}\frac{\partial v_1}{\partial y} + \mu \frac{\partial \phi_J}{\partial x}\frac{\partial v_2}{\partial x} + s\frac{\partial \phi_J}{\partial y}\frac{\partial v_2}{\partial y} \right) \\ = & \int_{\Omega_h} \rho(-\ddot{\boldsymbol{\xi}}_h + \mathbf{f}_h)\boldsymbol{v}_h + \int_{\Gamma_{N,h}} \mathbf{g}_h \boldsymbol{v}_h . \end{aligned}$$

We now split the discrete variational equation into the following system by choosing respectively as test function $\boldsymbol{v} = (\phi_I, 0)^T$ and $\boldsymbol{v} = (0, \phi_I)^T$:

$$\sum_J \xi_{J,1} \int_{\Omega_h} \left(s\frac{\partial \phi_J}{\partial x}\frac{\partial \phi_I}{\partial x} + \mu \frac{\partial \phi_J}{\partial y}\frac{\partial \phi_I}{\partial y} \right) + \qquad \sum_J \xi_{J,2} \int_{\Omega_h} \left(\lambda \frac{\partial \phi_J}{\partial y}\frac{\partial \phi_I}{\partial x} + \mu \frac{\partial \phi_J}{\partial x}\frac{\partial \phi_I}{\partial y} \right)$$

$$
\begin{aligned}
&= \sum_J (-\ddot{\xi}_{J,1} + f_{J,1}) \int_{\Omega_h} \rho \phi_J \phi_I + \qquad \sum_J g_{J,2} \int_{\Gamma_{N,h}} \phi_J \phi_I \,, \quad \text{for } (\phi_I, 0)^T \in V_h^2, \\
\sum_J \xi_{J,1} \int_{\Omega_h} \left(\mu \frac{\partial \phi_J}{\partial y} \frac{\partial \phi_I}{\partial x} + \lambda \frac{\partial \phi_J}{\partial x} \frac{\partial \phi_I}{\partial y} \right) + &\qquad \sum_J \xi_{J,2} \int_{\Omega_h} \left(\mu \frac{\partial \phi_J}{\partial x} \frac{\partial \phi_I}{\partial x} + s \frac{\partial \phi_J}{\partial y} \frac{\partial \phi_I}{\partial y} \right) \\
&= \sum_J (-\ddot{\xi}_{J,2} + f_{J,2}) \int_{\Omega_h} \rho \phi_J \phi_I + \qquad \sum_J g_{J,2} \int_{\Gamma_{N,h}} \phi_J \phi_I \,, \quad \text{for } (0, \phi_I)^T \in V_h^2.
\end{aligned}
$$

Therefore, we obtain the following linear system:

$$\mathbb{K}\Xi = -\mathbb{M}\ddot{\Xi} + \mathbb{M}F + \mathbb{S}G \,,$$

where $\mathbb{K}$ is called the stiffness matrix and is a block matrix having 2×2 block $\mathbb{K}_{IJ}$ at block indices (I, J) given by:

$$
\mathbb{K}_{IJ} = \begin{pmatrix}
\displaystyle\int_{\Omega_h} s \frac{\partial \phi_J}{\partial x} \frac{\partial \phi_I}{\partial x} + \mu \frac{\partial \phi_J}{\partial y} \frac{\partial \phi_I}{\partial y} & \displaystyle\int_{\Omega_h} \lambda \frac{\partial \phi_J}{\partial y} \frac{\partial \phi_I}{\partial x} + \mu \frac{\partial \phi_J}{\partial x} \frac{\partial \phi_I}{\partial y} \\
\displaystyle\int_{\Omega_h} \mu \frac{\partial \phi_J}{\partial y} \frac{\partial \phi_I}{\partial x} + \lambda \frac{\partial \phi_J}{\partial x} \frac{\partial \phi_I}{\partial y} & \displaystyle\int_{\Omega_h} \mu \frac{\partial \phi_J}{\partial x} \frac{\partial \phi_I}{\partial x} + s \frac{\partial \phi_J}{\partial y} \frac{\partial \phi_I}{\partial y}
\end{pmatrix},
$$

$\mathbb{M}$ is called the mass matrix and is a block matrix having 2×2 block $\mathbb{M}_{IJ}$ at block indices (I, J) given by:

$$
\mathbb{M}_{IJ} = \begin{pmatrix}
\displaystyle\int_{\Omega_h} \rho \phi_I \phi_J & 0 \\
0 & \displaystyle\int_{\Omega_h} \rho \phi_I \phi_J
\end{pmatrix},
$$

and $\mathbb{S}$ is called the surface mass matrix (as it is acting like a mass matrix in a lower dimension of one) and is a block matrix having 2×2 block $\mathbb{S}_{IJ}$ at block indices (I, J) given by:

$$
\mathbb{S}_{IJ} = \begin{pmatrix}
\displaystyle\int_{\Gamma_{N,h}} \phi_I \phi_J & 0 \\
0 & \displaystyle\int_{\Gamma_{N,h}} \phi_I \phi_J
\end{pmatrix}.
$$

The system can therefore be seen as:

$$
\begin{pmatrix}
\mathbb{K}_{11} & \vdots & & \\
 & \vdots & & \\
\dots & \mathbb{K}_{IJ} & \dots & \dots \\
 & \vdots & &
\end{pmatrix}
\begin{pmatrix}
\xi_{1,1} \\ \xi_{1,2} \\ \vdots \\ \xi_{J,1} \\ \xi_{J,2} \\ \vdots
\end{pmatrix} =
$$

$$
\begin{pmatrix}
\mathbb{M}_{11} & \vdots & & \\
 & \vdots & & \\
\dots & \mathbb{M}_{IJ} & \dots & \dots \\
 & \vdots & &
\end{pmatrix}
\begin{pmatrix}
-\ddot{\xi}_{1,1} + f_{1,1} \\ -\ddot{\xi}_{1,2} + f_{1,2} \\ \vdots \\ -\ddot{\xi}_{J,1} + f_{J,1} \\ -\ddot{\xi}_{J,2} + f_{J,2} \\ \vdots
\end{pmatrix} +
\begin{pmatrix}
\mathbb{S}_{11} & \vdots & & \\
 & \vdots & & \\
\dots & \mathbb{S}_{IJ} & \dots & \dots \\
 & \vdots & &
\end{pmatrix}
\begin{pmatrix}
g_{1,1} \\ g_{1,2} \\ \vdots \\ g_{J,1} \\ g_{J,2} \\ \vdots
\end{pmatrix}.
$$

B.5 3D case

The strain tensor reads:

$$\mathcal{S}(\mathcal{E}) = \begin{pmatrix} s\,\frac{\partial \xi_1}{\partial x} + \lambda\,\frac{\partial \xi_2}{\partial y} + \lambda\,\frac{\partial \xi_3}{\partial z} & \mu\,(\frac{\partial \xi_1}{\partial y} + \frac{\partial \xi_2}{\partial x}) & \mu\,(\frac{\partial \xi_1}{\partial z} + \frac{\partial \xi_3}{\partial x}) \\ \mu\,(\frac{\partial \xi_1}{\partial y} + \frac{\partial \xi_2}{\partial x}) & \lambda\,\frac{\partial \xi_1}{\partial x} + s\,\frac{\partial \xi_2}{\partial y} + \lambda\,\frac{\partial \xi_3}{\partial z} & \mu\,(\frac{\partial \xi_2}{\partial z} + \frac{\partial \xi_3}{\partial y}) \\ \mu\,(\frac{\partial \xi_1}{\partial y} + \frac{\partial \xi_2}{\partial x}) & \mu\,(\frac{\partial \xi_2}{\partial z} + \frac{\partial \xi_3}{\partial y}) & \lambda\,\frac{\partial \xi_1}{\partial x} + \lambda\,\frac{\partial \xi_2}{\partial y} + s\,\frac{\partial \xi_3}{\partial z} \end{pmatrix}.$$

Then, the variational formulation is given by:

Find $\boldsymbol{\xi}_h = (\xi_1, \xi_2, \xi_3)^T \in V_h^3$ such that, $\forall \boldsymbol{v}_h = (v_1, v_2, v_3)^T \in V_h^3$:

$$\int\limits_{\Omega_h} \begin{pmatrix} s\,\frac{\partial \xi_1}{\partial x} + \lambda\,\frac{\partial \xi_2}{\partial y} + \lambda\,\frac{\partial \xi_3}{\partial z} & \mu\,(\frac{\partial \xi_1}{\partial y} + \frac{\partial \xi_2}{\partial x}) & \mu\,(\frac{\partial \xi_1}{\partial z} + \frac{\partial \xi_3}{\partial x}) \\ \mu\,(\frac{\partial \xi_1}{\partial y} + \frac{\partial \xi_2}{\partial x}) & \lambda\,\frac{\partial \xi_1}{\partial x} + s\,\frac{\partial \xi_2}{\partial y} + \lambda\,\frac{\partial \xi_3}{\partial z} & \mu\,(\frac{\partial \xi_2}{\partial z} + \frac{\partial \xi_3}{\partial y}) \\ \mu\,(\frac{\partial \xi_1}{\partial y} + \frac{\partial \xi_2}{\partial x}) & \mu\,(\frac{\partial \xi_2}{\partial z} + \frac{\partial \xi_3}{\partial y}) & \lambda\,\frac{\partial \xi_1}{\partial x} + \lambda\,\frac{\partial \xi_2}{\partial y} + s\,\frac{\partial \xi_3}{\partial z} \end{pmatrix} : \begin{pmatrix} \frac{\partial v_1}{\partial x} & \frac{\partial v_1}{\partial y} & \frac{\partial v_1}{\partial z} \\ \frac{\partial v_2}{\partial x} & \frac{\partial v_2}{\partial y} & \frac{\partial v_2}{\partial z} \\ \frac{\partial v_3}{\partial x} & \frac{\partial v_3}{\partial y} & \frac{\partial v_3}{\partial z} \end{pmatrix}$$
$$= \int\limits_{\Omega_h} \rho(-\ddot{\boldsymbol{\xi}}_h + \mathbf{f}_h)\boldsymbol{v}_h + \int\limits_{\Gamma_{N,h}} \mathbf{g}_h \boldsymbol{v}_h. \tag{B.4}$$

By developing the above expression and reorganizing the different terms, we get:

$$\begin{aligned} & \int s\tfrac{\partial \xi_1}{\partial x}\tfrac{\partial v_1}{\partial x} + \int \mu\tfrac{\partial \xi_1}{\partial y}\tfrac{\partial v_1}{\partial y} + \int \mu\tfrac{\partial \xi_1}{\partial z}\tfrac{\partial v_1}{\partial z} && + \int \mu\tfrac{\partial \xi_1}{\partial y}\tfrac{\partial v_2}{\partial x} + \int \lambda\tfrac{\partial \xi_1}{\partial x}\tfrac{\partial v_2}{\partial y} && + \int \mu\tfrac{\partial \xi_1}{\partial z}\tfrac{\partial v_3}{\partial x} + \int \lambda\tfrac{\partial \xi_1}{\partial x}\tfrac{\partial v_3}{\partial z} \\ + \quad & \int \lambda\tfrac{\partial \xi_2}{\partial y}\tfrac{\partial v_1}{\partial x} + \int \mu\tfrac{\partial \xi_2}{\partial x}\tfrac{\partial v_1}{\partial y} && + \int \mu\tfrac{\partial \xi_2}{\partial x}\tfrac{\partial v_2}{\partial x} + \int s\tfrac{\partial \xi_2}{\partial y}\tfrac{\partial v_2}{\partial y} + \int \mu\tfrac{\partial \xi_2}{\partial z}\tfrac{\partial v_2}{\partial z} && + \int \mu\tfrac{\partial \xi_2}{\partial z}\tfrac{\partial v_3}{\partial y} + \int \lambda\tfrac{\partial \xi_2}{\partial y}\tfrac{\partial v_3}{\partial z} \\ + \quad & \int \mu\tfrac{\partial \xi_2}{\partial z}\tfrac{\partial v_1}{\partial y} + \int \lambda\tfrac{\partial \xi_2}{\partial y}\tfrac{\partial v_1}{\partial z} && + \int \lambda\tfrac{\partial \xi_3}{\partial z}\tfrac{\partial v_2}{\partial y} + \int \mu\tfrac{\partial \xi_3}{\partial y}\tfrac{\partial v_2}{\partial z} && + \int \mu\tfrac{\partial \xi_3}{\partial x}\tfrac{\partial v_3}{\partial x} + \int \mu\tfrac{\partial \xi_3}{\partial y}\tfrac{\partial v_3}{\partial y} + \int s\tfrac{\partial \xi_3}{\partial z}\tfrac{\partial v_3}{\partial z} \\ = \quad & \int\limits_{\Omega_h} \rho(-\ddot{\boldsymbol{\xi}}_h + \mathbf{f}_h)\boldsymbol{v}_h && + \int\limits_{\Gamma_{N,h}} \mathbf{g}_h \boldsymbol{v}_h, \quad \forall \boldsymbol{v}_h = (v_1, v_2, v_3) \in V_h^3. \end{aligned}$$

From now on, we note $\xi_{J,k}$ the unknown value of the k^{th} component of displacement $\boldsymbol{\xi}_h$ at vertex P_J. We now decompose each component of the displacement $\boldsymbol{\xi}_h$ on the finite element basis:

$$\begin{aligned} \boldsymbol{\xi}_h(\mathbf{x}) &= \left(\sum_J \xi_{J,1}\phi_J(\mathbf{x}) \,,\, \sum_J \xi_{J,2}\phi_J(\mathbf{x}) \,,\, \sum_J \xi_{J,3}\phi_J(\mathbf{x}) \right)^T, \\ \ddot{\boldsymbol{\xi}}_h(\mathbf{x}) &= \left(\sum_J \ddot{\xi}_{J,1}\phi_J(\mathbf{x}) \,,\, \sum_J \ddot{\xi}_{J,2}\phi_J(\mathbf{x}) \,,\, \sum_J \ddot{\xi}_{J,3}\phi_J(\mathbf{x}) \right)^T, \\ \mathbf{f}_h(\mathbf{x}) &= \left(\sum_J f_{J,1}\phi_J(\mathbf{x}) \,,\, \sum_J f_{J,2}\phi_J(\mathbf{x}) \,,\, \sum_J f_{J,3}\phi_J(\mathbf{x}) \right)^T, \\ \mathbf{g}_h(\mathbf{x}) &= \left(\sum_J g_{J,1}\phi_J(\mathbf{x}) \,,\, \sum_J g_{J,2}\phi_J(\mathbf{x}) \,,\, \sum_J g_{J,3}\phi_J(\mathbf{x}) \right)^T. \end{aligned}$$

We then reintroduce this decomposition in B.4:

$$\sum_J \xi_{J,1}\left(\int s\frac{\partial\phi_J}{\partial x}\frac{\partial v_1}{\partial x}+\int\mu\frac{\partial\phi_J}{\partial y}\frac{\partial v_1}{\partial y}+\int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial v_1}{\partial z}+\int\mu\frac{\partial\phi_J}{\partial y}\frac{\partial v_2}{\partial x}+\int\lambda\frac{\partial\phi_J}{\partial x}\frac{\partial v_2}{\partial y}+\int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial v_3}{\partial x}+\int\lambda\frac{\partial\phi_J}{\partial x}\frac{\partial v_3}{\partial z}\right)$$

$$+\sum_J \xi_{J,2}\left(\int\lambda\frac{\partial\phi_J}{\partial y}\frac{\partial v_1}{\partial x}+\int\mu\frac{\partial\phi_J}{\partial x}\frac{\partial v_1}{\partial y}+\int\mu\frac{\partial\phi_J}{\partial x}\frac{\partial v_2}{\partial x}+\int s\frac{\partial\phi_J}{\partial y}\frac{\partial v_2}{\partial y}+\int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial v_2}{\partial z}+\int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial v_3}{\partial y}+\int\lambda\frac{\partial\phi_J}{\partial y}\frac{\partial v_3}{\partial z}\right)$$

$$+\sum_J \xi_{J,3}\left(\int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial v_1}{\partial y}+\int\lambda\frac{\partial\phi_J}{\partial y}\frac{\partial v_1}{\partial z}+\int\lambda\frac{\partial\phi_J}{\partial z}\frac{\partial v_2}{\partial y}+\int\mu\frac{\partial\phi_J}{\partial y}\frac{\partial v_2}{\partial z}+\int\mu\frac{\partial\phi_J}{\partial x}\frac{\partial v_3}{\partial x}+\int\mu\frac{\partial\phi_J}{\partial y}\frac{\partial v_3}{\partial y}+\int s\frac{\partial\phi_J}{\partial z}\frac{\partial v_3}{\partial z}\right)$$

$$=\sum_J(-\ddot{\xi}_{J,1}+f_{J,1})\int_{\Omega_h}\rho\phi_J v_1+\sum_J(-\ddot{\xi}_{J,2}+f_{J,2})\int_{\Omega_h}\rho\phi_J v_2+\sum_J(-\ddot{\xi}_{J,3}+f_{J,3})\int_{\Omega_h}\rho\phi_J v_3$$

$$+\sum_J g_{J,1}\int_{\Gamma_{N,h}}\phi_J v_1+\sum_J g_{J,2}\int_{\Gamma_{N,h}}\phi_J v_2+\sum_J g_{J,3}\int_{\Gamma_{N,h}}\phi_J v_3,\quad \forall \boldsymbol{v}_h=(v_1,v_2,v_3)\in V_h^3.$$

We can now split this formulation in three different parts corresponding to the 3 lines of the linear system $\mathbb{K}\Xi = F$ associated with the three composants of vertex I. This is done by chosing as test function $\boldsymbol{v_h}=(\phi_I,0,0)$, $\boldsymbol{v_h}=(0,\phi_I,0)$ and $\boldsymbol{v_h}=(0,0,\phi_I)$ respectively. We get:

$$\begin{aligned}&\sum_J \xi_{J,1}\left(\int s\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial x}+\int\mu\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial y}+\int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial z}\right) &&+\sum_J \xi_{J,2}\left(\int\lambda\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial x}+\int\mu\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial y}\right)\\ &+\sum_J \xi_{J,3}\left(\int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial x}+\int\lambda\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial z}\right) &&=\sum_J(-\ddot{\xi}_{J,1}+f_{J,1})\int_{\Omega_h}\rho\phi_J\phi_I+\sum_J g_{J,1}\int_{\Gamma_{N,h}}\phi_J\phi_I,\\ &\sum_J \xi_{J,1}\left(\int\mu\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial x}+\int\lambda\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial y}\right) &&+\sum_J \xi_{J,2}\left(\int\mu\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial x}+\int s\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial y}+\int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial z}\right)\\ &+\sum_J \xi_{J,3}\left(\int\lambda\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial y}+\int\mu\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial z}\right) &&=\sum_J(-\ddot{\xi}_{J,2}+f_{J,2})\int_{\Omega_h}\rho\phi_J\phi_I+\sum_J g_{J,2}\int_{\Gamma_{N,h}}\phi_J\phi_I,\\ &\sum_J \xi_{J,1}\left(\int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial x}+\int\lambda\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial z}\right) &&+\sum_J \xi_{J,2}\left(\int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial y}+\int\lambda\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial z}\right)\\ &+\sum_J \xi_{J,3}\left(\int\mu\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial x}+\int\mu\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial y}+\int s\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial z}\right) &&=\sum_J(-\ddot{\xi}_{J,3}+f_{J,3})\int_{\Omega_h}\rho\phi_J\phi_I+\sum_J g_{J,3}\int_{\Gamma_{N,h}}\phi_J\phi_I.\end{aligned}$$

Therefore, we obtain the following linear system:

$$\mathbb{K}\Xi=-\mathbb{M}\ddot{\Xi}+\mathbb{M}F+\mathbb{S}G\,,$$

where $\mathbb{K}$ is called the stiffness matrix and is a block matrix having 3×3 block $\mathbb{K}_{IJ}$ at block indices (I,J) given by:

$$\mathbb{K}_{IJ}=\begin{pmatrix}\int s\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial x}+\mu\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial y}+\mu\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial z} & \int\lambda\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial x}+\mu\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial y} & \int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial x}+\lambda\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial z}\\ \int\mu\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial x}+\lambda\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial y} & \int\mu\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial x}+s\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial y}+\mu\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial z} & \int\lambda\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial y}+\mu\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial z}\\ \int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial x}+\lambda\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial z} & \int\mu\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial y}+\lambda\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial z} & \int\mu\frac{\partial\phi_J}{\partial x}\frac{\partial\phi_I}{\partial x}+\mu\frac{\partial\phi_J}{\partial y}\frac{\partial\phi_I}{\partial y}+s\frac{\partial\phi_J}{\partial z}\frac{\partial\phi_I}{\partial z}\end{pmatrix}.$$

$\mathbb{M}$ is called the mass matrix and is a block matrix having 3×3 block $\mathbb{M}_{IJ}$ at block indices (I,J) given by:

$$\mathbb{M}_{IJ}=\begin{pmatrix}\int_{\Omega_h}\rho\phi_I\phi_J & 0 & 0\\ 0 & \int_{\Omega_h}\rho\phi_I\phi_J & 0\\ 0 & 0 & \int_{\Omega_h}\rho\phi_I\phi_J\end{pmatrix},$$

and $\mathbb{S}$ is called the surface mass matrix (as it is acting like a mass matrix in a lower dimension of one) and is a block matrix having 3×3 block $\mathbb{S}_{IJ}$ at block indices (I,J)

given by:

$$\mathbb{S}_{IJ} = \begin{pmatrix} \int_{\Gamma_{N,h}} \phi_I \phi_J & 0 & 0 \\ 0 & \int_{\Gamma_{N,h}} \phi_I \phi_J & 0 \\ 0 & 0 & \int_{\Gamma_{N,h}} \phi_I \phi_J \end{pmatrix}.$$

B.6 Computation of mass and stiffness matrices

Now, let us detail how to compute the terms of a block $\mathbb{K}_{IJ}$ and $\mathbb{M}_{IJ}$ (or likewise $\mathbb{S}_{IJ}$ in a lower dimension of one). classicly, in the FEM method, the shape functions are defined by element as polynomial functions. As the shape functions are designed such that a vertex only interacts with its first neighbors (i.e the vertices and nodes of its ball), the matrix assembly is done in one shot with a loop on elements. For example, this means that the term $\int_{\Omega_h} s \frac{\partial \phi_J}{\partial x} \frac{\partial \phi_I}{\partial x}$ in the $\mathbb{K}_{IJ}$ matrix is computed as:

$$\int_{\Omega_h} s \frac{\partial \phi_J}{\partial x} \frac{\partial \phi_I}{\partial x} = \sum_{K \in \Omega_h} \int_K s \frac{\partial \phi_J}{\partial x} \frac{\partial \phi_I}{\partial x} = \sum_{K \ni P_I} \int_K s \frac{\partial \phi_J}{\partial x} \frac{\partial \phi_I}{\partial x}.$$

Likewise, a term $\int_{\Omega_h} \rho\, \phi_I \phi_J$ in the $\mathbb{M}_{IJ}$ matrix is computed as:

$$\int_{\Omega_h} \rho\, \phi_I \phi_J = \sum_{K \in \Omega_h} \int_K \rho\, \phi_I \phi_J = \sum_{K \ni P_I} \int_K \rho\, \phi_I \phi_J.$$

The only difference between $\mathbb{M}$ and $\mathbb{S}$ (apart from the presence of ρ) lies in the fact that if $\mathbb{M}$ is computed on tetrahedra (resp. triangles) then $\mathbb{S}$ is computed on triangles (resp. edges).
An example in P^1 of the contribution of a triangle and a tetrahedron is given in Figure B.1.

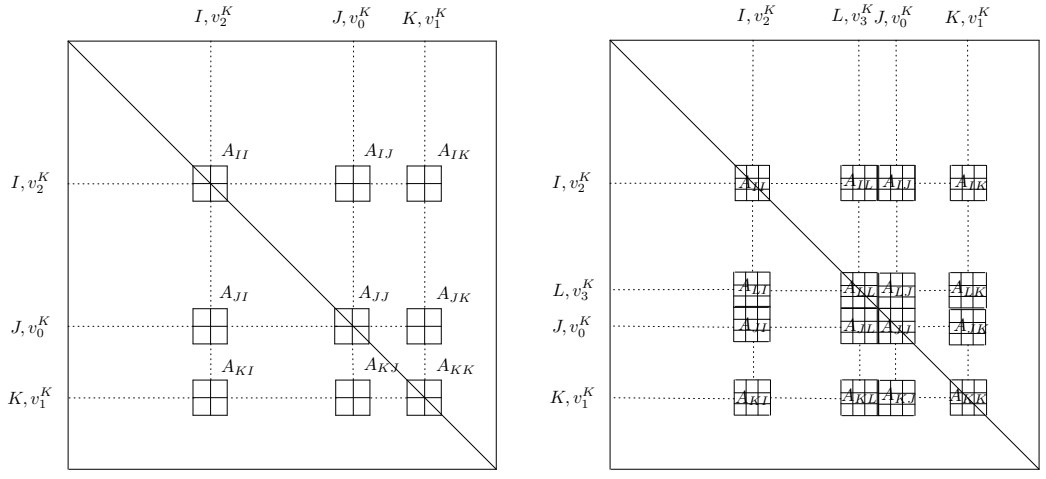

Figure B.1 – Contribution of a triangle and a tetrahedron to a given matrix A for a P^1 finite element resolution of the linear elasticity equation.

Note that the elasticity matrices are symmetric and that in P^1 a triangle contributes to

$9 \times 4 = 36$ boxes of the matrix (but we only need to fill $3 \times 4 + 3 \times 3 = 21$ boxes as the matrices are symmetric) and a tetrahedron contributes to $16 \times 9 = 144$ boxes of the matrix (but we only need to fill $6 \times 9 + 4 \times 6 = 78$ boxes as the matrices are symmetric).
For the following analysis, as λ, μ, s and ρ are constants, we will omit them in the computation of the integrals.

B.6.1 Integration on the reference element

In the case of a P^n approximation, we use a mapping F_K from the reference element $\hat{K}$ onto the current element to compute the integrals on K in $\mathbb{K}_{IJ}$ and $\mathbb{M}_{IJ}$. Depending on the dimension, the chosen reference element is:

- In 1D, the reference edge is chosen to be the one having as vertices: $\hat{P}_0 = 0$ and $\hat{P}_1 = 1$.
- In 2D, the reference triangle is chosen to be the one having as vertices: $\hat{P}_0 = (0,0)$, $\hat{P}_1 = (1,0)$ and $\hat{P}_2 = (0,1)$.
- In 3D, the reference tetrahedron is chosen to be the one having as vertices $\hat{P}_0 = (0,0,0)$, $\hat{P}_1 = (1,0,0)$, $\hat{P}_2 = (0,1,0)$ and $\hat{P}_3 = (0,0,1)$.

We note with a hat the quantities defined on the reference triangle and B_K the jacobian matrix of the mapping. If $f = f(x)$, then we change the variable x into $\hat{x}$ thanks to the mapping and we thus have the differentiation formula:

$$\nabla_{\hat{x}} \hat{f}(\hat{x}) = B_K^T \nabla_x f(x).$$

Now, we can use the above consideration to compute terms of the form $\int_K \frac{\partial \phi_J}{\partial x_k} \frac{\partial \phi_I}{\partial x_l}$ with $1 \leq k, l \leq d$ (where d is the dimension) by performing a variable change in the integrals:

$$\int_K \frac{\partial \phi_J}{\partial x_k} \frac{\partial \phi_I}{\partial x_l} \mathrm{d}x = \int_{\hat{K}} \left[B_K^{-T} \nabla_{\hat{x}} \hat{\phi}_J \right]_k \left[B_K^{-T} \nabla_{\hat{x}} \hat{\phi}_I \right]_l \det B_K(\hat{x}) \mathrm{d}\hat{x},$$

and [2]:

$$\int_K \phi_I \phi_J \mathrm{d}x = \int_{\hat{K}} \hat{\phi}_I \hat{\phi}_J \det B_K(\hat{x}) \mathrm{d}\hat{x}.$$

Note that I and J belong to K, otherwise the integral is zero.

P^1 case

1D case. In this case, we do not need to compute the stiffness matrix as it only appears in $1D$ linear elasticity and it is not studied in this appendix. We will only have a look at the case of the mass matrix. The shape functions are:

$$\begin{aligned} \hat{\phi}_0(\hat{x}) &= 1 - \hat{x}, \\ \hat{\phi}_1(\hat{x}) &= \hat{x}. \end{aligned} \tag{B.5}$$

The norm of the derivative of the transformation F_K is given by:

$$||F_K'|| = ||P_1 - P_0|| = |K|.$$

In other words, it is the length of the segment. Therefore, we have:

$$\int_K \phi_I \phi_J \mathrm{d}x = |K| \int_{\hat{K}} \hat{\phi}_I \hat{\phi}_J \mathrm{d}\hat{x},$$

which comes down to integrate on $\hat{K}$ as a current element like it will be the case in section B.6.2.

2. In the case of a surface mass matrix, the determinant of the jacobian matrix is replaced by the norm of the derivative in $2D$ and by the cross product of the two columns of the jacobian matrix in $3D$.

2D case. The jacobian matrix of the mapping from the reference triangle to the current triangle $K = (P_0, P_1, P_2)$ is given by:

$$B_K = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix} = \begin{pmatrix} \boldsymbol{e_2} & \boldsymbol{e_1} \end{pmatrix},$$

$$B_K^{-T} = \frac{1}{\det B_K} \begin{pmatrix} y_2 - y_0 & y_0 - y_1 \\ x_0 - x_2 & x_1 - x_0 \end{pmatrix} = \frac{1}{\det B_K} \begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} \end{pmatrix}.$$

The shape functions and their gradients on the reference triangle $\hat{K}$ are given by:

$$\begin{aligned} &\hat{\phi}_0(\hat{x}, \hat{y}) = 1 - \hat{x} - \hat{y} && \nabla_{\hat{\boldsymbol{x}}} \hat{\phi}_0 = \begin{pmatrix} -1, & -1 \end{pmatrix}^T, \\ &\hat{\phi}_1(\hat{x}, \hat{y}) = \hat{x} && \nabla_{\hat{\boldsymbol{x}}} \hat{\phi}_1 = \begin{pmatrix} 1, & 0 \end{pmatrix}^T, \\ &\hat{\phi}_2(\hat{x}, \hat{y}) = \hat{y} && \nabla_{\hat{\boldsymbol{x}}} \hat{\phi}_2 = \begin{pmatrix} 0, & 1 \end{pmatrix}^T, \end{aligned} \tag{B.6}$$

and, remembering that $\boldsymbol{n_0} + \boldsymbol{n_1} + \boldsymbol{n_2} = \boldsymbol{0}$, the scaled gradients are given by:

$$\begin{cases} B_K^{-T} \nabla_{\hat{\boldsymbol{x}}} \phi_0(\hat{\boldsymbol{x}}) = \dfrac{1}{\det B_K} \begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} \end{pmatrix} \begin{pmatrix} -1 \\ -1 \end{pmatrix} = \dfrac{1}{\det B_K} \boldsymbol{n_0}, \\ B_K^{-T} \nabla_{\hat{\boldsymbol{x}}} \phi_1(\hat{\boldsymbol{x}}) = \dfrac{1}{\det B_K} \begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \dfrac{1}{\det B_K} \boldsymbol{n_1}, \\ B_K^{-T} \nabla_{\hat{\boldsymbol{x}}} \phi_2(\hat{\boldsymbol{x}}) = \dfrac{1}{\det B_K} \begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \dfrac{1}{\det B_K} \boldsymbol{n_2}. \end{cases} \tag{B.7}$$

Now, we can note that $\det B_K = 2|K|$ and that its counterpart for a surface triangle gives the same result. Consequently, we have:

$$\int_K \frac{\partial \phi_J}{\partial x_k} \frac{\partial \phi_I}{\partial x_l} \mathrm{d}x = \int_{\hat{K}} \frac{1}{2|K|} n_{J,k} n_{I,l} \mathrm{d}\hat{x} = \frac{n_{J,k} n_{I,l}}{4|K|},$$

and

$$\int_K \phi_I \phi_J \mathrm{d}x = 2|K| \int_{\hat{K}} \hat{\phi}_I \hat{\phi}_J \mathrm{d}\hat{x},$$

which comes down to integrate on $\hat{K}$ as a current element like it will be the case in section B.6.2.

3D case. The jacobian matrix of the mapping from $\hat{K}$ to the current tetrahedron K is given by:

$$B_K = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 \end{pmatrix} = \begin{pmatrix} \boldsymbol{e_0} & \boldsymbol{e_1} & \boldsymbol{e_2} \end{pmatrix},$$

$$B_K^{-T} = \tfrac{1}{\det B_K} \begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} & \boldsymbol{n_3} \end{pmatrix} = \tfrac{1}{6|K|} \begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} & \boldsymbol{n_3} \end{pmatrix}.$$

The shape functions and their gradients on the reference tetrahedron are given by:

$$\begin{aligned} &\hat{\phi}_0 = 1 - \hat{x} - \hat{y} - \hat{z} && \nabla_{\hat{\boldsymbol{x}}} \hat{\phi}_0 = \begin{pmatrix} -1, & -1, & -1 \end{pmatrix}^T, \\ &\hat{\phi}_1 = \hat{x} && \nabla_{\hat{\boldsymbol{x}}} \hat{\phi}_1 = \begin{pmatrix} 1, & 0, & 0 \end{pmatrix}^T, \\ &\hat{\phi}_2 = \hat{y} && \nabla_{\hat{\boldsymbol{x}}} \hat{\phi}_2 = \begin{pmatrix} 0, & 1, & 0 \end{pmatrix}^T, \\ &\hat{\phi}_3 = \hat{z} && \nabla_{\hat{\boldsymbol{x}}} \hat{\phi}_3 = \begin{pmatrix} 0, & 0, & 1 \end{pmatrix}^T. \end{aligned} \tag{B.8}$$

and, remembering that $\boldsymbol{n_0} + \boldsymbol{n_1} + \boldsymbol{n_2} + \boldsymbol{n_3} = \boldsymbol{0}$ (see figure B.2), the scaled gradients are

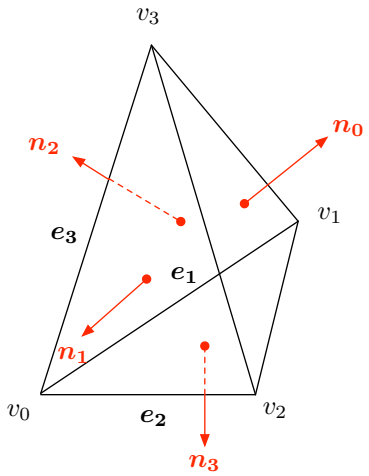

Figure B.2 – Outwards normals of a tetrahedron

given by:

$$\begin{cases} B_K^{-T}\nabla_{\hat{\boldsymbol{x}}}\phi_0(\hat{\boldsymbol{x}}) = \dfrac{1}{\det B_K}\begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} & \boldsymbol{n_3} \end{pmatrix}\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} = \dfrac{1}{\det B_K}\boldsymbol{n_0}, \\ B_K^{-T}\nabla_{\hat{\boldsymbol{x}}}\phi_1(\hat{\boldsymbol{x}}) = \dfrac{1}{\det B_K}\begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} & \boldsymbol{n_3} \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \dfrac{1}{\det B_K}\boldsymbol{n_1}, \\ B_K^{-T}\nabla_{\hat{\boldsymbol{x}}}\phi_2(\hat{\boldsymbol{x}}) = \dfrac{1}{\det B_K}\begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} & \boldsymbol{n_3} \end{pmatrix}\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \dfrac{1}{\det B_K}\boldsymbol{n_2}, \\ B_K^{-T}\nabla_{\hat{\boldsymbol{x}}}\phi_3(\hat{\boldsymbol{x}}) = \dfrac{1}{\det B_K}\begin{pmatrix} \boldsymbol{n_1} & \boldsymbol{n_2} & \boldsymbol{n_3} \end{pmatrix}\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \dfrac{1}{\det B_K}\boldsymbol{n_3}. \end{cases} \tag{B.9}$$

Now, we can note that $\det B_K = 6|K|$. Consequently, we have:

$$\int_K \frac{\partial \phi_J}{\partial x_k}\frac{\partial \phi_I}{\partial x_l}\mathrm{d}x = \int_{\hat{K}} \frac{1}{6|K|} n_{J,k} n_{I,l}\mathrm{d}\hat{x} = \frac{n_{J,k}n_{I,l}}{36|K|},$$

and

$$\int_K \phi_I \phi_J \mathrm{d}x = 6|K| \int_{\hat{K}} \hat{\phi}_I \hat{\phi}_J \mathrm{d}\hat{x},$$

which comes down to integrate on $\hat{K}$ as a current element like it will be the case in section B.6.2.

High-order case

In the high-order case, the mapping F_K is in general[3] not linear anymore. It is a polynomial mapping. Consequently, it will be required to integrate on the reference element

3. If in fact it is linear, the applied strategy is the one explained in the section B.6.2

a polynomial function for the mass matrix and a "rational" function (due to the presence of B_K^{-T}) for the stiffness matrix. To perform this integration, a quadrature formula on the reference element is used.

B.6.2 Integration on the current element

When the considered current element is straight, *e.g.* it is a P^1 element or it is a high-order element which in fact degenerates into a P^1, the integration of the mass and stiffness matrices with isoparametric finite elements can be simplified by doing the integration directly on the current element which is a simplex.

1D case

The shape functions are depending on the barycentrics (u, v) of the current element. Then, as the element is straight, we have that $\det B_K = |K|$. Finally, we need to recall that on an edge K we have the following result [Ciarlet 1978]:

$$\int_K u^{\alpha_0} v^{\alpha_1} \mathrm{d}\Omega = \frac{\alpha_0!\alpha_1!}{(1+\alpha_0+\alpha_1)!}|K|, \tag{B.10}$$

where $(\alpha_0, \alpha_1) \in \mathbb{N}^2$. Based on these results, we are able to compute analytically all the possible coefficients of the mass matrix at any order on a straight edge. In the following, we will explicit it from P^1 to P^3. Note that to avoid any numbering issue, we will note the shape functions in Bézier notations.

P^1 ***case.*** The P^1 shape functions are:

$$\begin{aligned}\phi_0(x) &= u,\\ \phi_1(x) &= v.\end{aligned}$$

The elementary integral products can be computed and are summarized in Table B.1.

$\int_K \phi_i\phi_j$	ϕ_0	ϕ_1
ϕ_0	$\frac{\lvert K\rvert}{3}$	$\frac{\lvert K\rvert}{6}$
ϕ_1	$\frac{\lvert K\rvert}{6}$	$\frac{\lvert K\rvert}{3}$

Table B.1 – Elementary integral products for the mass matrix computation on a straight edge with P^1 finite elements.

P^2 ***case.*** The P^2 shape functions are:

$$\begin{aligned}\phi_0(x) &= 2u(u-\frac{1}{2}),\\ \phi_1(x) &= 4uv,\\ \phi_2(x) &= 2v(v-\frac{1}{2}).\end{aligned}$$

The elementary integral products can be computed and are summarized in Table B.2.

P^3 ***case.*** The P^3 shape functions are:

$$\begin{aligned}\phi_0(x) &= \frac{u}{2}(3u-1)(3u-2),\\ \phi_1(x) &= \frac{9u}{2}(3u-1)v,\end{aligned}$$

$\int_K \phi_i\phi_j$	ϕ_0	ϕ_1	ϕ_2
ϕ_0	$\frac{2\|K\|}{15}$	$\frac{\|K\|}{15}$	$-\frac{\|K\|}{30}$
ϕ_1	$\frac{\|K\|}{15}$	$\frac{8\|K\|}{15}$	$\frac{\|K\|}{15}$
ϕ_2	$-\frac{\|K\|}{30}$	$\frac{\|K\|}{15}$	$\frac{2\|K\|}{15}$

Table B.2 – Elementary integral products for the mass matrix computation on a straight edge with P^2 finite elements.

$$\phi_2(x) = \frac{9v}{2}(3v-1)u,$$
$$\phi_3(x) = \frac{v}{2}(3v-1)(3v-2).$$

The elementary integral products can be computed and are summarized in Table B.3.

$\int_K \phi_i\phi_j$	ϕ_0	ϕ_1	ϕ_2	ϕ_3
ϕ_0	$\frac{8\|K\|}{105}$	$\frac{33\|K\|}{560}$	- $\frac{3\|K\|}{140}$	$\frac{19\|K\|}{1680}$
ϕ_1	$\frac{33\|K\|}{560}$	$\frac{27\|K\|}{70}$	- $\frac{27\|K\|}{560}$	$\frac{33\|K\|}{560}$
ϕ_2	- $\frac{3\|K\|}{140}$	- $\frac{27\|K\|}{560}$	$\frac{27\|K\|}{70}$	- $\frac{3\|K\|}{140}$
ϕ_3	$\frac{19\|K\|}{1680}$	$\frac{33\|K\|}{560}$	- $\frac{3\|K\|}{140}$	$\frac{8\|K\|}{105}$

Table B.3 – Elementary integral products for the mass matrix computation on a straight edge with P^3 finite elements.

2D case

The shape functions are depending on the barycentrics (u, v, w) of the current element. If the element is straight, these barycentrics can be easily computed:

$$u(x,y) = \frac{x_1y_2 - x_2y_1 + x\,(y_1-y_2) + y\,(x_2-x_1)}{2|K|},$$

$$v(x,y) = \frac{x_2y_0 - x_0y_2 + x\,(y_2-y_0) + y\,(x_0-x_2)}{2|K|},$$

$$w(x,y) = \frac{x_0y_1 - x_1y_0 + x\,(y_0-y_1) + y\,(x_1-x_0)}{2|K|}.$$

We can therefore directly compute the gradients of the barycentrics on the current element K:

$$\nabla_{\boldsymbol{x}} u(\boldsymbol{x}) = \frac{1}{2|K|}\begin{pmatrix} y_1-y_2 \\ x_2-x_1 \end{pmatrix} = \frac{1}{2|K|}\mathbf{n}_0,$$
$$\nabla_{\boldsymbol{x}} v(\boldsymbol{x}) = \frac{1}{2|K|}\begin{pmatrix} y_2-y_0 \\ x_0-x_2 \end{pmatrix} = \frac{1}{2|K|}\mathbf{n}_1,$$
$$\nabla_{\boldsymbol{x}} w(\boldsymbol{x}) = \frac{1}{2|K|}\begin{pmatrix} y_0-y_1 \\ x_1-x_0 \end{pmatrix} = \frac{1}{2|K|}\mathbf{n}_2.$$

Then, as the element is straight, we have that $\det B_K = 2|K|$. Finally, we need to recall that on a triangle K we have the following result [Ciarlet 1978]:

$$\int_K u^{\alpha_0} v^{\alpha_1} w^{\alpha_2} \mathrm{d}\Omega = \frac{\alpha_0!\alpha_1!\alpha_2!}{(2+\alpha_0+\alpha_1+\alpha_2)!} 2|K|, \tag{B.11}$$

where $(\alpha_0, \alpha_1, \alpha_2) \in \mathbb{N}^3$. Based on these results, we are able to compute analytically all the possible coefficients of both stiffness and mass matrices at any order on a straight triangle. In the following, we will explicit it from P^1 to P^3. Note that to avoid any numbering issue, we will note the shape functions in Bézier notations.

P^1 ***case.*** The P^1 shape functions and their gradients are:

$$\begin{aligned}
&\phi_{100}(\boldsymbol{x}) = u && \text{and} && \nabla_{\mathbf{x}}\phi_{100}(x) = \frac{1}{2|K|}\mathbf{n}_0,\\
&\phi_{010}(\boldsymbol{x}) = v && \text{and} && \nabla_{\mathbf{x}}\phi_{010}(x) = \frac{1}{2|K|}\mathbf{n}_1,\\
&\phi_{001}(\boldsymbol{x}) = w && \text{and} && \nabla_{\mathbf{x}}\phi_{001}(x) = \frac{1}{2|K|}\mathbf{n}_2.
\end{aligned}$$

Note that the same gradients as in the previous section are obtained.
The elementary integral products can be computed and are summarized in Tables B.4 and B.5.

$\int_K \phi_i\phi_j$	ϕ_{100}	ϕ_{010}	ϕ_{001}
ϕ_{100}	$\frac{\lvert K\rvert}{6}$	$\frac{\lvert K\rvert}{12}$	$\frac{\lvert K\rvert}{12}$
ϕ_{010}	$\frac{\lvert K\rvert}{12}$	$\frac{\lvert K\rvert}{6}$	$\frac{\lvert K\rvert}{12}$
ϕ_{001}	$\frac{\lvert K\rvert}{12}$	$\frac{\lvert K\rvert}{12}$	$\frac{\lvert K\rvert}{6}$

Table B.4 – Elementary integral products for the mass matrix computation on a straight triangle with P^1 finite elements.

$\int_K \frac{\partial\phi_i}{\partial x_l}\frac{\partial\phi_j}{\partial x_k}$	$\frac{\partial\phi_{100}}{\partial x_k}$	$\frac{\partial\phi_{010}}{\partial x_k}$	$\frac{\partial\phi_{001}}{\partial x_k}$
$\frac{\partial\phi_{100}}{\partial x_l}$	$\frac{n_{0,k}n_{0,l}}{4\lvert K\rvert}$	$\frac{n_{1,k}n_{0,l}}{4\lvert K\rvert}$	$\frac{n_{2,k}n_{0,l}}{4\lvert K\rvert}$
$\frac{\partial\phi_{010}}{\partial x_l}$	$\frac{n_{0,k}n_{1,l}}{4\lvert K\rvert}$	$\frac{n_{1,k}n_{1,l}}{4\lvert K\rvert}$	$\frac{n_{2,k}n_{1,l}}{4\lvert K\rvert}$
$\frac{\partial\phi_{001}}{\partial x_l}$	$\frac{n_{0,k}n_{2,l}}{4\lvert K\rvert}$	$\frac{n_{1,k}n_{2,l}}{4\lvert K\rvert}$	$\frac{n_{2,k}n_{2,l}}{4\lvert K\rvert}$

Table B.5 – Elementary integral products for the stiffness matrix computation on a straight triangle with P^1 finite elements.

P^2 ***case.*** The P^2 shape functions and their gradients are:

$$\begin{aligned}
&\phi_{200}(\boldsymbol{x}) = 2u(u-\frac{1}{2}) && \text{and} && \nabla_{\boldsymbol{x}}\phi_{200}(\boldsymbol{x}) = (4u-1)\frac{1}{2|K|}\mathbf{n}_0,\\
&\phi_{020}(\boldsymbol{x}) = 2v(v-\frac{1}{2}) && \text{and} && \nabla_{\boldsymbol{x}}\phi_{020}(\boldsymbol{x}) = (4v-1)\frac{1}{2|K|}\mathbf{n}_1,\\
&\phi_{002}(\boldsymbol{x}) = 2w(w-\frac{1}{2}) && \text{and} && \nabla_{\boldsymbol{x}}\phi_{002}(\boldsymbol{x}) = (4w-1)\frac{1}{2|K|}\mathbf{n}_2,
\end{aligned}$$

$$\phi_{110}(\boldsymbol{x}) = 4uv \quad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{110}(\boldsymbol{x}) = 4v\frac{1}{2|K|}\mathbf{n}_0 + 4u\frac{1}{2|K|}\mathbf{n}_1,$$
$$\phi_{011}(\boldsymbol{x}) = 4vw \quad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{011}(\boldsymbol{x}) = 4w\frac{1}{2|K|}\mathbf{n}_1 + 4v\frac{1}{2|K|}\mathbf{n}_2,$$
$$\phi_{101}(\boldsymbol{x}) = 4wu \quad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{101}(\boldsymbol{x}) = 4w\frac{1}{2|K|}\mathbf{n}_0 + 4u\frac{1}{2|K|}\mathbf{n}_2.$$

The elementary integral products can be computed. Some results for the mass matrix are shown below:

$$\int_K (\phi_{200})^2 = \frac{|K|}{30}, \quad \int_K \phi_{200}\phi_{020} = -\frac{|K|}{180}, \quad \int_K (\phi_{110})^2 = \frac{8|K|}{45},$$
$$\int_K \phi_{110}\phi_{011} = \frac{4|K|}{45}, \quad \int_K \phi_{200}\phi_{110} = 0, \quad \int_K \phi_{200}\phi_{011} = -\frac{|K|}{45}.$$

The remaining integrals can be deduced by analogy from these ones.
Concerning the stiffness matrix, we also have:

$$\int_K \frac{\partial\phi_{200}}{\partial x_k}\frac{\partial\phi_{200}}{\partial x_l} = \frac{n_{0,k}n_{0,l}}{4|K|}, \quad \int_K \frac{\partial\phi_{200}}{\partial x_k}\frac{\partial\phi_{020}}{\partial x_l} = -\frac{n_{0,k}n_{1,l}}{12|K|},$$
$$\int_K \frac{\partial\phi_{200}}{\partial x_k}\frac{\partial\phi_{110}}{\partial x_l} = \frac{1}{3|K|}n_{0,k}n_{1,l}, \quad \int_K \frac{\partial\phi_{200}}{\partial x_k}\frac{\partial\phi_{011}}{\partial x_l} = 0,$$
$$\int_K \frac{\partial\phi_{110}}{\partial x_k}\frac{\partial\phi_{110}}{\partial x_l} = \frac{1}{3|K|}(2n_{0,k}n_{0,l} + n_{0,k}n_{1,l} + n_{1,k}n_{0,l} + 2n_{1,k}n_{1,l}),$$
$$\int_K \frac{\partial\phi_{110}}{\partial x_k}\frac{\partial\phi_{101}}{\partial x_l} = \frac{1}{3|K|}(n_{0,k}n_{0,l} + n_{0,k}n_{2,l} + n_{1,k}n_{0,l} + 2n_{1,k}n_{2,l}).$$

Likewise, the remaining integrals can be deduced by analogy from these ones.

P^3 ***case.*** Some P^3 shape functions and their gradients are:

$$\phi_{300}(\boldsymbol{x}) = \frac{1}{2}u(3u-1)(3u-2) \quad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{300}(\boldsymbol{x}) = (\frac{27}{2}u^2 - 9u + 1)\frac{1}{2|K|}\mathbf{n}_0,$$
$$\phi_{210}(\boldsymbol{x}) = \frac{9}{2}u(3u-1)v \quad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{210}(\boldsymbol{x}) = \frac{9}{2}(6u-1)v\frac{1}{2|K|}\mathbf{n}_0 + \frac{9}{2}(3u-1)u\frac{1}{2|K|}\mathbf{n}_1,$$
$$\phi_{111}(\boldsymbol{x}) = 27uvw \quad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{111}(\boldsymbol{x}) = 27vw\frac{1}{2|K|}\mathbf{n}_0 + 27uw\frac{1}{2|K|}\mathbf{n}_1 + 27uv\frac{1}{2|K|}\mathbf{n}_2.$$

The other shape functions and their gradients can be deduced by analogy.
The elementary integral products can be computed. Some results for the mass matrix are shown below:

$$\int_K (\phi_{300})^2 = \frac{19|K|}{1680}, \quad \int_K \phi_{300}\phi_{030} = \frac{11|K|}{6720}, \quad \int_K (\phi_{111})^2 = \frac{81|K|}{280},$$
$$\int_K \phi_{210}\phi_{111} = \frac{27|K|}{1120}, \quad \int_K (\phi_{110})^2 = \frac{9|K|}{112}, \quad \int_K \phi_{210}\phi_{120} = -\frac{9|K|}{320},$$
$$\int_K \phi_{210}\phi_{201} = \frac{9|K|}{224}, \quad \int_K \phi_{210}\phi_{102} = -\frac{9|K|}{448}, \quad \int_K \phi_{120}\phi_{102} = -\frac{9|K|}{1120},$$

$$\int_K \phi_{300}\phi_{111} = \frac{3|K|}{560}, \quad \int_K \phi_{300}\phi_{021} = \frac{9|K|}{2240}, \quad \int_K \phi_{300}\phi_{210} = \frac{3|K|}{1120},$$
$$\int_K \phi_{300}\phi_{120} = 0.$$

The remaining integrals can be deduced by analogy from these ones.
Concerning the stiffness matrix, we also have:

$$\begin{aligned}
&\textstyle\int_K \frac{\partial\phi_{300}}{\partial x_k}\frac{\partial\phi_{300}}{\partial x_l} = \frac{17n_{0,k}n_{0,l}}{80|K|}, \quad \int_K \frac{\partial\phi_{300}}{\partial x_k}\frac{\partial\phi_{030}}{\partial x_l} = \frac{7n_{0,k}n_{1,l}}{160|K|},\\
&\textstyle\int_K \frac{\partial\phi_{111}}{\partial x_k}\frac{\partial\phi_{111}}{\partial x_l} = \frac{81}{80|K|}(\quad 2n_{0,k}n_{0,l} + n_{0,k}n_{1,l} + n_{0,k}n_{2,l} + n_{1,k}n_{0,l}\\
&\qquad\qquad +2n_{1,k}n_{1,l} + n_{1,k}n_{2,l} + n_{2,k}n_{0,l} + n_{2,k}n_{1,l} + 2n_{2,k}n_{2,l}),\\
&\textstyle\int_K \frac{\partial\phi_{111}}{\partial x_k}\frac{\partial\phi_{210}}{\partial x_l} = \frac{81}{160|K|}(n_{1,k}n_{0,l} + n_{1,k}n_{1,l} + 2n_{2,k}n_{0,l} + n_{2,k}n_{1,l}),\\
&\textstyle\int_K \frac{\partial\phi_{210}}{\partial x_k}\frac{\partial\phi_{210}}{\partial x_l} = \frac{27}{64|K|}(2n_{0,k}n_{0,l} + n_{0,k}n_{1,l} + n_{1,k}n_{0,l} + 2n_{1,k}n_{1,l}),\\
&\textstyle\int_K \frac{\partial\phi_{210}}{\partial x_k}\frac{\partial\phi_{120}}{\partial x_l} = -\frac{27}{320|K|}(2n_{0,k}n_{0,l} - 5n_{0,k}n_{1,l} + n_{1,k}n_{0,l} + 2n_{1,k}n_{1,l}),\\
&\textstyle\int_K \frac{\partial\phi_{210}}{\partial x_k}\frac{\partial\phi_{201}}{\partial x_l} = \frac{27}{64|K|}(n_{0,k}n_{0,l} + n_{0,k}n_{2,l} + n_{1,k}n_{0,l} + 2n_{1,k}n_{2,l}),\\
&\textstyle\int_K \frac{\partial\phi_{210}}{\partial x_k}\frac{\partial\phi_{012}}{\partial x_l} = -\frac{27}{320|K|}(n_{0,k}n_{1,l} + 2n_{0,k}n_{2,l} + n_{1,k}n_{1,l} + n_{1,k}n_{2,l}),\\
&\textstyle\int_K \frac{\partial\phi_{210}}{\partial x_k}\frac{\partial\phi_{021}}{\partial x_l} = -\frac{27}{320|K|}(n_{0,k}n_{1,l} + 2n_{0,k}n_{2,l} + n_{1,k}n_{1,l} + n_{1,k}n_{2,l}),\\
&\textstyle\int_K \frac{\partial\phi_{111}}{\partial x_k}\frac{\partial\phi_{300}}{\partial x_l} = 0, \int_K \frac{\partial\phi_{300}}{\partial x_k}\frac{\partial\phi_{210}}{\partial x_l} = \frac{3}{160|K|}n_{0,k}(n_{0,l} + 19n_{1,l}),\\
&\textstyle\int_K \frac{\partial\phi_{300}}{\partial x_k}\frac{\partial\phi_{120}}{\partial x_l} = \frac{3}{160|K|}n_{0,k}(n_{0,l} - 8n_{1,l}), \int_K \frac{\partial\phi_{300}}{\partial x_k}\frac{\partial\phi_{021}}{\partial x_l} = \frac{3}{160|K|}n_{0,k}(n_{1,l} + n_{2,l}).
\end{aligned}$$

Likewise, the remaining integrals can be deduced by analogy from these ones.

3D case

The shape functions are depending on the barycentrics (u, v, w, t) of the current element. If the element is straight, these barycentrics can be easily computed:

$$\begin{aligned}
u(x,y,z) &= \frac{(-y_2z_1 - z_2y_3 + y_1z_2 + y_3z_1 - y_1z_3 + y_2z_3)x}{6|K|}\\
&+ \frac{(x_1z_3 - x_3z_1 + x_3z_2 + x_2z_1 - x_1z_2 - x_2z_3)y}{6|K|}\\
&+ \frac{(x_1y_2 - x_3y_2 - x_1y_3 - x_2y_1 + x_2y_3 + x_3y_1)z}{6|K|}\\
&+ \frac{x_1z_2y_3 + x_2y_1z_3 - x_3y_1z_2 - x_2y_3z_1 - x_1y_2z_3 + x_3y_2z_1}{6|K|},
\end{aligned}$$

$$\begin{aligned}
v(x,y,z) &= \frac{(y_2z_3 + y_3z_0 + y_0z_2 - y_2z_0 - z_2y_3 - z_3y_0)x}{6|K|}\\
&+ \frac{(x_2z_0 - x_3z_0 + x_3z_2 + x_0z_3 - x_2z_3 - x_0z_2)y}{6|K|}\\
&+ \frac{(x_3y_0 - x_3y_2 - x_0y_3 - x_2y_0 + x_2y_3 + x_0y_2)z}{6|K|}\\
&+ \frac{x_3y_2z_0 + x_0z_2y_3 + x_2z_3y_0 - x_2y_3z_0 - x_3y_0z_2 - x_0y_2z_3}{6|K|},
\end{aligned}$$

$$
\begin{aligned}
w(x,y,z) &= \frac{(-z_0y_1 + y_1z_3 - y_3z_1 + y_3z_0 + y_0z_1 - z_3y_0)x}{6|K|} \\
&+ \frac{(x_3z_1 - x_3z_0 + x_0z_3 - x_0z_1 + x_1z_0 - x_1z_3)y}{6|K|} \\
&+ \frac{(x_0y_1 - x_3y_1 - x_1y_0 + x_3y_0 + x_1y_3 - x_0y_3)z}{6|K|} \\
&+ \frac{x_1z_3y_0 - x_0y_1z_3 - x_3y_0z_1 + x_3z_0y_1 - x_1y_3z_0 + x_0y_3z_1}{6|K|}, \\
t(x,y,z) &= \frac{(y_1z_2 + y_0z_1 - y_2z_1 - z_0y_1 - y_0z_2 + y_2z_0)x}{6|K|} \\
&+ \frac{(x_2z_1 - x_2z_0 + x_0z_2 - x_0z_1 + x_1z_0 - x_1z_2)y}{6|K|} \\
&+ \frac{(x_1y_2 - x_0y_2 - x_2y_1 + x_0y_1 + x_2y_0 - x_1y_0)z}{6|K|} \\
&+ \frac{x_1y_0z_2 - x_0y_1z_2 + x_2z_0y_1 + x_0y_2z_1 - x_2y_0z_1 - x_1y_2z_0}{6|K|}.
\end{aligned}
$$

We can therefore directly compute the gradients of the barycentrics on the current element K:

$$
\nabla_{\boldsymbol{x}} u(\boldsymbol{x}) = \frac{1}{6|K|}\begin{pmatrix} -y_2z_1 - z_2y_3 + y_1z_2 + y_3z_1 - y_1z_3 + y_2z_3 \\ x_1z_3 - x_3z_1 + x_3z_2 + x_2z_1 - x_1z_2 - x_2z_3 \\ x_1y_2 - x_3y_2 - x_1y_3 - x_2y_1 + x_2y_3 + x_3y_1 \end{pmatrix} = \frac{1}{6|K|}\mathbf{n}_0,
$$

$$
\nabla_{\boldsymbol{x}} v(\boldsymbol{x}) = \frac{1}{6|K|}\begin{pmatrix} y_2z_3 + y_3z_0 + y_0z_2 - y_2z_0 - z_2y_3 - z_3y_0 \\ x_2z_0 - x_3z_0 + x_3z_2 + x_0z_3 - x_2z_3 - x_0z_2 \\ x_3y_0 - x_3y_2 - x_0y_3 - x_2y_0 + x_2y_3 + x_0y_2 \end{pmatrix} = \frac{1}{6|K|}\mathbf{n}_1,
$$

$$
\nabla_{\boldsymbol{x}} w(\boldsymbol{x}) = \frac{1}{6|K|}\begin{pmatrix} -z_0y_1 + y_1z_3 - y_3z_1 + y_3z_0 + y_0z_1 - z_3y_0 \\ x_3z_1 - x_3z_0 + x_0z_3 - x_0z_1 + x_1z_0 - x_1z_3 \\ x_0y_1 - x_3y_1 - x_1y_0 + x_3y_0 + x_1y_3 - x_0y_3 \end{pmatrix} = \frac{1}{6|K|}\mathbf{n}_2,
$$

$$
\nabla_{\boldsymbol{x}} t(\boldsymbol{x}) = \frac{1}{6|K|}\begin{pmatrix} y_1z_2 + y_0z_1 - y_2z_1 - z_0y_1 - y_0z_2 + y_2z_0 \\ x_2z_1 - x_2z_0 + x_0z_2 - x_0z_1 + x_1z_0 - x_1z_2 \\ x_1y_2 - x_0y_2 - x_2y_1 + x_0y_1 + x_2y_0 - x_1y_0 \end{pmatrix} = \frac{1}{6|K|}\mathbf{n}_3.
$$

Then, as the element is straight, we have that $\det B_K = 6|K|$. Finally, we need to recall that on a tetrahedron K we have the following result [Ciarlet 1978]:

$$
\int_K u^{\alpha_0} v^{\alpha_1} w^{\alpha_2} t^{\alpha_3} \mathrm{d}\Omega = \frac{\alpha_0!\alpha_1!\alpha_2!\alpha_3!}{(3+\alpha_0+\alpha_1+\alpha_2+\alpha_3)!} 6|K|, \tag{B.12}
$$

where $(\alpha_0, \alpha_1, \alpha_2, \alpha_3) \in \mathbb{N}^4$. Based on these results, we are able to compute analytically all the possible coefficients of both stiffness and mass matrices at any order on a straight tetrahedron. In the following, we will explicit it from P^1 to P^3. Note that to avoid any numbering issue, we will note the shape functions in Bézier notations.

P^1 ***case.*** The P^1 shape functions and their gradients are:

$$
\begin{aligned}
&\phi_{1000}(\boldsymbol{x}) = u && \text{and} && \nabla_{\boldsymbol{x}}\phi_{1000}(\boldsymbol{x}) = \frac{1}{2|K|}\mathbf{n}_0, \\
&\phi_{0100}(\boldsymbol{x}) = v && \text{and} && \nabla_{\boldsymbol{x}}\phi_{0100}(\boldsymbol{x}) = \frac{1}{2|K|}\mathbf{n}_1, \\
&\phi_{0010}(\boldsymbol{x}) = w && \text{and} && \nabla_{\boldsymbol{x}}\phi_{0010}(\boldsymbol{x}) = \frac{1}{2|K|}\mathbf{n}_2,
\end{aligned}
$$

$$\phi_{0001}(\boldsymbol{x}) = t \qquad \text{and} \qquad \nabla_{\boldsymbol{x}}\phi_{0001}(\boldsymbol{x}) = \frac{1}{2|K|}\mathbf{n}_3.$$

Note that the same gradients as in the previous section are obtained.
The elementary integral products can be computed and are summarized in Tables B.6 and B.7.

$\int_K \phi_i\phi_j$	ϕ_{1000}	ϕ_{0100}	ϕ_{0010}	ϕ_{0001}
ϕ_{1000}	$\frac{\lvert K\rvert}{10}$	$\frac{\lvert K\rvert}{20}$	$\frac{\lvert K\rvert}{20}$	$\frac{\lvert K\rvert}{20}$
ϕ_{0100}	$\frac{\lvert K\rvert}{20}$	$\frac{\lvert K\rvert}{10}$	$\frac{\lvert K\rvert}{20}$	$\frac{\lvert K\rvert}{20}$
ϕ_{0010}	$\frac{\lvert K\rvert}{20}$	$\frac{\lvert K\rvert}{20}$	$\frac{\lvert K\rvert}{10}$	$\frac{\lvert K\rvert}{20}$
ϕ_{0001}	$\frac{\lvert K\rvert}{20}$	$\frac{\lvert K\rvert}{20}$	$\frac{\lvert K\rvert}{20}$	$\frac{\lvert K\rvert}{10}$

Table B.6 – Elementary integral products for the mass matrix computation on a straight tetrahedron with P^1 finite elements.

$\int_K \frac{\partial\phi_i}{\partial x_l}\frac{\partial\phi_j}{\partial x_k}$	$\frac{\partial\phi_{1000}}{\partial x_k}$	$\frac{\partial\phi_{0100}}{\partial x_k}$	$\frac{\partial\phi_{000}}{\partial x_k}$	$\frac{\partial\phi_{000}}{\partial x_k}$
$\frac{\partial\phi_{1000}}{\partial x_l}$	$\frac{n_{0,k}n_{0,l}}{36\lvert K\rvert}$	$\frac{n_{1,k}n_{0,l}}{36\lvert K\rvert}$	$\frac{n_{2,k}n_{0,l}}{36\lvert K\rvert}$	$\frac{n_{3,k}n_{0,l}}{36\lvert K\rvert}$
$\frac{\partial\phi_{0100}}{\partial x_l}$	$\frac{n_{0,k}n_{1,l}}{36\lvert K\rvert}$	$\frac{n_{1,k}n_{1,l}}{36\lvert K\rvert}$	$\frac{n_{2,k}n_{1,l}}{36\lvert K\rvert}$	$\frac{n_{3,k}n_{1,l}}{36\lvert K\rvert}$
$\frac{\partial\phi_{0010}}{\partial x_l}$	$\frac{n_{0,k}n_{2,l}}{36\lvert K\rvert}$	$\frac{n_{1,k}n_{2,l}}{36\lvert K\rvert}$	$\frac{n_{2,k}n_{2,l}}{36\lvert K\rvert}$	$\frac{n_{3,k}n_{2,l}}{36\lvert K\rvert}$
$\frac{\partial\phi_{0001}}{\partial x_l}$	$\frac{n_{0,k}n_{3,l}}{36\lvert K\rvert}$	$\frac{n_{1,k}n_{3,l}}{36\lvert K\rvert}$	$\frac{n_{2,k}n_{3,l}}{36\lvert K\rvert}$	$\frac{n_{3,k}n_{3,l}}{36\lvert K\rvert}$

Table B.7 – Elementary integral products for the stiffness matrix computation on a straight tetrahedron with P^1 finite elements.

P^2 ***case.*** Some P^2 shape functions and their gradients are:

$$\phi_{2000}(\boldsymbol{x}) = 2u(u-\frac{1}{2}) \qquad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{200}(\boldsymbol{x}) = (4u-1)\frac{1}{2|K|}\mathbf{n}_0,$$
$$\phi_{1100}(\boldsymbol{x}) = 4uv \qquad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{110}(\boldsymbol{x}) = 4v\frac{1}{2|K|}\mathbf{n}_0 + 4u\frac{1}{2|K|}\mathbf{n}_1.$$

The other shape functions and their gradients can be deduced by analogy.
The elementary integral products can be computed. Some results for the mass matrix are shown below:

$$\int_K (\phi_{2000})^2 = \frac{|K|}{70}, \quad \int_K \phi_{2000}\phi_{0200} = \frac{|K|}{420}, \quad \int_K (\phi_{1100})^2 = \frac{8|K|}{1055}, \quad \int_K \phi_{1100}\phi_{0011} = \frac{2|K|}{105},$$
$$\int_K \phi_{1100}\phi_{0110} = \frac{4|K|}{105}, \quad \int_K \phi_{2000}\phi_{1100} = -\frac{|K|}{105}, \quad \int_K \phi_{2000}\phi_{0110} = -\frac{|K|}{70}.$$

The remaining integrals can be deduced by analogy from these ones.
Concerning the stiffness matrix, we also have:

$$\int_K \frac{\partial\phi_{2000}}{\partial x_k}\frac{\partial\phi_{2000}}{\partial x_l} = \frac{n_{0,k}n_{0,l}}{60|K|}, \quad \int_K \frac{\partial\phi_{2000}}{\partial x_k}\frac{\partial\phi_{0200}}{\partial x_l} = -\frac{n_{0,k}n_{1,l}}{180|K|},$$

$$\int_K \frac{\partial\phi_{2000}}{\partial x_k}\frac{\partial\phi_{1100}}{\partial x_l} = \frac{1}{180|K|}n_{0,k}(-n_{0,l}+3n_{1,l}),$$
$$\int_K \frac{\partial\phi_{2000}}{\partial x_k}\frac{\partial\phi_{0110}}{\partial x_l} = -\frac{1}{180|K|}n_{0,k}-n_{1,l}+n_{2,l}),$$
$$\int_K \frac{\partial\phi_{1100}}{\partial x_k}\frac{\partial\phi_{1100}}{\partial x_l} = \frac{1}{45|K|}(2n_{0,k}n_{0,l}+n_{0,k}n_{1,l}+n_{1,k}n_{0,l}+2n_{1,k}n_{1,l}),$$
$$\int_K \frac{\partial\phi_{1100}}{\partial x_k}\frac{\partial\phi_{0011}}{\partial x_l} = \frac{1}{45|K|}(n_{0,k}n_{2,l}+n_{0,k}n_{3,l}+n_{1,k}n_{2,l}+n_{1,k}n_{3,l}),$$
$$\int_K \frac{\partial\phi_{1100}}{\partial x_k}\frac{\partial\phi_{1010}}{\partial x_l} = \frac{1}{45|K|}(n_{0,k}n_{0,l}+n_{0,k}n_{2,l}+n_{1,k}n_{0,l}+2n_{1,k}n_{2,l}).$$

Likewise, the remaining integrals can be deduced by analogy from these ones.

P^3 ***case.*** Some P^3 shape functions and their gradients are:

$$\phi_{3000}(\boldsymbol{x}) = \frac{1}{2}u(3u-1)(3u-2) \quad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{3000}(\boldsymbol{x}) = (\frac{27}{2}u^2-9u+1)\frac{1}{2|K|}\mathbf{n}_0,$$
$$\phi_{2100}(\boldsymbol{x}) = \frac{9}{2}u(3u-1)v \quad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{2100}(\boldsymbol{x}) = \frac{9}{2}(6u-1)v\frac{1}{2|K|}\mathbf{n}_0+\frac{9}{2}(3u-1)u\frac{1}{2|K|}\mathbf{n}_1,$$
$$\phi_{1110}(\boldsymbol{x}) = 27uvw \quad \text{and} \quad \nabla_{\boldsymbol{x}}\phi_{1110}(\boldsymbol{x}) = 27vw\frac{1}{2|K|}\mathbf{n}_0+27uw\frac{1}{2|K|}\mathbf{n}_1+27uv\frac{1}{2|K|}\mathbf{n}_2.$$

The other shape functions and their gradients can be deduced by analogy.
The elementary integral products can be computed. Some results for the mass matrix are shown below:

$$\int_K(\phi_{3000})^2 = \frac{|K|}{280}, \quad \int_K \phi_{3000}\phi_{0300} = \frac{|K|}{2240}, \quad \int_K(\phi_{1110})^2 = \frac{27|K|}{280},$$
$$\int_K \phi_{1110}\phi_{0111} = \frac{27|K|}{560}, \int_K \phi_{2100}\phi_{1110} = 0, \int_K \phi_{2001}\phi_{1110} = 0, \int_K \phi_{1002}\phi_{1110} = -\frac{27|K|}{2240},$$
$$\int_K \phi_{2100}\phi_{2100} = \frac{27|K|}{1120}, \quad \int_K \phi_{2100}\phi_{1200} = -\frac{27|K|}{2240}, \quad \int_K \phi_{2100}\phi_{2010} = \frac{27|K|}{2240},$$
$$\int_K \phi_{2100}\phi_{1020} = -\frac{27|K|}{4480}, \quad \int_K \phi_{1200}\phi_{1020} = 0, \quad \int_K \phi_{1200}\phi_{0012} = 0,$$
$$\int_K \phi_{3000}\phi_{1110} = \frac{9|K|}{2240}, \quad \int_K \phi_{3000}\phi_{0111} = \frac{9|K|}{1120}, \quad \int_K \phi_{3000}\phi_{0210} = \frac{3|K|}{4480},$$
$$\int_K \phi_{3000}\phi_{2100} = -\frac{3|K|}{1120}, \quad \int_K \phi_{3000}\phi_{1200} = \frac{3|K|}{2240}.$$

The remaining integrals can be deduced by analogy from these ones.
Concerning the stiffness matrix, we also have:

$$\int_K \frac{\partial\phi_{3000}}{\partial x_k}\frac{\partial\phi_{3000}}{\partial x_l} = \frac{5n_{0,k}n_{0,l}}{504|K|}, \quad \int_K \frac{\partial\phi_{3000}}{\partial x_k}\frac{\partial\phi_{0300}}{\partial x_l} = \frac{19n_{0,k}n_{1,l}}{10080|K|},$$

$$\begin{aligned}
\int_K \frac{\partial\phi_{1110}}{\partial x_k}\frac{\partial\phi_{1110}}{\partial x_l} &= \frac{27}{560|K|}(\quad 2n_{0,k}n_{0,l} + n_{0,k}n_{1,l} + n_{0,k}n_{2,l} + n_{1,k}n_{0,l} \\
&\quad +2n_{1,k}n_{1,l} + n_{1,k}n_{2,l} + n_{2,k}n_{0,l} + n_{2,k}n_{1,l} + 2n_{2,k}n_{2,l}), \\
\int_K \frac{\partial\phi_{1110}}{\partial x_k}\frac{\partial\phi_{1101}}{\partial x_l} &= \frac{27}{1120|K|}(\quad 2n_{0,k}n_{0,l} + n_{0,k}n_{1,l} + 2n_{0,k}n_{3,l} + n_{1,k}n_{0,l} \\
&\quad +2n_{1,k}n_{1,l} + 2n_{1,k}n_{3,l} + 2n_{2,k}n_{0,l} + 2n_{2,k}n_{1,l} + 4n_{2,k}n_{3,l}), \\
\int_K \frac{\partial\phi_{1110}}{\partial x_k}\frac{\partial\phi_{2100}}{\partial x_l} &= -\frac{9}{2240|K|}(\quad 2n_{0,k}n_{0,l} + n_{0,k}n_{1,l} - 5n_{1,k}n_{0,l} - 4n_{1,k}n_{1,l} \\
&\quad + - 10n_{2,k}n_{0,l} - 4n_{2,k}n_{1,l}), \\
\int_K \frac{\partial\phi_{1011}}{\partial x_k}\frac{\partial\phi_{2100}}{\partial x_l} &= -\frac{9}{2240|K|}(\quad n_{0,k}n_{0,l} + n_{0,k}n_{1,l} - 5n_{2,k}n_{0,l} - 4n_{2,k}n_{1,l} \\
&\quad + - 5n_{3,k}n_{0,l} - 4n_{3,k}n_{1,l}), \\
\int_K \frac{\partial\phi_{0111}}{\partial x_k}\frac{\partial\phi_{2100}}{\partial x_l} &= -\frac{9}{2240|K|}(\quad n_{1,k}n_{0,l} + n_{0,k}n_{1,l} + 2n_{2,k}n_{0,l} + n_{2,k}n_{1,l} \\
&\quad +2n_{3,k}n_{0,l} + n_{3,k}n_{1,l}),
\end{aligned}$$

$$\int_K \frac{\partial\phi_{2100}}{\partial x_k}\frac{\partial\phi_{2100}}{\partial x_l} = \frac{9}{1120|K|}(5n_{0,k}n_{0,l} + 2n_{0,k}n_{1,l} + 2n_{1,k}n_{0,l} + 4n_{1,k}n_{1,l}),$$

$$\int_K \frac{\partial\phi_{2100}}{\partial x_k}\frac{\partial\phi_{1200}}{\partial x_l} = -\frac{9}{2240|K|}(3n_{0,k}n_{0,l} - 3n_{0,k}n_{1,l} + n_{1,k}n_{0,l} + 3n_{1,k}n_{1,l}),$$

$$\int_K \frac{\partial\phi_{2100}}{\partial x_k}\frac{\partial\phi_{0021}}{\partial x_l} = -\frac{9}{2240|K|}(n_{0,k}n_{2,l} + n_{0,k}n_{3,l} + n_{1,k}n_{2,l} + n_{1,k}n_{3,l}),$$

$$\int_K \frac{\partial\phi_{2100}}{\partial x_k}\frac{\partial\phi_{2010}}{\partial x_l} = \frac{9}{2240|K|}(5n_{0,k}n_{0,l} + 4n_{0,k}n_{2,l} + 4n_{1,k}n_{0,l} + 8n_{1,k}n_{2,l}),$$

$$\int_K \frac{\partial\phi_{2100}}{\partial x_k}\frac{\partial\phi_{0120}}{\partial x_l} = -\frac{9}{2240|K|}(n_{0,k}n_{1,l} + 2n_{0,k}n_{2,l} + n_{1,k}n_{1,l} + n_{1,k}n_{2,l}),$$

$$\int_K \frac{\partial\phi_{2100}}{\partial x_k}\frac{\partial\phi_{0210}}{\partial x_l} = -\frac{9}{2240|K|}(2n_{0,k}n_{1,l} + 3n_{0,k}n_{2,l} + n_{1,k}n_{1,l} + n_{1,k}n_{2,l}),$$

$$\int_K \frac{\partial\phi_{1110}}{\partial x_k}\frac{\partial\phi_{3000}}{\partial x_l} = \frac{3}{1120|K|}n_{0,k}(2n_{0,l} - n_{1,l} - n_{2,l}),$$

$$\int_K \frac{\partial\phi_{3000}}{\partial x_k}\frac{\partial\phi_{0111}}{\partial x_l} = \frac{3}{560|K|}n_{0,k}(n_{1,l} + n_{2,l} + n_{3,l}),$$

$$\int_K \frac{\partial\phi_{3000}}{\partial x_k}\frac{\partial\phi_{2100}}{\partial x_l} = -\frac{1}{2240|K|}n_{0,k}(13n_{0,l} - 23n_{1,l}),$$

$$\int_K \frac{\partial\phi_{3000}}{\partial x_k}\frac{\partial\phi_{1200}}{\partial x_l} = \frac{1}{2240|K|}n_{0,k}(5n_{0,l} - 13n_{1,l}),$$

$$\int_K \frac{\partial\phi_{3000}}{\partial x_k}\frac{\partial\phi_{0210}}{\partial x_l} = \frac{1}{448|K|}n_{0,k}(n_{1,l} + n_{2,l}).$$

Likewise, the remaining integrals can be deduced by analogy from these ones.

B.7 Temporal discretization

After the spatial discretization [4], the semi discrete solution is a linear system that can be rewritten as:

$$\mathbb{M}\ddot{\Xi} + \mathbb{K}\Xi = F, \tag{B.13}$$

where F is a vector containing all the Neumann boundary conditions and volume forces. As there is no mesh deformation that is assumed, $\mathbb{M}$ and $\mathbb{K}$ do not vary in time. The time integration is performed with the Newmark scheme. It relies on the extrapolation of the vector of nodal displacement and nodal velocity at time step $n+1$:

$$\begin{aligned}
\Xi^{n+1} &= \Xi^n + \Delta t\dot{\Xi}^n + \frac{\Delta t^2}{2}(1-\beta_2)\ddot{\Xi}^n + \frac{\Delta t^2}{2}\beta_2\ddot{\Xi}^{n+1}, \\
\dot{\Xi}^{n+1} &= \dot{\Xi}^n + \Delta t(1-\beta_1)\ddot{\Xi}^n + \Delta t\beta_1\ddot{\Xi}^{n+1},
\end{aligned}$$

4. If the equation is static, there is no $\mathbb{M}\ddot{\Xi}$ and the system can be readily used to get the displacement.

where Δt is the time step and β_1 and β_2 are parameters of the Newmark time integration. Note that this time integration is an implicit scheme if $\beta_2 \neq 0$. Choosing both β_1 and β_2 to 0.5 gives an unconditionally stable second-order scheme and presume a constant acceleration in the considered time interval $[t_n, t_{n+1}]$. The Newmark scheme is used to solve equation (B.13) with two initial conditions $\Xi^0 = \Xi_0$ and $\dot{\Xi}^0 = \mathbf{V}_0$ that are the sampling of respectively $\boldsymbol{\xi}_0$ and $\mathbf{v}_0$ on the nodes of mesh. The initial acceleration is therefore deduced using the semi-discrete equation:

$$\mathbb{M}\ddot{\Xi}^0 = F^0 - \mathbb{K}\Xi^0.$$

The principle of the scheme is to compute first the predicted displacement vector Ξ_p^{n+1} and the predicted velocity vector $\dot{\Xi}_p^{n+1}$:

$$\begin{aligned}\Xi_p^{n+1} &= \Xi^n + \Delta t\dot{\Xi}^n + \frac{\Delta t^2}{2}(1-\beta_2)\ddot{\Xi}^n,\\ \dot{\Xi}_p^{n+1} &= \dot{\Xi}^n + \Delta t(1-\beta_1)\ddot{\Xi}^n.\end{aligned}$$

The acceleration vector at the step $n+1$ is the obtained by solving:

$$\left(\mathbb{M} + \frac{\beta_2 \Delta t^2 \mathbb{K}}{2}\right)\ddot{\Xi}^{n+1} = F^{n+1} - \mathbb{K}\Xi_p^{n+1}. \tag{B.14}$$

The displacement Ξ^{n+1} and the velocity $\dot{\Xi}^{n+1}$ are then updated:

$$\begin{aligned}\Xi^{n+1} &= \Xi^n + \Delta t\dot{\Xi}_p^{n+1} + \frac{\Delta t^2}{2}\beta_2\ddot{\Xi}^{n+1},\\ \dot{\Xi}^{n+1} &= \dot{\Xi}_p^{n+1} + \Delta t\beta_1\ddot{\Xi}^{n+1}.\end{aligned}$$

B.8 Boundary conditions

Neumann boundary conditions are *natural* boundary conditions and are consequently naturally taken into account by the finite element resolution. On the contrary, Dirichlet boundary conditions are *essential* boundary conditions and the displacement has to be strongly imposed on the domain boundaries using a pseudo-elimination technique. For the static system, this means that if a vertex I is on the boundary, we set $\mathbb{K}_{II} = I_d$, $\mathbb{K}_{IJ} = 0\ \forall J \neq I$ and $F_I = \begin{pmatrix} \xi_{D,I,1}, & \xi_{D,I,2}, & \xi_{D,I,3} \end{pmatrix}^T$ in the linear system, with $\begin{pmatrix} \xi_{D,I,1}, & \xi_{D,I,2}, & \xi_{D,I,3} \end{pmatrix}^T$ the imposed displacement at vertex I. For the dynamic system, this is a bit more tricky, as it needs also to be set in the advance in time, but the idea remains the same.

Appendix C

Numbering and storage of high-order entities

In this appendix, we describe how to deal with the high-order entities in terms of numbering and storage issues. These strategies are used in `Vizir`, `Feflo.a/AMG-CADSurf` and `Wolf-{PkCurved,Spyder,MovMsh,Bloom}`.

C.1 Numbering of the elements

The numbering of the elements is crucial as it directly impacts the data structure of the used code. A standard procedure to number high-order entities is to do it in a hierarchical fashion. First the vertices are stored, then the remaining high-order nodes belonging to the edges are added according to the ordering list of edges, then the remaining high-order nodes belonging to the faces (quad/tri) are added according to the ordering list of faces. Finally, the remaining points are high-order inner volume points this one are added. In most of the cases, the high-order nodes on a face and in a volume are added following the numbering of a high-order element of a lower degree. The first task is consequently to define the ordering procedure for this various entities. By convention, the indices of an array will be denoted from 0 to $size - 1$ as it is the case in `C` or `Python`.

C.1.1 Ordering of the sub-entities

Ordering of the vertices

In this section, the indices are denoted by their Bézier indices.

Edge. The vertices of an edge of degree n are defined as:

```
EdgVertices = ((0),(n))
```

Triangle. The vertices of a triangle of degree n are defined as:

```
TriVertices = ((n,0,0),(0,n,0),(0,0,n))
```

Quadrilateral. The vertices of a quadrilateral of degree n are defined as:

```
QuaVertices = ((0,0),(n,0),(n,n),(0,n))
```

Tetrahedron. The vertices of a tetrahedron of degree n are defined as:

```
TetVertices = ((n,0,0,0),(0,n,0,0),(0,0,n,0),(0,0,0,n))
```

Hexahedron. The vertices of an hexahedron of degree n are defined as:

```
HexVertices = ((0,0,0),(n,0,0),(n,n,0),(0,n,0),(0,0,n),(n,0,n),
               (n,n,n),(0,n,n))
```

Prism. The vertices of a prism of degree n are defined as:

```
PriVertices = ((n,0,0,0),(0,n,0,0),(0,0,n,0),(n,0,0,n),(0,n,0,n),
               (0,0,n,n))
```

Pyramid. The vertices of a prism of degree n are defined as:

```
PyrVertices = ((0,0,0),(n,0,0),(n,n,0),(0,n,0),(0,0,n))
```

Ordering of the edges

Following the ordering of the vertices defined in the previous section, we are now able to define an ordering for the edges of the elements. Note that the definition of an edges ordering is not unique. We illustrate it on the case of the triangle.

Triangle. In this case, it is possible to have different orderings. A first edges ordering for the triangle is:

```
TriEdges = ((0,1),(1,2),(2,0))
```

On the contrary, another ordering is:

```
TriEdges = ((1,2),(2,0),(0,1))
```

Quadrilateral. An edges ordering for the quadrilateral is:

```
QuaEdges = ((0,1),(1,2),(2,3),(3,0))
```

Tetrahedron. An edges ordering for the tetrahedron is:

```
TetEdges = ((0,1),(1,2),(2,0),(0,3),(1,3),(2,3))
```

Hexahedron. An ordering for the hexahedron is:

```
HexEdges = ((0,1),(1,2),(2,3),(3,0),(4,5),(5,6),(6,7),(7,4),
            (0,4),(1,5),(2,6),(3,7))
```

Prism. An edges ordering for the prism is:

```
PriEdges = ((0,1),(1,2),(2,0),(3,4),(4,5),(5,3),(0,3),(1,4),(2,5))
```

Pyramid. An ordering for the pyramid is:

```
PyrEdges = ((0,1),(1,2),(2,3),(3,0),(4,0),(4,1),(4,2),(4,3))
```

Ordering of the faces

In a same manner, we are also able to define an ordering for the faces (if applicable) of the elements. Note that the definition of a faces ordering is not unique. We illustrate it on the case of the tetrahedron.

Tetrahedron. In this case, it is possible to have different orderings. A first faces ordering for the tetrahedron is:

```
TetFaces = ((0,1,2),(0,1,3),(1,2,3),(2,0,3))
```

On the contrary, another ordering is:

```
TetFaces = ((1,2,3),(2,0,3),(0,1,3),(0,1,2))
```

Hexahedron. A faces ordering for the hexahedron is:

```
HexFaces = ((0,1,2,3),(4,5,6,7),(0,1,5,4),
            (1,2,6,5),(2,3,7,6),(3,0,4,7))
```

Prism. A faces ordering for the prism is:

```
PriFaces = ((0,1,2),(3,4,5),(0,1,4,3),(1,2,5,4),(2,0,3,5))
```

Prism. A faces ordering for the pyramid is:

```
PyrFaces = ((0,1,2,3),(0,1,4),(1,2,4),(2,3,4),(3,0,4))
```

C.1.2 Numbering of the entities

Using the ordering, we are now able to define a numbering for the elements. In the following, we give a sketch of what a numbering generator would be. For the sake of both genericity and compactness, these algorithms are simplified and cannot be applied as is. In practice, this type of algorithms is used to generate C/C++ header files according a given ordering convention.

Numbering of an edge

Given its two extremities given in Bézier indices, the function which creates the array containing all the Bézier indices is:

```
def EdgNumbering(n):

L = []

Ver = EdgVertices

if ( n == 0 ) :
  L.append((0)*size(Ver[0]))

elif ( n > 0 ):

  #P1 extremities
  L.append(Ver[0])
  L.append(Ver[1])

  #H0 entities
  for i in range(1,n):
    L.append((n-i)/n*Ver[0]+i/n*Ver[1]))
return L
```

Numbering of a face

Given its extremities given in Bézier indices, the generic function which creates the array containing all the Bézier indices for quadrilateral/triangle is given in the following. The key feature is to realize that an element whose exterior nodes are removed is in terms if node distributions an element of a lower degree $nsub$ where $nsub = n - 2$ for quads and $n - 3$ for triangles.

```
def FacNumbering(n):

L = []

Ver = FacVertices

if ( n == 0 ) :
  L.append((0)*size(Ver[0]))

elif ( n > 0 ):

  #P1 extremities
  for i in range(size(Ver)) :
    L.append(Ver[i])

  #HO entities
  #Get edges of the considered element
  Edg = EltEdges
  #create node from these edges
  for i in range(size(Edg)) :
    for j in range(1,n):
      L.append((n-j)/n*Ver[Edg[0]]+j/n*Ver[Edg[1]]))
  #Get remaining nodes by using the nesting HO entity
  #of degree nsub numbering
  #and applying the appropriate shift to the indices
  if (nsub >= 0 ) :
    Lsub = FacNumbering(nsub)
    for i in range(size(Lsub)) :
      L.append(Lsub[i] + (1)*size(Lsub[i]))
return L
```

Of course, a dedicated implementation should be done for each type of face.

Numbering of a volume element

Given its extremities given in Bézier indices, the generic function which creates the array containing all the Bézier indices for volume elements (apart form the prism) is the following. The key feature is to realize that an element whose exterior nodes are removed is in terms if node distributions an element of a lower degree $nsub$ where $nsub = n - 2$ for hexahedra, $n - 3$ for pyramids and $n - 4$ for tetrahedra.

```
def VolNumbering(n):

L = []

Ver= VolVertices

if ( n == 0 ) :
  L.append((0)*size(Ver[0]))

elif ( n > 0 ):

  #P1 extremities
  for i in range(size(Ver)) :
    L.append(Ver[i])

  #HO entities
  #Get edges of the considered element
  Edg = EltEdges
  #create node from these edges
  for i in range(size(Edg)) :
    for j in range(1,n):
      L.append((n-j)/n*Ver[Edg[0]]+j/n*Ver[Edg[1]]))
  #Get faces of the considered element
  Fac = EltFaces
  #create node from these faces by using nested faces
  #of degree nsub numbering.
  #and applying the appropriate shift to the indices
  #to get the volume indices
  for i in range(size(Fac)) :
    LsubFac = FacNumbering(nsubFac)
      for i in range(size(Lsub)) :
        L.append(FacShift(LsubFac))
  #Get remaining nodes by using the nesting HO entity
  #of degree nsub numbering
  #and applying the appropriate shift to the indices
  if (nsub >= 0 ) :
    Lsub = VolNumbering(nsub)
    for i in range(size(Lsub)) :
      L.append(Lsub[i] + (1)*size(Lsub[i]))
return L
```

Of course, a dedicated implementation should be done for each type of face.

The case of the prism is different as it is the only element that does not have a nested prism of lower degree. The remaining distribution of nodes after the exterior nodes are removed does not match the distribution of a prism of lower degree. A special procedure should be done to get the numbering.

```
def PriNumbering(n):

L = []

Ver= PriVertices

if ( n == 0 ) :
  L.append((0)*size(Ver[0]))

elif ( n > 0 ):

  #P1 extremities
  for i in range(size(Ver)) :
    L.append(Ver[i])

  #HO entities
  #Get edges of the considered element
  Edg = PriEdges
  #create node from these edges
  for i in range(size(Edg)) :
    for j in range(1,n):
      L.append((n-j)/n*Ver[Edg[0]]+j/n*Ver[Edg[1]]))
  #Get faces of the considered element
  Fac = PriFaces
  #create node from these faces by using nested faces
  #of degree nsub numbering
  #and applying the appropriate shift to the indices
  for i in range(size(Fac)) :
    LsubFac = FacNumbering(nsubFac)
  for i in range(size(Lsub)) :
    L.append(FacShift(LsubFac))
  #sub entity inside
  #we see it as a stack of n-1 triangles of degree n-3
  ninter = n-2
  t = 1
  while ( ninter >= 0 ) :
    L1 = TriNumbering(n-3)
    if ( ninter == 0 ):
      for i in range(len(L1)):
        L.append((1+L1[i][0],1+L1[i][1],1+L1[i][2],t))

      else :
        for i in range(len(L1)):
          L.append((1+L1[i][0],1+L1[i][1],1+L1[i][2],t))

      for i in range(len(L1)):
        L.append((1+L1[i][0],1+L1[i][1],1+L1[i][2],n-t))

    t      = t + 1
    ninter = ninter - 2
return L
```

C.2 Storage of high-order entities

As it has been shown, the chosen strategy in this thesis is to deal with high-order entities using its Bézier indices as it definitely gives a better understanding of what a given node represents. However, such indices have to be converted into a one dimensional function for storage purpose. For each element, the following storage functions are used:

```
# position in an array of an entity of degree d defined on an edg
def posEdg(d,i):

return j

# position in an array of an entity of degree d defined on a tri
def posTri(n,i,j,k):

return int(i * n  - ((i - 1) * i) / 2 +  (n - j) )

# position in an array of an entity of degree d defined on a qua
def posQua(n,i,j):

return int(i * (n + 1) + j)


# position in an array of an entity of degree d defined on a tet
def posTet(d,i,j,k,l):

return  int( ( 3*n*n*i  + 3*n*n  + 6*n*i + 3*d + 8*i - 3*n*i*i
        - 3*i*i + i*i*i - 6*n*j + 6*i*j + 3*j*j - 3*j + 6*k)/6)

# position in an array of an entity of degree d defined on a pri
def posPri(n,i,j,k,l):

return int( (l*(n+2)*(n+1))/2 + i * n - ((i - 1) * i )/2 + (n - j))

# position in an array of an entity of degree d defined on a pyr
def posPyr(n,i,j,k) :

return int((k+1)*(2*k*k -6*n*k - 5*k + 6*n*d + 12*n + 6)/6
           - i*(n+1-k) - j - 1)

# position in an array of an entity of degree d defined on an hex
def posHex(n,i,j,k):

return int(i * (n + 1)* (n + 1) + j*(n+1) + k)
```

These position functions enable to avoid the loss of space that would be created by considering elements stored in multi-dimensional arrays whereas it is not necessary for most of the elements. Note that these functions of compression work only if the input integers are effectively Bézier indices of the considered element of degree n.

Finally, to get an inverse function to entity numbering, the following function is considered:

```
#set array M so that (for instance)
#if TriNumbering[idx] = [i,j,k] then M[posTri(n,i,j,k)] = idx
def Bez2Lag(n):

  L = EntNumbering(n)

  size - len(L)

  M = [-1]*size

  for i in range(len(L)) :
    idx = posEdg(n,L)
    M[idx] = i
return(M)
```

Of course, a dedicated function should be made for each entity.

Bibliography

[Abgrall et al. 2014a] R. Abgrall, H. Beaugendre and C. Dobrzynski. *An immersed boundary method using unstructured anisotropic mesh adaptation combined with level-sets and penalization techniques.* Journal of Computational Physics, vol. 257, pages 83–101, 2014. Cited on pages 34 and 51.

[Abgrall et al. 2014b] R. Abgrall, C. Dobrzynski and A. Froehly. *A method for computing curved meshes via the linear elasticity analogy, application to fluid dynamics problems.* International Journal for Numerical Methods in Fluids, vol. 76, nb. 4, pages 246–266, 2014. Cited on pages 88, 91, 175, and 227.

[Alauzet and Frazza 2019] F. Alauzet and L. Frazza. *3D RANS anisotropic mesh adaptation on the high-lift version of NASA's Common Research Model (HL-CRM).* In AIAA Aviation 2019 Forum, AIAA Paper 2019-3675, Dallas, Tx, USA, Jun 2019. Cited on pages 11 and 19.

[Alauzet and Loseille 2010] F. Alauzet and A. Loseille. *High-order sonic boom modeling based on adaptive methods.* Journal of Computational Physics, vol. 229, nb. 3, pages 561 – 593, 2010. Cited on pages 10 and 19.

[Alauzet and Marcum 2015] F. Alauzet and D. Marcum. *A closed advancing-layer method with connectivity optimization based mesh movement for viscous mesh generation.* Eng. w. Comp., vol. 31, pages 545–560, 2015. Cited on pages 10, 18, 189, 191, and 192.

[Alauzet and Mehrenberger 2010] F. Alauzet and M. Mehrenberger. *P1-conservative solution interpolation on unstructured triangular meshes.* International Journal for Numerical Methods in Engineering, vol. 84, nb. 13, 2010. Cited on page 52.

[Alauzet et al. 2017] F. Alauzet, A. Loseille and D. Marcum. *On a robust boundary layer mesh generation process.* In 55th AIAA Aerospace Sciences Meeting, AIAA Paper 2017-0585, Grapevine, TX, USA, Jan 2017. Cited on pages 10, 18, 189, 191, and 192.

[Alauzet 2003] F. Alauzet. *Adaptation de maillage anisotrope en trois dimensions. Application aux simulations instationnaires en Mécanique des Fluides.* Ph.D. Thesis, Université de Montpellier, October 2003. Cited on page 34.

[Alauzet 2014] F. Alauzet. *A changing-topology moving mesh technique for large displacements.* Engineering with Computers, vol. 30, nb. 2, pages 175–200, 2014. Cited on pages 171, 182, and 187.

[Alauzet 2016] F. Alauzet. *A parallel matrix-free conservative solution interpolation on unstructured tetrahedral meshes.* Computer Methods in Applied Mechanics and Engineering, vol. 299, pages 116–142, 2016. Cited on page 53.

[Alauzet 2019] F. Alauzet. `Wolf` *documentation. A Navier-Stokes flow solver based on the Mixed Element-Volume numerical scheme.* Internal report, INRIA, 2019. Cited on pages 8, 16, and 35.

[Alleaume 2009] A. Alleaume. Automatic non-manifold topology recovery and geometry noise removal, pages 267–279. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. Cited on page 151.

[Amari et al. 2018] T. Amari, A. Canou, J.-J. Aly, F. Delyon and F. Alauzet. *Magnetic cage and rope as the key for solar eruptions.* Nature, vol. 554, pages 211–215, 2018. Cited on page 34.

[Angot et al. 1999] P. Angot, G.-H. Bruneau and P. Fabrie. *A penalization method to take into account obstacles in incompressible flows.* Numerische Mathematik, vol. 81, pages 497–520, 1999. Cited on page 51.

[Ansys Inc.] Ansys Inc. *Ensight.* https://www.ansys.com/en/products/platform/ansys-ensight. Cited on page 199.

[Aparicio-Estrems et al. 2019] G. Aparicio-Estrems, A. Gargallo-Peiró and X. Roca. *Defining a Stretching and Alignment Aware Quality Measure for Linear and Curved 2D Meshes.* In 27th International Meshing Roundtable, pages 37–55, Cham, 2019. Springer International Publishing. Cited on pages 136, 138, 148, and 229.

[Aubry and Löhner 2009] R. Aubry and R. Löhner. *Generation of viscous grids at ridges and corners.* International Journal for Numerical Methods in Engineering, vol. 77, nb. 9, pages 1247–1289, 2009. Cited on pages 10 and 18.

[Aubry et al. 2015] R. Aubry, S. Dey, E.L. Mestreau, B.K. Karamete and D. Gayman. *A robust conforming NURBS tessellation for industrial applications based on a mesh generation approach.* Computer-Aided Design, vol. 63, pages 26 – 38, 2015. Cited on page 152.

[Babuska et al. 1981] I. Babuska, B. A. Szabo and I. N. Katz. *The p-version of the finite element method.* SIAM journal on numerical analysis, vol. 18, nb. 3, pages 515–545, 1981. Cited on page 33.

[Baffet et al. 2019] D. H. Baffet, M. J. Grote, S. Impériale and M. Kachanovska. *Energy Decay and Stability of a Perfectly Matched Layer For the Wave Equation.* Journal of Scientific Computing, Nov 2019. Cited on page 219.

[Bank and Smith 1993] R. E. Bank and R. K. Smith. *A posteriori error estimates based on hierarchical bases.* SIAM Journal on Numerical Analysis, vol. 30, nb. 4, pages 921–935, 1993. Cited on page 49.

[Barral and Alauzet 2014] N. Barral and F. Alauzet. *Large displacement body-fitted FSI simulations using a mesh-connectivity-change moving mesh strategy.* In 44th AIAA Fluid Dynamics Conference, AIAA Paper 2014-2773, Atlanta, GA, USA, Jun 2014. Cited on page 189.

[Barral et al. 2014] N. Barral, E. Luke and F. Alauzet. *Two mesh deformation methods coupled with a changing-connectivity moving mesh method for CFD applications.* In Proceedings of the 23th International Meshing Roundtable. Elsevier, 2014. Cited on pages 189 and 198.

[Barral et al. 2015] N. Barral, F. Alauzet and A. Loseille. *Metric-based anisotropic mesh adaptation for three-dimensional time-dependent problems involving moving geometries.* In 53th AIAA Aerospace Sciences Meeting, AIAA Paper 2015-2039, Orlando, FL, USA, Jan 2015. Cited on page 189.

[Barral et al. 2017] N. Barral, G. Olivier and F. Alauzet. *Time-accurate anisotropic mesh adaptation for three-dimensional time-dependent problems with body-fitted moving geometries.* Journal of Computational Physics, vol. 331, pages 157 – 187, 2017. Cited on page 229.

[Barral 2015] N. Barral. *Time-accurate anisotropic mesh adaptation for three-dimensional moving mesh problems.* Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, November 2015. Cited on pages 34 and 172.

[Bassi and Rebay 1997] F. Bassi and S. Rebay. *High-order accurate discontinuous finite element solution of the 2D Euler equations.* Journal of computational physics, vol. 138, nb. 2, pages 251–285, 1997. Cited on pages 9, 17, and 87.

[Batten et al. 1997] P. Batten, N. Clarke, C. Lambert and D. M. Causon. *On the Choice of Wavespeeds for the HLLC Riemann Solver*. SIAM Journal on Scientific Computing, vol. 18, nb. 6, pages 1553–1570, November 1997. Cited on pages 37 and 38.

[Bécache et al. 2010] E. Bécache, P. Ciarlet Jr, C. Hazard and E. Lunéville. La méthode des eléments finis. de la théorie à la pratique. ii. compléments. Coll. Les Cours, Les Presses de l'ENSTA, 2010. Cited on page 240.

[Belda-Ferrín et al. 2018] G. Belda-Ferrín, Abel Gargallo-Peiró and R. Roca. *Local bisection for conformal refinement of unstructured 4D simplicial meshes*. In International Meshing Roundtable, pages 229–247. Springer, 2018. Cited on page 229.

[Benek et al. 1985] J.A. Benek, P.G. Buning and J.L. Steger. *A 3D Chimera Grid Embedding Technique*. In 7th AIAA Computational Fluid Dynamics Conference, AIAA Paper 1985-1523, Cincinnati, OH, USA, Jul 1985. Cited on page 33.

[Berger and Oliger 1984] M. J. Berger and J. Oliger. *Adaptive mesh refinement for hyperbolic partial differential equations*. Journal of computational Physics, vol. 53, nb. 3, pages 484–512, 1984. Cited on page 34.

[Bergot et al. 2010] M. Bergot, G. Cohen and M. Duruflé. *Higher-order Finite Elements for Hybrid Meshes Using New Nodal Pyramidal Elements*. Journal of Scientific Computing, vol. 42, nb. 3, pages 345–381, Mar 2010. Cited on page 133.

[Bézier 1986] P. Bézier. Courbes et surfaces. Hermès, 1986. Cited on page 91.

[Billon 2016] L. Billon. *Boundary-volume mesh generation and adaptation for turbulent flows around complex geometries*. Ph.D. Thesis, PSL Research University, December 2016. Cited on pages 33 and 229.

[Blacker and Stephenson 1991] T. D. Blacker and M. B. Stephenson. *Paving: A new approach to automated quadrilateral mesh generation*. International Journal for Numerical Methods in Engineering, vol. 32, nb. 4, pages 811–847, 1991. Cited on pages 8 and 16.

[Bonnet 1999] M. Bonnet. Boundary integral equations methods in solids and fluids. John Wiley and sons, 1999. Cited on pages 7 and 15.

[Borouchaki and George 1996] H. Borouchaki and P. L. George. *Maillage de surfaces paramétriques. Partie I: Aspects théoriques*. Research Report RR-2928, INRIA, 1996. Cited on page 152.

[Borouchaki and George 2017a] H. Borouchaki and P. L. George. Maillage, modélisation géométrique et simulation numérique 1: Fonctions de forme, triangulations et modélisation géométrique, Volume 1. ISTE Group, 2017. Cited on pages 3 and 16.

[Borouchaki and George 2017b] H. Borouchaki and P. L. George. Meshing, Geometric Modeling and Numerical Simulation 1: Form Functions, Triangulations and Geometric Modeling. John Wiley & Sons, 2017. Cited on pages 3, 8, 91, 95, 97, 100, 105, 106, 108, 116, 122, 232, and 236.

[Borouchaki et al. 2000] H. Borouchaki, P. Laug and P. L. George. *Parametric surface meshing using a combined advancing-front – generalized-Delaunay approach*. International Journal for Numerical Methods in Engineering, pages 233–259, 2000. Cited on pages 7, 8, 15, and 16.

[Borouchaki et al. 2005] H. Borouchaki, P. Laug, A. Cherouat and K. Saanouni. *Adaptive remeshing in large plastic strain with damage*. International Journal for Numerical Methods in Engineering, vol. 63, nb. 1, pages 1–36, 2005. Cited on page 34.

[Borouchaki et al. 2010] H. Borouchaki, T. Grosges and D. Barchiesi. *Improved 3D adaptive remeshing scheme applied in high electromagnetic field gradient computation.* Finite Elements in Analysis and Design, vol. 46, nb. 1, pages 84 – 95, 2010. Cited on page 34.

[Bowyer 1981] A. Bowyer. *Computing Dirichlet tessellations.* The Computer Journal, vol. 24, nb. 2, pages 162–166, 01 1981. Cited on page 232.

[Brière de L'isle and George 1995] E. Brière de L'isle and P. L. George. *Optimization of Tetrahedral Meshes.* In I. Babuska, W. D. Henshaw, J. E. Oliger, J. E. Flaherty, J. E. Hopcroft and T. Tezduyar, Editors, Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations, pages 97–127, New York, NY, 1995. Springer New York. Cited on pages 135, 171, and 237.

[Caplan et al. 2019] P. Caplan, R. Haimes, D. Darmofal and M. Galbraith. *Extension of local cavity operators to 3d+t space-time mesh adaptation.* In AIAA Scitech 2019 Forum, 2019. Cited on page 229.

[Castro-Díaz et al. 1997] M. J. Castro-Díaz, F. Hecht, B. Mohammadi and O. Pironneau. *Anisotropic unstructured mesh adaption for flow simulations.* International Journal for Numerical Methods in Fluids, vol. 25, nb. 4, pages 475–491, 1997. Cited on pages 10 and 19.

[Chaillat et al. 2018] S. Chaillat, S. Groth and A. Loseille. *Metric-based anisotropic mesh adaptation for 3D acoustic boundary element methods* . Journal of Computational Physics, vol. 372, pages 473–499, 11 2018. Cited on pages 34 and 218.

[Chan and Warburton 2016] J. Chan and T. Warburton. *A Short Note on a Bernstein–Bezier Basis for the Pyramid.* SIAM Journal on Scientific Computing, vol. 38, nb. 4, pages A2162–A2172, 2016. Cited on pages 116 and 133.

[Chrisochoides and Nave 2003] N. Chrisochoides and D. Nave. *Parallel Delaunay mesh generation kernel.* International Journal for Numerical Methods in Engineering, vol. 58, nb. 2, pages 161–176, 2003. Cited on pages 9 and 17.

[Ciarlet and Raviart 1972] P. G. Ciarlet and P.-A. Raviart. *The combined effect of curved boundaries and numerical integration in isoparametric finite element methods.* In The mathematical foundations of the finite element method with applications to partial differential equations, pages 409–474. Elsevier, 1972. Cited on pages 9, 17, and 87.

[Ciarlet Jr and Lunéville 2009] P. Ciarlet Jr and E. Lunéville. La méthode des eléments finis. de la théorie à la pratique. i. concepts généraux. Coll. Les Cours, Les Presses de l'ENSTA, 2009. Cited on page 91.

[Ciarlet 1978] P. G. Ciarlet. The Finite Element Method for Elliptic Problems. North-Holland Publishing Company, 1978. Cited on pages 7, 15, 91, 118, 249, 251, and 254.

[Clément 1975] P. Clément. *Approximation by finite element functions using local regularization.* ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique, vol. 9, nb. R2, pages 77–84, 1975. Cited on page 50.

[Coulaud and Loseille 2016] O. Coulaud and A. Loseille. *Very High Order Anisotropic Metric-Based Mesh Adaptation in 3D.* Procedia Engineering, vol. 163, pages 353–364, 2016. Cited on pages 11, 12, 19, 49, 152, 154, 155, 160, 162, 215, and 228.

[Dantzig and Thapa 2003] G. B. Dantzig and M. N. Thapa. Linear programming 2: Theory and extensions. Springer-Verlag, 2003. Cited on page 155.

[Dapogny et al. 2014] C. Dapogny, C. Dobrzynski and P. Frey. *Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems.* Journal of Computational Physics, vol. 262, pages 358 – 378, 2014. Cited on page 171.

[de Casteljau 1959] P. de Casteljau. *Outillages, méthodes, calcul.* André Citroën Automobiles SA, Paris, 1959. Cited on page 91.

[de Cougny and Shephard 1996] H. L. de Cougny and M. S. Shephard. *Surface meshing using vertex insertion.* In Proceedings of the 5th International Meshing Roundtable, pages 243–256, 1996. Cited on page 152.

[de Siqueira et al. 2010] D. de Siqueira, J. B. Cavalcante-Neto, C. A. Vidal and R. J. da Silva. *A Hierarchical Adaptive Mesh Generation Strategy for Parametric Surfaces Based on Tree Structures.* In 2010 23rd SIBGRAPI Conference on Graphics, Patterns and Images, pages 79–86. IEEE, 2010. Cited on page 152.

[de Siqueira et al. 2014] D. M. B. de Siqueira, M. O. Freitas, J. B. Cavalcante-Neto, C. A. Vidal and R. J. da Silva. *An Adaptive Parametric Surface Mesh Generation Method Guided by Curvatures.* In Proceedings of the 22nd International Meshing Roundtable, 2014. Cited on page 152.

[Debiez and Dervieux 2000] C. Debiez and A. Dervieux. *Mixed-element-volume MUSCL methods with weak viscosity for steady and unsteady flow calculations.* Computers & Fluids, vol. 29, nb. 1, pages 89 – 118, 2000. Cited on pages 39 and 62.

[Delaunay 1934] B. Delaunay. *Sur la sphère vide.* Изв. Акад. Наук СССР, Отделение Математическии и Естественныка Наук, vol. 7, pages 793–800, 1934. Cited on page 232.

[Dey et al. 1999] S. Dey, R. M. O'bara and M. S. Shephard. *Curvilinear Mesh Generation in 3D.* In Proceedings of the 7th International Meshing Roundtable, pages 407–417, 1999. Cited on pages 9, 18, 88, 135, and 175.

[Do Carmo 2016] M. P. Do Carmo. Differential geometry of curves and surfaces: Revised and updated second edition. Courier Dover Publications, 2016. Cited on pages 156 and 157.

[Dobrzynski and Frey 2008] C. Dobrzynski and P. Frey. *Anisotropic Delaunay Mesh Adaptation for Unsteady Simulations.* In R. V. Garimella, Editor, Proceedings of the 17th International Meshing Roundtable, pages 177–194, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. Cited on page 171.

[Dobrzynski 2005] C. Dobrzynski. *3D anisotropic mesh adaptation and application for air-conditionning.* Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, November 2005. Cited on page 34.

[Dompierre et al. 2005] J. Dompierre, M.-G. Vallet, P. Labbé and F. Guibault. *An analysis of simplex shape measures for anisotropic meshes.* Computer Methods in Applied Mechanics and Engineering, vol. 194, nb. 48, pages 4895 – 4914, 2005. Cited on pages 134, 135, and 137.

[Encinas and Masque 2003] L. H. Encinas and J. M. Masque. *A short proof of the generalized Faà di Bruno's formula.* Applied mathematics letters, vol. 16, nb. 6, pages 975–979, 2003. Cited on page 161.

[Faà di Bruno 1857] F. Faà di Bruno. *Note sur une nouvelle formule de calcul différentiel.* Quarterly J. Pure Appl. Math, vol. 1, nb. 359-360, page 12, 1857. Cited on page 159.

[Farin 1986] G. Farin. *Triangular Bernstein-Bézier patches.* Computer Aided Geometric Design, vol. 3, nb. 2, pages 83 – 127, 1986. Cited on page 97.

[Feuillet et al. 2018] R. Feuillet, A. Loseille, D. Marcum and F. Alauzet. *Connectivity-change moving mesh methods for high-order meshes: Toward closed advancing-layer high-order boundary layer mesh generation.* In 24th AIAA Fluid Dynamics Conference, AIAA Paper 2018-4167, Atlanta, GA, USA, Jun 2018. Cited on pages 189 and 198.

[Feuillet et al. 2019a] R. Feuillet, O. Coulaud and A. Loseille. *Anisotropic error estimate for high-order parametric surface mesh generation.* In 28th International Meshing Roundtable, 2019. Cited on page 170.

[Feuillet et al. 2019b] R. Feuillet, A. Loseille and F. Alauzet. *Mesh adaptation for the embedded boundary method in CFD.* Research Report RR-9305, INRIA Saclay - Ile-de-France, November 2019. Cited on page 83.

[Feuillet et al. 2019c] R. Feuillet, A. Loseille and F. Alauzet. *Optimization of P2 meshes and applications.* Computer Aided Deisgn, vol. x, nb. x, pages x–x, 2019. Submitted. Cited on page 198.

[Feuillet et al. 2019d] R. Feuillet, A. Loseille and F. Alauzet. *P2 Mesh Optimization Operators.* In 27th International Meshing Roundtable, pages 3–21, Cham, 2019. Springer International Publishing. Cited on page 198.

[Feuillet et al. 2019e] R. Feuillet, D. Marcum and F. Alauzet. *A closed advancing-layer method for generating curved boundary layer mesh.* In AIAA Aviation 2019 Forum, AIAA Paper 2019-3675, Dallas, Tx, USA, Jun 2019. Cited on page 198.

[Fidkowski and Darmofal 2007a] K.J. Fidkowski and D.L. Darmofal. *An adaptive simplex cut-cell method for discontinuous Galerkin discretizations of the Navier-Stokes equations.* In 18th AIAA Computational Fluid Dynamics Conference, AIAA-2007-3941, Miami, FL, USA, Jun 2007. Cited on pages 33, 51, and 57.

[Fidkowski and Darmofal 2007b] K.J. Fidkowski and D.L. Darmofal. *Triangular cut-cell adaptive method for high-order discretizations of the compressible Navier-Stokes equations.* Journal of Computational Physics, vol. 225, pages 1653–1672, 2007. Cited on pages 51 and 57.

[Forsythe and Wasow 1960] G. E. Forsythe and W. R. Wasow. Finite-difference methods for partial differential equations. Wiley, 1960. Cited on pages 7 and 15.

[Fortunato and Persson 2016] M. Fortunato and P.-O. Persson. *High-order Unstructured Curved Mesh Generation Using the Winslow Equations.* J. Comput. Phys., vol. 307, pages 1–14, February 2016. Cited on pages 88, 135, and 175.

[Frazza 2018] L. Frazza. *3D anisotropic mesh adaptation for Reynolds Averaged Navier-Stokes simulations.* Ph.D. Thesis, Sorbonne Université , UPMC, December 2018. Cited on pages 11, 19, 34, 36, and 43.

[Frey and Alauzet 2005] P. J. Frey and F. Alauzet. *Anisotropic mesh adaptation for CFD computations.* Computer methods in applied mechanics and engineering, vol. 194, nb. 48-49, pages 5068–5082, 2005. Cited on pages 10, 19, and 50.

[Frey and George 1999] P. J. Frey and P. L. George. Maillages: applications aux éléments finis. Hermès Science Publications, 1999. Cited on page 16.

[Frey and George 2008] P. J. Frey and P. L George. Mesh Generation: application to finite elements, 2nd Edition. John Wiley & Sons, 2008. Cited on pages 8, 92, 136, and 236.

[Frey et al. 1998] P. J. Frey, H. Borouchaki and P.-L. George. *3D Delaunay mesh generation coupled with an advancing-front approach.* Computer methods in applied

mechanics and engineering, vol. 157, nb. 1-2, pages 115–131, 1998. Cited on pages 8 and 16.

[Frey 2000] P. J. Frey. *About surface remeshing.* Proceedings of the 9th international meshing roundtable, 2000. Cited on page 157.

[Frey 2001] P. J. Frey. *Medit: An interactive mesh visualization software, INRIA Technical Report RT0253*, 2001. Cited on page 199.

[Gargallo-Peiró et al. 2013] A. Gargallo-Peiró, X. Roca, J. Peraire and J. Sarrate. *Defining quality measures for mesh optimization on parameterized CAD surfaces.* In Proceedings of the 21st International Meshing Roundtable, pages 85–102. Springer, 2013. Cited on pages 9, 18, 88, and 135.

[Gargallo-Peiró et al. 2015] A. Gargallo-Peiró, X. Roca, J. Peraire and J. Sarrate. *Distortion and quality measures for validating and generating high-order tetrahedral meshes.* Engineering with Computers, vol. 31, nb. 3, pages 423–437, 2015. Cited on page 135.

[Gargallo-Peiró et al. 2016] A. Gargallo-Peiró, X. Roca, J. Peraire and J. 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, vol. 106, nb. 13, pages 1100–1130, 2016. Cited on page 152.

[Gargallo-Peiró 2014] A. Gargallo-Peiró. *Validation and generation of curved meshes for high-order unstructured methods.* Ph.D. Thesis, Universitat Politècnica de Catalunya, July 2014. Cited on page 135.

[Gauci 2018] E. Gauci. *Goal-oriented metric-based mesh adaptation for unsteady CFD simulations involving moving geometries.* Ph.D. Thesis, Université Côte d'Azur, December 2018. Cited on page 34.

[George and Borouchaki 1998] P. L. George and H. Borouchaki. Delaunay triangulation and meshing. Hermes, 1998. Cited on pages 134, 163, 231, 232, and 236.

[George and Borouchaki 2012] P. L. George and H. Borouchaki. *Construction of tetrahedral meshes of degree two.* International Journal for Numerical Methods in Engineering, vol. 90, nb. 9, pages 1156,1182, 2012. Cited on pages 9, 11, 18, 20, 88, 91, 121, 122, 123, 127, and 227.

[George et al. 1991a] P. L. George, F. Hecht and E. Saltel. *Automatic mesh generator with specified boundary.* Computer Methods in Applied Mechanical Engineering, vol. 92, pages 269–288, 1991. Cited on pages 8, 16, 189, and 231.

[George et al. 1991b] P.L. George, F. Hecht and M.G. Vallet. *Creation of internal points in Voronoi's type method. Control adaptation.* Advances in Engineering Software and Workstations, vol. 13, nb. 5, pages 303 – 312, 1991. Cited on pages 10 and 19.

[George et al. 2003] P. L. George, H. Borouchaki and E. Saltel. *Ultimate robustness in meshing an arbitrary polyhedron.* International Journal for Numerical Methods in Engineering, vol. 58, nb. 7, pages 1061–1089, 2003. Cited on pages 7 and 16.

[George et al. 2014] P. L. George, H. Borouchaki and N. Barral. *Construction et validation des éléments réduits associés à un carreau simplicial de degré arbitraire.* Research Report RR-8571, INRIA, 2014. Cited on page 176.

[George et al. 2016] P. L. George, H. Borouchaki and N. Barral. *Geometric validity (positive jacobian) of high-order Lagrange finite elements, theory and practical guidance.* Engineering with computers, vol. 32, nb. 3, pages 405–424, 2016. Cited on pages 11, 20, 88, 91, and 227.

[George et al. 2018] P. L. George, H. Borouchaki, F. Alauzet, P. Laug, A. Loseille and L. Maréchal. Maillage, modélisation géométrique et simulation numérique 2: Métriques, maillages et adaptation de maillages, Volume 2. ISTE Group, 2018. Cited on pages 3 and 16.

[George et al. 2019] P. L. George, H. Borouchaki, F. Alauzet, P. Laug, A. Loseille and L. Maréchal. Meshing, Geometric Modeling and Numerical Simulation 2: Metrics, Meshes and Mesh Adaptation. John Wiley & Sons, 2019. Cited on pages 3, 8, 46, 91, 135, 136, 140, and 158.

[Geuzaine and Remacle 2009] C. Geuzaine and J.-F. Remacle. *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities.* International Journal for Numerical Methods in Engineering, vol. 79, nb. 11, pages 1309–1331, 2009. Cited on pages 199 and 200.

[Gruau and Coupez 2005] C. Gruau and T. Coupez. *3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric.* Computer Methods in Applied Mechanics and Engineering, vol. 194, nb. 48, pages 4951 – 4976, 2005. Cited on pages 10 and 19.

[Hachem et al. 2013] E. Hachem, S. Feghali, R. Codina and T. Coupez. *Immersed stress method for fluid-structure interaction using anisotropic mesh adaptation.* International Journal for Numerical Methods in Engineering, vol. 94, nb. 9, pages 805–825, 2013. Cited on pages 51 and 83.

[Haimes and Drela 2012] R. Haimes and M. Drela. *On The Construction of Aircraft Conceptual Geometry for High-Fidelity Analysis and Design.* In 50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, 2012. Cited on pages 163 and 223.

[Hartmann and Leicht 2016] R. Hartmann and T. Leicht. *Generation of unstructured curvilinear grids and high-order discontinuous Galerkin discretization applied to a 3D high-lift configuration.* International Journal for Numerical Methods in Fluids, vol. 82, nb. 6, pages 316–333, 2016. Cited on pages 88 and 175.

[Hecht and Kuate 2014] F. Hecht and R. Kuate. *An approximation of anisotropic metrics from higher order interpolation error for triangular mesh adaptation.* Journal of Computational and Applied Mathematics, vol. 258, pages 99 – 115, 2014. Cited on pages 11 and 19.

[Hesthaven and Warburton 2008] J. S. Hesthaven and T. Warburton. Nodal Discontinuous Galerkin Methods: algorithms, analysis and applications. Springer Publishing Company, Incorporated, 2008. Cited on pages 7 and 15.

[Hill 1994] F.S. Hill. *II.5. - The Pleasures of "Perp Dot" Products.* In Paul S. Heckbert, Editor, Graphics Gems, pages 138 – 148. Academic Press, 1994. Cited on page 53.

[Huang et al. 2010] W. Huang, L. Kamenski and J. Lang. *A new anisotropic mesh adaptation method based upon hierarchical a posteriori error estimates.* Journal of Computational Physics, vol. 229, pages 2179–2198, 2010. Cited on pages 33 and 49.

[Hughes et al. 2005] T. J. R. Hughes, J. A. Cottrell and Y. Bazilevs. *Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement.* Computer methods in applied mechanics and engineering, pages 4135–4195, 2005. Cited on page 87.

[Ims and Wang 2019] J. Ims and Z. J. Wang. *Automated low-order to high-order mesh conversion.* Engineering with Computers, vol. 35, nb. 1, pages 323–335, Jan 2019. Cited on pages 88 and 176.

[Intelligent Light] Intelligent Light. *FieldView*. http://www.ilight.com/en/products/fieldview-18. Cited on page 199.

[Johnen and Geuzaine 2015] A. Johnen and C. Geuzaine. *Geometrical validity of curvilinear pyramidal finite elements*. Journal of Computational Physics, vol. 299, pages 124 – 129, 2015. Cited on pages 116 and 133.

[Johnen et al. 2013] A. Johnen, J.-F. Remacle and C. Geuzaine. *Geometrical validity of curvilinear finite elements*. Journal of Computational Physics, vol. 233, nb. 15, pages 359–372, 2013. Cited on pages 9, 18, 88, 91, 123, and 227.

[Johnen et al. 2017] A. Johnen, J.-C. Weill and J.-F. Remacle. *Robust and efficient validation of the linear hexahedral element*. Procedia engineering, vol. 203, pages 271–283, 2017. Cited on page 132.

[Johnen et al. 2018] A. Johnen, C. Geuzaine, T. Toulorge and J-F Remacle. *Efficient computation of the minimum of shape quality measures on curvilinear finite elements*. Computer-Aided Design, vol. 103, pages 24–33, 2018. Cited on pages 135, 140, and 141.

[Johnen 2016] A. Johnen. *Indirect quadrangular mesh generation and validation of curved finite elements*. Ph.D. Thesis, Université de Liège, February 2016. Cited on page 140.

[Karman et al. 2016] S. L. Karman, J T. Erwin, R. S. Glasby and D. Stefanski. *High-Order Mesh Curving Using WCN Mesh Optimization*. In 46th AIAA Fluid Dynamics Conference, AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, 2016. Cited on pages 88, 171, and 175.

[Karman 2019] S. L. Karman. *Curving for Viscous Meshes*. In 27th International Meshing Roundtable, pages 303–325, Cham, 2019. Springer International Publishing. Cited on page 88.

[Khronos-Group 2017] Khronos-Group. OpenCL 2.2 Reference Guide . www.khronos.org/opencl, 2017. Cited on pages 9 and 17.

[KitWare Inc.] KitWare Inc. *ParaView*. https://www.paraview.org/. Cited on page 199.

[Knupp 2001] P. Knupp. *Algebraic Mesh Quality Metrics*. SIAM Journal on Scientific Computing, vol. 23, nb. 1, pages 193–218, 2001. Cited on pages 135 and 137.

[Koren 1993] B. Koren. *A robust upwind discretization method for advection, diffusion and source terms*. In Numerical methods for advection-diffusion problems, pages 117–138. Vieweg, 1993. Cited on page 40.

[Laug 2010] P. Laug. *Some aspects of parametric surface meshing*. Finite Elements in Analysis and Design, vol. 46, nb. 1-2, pages 216 – 226, 2010. Cited on pages 152 and 228.

[Law 1986] K. H. Law. *A parallel finite element solution method*. Computers & Structures, vol. 23, nb. 6, pages 845–858, 1986. Cited on pages 9 and 17.

[Lenoir 1986] M. Lenoir. *Optimal isoparametric finite elements and error estimates for domains involving curved boundaries*. SIAM journal on numerical analysis, vol. 23, nb. 3, pages 562–580, 1986. Cited on pages 9, 17, and 87.

[Li et al. 2005] X. Li, M. S. Shephard and M. W. Beall. *3D anisotropic mesh adaptation by mesh modification*. Computer methods in applied mechanics and engineering, vol. 194, nb. 48-49, pages 4915–4950, 2005. Cited on pages 10 and 19.

[Liu and Nocedal 1989] D. C. Liu and J. Nocedal. *On the limited memory BFGS method for large scale optimization*. Mathematical Programming, vol. 45, nb. 1, pages 503–528, 1989. Cited on page 173.

[Löhner and Baum 1992] R. Löhner and J. D. Baum. *Adaptive h-refinement on 3D unstructured grids for transient problems.* International Journal for Numerical Methods in Fluids, vol. 14, nb. 12, pages 1407–1419, 1992. Cited on page 34.

[Löhner and Parikh 1988] R. Löhner and P. Parikh. *Generation of three-dimensional unstructured grids by the advancing-front method.* International Journal for Numerical Methods in Fluids, vol. 8, nb. 10, pages 1135–1149, 1988. Cited on pages 8 and 16.

[Löhner et al. 2004] R. Löhner, J.D. Baum, E. Mestreau, D. Sharov, C. Charman and D. Pelessone. *Adaptive embedded unstructured grid methods.* International Journal for Numerical Methods Engineering, vol. 60, pages 641–660, 2004. Cited on pages 9, 17, and 33.

[Löhner et al. 2007a] R. Löhner, S. Appanaboyina and J.R. Cebral. *Comparison of body-fitted, embedded and immersed solutions of low Reynolds-number 3-D incompressible flows.* In 45th AIAA Aerospace Sciences Meeting, AIAA-2007-1296, Reno, NV, USA, Jan 2007. Cited on page 33.

[Löhner et al. 2007b] R. Löhner, J.D. Baum, E.L. Mestreau and D. Rice. *Comparison of body-fitted, embedded and immersed 3-D Euler predictions for blast loads on columns.* In 45th AIAA Aerospace Sciences Meeting, AIAA-2007-1113, Reno, NV, USA, Jan 2007. Cited on page 33.

[Löhner 2000] R. Löhner. *Generation of Unstructured Grids Suitable for Rans Calculations.* In Manuel D. Salas and W. Kyle Anderson, Editors, Computational Aerosciences in the 21st Century, pages 153–163, Dordrecht, 2000. Springer Netherlands. Cited on pages 10 and 18.

[Löhner 2008] R. Löhner. Applied computational fluid dynamics techniques: an introduction based on finite element methods. John Wiley & Sons, 2008. Cited on page 33.

[Loseille and Alauzet 2011a] A. Loseille and F. Alauzet. *Continuous mesh framework part I: well-posed continuous interpolation error.* SIAM J. Numer. Anal., vol. 49, nb. 1, pages 38–60, 2011. Cited on pages 46, 154, 160, and 162.

[Loseille and Alauzet 2011b] A. Loseille and F. Alauzet. *Continuous mesh framework part II: validations and applications.* SIAM J. Numer. Anal., vol. 49, nb. 1, pages 61–86, 2011. Cited on pages 46, 47, 135, 137, 154, 160, and 162.

[Loseille and Feuillet 2018] A. Loseille and R. Feuillet. *Vizir: High-order mesh and solution visualization using OpenGL 4.0 graphic pipeline.* In 2018 AIAA Aerospace Sciences Meeting, pages AIAA 2018–1174, 2018. Cited on pages 8, 16, 163, and 224.

[Loseille and Löhner 2013] A. Loseille and R. Löhner. *Robust boundary layer mesh generation.* In Proceedings of the 21st International Meshing Roundtable, pages 493–511. Springer, 2013. Cited on page 35.

[Loseille et al. 2016] A. Loseille, H. Guillard and A. Loyer. *An introduction to Vizir: an interactive mesh visualization and modification software.* EOCOE, Rome, Italy, 2016. Cited on page 199.

[Loseille et al. 2017] A. Loseille, F. Alauzet and V. Menier. *Unique cavity-based operator and hierarchical domain partitioning for fast parallel generation of anisotropic meshes.* Computer-Aided Design, vol. 85, pages 53 – 67, 2017. 24th International Meshing Roundtable Special Issue: Advances in Mesh Generation. Cited on pages 9 and 17.

[Loseille et al. 2018] A. Loseille, L. Frazza and F. Alauzet. *Comparing Anisotropic Adaptive Strategies on the Second AIAA Sonic Boom Workshop Geometry.* Journal of Aircraft, vol. 56, nb. 3, pages 938–952, 2018. Cited on pages 11 and 19.

[Loseille 2008] A. Loseille. *Adaptation de maillage anisotrope 3D multi-échelles et ciblée à une fonctionnelle pour la mécanique des fluides : Application à la prédiction haute-fidélité du bang sonique.* Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, December 2008. Cited on pages 34 and 46.

[Loseille 2014] A. Loseille. *Metric-orthogonal Anisotropic Mesh Generation.* Procedia Engineering, vol. 82, pages 403 – 415, 2014. 23rd International Meshing Roundtable (IMR23). Cited on page 229.

[Loseille 2017] A. Loseille. *Chapter 10 - Unstructured Mesh Generation and Adaptation.* In R. Abgrall and C.-W. Shu, Editors, Handbook of Numerical Methods for Hyperbolic Problems, Volume 18 of *Handbook of Numerical Analysis*, pages 263 – 302. Elsevier, 2017. Cited on pages 7 and 15.

[Marcum and Alauzet 2014] D. Marcum and F. Alauzet. *Aligned Metric-based Anisotropic Solution Adaptive Mesh Generation.* Procedia Engineering, vol. 82, pages 428 – 444, 2014. 23rd International Meshing Roundtable (IMR23). Cited on page 229.

[Marcum and Weatherill 1995] D. L. Marcum and N. P. Weatherill. *Unstructured grid generation using iterative point insertion and local reconnection.* AIAA journal, vol. 33, nb. 9, pages 1619–1625, 1995. Cited on pages 8 and 18.

[Marcum 1995] D. Marcum. *Generation of unstructured grids for viscous flow applications.* In 33rd Aerospace Sciences Meeting and Exhibit, page 212, 1995. Cited on pages 10 and 16.

[Marcum 1998] D.L. Marcum. *Unstructured Grid Generation Using Automatic Point Insertion and Local Reconnection.* In The Handbook of Grid Generation, Edited by J.F. Thompson, B. Soni, and N.P. Weatherill, Chapter 18, pages 1–31. CRC Press, 1998. Cited on pages 189 and 191.

[Marcum 2000] D.L. Marcum. *Efficient generation of high-quality unstructured surface and volume grids.* In Proceedings of the 9th International Meshing Roundtable, New Orleans, LA, USA, 2000. Cited on page 189.

[Maréchal 2001] L. Maréchal. *A new approach to octree-based hexahedral meshing.* Proceedings of the 10th International Meshing Roundtable, 2001, pages 209–221, 2001. Cited on pages 8 and 16.

[Maréchal 2009] L. Maréchal. *Advances in octree-based all-hexahedral mesh generation: handling sharp features.* Proceedings of the 18th international meshing roundtable, pages 65–84, 2009. Cited on pages 7 and 16.

[Maréchal 2010] L. Maréchal. *The LP3 library: A parallelization framework for numerical simulation.* Technical report, INRIA, 2010. Cited on pages 9 and 17.

[Maréchal 2012] L. Maréchal. *The GM2 library: Port meshing tools that deal with unstructured meshes on GPUs.* Technical report, INRIA, 2012. Cited on pages 9 and 17.

[Maréchal 2016] L. Maréchal. *All hexahedral boundary layers generation.* Procedia engineering, vol. 163, pages 5–19, 2016. Cited on pages 10 and 18.

[Maréchal 2018] L. Maréchal. *The libMeshb library: An easy way to access files in Gamma Mesh Format.* Technical report, INRIA, 2018. Cited on pages 9 and 17.

[Marot et al. 2019] C. Marot, J. Pellerin and J.-F. Remacle. *One machine, one minute, three billion tetrahedra.* International Journal for Numerical Methods in Engineering, vol. 117, nb. 9, pages 967–990, 2019. Cited on pages 9 and 17.

[Maunoury et al. 2018] M. Maunoury, C. Besse, V. Mouysset, S. Pernet and P.-A. Haas. *Well-suited and adaptive post-processing for the visualization of hp simulation results.* Journal of Computational Physics, vol. 375, pages 1179 – 1204, 2018. Cited on pages 88 and 199.

[Maunoury 2019] M. Maunoury. *Méthode de visualisation adaptée aux simulations d'ordre élevé. Application à la compression-reconstruction de champs rayonnés pour des ondes harmoniques.* Ph.D. Thesis, Université de Toulouse, February 2019. Cited on page 199.

[Mbinky 2013] E. C. Mbinky. *Mesh adaptation for very high order numerical schemes.* Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, December 2013. Cited on pages 11 and 19.

[Menier 2015] V. Menier. *Numerical methods and mesh adaptation for reliable RANS simulations.* Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, November 2015. Cited on pages 34, 36, and 43.

[Miranda and Martha 2002] A. C. Miranda and L. F. Martha. *Mesh generation on high-curvature surfaces based on a background quadtree structure.* In Proceedings of the 11th International Meshing Roundtable, 2002. Cited on page 152.

[Mirebeau 2010] J.-M. Mirebeau. *Adaptive and anisotropic finite element approximation: Theory and algorithms.* Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, December 2010. Cited on pages 11 and 19.

[Moxey et al. 2015a] D. Moxey, Mashy D. Green, S. Sherwin and J. Peiró. *On the generation of curvilinear meshes through subdivision of isoparametric elements.* In New Challenges in Grid Generation and Adaptivity for Scientific Computing, pages 203–215. Springer, 2015. Cited on pages 124, 128, and 131.

[Moxey et al. 2015b] D. Moxey, M.D. Green, S.J. Sherwin and J. Peiró. *An isoparametric approach to high-order curvilinear boundary-layer meshing.* Computer Methods in Applied Mechanics and Engineering, vol. 283, pages 636 – 650, 2015. Cited on page 88.

[Moxey et al. 2016] D. Moxey, D. Ekelschot, Ü. Keskin, S.J. Sherwin and J. Peirò. *High-order curvilinear meshing using a thermo-elastic analogy.* Computer-Aided Design, vol. 72, pages 130 – 139, 2016. Cited on pages 88 and 175.

[MPI-Forum 2015] MPI-Forum. MPI: A Message-Passing Interface Standard, Version 3.1. `www.mpi-forum.org`, 2015. Cited on pages 9 and 17.

[Nelson et al. 2011] B. Nelson, R. Haimes and R. M. Kirby. *GPU-Based Interactive Cut-Surface Extraction From High-Order Finite Element Fields.* IEEE Transactions on Visualization and Computer Graphics, vol. 17, nb. 12, pages 1803–11, 2011. Cited on page 199.

[Nelson et al. 2012] B. Nelson, E. Liu, R. M. Kirby and R. Haimes. *ElVis: A System for the Accurate and Interactive Visualization of High-Order Finite Element Solutions.* IEEE Transactions on Visualization and Computer Graphics, vol. 18, nb. 12, pages 2325–2334, 2012. Cited on page 199.

[NVIDIA] NVIDIA. *Whitepaper WP-03014-001_v01.* Cited on page 214.

[Olivier 2011] G. Olivier. *Anisotropic metric-based mesh adaptation for unsteady CFD simulations involving moving geometries.* Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, April 2011. Cited on page 34.

[OpenMP 2018] OpenMP. OpenMP 5.0 Application Programming Interface. `www.openmp.org`, 2018. Cited on pages 9 and 17.

[Park and Darmofal 2010] M.A. Park and D.L. Darmofal. *Validation of an output-adaptive tetrahedral cut-cell method for sonic boom prediction.* AIAA Journal, vol. 48, nb. 9, pages 1928–1945, 2010. Cited on pages 51 and 57.

[Peiró et al. 2015] J. Peiró, D. Moxey, B. Jordi, S. J. Sherwin, B.W. Nelson, R. M. Kirby and R. Haimes. *High-Order Visualization with ElVis.* In Notes on Numerical Fluid Mechanics and Multidisciplinary Design, pages 521–534. Springer International Publishing, 2015. Cited on pages 9, 18, 88, and 199.

[Persson and Peraire 2009] P.-O. Persson and J. Peraire. *Curved mesh generation and mesh refinement using Lagrangian solid mechanics.* In 47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition, page 949, 2009. Cited on pages 9, 18, 88, 135, and 175.

[Peskin 1972] C. S. Peskin. *Flow patterns around heart valves: A numerical method.* Journal of Computational Physics, vol. 10, nb. 2, pages 252 – 271, 1972. Cited on page 33.

[Peyret 2017] C. Peyret. *A Full High Order Method for Computational AeroAcoustics.* In 23rd AIAA/CEAS Aeroacoustics Conference, page 3174, 2017. Cited on page 221.

[Piegl and Tiller 1997] L. Piegl and W. Tiller. The nurbs book (2nd ed.). Springer-Verlag New York, Inc., New York, NY, USA, 1997. Cited on pages 151, 156, 159, 161, and 176.

[Piperno and Depeyre 1998] S. Piperno and S. Depeyre. *Criteria for the design of limiters yielding efficient high resolution TVD schemes.* Computers & Fluids, vol. 27, nb. 2, pages 183 – 197, 1998. Cited on page 40.

[Pirzadeh 1994] S. Pirzadeh. *Unstructured viscous grid generation by the advancing-layers method.* AIAA journal, vol. 32, nb. 8, pages 1735–1737, 1994. Cited on pages 10 and 18.

[Price and Armstrong 1997] M. A. Price and C. G Armstrong. *Hexahedral mesh generation by medial surface subdivision: Part II. Solids with flat and concave edges.* International Journal for Numerical Methods in Engineering, vol. 40, nb. 1, pages 111–136, 1997. Cited on pages 8 and 16.

[Price et al. 1995] M.A. Price, C. G Armstrong and M.A. Sabin. *Hexahedral mesh generation by medial surface subdivision: Part I. Solids with convex edges.* International Journal for Numerical Methods in Engineering, vol. 38, nb. 19, pages 3335–3359, 1995. Cited on pages 8 and 16.

[Puscas et al. 2015] M. A. Puscas, L. Monasse, A. Ern, C. Tenaud and C. Mariotti. *A conservative Embedded Boundary method for an inviscid compressible flow coupled with a fragmenting structure.* International Journal for Numerical Methods in Engineering, vol. 103, nb. 13, pages 970–995, 2015. Cited on page 33.

[Quan et al. 2014a] D. Quan, T. Toulorge, G. Bricteux, J.-F. Remacle and E. Marchandise. *Anisotropic adaptive nearly body-fitted meshes for CFD.* Engineering with Computers, vol. 30, nb. 4, pages 517–533, Oct 2014. Cited on pages 33, 34, 51, 83, and 229.

[Quan et al. 2014b] D.-L. Quan, T. Toulorge, E. Marchandise, J.-F. Remacle and G. Bricteux. *Anisotropic mesh adaptation with optimal convergence for finite elements using embedded geometries.* Computer Methods in Applied Mechanics and Engineering, vol. 268, pages 65 – 81, 2014. Cited on pages 33, 34, and 51.

[Rebay 1993] S. Rebay. *Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm.* Journal of Computational Physics, vol. 106, nb. 1, pages 125 – 138, 1993. Cited on pages 8 and 16.

[Remacle et al. 2007] J.-F. Remacle, N. Chevaugeon, E. Marchandise and C. Geuzaine. *Efficient visualization of high-order finite elements.* International Journal for Numerical Methods in Engineering, vol. 69, nb. 5, pages 750–771, 2007. Cited on pages 9, 18, 88, and 199.

[Remacle et al. 2015] J.-F. Remacle, V. Bertrand and C. Geuzaine. *A Two-Level Multithreaded Delaunay Kernel.* Procedia Engineering, vol. 124, pages 6 – 17, 2015. 24th International Meshing Roundtable. Cited on pages 9 and 17.

[Roca et al. 2012] X. Roca, A. Gargallo-Peiró and J. Sarrate. *Defining Quality Measures for High-Order Planar Triangles and Curved Mesh Generation.* In W. R. Quadros, Editor, Proceedings of the 20th International Meshing Roundtable, pages 365–383. Springer Berlin Heidelberg, 2012. Cited on pages 135 and 137.

[Ruiz-Gironès et al. 2015] E. Ruiz-Gironès, J. Sarrate and X. Roca. *Defining an L^2-disparity Measure to Check and Improve the Geometric Accuracy of Non-interpolating Curved High-order Meshes.* Procedia Engineering, vol. 124, pages 122–134, 2015. Cited on pages 88 and 152.

[Ruiz-Gironès et al. 2016a] E. Ruiz-Gironès, X. Roca and J. Sarrate. *High-order mesh curving by distortion minimization with boundary nodes free to slide on a 3D CAD representation.* Computer-Aided Design, vol. 72, pages 52 – 64, 2016. 23rd International Meshing Roundtable Special Issue: Advances in Mesh Generation. Cited on pages 88, 135, 152, and 175.

[Ruiz-Gironès et al. 2016b] E. Ruiz-Gironès, J. Sarrate and X. Roca. *Generation of curved high-order meshes with optimal quality and geometric accuracy.* Procedia engineering, vol. 163, pages 315–327, 2016. Cited on pages 88 and 152.

[Schneiders and Bünten 1995] R. Schneiders and R. Bünten. *Automatic generation of hexahedral finite element meshes.* Computer Aided Geometric Design, vol. 12, nb. 7, pages 693–707, 1995. Cited on pages 8 and 16.

[Schroeder et al. 2006] W. J. Schroeder, F. Bertel, M. Malaterre, D. Thompson, P. P. Pebay, R. O'Bara and S. Tendulkar. *Methods and framework for visualizing higher-order finite elements.* IEEE Transactions on Visualization and Computer Graphics, vol. 12, nb. 4, pages 446–460, 2006. Cited on pages 9, 18, and 88.

[Sevilla et al. 2008] R. Sevilla, S. Fernández-Méndez and A. Huerta. *NURBS-enhanced finite element method (NEFEM).* International Journal for Numerical Methods in Engineering, vol. 76, nb. 1, pages 56–83, 2008. Cited on page 87.

[Shephard and Georges 1991] M. S. Shephard and M. K. Georges. *Automatic three-dimensional mesh generation by the finite octree technique.* International Journal for Numerical methods in engineering, vol. 32, nb. 4, pages 709–749, 1991. Cited on pages 8 and 16.

[Sherwin and Peiró 2002] S. J. Sherwin and J. Peiró. *Mesh generation in curvilinear domains using high-order elements.* International Journal for Numerical Methods in Engineering, vol. 53, nb. 1, pages 207–223, 2002. Cited on pages 9, 18, 88, 135, and 175.

[Tam and Armstrong 1991] T. K. H. Tam and C. G Armstrong. *2D finite element mesh generation by medial axis subdivision.* Advances in engineering software and workstations, vol. 13, nb. 5-6, pages 313–324, 1991. Cited on pages 8 and 16.

[Taubin 1994] G. Taubin. *Distance approximations for rasterizing implicit curves.* In ACM Trans. Graph., Volume 13, pages 3–42, 1994. Cited on page 213.

[TecPlot Inc.] TecPlot Inc. *TecPlot.* `https://www.tecplot.com/`. Cited on page 199.

[Toulorge et al. 2013] T. Toulorge, C. Geuzaine, J.-F. Remacle and J. Lambrechts. *Robust untangling of curvilinear meshes.* Journal of Computational Physics, vol. 254, pages 8 – 26, 2013. Cited on pages 88, 91, 135, 171, 172, and 175.

[Toulorge et al. 2016] T. Toulorge, J. Lambrechts and J.-F. Remacle. *Optimizing the geometrical accuracy of curvilinear meshes.* Journal of Computational Physics, vol. 310, pages 361 – 380, 2016. Cited on pages 88, 152, and 171.

[Tristano et al. 1998] J. R. Tristano, S. J. Owen and S. A. Canann. *Advancing front surface mesh generation in parametric space using a riemannian surface definition.* In Proceedings of the 7th International Meshing Roundtable, 1998. Cited on page 152.

[Turner et al. 2016] M. Turner, J. Peirò and D. Moxey. *A Variational Framework for High-order Mesh Generation.* Procedia Engineering, vol. 163, nb. Supplement C, pages 340 – 352, 2016. 25th International Meshing Roundtable. Cited on pages 88, 135, and 175.

[Turner 2018] M. Turner. *High-order mesh generation for CFD solvers.* Ph.D. Thesis, Imperial College London, February 2018. Cited on page 152.

[Vallet 1992] M.-G. Vallet. *Génération de maillages éléments finis anisotropes et adaptatifs.* Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, 1992. Cited on page 34.

[Van Leer 1979] B. Van Leer. *Towards the ultimate conservative difference scheme. A second-order sequel to Godunov's method.* Journal of Computational Physics, pages 101–136, 1979. Cited on pages 7, 15, 38, and 62.

[Vanella et al. 2010] M. Vanella, P. Rabenold and E. Balaras. *A direct-forcing embedded-boundary method with adaptive mesh refinement for fluid-structure interaction problems.* Journal of Computational Physics, vol. 229, nb. 18, pages 6427 – 6449, 2010. Cited on page 34.

[Vanharen et al. 2017] J. Vanharen, G. Puigt, X. Vasseur, J.-F. Boussuge and P. Sagaut. *Revisiting the spectral analysis for high-order spectral discontinuous methods.* Journal of Computational Physics, vol. 337, pages 379 – 402, 2017. Cited on page 7.

[Vanharen et al. 2018] J. Vanharen, R. Feuillet and F. Alauzet. *Mesh adaptation for fluid-structure interaction problems.* In 24th AIAA Fluid Dynamics Conference, AIAA Paper 2018-3244, Atlanta, GA, USA, Jun 2018. Cited on pages 11, 19, 34, and 239.

[Vanharen 2017] J. Vanharen. *High-order numerical methods for unsteady flows around complex geometries.* Ph.D. Thesis, Université de Toulouse, 2017. Cited on pages 15 and 87.

[Viquerat 2019] J. Viquerat. *Efficient time-domain numerical analysis of waveguides with tailored wideband pulses.* Microwave and Optical Technology Letters, vol. 61, nb. 6, pages 1534–1539, 2019. Cited on page 222.

[Vlachos et al. 2001] A. Vlachos, P. Jörg, C. Boyd and J. L. Mitchell. *Curved PN Triangles.* In Proceedings of the 2001 Symposium on Interactive 3D Graphics, pages 159–166, 2001. Cited on pages 88, 176, and 199.

[Wang et al. 2006] D. Wang, O. Hassan, K. Morgan and N. Weatherill. *EQSM: An efficient high quality surface grid generation method based on remeshing.* Computer Methods in Applied Mechanics and Engineering, vol. 195, nb. 41-43, pages 5621–5633, 2006. Cited on page 152.

[Wang et al. 2011] K. Wang, A. Rallu, J.-F. Gerbeau and C. Farhat. *Algorithms for interface treatment and load computation in embedded boundary methods for fluid and fluid-structure interaction problems.* International Journal for Numerical Methods in Fluids, vol. 67, pages 1175–1206, 2011. Cited on pages 33, 51, and 57.

[Warburton 2006] T. Warburton. *An explicit construction of interpolation nodes on the simplex*. Journal of Engineering Mathematics, vol. 56, nb. 3, pages 247–262, Nov 2006. Cited on page 120.

[Watson 1981] D. F. Watson. *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*. The Computer Journal, vol. 24, nb. 2, pages 167–172, 01 1981. Cited on page 232.

[Weatherill and Hassan 1994] N. P. Weatherill and O. Hassan. *Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints*. International Journal for Numerical Methods in Engineering, vol. 37, nb. 12, pages 2005–2039, 1994. Cited on pages 8 and 16.

[Xie et al. 2013] Z. Q. Xie, R. Sevilla, O. Hassan and K. Morgan. *The generation of arbitrary order curved meshes for 3D finite element analysis*. Computational Mechanics, vol. 51, nb. 3, pages 361–374, Mar 2013. Cited on pages 9, 18, 88, and 175.

[Xu et al. 2017] L. Xu, X. Ren, X. Xu, H. Li, Y. Tang and Y. Feng. *An adaptive visualization tool for high order discontinuous galerkin method with quadratic elements*. In 2017 IEEE International Conference on Computer and Information Technology (CIT), pages 176–183. IEEE, 2017. Cited on pages 9 and 18.

[Yang et al. 1997a] G. Yang, D. M. Causon, D. M. Ingram, R. Saunders and P. Batten. *A Cartesian cut cell method for compressible flows part A: Static body problems*. Aeronautical Journal, vol. 101, nb. 1002, pages 47–56, 1997. Cited on pages 9, 17, 33, 57, and 83.

[Yang et al. 1997b] G. Yang, D. M. Causon, D. M. Ingram, R. Saunders and P. Batten. *A cartesian cut cell method for compressible flows Part B: moving body problems*. Aeronautical Journal, vol. 101, nb. 1002, pages 57–65, 1997. Cited on pages 33, 57, and 83.

[Yerry and Shephard 1984] M. A. Yerry and M. S. Shephard. *Automatic three-dimensional mesh generation by the modified-octree technique*. International Journal for Numerical Methods in Engineering, vol. 20, nb. 11, pages 1965–1990, 1984. Cited on pages 8 and 16.

[Zhang et al. 2019] R. Zhang, A. Johnen and J.-F. Remacle. *Curvilinear Mesh Adaptation*. In 27th International Meshing Roundtable, pages 57–69, Cham, 2019. Springer International Publishing. Cited on pages 136 and 229.

[Zienkiewicz and Zhu 1992a] O. C. Zienkiewicz and J. Z. Zhu. *The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique*. International Journal for Numerical Methods in Engineering, vol. 33, nb. 7, pages 1331–1364, 1992. Cited on pages 49 and 50.

[Zienkiewicz and Zhu 1992b] O. C. Zienkiewicz and J. Z. Zhu. *The superconvergent patch recovery and a posteriori error estimates. Part 2: Error estimates and adaptivity*. International Journal for Numerical Methods in Engineering, vol. 33, nb. 7, pages 1365–1382, 1992. Cited on page 50.

[Zwanenburg and Nadarajah 2017] P. Zwanenburg and S. Nadarajah. *On the Necessity of Superparametric Geometry Representation for Discontinuous Galerkin Methods on Domains with Curved Boundaries*. In 23rd AIAA Computational Fluid Dynamics Conference, 2017. Cited on page 87.

université PARIS-SACLAY | École doctorale de mathématiques Hadamard (EDMH)

Titre : Maillages immergés et d'ordre élevé: deux alternatives à la représentation linéaire des maillages en géométrie inscrite.

Mots Clefs : Méthodes d'ordre élevé, Visualisation scientifique, Méthodes immergées, Mécanique des fluides numérique, Adaptation de maillage.

Résumé : La simulation numérique de phénomènes physiques complexes requiert généralement l'utilisation d'un maillage. En mécanique des fluides numérique, cela consiste à représenter un objet dans un gros volume de contrôle. Cet objet étant celui dont l'on souhaite simuler le comportement. Usuellement, l'objet et la boîte englobante sont représentés par des maillages de surface linéaires et la zone intermédiaire est remplie par un maillage volumique. L'objectif de cette thèse est de s'intéresser à deux manières différentes de représenter cet objet. La première approche - dite immergée - consiste à mailler intégralement le volume de contrôle et ensuite à simuler le comportement autour de l'objet sans avoir à mailler explicitement dans le volume ladite géométrie. L'objet étant implicitement pris en compte par le schéma numérique. Le couplage de cette méthode avec de l'adaptation de maillage linéaire est notamment étudié. La deuxième approche - dite d'ordre élevé - consiste quant à elle à augmenter le degré polynomial du maillage de surface de l'objet. La première étape consiste donc à générer le maillage de surface de degré élevé et ensuite à propager l'information de degré élevé dans les éléments volumiques environnants si nécessaire. Dans ce cadre-là, il s'agit de s'assurer de la validité de telles modifications et de considérer l'extension des méthodes classiques de modification de maillages linéaires. Il convient également de développer de nouvelles méthodes de visualisation de maillages et de solutions, les méthodes classiques étant mises en défaut pour ce nouveau type d'entités.

Title : Embedded and high-order meshes: two alternatives to linear body-fitted meshes.

Keywords : High-order methods, Scientific visualization, Embedded methods, CFD, Mesh adaptation.

Abstract : The numerical simulation of complex physical phenomena usually requires a mesh. In Computational Fluid Dynamics, it consists in representing an object inside a huge control volume. This object is then the subject of some physical study. In general, this object and its bounding box are represented by linear surface meshes and the intermediary zone is filled by a volume mesh. The aim of this thesis is to have a look at two different approaches for representing the object. The first approach, called embedded method, consists in fully meshing the bounding box volume without explicitly meshing the object in it. In this case, the presence of the object is implicitly simulated by the CFD solver. The coupling of this method with linear mesh adaptation is in particular discussed. The second approach, called high-order method, consists on the contrary in increasing the polynomial order of the surface mesh of the object. The first step is therefore to generate a suitable high-order surface mesh and then to propagate the high-order information in the neighboring volume if necessary. In this context, it is mandatory to make sure that such modifications are valid and to consider the extension of classic linear mesh modification techniques. Also, a special care has to be done on the development of new mesh and solution visualization methods as the classic ones fail to properly render this new type of entities.