Université Paris-Saclay

École doctorale de mathématiques Hadamard (ED 574)
Génération automatique de maillages et méthodes d'adaptation (INRIA)

Mémoire présenté pour l'obtention du

Diplôme d'habilitation à diriger les recherches

Discipline : Mathématiques

par

Adrien Loseille

Génération et Adaptation de Maillages pour le Calcul Scientifique

Rapporteurs :	Prénom NOM
	Prénom NOM
	Prénom NOM

Date de soutenance : Jour Mois Année

Composition du jury :	Prénom NOM	(Rapporteur)
	Prénom NOM	(Rapportrice)
	Prénom NOM	(Présidente)
	Prénom NOM	(Examinateur)
	Prénom NOM	(Examinatrice)
	Prénom NOM	(Examinateur)
	Prénom NOM	(Invité)

Résumé

Cette introduction en français présente de manière très succincte quelques problématiques de recherche qui sont présentées plus en détail dans la suite. Pour l'ensemble du manuscript, les citations ayant une numérotation numérique, comme [1] sont relatives à mes publications, données dans l'introduction, les autres concernent la bibliographie générale contenue à la fin du manuscript.

Dans le cadre de la simulation numérique, l'adaptation de maillage est une stratégie visant à modifier automatiquement la discrétisation du domaine de calcul pour contrôler la précision de la solution numérique d'un système d'équations aux dérivées partielles (EDP) [FG08]. Son efficacité repose sur un couplage complexe entre l'estimateur d'erreur, les algorithmes de génération de maillages adaptés et le solveur numérique. Réussir ce couplage permet alors de diminuer le temps de calcul, de réduire le nombre de degrés de liberté (gain mémoire) et d'améliorer la précision de la solution numérique. Ces gains sont d'autant plus importants que les phénomènes physiques en jeu sont anisotropes. Cependant, la prise en compte de l'anisotropie fragilise et complexifie le processus adaptatif limitant considérablement les gains réels par rapport aux gains théoriques attendus. Les thématiques et problématiques de recherche présentées dans ce manuscript concernent l'amélioration des performances du processus adaptatif à la fois sur les points théoriques et algorithmiques. Les principaux axes de recherche concernent:

▷ **L'élaboration d'un cadre théorique et pratique pour l'adapation de maillages basés sur la dualité entre maillages discrets et espaces Riamanniens.**

▷ **Le développement d'algorithmes de maillage générique, parallèle et robuste permettant de générer plusieurs milliards de tétraèdres en quelques minutes pour des maillages anisotropes et en géométries complexes.**

▷ **Le développement de nouvelles technologies de maillage adaptif comme les approches metric-aligned ou metric-orthogonal permettant d'améliorer drastiquement la qualité des maillages anisotropes, pour une meilleure précision tout en minimisant le coût de calcul.**

Dans les mailleurs adaptatifs, les espaces Riemanniens servent à définir le calcul des distances [19]. Cependant, ces espaces vont bien au delà d'une simple définition de distances. Ils constituent également un outil théorique puissant pour l'estimation d'erreur car ils permettent de s'affranchir des contraintes liées aux maillages discrets: forme géométrique des éléments, positions des points ... Le fondement théorique repose sur des égalités mathématiques liant l'erreur d'interpolation discrète à l'erreur d'interpolation continue [11, 12]. On montre alors qu'à un unique maillage continu optimal correspondait un ensemble infini de maillages discrets. Chaque représentant discret est simplement lié au choix de la méthode pratique utilisée pour générer le maillage. Ce concept allie donc à la fois un outil mathématique simplifiant l'analyse d'erreur pour décrire les maillages optimaux mais aussi une méthode algorithmique permettant de générer un maillage discret approprié. Cette double caractéristique discret-continu est complètement

originale. Elle se différencie des approches classiques d'estimation [Verfürth 1996, Cao 2005] qui négligent souvent l'aspect algorithmique pourtant nécessaire à la génération effective du maillage optimal. L'adaptation de maillage multi-échelles [58] est issue du concept de maillage continu et propose un estimateur d'erreur anisotrope basé sur l'erreur d'interpolation. Elle permet de lever un ensemble de verrous théoriques et pratiques liés aux estimateurs classiques [Peraire 1987, Castro-Díaz 1997]. D'une part, elle offre l'avantage de raffiner l'ensemble des échelles de la solution permettant ainsi de générer des maillages très anisotropes. En 3D, l'anisotropie est généralement bornée à des élancements de 1 pour 10 pour les approches classiques [Frey 2005, Remacle 2005] alors que l'on dépasse 1 pour 1000 avec l'approche multi-échelles. D'autre part, elle offre des garanties sur la consistance du schéma numérique quel que soit la régularité de l'écoulement. La prise en compte dans l'analyse d'erreur des performances des schémas numériques est une innovation en adaptation anisotrope. Cela contribue à l'excellente synergie entre le schéma numérique et la physique, ce qui permet d'obtenir une précision qui est inaccessible sur des maillages uniformes ou adaptés d'une manière isotrope. Les espaces Riemanniens et la dualité avec les maillages discrets sont utilisés dans tous les chapitres du manuscrit. En particulier, dans le Chapitre 4, l'extension de cette dualité pour des approximations d'ordre élevées est présentée, démontrant d'autant plus l'intêret théorique et pratique de ces espaces pour l'adaptation de maillage.

Sur le plan algorithmique, un nouveau module complet de génération de maillage fortement anisotrope et robuste est implémenté depuis 2010. Le principal enjeu est de parvenir à coupler l'anisotropie et la robustesse qui ont des comportements antagonistes. Une première stratégie simple d'adaptation consiste à vérifier la précision des nouveaux éléments créés explicitement par des fonctions qualités spécifiques chaque étape modifiant le maillage. L'algorithme de remaillage, pour le volume, s'appuie sur l'application successive d'opérateurs simples (insertion d'un point sur une arête ou suppression d'une arête) qui modifient le maillage de manière locale. Le choix d'effectuer une opération particulière repose sur un critère global de qualité. On utilise l'algorithme de recuit simulé [Kirkpatrick 1983] à l'on accepte initialement de dégrader la qualité du maillage de manière aléatoire pour arriver à une méthode d'optimisation stricte à la fin du processus. Sous cette forme, la stratégie d'adaptation reposant sur le nouveau module de génération de maillages adaptés permet d'obtenir un niveau d'anisotropie dans le volume jamais atteint auparavant [54] dans le cas d'écoulement non turbulent et ce, sans compromis sur la qualité du maillage nécessaire pour garantir la stabilité du solveur numérique. Cette stratégie est décrite dans la Chapitre 1. Une approche plus générique et performante, *via* l'opérateur de cavité, est ensuite introduite au Chapitre 2. Cet opérateur généralise tous les opérateurs de modification de maillages. Il permet de générer des maillages hybrides (de couches limites) ou de gérer de manière robuste une forte anisotropie près des géométries complexes pour, par exemple, capturer les fortes interactions entre chocs et couches limites dans des écoulements turbulents.

Une des problématiques majeures pour utiliser (et transférer) l'adaptation de maillages aux problèmes industriels est liée au parallélisme. En effet, beaucoup de solveurs utilisent de manière efficace les architectures HPC (algèbre linéaire, partitionnement de domaine, . . .) alors que les logiciels de maillage restent principalement séquentiels. Les raisons de ce décalage sont multiples. Le premier est lié à la multitude d'opérateurs qu'il est difficile de paralléliser : insertion de points, suppression d'arêtes, bascule d'arêtes ou de faces. De plus, les structures de données ne sont pas

statiques mais dynamiques car elles évoluent au cours du temps. Enfin, les partitionneurs de domaines visent à minimiser les communications entre les processus, ce qui n'est pas nécessaire en maillage. Pour contrer ces effets, , Il est nécessaire de proposer un partitionneur de domaine hiérarchique capable de prédire les charges de calcul spécifiques aux mailleurs. L'utilisation d'un unique opérateur de cavité permet de se libérer de la la contrainte de devoir paralléliser tous les composants. Avec cette stratégie, il est possible de générer plusieurs milliards de tétraèdres en 20 minutes sur des clusters de calcul standards, composés d'une centaine de cœurs. Cette extension de l'opérateur de cavité est présenté au Chapitre 3.

Les algorithmes de maillages anisotropes doivent contrôler des grandeurs antinomiques. Il faut à la fois garantir une forte anisotropie, car c'est elle qui donne le gain en terme de temps de calcul et de précision, mais aussi la qualité du maillage. C'est cette dernière qui permet la stabilité du solveur utilisé. Jusqu'à présent, la qualité des maillages anisotropes rest très mauvaise comparativement aux méthodes traditionnelles en maillage uniforme (frontales ou Delaunay). Cela constitue un frein à la diffusion de l'adaptation. Pour améliorer drastiquement cette qualité, il est possible d'étendre les approches classiques en ajoutant des contraintes directionnelles. Ces méthodes, dite metric-aligned ou metric-orthogonal, suivents alors des directions privilégiées pour proposer des points positionnés de manière optimale dans le domaine. Ces directions proviennent soit de la métrique (définissant l'orientation souhaitée et l'anisotropie), soit des directions de courbure de la surface. Avec cette approche, la qualité des maillages, notamment des angles, est drastiquement améliorée. On atteint des qualités équivalentes aux méthodes traditionnelles. Les impacts sont multiples. Les solveurs convergent plus vite et le nombre d'itérations pour converger le problème global est réduit. Cette approche est décrite dans le Chapitre 3.

A brief introduction

In this manuscript, I synthesize my research as a full-time research scientist at INRIA since 2011. My research has been focusing on the core components of anisotropic mesh adaptation employed to obtain high-fidelity numerical predictions of complex nonlinear PDEs. This includes the development of adaptive mesh generation algorithms and the derivation of anisotropic error estimates. However, these two fields of research are not self-sufficient. They always require a numerical prediction provided by an external *solver*. Depending on the field of applications, many collaborations have been built to validate and challenge my research. So, I would like to thank from the very beginning all my research partners. Without their inputs, this research would not have been possible. In this thesis, most of the examples rely on `Wolf` (INRIA) solver by F. Alauzet. This includes examples from compressible flows mechanics: sonic-boom prediction, drag and high-lift studies. `Feflo` (GMU) by R. Löhner and `SU2` (Stanford) solvers have been used for similar applications. `Cedre` (Onera) with W. Ghedhaifi and E. Montreuille, has been used for the prediction of contrail formation. `Ananas` (Lemma Ing.) is used for incompressible bi-fluid dynamics with application to the petroleum and spacial industry. `Coffee` (CNRS) by S. Chaillat is used for acoustic waves propagation using the Boundary Element Method (BEM).

I would like to also thank my co-workers from the Unstructured Grid Adaptation Working Group, mainly M. Park (NASA) and T. Michal (Boeing), my mentors : P. L. George (INRIA) and R. Löhner (GMU) for mesh generation and A. Dervieux (INRIA) for error estimations, and of course, all the students and post-docs that have been always highly motivated and dedicated to this research.

To enlighten the collaborations, each chapter introduction is followed by the list of associated publications along with the list of collaborators and students involved in the work.

Scientific context and challenges

The rapid advance of both computer hardware and physical simulation capabilities has revolutionized science and engineering, placing computational simulation on an equal footing with theoretical analysis and physical experimentation. When very approximated numerical models were used in the 80s with 3D meshes composed of less than 10 000 elements, nowadays, fully detailed complex models are used in the analysis, and the size of the computations are routinely of the order of $10^7 - 10^9$ degrees of freedom. This reliance on the predictive capabilities has created the need for an accurate control of numerical errors (having a strong impact on these predictions). In this context, mesh adaptation has a prominent role since the quality of the mesh, its refinement and its alignment with the physics, has major contributions to the numerical errors. It is now widely admitted that a single computation is not enough to assess a numerical prediction [Slotnick 2014], therefore multiplying the computations is required as in Uncertainty Quantification or in a mesh adaptation process. This puts an even stronger constraint, in terms of reliability or CPU time efficiency, on the required qualities of an adaptive mesh generation process. If the mesh generation capabilities of commercial packages [Csimsoft , Distene , PointWise] have made significant progresses for the generation of the

first mesh, including more robustness in CAD (Geometry) imports, CAD cleaning, faster or more reliable surface and 3D mesh generations, there is still little interest both in the scientific community and from commercial providers to offer anisotropic adaptive capabilities.

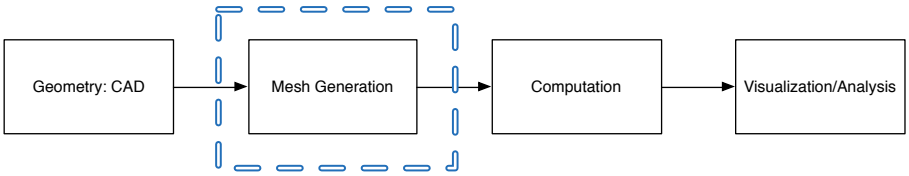


Figure 1 – *Computational pipeline*

The mesh generation process is one of the crucial components of the computational pipeline (Fig. 1). The quality of the mesh will directly impact the quality of the numerically predicted quantities of interest. The properties of the mesh, alignment, density and orientation, will impact the stability and accuracy of the computation. In addition, it is now widely acknowledged that adaptive simulations [38] are required to gain confidence in numerical results. For instance, adaptive meshing has been seen as one the major bottleneck for future CFD analysis in the 2030 NASA report on the future of Computational Fluid Dynamics (CFD) [Slotnick 2014] and in the dedicated report [38] focusing exclusively on mesh adaptation. Naive mesh refinement techniques (i.e. where the mesh size is divided by two at each step) cannot be used in practice as the number of degrees of freedom is multiplied by 8 at each step, leading to very large mesh very rapidly. It is then necessary to consider more elaborate techniques, such as automatic mesh adaptation to assess numerical simulations. Standard meshing techniques, e.g. the advancing front techniques [Löhner 1988a, Marcum 2013] or the constrained Delaunay approach [Borouchaki 2000a, George 2003a], are now very mature to generate an unstructured mesh even in the presence of very complex geometries. However their extension to generate adaptive meshes is difficult, if not almost impossible for the advancing front method. In addition, the core components of these legacy approaches have not bring too much attention from the meshing community from the past 10 years. In the meantime, new classes of numerical schemes simulating more complex physics have emerged, putting more constrains on the mesh generation process. Hybrid elements or high-order curved elements are now a classical requirement. The generation of each kind of mesh has been considered so far as a standalone problem [Aubry 2009, Marcum 2013, Remacle 2012, Ruiz-Gironès 2016a] without adaptivity in mind.

The core motivation of my research is then to explore a mathematically sounded mesh generation framework complying with all kinds of meshes at once and having in mind the absolute necessity of an adaptive mesh strategy. Additional mesh requirements are imposed by the diversity of the current computing platforms, with different levels of parallelism. Modern numerical solvers are now using advantageously emerging architectures, e.g. GPU or Xeon Phi. If no specific attention is devoted to mesh generation or adaptation, it may become again in the next years, the bottleneck of realistic computations (*i.e.*, to really push the level of fidelity in the computation). In practice, this may prevent the scientific community to explore highly accurate and reliable solutions and have more insights in complex physical phenomena such as turbulence in the context of CFD. The meshing framework has to offer competitive CPU timings compared

to state-of-the-art parallelized solvers. This is especially true in the adaptive context where sequences of meshes and solutions are generated and computed iteratively.

From a meshing point-of-view, my main contributions are based on the development of a unique cavity-based framework. Indeed, this framework extends the classical Delaunay-based insertion. A constrained version allows to generate hybrid meshes as classically required to generate boundary layer meshes. This operator is described in Chapter 2. Its parallel extension and quality enhancements, with so called metric-aligned and metric-orthogonal approaches are summarized in Chapter 3.

From an error estimate point of view, my main contributions concern the development of the continuous mesh framework. This framework allows to recast standard error estimates (interpolation based) in a continuous setting. These errors estimates are anisotropic by nature. Classical error estimates based on this framework are reviewed in Chapter 1. Extensions to control numerical solution with high-order approximations are reviewed in Chapter 4.

The manuscript is organized as follows:

Chapter 1. This chapter provides a quick review of the standard meshing pipeline and emphasizes all the mathematical and algorithmic background to implement a basic but robust 3D anisotropic local remesher. All the developments are based on Riemannian metric fields. They are manipulated at a continuous level to derive error estimates while discrete operators are used during the mesh generation phase. I also review past contributions related to the derivation of error estimates to control interpolation or approximation error on PDEs. Several examples, mostly from Computational Fluids Dynamics (CFD), are commented. In particular, I illustrate that these error estimates allow to recover an optimal order of convergence of the underlying numerical scheme, even if discontinuities are present in the flow field.

Chapter 2. This chapter details the implementation of the "unique cavity-based" framework that extends traditional Delaunay-based mesh generation methods. We show that this approach allows to encompass and generalize all traditional local remeshing operators. It allows the generation of boundary layer mesh as well by using a constrained version of the operator. We illustrate that a high-level of flexibility and reliability is obtained. In comparison with previous examples, much more complex physics and geometries are involved. All simulations are solutions of the Reynolds-Averaged Navier-Stokes (RANS) equations where there exists a strong interaction between the boundary layer and the outer field. In particular, a higher level of anisotropy ($O(1 - 100000)$) is required to capture these interactions. In addition, the geometry is much more complex and the fidelity of the surface mesh evolving during the refinement is capital to ensure the convergence of quantities of interest like lift or drag.

Chapter 3. The chapter reviews two enhancements to the cavity-based framework. This includes two drastic improvements for the quality of anisotropic mesh generation with the metric-aligned and metric-orthogonal approaches. These approaches use additional information in the Riemannian metric field (as orientation and natural alignment) to generate high-quality surface and volume meshes. The parallel implementation of the operator is also reviewed. The use of a cavity-operator yields on a fully predictable process in terms of CPU with respect to the input mesh and desired metric. Examples of anisotropic meshes composed with more than a billion tetrahedra are generated on small size cluster (120 processors) within 15 minutes.

Chapter 4. This chapter reviews some contributions related to mesh adaptation when high-order numerical schemes (Discontinuous Galerkin, spectral elements) are used. In this case, all the error estimates of Chapter 1 are no more suited to control the anisotropy of the solution. The local error is then represented by high-order homogenous polynomials. The strong nonlinearity makes the finding of anisotropy difficult. We present the log-simplex approach that allows to bound the variation these high-order polynomials by quadratic functions to a given power. Additional challenges like high-order pixel-exact rendering or high-order meshing techniques for CAD geometry are also discussed.

Chapter 5. The last chapter is a short general discussion on the perspectives for adaptive mesh generation to fit upcoming computational architectures and numerical schemes to address the future challenges described in [Slotnick 2014].

Supervising activities

During this period, I have co-advised 6 PhD students, 3 Postdocs and 4 Research engineers.

Supervised PhD students

2010-2013 E. Mbinky, PhD co-supervised at 50% with F. Alauzet, on *Anisotropic mesh adaptation for very high order numerical schemes*. Publication [51] is related to this work.

2012-2015 V. Menier, PhD co-supervised at 50% with F. Alauzet, on *Numerical Methods and Mesh Adaptation for Reliable RANS Simulations*. Publications [9, 44, 48] are related to this work.

2015-2018 L. Frazza, PhD co-supervised at 50% with F. Alauzet, on *3D anisotropic mesh adaptation for Reynolds Averaged Navier-Stokes simulations*. Publications [3, 5, 28, 29] are related to this work.

2016-2019 R. Feuillet, PhD co-supervised at with F. Alauzet and P. Ciarlet, on *Embedded and high-order meshes: two alternatives to linear body-fitted meshes*. Publications [2, 26, 32, 33] are related to this work.

2018-2021 L.-M. Tenkès, PhD co-supervised at 50% with F. Alauzet, on *Methods for generating quad-dominant and hex-dominant adaptive meshes for CFD applications*.

2019-2022 L. Rochery, PhD on *Methods for generating adaptive high-order curved meshes*.

Supervised post-doctoral students

2015-2017 O. Coulaud, Postdoc supervised at 100%, on *Anisotropic mesh adaptation for very-high order interpolates in 3D*. Publications are related to this work [26, 37].

2015-2016 S. Groth, Postdoc supervised at 50% with S. Chaillat (INRIA Poems), on *Anisotropic mesh adaptation for the Boundary Element Methods*. Publication [6] is related to this work.

2016-2017 F. Amlani, Postdoc supervised at 50% with S. Chaillat (INRIA Poems), on *Anisotropic mesh adaptation for the Boundary Element Methods*. Publication [4] is related to this work.

Supervised research engineers

2011-2013 J. Castelneau, supervised at 100%, on *Mesh visualization and modification on ViZiR software.*

2014-2015 A. Loyer, supervised at 50% with H. Guillard (INRIA Castor), on *Meshing framework for Tokamac simulations.*

2015-2016 Q. Arnoldus, supervised at 50% with D. Marcum (MSU), on *Mesh visualization and modification software for modern architectures and high-order solution visualization.*

2019- M. Maunoury, supervised at 100%, on *Mesh visualization high-order solution visualization* for ViZiR4 project.

These engineers have participated on the developments of the Adaptive Mesh Generation suite, see pyamg.saclay.inria.fr.

Publications

[1] R. Feuillet and A. Loseille, On pixel-exact rendering for high-order mesh and solution, *Submitted*, 2020.

[2] R. Feuillet, A. Loseille and F. Alauzet, Optimization of meshes and applications, *Computer-Aided Design*, Vol 124 pp. 1-20, 2020. [pdf]

[3] L Frazza, A Loseille, A Dervieux, F Alauzet, Nonlinear corrector for Reynolds-averaged Navier-Stokes equations, *International Journal for Numerical Methods in Fluids*, Vol 91(11), pp. 557-585, 2019. [pdf]

[4] F. Amlani, S. Chaillat and A. Loseille, An efficient preconditioner for adaptive Fast Multipole accelerated Boundary Element Methods to model time-harmonic 3D wave propagation, *Computer Methods in Applied Mechanics and Engineering*, Vol 352, pp. 189-210, 2019. [pdf]

[5] A. Loseille, L. Frazza and F. Alauzet, Comparing anisotropic adaptive strategies on the 2nd AIAA sonic boom workshop geometry, *Journal of Aircraft.* Vol 56(3), 2018, [pdf]

[6] S. Chaillat, S. P. Groth and A. Loseille, Metric-based anisotropic mesh adaptation for 3D acoustic boundary element methods, *Journal of Computational Physics*, Vol 372, pp. 473-499, 2018. [pdf]

[7] F. Alauzet, A. Loseille and G. Olivier, Time-Accurate multi-scale anisotropic mesh adaptation for unsteady flows in CFD, *Journal of Computational Physics*, Vol 373, pp. 28-63, 2018. [pdf]

[8] A. Carabias, A. Belme, A. Loseille and A. Dervieux, Anisotropic Goal-oriented error analysis for a third order accurate CENO Euler discretization, *International Journal for Numerical Methods in Fluids*, Vol 86, Issue 6, pp. 392-413, 2018. [pdf]

[9] 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, pp. 53-67, 2017. [pdf].

[10] F. Alauzet and A. Loseille, A decade of progress on anisotropic mesh adaptation for computational fluid dynamics, *Computer-Aided Design*, Vol 72, pp. 13-39, 2016. [pdf]

[11] A. Loseille and F. Alauzet, Continuous mesh framework, Part I: well-posed continuous interpolation error, *SIAM Journal on Numerical Analysis*, Vol 49, pp. 38-60, 2011. [pdf].

[12] A. Loseille and F. Alauzet, Continuous mesh framework, Part II: validations and applications, *SIAM Journal on Numerical Analysis*, Vol 49, pp. 61-86, 2011. [pdf].

[13] A. Loseille, A Dervieux and F. Alauzet, Fully anisotropic goal-oriented mesh adaptation for 3D steady Euler equations, *Journal of Computational Physics*, Vol. 229, Issue 8, pp. 2866-2897, 2010. [pdf].

[14] F. Alauzet and A. Loseille, High order sonic boom modeling based on adaptive techniques, *Journal of Computational Physics*, Vol. 229, Issue 3, pp. 561-593, 2010. [pdf].

[15] Y. Bourgault, M. Picasso, F. Alauzet and A. Loseille, On the use of anisotropic a posteriori error estimators for the adaptive solution of 3D inviscid compressible flows, *International Journal for Numerical Methods in Fluids*, Vol. 59, Issue 1, pp. 47-74, 2009. [pdf].

[16] F. Alauzet, S. Borel-Sandou, L. Daumas, A. Dervieux, Q. Dinh, S. Kleinveld, A. Loseille, Y. Mesri and G. Rog, Multi-model and multi-scale optimization strategies. Application to sonic boom reduction, *European Journal of Computational Mechanics*, Vol. 17, Issue 1-2, pp. 191-214, 2008. [pdf].

[17] A. Dervieux, Y. Mesri, F. Alauzet, A. Loseille, L. Hascoet and B. Koobus, Continuous mesh adaptation models for CFD, *CFD Journal*, Vol. 16, Issue 4, pp. 346-355, 2008. [pdf].

Books and book chapters

[18] F. Alauzet, A. Dervieux, L. Frazza and A. Loseille, Numerical uncertainties estimation and mitigation by mesh adaption, *Uncertainty Management for Robust Industrial Design in Aeronautics. Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, Edited by C. Hirsch, D. Wunsch, J. Szumbarski, L. Łaniewski-Wołłk, J. Pons-Prats J, Springer, Vol. 140, pp. 89-107, 2019.

[19] P.L. George, H. Borouchaki, F. Alauzet, P. Laug, A. Loseille and L. Marchal, Maillage, modélisation et simulation numérique, Volume 2, *ISTE ditions*, 2018.

[20] P.L. George, H. Borouchaki, F. Alauzet, P. Laug, A. Loseille and L. Marchal, Meshing, Geometric Modeling and Numerical Simulation, Volume 2, *ISTE ditions and Wiley*, 2019.

[21] A. Loseille, Mesh generation and adaptation, *Handbook on Numerical Methods for Hyperbolic Problems: Applied and Modern Issues*, Chapter 12, Edited by R. Abgrall and C.-W. Shu, Elsevier, 2017 [link].

[22] P.L. George, H. Borouchaki, F. Alauzet, P. Laug, A. Loseille, D. Marcum and L. Marchal, Mesh generation and mesh adaptivity: theory and practice, *Encyclopedia of Computational*

Mechanics, Edited by E. Stein, R. de Borst and T.J.R. Hughes, John Wiley & Sons, Ltd., 2016.

[23] M. Picasso and A. Loseille, Anisotropic, Adaptive Finite Elements for a Thin 3D Plate,*New Challenges in Grid Generation and Adaptivity for Scientific Computing, SEMA SIMAI Springer Series*, Vol 5, pp. 217-230, 2015,

Editorial work

[24] 27th International Meshing Roundtable in *Lecture Notes in Computational Science and Engineering*, Edited by X. Roca and A. Loseille, Springer Nature, 2019. [pdf]

[25] Guest editor for Computer-Aided Design, Elsevier, 2019.

Proceedings with peer review

[26] R. Feuillet, O. Coulaud, A. Loseille, Anisotropic Error Estimate for High-order Parametric Surface Mesh Generation, *28th International Meshing Roundtable*, 2019

[27] V. Chmielarski, W. Ghedhaifi, E. Montreuil, A. Loseille, Comparison between RANS and hybrid RANS/LES simulations of jet/vortex interaction,*AIAA Aviation Forum*, 2019. [pdf]

[28] L. Frazza, A. Loseille, F. Alauzet, Unstructured anisotropic mesh adaptation for 3D RANS turbomachinery applications, *AIAA Aviation Forum*, 2019. [pdf]

[29] M. A. Park, A.Balan, W. K Anderson, M. C. Galbraith, P. Caplan, H. A. Carson, T. R. Michal, J. A. Krakos, D. S. Kamenetskiy, A. Loseille, F. Alauzet, L.Frazza, N. Barral, Verification of unstructured grid adaptation components, *AIAA Aviation Forum*, 2019. [pdf]

[30] M. A. Park, W. L. Kleb, W. T. Jones, J. A. Krakos, T. R. Michal, A. Loseille, R. Haimes and J. Dannenhoffer, Geometry Modeling for Unstructured Mesh Adaptation, *AIAA Aviation Forum, 2019*. [pdf]

[31] C. Tsolakis, N. Chrisochoides, M. A. Park, A. Loseille and T. R. Michal, Parallel Anisotropic Unstructured Grid Adaptation, *AIAA SciTech*, 2019. [pdf]

[32] R, Feuillet, A, Loseille, F. Alauzet, P2 Mesh Optimization Operators, *27th International Meshing Roundtable, Lecture Notes in Computational Science and Engineering*, vol 127, pp. 3-21, 2018. [pdf]

[33] A. Loseille and R. Feuillet, ViZiR: High-order mesh and solution visualization using OpenGL 4.0 graphic pipeline, *AIAA SciTech*, 2018. [pdf]

[34] Emmanuel Montreuil, W. Ghedhaifi, V. Chmielaski, F. Vuillot, F. Gand and A. Loseille , Numerical Simulation of contrail formation on the Common Research Model wing/body/engine configuration, *AIAA SciTech*, 2018. [pdf]

[35] M. A. Park, N. Barral, D.Ibanez, D.S. Kamenetskiy, J. A. Krakos, T. R. Michal and A. Loseille. Unstructured Grid Adaptation and Solver Technology for Turbulent Flows, *AIAA Scitech*, 2018. [pdf]

[36] F. Alauzet, A. Loseille and D. Marcum. On a robust boundary layer mesh generation process, *AIAA Scitech*, 2017. [pdf]

[37] O. Coulaud and A. Loseille, Very High Order Anisotropic Metric-Based Mesh Adaptation in 3D, *Procedia Engineering*, Vol. 163, pp. 353-365, 2016. [pdf]

[38] M. A. Park, J. Krakos, T. R. Michal, A. Loseille and J. J. Alonso, Unstructured Grid Adaptation: Status, Potential Impacts, and Recommended Investments Toward CFD Vision 2030, *46th AIAA Fluid Dynamics Conference, AIAA Aviation*, 2016.

[39] A. Loseille, V. Menier and F. Alauzet, Parallel Generation of Large-size Adapted Meshes, *Procedia Engineering*, Vol 124, pp. 57-69, 2015, [pdf].

[40] M. A. Park, A. Loseille, J. A. Krakos, and T. R. Michal, Comparing Anisotropic Output-Based Grid Adaptation Methods by Decomposition, *22nd AIAA Computational Fluid Dynamics Conference, AIAA Aviation*, 2015.

[41] D. Luquet, R. Marchiano, F. Coulouvrat, I. S. El Din, and A. Loseille, Sonic Boom Assessment of a Hypersonic Transport Vehicle with Advanced Numerical Methods, *22nd AIAA Computational Fluid Dynamics Conference, AIAA Aviation*, 2015.

[42] A. Loseille, A. Dervieux, and F. Alauzet, Anisotropic Norm-Oriented Mesh Adaptation for Compressible Flows, *53rd AIAA Aerospace Sciences Meeting, AIAA SciTech*, 2015.

[43] A. Loseille, D. L. Marcum, and F. Alauzet, Alignment and orthogonality in anisotropic metric-based mesh adaptation, *53rd AIAA Aerospace Sciences Meeting, AIAA SciTech*, 2015.

[44] V. Menier, A. Loseille, and F. Alauzet, Multigrid Strategies Coupled with Anisotropic Mesh Adaptation,*53rd AIAA Aerospace Sciences Meeting, AIAA SciTech*, 2015.

[45] A. Loseille, Metric-orthogonal Anisotropic Mesh Generation, *Procedia Engineering*, Vol 82, pp. 403-415, 2014, [pdf]

[46] I. S. El Din, F. Dagrau and A. Loseille, Computational and Experimental Assessment of Models for the First AIAA Sonic Boom Prediction Workshop Using Adaptive High Fidelity CFD methods, *32nd AIAA Applied Aerodynamics Conference, AIAA Aviation*, 2014.

[47] V Menier, A. Loseille and F. Alauzet, CFD Validation and Adaptivity for Viscous Flow Simulations, *7th AIAA Theoretical Fluid Mechanics Conference, AIAA Aviation*, 2014.

[48] A. Loseille and V. Menier, Serial and Parallel Mesh Modification Through a Unique Cavity-Based Primitive, *Proceedings of the 22nd International Meshing Roundtable*, pp. 541-558, 2013,

[49] A. Loseille and R. Löhner, Cavity-Based Operators for Mesh Adaptation, AIAA Scitech, 2013

[50] A. Loseille and R. Löhner, Robust Boundary Layer Mesh Generation, *Proc. in 21th International Meshing Roundtable*, San Jose, CA, USA, 2012, [Preprint].

[51] E. Mbinky, F. Alauzet and A. Loseille, Higher order interpolation for mesh adaptation, *Short Research note, 21th International Meshing Roundtable*, San Jose, CA, USA, 2012, [Preprint].

[52] A. Loseille and R. Löhner, Boundary layer mesh generation and adaptivity, *49th AIAA Aerospace Sciences Meeting*, AIAA 2011-0894, Olando, FL, USA, 2011, [Preprint].

[53] R. Löhner and A. Loseille, A simple scheme to limit refinement zones for supersonic flows, *49th AIAA Aerospace Sciences Meeting*, AIAA 2011-0470, Olando, FL, USA, 2011, [Preprint].

[54] A. Loseille and R. Löhner, Adaptive anisotropic simulations in Aerodynamics, *48th AIAA Aerospace Sciences Meeting*, AIAA 2010-169, Olando, FL, USA, 2010, [Preprint].

[55] A. Loseille, A. Dervieux and F. Alauzet, A 3D goal-oriented anisotropic mesh adaptation applied to inviscid flows in Aeronautics, *48th AIAA Aerospace Sciences Meeting*, AIAA 2010-1067, Orlando, FL, USA, 2010, [Preprint].

[56] F. Alauzet and A. Loseille, High-order sonic boom prediction by utilizing mesh adaptive methods, *48th AIAA Aerospace Sciences Meeting*, AIAA 2010-1390, Orlando, FL, USA, 2010, [Preprint].

[57] A. Loseille and R. Löhner, On 3D anisotropic local remeshing for surface, volume, and boundary layers, *Proc. in 18th International Meshing Roundtable*, Springer-Verlag, pp. 611-630, Salt-Lake City, UT, USA, 2009, [Preprint], [doi].

[58] A. Loseille and F. Alauzet, Optimal 3D highly anisotropic mesh adaptation based on the continuous mesh framework, *Proc. in 18th International Meshing Roundtable*, Springer-Verlag, pp. 575-594, Salt-Lake City, UT, USA, 2009, [Preprint], [doi].

[59] F. Alauzet and A. Loseille, On the use of space filling curves for parallel anisotropic mesh adaptation, *Proc. in 18th International Meshing Roundtable*, Springer-Verlag, pp. 337-357, Salt-Lake City, UT, USA, 2009, [Preprint], [doi].

[60] A. Dervieux, A. Loseille and F. Alauzet, High-order adaptive method applied to high speed flows, *Proc. of WEHSFF2007*, Moscow, Russia, 2007, [Preprint].

[61] L. Agelas, A. Benali, S. Benteboula, A. Claisse, P. Hav and A. Loseille, Co-refinement of fault surfaces : convexification process, Vol. 24, pp. 60-76, *Proc. of ESAIM2007*, Luminy, France, 2007, [doi].

[62] A. Loseille, A. Dervieux, P. Frey and F. Alauzet, Achievement of second order mesh convergence for discontinuous flows with adapted unstructured meshes, *18th Computational fluids dynamics conference*, AIAA 2007-4186, Miami, FL, USA, 2007, [Preprint].

[63] F. Alauzet, A. Loseille, High-order sonic boom modeling by adaptive method, *Proc. in AAAF 2007*, Sophia-Antipolis, France, 2007.

Research reports

[64] Guillard, H. and Lakhlili, J. and Loseille, A.and Loyer, A.and Nkonga, B. and Ratnani, A. and Elarif, A., Tokamesh : A software for mesh generation in Tokamaks, *INRIA RR-9230*, 2018. [pdf]

[65] A. Loseille and F. Alauzet, Continuous mesh model and well-posed continuous interpolation error estimation, *INRIA RR-6846*, 2009. [pdf]

[66] F. Alauzet and A. Loseille, High order sonic boom modeling by adaptive methods, *INRIA RR-6845*, 2009. [pdf]

Software

[67] **Feflo.a/AMG-Lib** is a robust anisotropic local remeshing sotware. Surface and volume mesh adaptation are handled in a coupled way. It also includes boundary layers mesh generation for RANS simulations. This computer program is one of the core components of the adaptive loop. The following publications [3, 5, 6, 7, 9, 10, 18, 21, 23, 26, 27, 28, 29, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 52, 54] use it.

The boundary layer module alone (abl4flo) is registered with the APP under nbr. IDDN.FR.001.080032.00.S.P.2012.000.10000 The mesh adaptation module alone (ama4flo) is registered with the APP under nbr. IDDN.FR.001.460014.000.S.P.2012.000.10000

It is composed of 500 000 lines of C/Fortran. The software is distributed by Lemma (with the following customers EADS, PCM, Technip, Thalès, Total) and Distene (as an anisotropic adaptive component). It has been used with multiple solvers Cedre(ONERA), COFFEE(INRIA), Elsa(ONERA), FUN3D (NASA), ProjectX/SANS(MIT), SU2 (Stanford University) and Wolf (INRIA).

[68] **ViZiR/ViZiR4** is an interactive mesh visualization and modification software. It is intended for pixel-exact rendering of high-order mesh and solution [33], see `pyamg.saclay.inria.fr` for the freely available components.

It is composed 60 000 lines of C++. **ViZiR** has been used by the following INRIA teams : CASTOR, ECUADOR, BACCHUS, GAMMA3, M3DESIM, POEMS, REO et TROPICS. It is used at ONERA and MIT for accurate ordering on high-order curved meshes and solutions. Since 2011, all post-processing and pictures for all my articles and presentations have been based on **ViZiR/ViZiR4**.

[69] **SHRIMP** is used to perform mesh renumbering for the reduction of the cache misses. It also allows to perform metric-based parallel mesh partitioning for parallel mesh generation. It is one of the core components of the parallel adaptive strategy [9, 39, 48, 59].

Shrimp is composed of 16 000 lines of C. Shrimp is registered with the APP under nbr. IDDN.FR.001.070013.000.S.P. 2009.000.10000.

[70] **Metrix** Metrix is a software that provides by various ways a metric field o govern the mesh generation. Generally, these metrics are derived from error estimates (a priori or a posteriori)

applied to the numerical solution. Metrix computes metric fields from scalar solutions by means of several error estimates: interpolation error, iso-lines error estimate, interface error estimate and goal oriented error estimates. It also contains several modules that handle meshes and metrics. For instance, it extracts the metric associated with a given mesh and it performs some metric operations such as: metric gradation and metric intersection. Most of the publications on error estimates [7, 10, 11, 12, 58, 62] are based on **Metrix**.

Metrix is composed of 52 000 lines of C. Metrix is registered with the APP under nbr. IDDN.FR.001.070014.000.S.P. 2009.000.10000

Contents

Chapter 1

An introduction to mesh adaptation for scientific computing

We first describe the well established unstructured mesh generation methods as involved in the computational pipeline, from geometry definition to surface and volume mesh generation. These components are always a preliminary but a mandatory step to any numerical computations. From a historical point of view, the generation of fully unstructured mesh generation in 3D has been a real challenge so as to design of robust and accurate second order schemes on such unstructured meshes. If the issue of generating volume meshes for geometries of arbitrary complexity is now mostly solved, the emergence of robust numerical schemes on unstructured meshes has paved the way to adaptivity. Indeed, unstructured meshes in contrast with structured or block structured grids have the necessary flexibility to control the discretization both in size and orientation.

In the second part of this chapter, we review the main components to perform adaptative computations: (i) anisotropic mesh prescription *via* a metric field tensor (ii) anisotropic error estimates, and (iii) anisotropic mesh generation. For each component, we focus on a particularly simple method to implement. In particular, we describe a simple but robust strategy for generating anisotropic meshes. Each adaptation entity, *i.e.* surface, volume or boundary layers, relies on a specific metric tensor field. The metric-based surface estimate is then used to control the deviation to the surface and to adapt the surface mesh. The volume estimate aims at controlling the interpolation error of a specific field of the flow.

Several 3D examples issued from steady and unsteady simulations from systems of hyperbolic laws are presented. In particular, we show that, despite the simplicity of the introduced adaptive meshing scheme, a high level of anisotropy can be reached. This includes the direct prediction of the sonic boom of an aircraft by computing the flow from the cruise altitude to the ground, the interaction between shock waves and boundary layers, or the prediction of complex unsteady phenomena in 3D.

The work related to this chapter is part of my early contributions. Initial developments for the continuous mesh and metric error estimates have been done during my PhD thesis [11, 12, 14, 54]. The work on anisotropic mesh generation has been initiated during my postdoctoral time with the following associated publications [53, 57, 58]. The simple mesh adaptation strategy has been used on collaborative work on supersonic and hypersonic aircraft with ONERA and Dassault Aviation [41, 46].

Introduction

For flows involved in aerospace, naval, train and automotive industries or more generally in Computational Fluids Dynamics (CFD), the numerical prediction of a physical phenomenon follows the computational pipeline, see Fig. 1 of previous chapter. From a continuous description of the geometry, a surface then a volume mesh is generated, see Fig. 1.1. This mesh is used as a discrete support to solve a set of partial differential equations (PDEs) by using any typical second order accurate numerical schemes [Abgrall 2001, Cournède 2006, Johnson 1985, Shu 1988]. When unstructured meshes are used, the meshing and computation steps have reached a great level of maturity and automaticity, allowing to quickly modify the design and run a new simulation, even for highly complex geometries [Aubry 2015, Aubry 2009, George 1990, Laug 2010, Löhner 1988b, Marcum 1996, Marcum 2001, Mavriplis 1995]. During the mesh generation process, the sizing of the elements [Aubry 2016] is either based on a user *a priori* knowledge of the flow or is induced by the geometry. To take into account the whole flow features evaluated at the computation step while keeping automaticity, mesh adaptivity is required.

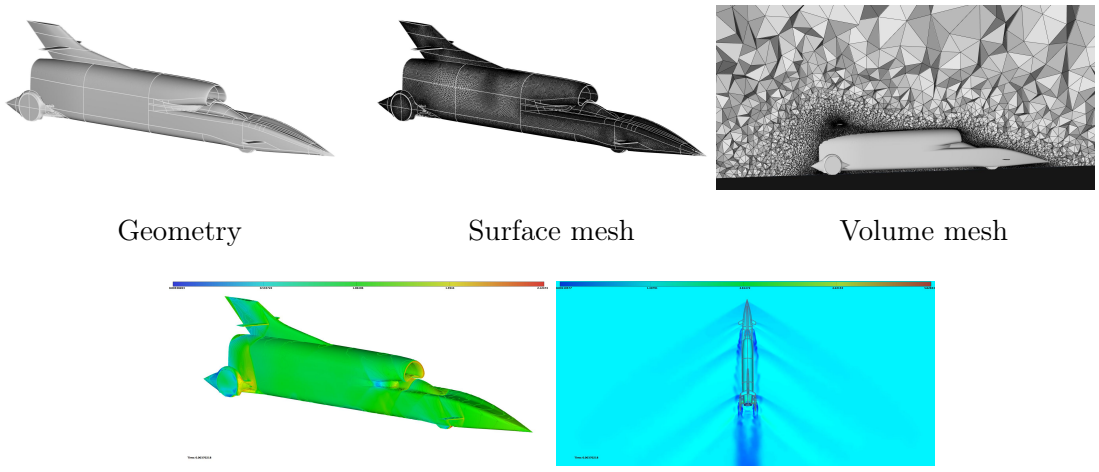


Figure 1.1 – Illustration of the computational pipeline on the Bloodhound© supersonic car

Indeed, we observe that the solutions of nonlinear systems of PDEs like the Euler of Navier-Stokes equations have complex features and multiscale phenomena: shock waves, boundary layers, turbulence, ... When dealing with complex geometries, all these features are present in the flow field and interact with each other. It is then hardly impossible to design a tailored mesh (by means of sources i.e., local information) to capture all these phenomena. To capture accurately them automatically, we typically use specific mesh adaptation procedures. We can distinguish: (i) isotropic and structured grids for turbulent flows, (ii) anisotropic meshes for shock capturing with an anisotropic ratio of the order of $O(1 : 100 - 1000)$ and (iii) highly stretched quasi-structured meshes with a ratio of $O(1 : 10^4 - 10^6)$ for boundary-layers. Many numerical examples have proved that the performance of a numerical scheme is bounded by the quality and the features of the discretization. For instance, we prefer anisotropic meshes to capture accurately shocks [62] while we use Cartesian grids at a turbulent regime to allow

high-order capturing of vortices. In the vicinity of bodies, quasi-structured grids are employed to capture the boundary layer in viscous simulations [Aubry 2009, Marcum 1996]. If all these methods have now reached a good level of maturity, they are generally studied on their own. Consequently it seems difficult to handle together all the optimal meshes for all these phenomena. In this chapter, we will focus on one simple solution to generate all these kinds of meshes within a common framework. In this framework, the requirements on the mesh (for sizes, shapes and orientations) are expressed in terms of a field of metric tensors and dedicated quality functions. We will consider metric fields issued from interpolation error, surface geometric approximation, and boundary-layer model. These fields are then used as a continuous support to drive the adaptation. From a practical point of view, simple anisotropic local operators as edge collapse, point insertion, edge swapping and point smoothing are recursively used to modify and improve the mesh. Note that each operator is monitored by a quality function to ensure that a quality mesh is outputted. This requirement is important to ensure the stability and enhance the performance of the flow solver.

Outline. The chapter is decomposed as follows. In Section 1, we describe the main steps involved in generating a first mesh for complex geometries in an unstructured context. In Section 2, we recall the main concepts of metric-based mesh adaptation. We then define various metric-field expressions used for surface, volume, boundary layer and error control. In Section 3, we describe the algorithms used to generate an anisotropic mesh with respect to a prescribed metric. In Section 4, we briefly comment the adaptive loop, and we illustrate the previous concepts on both steady and unsteady simulations.

1.1 An introduction to unstructured mesh generation

We quickly describe and illustrate on simple examples the principles underlying the generation of unstructured meshes for complex geometries. For a complete description, we refer to the following monographs [Frey 2008, George 1998, Lo 2015, Löhner 2001].

1.1.1 Surface mesh generation

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 representation of a patch exists *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:

$$\text{model} \longrightarrow \text{bodies} \longrightarrow \text{faces} \longrightarrow \text{loops} \longrightarrow \text{edges} \longrightarrow \text{nodes}$$

Each entity of upper level is described by a list of entities of lower level. This is represented in Fig. 1.2 for an 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 watertight valid surface mesh on output for the whole computational domain. This step makes the surface

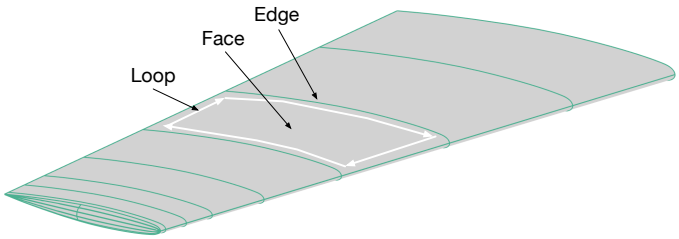


Figure 1.2 – Topology hierarchy (Face, Loop, Edge) of the continuous representation of models using the Boundary REPresentation (BREP).

mesh generation of equal difficulty as volume mesh generation and has been shown not to be trivial [Alleaume 2008].

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. Generally, σ is a NURBS function (Non-Uniform Rational B-Spline) as it is a common tool in geometry modeling and CAD systems [Piegl 1997]. From a conceptual point of view, meshing a parametric surface consists in meshing a 2D domain in the parametric space. However, surface mesh generation is not as naive as it seems, as several issues are to be faced to get a valid surface mesh:

- The mapping function is not bijective, *i.e.,* an infinite number of parameters values may have the same value in $\mathbb{R}^3$;
- A valid mesh in the parametric space may be invalid when mapped to 3D as σ is not necessary monotone;
- Having a uniform mesh in $\mathbb{R}^3$ requires to have a highly anisotropic adapted mesh in $\mathbb{R}^2$ due to the length distortion imposed by σ;
- The typical CAD queries (normal, tangent planes, principal curvatures) are based on the derivatives of σ that may have undefined behaviors especially near the boundaries of the parametric space.

We illustrate this on the mesh of a torus composed of two edges and one face, see Fig. 1.3. We notice that if the mesh in $\mathbb{R}^3$ is perfectly uniform, it is not the case in the parametric space.

For further readings, the aforementioned issues of CAD parameterizations and their consequences for adaptivity are discussed in [38]. Robust meshing of NURBS surface is studied in [Aubry 2015] and implementation details are provided in [Laug 2010].

1.1.2 Volume mesh generation

Once the generation of the surface mesh is completed, a volume mesh is generated to fill the domain with tetrahedra. The surface mesh then becomes an input but also a constraint as all

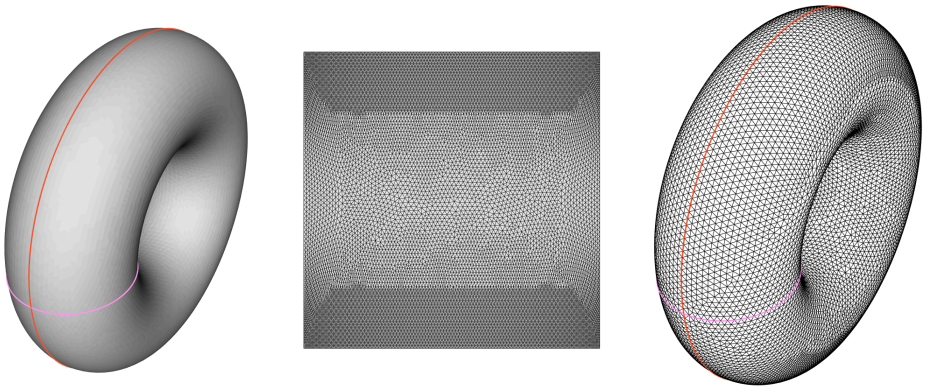


Figure 1.3 – From left to right, CAD of a torus with 2 edges and one face, 2D mesh in the parametric space, and mapped uniform surface mesh.

the input triangles have to match a face of the tetrahedral mesh. Two different approaches have emerged and have proved to be robust *w.r.t.* the complexity of the geometry: the frontal and the Delaunay methods.

The frontal approach is the easiest to understand in its principles. The process starts from the surface mesh that defines an initial front (a set of faces). From this front, a set of optimal points are created such that for each face of the front, an optimally shaped element would be created. This set of points is then checked and filtered to avoid collision and overlapping of faces. A reduced set of points is then inserted one point at a time and the front is updated. The same procedure is repeated until the whole domain is filled. The pros of this approach is that the shape of elements can be controlled and different kinds of meshes can be obtained by modifying the optimal point procedure: Cartesian core, iso-tetrahedra, . . . ,see [Löhner 2001] for more details. If meshes of very high quality are obtained when starting from isotropic surface meshes, the critical steps is in the closure of the front. Indeed, there is no guarantee that the procedure will end up with an empty front. This weakness tends to increase when anisotropic triangles are present in the initial surface mesh. We refer to [Löhner 2014] for an updated description of the frontal approach.

The second approach is a constrained Delaunay method. It starts from an initial simple mesh of a box surrounding the surface mesh (composed of six tetrahedra). We then have the following steps:

1. Insert the points of the surface mesh in the current mesh;
2. Recover the boundary corresponding to the initial surface mesh (list of edges and faces);
3. Fill the interior of the domain by inserting internal points;
4. Optimize the mesh with the smoothing of points and the swap of edges and faces.

Contrary to the frontal approach, a valid 3D mesh is always kept through the entire process. This is due to the insertion procedure based on an iterative process, see Fig. 1.4. Once Step (i) is completed, some faces or edges of the initial mesh may not be present in the current mesh, a boundary

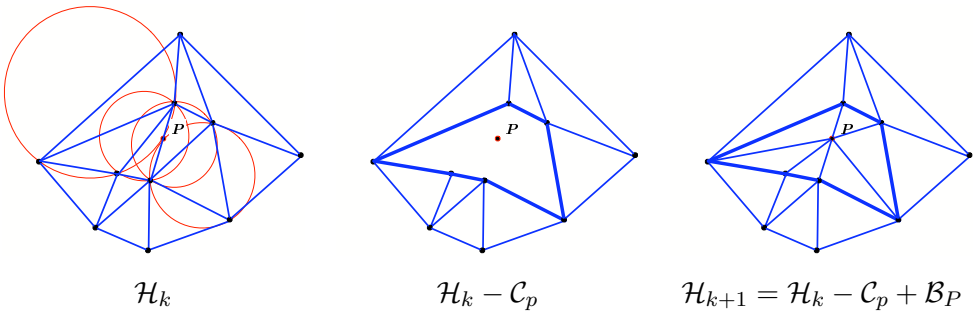


Figure 1.4 – Illustration of the incremental Delaunay insertion of a point in a mesh.

recovery step is used. It is generally used in enforcing these entities by applying successively or randomly standard optimization operators as the swap of edges and faces [George 2003c]. In addition, some theoretical and constructive proofs exist to show that this procedure can succeed to generate a mesh, see [George 2003a, Si 2015]. The most critical step is the second one. However, if we accept to modify the initial surface mesh, this procedure can always succeed to output a volume mesh with a (slightly) modified surface mesh. Consequently, this approach is more robust than the frontal approach. The procedure is illustrated in Fig. 1.5.

Note that a lot of hybrid approaches are combinations of both. The frontal creation of points can be used with Delaunay insertion, or the closure of the front can use a complete constrained Delaunay approach. For the two core methods, a simple example comparing both approaches is depicted in Fig. 1.6.

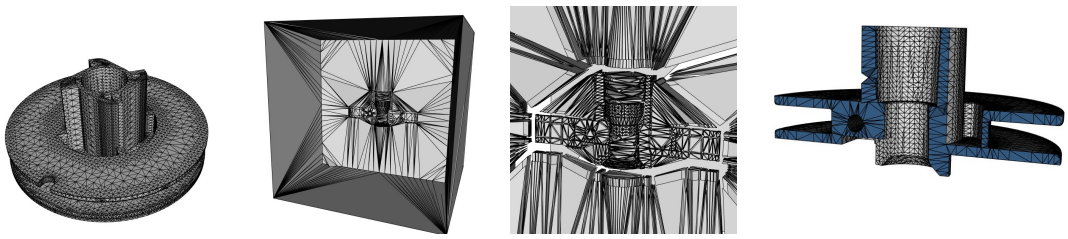


Figure 1.5 – Constrained Delaunay method. From left to right, initial surface mesh, volume mesh after insertion of the surface points, volume mesh after boundary recovery and final mesh by removing elements connected to the initial mesh of the surrounding box.

1.2 Metric-based mesh adaptation

If unstructured meshes have been employed primarily to handle complex geometries, their great flexibility allows us to consider anisotropic mesh adaptation. The intent of adaptivity is then to optimize the ratio between the level of accuracy and the CPU time to run a simulation. The expected gain is mostly motivated by the physical features of the flow, especially for systems of

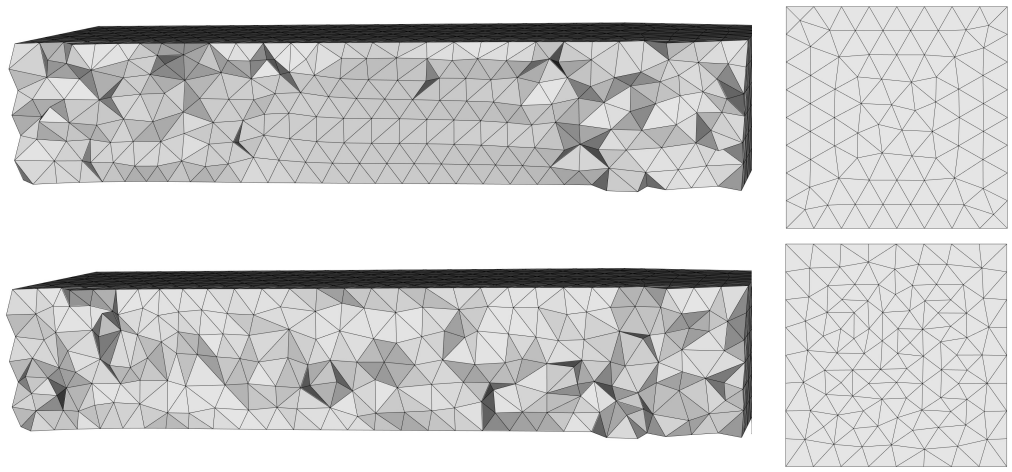
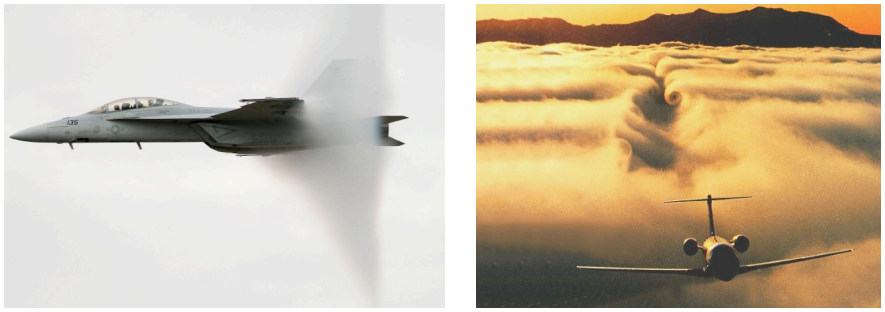


Figure 1.6 – Cuts in a volume mesh filled with the frontal method (top left) and Delaunay insertion (bottom left) and right, 2D square domain filled with frontal (top right) and Delaunay (top bottom).



sonic boom vorticity in wake

Figure 1.7 – Examples of phenomena with strong anisotropic features concentrated in small regions of the domain: shock waves (left), and vorticity (right).

hyperbolic laws where the solutions have strong anisotropic components. It is then clear that using uniform meshes is not optimal (for the distribution of the degrees of freedom) to reach a given level of accuracy. Two examples of flows with anisotropic features are given in Fig. 1.7 with a supersonic flow and the vorticity behind a business jet.

To perform anisotropic mesh adaptation, we have to define the following: (i) a directional error estimate, (ii) a way to prescribe the desired sizes and orientations (iii) and finally a set of mesh modification operators to generate anisotropic meshes. In this section, we introduce the **metric-based** approach where continuous and discrete tensor fields are used to handle (i)-(iii). The key idea is to generate a uniform mesh, **a unit mesh**, with respect to a Riemannian metric space. More precisely, the geometric quantities as length, volume, angle, quality, ..., are then evaluated in this space instead of using the standard Euclidean space.

1.2.1 Metric tensors in mesh adaptation

A metric tensor field of domain Ω is a Riemannian metric space denoted by $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$, where $\mathcal{M}(\mathbf{x})$ is a 3×3 symmetric positive definite matrix. Taking this field at each vertex $\mathbf{x}_i$ of a mesh $\mathcal{H}$ of Ω defines the discrete field $\mathcal{M}_i = \mathcal{M}(\mathbf{x}_i)$. If N denotes the number of vertices of $\mathcal{H}$, the linear discrete metric field is denoted by $(\mathcal{M}_i)_{i=1\ldots N}$. As $\mathcal{M}(\mathbf{x})$ and $\mathcal{M}_i$ are symmetric definite positive, they can be diagonalized in an orthonormal frame, such that

$$\mathcal{M}(\mathbf{x}) = {}^t\mathcal{R}(\mathbf{x})\Lambda(\mathbf{x})\mathcal{R}(\mathbf{x}) \text{ and } \mathcal{M}_i = {}^t\mathcal{R}_i\Lambda_i\mathcal{R}_i,$$

where $\Lambda(\mathbf{x})$ and Λ_i are diagonal matrices composed of strictly positive eigenvalues $\lambda(\mathbf{x})$ and λ_i and $\mathcal{R}$ and $\mathcal{R}_i$ orthonormal matrices verifying ${}^t\mathcal{R}_i = (\mathcal{R}_i)^{-1}$. Setting $h_i = \lambda_i^{-2}$ allows to define the sizes prescribed by $\mathcal{M}_i$ along the principal directions given by $\mathcal{R}_i$. Note that the set of points verifying the implicit equation ${}^t\mathbf{x}\,\mathcal{M}_i\,\mathbf{x} = 1$ defines a unique ellipsoid. This ellipsoid is called the unit ball of $\mathcal{M}_i$ and is used to represent geometrically $\mathcal{M}_i$.
The two fundamental operations in a mesh generator are the computation of length and volume. The length of an edge $\mathbf{e} = [\mathbf{x}_i, \mathbf{x}_j]$ and the volume of an element K are continuously evaluated in $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ by:

$$\ell_{\mathcal{M}}(\mathbf{e}) = \int_0^1 \sqrt{{}^t\mathbf{e}\,\mathcal{M}(\mathbf{x}_i + t\,\mathbf{e})\,\mathbf{e}}\,\mathrm{d}t \text{ and } |K|_{\mathcal{M}} = \int_K \sqrt{\det(\mathcal{M}(\mathbf{x}))}\,\mathrm{d}\mathbf{x}$$

From a discrete point of view, the metric field needs to be interpolated [Frey 2008] to compute approximate length and volume. For the volume, we consider a linear interpolation of $(\mathcal{M}_i)_{1\ldots N}$ and the following edge length approximation is used:

$$|K|_{\mathcal{M}} \approx \sqrt{\det\left(\frac{1}{4}\sum_{i=1}^{4}\mathcal{M}_i\right)}|K| \text{ and } \ell_{\mathcal{M}}(\mathbf{e}) \approx \sqrt{{}^t\mathbf{e}\,\mathcal{M}_i\,\mathbf{e}}\,\frac{r-1}{r\ln(r)}, \tag{1.1}$$

where $|K|$ is the Euclidean volume of K and r stands for the ratio $\sqrt{{}^t\mathbf{e}\,\mathcal{M}_i\,\mathbf{e}}/\sqrt{{}^t\mathbf{e}\,\mathcal{M}_j\,\mathbf{e}}$. The approximated length arises from considering a geometric approximation of the size variation along end-points of $\mathbf{e}$: $\forall t\in[0,1]\, h(t) = h_i^{1-t}\,h_j^t$.
The task of the adaptive mesh generator is then to generate a unit-mesh with respect to $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\boldsymbol{\Omega}}$. A mesh is said to be unit when it is only composed of unit-volume elements and unit-length edges. Practically, these two requirements are combined into a quality function computed in the metric field. A mesh $\mathcal{H}$ is unit with respect to $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ when each tetrahedron $K\in\mathcal{H}$ defined by its list of edges $(\mathbf{e}_i)_{i=1\ldots 6}$ verifies:

$$\forall i\in[1,6],\quad \ell_{\mathcal{M}}(\mathbf{e}_i)\in\left[\frac{1}{\sqrt{2}},\sqrt{2}\right]\quad\text{and}\quad Q_{\mathcal{M}}(K)\in[1,\alpha]\ \text{ with }\ \alpha>0\,, \tag{1.2}$$

with:

$$Q_{\mathcal{M}}(K) = \frac{3^{\frac{1}{3}}}{36}\frac{\sum_{i=1}^{6}\ell^2_{\mathcal{M}}(\mathbf{e}_i)}{|K|_{\mathcal{M}}^{\frac{2}{3}}}\in[1,\infty]. \tag{1.3}$$

Perfect elements are in the range $[1-2]$. This range arises from some discussions on the possible tessellation of $\mathbb{R}^3$ with unit-elements [11]. Note the quality tends to ∞ when the volume tends

to 0. The $\sqrt{2}$ and $1/\sqrt{2}$ factors to control the length of edges are used to avoid to cycle during the remeshing step. If a long edge is split, the two new edges should not be considered too small, in order to avoid an infinite sequence of insertions and collapses.

There exists a large set of adaptive mesh generators that use a metric tensor as an input to generate anisotropic meshes. Let us cite `Bamg` [Hecht 1998] and `BL2D` [Laug 2003] in 2D, `Yams` [Frey 2001b] for discrete surface mesh adaptation and `EPIC` [Michal 2011], `Feflo.a` [54], `Forge3d` [Coupez 2011], `Refine` [Jones 2006], `Gamanic3d` [George 2003b], `MadLib` [Compère 2010], `MeshAdap` [Li 2005], `Mmg3d` [Dobrzynski 2008], `Mom3d` [Tam 2000], `Tango` [Bottasso 2004], `Lib-Adaptivity` developed by [Pain 2001] and `Pragmatic` [Rokos 2015] in 3D.

1.2.2 Techniques for enhancing robustness and performance

The metric field provided has a direct, albeit complex, impact on the quality of the resulting mesh. A smooth and well-graded metric field makes the generation of the anisotropic mesh generation easier and generally improves the final quality. We consider two techniques that tend to give a substantial positive impact on the quality of the resulting mesh: The **anisotropic mesh gradation** tends to smooth the metric field, while the **Log-Eucidean** interpolation allows to properly define metric tensors interpolation, thereby preserving the anisotropy even after a numerous number of interpolations.

Anisotropic mesh gradation. The mesh gradation is a process that smoothes the initial metric field that is generally noisy as it is derived from discrete data. Gradation strategies for anisotropic meshes are available in [Li 2004, Alauzet 2010]. From a continuous point of view, the mesh gradation process consists in verifying the uniform continuity of the metric field:

$$\forall(\mathbf{x},\mathbf{y}) \in \Omega^2 \ \|\mathcal{M}(\mathbf{y}) - \mathcal{M}(\mathbf{x})\| \leq C\|\mathbf{x}-\mathbf{y}\|_2,$$

where C is a constant and $\|.\|$ a matrix norm. This requirement is far more complex than imposing only the continuity of $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$. From a practical point of view, this is done by ensuring that for all couples $(\mathbf{x}_i, \mathcal{M}_i)$ defined on $\mathcal{H}$ verify:

$$\forall(\mathbf{x}_i,\mathbf{y}_j) \in \mathcal{H}^2 \quad \mathcal{G}(\|\mathbf{x}_i-\mathbf{y}_j\|_2)\,\mathcal{M}_i \cap \mathcal{M}_j = \mathcal{M}_j \text{ and } \mathcal{G}(\|\mathbf{x}_i-\mathbf{y}_j\|_2)\,\mathcal{M}_j \cap \mathcal{M}_i = \mathcal{M}_i,$$

where $\mathcal{G}(.)$ is a matrix function defining a growth factor and $\cap$ is the classical metric intersection based on simultaneous reduction [Frey 2008]. This standard algorithm has $O(N^2)$ complexity. Consequently, less CPU-intensive correction strategies need to be devised; we refer to [Alauzet 2010] for some suggestions. Note that bounding the number of corrections to a fixed value is usually sufficient to correct the metric field near strongly anisotropic areas as the shocks. Two options are considered giving either an isotropic growth or an anisotropic growth:

$$\mathcal{G}(d_{ij})\,\mathcal{M}_i = \begin{pmatrix} \eta_1(d_{ij})\,\lambda_1 & & \\ & \eta_2(d_{ij})\,\lambda_2 & \\ & & \eta_3(d_{ij})\,\lambda_3 \end{pmatrix}$$

with

$$(i)\ \eta_k(d_{ij}) = (1+\sqrt{{}^t\mathbf{e}_{ij}\,\mathcal{M}_i\,\mathbf{e}_{ij}}\,\log(\beta))^{-2} \ \text{ or } \ (a)\ \eta_k(d_{ij}) = (1+\lambda_k\,d_{ij}\,\log(\beta))^{-2}, \tag{1.4}$$

where $d_{ij} = \|\mathbf{x}_j - \mathbf{x}_i\|_2$, $\mathbf{e}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ and β the gradation parameter > 1. The isotropic growth is given by law (i) while the anisotropic by law (a). Note that (i) is identical for all directions, contrary to anisotropic law (a) that depends on each eigenvalue along its principal direction. In the sequel, we use the gradation to smooth the transition between the various metric fields: surface and volume, surface and boundary layers.

Log-Euclidean framework and applications. After each point insertion or during the computation of edge lengths, a metric field must be interpolated. Interpolation schemes based on the simultaneous reduction [Frey 2008] lack several desirable theoretical properties. For instance, the unicity is not guaranteed. A framework introduced in [Arsigny 2006] proposes to work in the logarithm space as if one were in the Euclidean one. Consequently, a sequence of n metric tensors can be interpolated in any order while providing a unique metric. Given a sequence of points $(\mathbf{x}_i)_{i=1\dots k}$ and their respective metrics $\mathcal{M}_i$, then the interpolated metric in $\mathbf{x}$ verifying

$$\mathbf{x} = \sum_{i=1}^{k} \alpha_i \, \mathbf{x}_i, \text{ with } \sum_{i=1}^{k} \alpha_i = 1, \text{ is } \mathcal{M}(\mathbf{x}) = \exp\left(\sum_{i=1}^{k} \alpha_i \ln(\mathcal{M}_i)\right). \tag{1.5}$$

On the space of metric tensors, logarithm and exponential operators are acting on metric's eigenvalues directly:

$$\ln(\mathcal{M}_i) = {}^t\mathcal{R}_i \, \ln(\Lambda_i) \, \mathcal{R}_i \quad \text{and} \quad \exp(\mathcal{M}_i) = {}^t\mathcal{R}_i \, \exp(\Lambda_i) \, \mathcal{R}_i.$$

Numerical experiments confirm that using this framework during interpolation allow to preserve the anisotropy. Note that the evaluation of length given by (1.1) corresponds to the Log-Eucldiean interpolation between the two metrics of the edge extremities.

1.2.3 Metric-based error estimates

From the previous concepts, metric-based error estimates are well suited for the generation of anisotropic meshes. We focus on this set of estimates in the sequel. We then describe in more details the case of the interpolation error as it is the easiest to implement.

1.2.4 Controlling the interpolation error

Controlling the linear interpolation error of a given flow field allows to derive a very simple anisotropic metric-based estimate. Interpolation estimate is the first introduced in the pioneering work [Castro-Díaz 1997] by equi-distributing the interpolation error in L^∞ norm. Here, we prefer to control the L^p norm of the interpolation error. Such control allows to recover the order of convergence of the scheme for flows with shocks and to capture all the scales of the numerical solution [62]. Given a numerical solution W_h (density, pressure, Mach number, ...), a solution of higher regularity $R_h(W_h)$ is recovered, so that the following interpolation error estimate [Chen 2007, 12] holds:

$$\|R_h(W_h) - \Pi_h R_h(W_h)\|_{L^p} \leq N^{-\frac{2}{3}} \left(\int_\Omega \det\left(|H_{R_h(W_h)}|\right)^{\frac{p}{2p+3}} \right)^{\frac{2p+3}{3p}} \tag{1.6}$$

where $H_{R_h(W_h)}$ is the Hessian of the recovered solution and N an estimate of the desired number of nodes, and Π_h the piecewise linear interpolate of a function. Using the continuous mesh framework, the point-wise metric tensor minimizing (1.6) norm of the error is given by:

$$\mathcal{M}_{L^p}(W_h) = \det(|H_R(W_h)|)^{\frac{-1}{2p+3}} \, |H_R(W_h)|, \tag{1.7}$$

where $|H_R(W_h)|$ is deduced from $H_R(W_h)$ by taking the absolute value of the eigen-values of $H_R(W_h)$. The optimal metric is obtained by solving a calculus of variation [11, 12], such derivation is summarized in Chapter 4 for the case of high order interpolations.

In the sequel, the interpolation error is controlled in L^2 norm exclusively, while the H_R operator is based on the double L^2 projection, see [Vallet 2007] for a review on numerical derivatives recovery. For the numerical examples, we will use the complexity to control the level of accuracy. The complexity is defined by $\mathcal{C}(\mathcal{M}) = \int_\Omega \sqrt{\det(\mathcal{M})}$. Imposing a complexity of $\mathcal{N}$ leads to the following scaling of the metric:

$$\mathcal{M}_{L^p}(W_h, \mathcal{N}) = \left(\frac{\mathcal{N}}{\int_\Omega \det(|H_R(W_h)|)^{\frac{p+1}{2p+3}}} \right) \det(|H_R(W_h)|)^{\frac{-1}{2p+3}} \, |H_R(W_h)|. \tag{1.8}$$

For time dependent problems, we use an extension of the multi-scale approach [Alauzet 2011, 7]. The process may be summarized as follows. The whole time frame $[0, t_f]$ is split in n_t sub-intervals:

$$[0, t_f] = \bigcup_{i=1}^{n_t} [t_i, t_{i+1}], \text{ with } t_1 = 0 \text{ and } t_{n_t} = t_f.$$

Then, the main idea consists in deriving n_t meshes $(\mathcal{H}_i)_{i=1,n_t}$ that minimize the interpolation error on the solution u defined on Ω:

$$\text{Find } (\mathcal{H}^i_{opt})_{i=1,n_t} = \min \sum_{i=1}^{n_t} \int_{t_i}^{t_{i+1}} \int_\Omega |W - \Pi_h W|^p \, \mathrm{d}\Omega \, \mathrm{d}t \text{ for all } i \in [1, n_t]. \tag{1.9}$$

The solution of this problem gives a sequence of metric tensor fields $(\mathcal{M}_i)_{i=1,nt}$ for each sub interval $[t_i, t_{i+1}]$. The continuous problem is then solved using a calculus of variations. From a practical point of view, on a time interval $[t_i, t_{i+1}]$, the flow solver outputs a sequence of n_s solutions every $\Delta t = (t_{i+1} - t_i)/n_s$. From this sequence, a maximal or mean Hessian $\tilde{H}_i$ is recovered [Alauzet 2011] accounting for the error for the sub-window time frame. Then, once all $\tilde{H}_i$ are recovered, a global normalization is applied for the whole time frame $[0, t_f]$ to derive $(\mathcal{M}_i)_{i=1,nt}$, see Fig. 1.20.

A detailed review of metric-based estimates for steady and unsteady problems can be found in [10].

1.2.5 Geometric estimate for surfaces

Controlling the deviation to a surface has been studied in previous works, see [Aubry 2011, Frey 2000, Frey 2003] for anisotropic remeshing. We recall that the surface remeshing is done by considering only discrete data, either inherited by the CAD or recovered directly from the

discrete mesh. Prior to surface remeshing, normals and tangents are then assigned to each boundary point. We denote by $\mathbf{n}_i$ the normal of the vertex $\mathbf{x}_i$. As in [Frey 2000], a quadratic surface model is computed locally around a surface point $\mathbf{x}_i$. Starting from the topological neighbors of $\mathbf{x}_i$, the coordinates of each point are mapped onto the local orthonormal Frenet frame $(\mathbf{u}_i, \mathbf{v}_i, \mathbf{n}_i)$ centered in $\mathbf{x}_i$. Vectors $(\mathbf{u}_i, \mathbf{v}_i)$ lie in the orthogonal plane to $\mathbf{n}_i$. We denote by $(u_j, v_j, \sigma_j) = ({}^t\mathbf{x}_j.\mathbf{u}_i, {}^t\mathbf{x}_j.\mathbf{v}_i, {}^t\mathbf{x}_j.\mathbf{n}_i)$ the new coordinates of vertex $\mathbf{x}_j$. $\mathbf{x}_i$ is set as the new origin so that $(u_i, v_i, \sigma_i) = (0, 0, 0)$. The surface model consists in computing by a least squares approximation a quadratic surface:

$$\sigma(u, v) = au^2 + bv^2 + cuv, \text{ where } (a, b, c) \in \mathbb{R}^3. \tag{1.10}$$

The least squares problem gives the solution to $\min_{(a,b,c)} \sum_j |\sigma_j - \sigma(u_j, v_j)|^2$, where j is the set of neighbors of $\mathbf{x}_i$. Note that at least 3 neighbors points are necessary to recover the surface model. Finally, if the degree of $\mathbf{x}_i$ is d, the linear system is:

$$A\,X \;=\; B \iff \begin{pmatrix} u_1^2 & v_1^2 & u_1v_1 \\ \vdots & \vdots & \vdots \\ u_d^2 & v_d^2 & u_dv_d \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} \sigma_1 \\ \vdots \\ \sigma_d \end{pmatrix}.$$

The least square formulation consists in solving ${}^tA\,A\,X = {}^tA\,B$. From this point, one may apply the surface metric given in [Frey 2000]. We propose here a simplified version. We can first remark that the orthogonal distance from the plane $\mathbf{n}_i^\perp$ onto the surface is given by $\sigma(u, v)$ by definition. The trace of $\sigma(u, v)$ on $\mathbf{n}_i^\perp$ is a function that gives directly the distance to the surface. The 2D surface metric $\mathcal{M}_S^{2D}$ such that the length $\ell_{\mathcal{M}_S^{2D}}((u, v))$ is constant and equal to ε is easy to find starting from the diagonalization of the quadratic function (1.10). Geometrically, it consists in finding the maximal area metric included in the level-set ε of the distance map. We assume that $\mathcal{M}_S^{2D}$ admits the following decomposition:

$$\mathcal{M}_S^{2D} = (\bar{\mathbf{u}}_S, \bar{\mathbf{v}}_S) \begin{pmatrix} \lambda_{1,S} & 0 \\ 0 & \lambda_{2,S} \end{pmatrix} {}^t(\bar{\mathbf{u}}_S, \bar{\mathbf{v}}_S), \text{ with } (\bar{\mathbf{u}}_S, \bar{\mathbf{v}}_S) \in \mathbb{R}^{2\times 2}.$$

If we want to achieve the same error as the initial mesh, we compute $\varepsilon = \min_j |\sigma(u_j, v_j)|$ among the neighbors of $\mathbf{x_i}$. The anisotropic 2D metric achieving an ε error becomes:

$$\mathcal{M}_S^{2D}(\varepsilon) = \frac{1}{\varepsilon}\mathcal{M}_S^{2D}.$$

The final 3D surface metric in $\mathbf{x}_i$ is:

$$\mathcal{M}_S(\varepsilon) = (\mathbf{u}_S, \mathbf{v}_S, \mathbf{n}_i) \begin{pmatrix} \frac{\lambda_{1,S}}{\varepsilon} & 0 & \\ 0 & \frac{\lambda_{2,S}}{\varepsilon} & 0 \\ 0 & 0 & h_{max}^{-2} \end{pmatrix} {}^t(\mathbf{u}_S, \mathbf{v}_S, \mathbf{n}_i), \tag{1.11}$$

$$\text{with } \begin{cases} \mathbf{u}_S = \bar{\mathbf{u}}_S(1)\mathbf{u}_i + \bar{\mathbf{u}}_S(2)\mathbf{v}_i, \\ \mathbf{v}_S = \bar{\mathbf{v}}_S(1)\mathbf{u}_i + \bar{\mathbf{v}}_S(2)\mathbf{v}_i. \end{cases}$$

The parameter h_{max} is initially chosen very large (e.g. 1/10 of the domain size). This normal size is corrected during various steps. A first anisotropic gradation using (1.4)(i) is applied on surface edges only. The surface metric is then intersected with any computation metrics as given by (1.7). These two steps set automatically a proper element size in the normal direction. Note that different local surface estimates can be derived depending on the local information available, see [Vlachos 2001b].

During the mesh adaptation process, the previous procedure is not applied independently on each current mesh to be adapted. On the contrary, the surface metric is computed once on a fixed background mesh. This metric is then interpolated on each adapted mesh in the course of the iterative process. This tends to maintain a consistent gap with respect to the true geometry.

1.2.6 Boundary layers metric

Boundary layers mesh generation has been devised to capture accurately the speed profile around a body during a viscous simulation. The width of the boundary layer depends on the local Reynolds number [Löhner 2001]. So far, the generation of the boundary layer grids has been carried out by an extrusion of the initial surface along the normals to the surface or by local modification of the mesh [Marcum 1996]. Note that using the normals as sole information requires several enrichments to obtain a smooth layers transition on complex surfaces [Aubry 2009]. In this chapter, we consider a simple approach that is naturally compatible with anisotropic adaptation procedures. The idea consists in representing the boundary layer mesh by a continuous metric field.

The distance to the body is computed using classical algorithms of level-set methods, see [Löhner 2001]. This step can be done quickly and has generally a complexity of $O(N\ln(N))$ where N is the number of points in the current mesh. (Furthermore, note that from a practical point of view, this function is evaluated only in the vicinity of the body). To control the size in the tangential directions, a metric is recovered from the current surface mesh or a background mesh. It takes advantage of the Log-Euclidean framework. Starting from elements $(K)_{P\in K}$ of vertex P, the ball of vertex P, the unique surface metric tensor $\mathcal{M}_K$ (for which K is unit) is computed by solving the following 6×6 linear system:

$$(S)\ \begin{cases} \ell^2_{\mathcal{M}_K}(\mathbf{e_1}) = 1 \\ \dots \\ \ell^2_{\mathcal{M}_K}(\mathbf{e_6}) = 1\,. \end{cases} \tag{1.12}$$

where $(\mathbf{e}_i)_{i=1,6}$ are elements edges. (S) has a unique solution as long as the volume of K is not null. The logarithm of each metric is computed so that a classical Euclidean mean weighted by the area of elements is done. Finally, the body point metric $\mathcal{M}_P$ is mapped back using the exponential operator:

$$\mathcal{M}_P = \exp\left(\frac{\sum_{P\in K}|K|\ln(\mathcal{M}_K)}{\sum_{P\in K}|K|}\right).$$

The final boundary layers metric is based for a continuous exponential law of the form $h_0\exp(\alpha\phi(.))$, where h_0 is the initial boundary layer size and α the growing factor. For a volume point $\mathbf{x}_i$, the boundary layers metric depends on the body point P_i for which the minimum distance is reached. The following operations conclude this step:

1. Compute the local Frenet frame $(\mathbf{u}_i, \mathbf{v}_i, \nabla\Phi(\mathbf{x}_i))$ associated with $\nabla\Phi(\mathbf{x}_i)$
2. Set the size in the normal direction to $h_{\mathbf{n}_i} = h_0 \exp(\alpha\,\Phi(\mathbf{x}_i))$, the sizes in the orthogonal plane to:
$$h_{\mathbf{u}_i} = \left({}^t\mathbf{u}_i\,\mathcal{M}_{P_i}\,\mathbf{u}_i\right)^{-2} \text{ and } h_{\mathbf{v}_i} = \left({}^t\mathbf{v}_i\,\mathcal{M}_{P_i}\,\mathbf{v}_i\right)^{-2},$$
3. The final metric is given by:
$$\mathcal{M}_{bl}(\mathbf{x}_i) = {}^t(\mathbf{u}_i, \mathbf{v}_i, \nabla\Phi(\mathbf{x}_i)) \begin{pmatrix} h_{\mathbf{u}_i}^{-2} & & \\ & h_{\mathbf{v}_i}^{-2} & \\ & & h_{\mathbf{n}_i}^{-2} \end{pmatrix} (\mathbf{u}_i, \mathbf{v}_i, \nabla\Phi(\mathbf{x}_i)). \tag{1.13}$$

The key idea is again to simplify the coupling by using only metric tensor fields. Indeed, taking altogether the viscous and un-viscous contributions simply consists in intersecting the corresponding metric tensor fields.

1.3 Algorithms for generating anisotropic meshes

This section describes the local operators used to adapt the mesh once one or several tensor fields are provided on input. We then describe additional operators used to optimize the mesh and to guarantee an optimal time step for unsteady simulations.

1.3.1 Insertion and collapse

To generate a unit-mesh in a given metric field $(\mathcal{M}_i)_{i=1\ldots N}$, two operations are recursively used: edge collapse and point insertion on edge.
The starting point for the insertion of a new point on an edge $\mathbf{e}$ is the shell of $\mathbf{e}$ composed of all elements sharing this edge. Each element of the shell is then divided into two new elements. The new point is accepted if each new tetrahedron has a positive volume. When a point is inserted on a boundary edge, either a linear approximation of the surface is used or a query to the CAD. The newly inserted point is created at the mid-edge point in the metric. To compute it, we first evaluate the size of the current edge with respect to the two end-points $(A, \mathcal{M}_A)$ and $(B, \mathcal{M}_B)$:
$$\ell_{\mathcal{M}_A} = \sqrt{{}^t AB\,\mathcal{M}_A\,AB} \text{ and } \ell_{\mathcal{M}_B} = \sqrt{{}^t AB\,\mathcal{M}_B\,AB}.$$
If $\ell_{\mathcal{M}_A}$ equals $\ell_{\mathcal{M}_B}$, the mid-edge point in the metric is the geometric mid-point $\frac{1}{2}(A+B)$. When they differ, we need to solve the nonlinear problem in $t \in [0,1]$ arising for the length approximation of (1.1):
$$\text{Find } t \text{ such that} \frac{1}{2} = \ell_{\mathcal{M}_A} \frac{r^t - 1}{\log(r)} \text{ with } r = \frac{\ell_{\mathcal{M}_B}}{\ell_{\mathcal{M}_A}}. \tag{1.14}$$
We use a dichotomy approach to solve (1.14), the mid-point is then $(1-t)\,A + t\,B$.
The edge collapse starts from the ball of the vertex to be deleted. Again, for the deletion of points inside the volume, the only possible rejection is the creation of a negative volume element. A special care is also required to avoid the creation of an element that already exists, see Fig. 1.8

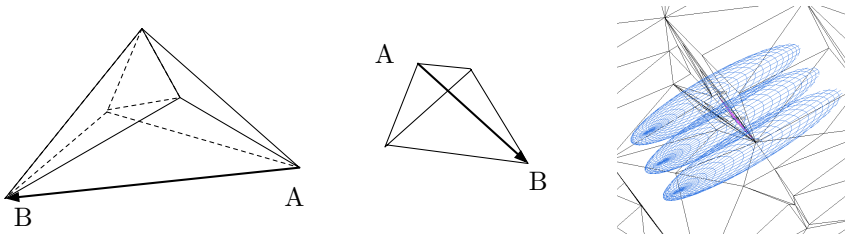


Figure 1.8 – Left and middle, volume and surface collapse of edge AB leading to the creation of an element that already exists . Right, example where an edge is recursively refined to get a unit-length without checking the length requirement in the edge's orthogonal direction; the configuration may lead to edges acting as a barrier for future refinement.

(left and middle). The rejections are more complicated in the case of a surface point. We first avoid each collapse susceptible to modify the topology of the object. This is simply done by assigning an order on each surface point types: corner, ridge (line), inside surface. The collapse can also be rejected if the normal deviation between old and new normals becomes too large. Currently, if $\mathbf{n}$ denotes the normals to an old face, we allow the collapse if each new normal $\mathbf{n}_i$ verifies ${}^t\mathbf{n}\,\mathbf{n}_i > \cos(\pi/4)$. Note that the control to the surface deviation is given by the surface metric and so it does not need to be handled directly in the collapse operation.
With these operations, the core of the adaptive algorithm consists in scanning each edge of the current mesh and, depending on its length, creating a new point on the edge or collapsing the edge. An edge is declared too small or too large according to the bounds given in (1.2). Without any more considerations, such adaptive mesh generator is known not to be efficient and to require a lot of CPU consuming optimizations as point smoothing and edge swapping. This inefficiency is simply due to the locality of these operations. Compared to an anisotropic Delaunay kernel [Dobrzynski 2008], when an edge needs to be refined, the metric lengths along the orthogonal directions are controlled by the creation of the cavity. Consequently, in one shot, the area of refinement must be large. With the present approach, the size is controlled along one direction only (along the edge being scanned). Consequently, one can reach intractable configurations where the same initial edge is refined successively to get the desired size whereas the sizes in the other directions get worse. A typical configuration is depicted in Fig. 1.8 (right).

A simple way to overcome this major drawback is to use the quality function (1.3) together with the unit-length check. This supplementary check can be done at no cost since a lot of information can be reused: the volume is already computed, as well as the length of the edges. By simply computing the quality function, we give to these operators the missing information on the orthogonal directions of the current scanned edge. For the collapse, to decide which vertex is deleted from the edge, the qualities of the two configurations are compared and the best one is kept. For an optimal performance, two parameters are added in the rejection cases of insertion and collapse: a relative quality tolerance $q_r \geq 1$ and a global quality tolerance q_a. Indeed, it seems particularly interesting not to try to implement a full descent direction by imposing the quality to increase on each operation. We prefer to allow the quality to decrease in order to get out of possible local minima. Consequently, a new configuration of elements is accepted if:

$$q_r\, Q_{\mathcal{M}}^{ini} \leq Q_{\mathcal{M}}^{new} \quad \text{and} \quad Q_{\mathcal{M}}^{new} < q_a,$$

where $Q_{\mathcal{M}}^{ini}$ is the worse element quality of the initial configuration and $Q_{\mathcal{M}}^{new}$ is the worse quality of the new configuration. This approach is similar to the simulated annealing global optimization technique [Kirkpatrick 1983]. Note that the current version does not fully implement the classical metropolis algorithm where the rejection is based on a random probability. To ensure the convergence of the algorithm, the relative tolerance q_r is decreased down to 1 after each pass of insertions and collapses. At the end of the process, the absolute tolerance q_a is set up to the current worse quality among all elements.

1.3.2 Optimizations and enhancements for unsteady simulations

In addition to the quality-driven insertion and collapse, we use standard anisotropic mesh optimization techniques such as edges and faces swaps and point smoothing in order to increase the level of anisotropy and the quality of the mesh. By improving the overall quality, they usually improve the stability of the flow solver as well. For unsteady simulations, we add an additional control parameter in order to ensure that an optimal time step is guaranteed.

Swaps of edges and faces are standard mesh modifications operators, see [Freitag 1997, Frey 2008]. In the context of anisotropic remeshing, theses operators are simply monitored by anisotropic quality (1.3). Once the topological and geometrical validity of a swap is verified (positive volume and valid new configurations), it is actually performed only if the quality of the new configuration is strictly lower that the initial quality. We use an improvement factor $q_r = 0.95$ for all the numerical examples.

The point smoothing is also a popular simple operator [Frey 2008]. It consists in computing a new optimal position of a vertex to improve the quality of the surrounding elements. The main difficulty is the computation of the optimal position. In our case, we also want to optimize the length distribution. Consequently, for an edge PP_i with metric $\mathcal{M}(P)$ and $\mathcal{M}(P_i)$, the optimal point position of P is approximated in a Riemannian way by computing :

$$\theta = 1 - \log\left(\frac{\ell_{\mathcal{M}}(P)}{\ell_{\mathcal{M}}(P) - \log(r)}\right)\frac{1}{\log(r)} \text{ with } \begin{cases} \ell_{\mathcal{M}}(P) &= \sqrt{PP_i\mathcal{M}(P)PP_i}, \\ \ell_{\mathcal{M}}(P_i) &= \sqrt{PP_i\mathcal{M}(P_i)PP_i}, \\ r &= \ell_{\mathcal{M}}(P)/\ell_{\mathcal{M}}(P_i). \end{cases}$$

The formula arises from seeking the optimal size to get a unit-edge length along PP_i:

$$\int_0^{\theta} \ell_{\mathcal{M}}(P)^{t-1}\ell_{\mathcal{M}}(P_i)^{-t}\mathrm{d}t = 1.$$

Then the optimal position for P from P_i is :

$$P_{opt_i} = P + \theta\, P_iP.$$

This procedure is repeated with all the neighboring vertices of P:

$$P_{opt} = \alpha P + \frac{1-\alpha}{n_P}\left(\sum_{i=1}^{n_P} P_{opt_i}\right) \text{ with } \alpha \in [0,1].$$

If P_{opt} generates positive volume elements and improves the final quality, P is moved to this new position. Starting with $\alpha = 0.2$, a greater value of α is considered in case of rejection. Note that

the metric of P is interpolated at the new position to evaluate the new quality. When a surface point is moved, it is also projected back to the surface and the surface deviation is checked in a similar way as for the insertion and collapse operators.

For unsteady simulations, the mesh adaptation becomes critical as the CPU time of the simulation depends on the quality of the worse element. Indeed, when an explicit time stepping is used, the minimal time step governs the speed of the simulation. Consequently, the minimal size (or height) generated during the remeshing process may impact drastically the CPU time. If the generated size if 0.01 of the minimal target, then the whole CPU time will be multiplied by 100. To overcome this issue, we add an additional control to the quality based on the height of the tetrahedra. We start from the definition of the minimal height of a tetrahedron:

$$h^2 = \frac{1}{3}\frac{V}{S_{max}}, \tag{1.15}$$

where h is the minimal height, V the volume and S_{max} the maximal area of the faces. For each provided metric, we consider then the regular tetrahedron of side h_1, h_2, h_3, where $(h_i)_i$ are the unit lengths along the eigenvectors of the metric. Then, assuming that the sizes may be in the range $[\frac{1}{\sqrt{2}}h_i, \sqrt{2}h_i]$, see (1.2), we can estimate the global minimal height h_{tar} using

$$h_{tar} = \max(\lambda_1, \lambda_2, \lambda_3)^{-2}\frac{\sqrt{3}}{6}, \tag{1.16}$$

where $(\lambda_i)_{i=1,3}$ are the eigenvalues of $\mathcal{M}(P)$ and $\frac{\sqrt{3}}{6}$ the height of the regular tetrahedron. A mesh modification is then rejected if the minimal height of the new set of tetrahedra is lower than h_{tar} and the minimal height of the initial set of elements. Numerical experiments have proved that this additional constraint does not have a negative impact on the level of anisotropy while preserving an optimal CPU time step.

1.4 Adaptive algorithm and numerical illustrations

The previous mesh adaptation strategy is used inside an adaptive loop that couples the error estimations, the mesh adaptation and the flow solver. In this section, we give some additional details on the components that are not relative to the local remeshing. We then first validate the full adaptive approach on a supersonic wing-body configuration where the adapted numerical solution is compared with experiments. Then, we consider the direct sonic boom prediction of a complex aircraft. The adaptive strategy is then applied to the prediction of boundary layer/shock interaction. Finally, we consider unsteady simulations with the double Mach reflection and a blast prediction.

1.4.1 Adaptive loop

The complete adaptive algorithm for steady simulations is composed of the following steps.

1. Compute the flow field (i.e. converge the flow solution on the current mesh);
2. Compute the metric estimates: surface, volume, boundary layers, etc.

3. Generate a unit mesh with respect to these metric fields;

4. Re-project the surface mesh onto the geometry using the CAD data or a fixed background mesh;

5. Interpolate the flow solution on the new adapted mesh;

6. Goto 1.

For Step 1, two flow solvers have been used in the numerical section. The first flow solver, `FEFLO` [Löhner 2001], works on unstructured grids with finite element discretization of space and edge-based data structures. The Galerkin edge-fluxes are replaced by numerically consistent fluxes, typically given by approximate Riemann solvers (van Leer, Roe, HLLC, ...) with limited variables (van Leer, van Albada, ...). The second flow solver is `WOLF` [Alauzet 2010] and it uses a mixed Finite Element/Finite Volume discretization with a MUSCL extrapolation. Both codes have been verified to be second order accurate on smooth flows and second order accurate for flows with shocks by using adaptivity [62, 54]. The flow solvers use an implicit LU-SGS scheme [Luo 1998, 44]. `FEFLO` is used for simulations 4.4 and 4.5, `WOLF` for simulations 4.2, 4.3 and 4.6. For the unsteady simulations, an explicit time stepping based on Runge-Kutta schemes is used and the explicit control of the height of the tetrahedra of Section 3.2 is activated.

For Step 4., if the surface approximation ε is small enough with respect to the minimal metric size controlling the interpolation error, the simple smoothing procedure usually succeeds to directly move the point onto the geometry. For more complex cases, with boundary layer or when the surface approximation is low, most advanced operators like the cavity-based operators [48] are needed.

For all the simulations, we use a 8-processors 64-bits MacPro with an IntelCore2 chipsets with a clockspeed of 2.8GHz with 32Gb of RAM. The flow solver is multi-threaded while the local remeshing is serial. The final metric field (multi-scale, surface, boundary-layer) is always smoothed by using isotropic gradation law (1.4)(i), with a parameter of 1.2.

To evaluate the level of anisotropy, we use the anisotropic ratio and anisotropic quotients of an element. Both measures are uniquely defined by computing the metric solution of (1.12), and then by evaluating the following quantities from its eigenvalues with $h_i = \lambda_i^{-\frac{1}{2}}$:

$$r = \frac{\max_{i=1,3} h_i}{\min_{i=1,3} h_i} \text{ and } q_i = \frac{h_i^3}{h_1\, h_2\, h_3}$$

Anisotropic quotients measure the gain with respect to an isotropic mesh adaptation, in particular, they increase when two anisotropic directions exist.

For all cases, the initial meshes are generated by using either a constrained Delaunay approach for simulations 4.2 and 4.3 and a frontal approach for simulations 4.4, 4.5 and 4.6. Note that only the material described in the previous sections are used. The interpolation error in L^2 norm is used in addition to the surface metric, metric smoothing and boundary layer metric described in Section 2. The algorithm used to generate the meshes exactly fits the procedures given in Section 3.

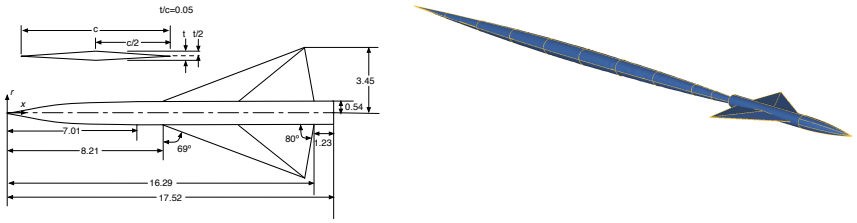


Figure 1.9 – Wing-body example: Left, planform of the wing-body model. Right, CAD of the model equipped with a parabolic sting to emulate experiment apparatus.

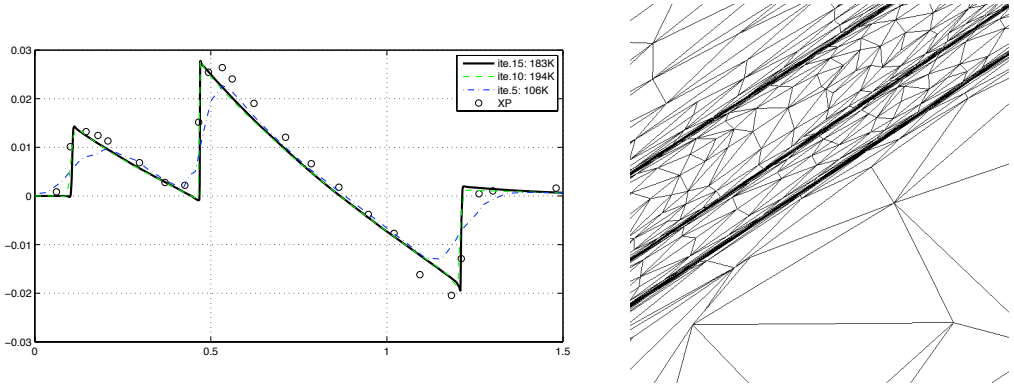


Figure 1.10 – Wing-body example: Left, normalized pressure signature $\frac{p-p_\infty}{p_\infty}$ at $R/L = 3.6$ for the final meshes for each fixed complexity. Right, closer view of the final anisotropic adaptive mesh near the observation line.

1.4.2 A wing-body configuration

The first example is a supersonic flows around the 4^{th} wing-body configuration described in the report [Hunton 1973]. The planform of the model and the corresponding CAD are depicted in Fig. 1.9. The inflow is at Mach 1.68 with a lift of 0.15. We observe the pressure below the aircraft at a distance $R = 3.1\,L$ where L is the reference length of the aircraft (here 17.52 cm). Experimental data are available at this distance, see [Hunton 1973]. The adaptive process is based on metric (1.7) coupled with the surface metric (1.11) with $\varepsilon = 0.001$. The simulation is composed of 3 steps at the following complexity : 25 000, 50 000 and 75 000 with 5 sub-iterations at a fixed complexity yielding to a total of 15 iterations. We control the interpolation error of the Mach number in L^2 norm. The final mesh is here composed of 283 625 vertices and 1 582 309 tetrahedra. The worst volume quality is 20 and the worst surface quality is 9.9 The average anisotropic ratio is 61 and the mean anisotropic quotient is 2711. 92 % and 99.9 % of the volume and surface edges respectively are unit. The total CPU time for this run is 61 mn. This case features 3 strong shocks that are well and early captured by the adaptive process, see Fig. 1.10 for comparisons with experiments.

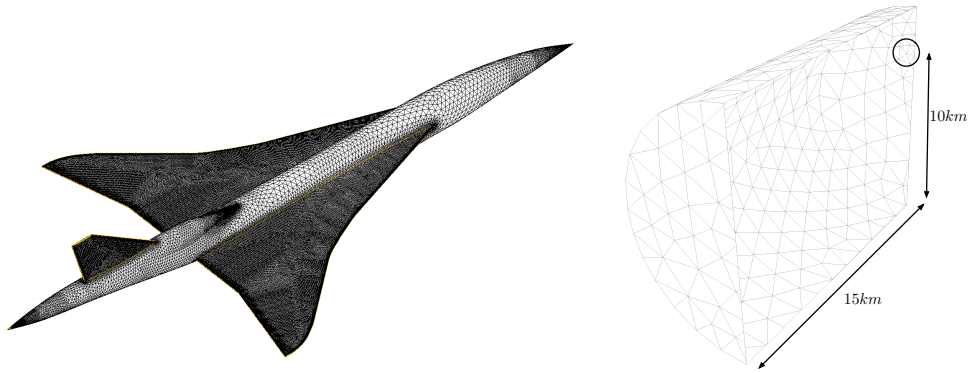


Figure 1.11 – SSBJ example: Left, initial surface mesh of the SSBJ geometry, right, computational domain with the position of the aircraft in the domain.

1.4.3 Direct sonic boom simulation

We consider in this example the accurate prediction of the pressure signal below the SSBJ design provided by Dassault-Aviation. The length of the aircraft is $L = 43$ m while the distance of observation from the aircraft is denoted by R. The initial surface mesh is depicted in Fig. 1.11. The aircraft is put in a 10 km domain as depicted in Fig. 1.11 (right). The initial mesh was generated automatically by using an advancing-front technique [Löhner 1988b]. The size ratio in the initial mesh is $h_{min}/h_{max} = 1e^{-9}$ and the volume of the elements ranges from $5.4e^{-11}$ to $4.7e^{10}$. The flow condition is Mach number 1.6 with an angle of attack of 3 degrees. Our intent is to observe the pressure field for various R up to 9 km. This corresponds to a ratio R/L of about 243. According to the flow conditions, for $R = 9$ km, the length of the propagation of the shock waves emitted by the SSBJ is actually around 15 km.

The interpolation error on the Mach number in L^2 norm is controlled and the surface is controlled with (1.11) and $\varepsilon = 0.001$. The strategy employed here is based on 30 adaptations at the following complexities: 80 000, 160 000, 240 000, 400 000, 600 000 and 800 000. Each step is composed of 5 sub-iterations at a fixed complexity. The final mesh is composed of 3 299 367 vertices and 19 264 402 tetrahedra only. The average anisotropic ratio is 1907 while the mean anisotropic quotient is 50 3334. All the scales involved in this simulation are depicted in Fig. 1.16. This example shows that a very high level of anisotropy is reached using unstructured mesh adaptation. Indeed, it is at least one order of magnitude higher than in the previous examples. Local refinement allows to keep a maximum accuracy and enables to generate quality anisotropic meshes. We mention that for each generated mesh, the worst element quality computed with (1.3) is always below 50 for the volume and below 20 for the surface while the percentage of unit elements is always greater than 90 %. In addition, the flow solver still converges on such meshes leading to accurate pressure signatures for $R/L \approx 250$. Anisotropic ratios and quotients for the whole sequence of meshes are reported in Table 1.1. They are increasing along the iterations. This shows that the accuracy across the shocks is increasing while the sizes in the anisotropic directions are decreasing at a lower rate. The fact that the anisotropic quotient is increasing simply shows that there exist two anisotropic directions. Note that using (1.7) avoid to prescribe a minimal size during the adaptation leading to even stronger anisotropy. This property is due

Table 1.1 – SSBJ example: Properties of each final adapted mesh : mean anisotropic ratio, mean anisotropic quotient, number of vertices and number of tetrahedra for each complexity. The last column gives the cumulative CPU time.

Iteration	Complexity	Ratio	Quotient	# Vertices	# Tet.	CPU time
5	80 000	200	10 964	432 454	2 254 826	1 h 10 mn
10	160 000	383	30 295	608 369	3 294 197	2 h 54 mn
15	240 000	698	81 129	1 104 910	6 243 462	6 h 9 mn
20	400 000	1 089	177 295	1 757 865	10 125 724	11 h 15 mn
25	600 000	1 575	340 938	2 572 814	14 967 820	18 h 47 mn
30	800 000	1 907	503 334	3 299 367	19 264 402	28 h 35 mn

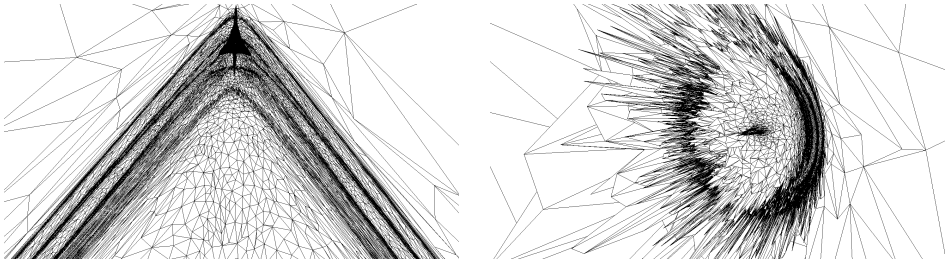


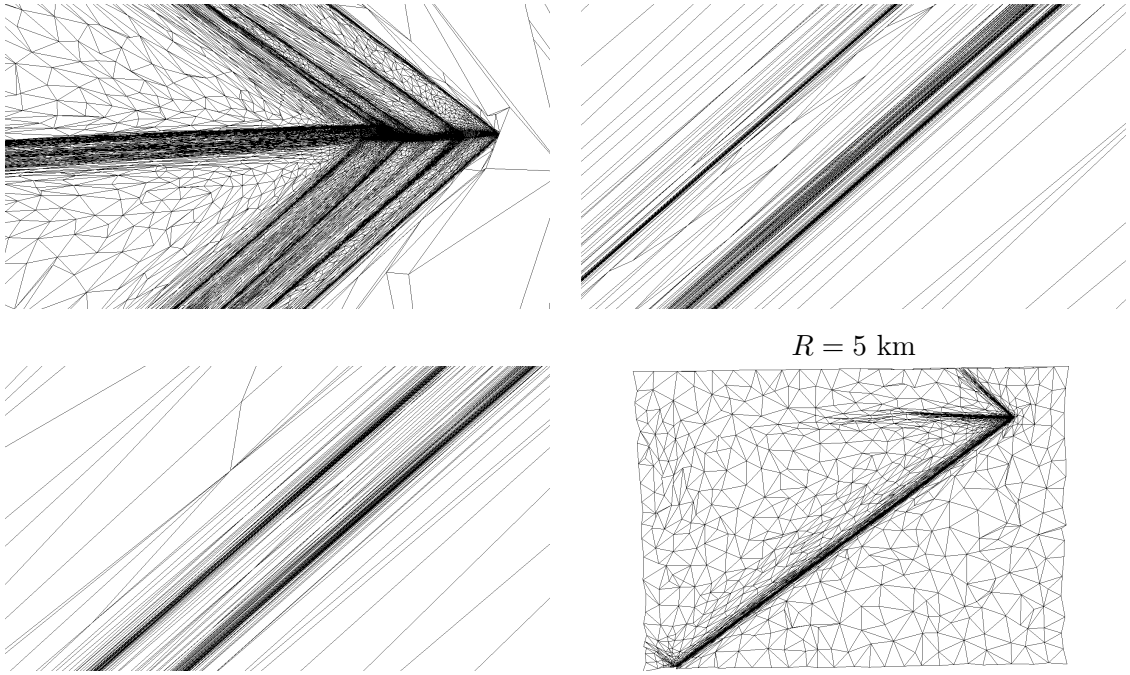
Figure 1.12 – SSBJ example: Left, cut in the final adapted mesh 10 m below the aircraft. Right, cut 10 m behind the aircraft showing how anisotropic tetrahedra are aligned with the Mach cones.

to the sensitivity property of (1.7) given by the local normalization term $\det(|H_R(u_h)|)^{\frac{-1}{2p+3}}$. An example of the scales of the solution is given by the pressure extractions in Fig. 1.14 at $R = 5$ km and $R = 9$ km. Indeed, the normalized pressure signal at $R = 9$ km is of order $8e^{-4}$ while the magnitude is around $3e^{-2}$ at $R = 43$ m. Consequently, we can expect for the volume interpolation error to have a magnitude ratio of $(10^2)^3$. It is then necessary to guarantee that the error estimate detects such small amplitudes even in the presence of large amplitudes. This example demonstrates that using (1.7) complies with this requirement allowing to detect automatically all the scales of the solution, see Fig. 1.14. Several cuts in the symmetry plane are depicted in Fig. 1.13. At $R = 5$ km, we still distinguish 3 separated shocks waves and at $R = 9$ km only two shock waves are separated leading the classical N-wave signature. These features are even more emphasized on the pressure signatures in Fig. 1.14.

The total CPU time is around 28 h 35 mn. 75 % of the CPU time is spent in the flow solver and 35 % in the remeshing, interpolation and error estimate. Note that accurate signatures at $R = 5$ km are already obtained after 11 h of CPU (corresponding to the 20th iteration) as depicted in Fig. 1.14. We give in Table 1.1 the full sequence of CPU times. The first three steps provide an accurate signal for $R/L < 20$ and below.

1.4.4 Boundary layer shock interaction

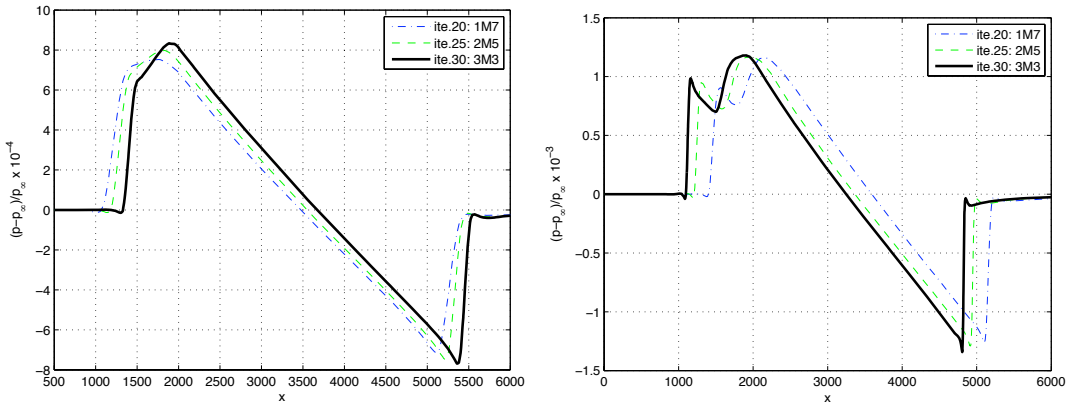
We apply this strategy to study shock/boundary layer interaction. The test case is depicted in Fig. 1.17. The shock waves are generated by a double wedge wing at Mach 1.4 with an



$R = 5$ km

$R = 9$ km

Figure 1.13 – SSBJ example: From top to bottom, from left to rigth, cut in the final anisotropic mesh close to the aircraft, closer view of the mesh 5 km below the aircraft, closer view at 9 km below the aircraft, global view of the mesh.



$R = 9$ km $R = 5$ km

Figure 1.14 – SSBJ example: Left, pressure signature at $R = 9$ km for the final meshes corresponding the last three complexities. Right, pressure signature at $R = 5$ km. The legend reports the number of vertices of each mesh in million (Iteration 20, 25 and 30). The pressure curves are deliberately shifted for visibility.

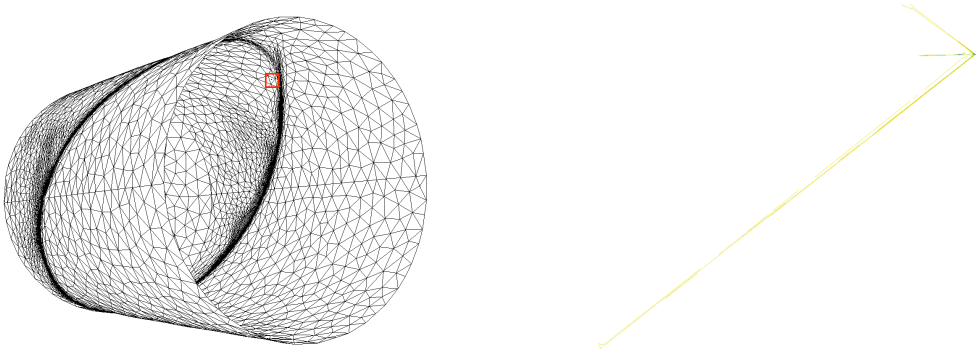


Figure 1.15 – SSBJ example: Left, adapted surface mesh, the red square shows the position of the SSBJ. Right, Mach number iso-lines on the symmetry plane $y = 0$.

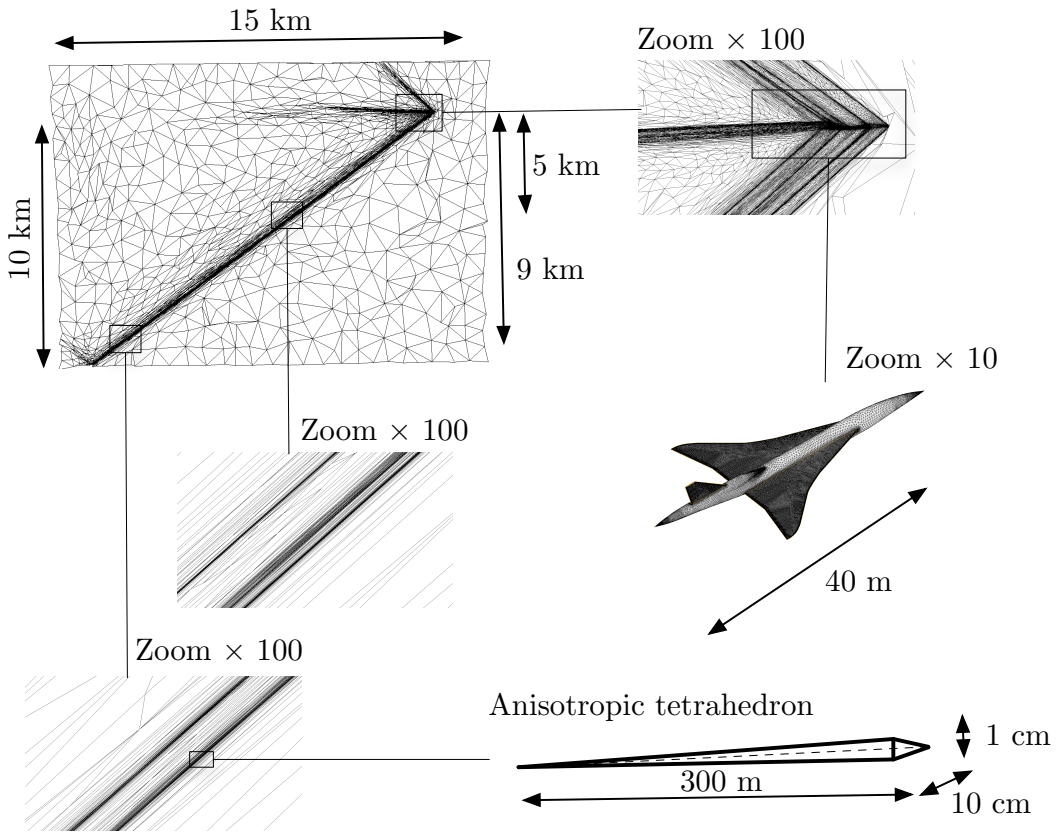


Figure 1.16 – SSBJ example: Example of the scales of anisotropic elements reached in this simulation. An typical anisotropic element in the path of the shock has sizes of the order of 300 m , 10 cm and 1 cm

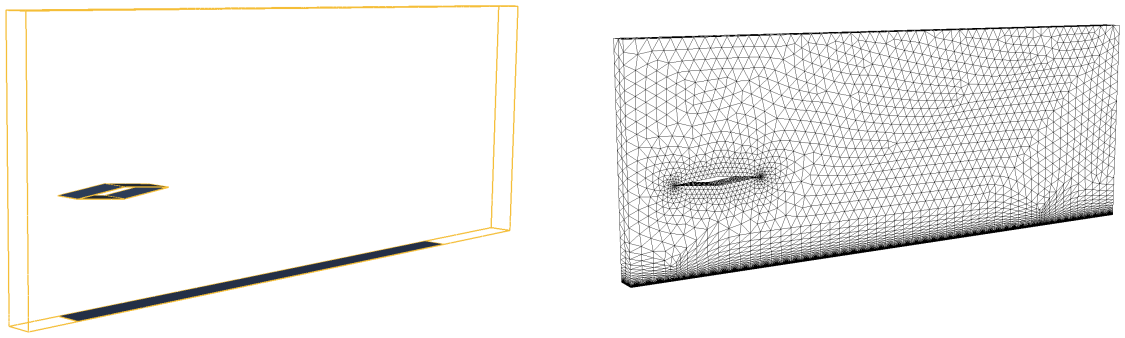


Figure 1.17 – Shock/boundary layer interaction example: from left to right, computational domain and initial surface mesh

angle of attack of 0 degree and a Reynolds number of $3.4\,10^6$. Only the plate is treated as a viscous body. We solve the set of the Reynolds-average Navier-Stokes equations with the Baldwin Lomax turbulence model. The final adapted mesh and the Mach number iso-values are depicted in Fig. 1.17, closer views around the two shocks are depicted in Fig. 1.18. The final mesh is composed of 280 000 vertices and 1.3 million tetrahedra and is obtained after 20 iterations with a complexity of 10 000. In this example, we control the interpolation error on the Mach number coupled with boundary layer metric (1.13). The initial mesh is used as the background mesh to compute (1.13) with parameters $h_0 = 10^{-6}$ and $\alpha = 1.2$. As the boundary-layer metric is intersected with the interpolation error based metric, the resulting complexity is naturally greater. The unstructured boundary layer mesh height is around 10^{-7} near the plate. The average anisotropic ratio is greater than 10^6 and the average ratio is around 500. The worst surface element quality is 33 and 500 for the volume. For the final mesh, the minimal size in the unstructured layer is of the order of 10^{-7}. As shown in Fig. 1.18 (bottom), we successfully capture the typical bubbles and re-circulations at the intersection between the shocks and the boundary layer.

This simulation leads to the following observations. Generating a semi-structured boundary layer mesh extruded from the surface mesh gives only the require accuracy for the smaller layers. Indeed, the distance of the bottom of the shock from the viscous plate is around 10^{-3} whereas the initial height of the uniform boundary layer mesh was at 0.2. Consequently, the example emphasizes the difficulty of capturing these phenomena only with an *a priori* fixed quasi-structured boundary layer mesh. In addition, this approach is completely generic and robust and can handle complex geometries. However, if the shock/boundary layer interaction is automatically handled, the impact of having a fully unstructured mesh is not yet analyzed in terms of solution accuracy and solver stability in the viscous area. Consequently, it also seems interesting to derive a method to generate structured mesh for the smaller layers (at least) while preserving (upper) anisotropic refinements. Metric-orthogonal and metric-aligned anisotropic mesh adaptations are possible solutions to generate highly anisotropic meshes with quasi-structured elements [45, 43].

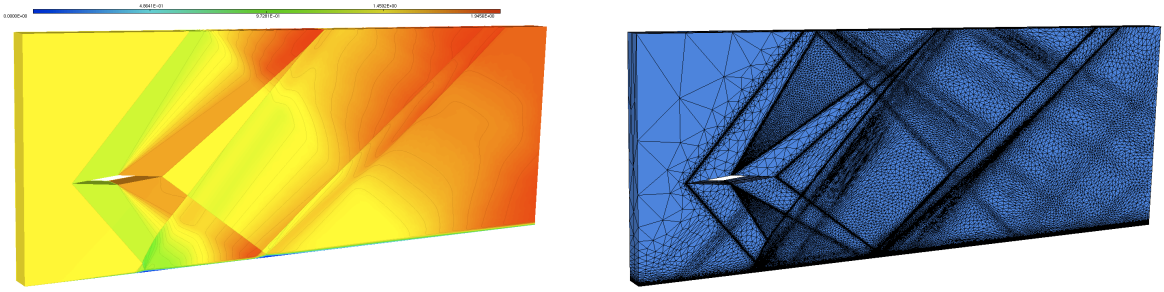


Figure 1.18 – Shock/boundary layer interaction example: from left to right, Mach umber iso values and final adapted surface mesh.

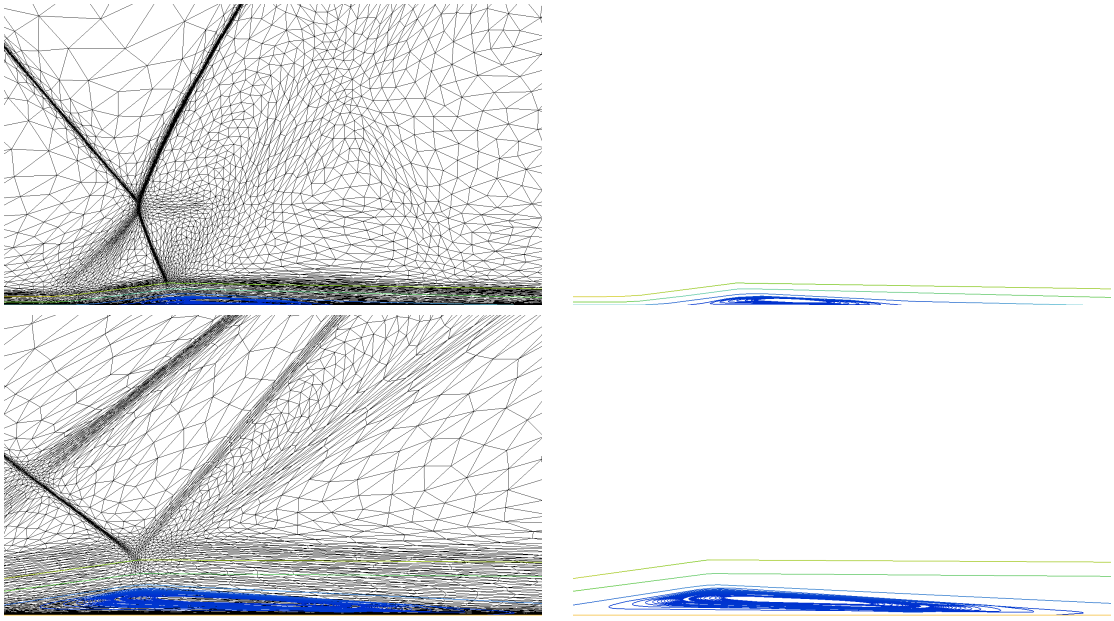


Figure 1.19 – Shock/boundary layer interaction example: Stream lines of the velocity in two bubbles creating from the interaction shock and boundary layer.

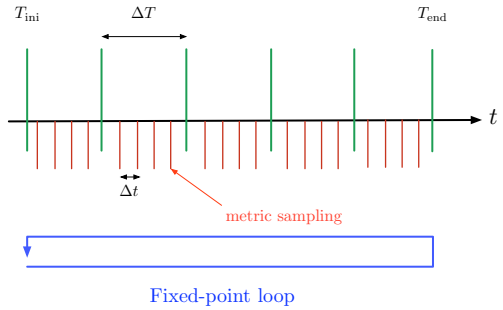


Figure 1.20 – Unsteady adaptation algorithm: a fixed mesh is generated for sub-time window by sampling the solution at different time steps.

1.4.5 Double Mach reflection and blast prediction

The first unsteady case is double Mach reflection. This simulation starts from a 2-state initialization of a shock wave impacting a ramp. The density, speed and pressure for the right side are $(5.71, 9.76, 0, 0, 116.5)$ and $(1, 0, 0, 0, 1)$ for the left side, the shock wave propagates along the x-direction. The total physical time of the simulation is 0.18s. For this simulation, the time frame $[0, 0.18]$ is divided in 30 sub-time windows with 5 fixed point iterations and 21 metric intersections for each sub-time window. We control the L^2 norm of the density interpolation error. The simulation CPU time is 8h55m, 80% is spent in the flow solver and 20% in the mesh adaptation. The final mesh is composed of 235 095 vertices, 1 310 082 tetrahedra and 5 864 boundary faces. The mesh at final time and density iso-values are depicted in Fig. 1.21. We can see that the contact discontinuity is impacting the ramp and that the generated vortices are pushing forward the initial front shock. If this phenomenon is usually observed in 2D simulation [Woodward 1984], its observation on 3D geometry is more complex. Moreover, the thickness of the adaptation is due to the fixed point strategy as the mesh is adapted for all the times step belonging to a sub-time frame.

We then consider a blast propagation on a more complex geometry: the US Capitol. Applying successfully an anisotropic adaptive simulation on it is challenging as it features many complex details as many columns, cupola, ... A classic load is considered, see Fig. 1.22. The final physical time is 0.1 s. The whole time frame has been divided into 20 time slots of 0.005 s. The flow solver outputs density field every 0.0005 s. The final anisotropic mesh for the time frame $[0.05, 0.055]$ is depicted in Fig. 1.22. The interpolation error on the density is controlled in L^2 norm in space and time. The mesh is composed of almost 200 000 vertices for a total CPU time of 8 hours.

1.5 Conclusion

The standard computational pipeline has been described for complex geometries and unstructured mesh generation from CAD to surface and volume mesh generation. For adaptivity, we have described the basic principles of anisotropic mesh adaptation based on metric tensor fields: concept of unit mesh, metric interpolation, metric smoothing, ... A simple to implement but

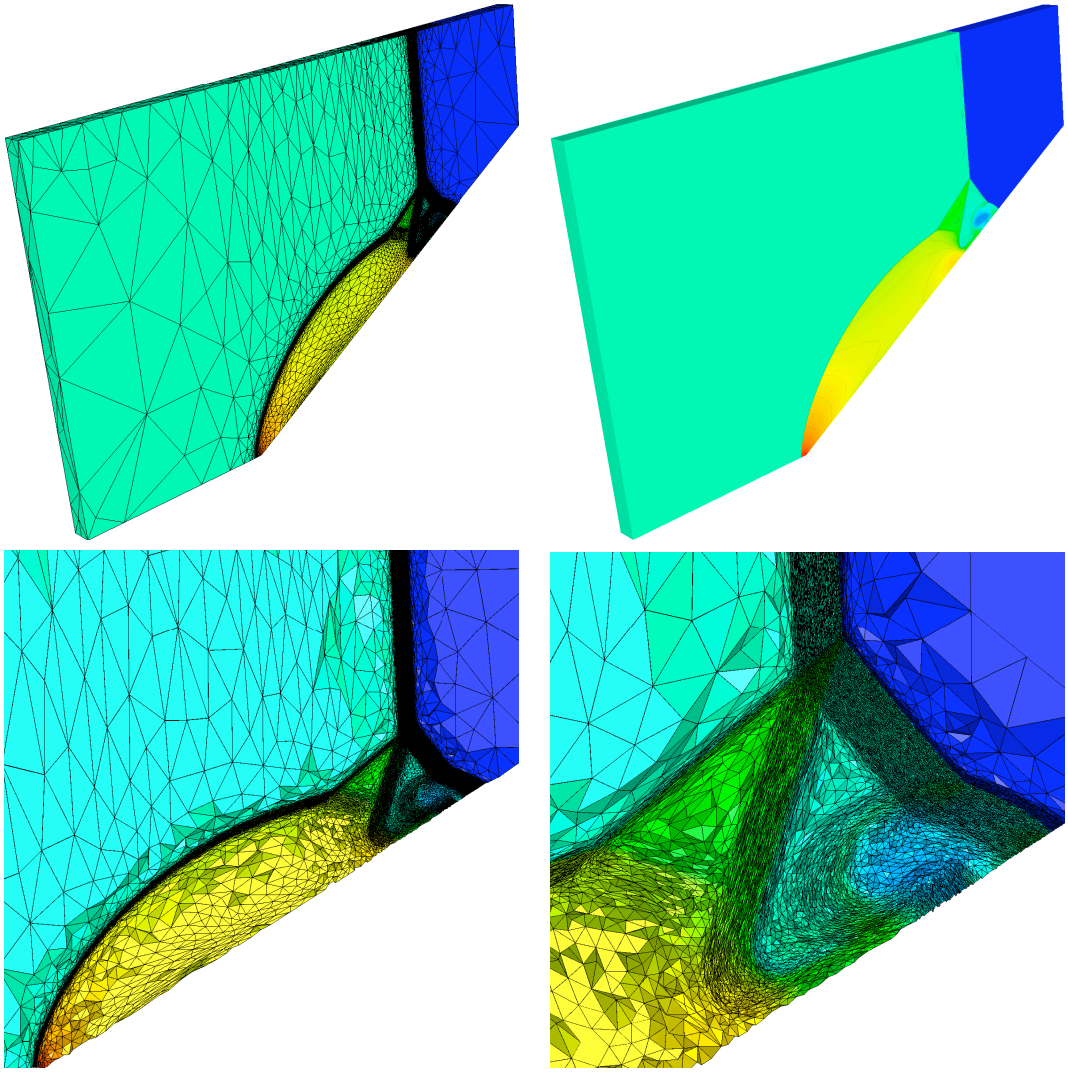


Figure 1.21 – Double Mach reflection at final time : final adapted surface mesh (top left), density iso-values (top right), cut in the volume mesh (bottom left) and closer view near the contact discontinuity and vortex shock interaction (bottom right).

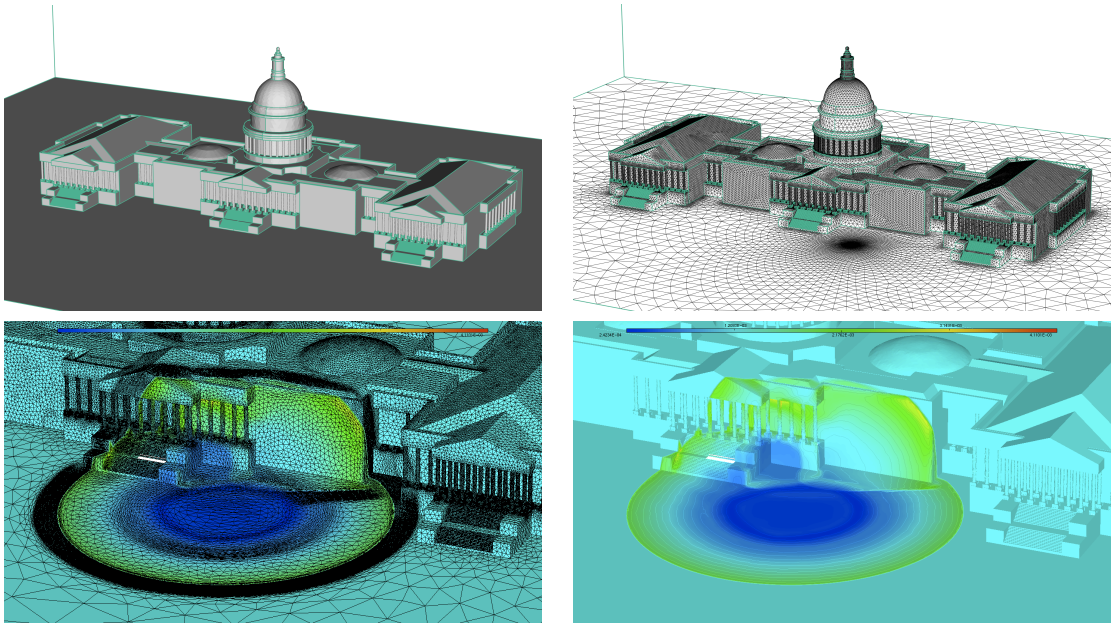


Figure 1.22 – Top, the US capitol CAD (left) and initial surface mesh (right) with the initial blast location. Bottom, anisotropic surface mesh at $t = 0.025$s (left) and the density solution iso-values (right).

robust local remeshing strategy has been detailed. It allows to adapt different components of the flows. For each adaptation, we use a dedicated metric field issued from various estimates: surface curvatures, interpolation errors, distance to a body, ... Numerical examples show the robustness of the method to (i) reduce solver diffusion and (ii) reach a high level of anisotropy that is hardly tractable with a structured approach or a global remeshing method.

This chapter has covered only the basic processes that are required to reach a high level of anisotropy and recover a second order accuracy in space when simulating flows with shocks. It is important to mention that each component is crucial to gain all the benefit of adaptivity. Any improvement in one component may improve the whole process. We refer to [38] for a detailed discussion on the current issues of unstructured mesh adaptation. Mesh generation and adaptation are still active fields of research and many topics are not discussed in this chapter. This concerns the generation of boundary layer grids [Aubry 2009, Bottasso 2002, Garimella 2000], the design of metric-aligned or metric-orthogonal grids [45, 43], the design of very high-order error estimates [Yano 2012], the generation of high-order curved meshes [Abgrall 2014, Sahni 2010, Toulorge 2013, Xie 2013] and parallel (adaptive) mesh generation [Alleaume 2008, Coupez 2000, 9, Ovcharenko 2013, Remacle 2015].

Chapter 2

Unique cavity-based anisotropic framework

When dealing with inviscid flows, anisotropic mesh adaptation is commonly used to automatically get a high solution accuracy with a much lower computational effort than standard methods (uniform meshes, tailored meshes, h-refinement, ...). This gain increases with the level of anisotropy of the flow at hand. However, when viscous flows are involved, several issues reduce the efficiency and the use of unstructured mesh adaptation. The first limitation concerns the adaptation of the surface mesh due to the presence of the boundary layer mesh. Projection to the true geometry may be either unfeasible or inefficient. Then, the creation of the boundary layer mesh itself becomes an even more complex issue when an anisotropic surface is provided. Finally, the transition between the boundary layer and the adaptive mesh becomes of main concern, especially at transonic speeds where strong interactions between shocks and boundary layers exist. Consequently, the baseline adaptive algorithm described in Chapter 1 has several drawbacks when fully detailed geometries and complex viscous flows are considered. As an example, the edge collapse (used to remove a small edge) operator becomes one of the most expensive operators as it is rejected in more than 80% of case in the context anisotropy. It is then mandatory to offer more "global" operators that handles strong anisotropic elements. This is the scope of the unique cavity-based operator that is used at each adaptive step: from surface remeshing to boundary layer extrusion and volume mesh adaptation. For each case, the operator is based on a variety of choices for the initial cavity in order to insert or reinsert a surface or a volume point in an unstructured or quasi-structured fashion. To illustrate the benefits of a unique cavity-based framework, more complex examples are presented involving complex geometries with RANS simulations. The required level of anisotropy is at least two levels of magnitude higher than the examples of Chapter 1. The genesis of the cavity-based framework is inherent to the intrinsic limitations of the state-of-the-art meshing strategies used so far for RANS simulations. We quickly review them in the introduction.

The work on the cavity-based operators and boundary layer mesh generation was initiated in [49, 50, 52] and is part of collaboration with Mississippi State University [36]. It has been then pursued during V. Menier's Phd leading to the publications [44, 47, 48]. Computations for RANS solutions, including periodic boundary conditions has been developed during L. Frazza's Phd with associated publications [3, 5, 28]. The computation for the contrail formation is part of collaboration with ONERA [27, 34]. The validation of this work is part of a long-lasting collaboration within the Unstructured Grid Adaptation Working Group with NASA and The Boeing Company [30, 35, 38, 40].

2.1 An introduction to standard boundary layer meshing techniques and adaptivity

Despite the increasing computing power and new developments with Large Eddy Simulations (LES) simulations [Sagaut 2001], there is still a need for "cheaper" Reynolds-Average Navier-Stokes (RANS) simulations. The complexity of such flows are depicted in Fig. 2.1, showing interaction between anisotropic components (shocks wave) and viscous boundary layers. Numerical computations with viscous flows are usually based on so-called boundary layer (BL) meshes. Most of second-order unstructured CFD numerical schemes need a boundary layer (BL) mesh to accurately approximate the speed profile around a body during a viscous simulation (see, e.g. [Löhner 1993, Pirzadeh 1994]). When generating a BL mesh, the first difficulty is to deal with the very high aspect ratios of the elements ($O(10^3 - 10^5)$) as the width of the boundary layer depends on the local Reynolds number [Löhner 2001]. So far, the generation of BL has been carried out by an extrusion of the initial surface along the normals to the surface [Bottasso 2002, Löhner 1993, Löhner 1999, Pirzadeh 1994] or by local modification of the mesh [Marcum 1996]. Using the normals as sole information requires several enrichments to obtain a smooth layers transition on complex surfaces [Aubry 2009, Garimella 2000, Ito 2002, Ito 2006]. These techniques rely on a complex pre-processing of the surface in order to extract additional geometric information as convex or concave ridges. The last but not least difficulty arises once the BL process is finished. Indeed, in most methods, a global mesh generation process is used to close the volume. Both Delaunay method or frontal methods are susceptible to fail in presence of anisotropic faces. In the aforementioned methods, the BL mesh generation is performed in a unique pre-processing procedure/step that is done prior to any computation. Consequently, achieving a full coupling between RANS mesh generation and mesh adaptation is tedious and remains a challenge if standard techniques are used. Simpler strategies have been devised as keeping the BL layer mesh. However, in many applications adapting the surface is mandatory to keep an accurate result. As a result, the way classical BL mesh generation has been designed so far requires major changes in order to be used together with anisotropic mesh adaptation.

One solution is to design an operator than handles conjointly quasi-structured mesh generation and anisotropic mesh generation. This is the scope of the unique cavity-based operator.

Outline. We first introduce the cavity-based framework along with a simple validity principle. Extensions of standard operators (insertion, collapse, swaps) are detailed. We then describe the optimized adaptive mesh generation process based on this framework. The case of boundary layer mesh generation is then described by using a constrained version of the operator. Finally, several numerical simulation with fixed boundary layer or fully adapted boundary layers are presented.

2.2 Cavity-based operators

A complete mesh generation or mesh adaptation process usually requires a large number of operators: Delaunay point insertion, edge-face-element point insertion, edge collapse, point smoothing, face/edge swaps, etc. Independently of the complexity of the geometry, the more

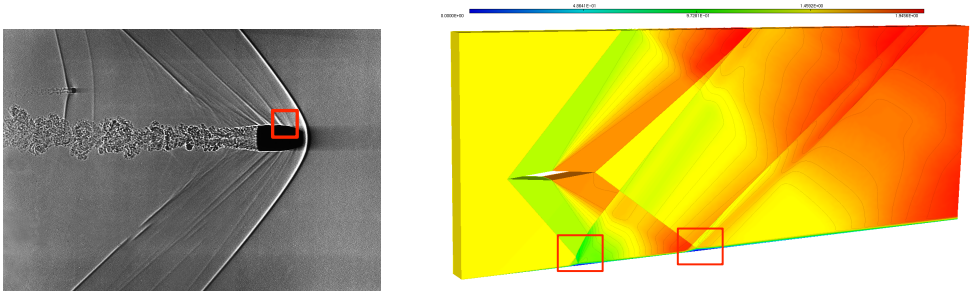


Figure 2.1 – *A supersonic viscous bullet (left) from NASA media library and adaptive shock/boundary layer interaction (rigth) excerpt from [52]. Both examples contain interaction between the viscous boundary layer and anisotropic phenomena (red squares).*

operators are involved in a remeshing process, the less robust the process may become. Consequently, the multiplication of operators implies additional difficulties in maintaining, improving and parallelizing a code. The unique operator is then an additional way to help maintain and improve globally an adaptive mesh generation process. We describe in this section its derivation, validity and how it naturally extends traditional standard local operators. We also focus on the case where a surface point needs to be inserted or move to the real surface position.

2.2.1 Extension of standard operators

The cavity-based operator is inspired from incremental Delaunay method [Bowyer 1981] and [Watson 1981, Hermeline 1982] where the current mesh $\mathcal{H}_k$ is modified iteratively through sequences of point insertion. The insertion of a point P can be written:

$$\mathcal{H}_{k+1} = \mathcal{H}_k - \mathcal{C}_P + \mathcal{B}_P, \tag{2.1}$$

where, for the Delaunay insertion, the cavity $\mathcal{C}_P$ is the set of elements of $\mathcal{H}_k$ such that P is contained in their circumsphere and $\mathcal{B}_P$ is the ball of P, i.e., the set of new elements having P as vertex. These elements are created by connecting P to the set of the boundary faces of $\mathcal{C}_P$. We assume that $\mathcal{H}_k$ and $\mathcal{H}_{k+1}$ are 3D simplicial meshes, *i.e.*, composed only of tetrahedra and triangles. These meshes are said to be **valid** if all tetrahedra have a positive volume. note that if a hybrid mesh is given as input, it is pre-processed to be decomposed into a simplicial mesh following the procedure given in [Dompierre 1999]. In what follows, capital letters as A, B, ..., P denote points of $\mathbb{R}^3$, except K that usually denotes an element (triangle or tetrahedron). In addition to $\mathcal{B}_P$, the ball of P, *i.e.,* the list of elements surrounding P (having P as vertex), we also consider $\mathcal{S}_{AB}$, the shell of edge AB, *i.e.,* the list of elements having A and B as vertices. Topological entities as balls and shells are computed on the fly by using the elements storage data structure. Consequently, we assume that for each tetrahedron, we know the neighboring tetrahedra (or boundary faces) seen across the 4 faces. We refer to [George 1998] for more details. In the cavity-based framework, the cavity is just a set of tetrahedra and is no longer

related to any Delaunay criteria. Instead, we assume that $\mathcal{C}_p$ has the following property:

$$\begin{gathered}\text{Given any two tetrahedra } K_1 \text{ and} K_2 \text{ in } \mathcal{C}_p,\\ \text{there exists a path trough faces' of tetrahedra in } \mathcal{C}_p\\ \text{that links } K_1 \text{ and } K_2.\end{gathered} \tag{2.2}$$

Note that this property holds for the balls of vertices and the shells of edges. We denote by $\overline{\mathcal{C}}_P$ the external faces of $\mathcal{C}_P$. It is composed of boundary faces (triangles) and internal faces. For each internal face $\overline{\mathcal{C}}_P$, the neighboring tetrahedron viewed from this face is not contained in $\mathcal{C}_P$. Given an oriented face $[A, B, C]$ and a normal $\mathbf{n}$, the visibility of $\mathbf{n}$ with respect to $[A, B, C]$ is the dot product between $\mathbf{n}$ and the normal to the face. In a similar way, a point P is visible for an oriented face $[A, B, C]$ if the volume of the tetrahedron $[P, A, B, C]$ is positive. We now state the fundamental property: Given a **valid** mesh $\mathcal{H}$, a point P and a set $\mathcal{C}_P$, if (2.2) holds for $\mathcal{C}_P$ and P is visible for all faces of $\overline{\mathcal{C}_P}$ then the mesh given by $\mathcal{H} - \mathcal{C}_P + \mathcal{B}_P$ is **valid**. The local mesh modification operators derived in this chapter are based on this property.

The main idea of the cavity definition consists in recasting each meshing operator as a node (re)insertion. The initial choice of the cavity defines the underlying operator. For each operator, we just have to define judiciously which node P to (re)insert and which set of volume and surface elements will form the cavity $\mathcal{C}$ where point P will be reconnected with $\mathcal{R}_P$:

$$\mathcal{H}_{k+1} = \mathcal{H}_k - \mathcal{C}_P + \mathcal{R}_P. \tag{2.3}$$

When a point is reinserted, a subset of its ball is reconnected or re-created. We then prefer the notation $\mathcal{R}_P$ rather to $\mathcal{B}_P$. Similarly, the cavity is no more only related to the point being inserted as in the Delaunay point insertion method. According to the previous validity principle, if $\mathcal{H}_k$ is a valid mesh (only composed of elements of positive volume) then $\mathcal{H}_{k+1}$ will be valid if and only if $\mathcal{C}_P$ is connected (through internal faces of tetrahedron) and $\mathcal{R}_P$ generates only valid elements. In Fig. 2.2, we list the initial cavity choice along with the point to (re)insert for the collapse, insertion and swap. As it, the cavity operators are equivalent to their standard counterparts. However, using the cavity formalism allows to easily modify the cavity to enforce automatically the operator. The cavity enlargement correction is one example of such correction and is given in Algorithm 1. The basic idea is to enlarge the cavity to make sure that $\mathcal{C}_P$ becomes valid. To illustrate this feature, we consider a simple 2D example where we want to relocate a point A to a new position A_{new}, see Fig. 2.3. Given the initial configuration, we see that a collapse, then a swap and finally a point-smoothing is needed to actually move A to A_{new}. To do this, 4 volumes are computed for the collapse, 2 for the swap and finally 7 for the point-smoothing. Then, if we use the cavity version, the initial cavity has 2 negative faces (in red in Fig. 2.3, bottom). Using the cavity enlargement Algorithm 1, a valid cavity is found in 3 enlargement iterations. To build the final $\mathcal{C}_P$, 4 volumes are computed with the initial cavity, 4 for the first iterations, 2 for the second and 2 for the third. The cost of using the cavity moving is 12 volumes computations whereas 13 volumes are needed with the standard operators. The most interesting feature is that the cavity operator creates automatically the combination of simple operators without the need to know in practice the sequence. From a practical point of view, only one operator is used for the meshing operators. This feature is particularly interesting

when the initial mesh has strong anisotropic components near the boundary surface mesh. In this case, the cavity operator is used to enforce the insertion of a surface independently of the density and configuration of the given initial mesh.

The use of the previous cavity-based operators allows us to design a remeshing algorithm that has a linear complexity in time with respect to the required work (the sum of the number of collapses and insertions). On a typical laptop computer Intel Core I7 at 2.7 GHz, the speed for the (cavity-based) collapse is around 20 000 points removed per second and the speed for the insertion is also around 20 000 points or equivalently 120 000 elements inserted per second. Both estimates hold in an anisotropic context [45].

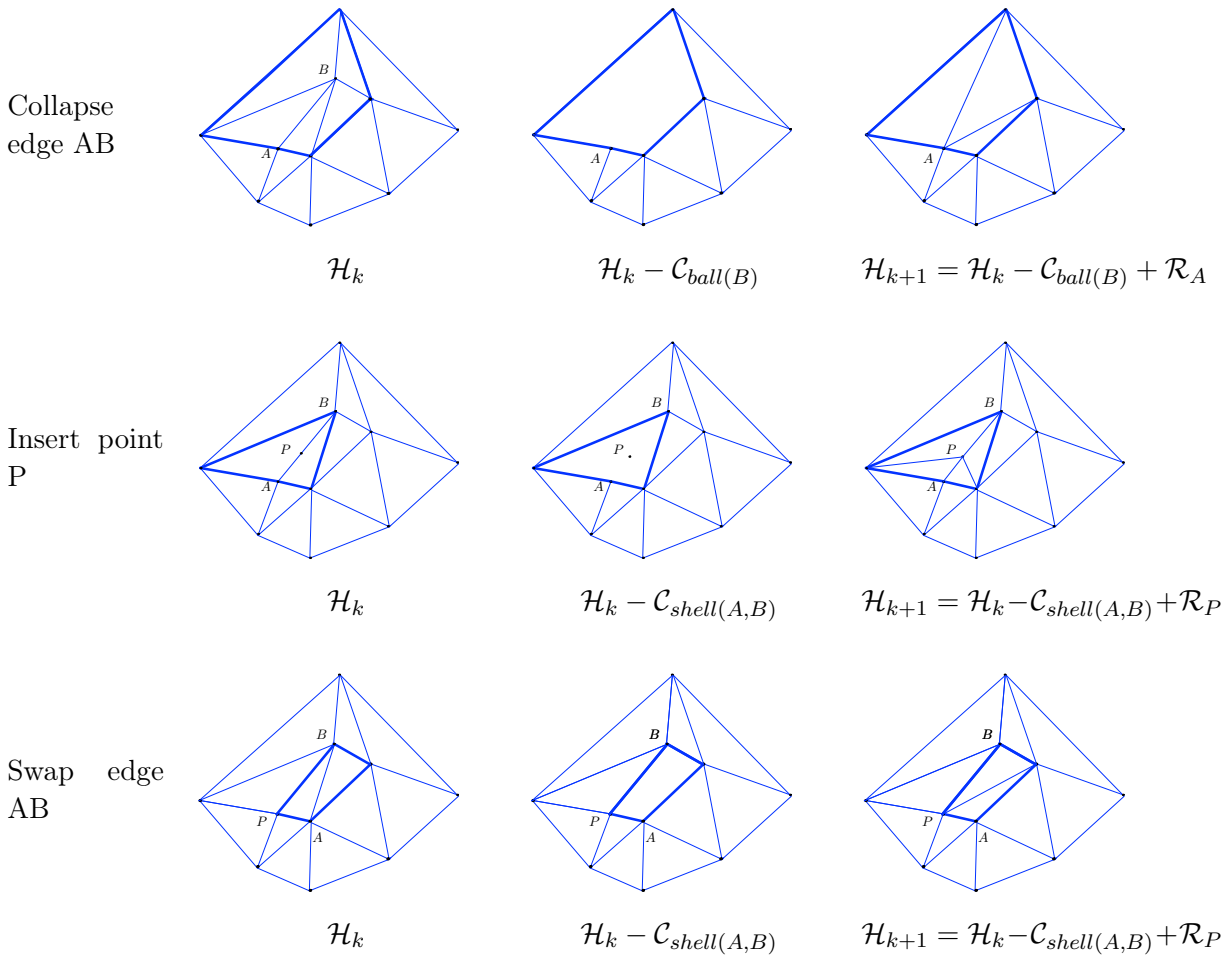


Figure 2.2 – Three 2D meshing operators reinterpreted as a cavity-based operator with an appropriate choice of the point to be (re)inserted and cavity to be remeshed. From top to bottom, the collapse, insertion and swap operators.

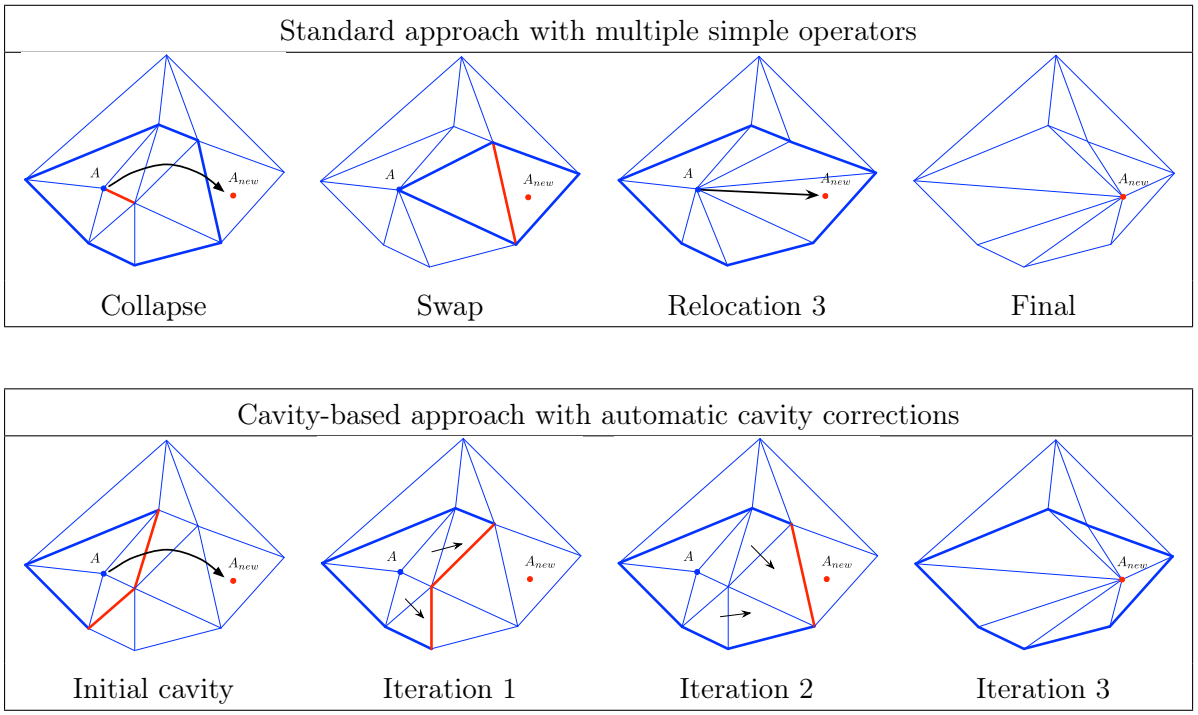


Figure 2.3 – Illustration of the relocation of point A to a new position A_{new}. Top, if standard operators are used, the following sequence has to be applied: collapse, swap, relocation. Bottom, with the cavity enlargement, 3 enlargement iterations are needed to perform the operation.

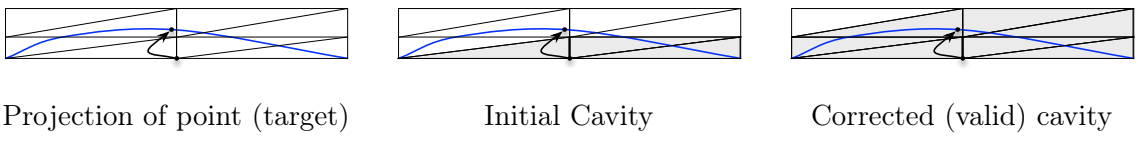


Figure 2.4 – Illustration of surface point insertion (blue curve) where the standard operator (middle) is rejected whereas the cavity correction allows to insert the point (right).

Volume Part:
For each K in $\mathcal{C}_P$

For each face $[A, B, C]$ such that $P \notin [A, B, C]$:

if $volume(A, B, C, P) < 0$, then

if P is a surface point then **reject**

else add neighboring tetrahedron to $\mathcal{C}_P$

endif

endif

EndFor

EndFor
if $\mathcal{C}_P$ is modified goto **Volume Part**.

Algorithm 1: Cavity enlargement for (re)insertion of P

2.3 Optimized unit mesh generation

The scope of this step is to obtain a mesh where the lengths of the edges are in the interval $[\frac{1}{\sqrt{2}}, \sqrt{2}]$. Instead of repeatedly doing insert and collapse as in Chapter 1, this procedure is composed of 3 fixed phases: collapse, creation of new points, anisotropic filtering and insertion. The anisotropic filtering is then added to make sure that a new insertion will not create the need for a future collapse.

2.3.1 Collapse

For this phase, an iterative procedure is used. The current mesh is iteratively scanned and while there exists an edge with a length lower than $1/\sqrt{2}$, we try to collapse this edge with the cavity-based collapse. At the end of the process, all edges must have a length greater than $1/\sqrt{2}$. During all the following phases, the collapse is never used again.

2.3.2 Creation of edges

In this phase, we create the set of points that would be needed to decompose all long edges in segments having a length close to one in the metric. As for the collapse, the algorithm consists in scanning the current mesh and while there exists an edge with a length greater than $\sqrt{2}$, create one or multiple points. During this phase, the topology of the mesh is kept unchanged so that the points are not inserted. Indeed, neighboring edges can generate similar points or points very close to each other, so it is important to filter out the points that are too close (in the metric). For that, we define the anisotropic filtering.

2.3.3 Anisotropic filtering and insertion

In this phase, the length between the points created in the previous phase is checked and only a subset of points is inserted. For the filtering, we use an octree of points. Each octant can contain up to 10 points before being subdivided. Initially, the octree contains the surface points and the volume points remaining from the collapse phase. To validate the insertion of a point, we first check the distances between all points that are in the octant containing the point to be inserted. If no rejection occurs, then the current octant is intersected with the bounding box of the metric. All the intersected octants are checked starting from the octants closer to the point being inserted. Then, each point that is accepted for insertion is inserted in the octree together with its metric. At the end of the filtering, whatever the connectivity generated by the insertor the edges will have an admissible length (as the length was checked in every direction with the octree). This property prevents us from having to perform additional collapses that is the most costly operator.

The standard initial cavity-based insertion is based on an Delaunay cavity criterion that is modified to comply with the size of the metric field. It is composed of any element K verifying:

$$\alpha_{\mathcal{M}}(P,K) = \frac{\|OP\|_{\mathcal{M}}}{(r_K)_{\mathcal{M}}} < 1, \tag{2.4}$$

where P is the point being inserted, O the center of the circumcenter of K computed in $\mathcal{M}$ and r_K the radius computed in $\mathcal{M}$. When $\mathcal{M}$ is varying, Equation (2.4) is not straightforward to compute, so we use the modified Delaunay criterion defined in [Dobrzynski 2008] instead. It relies only on point-wise metric evaluation:

$$\begin{cases} \alpha_{\mathcal{M}(P)}(P,K) < 1, \\ \sum_{i=1}^{4} \alpha_{\mathcal{M}(P_i)}(P,K) + \alpha_{\mathcal{M}(P)}(P,K) < 5,\ P_i \in K. \end{cases}$$

The previous operator prevents the suppression of edges/faces having an admissible size in $\mathcal{M}$ while having bad angle for the standard Delaunay. Consequently, the creation of slivers is not handled by this operator. We add the additional control (1.16) on the height defined in Chapter 1 to avoid their creation. The idea consists in controlling the height of the tetrahedron in addition to its volume. Once the anisotropic cavity is computed, the external faces of the cavity leading to a valid tetrahedron (positive volume) are checked to verify that their heights are greater than the minimal possible height given by the metric $\mathcal{M}(P)$. The cavity is then reduced to remove negative volume elements and elements that do not verify (1.16). The previous formula simply states that the worst height of a unit tetrahedron is found when the height vector is aligned with the eigenvector of minimal size. This modification reduces the number of slivers and also reduces the amount of optimization (swaps).

2.3.4 Optimization of the mesh

During this phase, only the topology of the mesh is modified by using edges or faces swaps, see [Frey 2008] for the details of these operators. The only constraint is to make sure that the quality in the metric is strictly improved at each application of a swap.

2.3.5 Surface approximation

During the generation of the adapted surface mesh, two components play a crucial role for the quality of the final generated mesh. The first one is related to the surface approximation, it is necessary to maintain a sufficient level of fidelity of the geometry. To do so, two different options, a discrete and a continuous one, are used to control the geometry approximation. The second component is related to the insertion of a surface point with a volume attached to it.

For the discrete approach, a fine and fixed discrete mesh is used as a background support. The surface points along with their normals at the points are computed using this support. In addition, a surface-based metric is recovered and intersected with the current computational metric in order to control the required level of fidelity. We refer to [57] for a detailed description of the process. When a continuous description of the geometry is provided, as a CAD geometry, the newly created surface points are projected onto the continuous position by querying the CAD. In what follows, we use the discrete approach for all the numerical examples where the surface is approximated by a 3rd order (curved) mesh.

2.4 A constrained version of the operator

We consider in this section the generation of a boundary layer mesh starting from a given surface mesh. For simplicity, we focus only on the generation of the volume mesh meaning that the surface mesh that supports the boundary layer is kept constant during the whole process. No hypothesis is made on the features of the surface mesh. In particular, the process should handle anisotropic surface meshes. The main modification consists in defining a constrained version of the cavity operator. We then exemplify specific choices of $\mathcal{C}_P$ leading to the generation of quasi-structured elements.

2.4.1 Boundary layer mesh generation by point insertion

The cavity inserter is now turned into a constrained point inserter. We detail the main modifications. When P_{new} is inserted along a normal, the previously created elements that are in the boundary layer mesh should be kept. Consequently, a set $\mathcal{K}$ of (constrained) tetrahedra in the boundary layer is created and updated after each insertion. The cavity enlargement procedure of Algorithm 1 can exit if during the process a constrained element is added to $\mathcal{C}_P$. The cavity initialization is also modified to remove from $\mathcal{B}_P$ elements that belong to $\mathcal{K}$. In order to ensure (2.2), it is necessary to verify that $\mathcal{B}_P$ minus the constrained elements is still connected (to verify the validly principle). From a technical point of view, as P_{new} is not initially (topologically) present in the mesh (contrary to the case where P is moved), the main difficulty is related to the surface part of Algorithm 1. Indeed, every boundary face [A,B,C] will never contain P_{new}, so additional information is required to derive each connected component type. We can now state the main result for the case where the point is extruded along one normal only (mono-normal):

$$\text{In a mono-normal context, previous inserter with initialization } \mathcal{C}_p = \mathcal{B}_P - \mathcal{K}, \quad \text{automatically generates quasi-structured elements.} \tag{2.5}$$

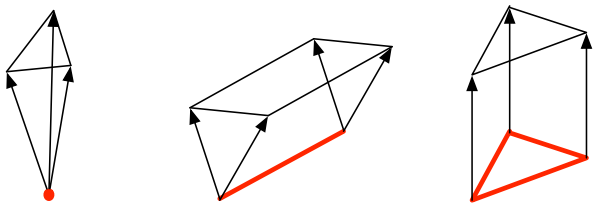


Figure 2.5 – Depending on the configuration of normals, different kinds of elements (tetrahedron or prism) are recovered: vertex-based (left), edge-based (middle) and face-based (right).

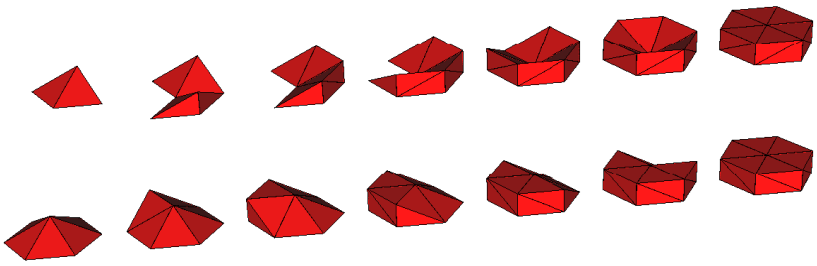


Figure 2.6 – *Two examples of the automatic process of prisms creation around a point. The final decomposition of prisms (in tetrahedra) depends on the order of the insertion of points.*

We first remark that if P_{new} is inserted along a normal direction issued from P, the final mesh will contain the edge PP_{new}. Then if face $[A, B, C]$ belongs to $\overline{\mathcal{C}}_P$, then tetrahedron $[A, B, C, P_{new}]$ is created. Consequently, given a face $[A, B, C]$ with the extruded vertices $[A_{new}, B_{new}, C_{new}]$, insertion of A_{new} will create tetrahedron $K_1 = [A, B, C, A_{new}]$, insertion of B_{new} will create $K_2 = [A_{new}, B_{new}, B, C]$ and insertion of C_{new} will create $K_3 = [C_{new}, A_{new}, B_{new}, C]$. The union of K_1, K_2 and K_3 forms the prism $[A, B, C, A_{new}, B_{new}, C_{new}]$. Note that K_1, K_2 and K_3 should be added to $\mathcal{K}$ after each insertion. The update of $\mathcal{K}$ is based on the different sets of hybrid elements than can be created, see Fig. 2.5. We illustrate in Fig. 2.6, different prisms construction around the ball of a point. Note that changing the order of insertion of the extruded points will lead to different decompositions of the prismatic mesh. Consequently, the point insertion can be *a priori* optimized in order to favor the creation of the smallest diagonal edges when a quadrilateral face of a prism is decomposed. In addition, according to the current configuration, the remaining points cannot be inserted in any order as this may lead to an invalid decomposition (known as the Schönhardt's prism). It is thus necessary to loop over the remaining points in order to solve this issue. Usually, no more than 5 iterations are required to insert all the points.

2.4.2 Possible enhancements with multi-normals and merge

Two main drawbacks of using only one normal per point arise at closed and opened ridges. At closed ridges, the normals may cross, leading to invalid elements. At opened ridges, the quality of the elements decreases as the deviation between the normals may be large. A common practice to solve this issue is to smooth the normals [Aubry 2009]. In our approach, we can choose different initialization of $\mathcal{C}_P$ in order to improve the boundary layer mesh quality.

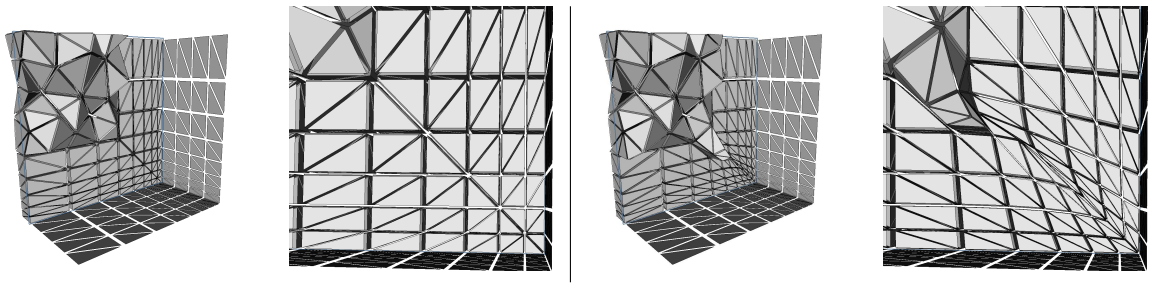


Figure 2.7 – *Example of extruding a boundary layer from a closed ridge. The merge of normals (left) allows to create regular elements (hexahedra) whereas smoothing of normals is required (right) to avoid normal crossings.*

In order to reduce the number of invalid elements, we pre-process the given normals and current size and predict the volume of elements. For each invalid element found, we attempt to collapse the normals until no more negative element is found. For a face, if three normals are merged, a tetrahedron is created, whereas a pyramid is created when two normals are merged. Normals are merged according to a minimal distance criterion. If we denote by $(P_i)_{i=1,k}$ the list of points associated with a given list of merged normals, the cavity is then initialized by $\bigcup \mathcal{B}_{P_i} - \mathcal{K}$. The standard inserter is then called with this initialization. We illustrate this operator on a simple cube geometry where 2 faces support the boundary layer mesh. Theses surfaces are adapted to follow the normal size distribution. Consequently, without normal smoothing, normals cross each other at each step. We can see that the quality of the generated boundary layer mesh is improved with the merge of normals, see Fig. 2.7.
The multi-normal case is the most complicated. The way the cavity $\mathcal{C}_P$ is initialized is crucial to ensure the desired connectivity. Given P and a (minimal) set of normals, we start from the list of the (interface) faces that surround P. Interface faces are the boundary triangles for the first layer and then become the internal faces defining the frontier between the previous and current layers. Note that these faces are part of $\overline{\mathcal{C}}_P$. We then assign each face to a normal by trying to maximize the visibility criterium. In addition, for a given normal, the faces of the list should be adjacent by edges. If two many normals are given, remaining normals with no more faces are not inserted. For a normal $\mathbf{n}$ associated to the list of faces $L = (F_i)_i$, the cavity is initialized by $\mathcal{C}_P = \bigcup_{AB \in F_i, F_j} \mathcal{S}_{AB}$, where AB is an edge shared by two faces of L. The inserter is then called normally with this initial choice of $\mathcal{C}_P$. We illustrate this procedure on a simple opened ridge. Fully structured elements are automatically created as in the merged case, see Fig. 2.8.
The main difficulty when using merged and multi-normals consists in the recovery of constrained elements in order to update $\mathcal{K}$. In the mono-normal context, only face prisms are recovered whereas in this case, constrained elements can also be point-based and edge-based, see Fig. 2.5. Algorithm 2 summarizes the complete process.

2.4.3 Boundary layer examples

We exemplify the use of this operator on different complex geometries: an ONERA M6 wing, a shuttle, a missile and a landing gear. The CPU time for the missile to insert a layer is around 10 sec (600 000 prisms / layer), see Fig. 2.9. For the landing gear geometry, the CPU time to insert

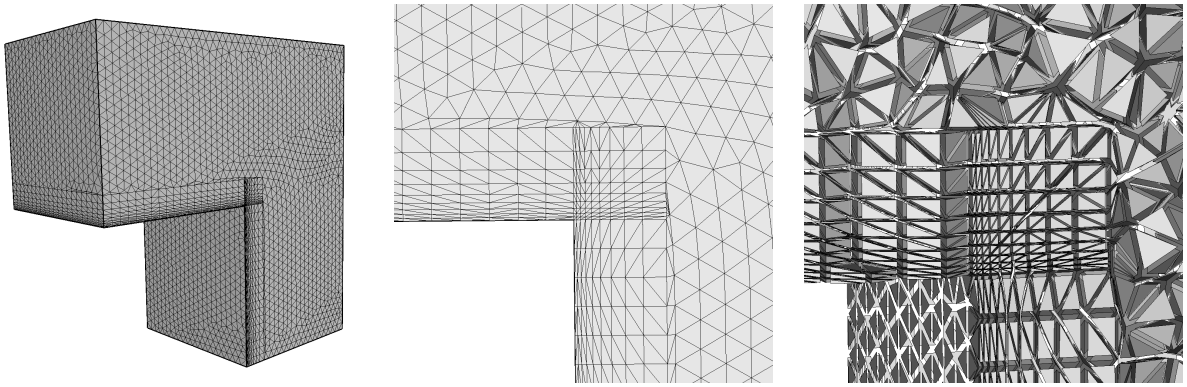


Figure 2.8 – *Example of extruding a boundary layer from an opened ridge. The use of multi-normals allows to create regular elements (hexahedra): global and closer views around the ridge of the surface mesh (left and middle), view in the volume mesh (right).*

While (1)

1. Recover the interface surface mesh (between current layer and previous layer)
2. Compute normals and multi-normals
3. Fictive extrusion of the boundary layer : optimize the layer with the merge of normals
4. Insert extruded points in the following order:
   - along merged normals
   - along multi-normals
   - along mono-normals to close the boundary layer volume
5. Optimize the current layer: diagonal swapping and point smoothing

EndWhile

Algorithm 2: Boundary layer mesh generation

a layer spans from 8 sec to 35 sec when all the points are inserted (e.g. 800 000 prisms / layer), see Fig. 2.10. For 25 layers, the total CPU time is around 5 min. Note that the remaining volume mesh between the boundary layer mesh and the outer surface of the domain is not optimized either in quality or in size. This is done in a different optimization step allowing to adapt the mesh with respect to a given anisotropic metric field or to a uniformly graded metric. Consequently, we can see that the process can insert a boundary layer mesh in an already high density mesh, see Fig. 2.10, but also in a very coarse mesh, see Fig. 2.9.

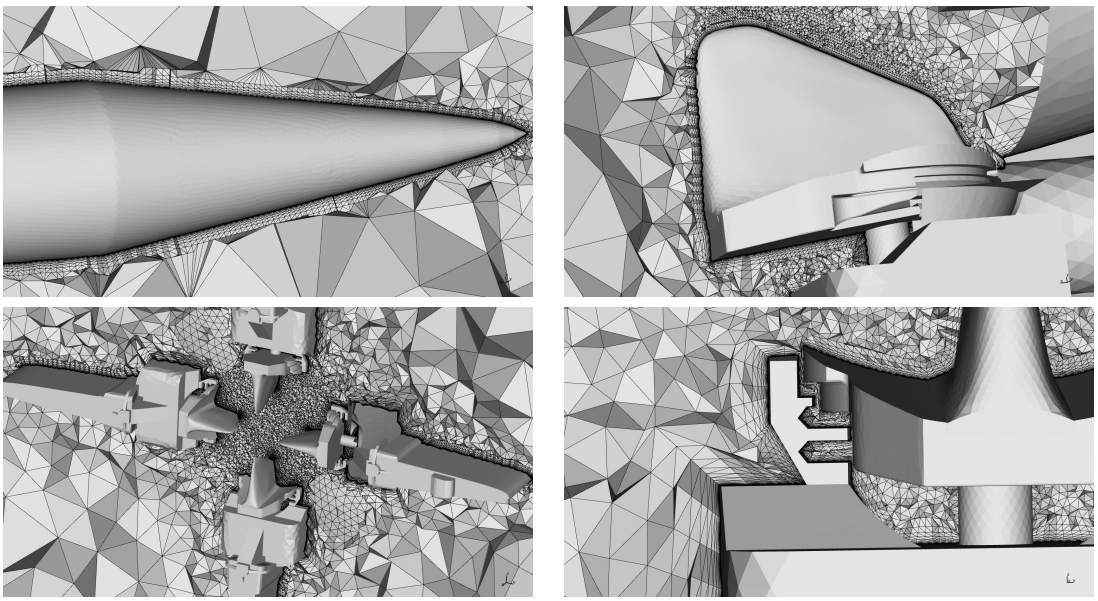


Figure 2.9 – *Boundary layer mesh generation around a complex missile geometry starting from a coarse initial volume mesh.*

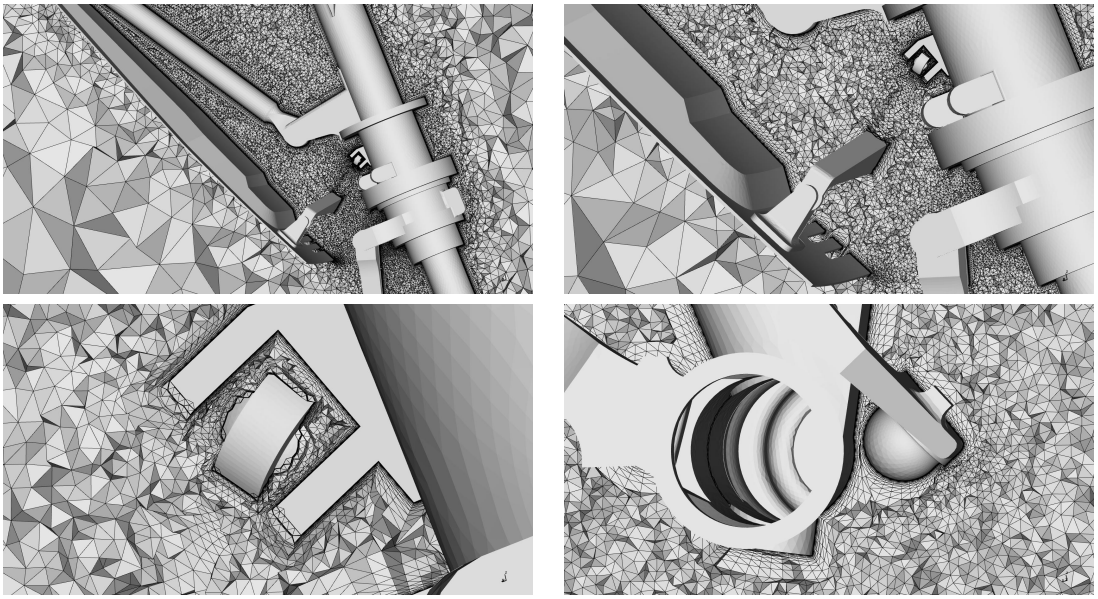


Figure 2.10 – *Boundary layer mesh generation around a landing gear geometry.*

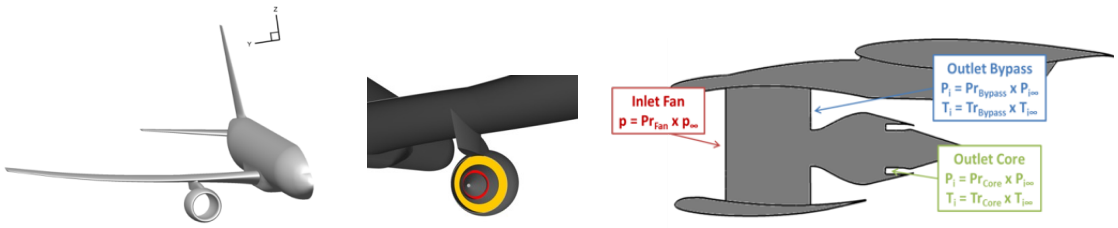


Figure 2.11 – CRM geometry and engine design with outlet bypass and outlet core.

2.5 Numerical examples

We describe in the section several complex numerical examples. They are all based on 3D cases when solving the RANS equations.

2.5.1 Contrail formation

This example investigates contrail formation in the near field of an aircraft. A complete description and physical findings of the test case are documented in [34]. Contrail formation is a complex topic, since several physical processes are involved, covering a large range of space and time scales, from the engine exit to the atmospheric global scale. We focus here on the mesh adaptation procedure that is used to observe the plume 1km after the aircraft. Three-dimensional Reynolds-Averaged Navier-Stokes (RANS) simulations of contrails produced by the Common Research Model wing/body/engine configuration during cruise flights is performed. The `Cedre` flow solver of ONERA is used. In the present work, a dedicated internal nozzle geometry has been designed to replace the through flow nacelle of the original CRM configuration, see Fig. 2.11. The initial atmospheric conditions were chosen so that a contrail was expected to appear. The ambient temperature was set at 223 K and pressure at 264 hPa to simulate cruise conditions at an altitude of about 34 000 ft. The mesh is composed of a boundary layer mesh (generated with the constrained operator) while the outer field is adapted on the cross flow velocity in L^2 norm. Five adaptative steps were performed leading to sizes with 6 701 472, 4 140 630, 4 514 154, 5 380 312, 7 219 502 tetrahedra. The boundary layer is kept constant during the refinement and it is composed of 13 143 807 prisms and 95 668 pyramids. We observe that the final adapted mesh contains almost the same number of tetrahedra that the initial one. The temperature and mesh distribution is depicted in a plane located at 8 spans behind the aircraft, see Fig. 2.12. These results show the efficiency and the relevance of the mesh refinement algorithm which allows to capture the salient features of the vortical flow in the wake of the aircraft, but also the jet stream and the interaction between the vortex and the jet, see Fig. 2.13. This capability is really of great importance when you are studying contrail formation. Indeed, as already said before, contrail formation and its evolution are complex processes leading to a spatially inhomogeneous distribution of gaseous constituents and primary particles, whose evolution is led by thermodynamic conditions and aircraft parameters. In particular the vortex/jet dynamic is verified by using this procedure.

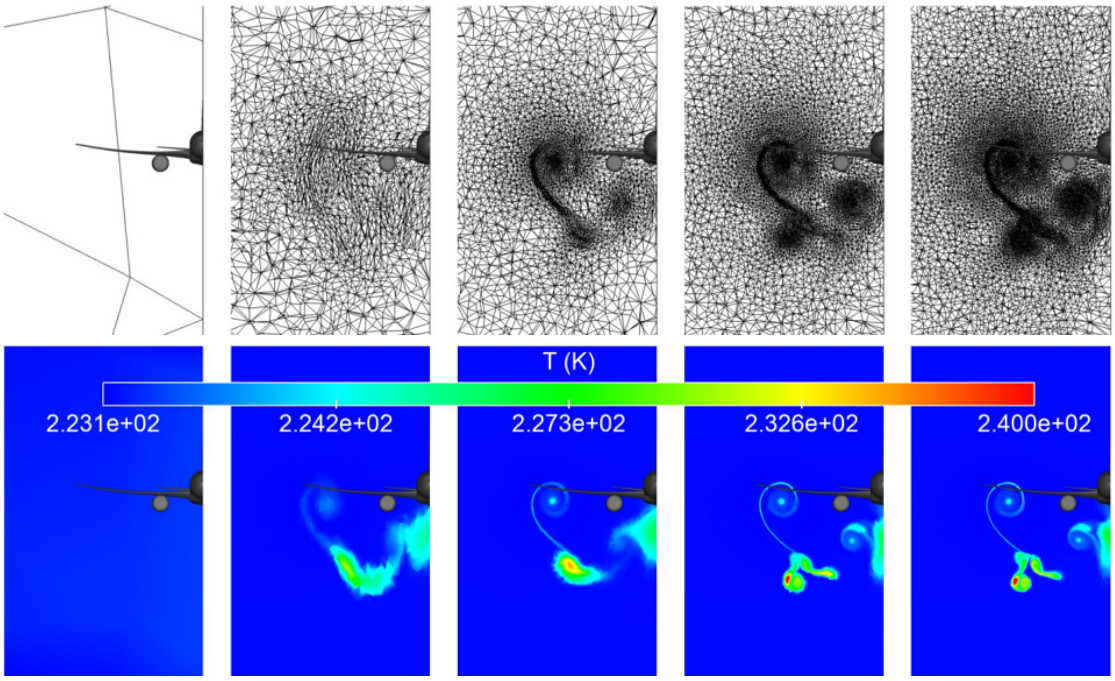


Figure 2.12 – Cut plane and temperature at 8 span behind the wingtip: mesh 1 to 5 from left to right side.

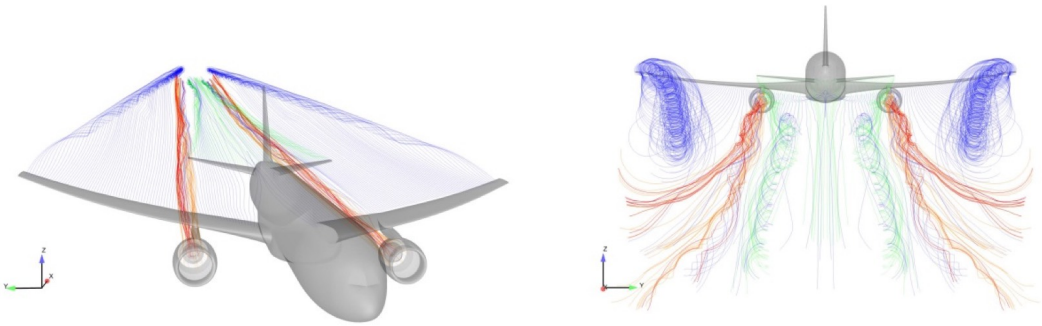


Figure 2.13 – Streamlines colored by element on aircraft: wing in blue, horizontal tail plane in green, engine core flow in red and engine bypass in orange.

2.5.2 High-lift CRM

We present our mesh-adaptive solution computations on the high-lift version of the NASA CRM (HL-CRM) geometry used for the 3rd AIAA CFD High Lift Prediction Workshop [Rumsey] (HLPW3). We consider the geometry with the full chord flap gap which corresponds to the case 1b of the workshop with the following flow conditions:

Mach number	Angle-of-Attack	Reynolds number	Temperature (K)
0.2	16	3.26e6	288.15

The geometry has been transformed in meters, so in that case the reference length is 7.00532 m and the reference surface is 191.84477 m^2. For the mesh adaptation, we the feature-based error estimate controlling the interpolation of the local Mach number in L^4-norm. We use the INRIA `Wolf` flow solver [47] in this case. The underlying numerical scheme to solve the RANS equations is detailed in [5]. This solver allows to use fully unstructured anisotropic grids, even for the boundary layer. For each error estimate, we perform a maximum of $n_{adap} = 20$ mesh adaptation iterations at each fixed complexity and for the convergence study we consider five complexities:

$$\{320\,000,\ 640\,000,\ 1\,280\,000,\ 2\,560\,000,\ 5\,120\,000\}\ .$$

We start the convergence study with an initial mesh composed of 229 263 vertices, 726 852 tetrahedra and 384 868 triangles on the surface. This is a very coarse inviscid mesh without any boundary layer or any specific refinement for viscous flows, in other words no meshing guidelines, thus very easy and quick to generate. We choose to start from this coarse and clearly unresolved mesh to illustrate the non-dependency of the mesh-adaptive solution platform to the initial data. In Figs. 2.14 and 2.15, we observe the major difference between the meshes obtained with adaptation and the tailored meshes based on *a priori* knowledge. In particular, the point distribution on the leading edge are not the following the same pattern. The traditional boundary layer as depicted in Fig. 2.15 clearly shows that a loss of accuracy is obtained for the main wake whereas the wake is fully adapted in the adaptive case from the slats to the flaps. The convergence of the adaptive procedure depicted in Fig. 2.16 emphasizes the fast convergence of the lift and drag with a drastic reduction of the degrees of freedom.

2.5.3 NASA Rotor 37 and periodic mesh adaptation

The NASA Rotor 37 is a low aspect ratio compressor inlet stage 3D test case [Reid 1978], which geometry is shown in Fig. 2.17 (left). The regime considered is described in Table 2.1. The compressor being transonic, shocks appear, interacting with other blades through periodicity which requires an appropriate discretization, unknown a priori and that depends on the flow regime. The bow shock formed in front of each blade induces a boundary layer detachment on the next blade, which modifies the mass flow. Similarly, the tip gap vortex that forms in the gap between the blade and the casing then interacts with the neighboring blade and is responsible for instabilities in some cases. It is thus critical to accurately predict the behavior of these flow features.

For numerical simulation, the flow is assumed to be periodic, so that the numerical domain can be restricted to a single blade in a 10° sector depicted in Fig. 2.17. In this work, we

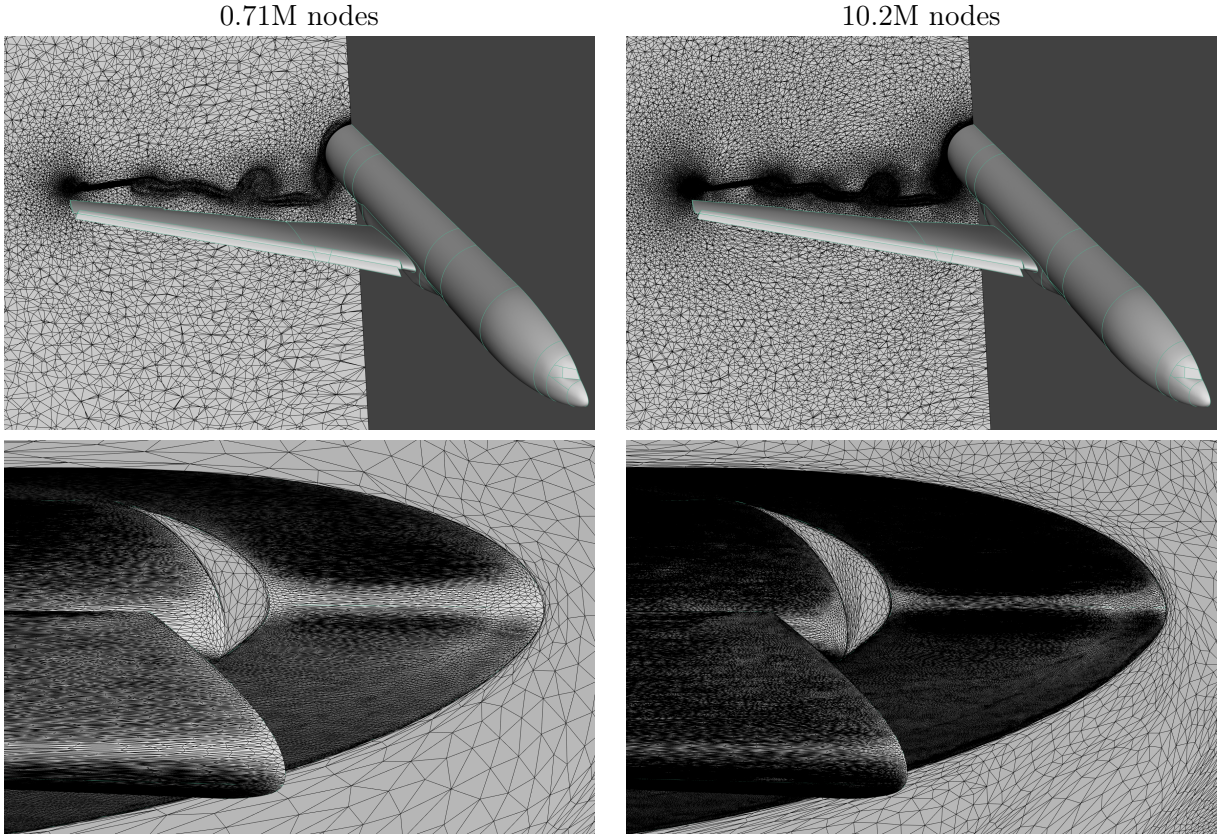


Figure 2.14 – Wakes and surfaces meshes for adapted meshes for the two last complexities with 0.71M nodes (left) and 10.2M nodes (right).

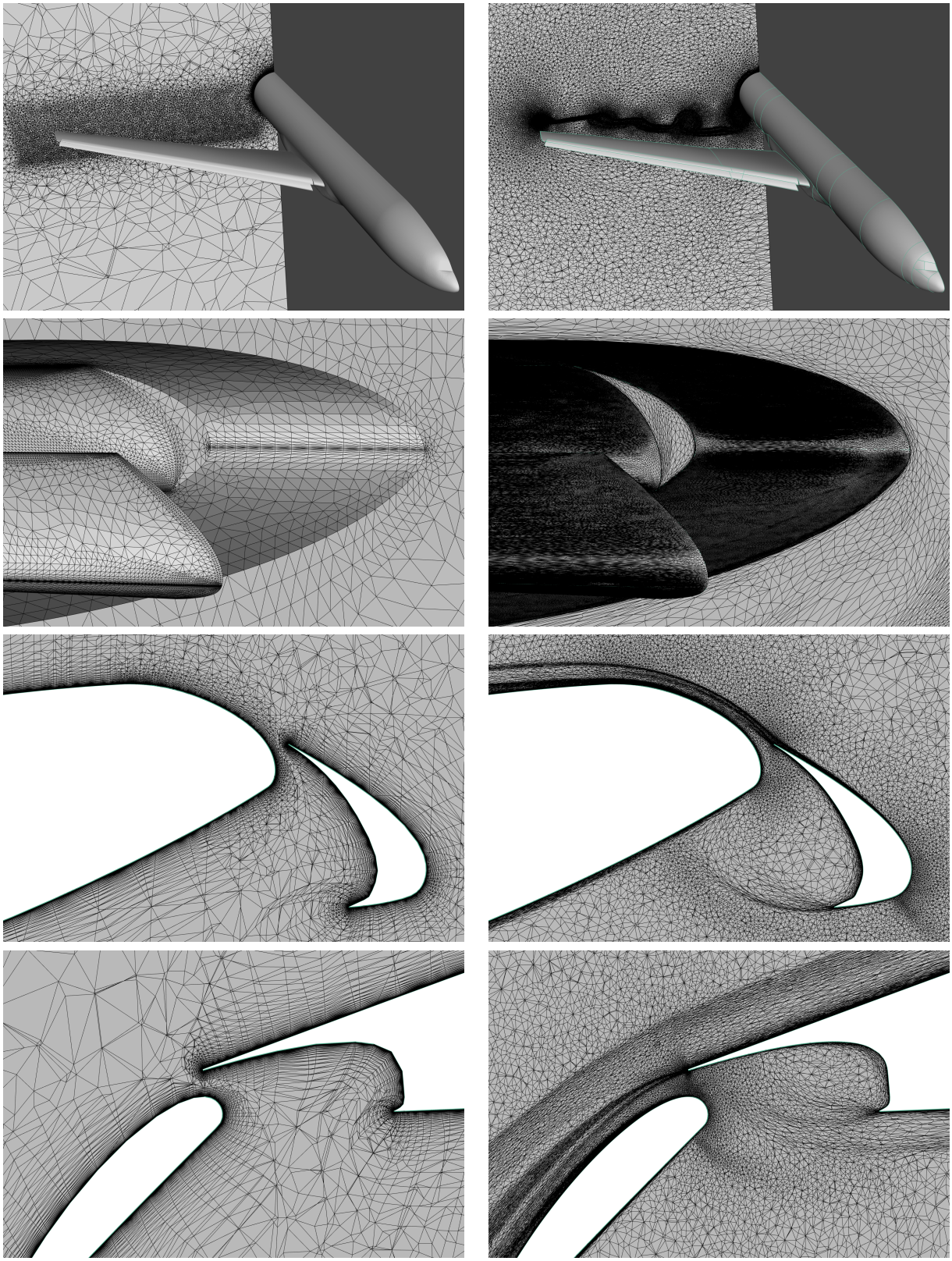


Figure 2.15 – Mesh density comparisons between tailored meshes (left) and adapted meshes (right).

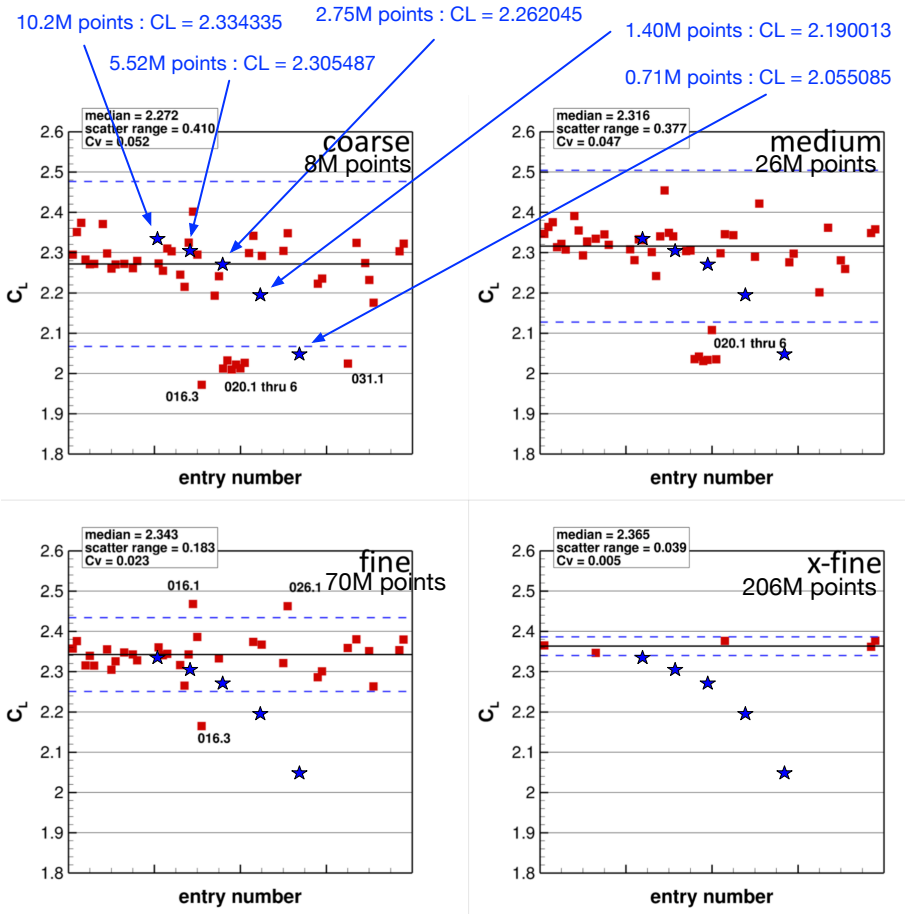


Figure 2.16 – Comparison of the estimated lift (blue stars) with all the workshop entries

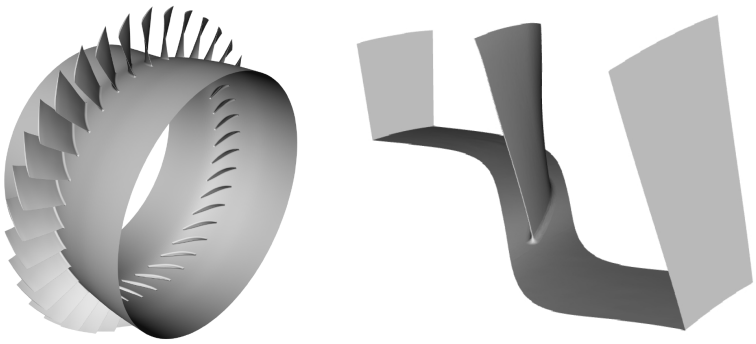


Figure 2.17 – Full compressor stage geometry with RO37 inlet (left) and computational domain (right).

perform feature-based mesh adaptation, minimizing the interpolation error of the Mach field in L^2-norm. The `Wolf` flow solver is used [28]. Five iterations are computed at each of the following complexities:

$$\{200\,000,\ 400\,000,\ 800\,000,\ 1\,600\,000,\ 3\,200\,000\}\,.$$

Inflow Total Pressure	$1.013 \times 10^5\ Pa$
Inflow Total Temperature	$300.0\ K$
Outflow Pressure	$1.013 \times 10^5\ Pa$
Rotating Speed	$1\ 800\ rad.s^{-1}$
Dynamic viscosity	$1.716 \times 10^{-5}\ Pa.s^{-1}$

Table 2.1 – NASA Rotor 37 test case description.

The whole domain is adapted, including periodic boundaries and boundary layers with full unstructured mesh. The geometry of the domain is prescribed with cubic reconstruction from the initial mesh, for surface adaptation.

In order to assess the effect of periodic mesh adaptation, we compare an adapted mesh generated with periodic mesh adaptation containing 3.516.488 vertices and 20.294.221 tetrahedra, to an adapted mesh generated without periodic mesh adaptation, containing 6.607.814 vertices and 38.835.389 tetrahedra. Moreover, in the latter, a fixed structured boundary layer has been added. Fig. 2.18 shows a global view of the periodic frontier in both cases. In the non-adapted case, it is left unchanged and is thus the same as the initial mesh. Meanwhile, in the adapted case we observe that all structures of the flow have been successfully adapted, in particular the shocks upstream and the wake downstream. This discretization has in turn a very strong impact on the solution. In particular, we can see in Fig. 2.19 a close up view on the periodic frontier, where a shock crosses it. It is clear that in the non-adapted case the initial discretization is too coarse to deal with such structures. Moreover, we can tell the discrepancy between the mesh size required in the volume by the adaptation and on the surface. This apparent discontinuity

creates ill-conditioned elements with a strong anisotropy in the opposite direction and an abrupt transition. This will in turn lead to the breakdown of the solver.

We now investigate the impact of this improper discretization on the mesh. Although both computations have been done in a single 10° sector, we duplicated the meshes for visualization. Fig. 2.20 shows a cut in the mesh perpendicular to the axis of the blade. We can see the shocks produced by the leading edge of the blade propagating upstream. These shocks are clearly diffused by the periodic frontier when it is not adapted, they barely cross it four times, while in the adapted case, shocks are unaffected by the boundary and propagate up to the inflow. Moreover, while in the adapted case the periodic junction is seamless and barely visible, the non-adaptation of the periodic frontier yields a strong constraint in the mesh which makes it frankly visible. This produces a lot of noise in the mesh and the solution.

A closer look at the shocks near the leading edge reveals in Fig. 2.21 how the two leading edge shocks are actually suddenly artificially diffused as they cross the periodic frontier. We can tell how this affects the solution as the second one seems to even disappear while the first one is actually reflected and hits the blade back. After, in front of the lambda shock we see in the adapted case a secondary shock which is absent in the non-adapted case. Here again, the junction is seamless in the adapted case, while the non-adapted case produces concentration of ill-conditioned elements next to the periodic boundary condition. An even closer look at the shock boundary layer interaction where the lambda shock hits the blade shows how the shock is diffused by the non-adapted domain. We can tell the size of the elements on the frontier by the spread of the shock that is created by this interaction. This interaction directly modifies the shape of the shock but also generated a wake in the middle of the vane which modifies the debit and velocity of the flow.

Finally, Fig. 2.22 shows a global view of another cut perpendicularly to the blade. This summarizes the aforementioned observations, we can see how the Mach field shocks are diffused in the non-adapted case and does not propagate upwind as far as with periodic adaptation. We also observe in the Mach field the reflection of the shocks on the periodic frontier and the generation of a huge artificial wake. Finally, we distinguish a similar impact of the discretization on the wake. Overall, the solution obtained with periodic mesh adaptation is definitely cleaner and the use of mesh adaptation without periodicity definitely pollutes the solution.

2.6 Conclusion

The unique-cavity based operator defines a convenient framework for adaptive mesh generation. It allows us to define a much more efficient adaptive mesh generation strategy. The choice of the initial cavity defines the baseline desired operator. From a practical point of view, the same operator is used throughout the code and any modification of the operator (to handle hybrid entities, non-manifold entities) implies an automatically upgrade of the entire code. The software is then easily maintained and extended. By minimizing the number of rejected mesh modification operations with appropriate cavity corrections, like the collapse or insertion of surface point, it naturally speeds up the overall process, up to a factor 40 with respect to the trivial approach of Chapter 1. The high level of flexibility of the operator is illustrated with its constrained version of the operator to generate hybrid or boundary layer meshes from an initial

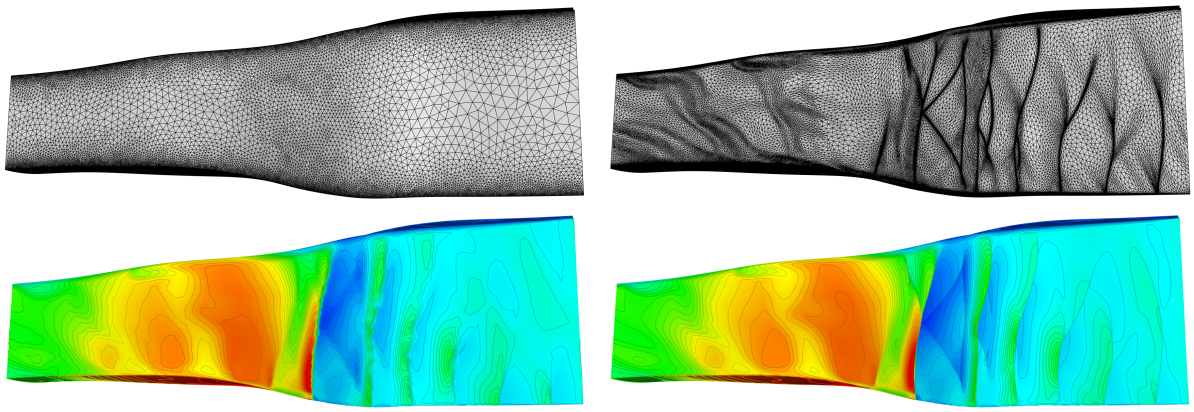


Figure 2.18 – NASA RO37: Global view of the mesh (top) on the periodic frontier in the non-adapted (left) and adapted case (right), and the corresponding Mach field (bottom).

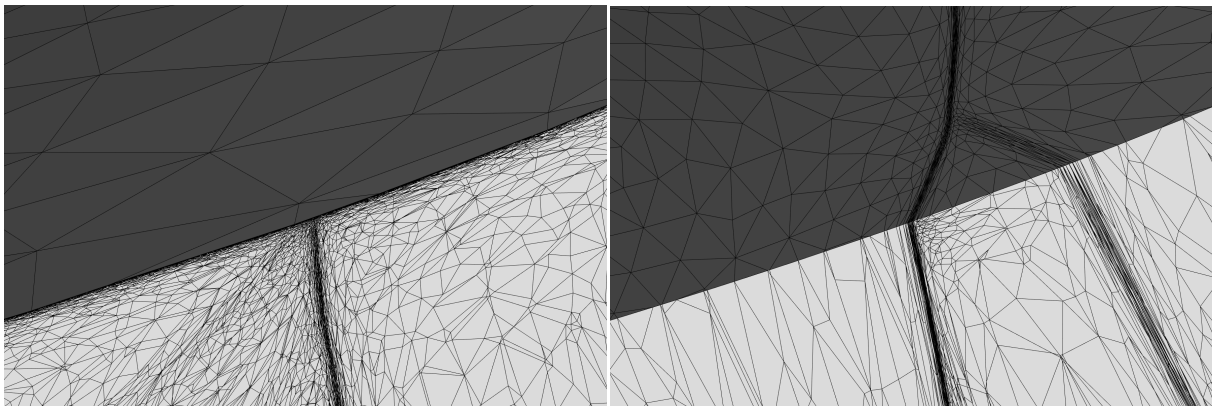


Figure 2.19 – NASA RO37: Impact of the discretization of the periodic frontier on a shock.

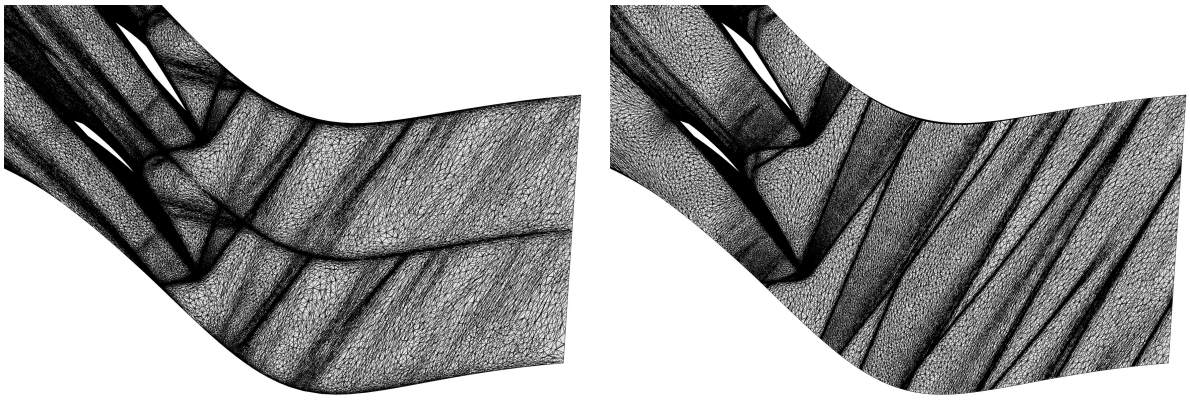


Figure 2.20 – NASA RO37: Propagation of shocks upstream of the RO37 blades without (left) and with (right) periodic mesh adaptation.

3D mesh. In addition to the improvements of turbulent flow solver and error estimates, the level of anisotropy on the surface and volume becomes sufficient to observe spatial convergence rate on challenging RANS computing as in turbomachinery or high-lift computations while mostly

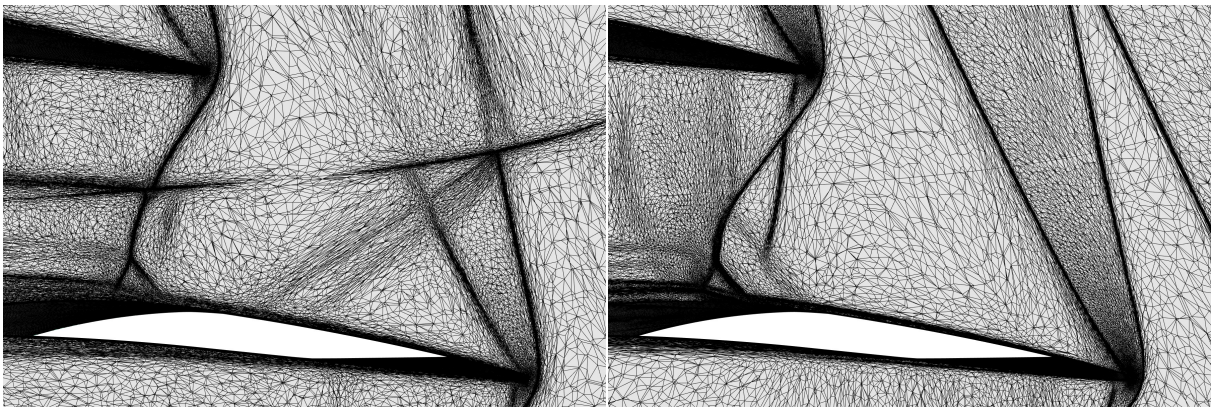


Figure 2.21 – NASA RO37: Zoom on the leading edge shocks of the RO37 blades without (left) and with (right) periodic mesh adaptation.

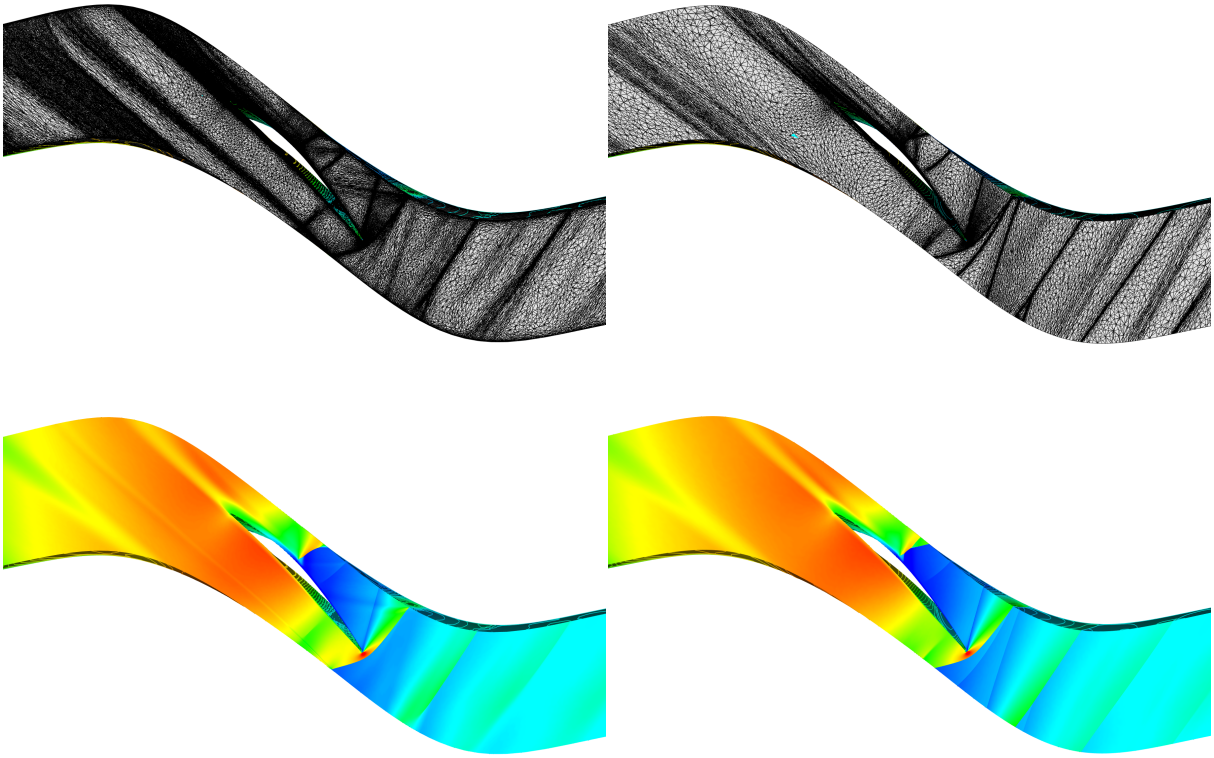


Figure 2.22 – NASA RO37: Global view of the mesh (top) and the solution (bottom) in a cut in the domain perpendicular to the blade without (left) and with (right) periodic mesh adaptation.

inviscid computations were performed in the previous chapter.

A uniform and predictable speed is guaranteed for the mesh adaptation phase. On typical intel Core i7 at 2.7Ghz, the cavity-based collapse reaches 20 000 points removed/sec and the cavity-based insertion 20 000 points inserted/sec. This predictability in speed with a strong anisotropy is one of the crucial properties that are used for the parallelization introduced in the next chapter.

Chapter 3

Metric-aligned, metric-orthogonal and parallelism

The mesh generation algorithm of Chapter 2 reaches two limitations. First, the shape of the elements, and thus the quality, cannot be explicitly controlled. Indeed the new vertices are generated from the set of long edges in the metric of the current mesh. The second limitation is the parallelism. If meshes containing more than 120 million tetrahedra are routinely generated in serial, a much bigger number of elements are needed to verify the asymptotic convergence on complex cases. Consequently, a parallel adaptive strategy is required.

To address the first issue, we define a strategy to generate metric-orthogonal meshes, *i.e.* composed of elements aligned with the eigenvectors of the metric field. This strategy relies on the cavity-based operator within an anisotropic frontal insertion of points. For the parallel strategy, dedicated mesh partitioning schemes are presented. They are based on metric-based work estimates. Again the cavity operator allows to accurately predict the CPU time of each step of the remeshing process. We illustrate these two enhancements on several 3D numerical examples.

The metric-aligned and metric-orthogonal approaches are based on a collaboration with Mississippi State University [45, 43] while the parallel implementation was done during the PhD thesis of V. Menier [9, 48, 39] and validated with the Unstructured Grid Adaptation Working Group [31].

Ouline. The chapter is decomposed into two distinct parts. The first one focuses on the metric-aligned and metric-orthogonal meshing strategies while the second part focuses on the parallel implementation of the adaptive mesh generation process.

3.1 Mesh adaptation with orthogonality and alignment

During an adaptive remeshing process based on the unit-mesh concept using standard or cavity-based operators (as described in Chapters 1 and 2), the shape of the created unit elements is not controlled. Indeed, as a unit element is given by $\mathcal{M}^{-\frac{1}{2}} R K$ [11], where K is the regular triangle or tetrahedron and R a rotation matrix, the edges of a unit element can span all the directions of the space, see Fig. 3.2. In addition, the eigenvectors of $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ are never explicitly used in the distance, volume or quality function. As a result, all the anisotropic mesh generators produce anisotropic elements where the edges have no particular orientation. If an equal level of interpolation error (in L^1 norm) can be achieved on these unit elements [11], their dihedral angles or error level in L^2 or H^1 norms are greatly different. These quantities are not optimized in classical anisotropic mesh adaptation although they may unfavorably impact

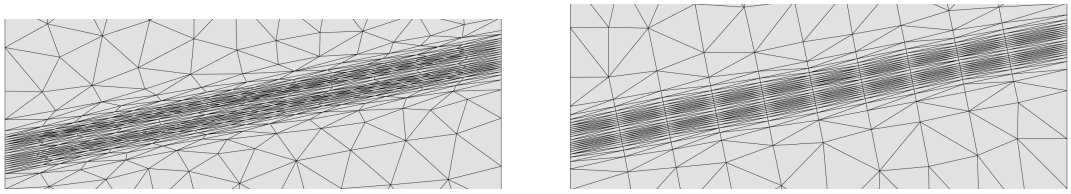


Figure 3.1 – Comparison with a standard anisotropic mesh (left) and the equivalent aligned mesh.

the quality of a numerical simulation. We will show how to enforce the alignment of the edges with the local eigenvectors of the provided metric. As the eigenvectors are orthogonal, such meshes are called *metric-orthogonal* or *quasi-structured* meshes [Sharbatdar 2013, Krause 2003]. In particular, when an isotropic metric field is provided, *Cartesian* grids are recovered with elements aligned with the x-y-z directions. An example of metric-orthogonal mesh for a shock waves is depicted in Fig. 3.1 (left) and compared to classical anisotropic mesh generation (right). These two meshes are equivalent in terms of length distribution (and then on interpolation error level). However, the quality and angles of the triangles are clearly improved for the orthogonal and aligned approach.

As we want to force the alignment of the edges, standard local remeshing approaches based on a set of classical operators (insertion, collapse, swap, ...) as in [57, Michal 2011] seem to be more delicate to use as they iteratively modify the mesh with no specific ordering. On the contrary, frontal methods have been used to generate high-quality isotropic meshes but with little success for anisotropic mesh generation. We combine both approaches: only local operators are used in order to ensure robustness and a frontal insertion of points is used in order to control the alignment of vertices along the eigenvectors of $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$. However, contrary to fully frontal mesh generation techniques [Löhner 1988b, Mavriplis 1995] where a front of points/elements is used to fill the computational domain, the points are inserted in an *empty volume mesh*. An *empty mesh* is a valid volume mesh composed of a minimum (or a small) number of volume points, while the surface mesh is assumed to be adapted to the input metric. Inserting the points in an empty volume mesh is motivated to avoid the collision of the frontal points with already existing volume points. Note that empty meshes are usually generated after the boundary recovery phase in typical mesh generation algorithm [Baker 1987, George 1998]. However, if such mesh is not available, it is possible to define a fast coarsening cavity-operator to quickly reach the empty mesh state. The initial metric field is stored on a background mesh so that it is easy to interpolate the metric for the frontal creation of vertices. The interpolation scheme is based on the log-Euclidean framework, see Eq. (1.5), as explained in Chapter 1. Consequently, in order to reduce the computation of the interpolation, the metric is stored on the background mesh and it is stored in logarithmic form. This avoids multiple diagonalization steps. Starting from the initial set of points of the surface mesh, new points are created along eigenvectors or preferred directions at unit length and then inserted in the current mesh. Both coarsening and insertion operators are based on the cavity-based framework described in Chapter 2.

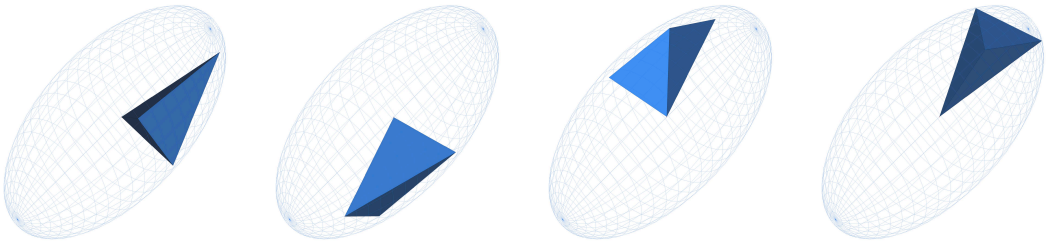


Figure 3.2 – Illustration of several unit tetrahedron with respect to a 3D metric represented by its unit ball.

3.2 Metric-orthogonal and metric-aligned anisotropic mesh generation

Starting from a provided input 3D valid volume mesh and metric, the generation of an adapted metric-orthogonal or metric-aligned mesh is a combination of the previous operators with a frontal algorithm to propose the points to be inserted. We detail in the following, all the steps involved in this process.

3.2.1 Frontal creation of vertices

In order to favor orthogonality or alignment of the final mesh, a frontal approach is used. However, contrary to standard frontal approaches, we use a front of vertices instead of a front of faces. From a practical point of view, the new points are proposed by vertices and not by faces. In an anisotropic context, the new points depend only on the eigenvectors and eigenvalues of the metric of the front point. Note that the directions depend on the orthogonal or aligned desired pattern. These directions are illustrated in Fig. 3.3 in 2D. For all cases, we observe that the direction aligned with the shortest size eigen direction is used. The frontal approach proposes 6 (2D) and 24 (3D) points are proposed, In the orthogonal approach, all directions are aligned with the eigen vectors and 4 (2D) and 6 (3D) points are proposed. For simplicity we focus on the orthogonal approach.

The initial front of points is given by the list of the surface points. Given a point $\mathbf{x}_o$ and its metric $\mathcal{M}_o$ of the current front with eigenvectors $(\mathbf{u}_i)_{i=1,3}$ and eigenvalues $(\lambda_i)_{i=1,3}$, six points are proposed:

$$\mathbf{x}_i = \mathbf{x_o} \pm \lambda_i^{-\frac{1}{2}} \mathbf{u}_i. \tag{3.1}$$

When the metric is isotropic, we force the eigenvectors to be aligned with the natural axis of $\mathbb{R}^3$. Note that these points are just a first guess and several additional checks are performed before trying insertion. The first check consists in verifying that the new points are in the current volume mesh by using a simple mesh localization algorithm. This check is also performed on the background mesh. The background mesh localization also provides the metric $\mathcal{M}_i$ of $\mathbf{x}_i$.

In order to take into account the variation of the metric, the final position of $\mathbf{x}_i$ and metric $\mathbf{x}_i$ are updated. The procedure is based on a dichotomy along the segment $[\mathbf{x}_o, \mathbf{x}_i]$ in order to

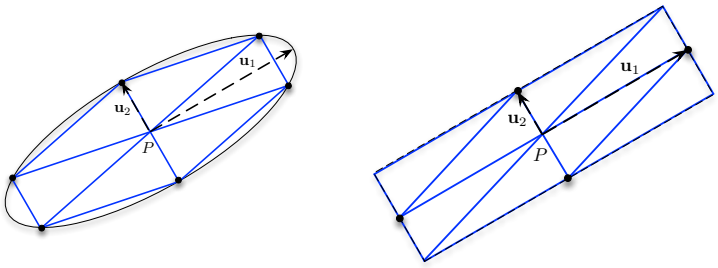


Figure 3.3 – Examples of the six directions for metric-aligned (left) and metric-orthogonal (right).

make sure that the Riemannian length evaluation of the vector $[\mathbf{x}_o\mathbf{x}_i]$ is unit:

$$\int_0^1 \sqrt{{}^t[\mathbf{x}_o\mathbf{x}_i]\,\mathcal{M}(t)\,[\mathbf{x}_o\mathbf{x}_i]}\,\mathrm{d}t = 1,$$

where $\mathcal{M}(t)$ is a geometric interpolation between metrics $\mathcal{M}(0) = \mathcal{M}_o$ and $\mathcal{M}(1) = \mathcal{M}_i$. Note that the original guess (3.1) only guarantees:

$${}^t[\mathbf{x}_o\mathbf{x}_i]\,\mathcal{M}_o\,[\mathbf{x}_o\mathbf{x}_i] = 1.$$

Consequently, we seek for an optimal point $\mathbf{x}_{opt}$ with back-mesh interpolated metric $\mathcal{M}_{opt}$ lying along the initial direction $\mathbf{x}_o\mathbf{x}_i$. Note that we need to iterate, because we interpolate the metric from the background mesh. If $\mathcal{M}_{opt}$ were interpolated by $\mathcal{M}_o$ and $\mathcal{M}_i$, an analytical formula exists depending on the metric interpolation scheme used. This list of points is then filtered in order to suppress from insertion points that are too close in the distance computed in the metric. The filtering process (similar to the point filtering defined in Chapter 2) gives the list of points to be inserted. This list of points defines the next front. This algorithm is applied until the list of points to be inserted becomes empty. We mention that different procedures may be used to generate the list of points to be inserted. In [Marcum 2014], the list is issued from a front of faces instead of a front of points. In order to have an optimal ordering and to favor alignments at smallest size, a heap list of composed of point/size/direction $(P_i, h_k, \mathbf{u}_k)_{ik}$ is used. The advancing point algorithm is summarized as :

1. Pop the first heap list entry, creates $P_{new} = P_i \pm h_k\,\mathbf{u}_k$,
2. Update length/position of P_{new} according to Riemannian metric field.
2. Metric-based length filtering, add P_{new} for insertion,
3. Update the heap list with $(P_{new}, h_k, \mathbf{u}_k)_k$,
4. If the heap list is not empty goto 2.

The points obtained for a half circle metric are depicted in Fig 3.4. The obtained mesh with standard and metric-aligned are depicted in Fig. 3.5.

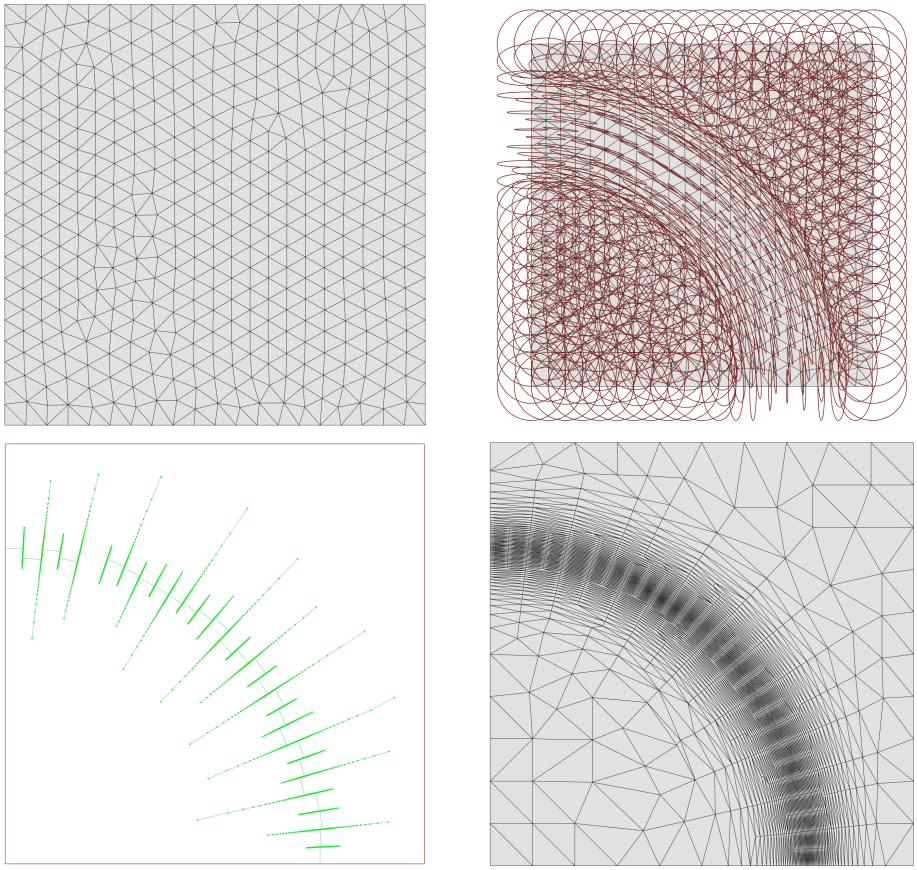
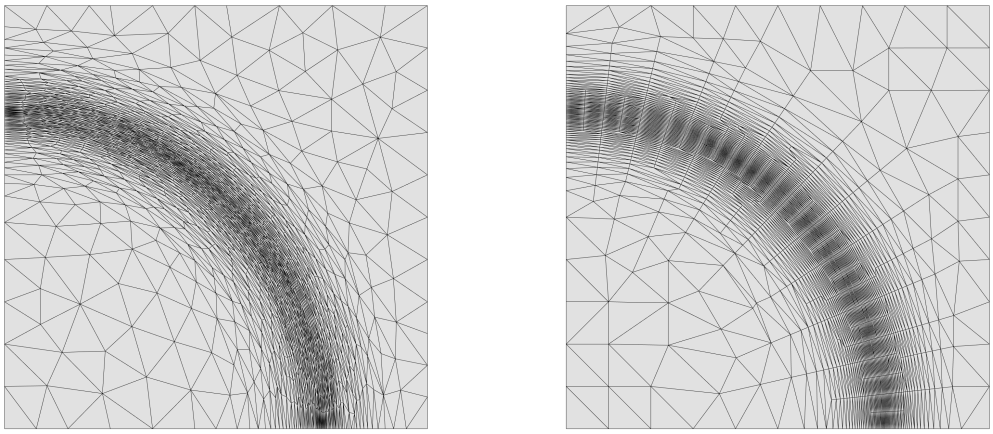


Figure 3.4 – Illustration of the frontal point creation on a simple half-circle metric.



Standard Advancing-point

Figure 3.5 – Final unit mesh for the circle metric with the standard approach (left) and aligned approach (right).

3.3 Numerical examples

The first example is a tailored mesh where isotropic source terms are used to generate an axes-aligned mesh around and in the wake of an f117 aircraft. The aircraft has an angle of attack of 20 degrees. The scope is to study the vortical flow generated by the delta shaped wing. For this example, the use of locally structured grids is preferred as the smooth vortices interact behind the aircraft. Reducing the solver diffusion is thus a key condition to observe the full picture of the physical phenomenon. The flow solver is `Wolf` [14], and an unsteady inviscid simulation is performed. The total CPU time is 12 hours on 8 cores of an i7 at 2.7Ghz. The mesh composed of 7 532 632 vertices and 45 721 814 tetrahedra. The CPU time to generate this mesh is 25 minutes. In Fig. 3.6 (middle left), we see how the fronts merge smoothly with nothing but filtering as specific treatments. For the standard approach, the dihedral angles follow a Gaussian distribution centered on the mean dihedral angle of the regular tetrahedron whereas the distribution is centered around this mean value and the right angle value for the metric-orthogonal approach, see Fig. 3.6 (bottom left).

If the previous example is isotropic, this approach can be used to generate anisotropic meshes as well. We consider a transonic flow computation around a generic Falcon geometry at Mach 0.8 with an angle of attack of 3 degrees. We adapt the solution to the Mach number by controlling the interpolation error in L^2 norm [57], the final mesh is obtained after 30 iterations and is composed of 1 110 735 vertices and 6 546 789 tetrahedra. The total CPU time is 40 minutes on an Intel Core i7 laptop at 2.7Ghz. Local orthogonal features clearly appear in the wake, see Fig. 3.8 (bottom left). For all meshes, more than 95% of the edges have a unit length in the metric. A comparison with a standard mesh adaptation approach is given in Fig. 3.9 (left) where dihedral angles are compared between the metric-orthogonal approach and the standard one. For the metric-orthogonal approach, we see that more than 25% (resp. 5% for the standard approach) of the elements contain right dihedral angles while minimizing the number of large dihedral angles. The number of elements with small angles is also greater than in the standard approach. This reveals that the level of anisotropy is even higher for the metric-orthogonal approach.

We now consider the example of a dam break on a rectangular obstacle. In this simulation, the compressible bi-fluid solver ANANAS©[Ing.] is used. The error estimate is based on a level-set metric in order to capture and predict accurately the interface between the water and the air. The total CPU time for a physical time of $15s$ is $36h$ for the fully unstructured and $24h$ for the Cartesian approach. Both simulations were run on 4-cores of an Ivy-Bridge i7 at $3.4Ghz$. We observe in Fig. 3.10 the dynamic of the flow. As the metric-orthogonal approach tends to insert fewer points (especially in the transverse direction), the CPU time is lower than the classical anisotropic approach. The distribution of the dihedral angles for the standard and metric-orthogonal approach are reported in Fig. 3.9 (right) for time $0.85s$. The histogram shows that the angle distribution is centered around small angles and right angle whereas a uniform distribution is observed for the standard approach. Some cuts in the volume mesh are reported in Fig. 3.11 at different times. It shows how the mesh around the interface is locally structured and how the edges are automatically aligned.

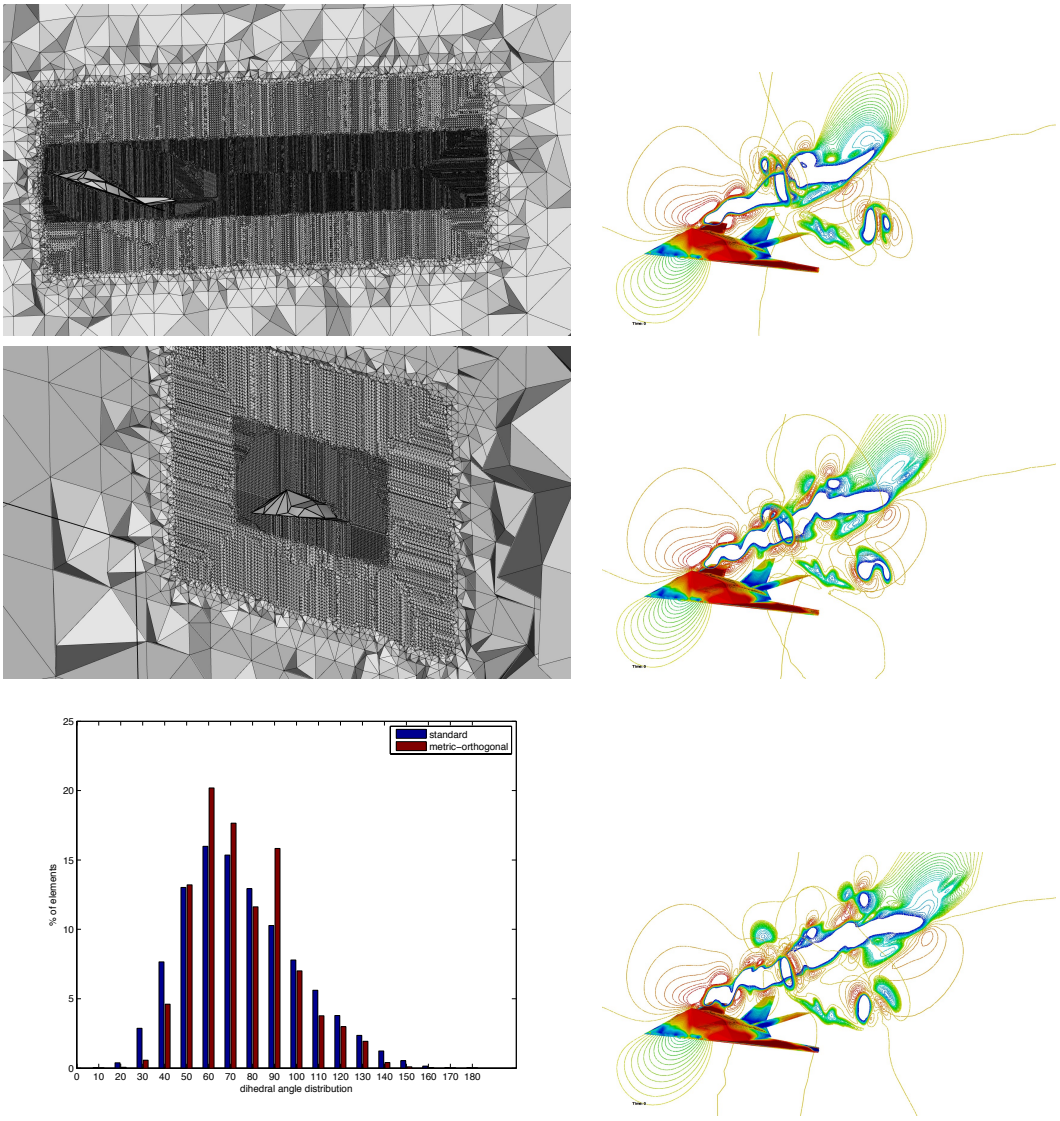


Figure 3.6 – Example of metric-orthogonal mesh generated from an analytical isotropic metric. Different views of the 3D mesh (left) and iso-lines of the vortices generated by the f117 at different times (right). Distribution of the dihedral angles with the standard and metric-orthogonal approaches (bottom left).

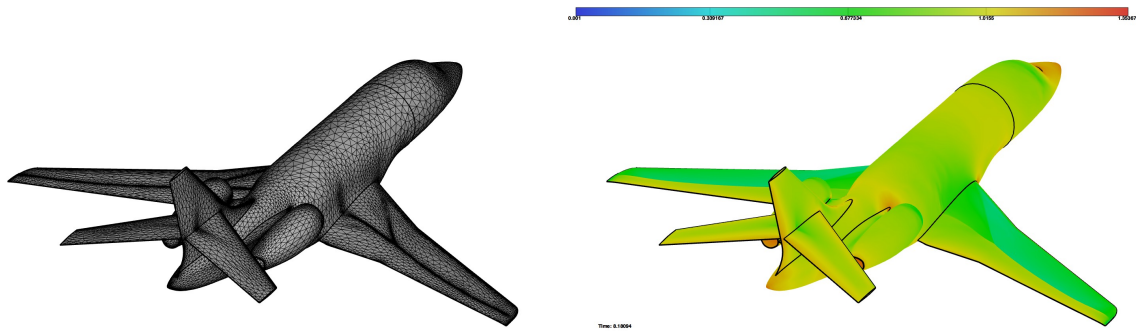


Figure 3.7 – Final adapted surface mesh on the falcon geometry (left) and density iso-values (right).

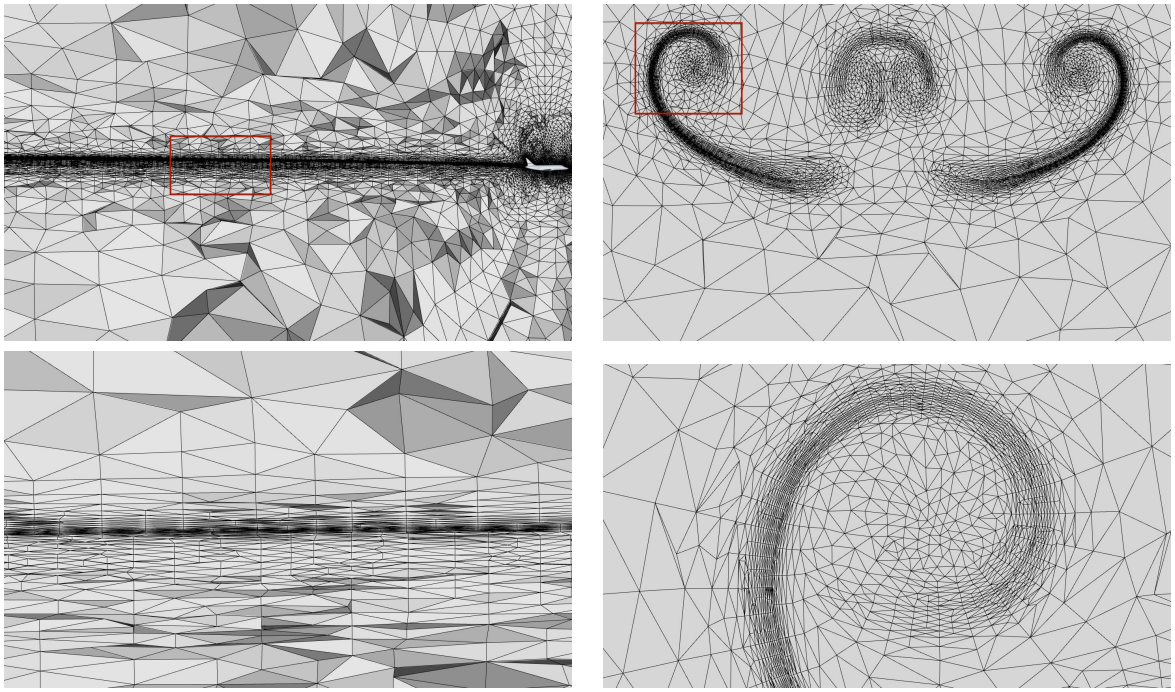


Figure 3.8 – Cuts in the final adapted volume mesh showing the aligned anisotropic elements: in the wake (top left), the vortex 100m behind the falcon (top right). Closer view in the red rectangles of the wake (bottom left) and the vortex (bottom right)

3.4 Parallel large scale mesh adaptation

Parallel mesh generation has been an active field of research [Löhner 2013, Tremel 2007, Ito 2007, Foteinos 2012]. Two main frames of parallelization exist: coarse-grained [Digonnet 2013, Lachat 2014, Löhner 2013], and fine-grained [Foteinos 2012, Shephard 2013, Chernikov 2010, Özturan 1994]. Fine-grained parallelization requires to implement directly in parallel all the mesh modification operators at the lowest level: insertion, collapse, swap ... This usually implies the use of specific data structures to handle distributed dynamic meshes, especially for adaptive proce-

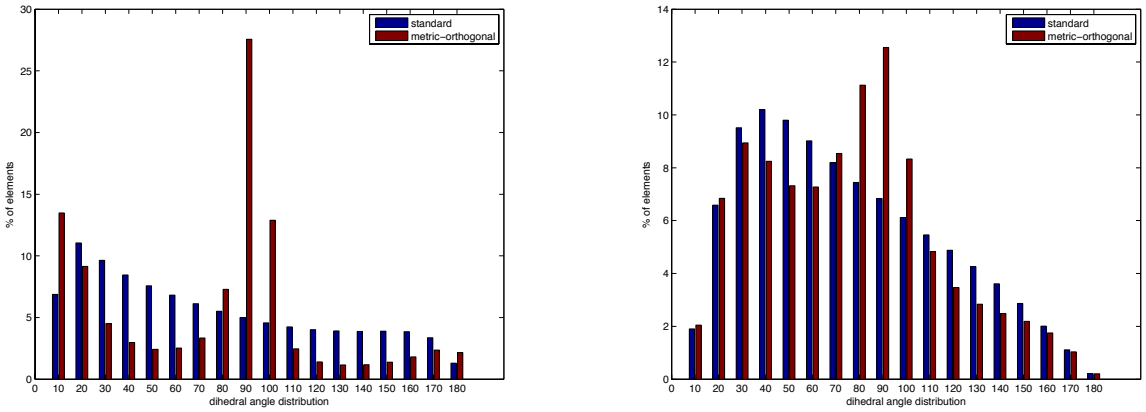


Figure 3.9 – Comparison of the dihedral angle distribution of the standard anisotropic mesh adaptation with the metric-orthogonal approach: falcon case (left) and dam break case (right).

dures [Alauzet 2006]. The second approach consists in the use of a bigger set of operators in parallel. Most of the time a complete sequential mesh generator or mesh optimizer is used. Both approaches have been also extended to adaptive frameworks, see [Digonnet 2013, Lachat 2014] for the coarse-grained approach and [Shephard 2013] for the fine-grained. We follow the coarse-grained parallelization in an adaptive context within the metric-based framework. In particular, we address the following issues.

Surface-volume problematic. When considering the coarse-grained strategy, parallel mesh generators or parallel local remeshers generally adapt either the surface or the volume mesh. In [Alleaume 2008, Lachat 2014, Löhner 2013], the initial fine surface mesh is unchanged during the parallel meshing process. If this consideration works well for uniform or isotropic meshes, it turns out that it is mandatory to generate the volume and the surface meshes simultaneously when anisotropic meshes are considered. Indeed, the set of methods that have demonstrated a good efficiency and reliability to mesh a given complex surface mesh: advancing front method [Löhner 1988b, Mavriplis 1995], constraint global Delaunay [Baker 1987, George 1998, George 1990] or a combination of both [Marcum 2001] are mostly susceptible to fail when an anisotropic surface mesh is provided on input. The frontal methods generally do not succeed to close the front, while the Delaunay-based methods will generally fail during the boundary recovery phase. Consequently, being able to adapt the surface and the volume into a single thread is necessary to gain in robustness [57]. However, adapting both the surface and the volume meshes at the same time implies additional complexity for the load balancing as the costs of the volume or surface operators differ.

Domain partitioning. Domain partitioning is a critical task as each partition should represent an equal level of work [De Cougny 1999]. Graph-based techniques [Karypis 1998] tend to minimize the size of the cuts to reduce the communication cost. However, this cost function is not relevant for adaptative mesh generation, especially if the coarse-grained approach is used where there is no communication at the interface during the remeshing step. For adaptive mesh genera-

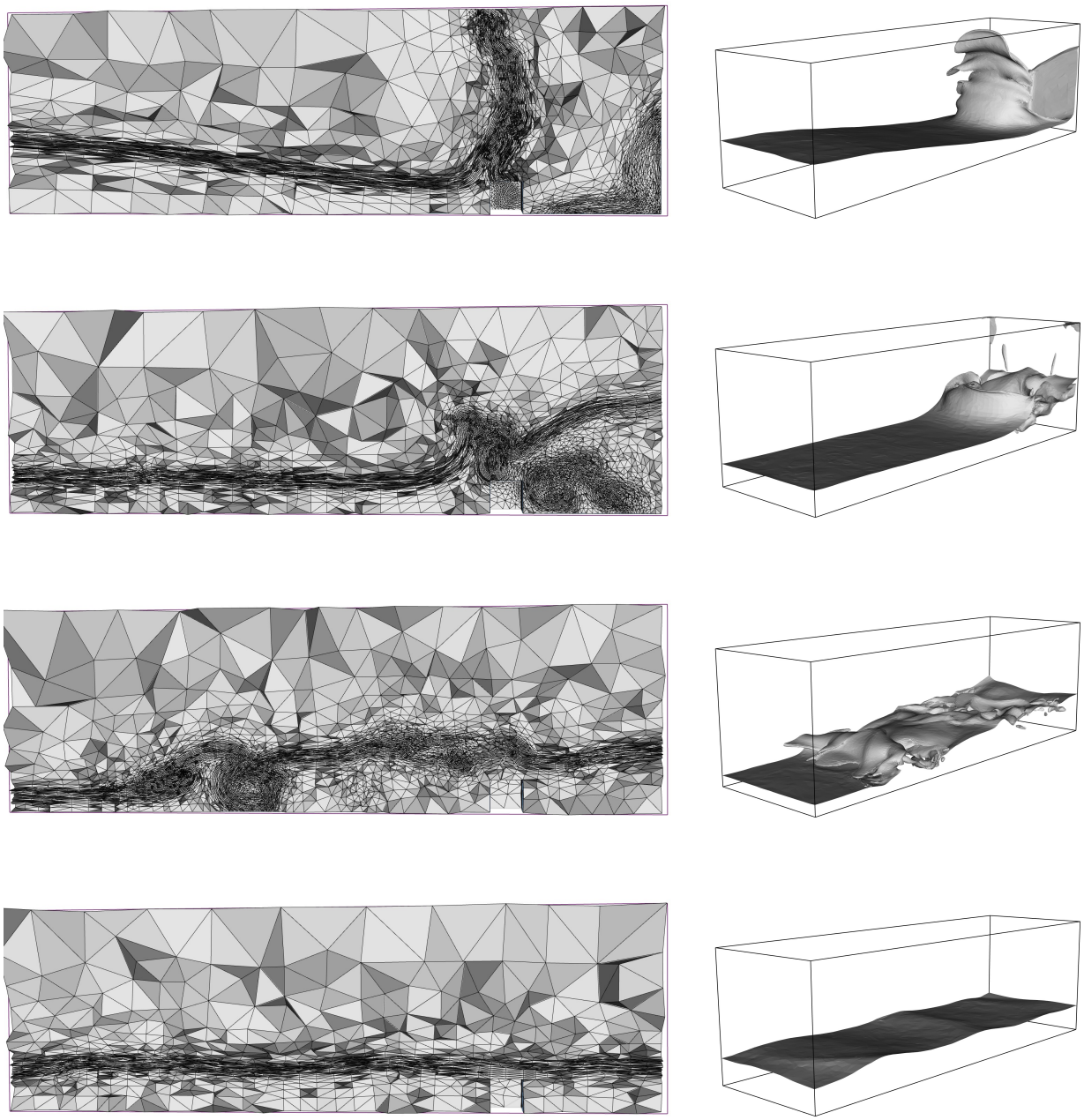


Figure 3.10 – Dam break simulation: cuts in the volume meshes for times $0.85s$, $1s50$, $2.50s$, and $7s50$ (left) and representation of the interface water/air (right).

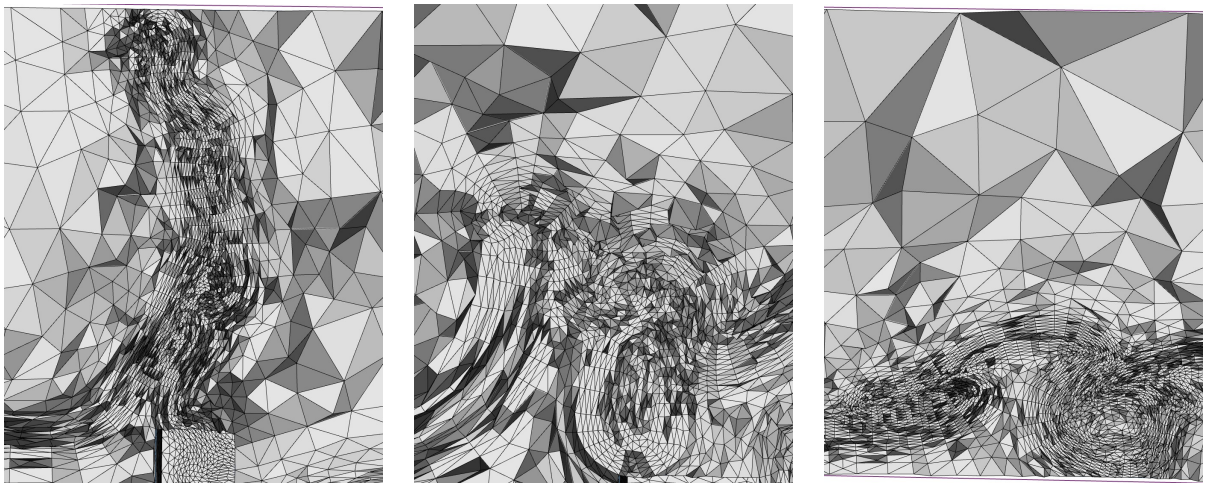


Figure 3.11 – Dam break simulation: Close view in the volume meshes for time $0.85s$, and $1s50$ and $2.50s$.

tion, the cost function (to define the cuts) is related to the amount of work (number of collapses, insertions, optimization steps, ...) needed on each partition. This becomes even more critical for anisotropic mesh adaptation where refinements have a large variation in the computational domain. Estimating accurately this work *a priori* is also challenging for anisotropic meshing as it strongly depends on the properties of the serial meshing algorithm. Additional developments of graph-based methods are then necessary to work in the anisotropic framework [Lachat 2014]. Domain partitioning also represents one of the main parallel overheads of the method. In particular, general purpose graph partitioners cannot take into account the different geometrical properties of the sub-domains to be partitioned. Indeed, splitting the initial domain is completely different from splitting the interface mesh, see Figs. 3.12 (top left and bottom left) and 3.13. In addition, there exist additional requirements that the partitioning algorithm should ensure in order to ease the work of the serial mesh generator. The first requirement is to ensure that the parts are connected and the second requirement is to ensure that the number of non-manifold (surface) edges is as minimal as possible.

Partition remeshing. This is the core component of the coarse-grained parallelization. The overall efficiency of the approach is bounded by the limits of the sequential mesh generator. One limit is the speed of the sequential remesher that defines the optimal potential speed in parallel. In addition, as for the partitioning of interfaces, meshing a partition is different from meshing a standard complete domain. Indeed, the boundary of the partition usually features non-manifold components and constrained boundary faces. In particular, it is necessary to ensure that the speed and robustness of the remesher is guaranteed on interface meshes. Consequently, when a black-box mesher is used, achieving good performances in parallel may be difficult. Additional developments and cares are usually needed in the serial meshing algorithm [Digonnet 2013]. In addition, estimating the cost of the mesh modification operators of the serial meshing algorithm is required to estimate the required work and to drive accordingly the partitioning algorithm.

Out-of-core. Out-of-core meshing was originally designed to store the parts of the mesh that were completed on disk to reduce the memory footprint [Alleaume 2008]. Despite the high increase of memory (in terms of storage and speeds with solid state drives), coupling out-of-core meshing with a parallel strategy may be advantageously used. On multi-socket shared memory machines (with 40-200 cores), if the memory used by a thread is bigger than the memory of a socket, then the memory exchange between neighboring sockets implies a huge overhead on the sequential time (when running the procedure with one thread only). For instance, on a DELL PowerEdge R900 with 4 Intel Xeon E7 sockets with 10-cores with 1 Tb of RAM, we observe that the speed of the serial meshing algorithm is twice slower when the used memory exceeds 256 Gb, *i.e.*, the RAM of one socket. This drawback is even more critical on NUMA architectures.

Our procedure is based on standard coarse-grained parallel strategies [Lachat 2014, Löhner 1992, Löhner 2013] where the initial domain is split into several sub-domains that are meshed in parallel. A sketch of the procedure is depicted in Fig. 3.12. Note that we can decompose the procedure by level, where 3 levels are shown on the simple example of Fig. 3.12. At the first level, the initial domain is split and considered for adaptation. From the set of constrained faces (at interface) of the previous level, a new volume mesh (domain) is deduced for the next level. This new domain is then split and adapted in parallel. This process is then applied until convergence. We will show that a maximum of 5 levels is needed to complete the process.

To address the mesh partitioning issues, we define a hierarchical partitioning technique that depends on the current level of the procedure. For the first level, a fast and parallel Hilbert based partitioning is used while a breadth-first search with restart algorithm is used at the next level. This allows us to take advantage of the geometry of the mesh at the interface in order to minimize and reduce the number of constrained faces at each step. For each level, specific partition corrections are designed to guarantee that each final partition is connected while remaining well-balanced. To handle nonuniform refinements (in terms of sizes and directions), a metric-based static load balancing formula is used to *a priori* equilibrate the work on each sub-domain.

For the serial meshing algorithm, we use the unique anisotropic cavity-based operator, described in Chapter 2, to perform the mesh adaptation . The main advantage is that we obtain a constant speed whatever the considered operator. This feature allows us to derive an accurate metric-based work that is easily deduced from the input metric-field and input mesh only.

Once the remeshing of a sub-domain is completed, two additional sub-domains are created. The first one represents an interface mesh composed of elements that need additional refinement. The second one is the completed part that is stored to disk. To define the interface mesh, mesh modification operators (insertion/collapse) are emulated in order to enlarge the initial interface mesh to perform a quality remeshing in the subsequent levels. Current state-of-art parallel mesh generation approaches [Digonnet 2013] for unstructured (and adapted) meshes require thousands of cores (4092-200 000 cores) to generate meshes with a billion elements. Our scope is to make this size of mesh affordable on cheaper parallel architectures (≈120 cores) with an acceptable runtime for an adaptive design process (less than 20 min).

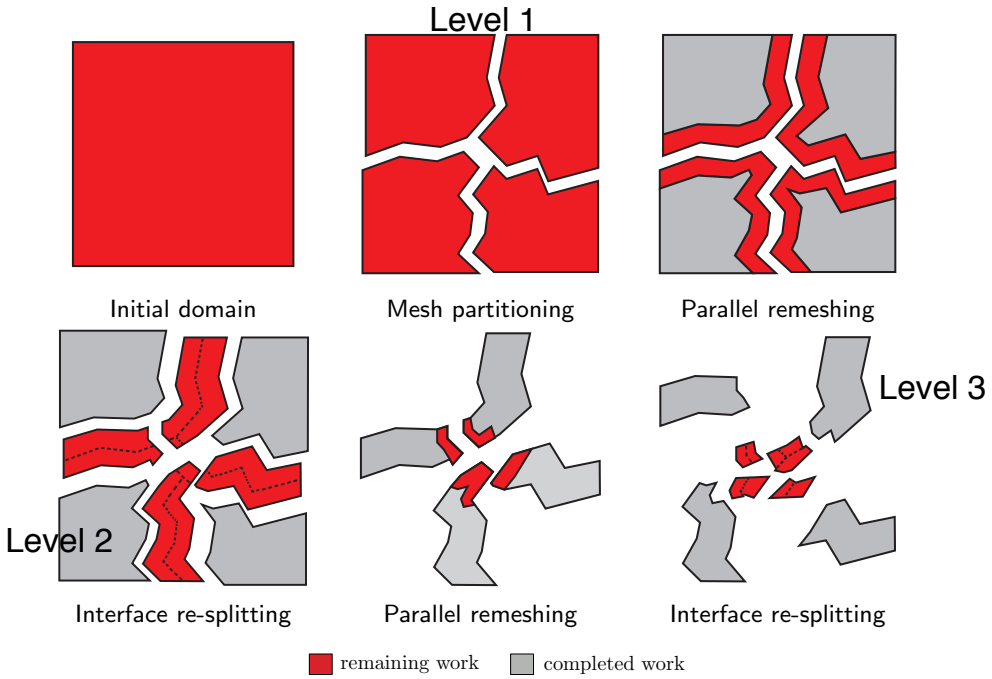


Figure 3.12 – Sketch of the parallel process remeshing. The red-colored parts represent the part of the domain that remains to be adapted while the gray-colored parts are the final adapted parts that can be stored to disk. 3 levels are depicted.

3.5 Hierarchical Domain partitioning

In the context of parallel remeshing, the domain partitioning method must be fast, low memory, able to handle domains with many connected components and effective to balance the remeshing work. Moreover, we should have efficient partitioning method for several hierarchical level of partitions. In particular, the method should be such that the size of the interface between the partitions converges toward zero when the partitioning level increases in order to have a converging parallel algorithm. More precisely, we first - level 1 - split the domain volume. Level 2, we split the interface of the partitions of level 1; the interface volume domain being formed by all the elements having at least one vertex sharing several sub-domains. Level 3, we split the interface of the partitions of level 2, and so on. The different levels for the hierarchical decomposition of a cubic domain into 32 partitions are shown in Fig. 3.13. In this example, we observe that the domain topology varies drastically with the level. We also observe that the size of interface meshes decreases at each level and converges toward zero.

To emphasize the choices made in this work for the hierarchical partitioning method, in this section, we will always compare the proposed methodology on the same example. The considered example is the adaptation of an initial uniform mesh to the numerical solution (at one time step) of a spherical blast problem. We will refer to it as the blast example. The initial uniform mesh is composed of 821 373 vertices and 4 767 431 tetrahedra while the adapted resulting mesh is composed of 82 418 vertices and 511 998 tetrahedra. These two meshes are shown in Fig. 3.14.

It takes 36 seconds to generate that adapted mesh in serial.

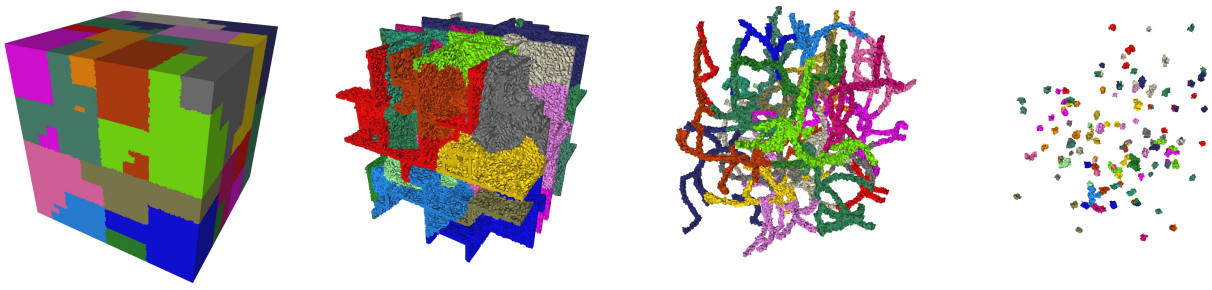
This example is very interesting because the positions of the spherical shock waves imply that a large number of insertions and collapses are needed while being non-uniformly distributed in the domain. Hence, the amount of work and the kind of meshing operation varies drastically in the domain.

3.5.1 Element work evaluation

An effective domain partitioning strategy should balance the work which is going to be done by the local remesher on each partition, knowing that each partition is meshed independently. Thus, there is no communication between partitions and the partition interfaces are constrained (they are not remeshed). The work to be performed depends on the used mesh operations (insertion, collapse, swap, smoothing), the given metric field $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$, and the initial mesh $\mathcal{H}$ natural metric field $(\mathcal{M}_{\mathcal{H}}(\mathbf{x}))_{\mathbf{x}\in\Omega}$, where Ω is the domain to be remeshed. Indeed, if the initial mesh already satisfies the given metric, *i.e.,* $(\mathcal{M}_{\mathcal{H}}(\mathbf{x}))_{\mathbf{x}\in\Omega} = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$, then nothing has to be done. It is convenient to define the work at the element level because it is the elements that are uniquely distributed to each partition. We recall that the natural metric of an element K is the unique metric tensor $\mathcal{M}_K$ such that all edges of K are of length 1 for $\mathcal{M}_K$. It is obtained by solving a simple linear system [11]. And, metric field $(\mathcal{M}_{\mathcal{H}}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ is the union of the element metrics $\mathcal{M}_K$.

To define the remesher work per element and the total work, we use a continuous approach - similarly to the error estimate theory [11, 12] - because the initial mesh and the targeted final adapted mesh are represented by their respective metric fields $(\mathcal{M}_{\mathcal{H}}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ and $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$. We recall that, for a given metric field $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$, also called continuous mesh, the point-wise mesh density is given by $d_{\mathcal{M}}(\mathbf{x}) = \sqrt{\det \mathcal{M}(\mathbf{x})}$ and the continuous mesh complexity is

$$\mathcal{N} = \int_{\mathbf{x}\in\Omega} \sqrt{\det \mathcal{M}(\mathbf{x})}\, \mathrm{d}\mathbf{x} = \int_{\mathbf{x}\in\Omega} d_{\mathcal{M}}(\mathbf{x})\, \mathrm{d}\mathbf{x}\,.$$



Level 1 Level 2 Level 3 Level 4

Figure 3.13 – Hierarchical partitioning into 32 sub-domains of a cubic domain for a constant work per element. From left to right, levels 1, 2, 3 and 4 of partitioning. We observe that the domain topology varies drastically with the level, and the size of interface meshes decreases at each level and converges toward zero.

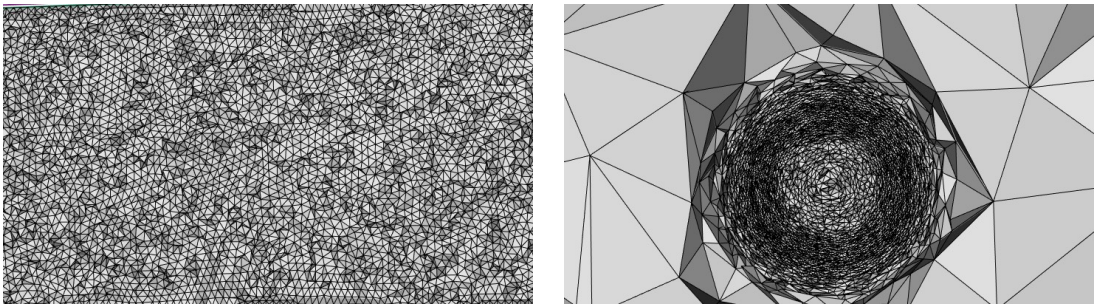


Figure 3.14 – Blast example to assess the hierarchical partitioning techniques and anisotropic work prediction: initial uniform mesh (left) and final adapted mesh (right). The serial adaptation takes 36 seconds.

The continuous mesh complexity $\mathcal{N}$ is the dual of the mesh number of vertices N in the continuous mesh framework [12]. As the work to be done by the local remesher is clearly proportional to the mesh size, in the continuous approach, the work is thus proportional to the integral of the mesh density.

We propose to define the work per element, the total work of the remesher being the sum of the mesh element works. Each element K of initial mesh $\mathcal{H}$ is supplied with its natural metric $\mathcal{M}_K$ and the given metric $\mathcal{M}$. The given metric at the element is obtained by averaging (in the log-Euclidean framework) the metric at its vertices:

$$\mathcal{M} = \exp\left(\sum_{i=1}^{k} \frac{1}{k} \ln(\mathcal{M}(\mathbf{x}_i))\right) \quad \text{where the } \mathbf{x}_i \text{ are the vertices of K}\,,$$

as the given metric field $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ is generally point-wise. We also compute the intersection [Frey 2005] of these two metrics: $\mathcal{M}_\cap = \mathcal{M} \cap \mathcal{M}_K$. We denote by $d_\mathcal{M}$, d_K and $d_\cap$ the density of metric $\mathcal{M}$, $\mathcal{M}_K$ and $\mathcal{M}_\cap$, respectively. Now, we analyze the work depending on specific remeshing case and, then, we propose a work for the general case.

Insertion case. Assuming the initial mesh is only going to be refined (first case in Fig. 3.15), the density of points to be inserted is $d_{\mathcal{M}\setminus\mathcal{M}_K}$ then the work per element is

$$wrk(K) = \alpha\,|K|\,(d_\mathcal{M} - d_K) = \alpha\,|K|\,(d_\cap - d_K)\,,$$

where $|K|$ is the element volume, α is the coefficient defining the cost of the insertion operator, and in that case we have $\mathcal{M}_\cap = \mathcal{M}$. Note that the metric density is inversely proportional to the ellipse area in the graphic representation. If we assume that $d_\mathcal{M} = d_\cap \gg d_K$, then $wrk(K) \approx \alpha\,|K| d_\mathcal{M}$. Thus, the total work will be:

$$wrk(\mathcal{H}) = \sum_{K\in\mathcal{H}} \alpha\,|K|\,d_\mathcal{M} = \alpha \int_{\mathbf{x}\in\Omega} d_\mathcal{M}(\mathbf{x})\,\mathrm{d}\mathbf{x} = \alpha\mathcal{N}^{new}\,,$$

which is logical because in that case the work is effectively proportional to the size of the final mesh. Let us give a concrete example in three dimensions. We have a uniform isotropic mesh of

size $\mathcal{N}$ with length size h thus $\mathcal{M}_K = h^{-2}\,\mathcal{I}_3$ for all elements K. We want to generate a mesh at $h/2$, thus $\mathcal{M} = 4\,h^{-2}\,\mathcal{I}_3$. This new mesh will have a size of $8\mathcal{N}$. The work per element is $wrk(K) = 7\,\alpha\,|K|\,h^{-3}$ leading to a total work of $wrk(\mathcal{H}) = 7\,\alpha\,\mathcal{N}$. This result is the expected answer as $7\mathcal{N}$ vertices will be inserted to generate the new mesh.

Collapse case. Assuming the initial mesh is only going to be collapsed (second case in Fig. 3.15), the density of points to be removed is $d_{\mathcal{M}_K \backslash \mathcal{M}}$ then the work per element is

$$wrk(K) = \beta\,|K|\,(d_K - d_{\mathcal{M}}) = \beta\,|K|\,(d_\cap - d_{\mathcal{M}})\,,$$

where β is the coefficient defining the cost of the collapse operator, and in that case we have $\mathcal{M}_\cap = \mathcal{M}_K$. If we assume that $d_K = d_\cap \gg d_{\mathcal{M}}$, then $wrk(K) \approx \beta\,|K| d_K$. Thus, the total work will be:

$$wrk(\mathcal{H}) = \sum_{K\in\mathcal{H}} \beta\,|K|\,d_K = \beta \int_{\mathbf{x}\in\Omega} d_{\mathcal{H}}(\mathbf{x})\,\mathrm{d}\mathbf{x} = \alpha\mathcal{N}\,,$$

which is logical because in that case the work is effectively proportional to the size of the initial mesh.

Optimization case. At the end of the remeshing process, an optimization phase is applied to improve the mesh quality. Thus, the work due to the optimization is proportional to the size of the final mesh:

$$wrk(K) = \gamma\,|K|\,d_{\mathcal{M}}\,,$$

where γ is the coefficient defining the cost of the optimization operator.

General case. In the general case, we may have to insert and collapse locally because the anisotropic information may be contradictory depending on the considered direction. For instance, two metrics may have the same density but opposite directions hence in one direction we should refine the mesh and in the other direction we should coarsen the mesh (third case in Fig. 3.15). The general case gathers all the previous cases and the density of the intersected metric represents the common ground. The work per element is:

$$wrk(K) = |K|\,\left(\alpha\,(d_\cap - d_K) + \beta\,(d_\cap - d_{\mathcal{M}}) + \gamma\,d_{\mathcal{M}}\right)\,.$$

The constants depend on the underlying remesher properties. In our case, the local remeshing strategy uses a unique cavity operator for all mesh modifications (see Chapter 2), therefore all mesh modifications have exactly the same cost. We thus set: $\alpha = \beta = \gamma = 1$, the work per element becomes:

$$wrk(K) = |K|\,\left(2\,d_\cap - d_K - d_{\mathcal{M}} + d_{\mathcal{M}}\right)\,, \tag{3.2}$$

or if no optimization step is performed $\gamma = 0$, thus:

$$wrk(K) = |K|\,\left(2\,d_\cap - d_K - d_{\mathcal{M}}\right)\,.$$

If we are in the case where only refinements are performed, at a first order approximation, we can assume that $d_{\mathcal{M}} = d_\cap \gg d_K$ and the work per element is:

$$wrk(K) \;\approx\; |K|\,(2\,d_\cap + (\gamma - 1)d_{\mathcal{M}}) \;=\; |K|\,(1+\gamma)\,d_{\mathcal{M}}\,.$$

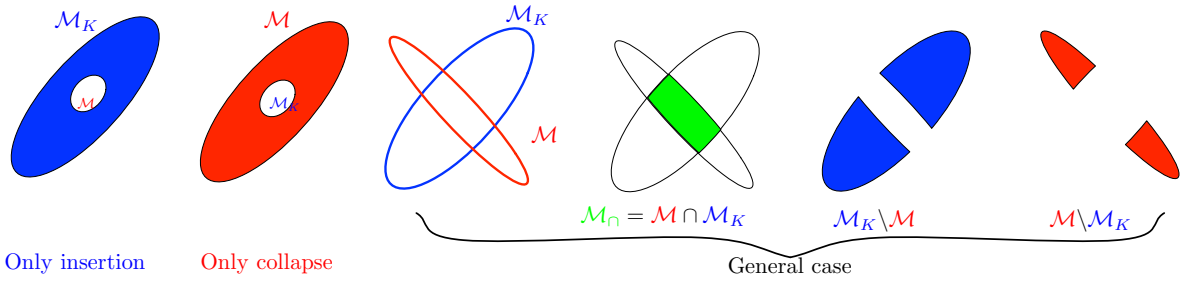


Figure 3.15 – Illustration of the continuous work in the metric-based framework. Blue regions represent insertion work. Red regions represent collapse work. Note that the metric density d is inversely proportional to the ellipse area in the graphic representation.

This means that the work without optimization is proportional to the work with optimization. In that case, it changes nothing to take into account the work due to the optimization to define the work per element.
If we are in a case where only collapses is performed, at a first order approximation, we can assume that $d_K = d_\cap \gg d_\mathcal{M}$, thus the element work is:

$$wrk(K) \;\approx\; |K|\,(2\,d_\cap - d_K) \;=\; |K|\,d_K\,.$$

Thus, the work without optimization is equal to the work with optimization and it changes nothing to take into account the work due to the optimization to define the work per element. This simple analysis shows why it is very important to choose a simple example that involves the general case (and not only coarsening or refinement) to validate the obtained work function. Indeed, in the general case, it is primordial to take into account the cost of the mesh optimization to have well-balanced partitions. This is why the blast example has been chosen.

Surface work. When the surface is remeshed with the volume, the work to remesh the surface is added to the work to remesh the volume. Therefore, the same formula is used to estimate the work per face by taking into account the surface metric and the surface density. Then, the work of each face is added to the work of the element sharing that face.

Blast example. We first illustrate on the blast example why it is crucial to take into account the metric of the initial mesh and the given metric to define the remesher work (as we have done in this section), and not to use - as frequently observed - only the given metric (which is valid if only insertion is done, see previous remark). The results obtained on 8 processors with the anisotropic work given by Relation (3.2) and the work only based on the density of the given metric $\mathcal{M}$ are given in Table 3.1. In both cases, the Hilbert partitioning method is used. The anisotropic work leads to a quasi-uniform CPU time for each of the 8 partitions whatever the number of collapses or insertions. On the contrary, considering only the given metric density to compute the work to balance the partitions leads to a completely non-uniform CPU time. In fact, we have balanced the number of insertions on each partition ($\approx 8,600$ per partitions) which is expected if only $d_\mathcal{M}$ is used. But, the collapses have not been taken into account. It

results in a larger maximal time and a large overhead in waiting for the end of the remeshing of the most time-consuming partition.

Anisotropic work	Statistics for each partition								Waiting time
Cpu time (sec)	5.1	4.8	4.7	4.6	4.6	4.7	4.6	4.6	0.5
# of collapses	78 844	95 615	96 457	81 437	91 787	85 754	96 020	83 763	
# of insertions	14 029	4 952	4 299	12 667	7 121	10 058	4 586	11 103	
Density-based work	**Statistics for each partition**								**Waiting time**
Cpu time (sec)	3.9	4.2	6.7	2.4	7.0	0.9	9.6	3.4	8.7
# of collapses	63 047	73 404	133 349	33 764	138 473	2 801	205 677	59 218	
# of insertions	8 626	8 530	8 526	8 614	8 605	8 604	8 585	8 598	

Table 3.1 – Remeshing statistics on 8 processors for the blast problem test case (Fig. 3.14). The total CPU time, the number of collapses and the number of insertions for each partition are presented. Top and bottom tables show the statistics for a mesh partitioning based on the anisotropic work (Relation (3.2)) and on the given metric density-based work, respectively. In both cases, the Hilbert partitioning method is used.

3.5.2 Partitioning methods

Before using any of the partitioning methods presented below, the mesh vertices are first renumbered using a Hilbert space filling curve based reordering [59]. A Hilbert index (the position on the curve) is associated with each vertex according to its position in space. This operation has a linear complexity and is straightforward to parallelize as there is no dependency. Then, the vertices renumbering is deduced from the vertices Hilbert indices. Vertices are sorted using the standard C-library `quicksort`.

The domain partitioning problem can be viewed as a renumbering problem of the elements. In that case, the first partition is composed of the elements from 1 to N_1 such that the sum of these elements work is equal to the total mesh work divided by the total number of partitions. Then, the second partition is composed of the elements from N_1+1 to N_2 such that that the sum of these elements work is equal to the total mesh work divided by the total number of partitions. And so on. The difference between all strategies lies on the choice of the renumbering strategy. Note that, for efficiency purposes, the elements are not explicitly reordered but they are only assigned an index or a partition index on the fly.

Now, assuming the vertices have been renumbered, we propose three methods to split the mesh: Hilbert based, breadth-first search (BFS) or frontal approach, and BFS with restart.

Hilbert partitioning. It consists in ordering the elements list according to the element minimal vertex index. In other words, we first list the elements sharing vertex 1 (the ball of vertex 1), then we list the elements sharing vertex 2 (the ball of vertex 2 not already assigned), etc. This splitting of the domain is based on the Hilbert renumbering of the vertices. For level 1 domain (initial domain splitting), it results in block-shaped partitions which is very convenient for subdomain remeshing (see Fig. 3.16 (c)). But, it may lead to partitions with several connected components on complex geometry due to domain holes not seen by the Hilbert curve. For level 2 or more

domains, it is not effective because it will reproduce the previous level result and thus it will not gather the interfaces of different sub-domains. The size of the interface mesh will not decrease at each level.

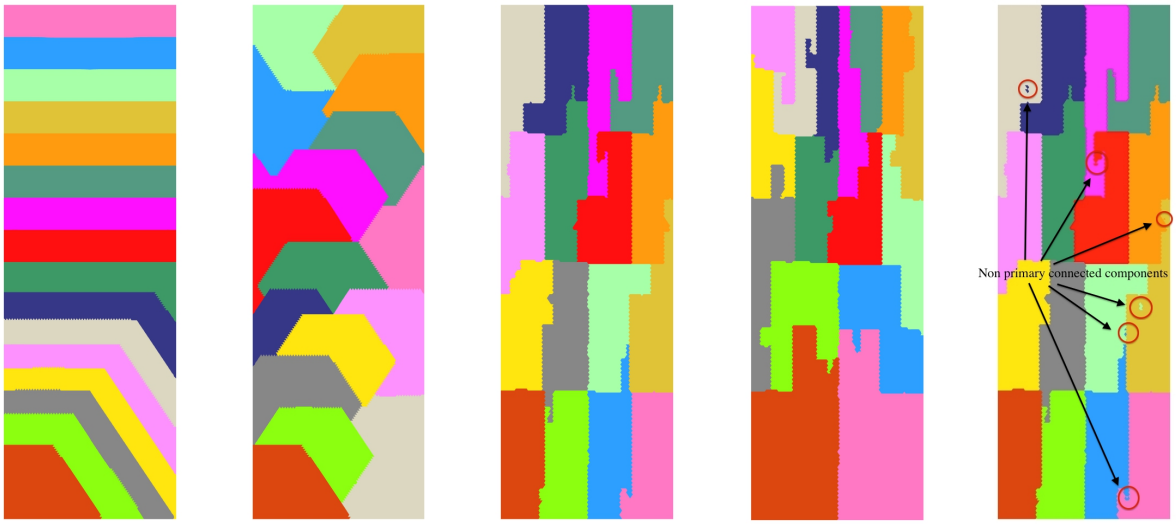
Breadth-first search (BFS) partitioning. This method starts from an element seed - generally, element 1 - and adds the neighbor elements of the seed first, *i.e.,* the neighbors are the next elements in the renumbered list. Then, we move to the next level of neighbors, in other words, we add the neighbors of the neighbors not already assigned. And so on. This splitting of the domain progresses by front. Indeed, each time an element is assigned, its non-assigned neighbors are added to a stack. The elements in this stack represent the current front. For level 1 domain, it results in layered partitions which contains only one connected component (see Fig. 3.16 (a)) except the last one(s) which could be multi-connected. But, it results in several unconnected interface domains at level 2 which is not appropriate here. For level 2 or more domains, this method is able to gather the interfaces of different sub-domains but, as the stack is always growing, the number of connected components grows each time a bifurcation is encountered (see Fig. 3.17 (a)). This leads to very unbalanced sub-domains after the connected component correction presented below. Therefore, we prefer to consider the modified BFS method described hereafter.

Breadth-first search (BFS) with restart partitioning. In the previous BFS algorithm, the splitting progresses by front, and generally this front grows until it reaches the diameter of the domain. During the splitting of interface domains (level 2 or more), this is problematic because the resulting partitions are multi-connected, cf. Fig. 3.17 (a). One easy way to solve this issue is to reset the stack each time we deal with a new partition. The seed of the new partition is the first element of the present stack, all the other elements are removed from the stack. For level 1 domain, it results in more circular (spherical) partitions (see Fig. 3.16 (b)). For level 2 or more domains, this method is able to gather the interfaces of different sub-domains and also to obtain one connected component for each partition except the last one(s), see Fig. 3.17 (c). Therefore, this method is very efficient to deal with the level 2 or higher domains of the hierarchical partitioning. Moreover, we observe in Fig. 3.13 that the size of the partition interface meshes reduces at each level.

Connected components correction. As the interface are constrained and not remeshed, the number of connected components per subdomain should be minimized to maximize the work done by the remeshing strategy. In other words, each partition should have only one connected component if possible. All elements of the same connected component are linked by at least a neighboring face.

After the domain splitting, a correction is applied to merge isolated connected components, see Fig. 3.16 (e). First, for each subdomain, the number of connected components is computed and the primary connected component (the one with the most work) of each partition is flagged. Second, we compute the neighboring connected components of each non-primary connected component. Then, iteratively, we merge each non-primary connected component with a neighboring primary connected component. If several choices occur, we pick the primary connected component with the smallest work. The impact of this correction is illustrated in Fig. 3.16 from (e) to (c).

We may end up with non-manifold (but connected) partitions, *i.e.*, elements are linked by a



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

Figure 3.16 – Partitioning into 16 sub-domains of a - level 1 - rectangular domain for a constant work per element with the BFS (a), BFS with restart (b) and Hilbert (c) methods. Picture (d) shows the Hilbert partitioning with a linear work function (the work per element increase with y) which has to be compared with picture (c) for a constant work per element. Picture (e) shows the Hilbert partitioning before the connected components correction.

vertex or an edge. As the local remeshing strategy is able to take care of such configurations, no correction is applied. Otherwise, such configurations should be detected and corrected.

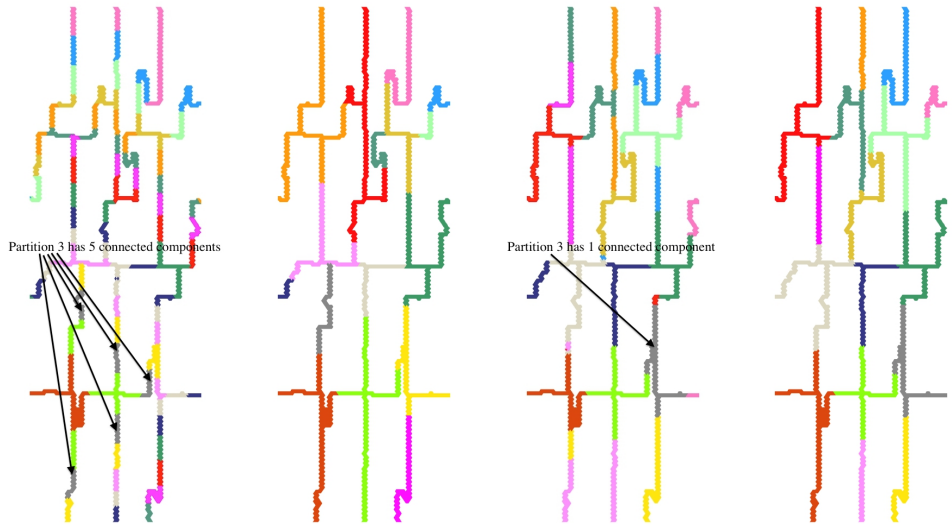
Blast example. We compare, on the blast example, the efficiency of the proposed partitioning methods on the level 1 and level 2 meshes.

For the level 1 partitioning, we first analyze the size of the interface provided by each method. Indeed, the best method should minimize the number of interface faces as these faces are constrained and prevent the remesher to work. The result is given in Table 3.2. Clearly, the Hilbert method minimizes the number of interface faces (by a factor two w.r.t. the BFS method), and it also somehow balances the number of interface faces between the partitions. This should have an impact on the efficiency. Second, we analyze the CPU time and the number of operations done with each method, the result is presented in Table 3.3. The Hilbert method achieves the

Method	Number of interface faces for level 1 partitions								Total	Max difference
Hilbert	18 043	19 408	21 870	16 849	19 580	19 268	22 256	16 270	153 542	5 986
BFS	14 550	34 115	41 550	45 883	50 885	56 579	47 350	17 758	308 670	42 029
BFS restart	14 532	26 615	23 710	11 344	37 407	33 773	30 290	23 683	201 334	26 063

Table 3.2 – Interface size between level 1 partitions on 8 processors for the blast problem test case (FIg. 3.14): the number of interface faces - which are constrained faces for the remesher - for each level 1 partition are given. The three partitioning methods are compared.

lowest maximal CPU time and minimizes the waiting time between the partitions. It leads to a quasi-uniform CPU times for each of the 8 partitions whatever the number of collapses or



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

Figure 3.17 – Partitioning into 16 sub-domains of a - level 2 - interface mesh of a rectangular domain for a constant work per element. The interface mesh results from the Hilbert partitioning of the level 1 domain. Partitions obtained with the BFS method before and after correction are shown in pictures (a) and (b), respectively. Partitions obtained with the BFS method before and after correction are shown in pictures (c) and (d), respectively.

Hilbert method	Statistics for each level 1 partition								Waiting time
Cpu time (sec)	5.1	4.8	4.7	4.6	4.6	4.7	4.6	4.6	0.5
# of collapses	78 844	95 615	96 457	81 437	91 787	85 754	96 020	83 763	
# of insertions	14 029	4 952	4 299	12 667	7 121	10 058	4 586	11 103	

BFS method	Statistics for each level 1 partition								Waiting time
Cpu time (sec)	4.6	5.0	5.1	5.4	5.9	5.9	5.5	4.8	1.3
# of collapses	106 496	100 976	50 380	42 465	73 217	98 160	97 906	102 908	
# of insertions	1	582	25 726	29 666	12 360	0	0	1	

BFS restart method	Statistics for each level 1 partition								Waiting time
Cpu time (sec)	5.2	5.5	5.4	4.6	5.3	5.0	5.4	4.8	0.9
# of collapses	106 508	103 028	93 868	3 702	92 508	103 594	101 370	94 954	
# of insertions	3	515	5 474	54 978	7 331	1	0	0	

Table 3.3 – Remeshing statistics on 8 processors for the blast problem test case (Fig. 3.14) on the level 1 mesh. The total CPU time, the number of collapses and the number of insertions for each partition are presented. From top to bottom, tables show the statistics for the Hilbert, the BFS, and the BFS with restart methods.

insertions.

Now, we analyze the results for the level 2 partitioning where the level 1 partitioning has been done with the Hilbert method. The size of the interface provided by each method is given in Table 3.4. As previously mentioned, the Hilbert method does not reduce the interface on the subsequent level and it is thus not appropriate. We notice that the BFS with restart is clearly better than the BFS, and the number of interface faces drop from 153 542 on the level 1 to 25 682 on the level 2. As regards the efficiency, the BFS with restart achieves the best CPU time and minimize the waiting time, see Table 3.5.

The following strategy is thus proposed for the hierarchical mesh partitioning. The Hilbert method is used to partition the initial volume mesh, *i.e.,* the level 1 partitioning, as it is very efficient, it ends-up with well-balanced partitions, and it minimizes the size of the interface. Moreover, the Hilbert partitioning of the initial volume mesh provides a nice continuous network-shaped domain (see Figs. 3.13 and 3.16) for the interface mesh which is very convenient for the subsequent partitioning levels.

Method	Number of interface faces for level 2 partitions								Total	Max difference
Hilbert	18 934	21 583	22 378	16 956	19 843	19 766	23 040	16 300	158 794	6 084
BFS	3 013	5 476	6 129	8 422	9 492	8 563	4 737	910	46 742	8 582
BFS restart	3 002	4 585	5 288	5 602	2 036	1 390	1 408	2 373	25 682	4 212

Table 3.4 – Interface size between level 2 partitions on 8 processors for the blast problem test case (Fig. 3.14): the number of interface faces - which are constraint faces for the remesher - for each level 1 partition are given. The three partitioning methods are compared.

Hilbert method	Statistics for each level 2 partition								Waiting time
Cpu time (sec)	0.97	0.96	0.95	0.90	0.88	0.94	0.99	0.90	0.11
# of collapses	0	264	47	84	16	8	118	27	
# of insertions	2 488	1,617	1 789	2 335	2 000	2 117	1 506	2 145	

BFS method	Statistics for each level 2 partition								Waiting time
Cpu time (sec)	0.44	0.54	0.60	0.67	0.70	0.68	0.71	0.41	0.30
# of collapses	10 074	5 817	1 860	1 568	1 659	1 357	5 026	0	
# of insertions	0	852	2 516	2 356	2 281	2 466	2 801	2 529	

BFS restart method	Statistics for each level 2 partition								Waiting time
Cpu time (sec)	0.47	0.57	0.54	0.51	0.55	0.52	0.53	0.53	0.10
# of collapses	10 074	6 122	4 763	9 160	603	2	0	76	
# of insertions	0	876	1 398	17	3 099	3 616	3 950	3 100	

Table 3.5 – Remeshing statistics on 8 processors for the blast problem test case (Fig. 3.14) on the level 2 mesh. The total CPU time, the number of collapses and the number of insertions for each partition are presented. From top to bottom, tables show the statistics for the Hilbert, the BFS, and the BFS with restart methods.

3.5.3 Partitions balancing optimization by migration

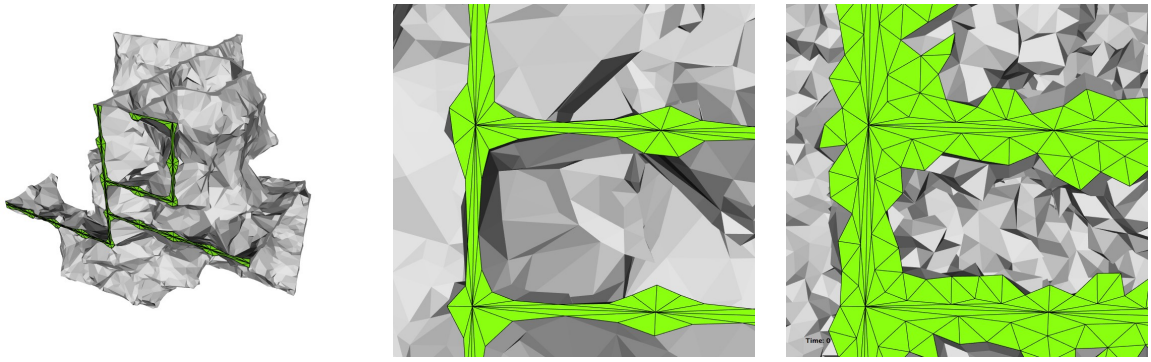
On some complex configurations, the connected components correction leads to unbalanced partitions because the size of the non-primary connected components is non-negligible. The partition balancing is then optimized by migrating elements between neighboring partitions. To this end, each element is analyzed and if it has a neighboring element on another partition which has a lower total work, then these elements are migrated to that partition. This optimization phase improves the partitions balancing but may create new connected components for partitions, thus the correction presented in the previous section is again applied. This process is iterated until the partitions are well-balanced with respect to the given work.

3.5.4 Efficiency of the method

The presented domain partitioning methods minimize the memory requirement as the data structures they use are only: the elements list, the elements' neighbors list, the elements' partitions indices list and a stack. They are efficient in CPU because the elements assignment to a subdomain is done in one loop over the elements. Then, the connected components correction requires only a few loops over the partitions. For instance, let us consider the domain partitioning of a cubic domain composed of 10 million tetrahedra into 64 sub-domains. In serial on a Intel Core i7 at 2.7Ghz, it takes 0.52, 0.24 and 0.24 seconds for the partitioning of the level 1, 2 and 3 domains, respectively, where the Hilbert partitioning has been used for level 1 domain and the BFS with restart partitioning has been used for the level 2 and 3 domains.

3.5.5 Definition of the interface mesh

During the remeshing phase, the set of elements that surrounds the constrained faces (defining the boundary of the current partition) is not adapted. It is then necessary to define a set of elements that needs to be adapted at the next iteration (or level). An initial choice consists in introducing all the elements having at least one node on the boundary of the interface. This choice is illustrated in Fig. 3.18 (middle). Despite its simplicity, this choice is appropriate only when the size of the elements of the interface is of the same order as the size imposed elsewhere. However, when large size variation occurs, additional elements need to be part of the new interface volume. Optimally, a sufficient number of elements needs to be added to make sure that underlying local modification will be possible at the next level. An automatic way to find these elements is to add the relevant set of elements of the cavity [45] for each operator. Two situations occur. When an edge of the interface is too short, a collapse will be needed at the next level. Consequently, for all interfaces sharing this edge, the ball of the two end-points edges is added. When an edge is too long, a point will be inserted at the next level, consequently, the Delaunay cavity of the mid-point edge is added. Note that these modifications are done in parallel at the end of the remeshing step, thus limiting the overhead of this correction. The modification of the set of elements defining the interface is illustrated in Fig. 3.18 where a cubic domain is refined from a size h to $h/4$. If we select only the balls of the interface vertices, then the remeshing process is much more constrained, see Fig. 3.18 (middle). Including additional elements based on the cavity defining the relevant mesh modification operator (collapse or insertion) gives additional room to the mesh generator to perform a quality modification, Fig. 3.18 (right).



Interface | No correction | Cavity-based correction

Figure 3.18 – Definition of the interface mesh on a cube example. Global view of interface geometry (left), interface defined by the balls of the vertices belonging to the interface (middle), and interface mesh defined by predicting the set of elements needed to perform the remeshing operation (insertion or collapse).

3.6 Numerical Results

Several examples are illustrated in this section. For each case, the parallel mesh generation converges in 5 iterations. The number of cores is chosen to ensure that at least 100 000 tetrahedra per core will be inserted. Consequently, the number of cores is reduced when the remaining work decreases. All the examples are run on a cluster composed of 40 nodes with 48Gb of memory, composed of two-chip Intel Xeon X56650 with 12 cores. A high-speed internal network InfiniBand (40Gb/s) connects these nodes. For each example, we report the complete CPU time including the IOs, the initial partitioning and gathering along with the parallel remeshing time. To evaluate the overheads and efficiency of the parallel method, serial meshes are generated on a super-node having 1 Tb of memory.

Vortical flows on the F117 geometry. This case is part of an unsteady adaptive simulation to accurately capture vortices generated by the delta-shaped wings of the F117 geometry, see Fig. 3.19. The final adapted mesh of the simulation is depicted in Fig. 3.19. The final adapted mesh is composed of 83 752 358 vertices, 539 658 triangles and 520 073 940 tetrahedra. The initial background mesh is composed of 1 619 947 vertices, 45740 triangles and 9 710 771 tetrahedra. The complete CPU time (including initial domain partitioning and final gathering) is 12 min on 120 cores. The parallel mesh adaptation of the process takes 8 min 50 s. The parallel procedure inserts 10^6 vertices/min or equivalently $6 \,.\, 10^6$ tetrahedra/min, see Table 3.7. The maximal memory used per core is 1.25 Gb. The same example on 480 cores is reported in Table 3.8, the CPU for the parallel mesh generation part is 3 min 36 s while the maximal memory used per core is 0.6Gb. The speed up from 120 to 480 cores is limited to 1.5 (4 optimally), this is due to the large increase of the interfaces in the mesh, see Table 3.9 (left). For a partition, the typical time to create its interface mesh using the anisotropic Delaunay cavity is less than 10% of the meshing time.

The quality of the mesh along with the histogram of the length of edges is reported in

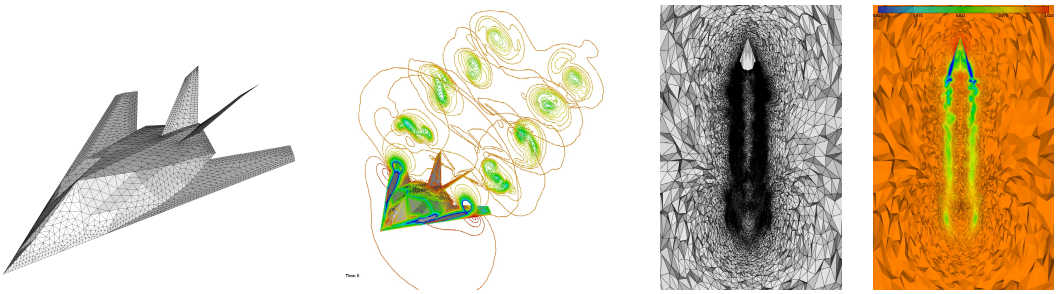


Figure 3.19 – F117 test case. From left to right, geometry of the f117 aircraft, representation of the vortical flow, top view of the mesh adapted to the local Mach number and local Mach number iso-values.

Table 3.6. More than 94% of edges have a unit length *w.r.t.* the metric. The serial CPU time for this case is 4h with an equivalent distribution of lengths and qualities. The mesh generated in serial is composed of 81 920 668 vertices, 510 052 triangles and 493 948 440 tetrahedra. For the complete process, we obtain a speed-up of 20 on 120 cores.

Blast simulation on the tower bridge. The example consists in computing a blast propagation on the London Tower Bridge. The geometry is the 23rd IMR meshing contest geometry. The initial mesh is composed of 3 837 269 vertices 477 852 triangles and 22 782 603 tetrahedra while the final mesh is composed of 174 628 779 vertices 4 860 384 triangles and 1 090 324 952 tetrahedra. From the previous example, the surface geometry and mesh adaptation is much more complex as many shock waves impact the bridge. The time to generate the adapted mesh on 120 cores is 22 min 30 s and 28 min for the total CPU time including the initial splitting, final gathering and IOs. On 480 cores, the time to generate the mesh reduces to 16 min 30 s. The maximal memory used on 120 cores is 1.8Gb and reduces to 1Gb on 480 cores. We report in Tables 3.9 (right), 3.10 and 3.11, the convergence of the process. This example exemplifies the robustness of this approach with complex geometries.

The quality of the mesh along with the histogram of the length of edges is reported in Table 3.12. As for as the F117 test case, more than 94% of edges have a unit length in the metric. The serial CPU time for this case is 10 h 40 min with an equivalent distribution of length and qualities. The mesh generated in serial is composed of 183 761 201 vertices, 4 572 302 triangles and 1 102 880 450 tetrahedra. For the complete process, we obtain a speed-up of 30 on 120 cores. In comparison with the F117 test case, the improvement of the speed-up is related to the serial approach that tends to be less efficient when the size of the meshes increases. The use of the parallel out-of-core approach tends to minimize this memory effect as the complete mesh is never stored on memory.

Landing gear geometry mesh refinement. This geometry is designed for the study of the propagation of the noise generated by a landing gear. This simulation requires large isotropic surface and volume meshes to capture the complex flow field which is used for aeroacoustic analysis. The initial background mesh is composed of 2 658 753 vertices 844 768 triangles and 14 731 068 tetrahedra while the adapted mesh is composed of 184 608 096 vertices 14 431 356

Distribution of $\ell_{\mathcal{M}}$ for edges				
0.00	$< \ell_{\mathcal{M}} <$	0.20	24 815	0.00 %
0.20	$< \ell_{\mathcal{M}} <$	0.50	555 487	0.09 %
0.50	$< \ell_{\mathcal{M}} <$	0.71	2 577 068	0.41 %
0.71	$< \ell_{\mathcal{M}} <$	0.90	181 600 710	**28.63** %
0.90	$< \ell_{\mathcal{M}} <$	1.11	224 256 106	**35.35** %
1.11	$< \ell_{\mathcal{M}} <$	1.41	195 459 095	**30.81** %
1.41	$< \ell_{\mathcal{M}} <$	2.00	25 649 625	4.04 %
2.00	$< \ell_{\mathcal{M}} <$	5.00	4 141 743	0.65 %
5.00	$< \ell_{\mathcal{M}}$		68 602	0.01 %

Distribution of $Q_{\mathcal{M}}$ for triangles				
1	$< Q_{\mathcal{M}} <$	2	495 271	**99.45** %
2	$< Q_{\mathcal{M}} <$	3	1 557	0.31 %
3	$< Q_{\mathcal{M}} <$	4	535	0.11 %
4	$< Q_{\mathcal{M}} <$	5	299	0.06 %
5	$< Q_{\mathcal{M}} <$	6	152	0.03 %
6	$< Q_{\mathcal{M}} <$	7	73	0.01 %
7	$< Q_{\mathcal{M}} <$	8	55	0.01 %
8	$< Q_{\mathcal{M}} <$	9	24	0.00 %
9	$< Q_{\mathcal{M}} <$	10	11	0.00 %
10	$< Q_{\mathcal{M}} <$	100	39	0.01 %

Distribution of $Q_{\mathcal{M}}$ for tetrahedra				
1	$< Q_{\mathcal{M}} <$	2	467 446 828	**94.15**%
2	$< Q_{\mathcal{M}} <$	3	14 064 980	2.83%
3	$< Q_{\mathcal{M}} <$	4	4 054 558	0.82%
4	$< Q_{\mathcal{M}} <$	5	2 333 665	0.47%
5	$< Q_{\mathcal{M}} <$	6	1 556 518	0.31%
6	$< Q_{\mathcal{M}} <$	7	1 118 945	0.23%
7	$< Q_{\mathcal{M}} <$	8	842 774	0.17%
8	$< Q_{\mathcal{M}} <$	9	657 686	0.13%
9	$< Q_{\mathcal{M}} <$	10	524 793	0.11%
10	$< Q_{\mathcal{M}} <$	100	3 773 750	0.76%
100	$< Q_{\mathcal{M}} <$	1000	130 948	0.03%
1000	$< Q_{\mathcal{M}} <$	10000	2 256	0.00 %

Table 3.6 – F117 test case on 120 cores: Distribution of the length of edges in the metric, histograms of the quality of surface and volume elements in the metric.

Iteration	% done	# in interface	# inserted	CPU time	# of cores	elt/sec	elt/sec/core
1	84%	69 195 431	433 495 495	180.8	120	$2.4\,10^6$	19 980
2	96%	1 692 739	502 706 732	95.0	120	$7.2\,10^5$	6 071
3	99%	1 231 868	518 850 149	35.9	91	$4.6\,10^5$	5 068
4	99%	6459	520 067 586	7.5	7	$1.6\,10^5$	2 318
5	100%	0	520 073 940	1.7	1	$3.7\,10^3$	3 737

Table 3.7 – F117 test case on 120 cores. Table gathering the number of tetrahedra in the interface and the number of inserted tetrahedra along with the CPU time in second for each iteration.

Iteration	% done	# in interface	# inserted	CPU time	# of cores	elt/sec	elt/sec/core
1	76%	109 269 782	389 476 861	109.9	480	$3.5\,10^6$	7 383
2	91%	42 836 303	486 695 293	67.0	480	$1.4\,10^6$	1 440
3	98%	5 567 744	525 073 846	28.1	228	$1.3\,10^6$	6 011
4	99%	32292	530 573 260	8.9	30	$6.1\,10^5$	20 597
5	100%	0	530 605 308	2.3	1	$1.4\,10^4$	13 933

Table 3.8 – F117 test case on 480 cores. Table gathering the number of tetrahedra in the interface and the number of inserted tetrahedra along with the CPU time in second for each iteration.

Iteration	120 cores	480 cores
1	590 038	954 166
2	1 711 512	4 306 256
3	130 262	589 532
4	869	4 018
5	0	0

Iteration	120 cores	480 cores
1	1 081 246	1 627 846
2	2 416 840	5 265 939
3	132 659	451 355
4	488	3 230
5	0	0

Table 3.9 – Number of faces at the interfaces at each iteration when running on 120 and 480 cores for the F117 (left) and the tower bridge (right) test cases.

Iteration	% done	# in interface	# inserted	CPU time	# of cores	elt/sec	elt/sec/core
1	84%	89 577 773	919 345 377	577.3	120	$1.5\,10^6$	13 277
2	95%	14 290 245	1 062 994 802	280.7	120	$5.1\,10^5$	4 264
3	97%	1 290 855	1 089 035 610	56.3	120	$4.6\,10^5$	3 854
4	97%	3636	1 090 321 352	8.0	7	$1.6\,10^5$	22 959
5	100 %	0	1 090 324 952	2.1	1	$1.7\,10^3$	1 714

Table 3.10 – Tower-bridge test case on 120 cores. Table gathering the number of tetrahedra in the interface and the number of inserted tetrahedra along with the CPU time in second for each iteration.

Iteration	% done	# in interface	# inserted	CPU time	# of cores	elt/sec	elt/sec/core
1	79%	193 529 057	922 145 088	255.8	480	$3.6\,10^6$	7 510
2	93 %	52 837 674	1 115 428 211	106.7	379	$1.8\,10^6$	4 779
3	96%	4 258 411	1 165 096 167	34.6	282	$1.4\,10^6$	5 090
4	97%	27 095	1 169 283 585	23.0	23	$1.8\,10^5$	7 915
5	100%	0	1 169 310 260	3.9	1	$6.8\,10^3$	6 839

Table 3.11 – Tower-bridge test case on 480 cores. Table gathering the number of tetrahedra in the interface and the number of inserted tetrahedra along with the CPU time in second for each iteration.

Distribution of $\ell_{\mathcal{M}}$ for edges				
0.00	$< \ell_{\mathcal{M}} <$	0.20	120 421	0.01 %
0.20	$< \ell_{\mathcal{M}} <$	0.50	1 489 489	0.11 %
0.50	$< \ell_{\mathcal{M}} <$	0.71	5 406 707	0.38 %
0.71	$< \ell_{\mathcal{M}} <$	0.90	394 284 285	**27.80** %
0.90	$< \ell_{\mathcal{M}} <$	1.11	496 597 258	**35.02** %
1.11	$< \ell_{\mathcal{M}} <$	1.41	444 060 651	**31.31** %
1.41	$< \ell_{\mathcal{M}} <$	2.00	66 636 395	4.70 %
2.00	$< \ell_{\mathcal{M}} <$	5.00	9 496 734	0.67 %
5.	$< \ell_{\mathcal{M}}$		110 093	0.01 %

Distribution of $Q_{\mathcal{M}}$ for triangles				
1	$< Q_{\mathcal{M}} <$	2	4 519 374	**98.95** %
2	$< Q_{\mathcal{M}} <$	3	19 900	0.44 %
3	$< Q_{\mathcal{M}} <$	4	7 661	0.17 %
4	$< Q_{\mathcal{M}} <$	5	4 693	0.10 %
5	$< Q_{\mathcal{M}} <$	6	3 209	0.07 %
6	$< Q_{\mathcal{M}} <$	7	2 169	0.05 %
7	$< Q_{\mathcal{M}} <$	8	1 648	0.04 %
8	$< Q_{\mathcal{M}} <$	9	1 276	0.03 %
9	$< Q_{\mathcal{M}} <$	10	976	0.02 %
10	$< Q_{\mathcal{M}} <$	100	6 352	0.14 %
100	$< Q_{\mathcal{M}} <$	1000	48	0.00 %

Distribution of $Q_{\mathcal{M}}$ for tetrahedra				
1	$< Q_{\mathcal{M}} <$	2	1 040 413 376	**93.86** %
2	$< Q_{\mathcal{M}} <$	3	32 540 901	2.94 %
3	$< Q_{\mathcal{M}} <$	4	9 077 486	0.82 %
4	$< Q_{\mathcal{M}} <$	5	5 316 550	0.48 %
5	$< Q_{\mathcal{M}} <$	6	3 618 423	0.33 %
6	$< Q_{\mathcal{M}} <$	7	2 643 492	0.24 %
7	$< Q_{\mathcal{M}} <$	8	2 021 664	0.18 %
8	$< Q_{\mathcal{M}} <$	9	1 597 095	0.14 %
9	$< Q_{\mathcal{M}} <$	10	1 294 143	0.12 %
10	$< Q_{\mathcal{M}} <$	100	9 691 925	0.87 %
100	$< Q_{\mathcal{M}} <$	1000	297 559	0.03 %
1000	$< Q_{\mathcal{M}} <$	10000	2 950	0.00 %

Table 3.12 – Tower bridge test case on 120 cores: Distribution of the length of edges in the metric, histograms of the quality of surface and volume elements in the metric.

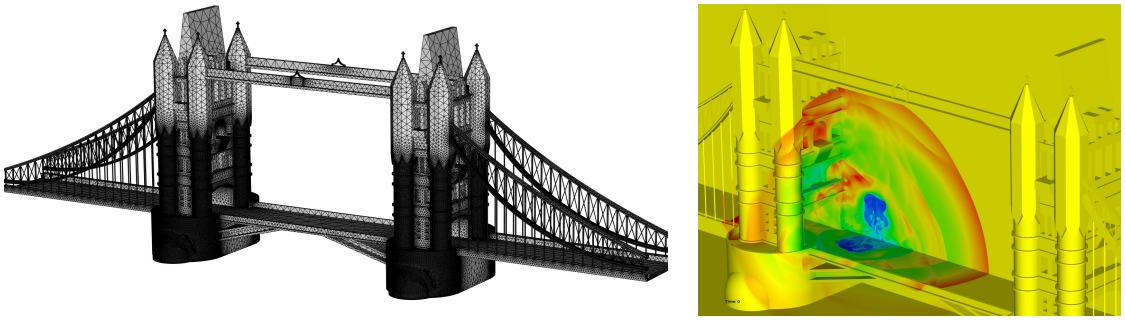


Figure 3.20 – Tower-bridge test case. Initial mesh and geometry (left) and density iso-values of the the blast on an adapted mesh (right).

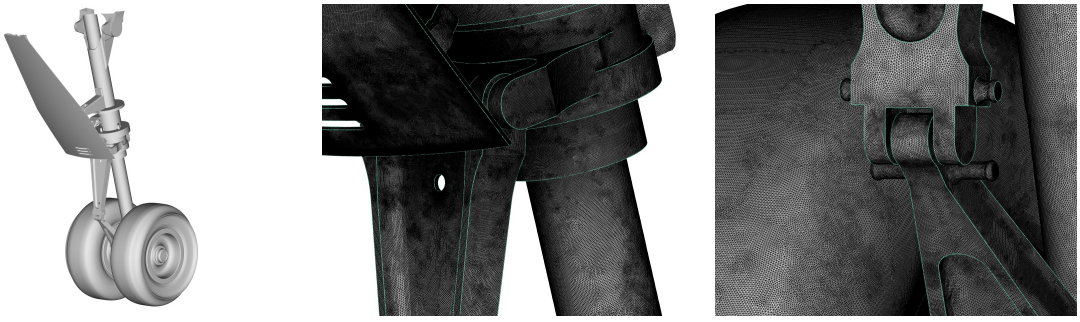


Figure 3.21 – Landing gear test case. Geometry of the landing gear (left) and closer view of the surface mesh around some geometrical details (middle and right).

Iteration	% done	# in interface	# inserted	CPU time	# of cores	elt/sec	elt/sec/core
1	84 %	89 718 245	1 009 783 723	487.5	120	$2.0\,10^6$	17 261
2	91 %	16 368 313	1 107 015 758	126.7	120	$7.6\,10^5$	6 395
3	92 %	645 035	1 122 857 778	36.6	87	$4.3\,10^5$	4 975
4	97%	2 351	1 123 488 597	5.6	4	$1.1\,10^5$	28 161
5	100%	0	1 123 490 929	1.7	1	$1.3\,10^3$	1 371

Table 3.13 – Landing gear test case on 120 cores. Table gathering the number of tetrahedra in the interface and the number of inserted tetrahedra along with the CPU time in second for each iteration.

triangles and 1 123 490 929 tetrahedra. The parallel remeshing time is 15 min 18 s and the total CPU time is 24 min 57 s (including the initial splitting and the final gathering). This example illustrates the stability of this strategy when the surface mesh contains most of the refinements. Indeed, the surface mesh is composed of more than 7.2 million vertices and 14.4 million triangles. Table 3.13 gathers all the data per iteration on this case. The geometry and closer view on the surface mesh are depicted in Fig. 3.21.

The quality of the mesh along with the histogram of the length of edges is reported in Table 3.14. More than 95% of edges have a unit length in the metric. The serial CPU time for this case is 14 h 5 min with an equivalent distribution of lengths and qualities. The mesh generated in serial is composed of 182 103 059 vertices, 14 348 710 triangles and 1 077 433 606 tetrahedra. For the complete process, we obtain a speed-up of 33 on 120 cores.

3.7 Conclusion

We have proposed a strategy to generate metric-orthogonal anisotropic meshes. It is an extension of classical anisotropic mesh adaptation. In addition to being unit with respect to a metric, metric-orthogonal meshes are composed of elements that are aligned with the eigenvectors. This allows to recover locally structured meshes while keeping the same level of anisotropy. Dihedral angles are also optimized: the distributions are concentrated around small angles (needed for

Distribution of $\ell_{\mathcal{M}}$ for edges				
0.00	$< \ell_{\mathcal{M}} <$	0.20	5 1384	0.00 %
0.20	$< \ell_{\mathcal{M}} <$	0.50	71 7381	0.05 %
0.50	$< \ell_{\mathcal{M}} <$	0.71	3 156 440	0.23 %
0.71	$< \ell_{\mathcal{M}} <$	0.90	395 251 255	**28.25** %
0.90	$< \ell_{\mathcal{M}} <$	1.11	507 393 290	**36.27** %
1.11	$< \ell_{\mathcal{M}} <$	1.41	435 996 200	**31.16** %
1.41	$< \ell_{\mathcal{M}} <$	2.00	49 278 188	3.52 %
2.00	$< \ell_{\mathcal{M}} <$	5.00	7 153 341	0.51 %
5.00	$< \ell_{\mathcal{M}}$		38 377	0.00 %

Distribution of $Q_{\mathcal{M}}$ for triangles				
1	$< Q_{\mathcal{M}} <$	2	14173311	**99.37** %
2	$< Q_{\mathcal{M}} <$	3	41197	0.29 %
3	$< Q_{\mathcal{M}} <$	4	22200	0.16 %
4	$< Q_{\mathcal{M}} <$	5	14378	0.10 %
5	$< Q_{\mathcal{M}} <$	6	7188	0.05 %
6	$< Q_{\mathcal{M}} <$	7	2839	0.02 %
7	$< Q_{\mathcal{M}} <$	8	1364	0.01 %
8	$< Q_{\mathcal{M}} <$	9	649	0.00 %
9	$< Q_{\mathcal{M}} <$	10	257	0.00 %
10	$< Q_{\mathcal{M}} <$	100	349	0.00 %

Distribution of $Q_{\mathcal{M}}$ for tetrahedra				
1	$< Q_{\mathcal{M}} <$	2	1 032 582 415	**95.46** %
2	$< Q_{\mathcal{M}} <$	3	23 777 126	2.20 %
3	$< Q_{\mathcal{M}} <$	4	6 604 689	0.61 %
4	$< Q_{\mathcal{M}} <$	5	3 973 735	0.37 %
5	$< Q_{\mathcal{M}} <$	6	2 736 791	0.25 %
6	$< Q_{\mathcal{M}} <$	7	2 010 912	0.19 %
7	$< Q_{\mathcal{M}} <$	8	1 540 367	0.14 %
8	$< Q_{\mathcal{M}} <$	9	1 208 878	0.11 %
9	$< Q_{\mathcal{M}} <$	10	966 202	0.09 %
10	$< Q_{\mathcal{M}} <$	100	6 281 111	0.58 %
100	$< Q_{\mathcal{M}} <$	1000	51 584	0.00 %
1000	$< Q_{\mathcal{M}} <$	10000	43	0.00 %

Table 3.14 – Landing gear test case on 120 cores: Distribution of the length of edges in the metric, histograms of the quality of surface and volume elements in the metric.

anisotropy) and right angles. The procedure is based on a combination of cavity-based operators with an advancing-point frontal algorithm to generate an optimal distribution of points.

The effectiveness of the method strongly depends on the quality and the properties of the input metric. Consequently, the current work is directed at improving usual anisotropic metric to comply with this metric-orthogonal and metric-aligned kernels. In particular, size and orientation smoothing should be performed with care in order to improve the orthogonality and alignment properties of the final mesh.

The parallel mesh generation approach is based on dedicated multi-level metric- based mesh partitioning techniques. The use of the cavity-based framework allows to predict effectively the CPU for each remeshing phase. The accurate estimate of element-based allows to define effective load balancing and migration strategies. Anisotropie meshes with more than a billion elements and with complex geometries are generated on small clusters (120 cores) within 10 to 20 minutes.

Chapter 4

High-order mesh visualization and adaptation

High-order approximations are becoming a standard approach to represent numerical solutions. Many numerical schemes then rely on high-order polynomials to solve PDEs. In addition to the typical issues for very high-order schemes (stability, convergence, ...), several additional difficulties arise in terms of meshing and rendering. For instance, the accurate rendering of high-order numerical solutions is one of them. As the graphic pipeline relies on linear interpolation and primitives, the cost of accurate high-order rendering is usually high as most of the process are based on subdividing into linear sub entities. Anisotropic mesh adaptation, based on interpolation error, is an additional field where the use of high-order polynomials is not trivial. Indeed, extending this approach to interpolation schemes of order strictly greater than 2 remains a challenge, especially in 3D. The main issue is the difficulty to retrieve anisotropic information from differential forms of degrees 3 or more that represent locally the spacial error with respect to the current mesh.

The chapter summarizes some contributions for high-order numerical solutions. The first one is related to pixel-exact rendering for high-order numerical mesh and solution. This work has been initiated during the Phd Thesis of R. Feuillet with the following contributions [1, 33]. Mesh adaptation for high-order solutions is a joint work with O. Coulaud during his postdoctoral time [37]. Extension for high-order CAD meshing [26, 32] is a joint work with R. Feuillet and O. Coulaud.

4.1 High-order techniques and related issues in meshing and visualization

For years, the resolution of numerical methods has consisted in solving PDEs by means of a piecewise linear representation of the physical phenomenon on linear meshes. This choice was mostly 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 [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 1978, 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 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 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.

Based on these issues, the development of high-order mesh generation procedures appears mandatory. To generate high-order meshes, several approaches exist. The first approach was tackled twenty years ago [Dey 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 2002] for biomedical 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 2014, Persson 2009, Fortunato 2016, Moxey 2016, Turner 2016, Xie 2013, Hartmann 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 2016, Gargallo-Peiró 2013, Toulorge 2013].

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 2016a, Toulorge 2016] or in a preprocessing phase [Ruiz-Gironès 2015, Ruiz-Gironès 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 2001a, Ims 2019].

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 2006, Remacle 2007, Maunoury 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 2015]. However, the cost of these methods in terms of CPU time and memory footprint is high.

Outline. In this chapter, we first review the main components allowing an accurate rendering of high-order meshes and solutions. Then, we review the log-simplex method, which provides a bound of any nonlinear k-differential forms in terms of quadratic form to the power $\frac{k+1}{2}$. From this, we can extend the continuous mesh analogy of Chapter 1 to the control of high-order interpolation in a given $\mathbf{L}^p$ norm. This estimate is then used to derive high-order parameter-independent geometric approximation of CAD geometry.

4.2 Almost pixel-exact rendering of high-order solution

Classic visualization software like ParaView [KitWare Inc.], TecPlot [TecPlot Inc.], FieldView [Intelligent Light], Ensight [Ansys Inc.], Medit [Frey 2001a], Vizir (OpenGL legacy based version) [Loseille 2016], Gmsh [Geuzaine 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 software 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 2001a]. Likewise, if a mesh has a high-order solution on it (Fig. 4.1, 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 (Fig. 4.1, second line). For this reason, mesh adaptation procedures [Remacle 2007, Maunoury 2018, Maunoury 2019] are used (Fig. 4.1, 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 (e.g. the range of values where the solution is studied) as the colors only vary when the associated values are 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 (Fig. 4.1, third line).
Other approaches are specifically devoted to high-order solutions and are based on ray casting [Nelson 2011, Nelson 2012, Peiró 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 exactly the color for this pixel. However, this method is based on two nonlinear 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.

In this section, we present how the `OpenGL` 4.0 rendering pipeline has the necessary flexibility for high-order elements and solution rendering. `OpenGL` is a graphic API widely used for graphic rendering. 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.

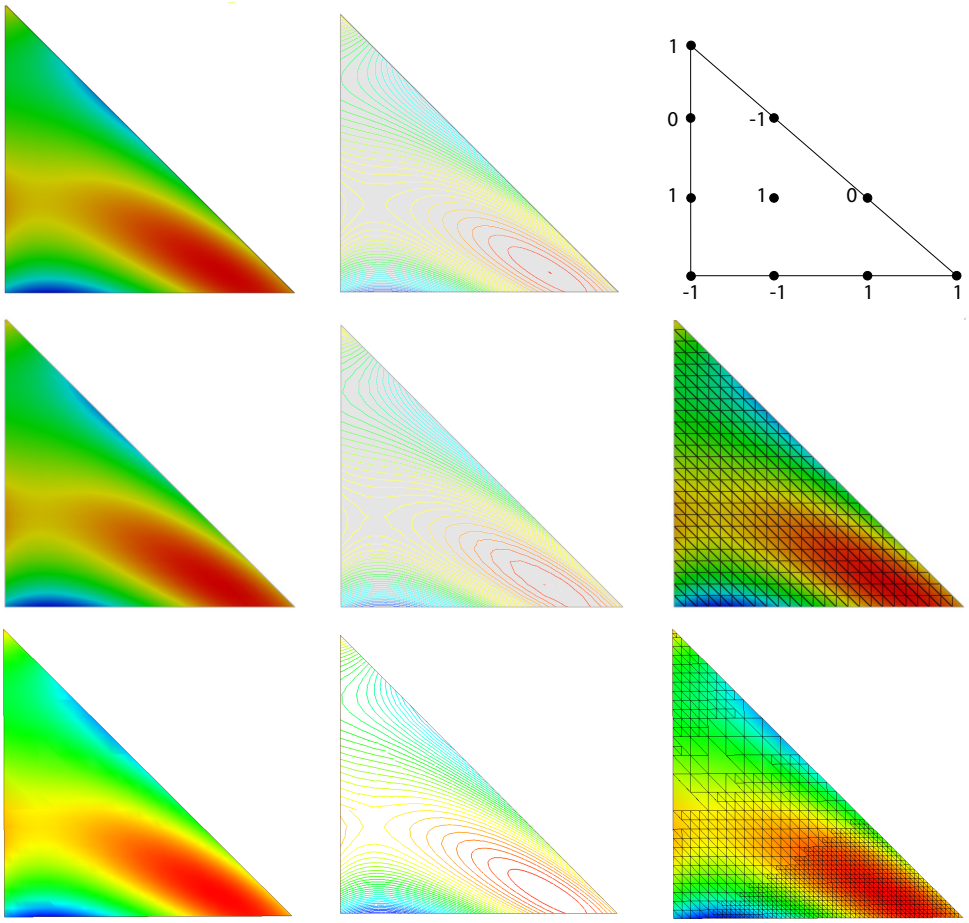


Figure 4.1 – From top to bottom, rendering of a P^3-solution: First Line: Definition of the solution on a triangle and its exact representation with its isovalues. Second line: representation of the solution linearly subdivided on a regular grid of 625 triangles with its isovalues. Third line: representation of the solution linearly subdivided on an adaptive grid of 1858 triangles (generated by Gmsh [Geuzaine 2009]) with its isovalues.

The `OpenGL` 4.0 pipeline can be customized with up to five different shader stages (see Fig. 4.7). More precisely, the description of the shaders is the following:

- The **Vertex Shader** (VS), that corresponds 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 corresponds 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 Fig. 4.2). In our context, the Fragment Shader is used to perform the wireframe rendering as well as high-order solution and isoline rendering (see Section 4.2.2). 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 Fig 4.3. In our case, we use this customization to display dots.

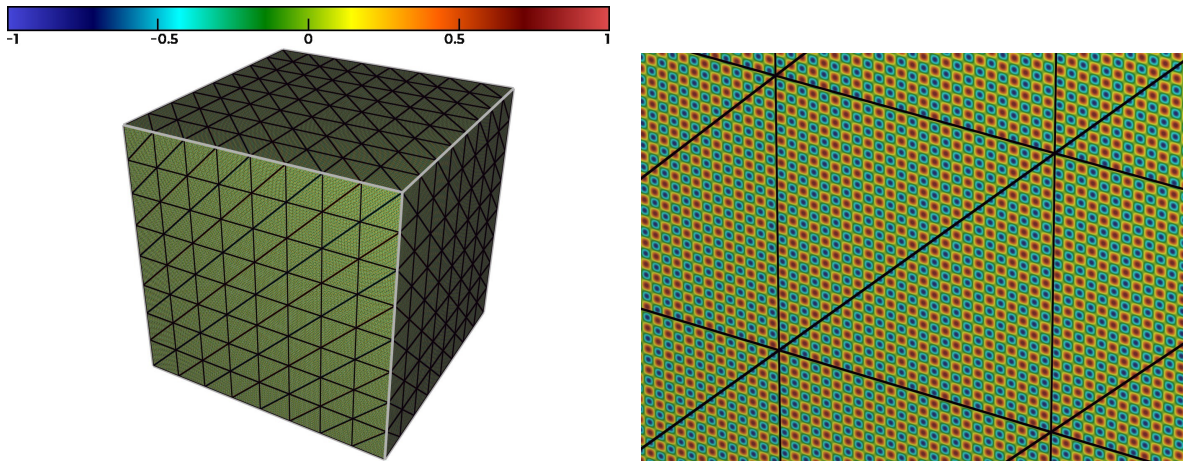


Figure 4.2 – Example of an exact rendering of the function $sin(100\pi x)+sin(100\pi y)+sin(100\pi z)$ (with its isolines on the right) on a coarse mesh of the unit cube. This is performed by using the Fragment Shader.

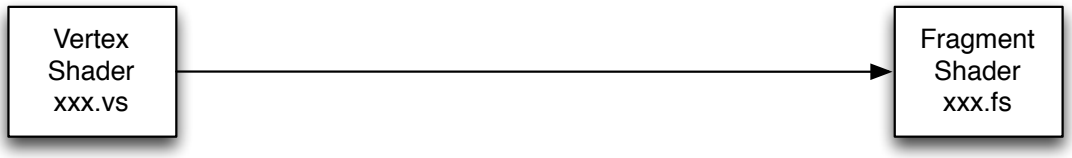


Figure 4.3 – Shaders used for a simple customization of the `OpenGL` graphic pipeline.

However, the `OpenGL` pipeline can be even more customized thanks to the following shaders:

- The **Geometry Shader** (GS), that corresponds 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 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 Fig. 4.4, left) and the shrinking of the linear elements (see Fig. 4.4, middle). For all the elements, when a clipping plane is used, the operation is performed at this stage (see Fig. 4.4, 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 [1].

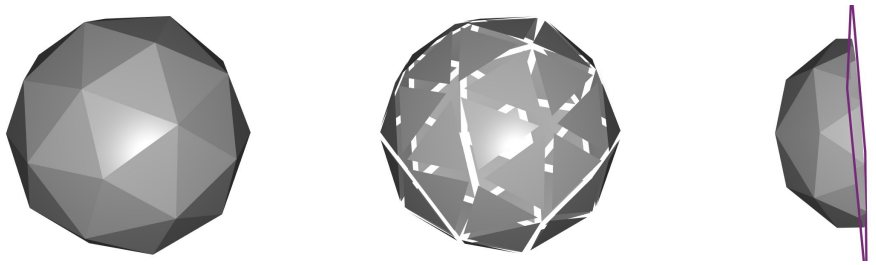


Figure 4.4 – Example of three different operations performed with the Geometry Shader.

The principle of the involved shaders is summed up in Fig. 4.5. In our case, we use this customization to display simple and linear geometries such as edges and triangles of degree 1.

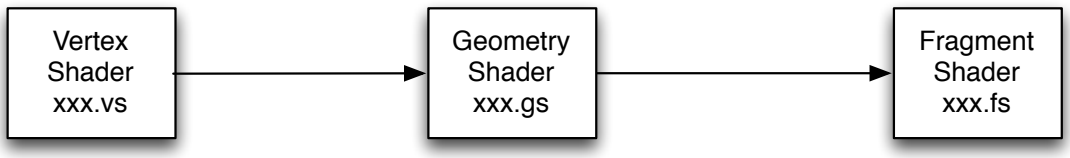


Figure 4.5 – Shaders used for the display of simple geometries using the `OpenGL` graphic pipeline.

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. 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 Fig. 4.6. For our purpose, we use this customization to display nonlinear geometries such as quadrilaterals and any other high-order element.

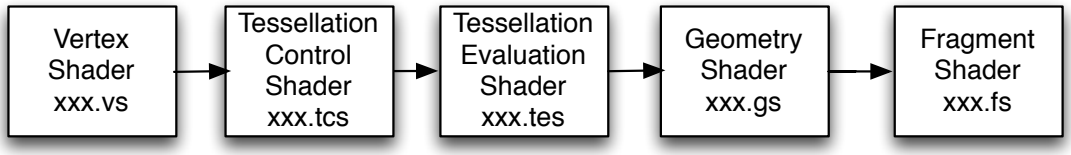


Figure 4.6 – Shaders used for the display of nonlinear geometries using the `OpenGL` graphic pipeline.

The general principle of all the combinations of shaders is summarized in Fig. 4.7.

4.2.1 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.

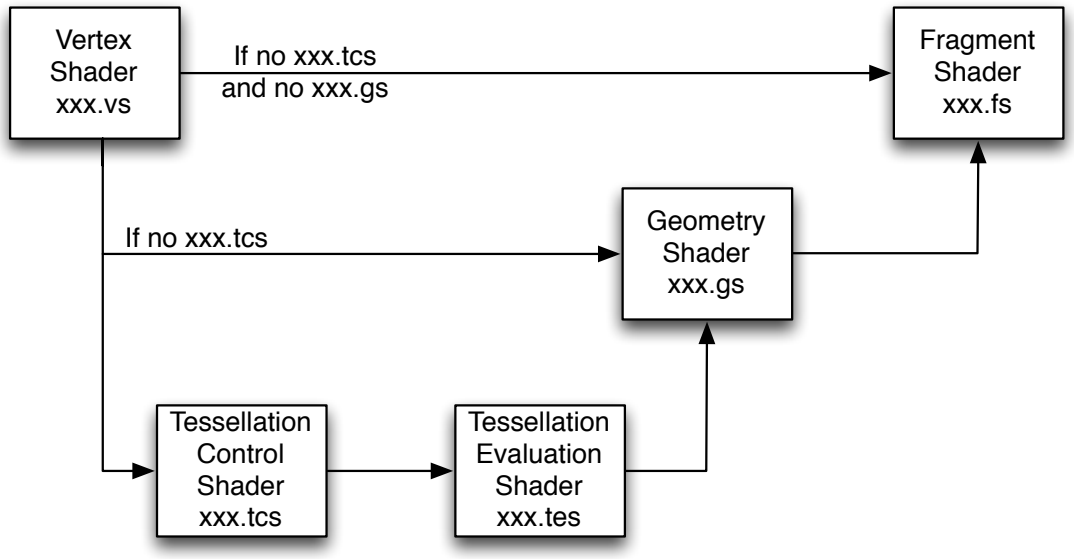


Figure 4.7 – Summary of all the possible combinations of shaders while using the `OpenGL` graphic pipeline.

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 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 Fig. 4.8). A P^3 triangle is generally defined by its 10 Lagrange points (here in barycentric notations): $(M_{ijk})_{i+j+k=3}$ (see Fig. 4.8, left). The first step is to compute the control points. 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.

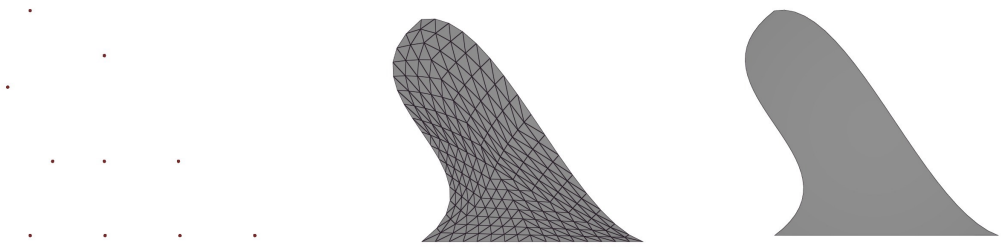


Figure 4.8 – Different steps involved in the drawing of P^3-Triangle. Left, initial distribution of points. Middle, created tessellation and right, final geometry rendering.

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 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 Fig. 4.8 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 Fig. 4.8, right and Fig. 4.9). More details are given in [1]. 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 Fig. 4.8 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.

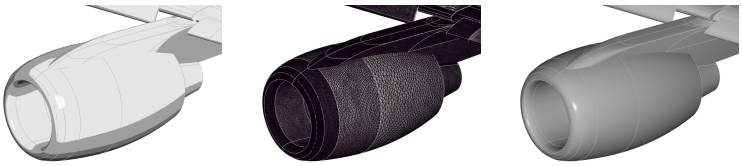


Figure 4.9 – Example of three different displays (toon, wire and Phong model) on an engine of the NASA Common Research Model.

4.2.2 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.

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 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 (Fig. 4.10), otherwise, it is almost pixel-exact.

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 (Fig. 4.11, first line). However, it is not enough for the rendering of a solution (Fig. 4.11, 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

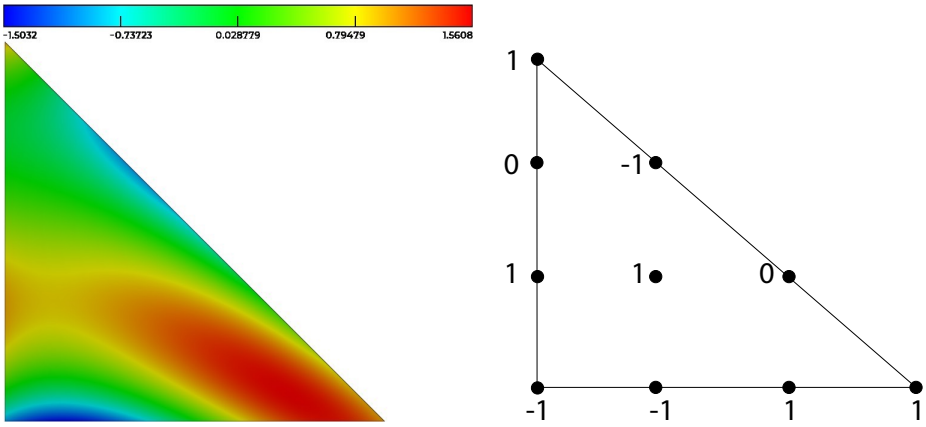


Figure 4.10 – Pixel-Exact rendering without subdivision of a P^3-solution on a triangle.

quadrilateral is not a parallelogram, the solution will not be reliably rendered. For this reason, another error estimate is considered:

$$\epsilon^{sol}_{quad} = \frac{||P_{00} - P_{n0} - P_{0n} + P_{nn}||}{\max(||P_{0n} - P_{00}||, ||P_{nn} - P_{0n}||)}.$$

This error estimate has the ability to detect if the mapping associated to a Q^1 quadrilateral has quadratic terms or not.
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.

4.2.3 Examples of high-ordre rendering

We illustrate this approach on two 3D numerical examples : a P^3 boundary element solution on an aircraft and an unsteady wave propagation using Q^6 solution. All the following examples of this chapter use this pixel-exact rendering.

High-order boundary element solution This example is taken from [6]. 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 Fig. 4.12, the resolution of a P^3 BEM on the aircraft is shown. In Fig. 4.13, 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.

Wave propagation with Perfectly Matched Layers (PML) This example is based on the numerical simulation explained in [Baffet 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

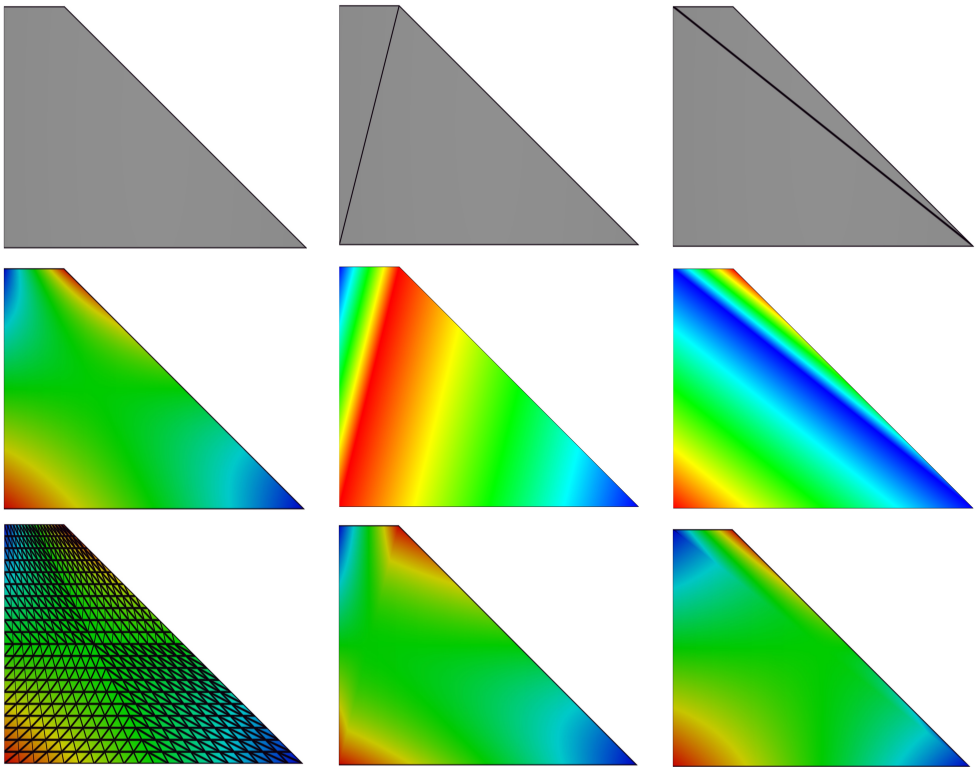


Figure 4.11 – Left, from top to bottom: rendering of a solution on a nonlinear Q^1-quadrilateral with the used tessellation. Middle and right, from top to bottom: rendering of the solution on quad for each of its two decomposition in triangles; linear rendering done using the decomposition; bilinear rendering with the Fragment Shader but with a wrong approximation of the parametric coordinates due to the decomposition in only two triangles.

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 Fig. 4.15 and 4.14, 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.

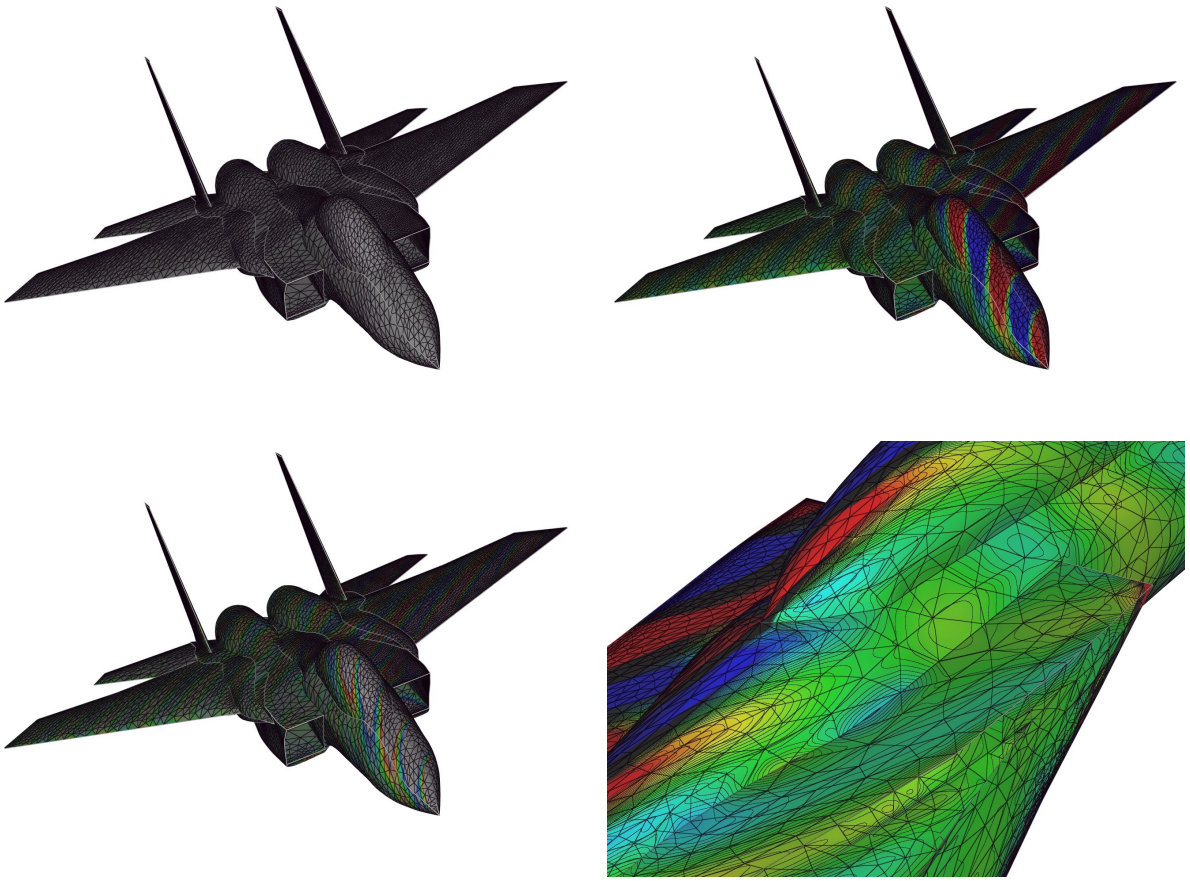


Figure 4.12 – High-order BEM resolution of the scattering of plane waves on an aircraft. Top, meshes. Middle, solution, Bottom, isolines. Bottom right, zoom around the cockpit. Courtesy of Stéphanie Chaillat (ENSTA).

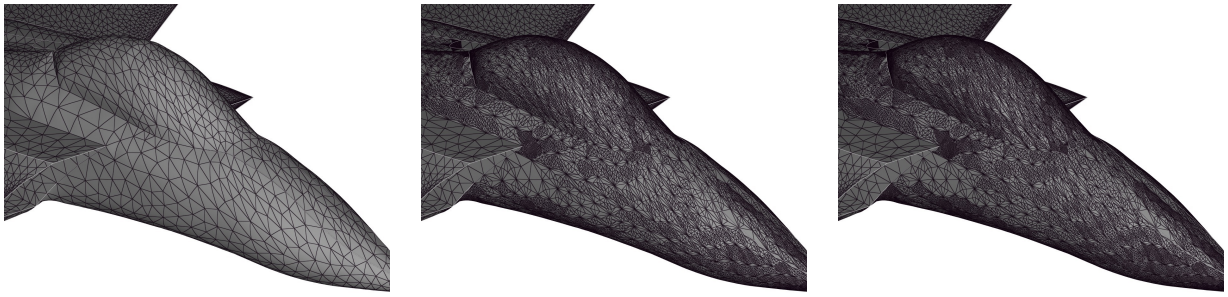


Figure 4.13 – Left, zoom on a curved part of the P^3-mesh used for the BEM resolution. Middle and right, various tessellation levels used for the display of the geometry. Courtesy of Stéphanie Chaillat (ENSTA).

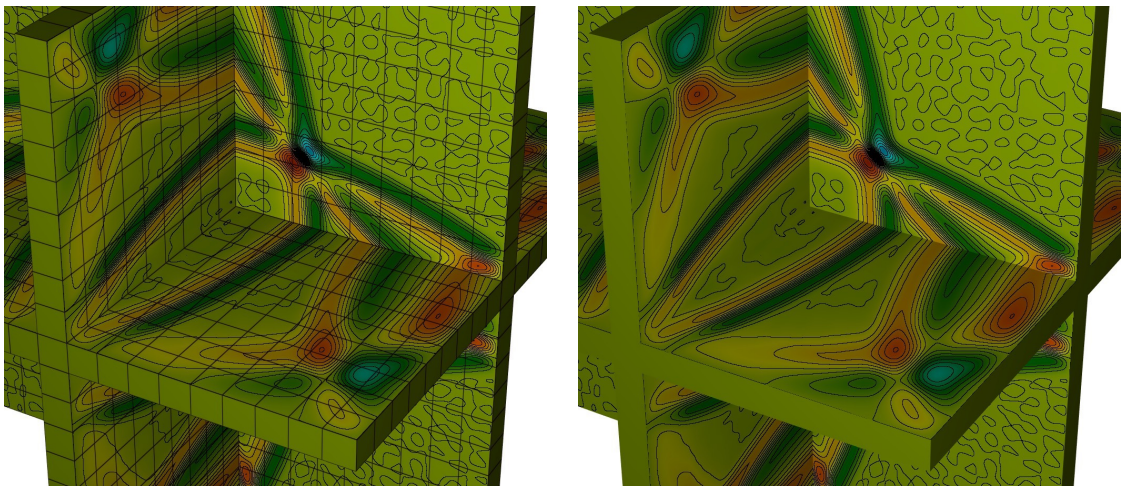


Figure 4.14 – Zoom in the middle of the cube with the solution of Fig. 4.15 with its isolines. Courtesy of Sébastien Impériale (INRIA).

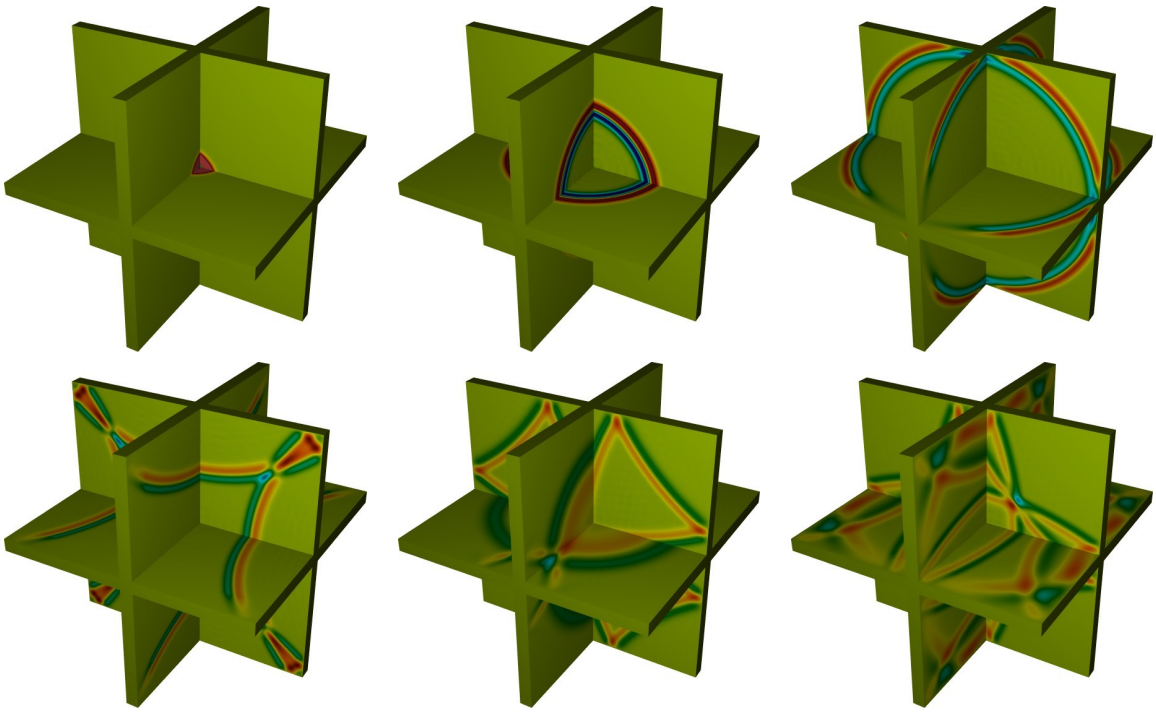


Figure 4.15 – Q^6 solution of a wave propagation problem on an hexahedral mesh. Courtesy of Sébastien Impériale (INRIA).

4.3 High-order mesh adaptation

Following most of the high-order *a priori* error estimates [Ciarlet 1978], we focus on a simple error model as in [37]. Let u be a smooth solution on the domain Ω, $\mathcal{H}$ be a mesh of Ω and k be an arbitrary positive integer. In what follows, $\Pi_k u$ denotes the projection of u onto the finite elements space $P^k(\mathcal{H})$, whose functions are polynomials of degree k 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 $x \in \mathbb{R}^n$,

$$\begin{aligned} |u(x) - \Pi_k u(x)| \leq C \left| d^{(k+1)}u(x_0)(x - x_0) \right| \\ + \mathcal{O}\left(\|x - x_0\|^{k+2} \right), \end{aligned} \tag{4.1}$$

where $d^{k+1}u(x_0)$ is the differential form of u of order $k+1$ at x_0, $|.|$ is the absolute value function and $\|.\|$ denotes the Euclidean norm of $\mathbb{R}^n$. In the high order case, the main idea is to replace the right hand side of (4.1) by a term governed by a metric field $\mathbf{Q} = (\mathcal{Q}(x))_{x\in\Omega}$, which approximates the $k+1$ differential form of u. More precisely, we are looking for $\mathbf{Q}$ such that for all $x_0 \in \Omega, x \in \mathbb{R}^n$

$$\left| d^{k+1}u(x_0)(x - x_0) \right| \leq \left({}^t(x - x_0)\, \mathcal{Q}(x_0)\, (x - x_0) \right)^{\frac{k+1}{2}} \tag{4.2}$$

The main issue is to find the metric field $\mathbf{Q}$ such that the inequality (4.2) is as optimal as possible. From a geometrical point of view, the local problem is 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 $d^{(k+1)}u(x_0)$. Indeed, let $\mathcal{B}_{\mathcal{Q}} = \left\{ x \in \mathbb{R}^n : {}^t x\, \mathcal{Q}\, x \leq 1 \right\}$ be the unit ball of a metric $\mathcal{Q}$, which is an ellipse in 2D (an ellipsoid in 3D). Now assume that, for all $x \in \mathbb{R}^n$ such that $d^{(k+1)}u(x_0)(x) = 1$, one has ${}^t x\, \mathcal{Q}\, x \geq 1$. Let $x \in \mathbb{R}^n$ and $y = \dfrac{x}{\left| d^{(k+1)}u(x_0)(x) \right|^{\frac{1}{k+1}}}$. In particular $d^{(k+1)}u(x_0)(y) = 1$, and:

$$d^{(k+1)}u(x_0)(y) \leq {}^t y\, \mathcal{Q}\, y.$$

Since $d^{(k+1)}u(x_0)$ is a homogeneous polynomial of degree $k+1$, it comes

$$1 \leq \frac{{}^t x\, \mathcal{Q}\, x}{\left| d^{(k+1)}u(x_0)(x) \right|^{\frac{2}{k+1}}},$$

and consequently

$$\left| d^{(k+1)}u(x_0)(x) \right| \leq \left({}^t x\, \mathcal{Q}\, x \right)^{\frac{k+1}{2}}, \quad \text{for all } x \in \mathbb{R}^n.$$

The purpose of the next section is to solve this minimization problem.

4.3.1 Log-simplex method

The main difference between the P^1 and the P^k adaptation methods 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 (4.2) for the P^k 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 $k+1$ on $\mathbb{R}^n$ which stands for $d^{(k+1)}u(x_0)$, a set of points $\{x_1, ..., x_m\}$ of $\mathbb{R}^n$ 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

$$\begin{aligned} &\det(\mathcal{Q}) \quad \text{is minimal,} \\ &{}^t x_i \, \mathcal{Q} \, x_i \geq 1, \quad \text{for all} \quad i \in \{1, ..., m\}\,. \end{aligned} \tag{4.3}$$

The first line of (4.3) translates the fact that we are looking for the metric with the largest area (or volume in 3D). This geometric framework has been notably studied from a theoretical point of view by [Cao 2007, Cao 2008] and from a numerical point of view by [Hecht 2014], but in 2D only. On the contrary to these works, the method which is introduced in the present article can be implemented numerically in both 2D and 3D, and is much faster than the one studied by [Hecht 2014]. Since the cost function of this problem is nonlinear,we rewrite 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 allows the discrete counterpart of the problem to be well posed. Indeed, in [37], it is shown that the discrete form of (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 (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 $x \in \mathbb{R}^n$ satisfies ${}^t x \, \mathcal{L} \, x \geq -\|x\|^2 \log\left(\|x\|^2\right)$, it ensures that ${}^t x \, \mathcal{Q} \, x \geq 1$. By this way, the following linear optimization problem is obtained.

Find a symmetric matrix $\mathcal{L}$ such that

$$\begin{aligned} &\text{trace}(\mathcal{L}) \quad \text{is minimal,} \\ &{}^t x_i \, \mathcal{L} \, x_i \geq -\|x_i\|^2 \log\left(\|x_i\|^2\right), \quad \forall i \in \{1, ..., m\}\,. \end{aligned} \tag{4.4}$$

Since this problem is linear in $\mathcal{L}$, it can be solved by a simplex method (see for instance [Dantzig 2003]). Unfortunately, in most of the cases, solving (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 Fig. 4.16).
This issue is dealt by an iterative process. More precisely, once we have computed the solution of (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 (4.4). Finally, the log-simplex algorithm is the following.
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 $d^{(k+1)}u$ disappear. All the theoretical and numerical issues of the log-simplex method are described in detail in [37].

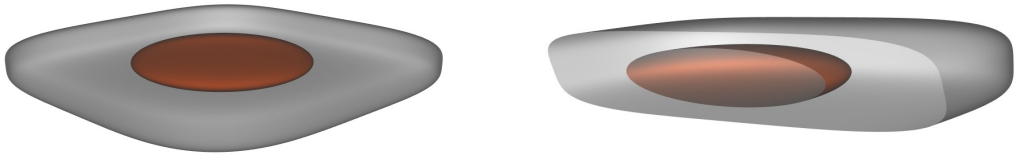


Figure 4.16 – Illustration of the log approximation for the constraints for an error level 1 (in grey). The optimal metric (in red) is far from the boundary of the error due to the convexity approximation.

input : A mesh $\mathcal{H}$ of Ω
$d^{(k+1)}u(x)$, 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 $\{x_1, ..., x_n\}$ 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*;
end

From (4.2), we are able to deduce an optimal metric field $\mathbf{M}$ minimizing the L^p high order interpolation error when considering unit meshes with respect to $\mathbf{M}$. Following the demonstration of [11, 12], the high order interpolation error in L^p norm is equivalent to

$$E_p^k(\mathbf{M}, u) = \left(\int_\Omega \left| \text{trace}\left(\mathcal{M}^{-\frac{1}{2}}(x)\mathcal{Q}(x)\mathcal{M}^{-\frac{1}{2}}(x) \right) \right|^p dx \right)^{\frac{1}{p}}. \tag{4.5}$$

By a calculus of variations, we show that, for a fixed complexity $\mathcal{N} \in (0, +\infty)$, the optimal metric field minimizing E_p^k is unique and given by:

$$\mathcal{M}_{opt}^{p,k}(u)(x) = \mathcal{N}^{\frac{2}{3}} \left(\int_\Omega (\det \mathcal{Q})^{\frac{p(k+1)}{2p(k+1)+6}} \right)^{-\frac{2}{3}} (\det \mathcal{Q}(x))^{-\frac{1}{p(k+1)+3}} \mathcal{Q}(x). \tag{4.6}$$

In particular, if $k = 1$ and $\mathcal{Q} = |H_u|$, one recovers the P^1 optimal metric field defined in Chapter 1 (Equation (1.6)). From an initial mesh $\mathcal{H}_0$ of Ω, the whole P^k adaptation process follows iteratively the next steps.

input : Initial mesh $\mathcal{H}_0$
Complexity N

output: Final mesh $\mathcal{H}_1$

repeat
Compute $d^{(k+1)}u(x)$, for every vertex x of $\mathcal{H}_0$
Compute $\mathcal{Q}(x)$ satisfying (4.2), for all $x \in \mathcal{H}_0$
Compute $\mathcal{M}(x) = (\det|\mathcal{Q}(x)|)^{-\frac{1}{p(k+1)+3}}\,\mathcal{Q}(x)$, for all x of $\mathcal{H}_0$
Compute $\mathcal{M}_{opt}^{p,k} = \alpha\mathcal{M}$, with $\alpha > 0$ such that $\mathcal{C}(\mathcal{M}_{opt}^{p,k}) = N$
Remesh $\mathcal{H}_0$ and obtain $\mathcal{H}_1$ which is unit with respect to $\mathcal{M}_{opt}^{p,k}$
Replace $\mathcal{H}_0$ by $\mathcal{H}_1$
until *convergence*;

4.3.2 Numerical examples

In this section, we consider two smooth functions. For each function, several adaptations are performed for interpolation orders from 1 to 5. Note that only the discrete P^k solution is used to recover the differential form of order $k+1$., *i.e.*, we never used the exact derivatives of the function, only its point-wise values are used. To do so, we have extended the L^2 projection to the case of high-order differential forms (see [Vallet 2007]). Once the numerical $(k+1)$ differential form of the smooth solution u is recovered, we apply the log-simplex algorithm and derive the optimal L^p metric $\mathcal{M}_{opt}^{p,k}(u)$ for a given complexity N. The interpolation error $\|u - \Pi_k u\|_{L^2(\Omega)}$ is computed using a 10^{th} order Gauss quadrature integration. To compare simultaneously different interpolation orders, the degrees of freedom (DoF) are used instead of the number of the nodes. The error is then used to compare the convergence rate to the optimal one. According to Equalities (4.5) and (4.6), the interpolation error induced by a unit mesh with respect to the optimal metric field $\mathcal{M}_{opt}^{p,k}(u)$ satisfies

$$\|u - \Pi_k u\|_{L^p(\Omega)} \leq \frac{C}{\mathcal{N}^{\frac{k+1}{3}}}, \quad \text{with} \quad C > 0. \tag{4.7}$$

The anisotropic meshes are generated by using a unique cavity operator of Chapter 2.

Toy problem. The first function is tailored such that it has isotropic variation for linear interpolation, strong y anisotropic component at third order and x anisotropic component at fourth order. The function is defined on $\left[-\frac{1}{2}, \frac{1}{2}\right]^2$ with $f(x,y) = x^2 + y^2 + \frac{x^3}{10} + \frac{y^4}{10^5}$. We observe in Fig. 4.17 P^1, P^2 and P^3 adaptation. We verify that an isotropic mesh is obtained at order one, x-aligned mesh at order 2 and finally y-aligned mesh at order 3.

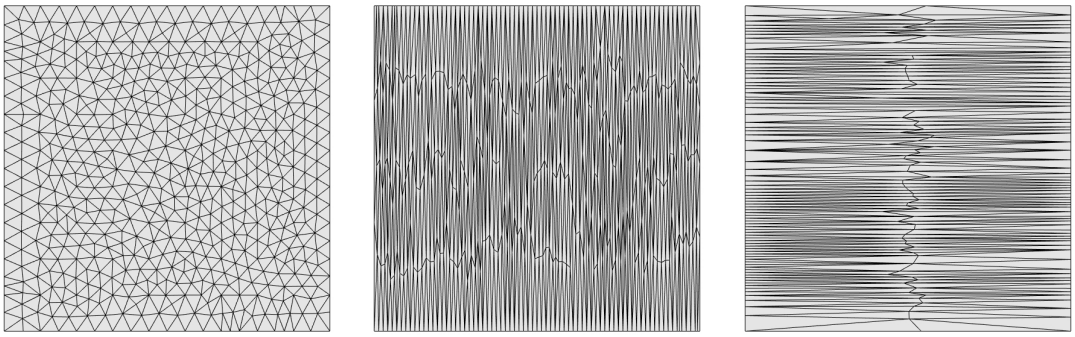


Figure 4.17 – P^1, P^2 and P^3 adapted meshes for function f.

High-frequency function The second function oscillates at high frequency and contains small details, it is given by :

$$f_r(x,y,z) = 8\,xyz\,\sin(5\,\pi\,xyz)^4 + \frac{1}{10}\left(1 - (\sin(5\,\pi\,xyz)^4\right)^8 \cos(100\,\pi\,xyz). \tag{4.8}$$

The high frequencies variations of the function are depicted in Fig. 4.18 (left). In this case, the asymptotic rate of convergence is not reached directly due to the small details of the function that are captured for sufficiently small sizes of the mesh, especially for $k > 1$. This does not appear for the linear case as the finest mesh is still too coarse with respect to these variations of small amplitudes. The final mesh for P^1 contains more than 2 000 000 vertices while the equivalent mesh for P^5 in term of DoF contains 19 779 vertices, 11 214 triangles and 101 592 tetrahedra for a error level 3 orders of magnitude below the linear curve. The uniform meshes have a similar behavior, see Fig. 4.18 (right). Again, the sequence of adaptive meshes have lower error curves and the error is 2 order of magnitude smaller in the asymptotic range for P^5 interpolation. A high level of anisotropy is kept for all k due to the small frequencies, see Fig. 4.19. 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. Table 4.1 compares the CPU times for one iteration of the adaptation process. Even if it is longer to compute the high order derivatives and the optimal metric field, the time needed for remeshing is much smaller when dealing with high order interpolation, and so is the total CPU time.

4.4 High-order surface mesh generation

As introduced in Chapter 1, CAD geometry is usually defined continuously as NURBS function (Non-uniform rational B-spline) as it is a common tool in geometry modeling and CAD systems [Piegl 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 [Tristano 1998, Miranda 2002, Wang 2006, Laug 2010, de Siqueira 2010, de Siqueira 2014,

	P^1	P^2	P^3	P^4	P^5
degrees of freedom	2 374 794	2 376 164	1 989 277	2 329 110	2 220 443
interpolation error	7.2×10^{-5}	6.9×10^{-6}	1.8×10^{-6}	3.9×10^{-7}	1.8×10^{-7}
total CPU time (s)	365	604	153	120	115
derivatives (s)	2	102	37	40	45
metric field (s)	26	409	100	70	64
remeshing (s)	330	63	11	6	4

Table 4.1 – CPU time and number of DoF for the sequence of adapted meshes for function f_r.

Aubry 2015] and some approaches consist of meshing the 2D domain according to a curvature-based metric [Borouchaki 2000b]. The use of this metric enables an 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 2016, Ruiz-Gironès 2015, Ruiz-Gironès 2016a, Ruiz-Gironès 2016b, Turner 2016]. 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 2013] by minimizing a *distortion measure*. In all these cases, this is *r-adaptation* which is performed. The intent is to provide theoretical analysis and practical algorithms for high-order parametric surface mesh generation. The use of the high-order estimates of previous Section provides 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.

4.4.1 Metrics for linear surface mesh generation

In the case of parametric surface meshing, the whole problem is to find a suitable metric thanks to which a mesh adaptation process in the parameters space will be performed. First let us have a look on the case of curve meshing.

Metrics for curve meshing

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 on 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.

$$\gamma(t) = \gamma(t_0) + \gamma'(t_0)(t - t_0) + \frac{\gamma''(t_0)}{2}(t - t_0)^2 + \mathcal{O}((t - t_0)^3).$$

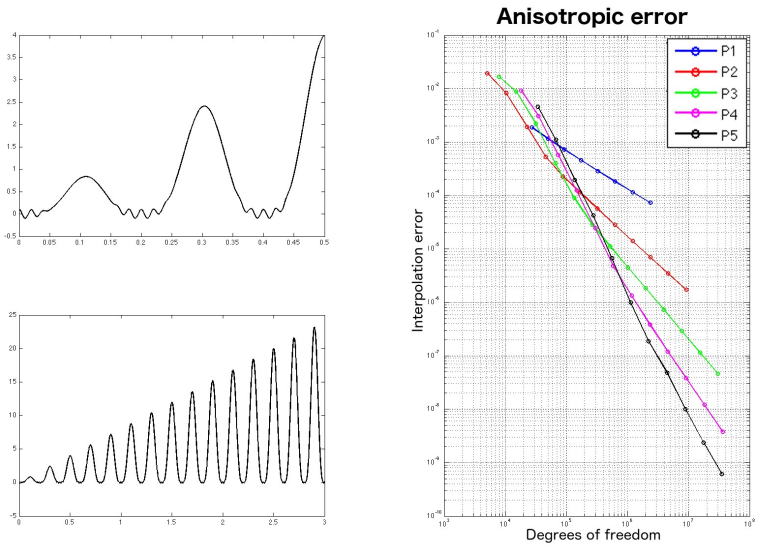


Figure 4.18 – From left to right, 1D extraction along the line $xyz = cst$, rate of convergence for the optimal meshes for order 1,2,3,4 and 5.

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 1976]. 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:

$$\begin{cases} y = y_0 + \frac{1}{2}\kappa(t_0)(x - x_0)^2 + \mathcal{O}(|x - x_0|^3) \\ z = z_0 + \mathcal{O}(|x - x_0|^3) \end{cases} \tag{4.9}$$

Based on the Frénet frame, and on the curvature, a 3D metric tensor can be deduced via:

$$\mathcal{M}_1 = \left({}^t\mathbf{T}\,{}^t\mathbf{N}\,{}^t\mathbf{B}\right) \begin{pmatrix} \frac{1}{(2\sqrt{\epsilon(2-\epsilon)}\rho(t))^2} & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{pmatrix} \begin{pmatrix} \mathbf{T} \\ \mathbf{N} \\ \mathbf{B} \end{pmatrix}$$

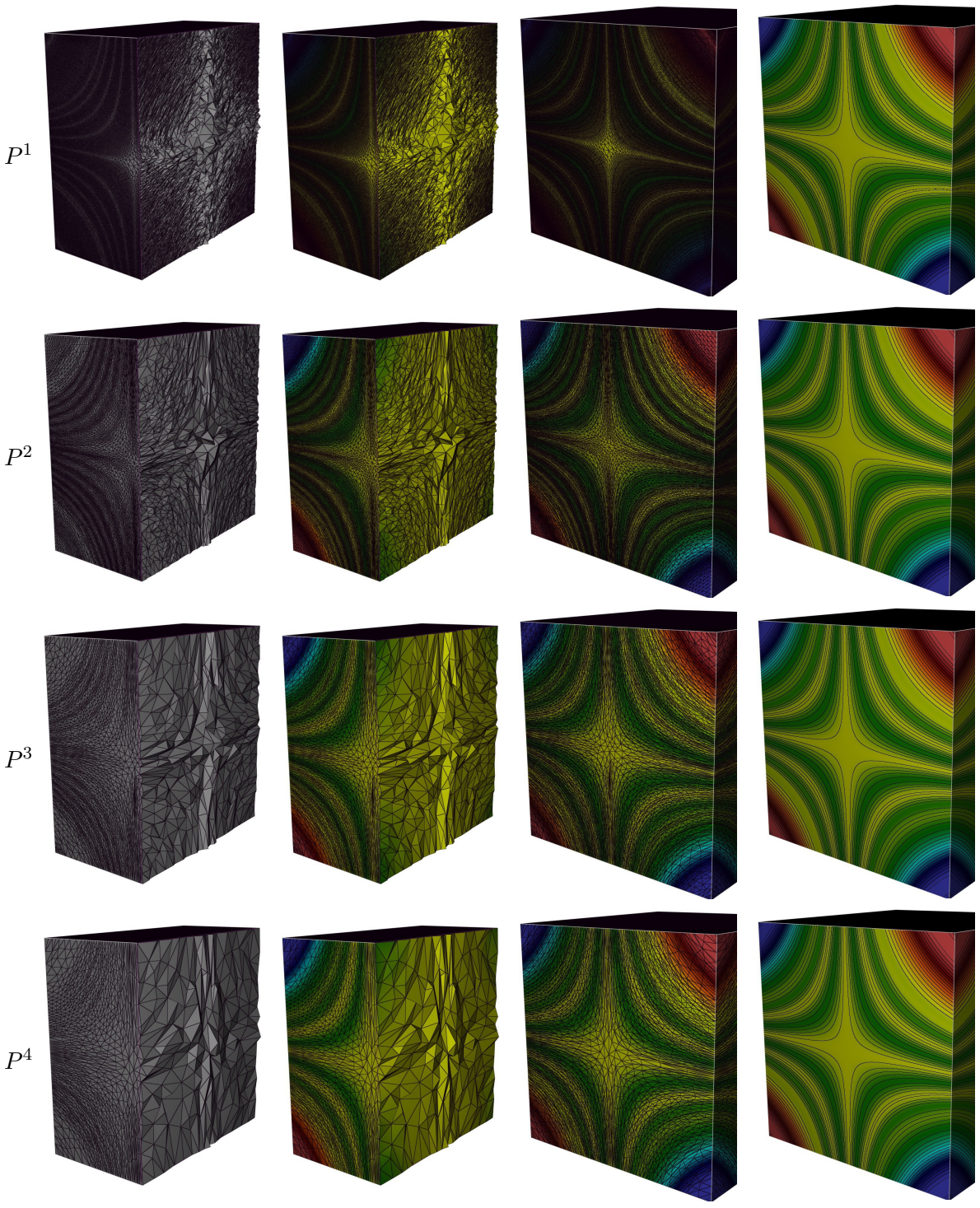


Figure 4.19 – From top to bottom, P^1 to P^4 adapted meshes (left) with the corresponding solution on it (right).

where $\lambda \in \mathbb{R}$ is an arbitrary constant[1], $\rho(t) = \frac{1}{\kappa(t)}$ 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 can be mapped back to the parameter space via the following formula:

$$\tilde{\mathcal{M}}_1 = {}^t\gamma'(t)\mathcal{M}_1\gamma'(t).$$

In this case, the formula simplifies to $\tilde{\mathcal{M}}_1 = \frac{1}{h_1^2} = \frac{||\gamma'(t)||^2}{(2\sqrt{\epsilon(2-\epsilon)}\rho(t))^2}$.
Once the metrics for curves are set, the next step is to define metrics for the surfaces.

Metrics for surface meshing

The meshing process of parametric surfaces is a bit more complex. It relies on some differential geometry notions [do Carmo 1976]. 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, comes also the *second fundamental form* $II(du, dv)$ that is defined as follows:

$$II(du, dv) = (du\ dv) \begin{pmatrix} (\sigma_{uu}, \mathbf{N}) & (\sigma_{uv}, \mathbf{N}) \\ (\sigma_{uv}, \mathbf{N}) & (\sigma_{vv}, \mathbf{N}) \end{pmatrix} \begin{pmatrix} du \\ dv \end{pmatrix},$$

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 order two.
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:

$$\begin{cases} \mathcal{M}_{II}\mathbf{v}_i = \kappa_i \mathcal{M}_I \mathbf{v}_i \\ \mathbf{V}_i = \dfrac{(\sigma_u\ \sigma_v)\,\mathbf{v}_i}{||\,(\sigma_u\ \sigma_v)\,\mathbf{v}_i||} \end{cases} i = 1, 2$$

[1] It sets the size in the normal plane to the curve

with $\mathcal{M}_{II}$ being 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:

$$\begin{aligned} z = z_0 + \frac{1}{2}(\kappa_1(u_0, v_0)(x - x_0)^2 + \kappa_2(u_0, v_0)(y - y_0)^2) \\ + \mathcal{O}(\|(x - x_0, y - y_0)\|^3). \end{aligned} \tag{4.10}$$

Now, thanks to this basis and on the curvatures, we can define the following 3D metric tensor:

$$\mathcal{M}_2 = \left({}^t\mathbf{V}_1 {}^t\mathbf{V}_2 {}^t\mathbf{N}\right) \times \begin{pmatrix} \frac{1}{(c_1\rho_1(u,v))^2} & 0 & 0 \\ 0 & \frac{1}{(c_2\rho_2(u,v))^2} & 0 \\ 0 & 0 & \lambda \end{pmatrix} \begin{pmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{N} \end{pmatrix}$$

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 2019]. Similarly to curves, the metric relies only on intrinsic data and is therefore independent of the parameterization

Now, the metric $\mathcal{M}_2$ can be mapped back to the parameters space by applying the first fundamental form:

$$\tilde{\mathcal{M}}_2 = \left({}^t\sigma_u \; {}^t\sigma_v\right) \mathcal{M}_2 \begin{pmatrix} \sigma_u \\ \sigma_v \end{pmatrix}.$$

This is the metric that will be used as an anisotropic metric for the mesh adaptation in the parameters space.

4.4.2 Computation of higher-order metrics

The object of this section is to deal with the extension of the previous framework for higher-order elements. In particular, we seek for parameterization independent Taylor expansion similar to (4.9) and to (4.10) with terms of degree greater than 2. First let us have a look on the case of the curve.

Case of the curve

As seen previously, the metric should rely on intrinsic data to be independent of the parameterization. A way to do so is to have a look at the formula (4.9). 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 with the physical coordinates which naturally give an independence with respect to any parameterization. A natural idea is therefore to

extend the previous Taylor expansion to get higher-order terms and deduce metrics that will be fitted to high-order approximation.
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 $\phi'(t_0) \neq 0$. The inversion function theorem can thus be applied and 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^{k+1} then ψ is C^{k+1} and $\psi'(X) = \frac{1}{\phi'(t)}$ with $X = \phi(t)$.
Based on this statement, it is thus possible to get a Taylor expansion of $t - t_0$ with respect to X up to order k. To do so, let us compute the higher-order derivatives of ψ in t_0. As γ (and consequently ϕ) is an analytical function issued from CAD models, all its derivatives can be computed using the implementation details of [Piegl 1997]. The derivatives of ψ are then deduced using the following result [Faà di Bruno 1857]:

Theorem 1 (Faà di Bruno's Formula)**.** *Let us consider $f, g : \mathbb{R} \to \mathbb{R}$ of class C^{k+1} with $k+1 \geq n$, then*

$$\frac{\mathrm{d}^n}{\mathrm{d}x^n}(g(f(x))) = \sum_E \frac{n!}{m_1!..m_n!} g^{(m_1+...+m_n)}(f(x)) \prod_{j=1}^{n} \left(\frac{f^{(j)}(x)}{j!} \right)^{m_j},$$

where $E = \{(m_1, .., m_n) \in \mathbb{N}^n | \sum_{k=1}^{n} k.m_k = n\}$[2].

In our case, we set $g = \psi$ and $f = \phi$ and it comes that $\frac{\mathrm{d}^n}{\mathrm{d}x^n}(g(f(x))) = 0$ for $k + 1 \geq n \geq 2$. This brings us to that for $k + 1 \geq n \geq 2$:

$$(\phi'(t))^n \psi^{(n)}(x) = F(\psi'(x), .., \psi^{(n-1)}(x), \phi'(t), .., \phi^{(n)}(t)),$$

where F is a function that can be directly deduced from Theorem 1.
This result means that as long as $\phi'(t) \neq 0$, then $\psi^{(n)}(x)$ can be recursively computed given its previous derivatives and the derivatives of ϕ.
Thanks to this, we can now write:

$$t = t_0 + \sum_{n=1}^{k+1} a_n X^n + \mathcal{O}(|X|^{k+2}),$$

where a_n have been computed with the derivatives of ψ. Now if we recall that γ is C^{k+1}, we also have:

$$\begin{cases} Y = \sum_{n=1}^{k+1} b_n (t - t_0)^n + \mathcal{O}(|t - t_0|^{k+2}), \\ Z = \sum_{n=1}^{k+1} c_n (t - t_0)^n + \mathcal{O}(|t - t_0|^{k+2}). \end{cases}$$

[2]Note that in practice, the set E can be precomputed once for all for the range of values of n that are used.

By composition of both Taylor expansion, we then obtain a Taylor expansion of Y and Z in X at the order $k+1 \geq 2$, which is independent of the parameterization and a generalization of (4.9).
From this, we have an intrinsic information of the gap of a curve to the straight edge up to the order $k+1$.
We can then write:

$$\begin{pmatrix} Y \\ Z \end{pmatrix} = F_k(X) + A_{k+1}X^{k+1} + \mathcal{O}(|X|^{k+2}),$$

where $F_k(X)$ is a polynomial of degree k in X and $A_{k+1} \in \mathbb{R}^2$. For an approximation of a curve at the degree k, the leading term of the error is therefore $A_{k+1}X^{k+1}$[11, 12, 37]. Thus, if we control $||A_{k+1}X^{k+1}||$, the error of approximation will be controlled. Now, if we note that:

$$||A_{k+1}X^{k+1}|| = (||A_{k+1}||^{\frac{2}{k+1}}X^2)^{\frac{k+1}{2}},$$

we can then set $\kappa_{k+1} = 2||A_{k+1}||^{\frac{2}{k+1}}$ and reuse the metrics used for the linear meshing with $\rho = \frac{1}{\kappa_{k+1}}$ for radii of curvature and $\epsilon_{k+1} = \epsilon^{\frac{2}{k+1}}$ for threshold. This way, the classic formula is found for $k = 1$ and a generalization is proposed for $k \geq 2$. Note that in some configurations, the found size for the order k can be significantly lower than the found size for the order $k+1$. In this case the size given by order $k+1$ is preferred.
Now , let us interest to the more complex case of the 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 (4.10). 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, then its Jacobian matrix J_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^{k+1} then Ψ is C^{k+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 $k+1$. For this purpose, let us compute the higher-order derivatives of Ψ.

As σ (and therefor Φ) is an analytical function issued from a CAD model, all its derivatives can be computed using the recipes in [Piegl 1997]. The derivatives of Ψ can be then deduced using the following result [Encinas 2003]:

Theorem 2 (2D Faà di Bruno's Formula)**.** *Let us consider $f, g : \mathbb{R}^2 \to \mathbb{R}^2$ of class C^{k+1} with $k+1 \geq |\alpha|$, then*

$$\frac{\partial^{|\alpha|}}{\partial x^\alpha}(g(f(x))) = \sum_{|\sigma|=1}^{n} \alpha! \frac{\partial^{|\sigma|}}{\partial x^\sigma}(g(f(x))) \times \sum_{E_\sigma} \prod_{i=1}^{2} \prod_{A_\alpha} \frac{1}{e_{i\alpha^i}!} \left(\frac{1}{\alpha^i!} \frac{\partial^{|\alpha^i|}}{\partial x^{\alpha^i}}(f_i(x)) \right)^{e_{i\alpha^i}},$$

where $f(x) = (f_1(x), f_2(x))$, $\alpha = (\alpha_1, \alpha_2) \in \mathbb{N}^2$,$n = |\alpha| = \alpha_1 + \alpha_2$, $\sigma = (\sigma_1, \sigma_2) \in \mathbb{N}^2$

$$E_\sigma = \left\{ (e_{1\alpha^1}, e_{2\alpha^2}) \in \mathbb{N}^2, 1 \leq |\alpha^i| \leq n, \left(\sum_{|\alpha^i|=1}^{n} e_{i\alpha^i} = \sigma_i, \right)_{i=1,2} \right\}$$

and [3]

$$A_\alpha = \left\{ (\alpha^1, \alpha^2) : 1 \leq |\alpha^i| \leq n, i = 1, 2, \sum_{i=1}^{2} \sum_{|\alpha^i|=1}^{n} e_{i\alpha^i}.\alpha^i = \alpha \right\}.$$

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 $k+1 \geq |\alpha| \geq 2$. If we consider all the $n+1$ α such that $|\alpha| = n$, then we have a system of equations of the kind:

$$A \left(\frac{\partial^{|\alpha|}\Phi}{\partial x^\alpha} \right)_{|\alpha|=n} \times \left(\frac{\partial^{|\alpha|}\Psi}{\partial x^\alpha} \right)_{|\alpha|=n} = F \left(\left(\frac{\partial^{|\alpha|}\Psi}{\partial x^\alpha} \right)_{|\alpha|<n}, \left(\frac{\partial^{|\alpha|}\Phi}{\partial x^\alpha} \right)_{|\alpha|\leq n} \right),$$

where A is $(n+1) \times (n+1)$ matrix, $\left(\frac{\partial^{|\alpha|}\Psi}{\partial x^\alpha} \right)_{|\alpha|=n}$ is a vector of size $n+1$ containing the $n+1$ derivatives of Ψ of order n and F is vector function of size $n+1$ that can be deduced from theorem 2. Moreover, it is shown in [Encinas 2003] that $|A| = |J_\Phi|^n$, which proves that the system has always a solution if the inverse function theorem is successfully applied.
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:

$$\begin{pmatrix} u - u_0 \\ v - v_0 \end{pmatrix} = \sum_{n=1}^{k+1} \sum_{i+j=n} A_{ij}^n X^i Y^j + \mathcal{O}(||(X,Y)||^{k+2}),$$

[3] 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 $A_{ij}^n \in \mathbb{R}^2$ and is defined thanks to the partial derivatives of Ψ.
But, as σ is C^{k+1}, we also have:

$$Z = \sum_{n=1}^{k+1} \sum_{i+j=n} c_{ij}^n (u-u_0)^i (v-v_0)^j + \mathcal{O}(||((u-u_0),(v-v_0))||^{k+2}).$$

By composition of both Taylor expansions, we then obtain a Taylor expansion of Z in (X,Y) at the order $k+1 \geq 2$ which is independent of the parameterization and a generalization of the formula (4.10).
The gap to the tangent plane is thus expressed up to the order $k+1$:

$$Z = F_k(X,Y) + R_{k+1}(X,Y) + \mathcal{O}(||(X,Y)||^{k+2}), \tag{4.11}$$

where F_k is a polynomial of degree k and R_{k+1} is an homogeneous polynomial of degree $k+1$. For an approximation of the surface at the degree k, the leading term of the error is $R_{k+1}(X,Y)$ [37]. So, if we want to control the P^k approximation, we need to control $|R_{k+1}(X,Y)|$.
By applying the log-simplex algorithm explained in the previous section, we are able to find a metric that satisfies an inequality like (4.2), that is to say, we are able to compute a matrix Q_{k+1} such that:

$$|R_{k+1}(X,Y)| \leq \left(\frac{1}{2} (X\ Y) Q_{k+1} \begin{pmatrix} X \\ Y \end{pmatrix} \right)^{\frac{k+1}{2}},$$

where Q_{k+1} is the optimal symmetric matrix (in a sense explained in the first section) that verifies this inequality.
If we note $(\kappa_{i,k+1}, \mathbf{v}_{i,k+1})_{i=1,2}$, the eigenvalues and eigenvectors of Q_{k+1}, we can then reuse the metrics used for the linear meshing with $\rho_i = \frac{1}{\kappa_{i,k+1}}$ the radii of curvature, $\mathbf{V}_{i,k+1} = (\mathbf{V}_{1,0}\mathbf{V}_{2,0})\,\mathbf{v}_{i,k+1}$ the principal directions in the tangent plane and $\epsilon_{k+1} = \epsilon^{\frac{2}{k+1}}$ the threshold. This way, the classic formula is found for $k=1$ and a generalization is proposed for $k \geq 2$. Like for curves, in some configurations, found sizes for the order k can be significantly lower than found sizes for the order $k+1$. In this case sizes given by order $k+1$ are preferred.

4.4.3 Meshing process

The mesh generation process is based on the classical unit-mesh 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 meshing, 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 the paper, 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 $\tilde{\mathcal{M}}_2$ 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 onto the current mesh,

   2.4 Move to 3D, convert $\tilde{\mathcal{M}}_2$ to $\mathcal{M}_2$ to adapt the mesh,

   2.5 Generate high-order mesh.

The `EGADS` API [Haimes 2012] is used to perform the CAD linking and the planar mesh generation process is performed using a Delaunay triangulation-based algorithm[George 1998].

4.5 Numerical illustrations

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 the point of the sphere using the inversion formula given by (4.11). Note that the sphere is parameterized as a circle of revolution. Far from the pole, we have :

$$\begin{aligned} Z &= -0.5Y^2 - 0.5X^2 - 0.125Y^4 \\ &\quad -0.25X^2Y^2 - 0.125X^4 \\ &\quad -0.0625Y^6 - 0.1875X^2Y^4 \\ &\quad -0.1875X^4Y^2 - 0.0625X^6 \\ &\quad +\mathcal{O}(||(X,Y)||^7). \end{aligned}$$

In the vicinity of the two poles, the expansion is

$$\begin{aligned} Z &= -0.5Y^2 - 0.5X^2 - 0.125Y^4 \\ &\quad -0.25X^2Y^2 - 0.125X^4 \\ &\quad -1.80444e^{-9}XY^4 + 1.9886e^{-9}X^3Y^2 \\ &\quad -0.0625Y^6 - 0.187501X^2Y^4 \\ &\quad -0.187499X^4X^2 - 0.0625X^6 \\ &\quad +\mathcal{O}(||(X,Y)||^7). \end{aligned}$$

We observe that a numerical noise appears while an almost perfect expansion is obtained as in regular points. As expected, the second order terms reveal half 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 a bit impacted by this.

Shuttle. We consider a shuttle geometry 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 Fig. 4.20. The error to the geometry are reported in Table 4.2.

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 4.2 – Deviation to the geometry for the shuttle geometry.

Conclusion

This Chapter is a review regarding some of the challenges related to the use of high-order meshes and solutions to approximate PDEs.

For high-order mesh and solution rendering, an almost pixel-exact approach has been presented. It is based on the complete customization of the OpenGL pipeline. In particular, the evaluation of nonlinear polynomials is computed exactly directly on the GPU. To approximate the nonlinear mapping of curved elements, an automatic tessellation is performed directly on the GPU. Exact representation is obtained for linear elements independently of the order of the solution. This approach also offer a low memory footprint as the initial high-order mesh or solution is never subdivided on the CPU.

To extend anisotropic mesh adaptation to high-order solutions, an iterative algorithm has been devised to derive a local optimal metric to approximate a given $(k+1)$ differential form of degree k. At each step, a linear log-simplex problem is solved in the logarithm space of metric fields. This optimal local metric is then globally optimized *via* a calculus of variations to obtain the optimal distribution of the DoF in $\mathbf{L}^p$ norm. This strategy has been tested on various 3D examples showing an optimal rate of convergence. For all the adaptive cases, the adapted meshes have a lower level error and reach faster the asymptotic rate of convergence.

Generating high-order curved surface meshes from a geometry requires 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.

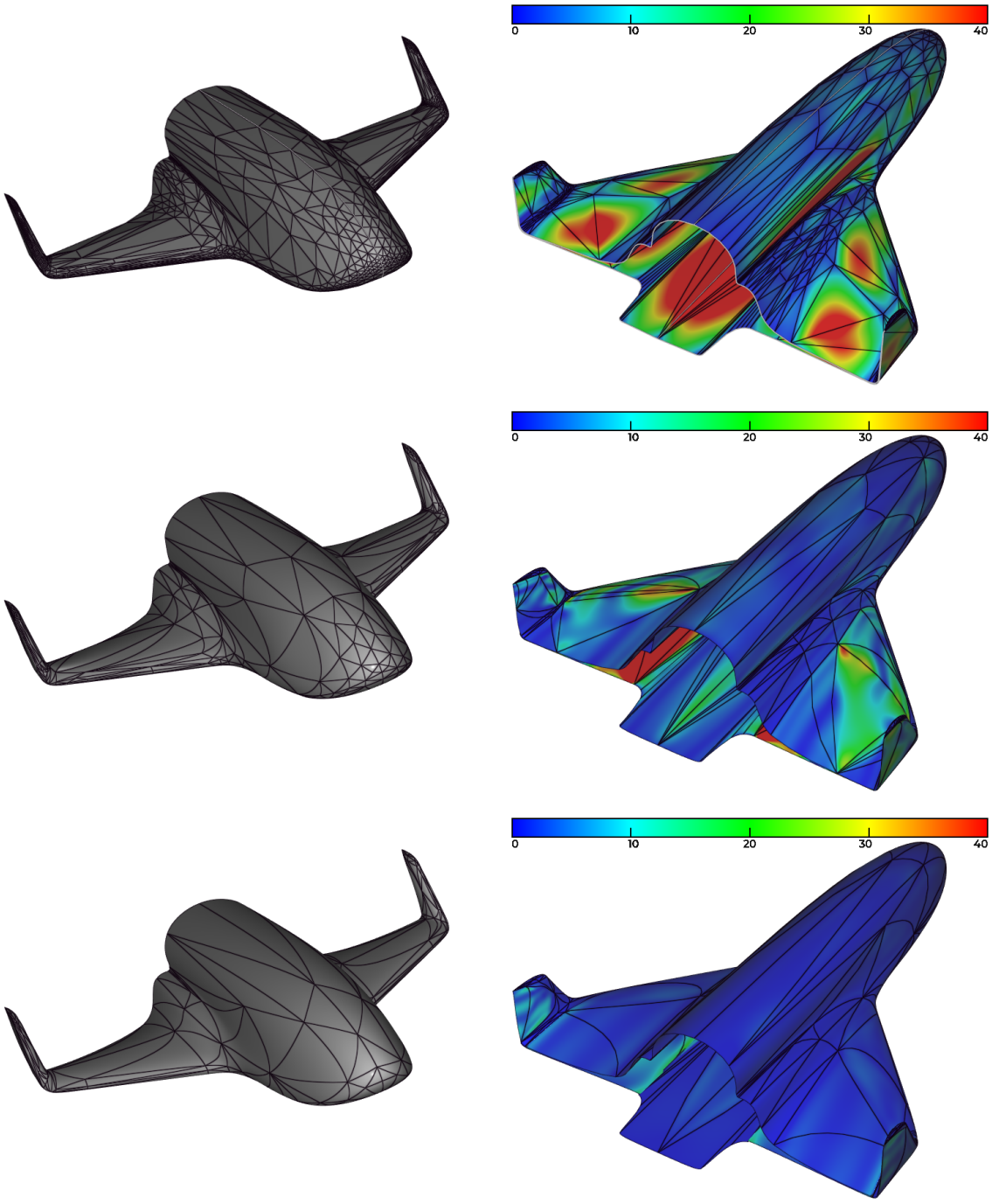


Figure 4.20 – P^1 (top), P^2 (middle) and P^3 (bottom) point-wise distance to the shuttle geometry.

As a metric field is naturally derived, these estimates can be used directly in any adaptive anisotropic mesh generation process.

Chapter 5

Conclusions and perspectives

I reviewed in this manuscript recent developments for mesh adaptation and error estimation employed within an anisotropic adaptive process. The underlying goal is to obtain high-fidelity numerical prediction of complex nonlinear PDEs.

In that respect, the metric-based framework has proven to be a key component to develop efficient error estimations and design adaptive meshing algorithms. From an error estimate point of view, the metric-based framework allows to define a complete continuous setting. Interpolation error estimates are then sought continuously through a calculus of variation. This approach is reviewed in Chapter 1 to derive multi-scale error estimates that controls the interpolation in $\mathbf{L}^p$ norm. Such estimates were then extended for very high order interpolation in Chapter 4. The continuous mesh framework has been extended by several groups to control high-order approximation error [Yano 2012], h-p mesh optimization [Ringue 2017] or derived h-p error estimates [Dolejsi 2015, Rangarajan 2018]. From a practical point of view, the concept of unit mesh with respect to a metric field allows to generate an anisotropic mesh. The discrete (computational) mesh is then a discrete representation of the continuous mesh. A compound of simple operators (insertion, collapses, swaps), monitored by a quality function, provides a highly robust meshing process. Indeed, as mesh adaptation is a nonlinear process, meshes and solutions are converged to a fix point. A single failure in a mesh generation completely broke the adaptive prediction : no mesh, no solution. Examples of Chapter 1 exhibit an early capturing of the underlying physical phenomena both for steady and unsteady simulations. The direct sonic boom prediction is a perfect illustration of one of the benefits of mesh adaptation : optimal distribution of the degrees of freedom. If a uniform mesh were generated with the accuracy obtained on the ground, 10^{18} tetrahedra would have been needed, leading to more than 1 000 years of computing ... The adaptive prediction is around 4 hours on a standard desktop computer. If most of the problems in this thesis are related to CFD, the concepts introduced in Chapter 1 can be studied for additional sets of physical problems or numerical scheme. For instance, the continuous mesh theory was extended to boundary element methods [4, 6]. In that case, all the developments are extended to work on solutions only known at the boundary. When applied to wave scattering frequency problems, mesh adaptation has proven to recover an optimal order of convergence for boundary solutions with low regularity. We then note a strong analogy with fluid dynamics where second order spatial accuracy is recovered for flows with shocks [62]. In Fig. 5.1, we illustrate the scattering at multiple frequency by a cavity. The relative $\mathbf{L}^2$ error is compared to uniform refinements.

The second component improved in this thesis is the mesh generation mechanism itself. In Chapter 2, the cavity-based operator was introduced to offer a robust and unique operator in an adaptive context. It can be easily extended to generate hybrid meshes. Complex operators

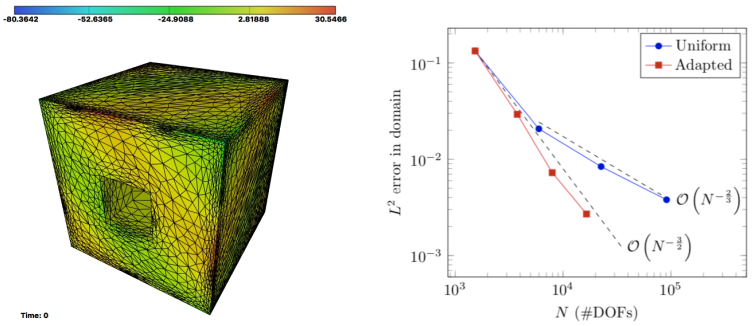


Figure 5.1 – Scattering by a sound-soft cube with cavity, mesh and solution (left) and rate of convergence (right) when compared with uniform refinements.

are then automatically defined thanks to cavity correction algorithms. In addition to a drastic gain in CPU times (40 times faster than the baseline adaptive algorithm of Chapter 1), it has the required robustness to handle highly complex geometries and high levels of anisotropy ($O(1-10^5)$) required for RANS simulations. This opens the way for mesh generation algorithms having the ability to drastically improve the quality of the meshes. Two techniques, metric-aligned and metric-orthogonal approaches, have been reviewed in Chapter 3. These techniques extends to anisotropy traditional advancing methods [Borouchaki 2000a, Remacle 2012]. The process relies on a global process where points have to be placed in preferred directions to favor alignment, orthogonality and quality. These preferred directions are obtained from the input metric field. The main advantage of this approach is that this step is completely decoupled from the point insertion problem. A second improvement concerns the generation of large size meshes. An original parallel mesh generation method was introduced. It relies on (i) a metric-based static load-balancing, (ii) hierarchical mesh partitioning techniques to (re)split the (complex) interfaces meshes, (iii) a fast, robust and generic sequential cavity-based mesh modification kernel. Again, we observe the high benefit of adopting a continuous and discrete representations of meshes. Again, the metric-based framework offers a convenient theoretical background : to define directions of interest for the metric-align and metric-orthogonal approach and to define optimal load balancing estimates for the parallel process.

The mesh adaptation process presented in the thesis has a sufficient level of maturity to tackle industrial problems. In this respect, we illustrate in Fig. 5.2 some mesh adaptation techniques for incompressible flows or bi-fluid simulations, all courtesy of Lemma company. From an industrial perspective, mesh adaptation defines a process that helps to converge to a solution of high fidelity, without requiring the needs of experts engineers to design tailored meshes.

Future work

Future research work will be directed at different levels. The first one is related to the improvements of the error estimates in particular their reliability. The second one focuses on the improvement and development of new meshing algorithms for adaptive high-order curved mesh and parallel strategies.

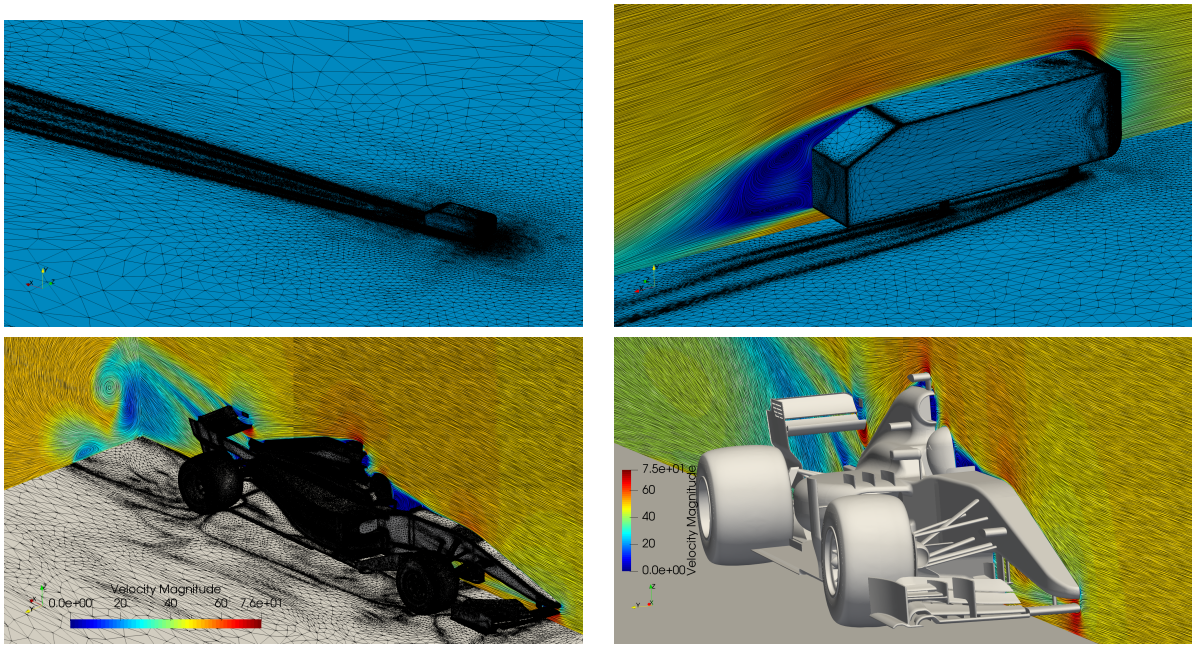


Figure 5.2 – Examples of mesh adaptation for incompressible flows performed by Lemma : Ahmed body (top) and Perrin F1 (bottom). Once an initial mesh and flow conditions are set up, no user intervention is required.

Norm oriented error estimates. The classical error estimates (Hessian-based in Chapter 1 and 4 or goal-oriented in the numerical section of Chapter 2) address specifically one goal or one sensor. In particular, the implicit error $W_h - \Pi_h W$ is not controlled. However, this term naturally arises when the functional of interest is for instance the norm $\|\Pi_h W - W_h\|_{L^2}$. One step is then to estimate the implicit error. This error can be interpreted as a point-wise error that estimates the point-wise error between the exact solution and the solution provided by the numerical scheme. First developments have been done in the case of inviscid flows [3, 42]. To do so, we estimate the defect, δ, of the continuous solution when applied to the numerical solution. In the framework of the Euler equations, we have

$$\mathrm{div}(F(W_h)) = \delta \approx 0.$$

To estimate δ, the continuous flux function F is evaluated on a finer mesh. Then we compute $F_{h/2}\left(\Pi_{h/2} W_h\right)$ which is equivalent to one solver flux evaluation. This flux is accumulated back on the current mesh, following geometric multi-grid methods, defining a source term:

$$S_{post} = A_{h/2 \to h}(F_{h/2}(\Pi_{h/2} W_h)),$$

where $A_{h/2\to h}$ is the operator consisting in localizing the point of the finer grid $h/2$ on to the finite volume cell of coarser grid h and accumulating the flux. The corrected solution is then solution of:

$$\mathrm{div}(F_h(W_c)) = S_{post},$$

starting with $W_c = W_h$. Note that the finer grid is never generated from a practical point of view, only a local subdivision is used. In addition, the computation of S_{post} is naturally highly parallel. Contrary to the goal-oriented mesh adaptation, the functional may be now any function of the approximation error. A particular case involves multiple functionals of interest to be minimized simultaneously. For instance, instead of the above δj, we can minimize the semi-norm-like functional:

$$j(W_h) = (\text{drag}(W) - \text{drag}(W_h))^2 + (\text{lift}(W) - \text{lift}(W_h))^2$$

while the goal-oriented should use two functionals, one for drag and one for lift or specify a combination of them. An example of correctors is depicted in Fig. 5.3, where the point implicit error is depicted as error bars on top of pressure extraction. We observe that the uncertainty on the pressure distribution decreases when a sequence of adaptive meshes is used, while this uncertainty remains almost unchanged on a sequence a tailored refined meshes. However, several challenges remains to obtain an accurate corrector near viscous body for RANS simulations. Indeed, the combination of complex geometries and strong anisotropy requires to design specific strategies to obtain an accurate defect.

High-order curved mesh adaptation. In Chapter 3, the aligned and orthogonal approaches have proved to increase the quality of the generated meshes by using additional features of the input metric field, like its natural alingnent and orthogonality. In a more general setting, new theoretical developments on Riemannian metric fields are necessary to extract pertinent mesh-curvature information, orthogonality, alignment and natural connectivity information. This information, that have not been used so far in traditional meshing techniques, will guide modern mesh generation algorithms to obtain high-quality curved, hybrid and adapted meshes. Most of the approaches to generate high-order curved meshes rely either on solving PDEs [Abgrall 2014, Aparicio-Estrems 2019, Fortunato 2016, Karman 2016, Moxey 2016, Roca 2012, Ruiz-Gironès 2016a] or defining specific optimizations to move the points in the domain. For all cases, these techniques focus on uniform meshes and their extension to generate anisotropic meshes is not natural. It is then necessary to extend both the theoretical background of unit-mesh and mesh modification operators to operate on anisotropic curved meshes. For adapted and curved meshes, the advancing-point technique of Chapter 3 seems an interesting alternative as the density of the point and direction of propagation can be controlled in a curved manner. A similar global approach is now considered to generate hexahedral meshes with the definition of a frame field [Beaufort 2017], however the developments are still for uniform meshes and no effort is put in the connecting phase of these points [Sokolov 2017]. In Fig. 5.4, we illustrate a simple approach where curved elements are generated by considering the variation of the metric field. The length of the edges is then optimized in the Riemannian field. In terms of parallelism, the approach of Chapter 3 allows to generate meshes where the memory is the main limitation. However, for classical multi-core architectures, a fine-grained parallelism seems required for efficiency [Remacle 2015]. Developing a parallel kernel of the cavity operators is then a natural research axis.

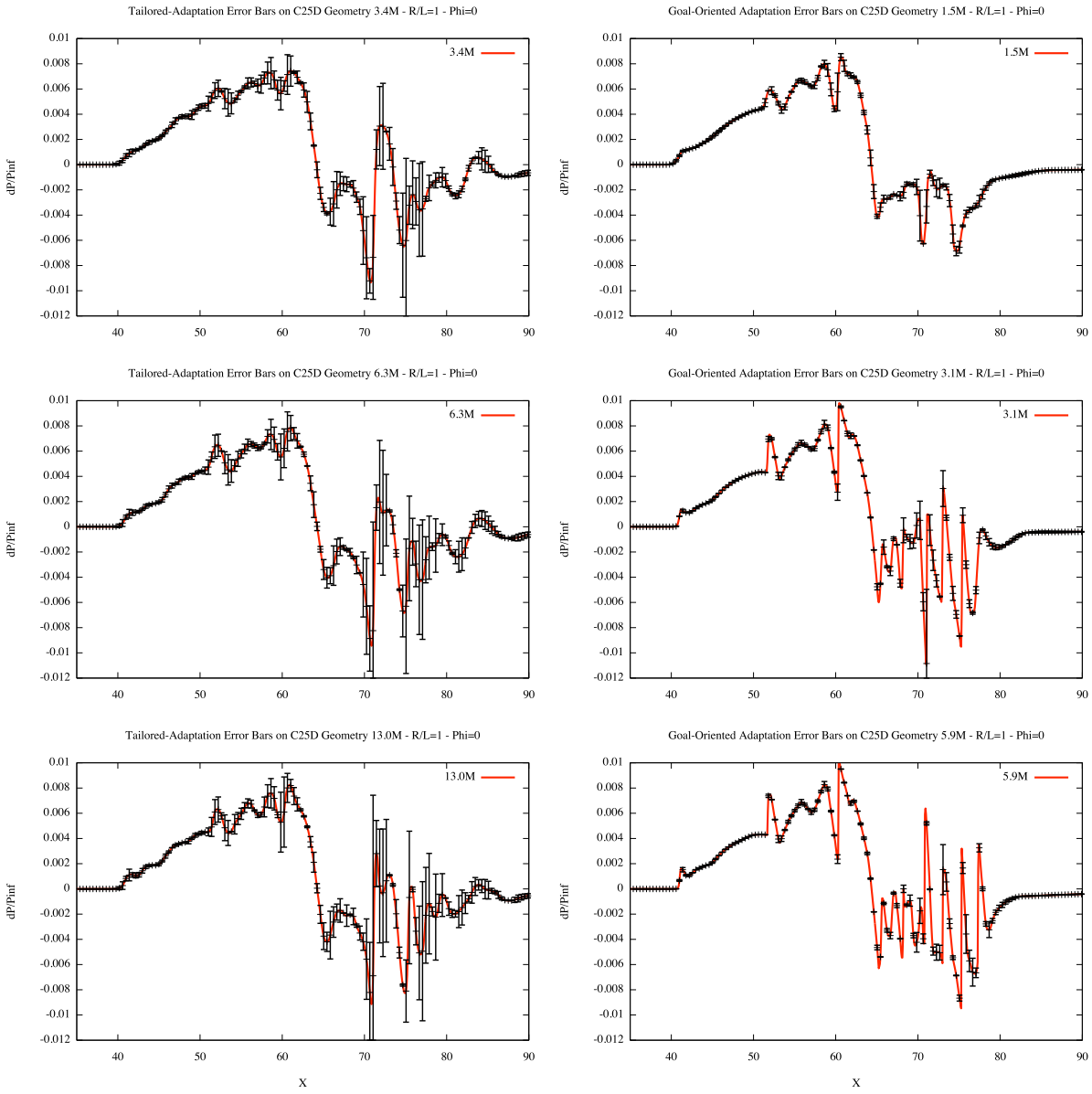
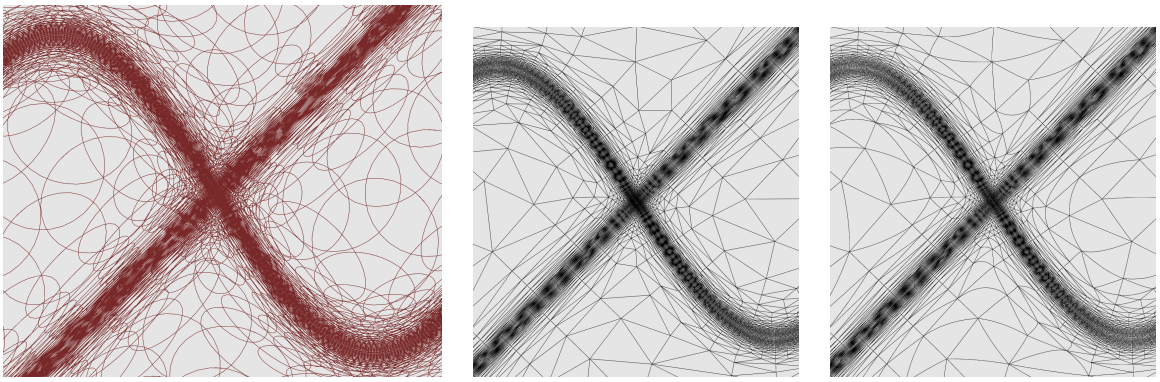


Figure 5.3 – Nonlinear corrector predictions of the pressure on tailored meshes (left) and adapted goal-oriented meshes (right).



Metric tensor P1 induced curved P2

Figure 5.4 – Illustration on a first attempt of an induced curvature implied by the metric tensor variation in 2D.

Bibliography

[Abgrall 2001] R. Abgrall. *Toward the Ultimate Conservative Scheme: Following the Quest.* Journal of Computational Physics, vol. 167, no. 2, pages 277 – 315, 2001.

[Abgrall 2014] 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, no. 4, pages 246–266, 2014.

[Alauzet 2006] F. Alauzet, X. Li, E. Seegyoung Seol and M.S. Shephard. *Parallel anisotropic 3D mesh adaptation by mesh modification.* Eng. w. Comp., vol. 21, no. 3, pages 247–258, 2006.

[Alauzet 2010] F. Alauzet. *Size gradation control of anisotropic meshes.* Finite Elem. Anal. Des., vol. 46, pages 181–202, 2010.

[Alauzet 2011] F. Alauzet and G. Olivier. *Extension of Metric-Based Anisotropic Mesh Adaptation to Time-Dependent Problems Involving Moving Geometries.* In 49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition. American Institute of Aeronautics and Astronautics, 2014/03/26 2011.

[Alleaume 2008] A. Alleaume, L. Francez, M. Loriot and N. Maman. *Automatic tetrahedral out-of-core meshing.* In M. L. Brewer and D. Marcum, editeurs, Proceedings of the 16th International Meshing Roundtable, pages 461–476. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Ansys Inc.] Ansys Inc. *Ensight.* https://www.ansys.com/en/products/platform/ansys-ensight.

[Aparicio-Estrems 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.

[Arsigny 2006] V. Arsigny, P. Fillard, X. Pennec and N. Ayache. *Log-Euclidean Metrics for Fast and Simple Calculus on Diffusion Tensors.* Magn. Reson. Med., vol. 56, no. 2, pages 411–421, 2006.

[Aubry 2009] R. Aubry and R. Löhner. *Generation of viscous grids at ridges and corners.* International Journal for Numerical Methods in Engineering, vol. 77, no. 9, pages 1247–1289, 2009.

[Aubry 2011] R. Aubry, G. Houzeaux and M. Vázquez. *A surface remeshing approach.* International Journal for Numerical Methods in Engineering, vol. 85, no. 12, pages 1475–1498, 2011.

[Aubry 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.

[Aubry 2016] R. Aubry, S. Dey, K. Karamete and E. Mestreau. *Smooth anisotropic sources with application to three-dimensional surface mesh generation.* Engineering with Computers, vol. 32, no. 2, pages 313–330, 2016.

[Baffet 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.

[Baker 1987] T. Baker. *Three-dimensional mesh generation by triangulation of arbitrary point sets.* AIAA Paper , vol. 1987-1124, 1987.

[Bassi 1997] F. Bassi and S. Rebay. *High-order accurate discontinuous finite element solution of the 2D Euler equations.* Journal of computational physics, vol. 138, no. 2, pages 251–285, 1997.

[Beaufort 2017] P.-A. Beaufort, J. Lambrechts, F. Henrotte, C. Geuzaine and J.-F. Remacle. *Computing cross fields A PDE approach based on the Ginzburg-Landau theory.* Procedia Engineering, vol. 203, pages 219–231, 2017.

[Borouchaki 2000a] 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, vol. 49, no. 1-2, pages 233–259, 2000.

[Borouchaki 2000b] H. Borouchaki, P. Laug and P.L. George. *Parametric surface meshing using a combined advancing-front – generalized-Delaunay approach.* Int. J. Numer. Meth. Engng, vol. 49, no. 1-2, pages 233–259, 2000.

[Bottasso 2002] C.L. Bottasso and D. Detomi. *A procedure for tetrahedral boundary layer mesh generation.* Engineering Computations, vol. 18, pages 66–79, 2002.

[Bottasso 2004] C.L. Bottasso. *Anisotropic mesh adaption by metric-driven optimization.* Int. J. Numer. Meth. Engng, vol. 60, pages 597–639, 2004.

[Bowyer 1981] A. Bowyer. *Computing Dirichlet tessellations.* Comput. J., vol. 24, no. 2, pages 162–166, 1981.

[Cao 2005] W. Cao. *On the Error of Linear Interpolation and the Orientation, Aspect Ratio, and Internal Angles of a Triangle.* SIAM J. Numer. Anal., vol. 43, no. 1, pages 19–40, 2005.

[Cao 2007] W. Cao. *An interpolation error estimate on anisotropic meshes in $\mathbb{R}^n$ and optimal metrics for mesh refinement.* SIAM J. Numer. Anal., vol. 45, no. 6, pages 2368–2391 (electronic), 2007.

[Cao 2008] W. Cao. *An interpolation error estimate in $\mathbb{R}^2$ based on the anisotropic measures of higher order derivatives.* Math. Comp., vol. 77, no. 261, pages 265–286 (electronic), 2008.

[Castro-Díaz 1997] M.J. Castro-Díaz, F. Hecht, B. Mohammadi and O. Pironneau. *Anisotropic Unstructured Mesh Adaptation for Flow Simulations.* Int. J. Numer. Meth. Fluids, vol. 25, pages 475–491, 1997.

[Chen 2007] L. Chen, P. Sun and J. Xu. *Optimal anisotropic meshes for minimizing interpolation errors in L^p-norm.* Math. Comp., vol. 76, no. 257, pages 179–204, 2007.

[Chernikov 2010] A.N. Chernikov and N.P. Chrisochoides. *A template for developing next generation parallel Delaunay refinement methods.* Finite Elements in Analysis and Design, vol. 46, no. 1–2, pages 96 – 113, 2010. Mesh Generation - Applications and Adaptation.

[Ciarlet 1978] P.G. Ciarlet. The finite element method for elliptic problems. North-Holland, Amsterdam, 1978.

[Compère 2010] G. Compère, J.-F. Remacle, J. Jansson and J. Hoffman. *A mesh adaptation framework for dealing with large deforming meshes.* Int. J. Numer. Meth. Engng, vol. 82, pages 843–867, 2010.

[Coupez 2000] T. Coupez, H. Digonnet and R. Ducloux. *Parallel meshing and remeshing.* Applied Mathematical Modelling, vol. 25, no. 2, pages 153 – 175, 2000. Dynamic load balancing of mesh-based applications on parallel.

[Coupez 2011] T. Coupez. *Metric construction by length distribution tensor and edge based error for anisotropic adaptive meshing.* Journal of Computational Physics, vol. 230, no. 7, pages 2391 – 2405, 2011.

[Cournède 2006] P.-H. Cournède, B. Koobus and A. Dervieux. *Positivity statements for a Mixed-Element-Volume scheme on fixed and moving grids.* European Journal of Computational Mechanics, vol. 15, no. 7-8, pages 767–798, 2006.

[Csimsoft] Csimsoft. *Trelis.* https://www.csimsoft.com/trelis.

[Dantzig 2003] G. B. Dantzig and M. N. Thapa. Linear programming 2: Theory and extensions. Springer-Verlag, 2003.

[De Cougny 1999] H. L. De Cougny and M. S. Shephard. *Parallel refinement and coarsening of tetrahedral meshes.* International Journal for Numerical Methods in Engineering, vol. 46, no. 7, pages 1101–1125, 1999.

[de Siqueira 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.

[de Siqueira 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.

[Dey 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.

[Digonnet 2013] Hugues Digonnet, Luisa Silva and Thierry Coupez. *Massively parallel computation on anisotropic meshes*. In 6th International Conference on Adaptive Modeling and Simulation, pages 199–211, Lisbon, Portugal, June 2013. International Center for Numerical Methods in Engineering.

[Distene] Distene. *MeshGems suite*. http://www.meshgems.com/.

[do Carmo 1976] M. do Carmo. Differential geometry of curves and surfaces. Prentice Hall, 1976.

[Dobrzynski 2008] C. Dobrzynski and P.J. Frey. *Anisotropic Delaunay Mesh Adaptation for Unsteady Simulations*. In Proceedings of the 17th International Meshing Roundtable, pages 177–194. Springer, 2008.

[Dolejsi 2015] V. Dolejsi. *Anisotropic hp-adaptive discontinuous Galerkin method for the numerical solution of time dependent PDEs*. Applied Mathematics and Computation, vol. 267, pages 682 – 697, 2015. The Fourth European Seminar on Computing (ESCO 2014).

[Dompierre 1999] J. Dompierre, P. Labbé, M.-G. Vallet and R. Camarero. *How to Subdivide Pyramids, Prisms, and Hexahedra into Tetrahedra*. In Proceedings of the 8th International Meshing Roundtable, pages 195–204, 1999.

[Encinas 2003] L. H. Encinas and J. M. Masque. *A short proof of the generalized Faà di Bruno's formula*. Applied mathematics letters, vol. 16, no. 6, pages 975–979, 2003.

[Faà di Bruno 1857] F. Faà di Bruno. *Note sur une nouvelle formule de calcul différentiel*. Quarterly J. Pure Appl. Math, vol. 1, no. 359-360, page 12, 1857.

[Fortunato 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.

[Foteinos 2012] P. Foteinos and N.P. Chrisochoides. *Dynamic Parallel 3D Delaunay Triangulation*. In WilliamRoshan Quadros, editeur, Proceedings of the 20th International Meshing Roundtable, pages 3–20. Springer Berlin Heidelberg, 2012.

[Freitag 1997] L. Freitag and C. Olliver Gooch. *Tetrahedral mesh improvement using swapping and smoothing*. Int. J. Numer. Meth. Engng, vol. 40, no. 21, pages 3979–4002, 1997.

[Frey 2000] P. Frey. *About surface remeshing*. In Proceedings of the 15th International Meshing Roundtable, pages 123–136. Springer, 2000.

[Frey 2001a] P. J. Frey. *Medit: An interactive mesh visualization software, INRIA Technical Report RT0253*, 2001.

[Frey 2001b] P.J. Frey. `Yams`, *A fully automatic adaptive isotropic surface remeshing procedure.* RT-0252, INRIA, November 2001.

[Frey 2003] P. Frey and H. Borouchaki. *Surface meshing using a geometric error estimate.* Int. J. Numer. Meth. Engng, vol. 58, no. 2, pages 227–245, 2003.

[Frey 2005] P.J. Frey and F. Alauzet. *Anisotropic mesh adaptation for CFD computations.* Comput. Methods Appl. Mech. Engrg., vol. 194, no. 48-49, pages 5068–5082, 2005.

[Frey 2008] P. Frey and P.-L. George. Mesh generation. Application to finite elements. ISTE Ltd and John Wiley & Sons, 2nd édition, 2008.

[Gargallo-Peiró 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.

[Garimella 2000] R.V. Garimella and M.S. Shephard. *Boundary layer mesh generation for viscous flow simulations.* Int. J. Numer. Meth. Fluids, vol. 49, pages 193–218, 2000.

[George 1990] P.L. George, F. Hecht and E. Saltel. *Fully Automatic Mesh Generator for 3D Domains of Any Shape.* Impact of Comuting in Science and Engineering, vol. 2, pages 187–218, 1990.

[George 1998] P.L. George and H. Borouchaki. Delaunay triangulation and meshing : application to finite elements. Hermès Science, Paris, Oxford, 1998.

[George 2003a] P. L. George, H. Borouchaki and H. Saltel. *'Ultimate' robustness in meshing an arbitrary polyhedron.* International Journal for Numerical Methods in Engineering, vol. 58, no. 7, pages 1061–1089, 2003.

[George 2003b] P.L. George. `Gamanic3d`, *Adaptive anisotropic tetrahedral mesh generator.* Technical Note, INRIA, 2003.

[George 2003c] P.L. George and H. Borouchaki. *Back to Edge Flips in 3 Dimensions.* In Proceedings of the 12th International Meshing Roundtable, pages 393–402, Santa Fe, NM, USA, 2003.

[George 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.

[Geuzaine 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, no. 11, pages 1309–1331, 2009.

[Haimes 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.

[Hartmann 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, no. 6, pages 316–333, 2016.

[Hecht 1998] F. Hecht. `BAMG`*: bidimensional Anisotropic Mesh Generator.* Available from http://www-rocq.inria.fr/gamma/cdrom/www/bamg/eng.htm, INRIA-Rocquencourt, France, 1998.

[Hecht 2014] F. Hecht and R. Kuate. *An approximation of anisotropic metrics from higher order interpolation error for triangular mesh adaptation.* J. Comput. Appl. Math., vol. 258, pages 99–115, 2014.

[Hermeline 1982] F. Hermeline. *Triangulation automatique d'un polyèdre en dimension N.* RAIRO - Analyse numérique, vol. 16, no. 3, pages 211–242, 1982.

[Hunton 1973] L.W. Hunton, R.M. Hicks and J.P. Mendoza. *Some effects of wing planform on sonic boom.* TN. D-7160, Nasa, 1973.

[Ims 2019] J. Ims and Z. J. Wang. *Automated low-order to high-order mesh conversion.* Engineering with Computers, vol. 35, no. 1, pages 323–335, Jan 2019.

[Ing.] Lemma Ing. *ANANAS.* `http://www.lemma-ing.com`.

[Intelligent Light] Intelligent Light. *FieldView.* `http://www.ilight.com/en/products/fieldview-18`.

[Ito 2002] Y. Ito and K. Nakahashi. *Unstructured mesh generation for viscous flow computations.* Proceedings of the 11th International Meshing Roundtable, pages 367–377, 2002.

[Ito 2006] Y. Ito and K. Nakahashi. *An approach to generate high quality unstructured hybrid meshes.* 44th AIAA Aerospace Sciences Meeting, Jan 2006.

[Ito 2007] Y. Ito, A.M. Shih, A.K. Erukala, B.K. Soni, A.N. Chernikov, N.P.Chrisochoides and K. Nakahashi. *Parallel Unstructured Mesh Generation by an Advancing Front Method.* Math. Comput. Simul., vol. 75, no. 5-6, pages 200–209, September 2007.

[Johnson 1985] C. Johnson, U. Navert and J. Pitkaranta. *Finite element methods for linear hyperbolic problems.* Math. Comp., vol. 69, pages 25–39, 1985.

[Jones 2006] W.T. Jones, E.J. Nielsen and M.A. Park. *Validation of 3D Adjoint Based Error Estimation and Mesh Adaptation for Sonic Boom Reduction.* In 44th AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2006-1150, Reno, NV, USA, Jan 2006.

[Karman 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.

[Karypis 1998] G. Karypis and V. Kumar. *A fast and high quality multilevel scheme for partitioning irregular graphs*. SIAM Journal on Scientific Computing, vol. 20, no. 1, pages 359–392, 1998.

[Kirkpatrick 1983] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. *Optimization by Simulated Annealing*. Science, vol. 220, no. 4598, pages 671–680, 1983.

[KitWare Inc.] KitWare Inc. *ParaView*. `https://www.paraview.org/`.

[Krause 2003] Jens Krause and Paul Louis George. *Construction d'un maillage 3-D anisotrope localement structuré*. Rapport de recherche RR-4834, INRIA, 2003.

[Lachat 2014] C. Lachat, C. Dobrzynski and F. Pellegrini. *Parallel mesh adaptation using parallel graph partitioning*. In 5th European Conference on Computational Mechanics (ECCM V), volume 3 of *Minisymposia in the frame of ECCM V*, pages 2612–2623, Barcelone, Spain, July 2014. IACM & ECCOMAS, CIMNE - International Center for Numerical Methods in Engineering. ISBN 978-84-942844-7-2.

[Laug 2003] P. Laug and H. Bourochaki. `BL2D-V2`*, Mailleur bidimensionnel adaptatif*. RT-0275, INRIA, 2003.

[Laug 2010] P. Laug. *Some aspects of parametric surface meshing*. Finite Elements in Analysis and Design, vol. 46, no. 1–2, pages 216 – 226, 2010. Mesh Generation - Applications and Adaptation.

[Lenoir 1986] M. Lenoir. *Optimal isoparametric finite elements and error estimates for domains involving curved boundaries*. SIAM journal on numerical analysis, vol. 23, no. 3, pages 562–580, 1986.

[Li 2004] X. Li, J.-F. Remacle, N. Chevaugeon and M.S. Shephard. *Anisotropic Mesh Gradation Control*. In Proceedings of the 13th International Meshing Roundtable, pages 401–412. Springer, 2004.

[Li 2005] X. Li, M.S. Shephard and M.W. Beal. *3D anisotropic mesh adaptation by mesh modification*. Comput. Methods Appl. Mech. Engrg., vol. 194, no. 48-49, pages 4915–4950, 2005.

[Lo 2015] D. S.H. Lo. Finite element mesh generation. CRC Press, 2015.

[Löhner 1988a] 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, no. 10, pages 1135–1149, 1988.

[Löhner 1988b] R. Löhner and P. Parikh. *Three-dimensionnal grid generation by the advancing-front method*. Int. J. Numer. Meth. Fluids, vol. 8, no. 8, pages 1135–1149, 1988.

[Löhner 1992] R. Löhner, J. Camberos and M. Merriam. *Parallel unstructured grid generation.* Computer Methods in Applied Mechanics and Engineering, vol. 95, no. 3, pages 343 – 357, 1992.

[Löhner 1993] R. Löhner. *Matching semi-structured and unstructured grids for Navier-Stokes calculations.* AIAA Paper , vol. 1993-3348, 1993.

[Löhner 1999] R. Löhner. *Generation of unstructured grids suitable for RANS calculations.* AIAA Paper , vol. 1999-0662, 1999.

[Löhner 2001] R. Löhner. Applied CFD techniques. An introduction based on finite element methods. John Wiley & Sons, Ltd, New York, 2001.

[Löhner 2013] R. Löhner. *A 2nd Generation Parallel Advancing Front Grid Generator.* In X. Jiao and J.-C. Weill, editeurs, Proceedings of the 21st International Meshing Roundtable, pages 457–474. Springer Berlin Heidelberg, 2013.

[Löhner 2014] R. Löhner. *Recent Advances in Parallel Advancing Front Grid Generation.* Archives of Computational Methods in Engineering, vol. 21, no. 2, pages 127–140, 2014.

[Loseille 2016] A. Loseille, H. Guillard and A. Loyer. *An introduction to Vizir: an interactive mesh visualization and modification software.* EOCOE, Rome, Italy, 2016.

[Luo 1998] H. Luo, J.D. Baum and R. Löhner. *A fast, matrix-free implicit method for compressible flows on unstructured grids.* J. Comp. Phys., vol. 146, pages 664–690, 1998.

[Marcum 1996] D. L. Marcum. *Adaptive Unstructured Grid Generation for Viscous Flow Applications.* AIAA Journal, vol. 34, no. 8, pages 2440–2443, 1996.

[Marcum 2001] D. L. Marcum. *Efficient Generation of High-Quality Unstructured Surface and Volume Grids.* Engrg. Comput., vol. 17, pages 211–233, 2001.

[Marcum 2013] D. L. Marcum and F. Alauzet. *Unstructured Mesh Generation Using Advancing Layers and Metric-Based Transition for Viscous Flowfields.* In 21st AIAA Computational Fluid Dynamics Conference. 2013.

[Marcum 2014] D. Marcum and F. Alauzet. *Aligned Metric-Based Anisotropic Solution Adaptive Mesh Generation.* In Proceedings of the 23rd International Meshing Roundtable. 2014.

[Maunoury 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.

[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.

[Mavriplis 1995] D.J. Mavriplis. *An advancing front Delaunay triangulation algorithm designed for robustness.* J. Comp. Phys., vol. 117, pages 90–101, 1995.

[Michal 2011] T. Michal and J. Krakos. *Anisotropic mesh Adaptation through edge primitive operations.* AIAA Paper , vol. 2011-0159, 2011.

[Miranda 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.

[Moxey 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.

[Nelson 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, no. 12, pages 1803–11, 2011.

[Nelson 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, no. 12, pages 2325–2334, 2012.

[Ovcharenko 2013] A. Ovcharenko, K. Chitale, O. Sahni, K. E. Jansen and M. S. Shephard. Parallel adaptive boundary layer meshing for cfd analysis, pages 437–455. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[Özturan 1994] C. Özturan, H.L. deCougny, M.S. Shephard and J.E. Flaherty. *Parallel adaptive mesh refinement and redistribution on distributed memory computers.* Computer Methods in Applied Mechanics and Engineering, vol. 119, no. 1–2, pages 123 – 137, 1994.

[Pain 2001] C.C Pain, A.P. Humpleby, C.R.E. de Oliveira and A.J.H. Goddard. *Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations.* Comput. Methods Appl. Mech. Engrg., vol. 190, pages 3771–3796, 2001.

[Peiró 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.

[Peraire 1987] J. Peraire, M. Vahdati, K. Morgan and O.C. Zienkiewicz. *Adaptive Remeshing for Compressible Flow Computations.* J. Comput. Phys., vol. 72, pages 449–466, 1987.

[Persson 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.

[Piegl 1997] L. Piegl and W. Tiller. The nurbs book (2nd ed.). Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[Pirzadeh 1994] S. Pirzadeh. *Viscous unstructured three dimensional grids by the advancing-layers method.* 32th AIAA Aerospace Sciences Meeting, Jan 1994.

[PointWise] PointWise. *Structured, Hybrid, and Overset Meshing.* http://www.pointwise.com/.

[Rangarajan 2018] A. M. Rangarajan, A. Balan and G. May. *Mesh Adaptation and Optimization for Discontinuous Galerkin Methods Using a Continuous Mesh Model.* In AIAA Modeling and Simulation Technologies Conference. 2018.

[Reid 1978] L. Reid and R. D. Moore. *Performance of single-stage axial-flow transonic compressor with rotor and stator aspect ratios of 1.19 and 1.26, respectively, and with design pressure ratio of 1.82.* 1978.

[Remacle 2005] J.-F. Remacle, X. Li, M.S. Shephard and J.E. Flaherty. *Anisotropic adaptive simulation of transient flows using discontinuous Galerkin methods.* Int. J. Numer. Meth. Engng, vol. 62, pages 899–923, 2005.

[Remacle 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, no. 5, pages 750–771, 2007.

[Remacle 2012] J.-F. Remacle, J. Lambrechts, B. Seny, E. Marchandise, A. Johnen and C. Geuzaine. *Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm.* International Journal for Numerical Methods in Engineering, vol. 89, no. 9, pages 1102–1119, 2012.

[Remacle 2015] J.-F. Remacle, V. Bertrand and C. Geuzaine. *A Two-Level Multithreaded Delaunay Kernel.* Procedia Engineering, vol. 124, pages 6 – 17, 2015.

[Ringue 2017] N. Ringue and S. Nadarajah. *Optimization-based Anisotropic hp-Adaptation for High-Order Methods.* In 23rd AIAA Computational Fluid Dynamics Conference. 2017.

[Roca 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, editeur, Proceedings of the 20th International Meshing Roundtable, pages 365–383. Springer Berlin Heidelberg, 2012.

[Rokos 2015] G. Rokos, G. J. Gorman, K. E. Jensen and P. H. J. Kelly. *Thread Parallelism for Highly Irregular Computation in Anisotropic Mesh Adaptation.* CoRR, vol. abs/1505.04694, 2015.

[Ruiz-Gironès 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.

[Ruiz-Gironès 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.

[Ruiz-Gironès 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.

[Rumsey] C. L. Rumsey, J. P. Slotnick and A. J. Sclafani. *Overview and Summary of the Third AIAA High Lift Prediction Workshop.* In 2018 AIAA Aerospace Sciences Meeting.

[Sagaut 2001] P. Sagaut. Large-eddy simulation for incompressible flows? an introduction. Springer, 2001.

[Sahni 2010] O. Sahni, X. J. Luo, K. E. Jansen and M. S. Shephard. *Curved Boundary Layer Meshing for Adaptive Viscous Flow Simulations.* Finite Elem. Anal. Des., vol. 46, no. 1-2, pages 132–139, January 2010.

[Schroeder 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, no. 4, pages 446–460, 2006.

[Sharbatdar 2013] M. Sharbatdar and C. Ollivier Gooch. *Anisotropic mesh adaptation: recovering quasi-structured meshes.* AIAA Paper , vol. 2013-0149, 2013.

[Shephard 2013] M.S. Shephard, C. Smith and J.E. Kolb. *Bringing HPC to Engineering Innovation.* Computing in Science Engineering, vol. 15, no. 1, pages 16–25, Jan 2013.

[Sherwin 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, no. 1, pages 207–223, 2002.

[Shu 1988] C.W. Shu and S. Osher. *Efficient implementation of essentially non-oscillatory shock-capturing schemes.* J. Comput. Phys., vol. 77, pages 439–471, 1988.

[Si 2015] H. Si. *TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator.* ACM Trans. Math. Softw., vol. 41, no. 2, pages 11:1–11:36, February 2015.

[Slotnick 2014] J P Slotnick, A Khodadoust, J J Alonso, D L Darmofal, W D Gropp, E A Lurie and D J Mavriplis. CFD vision 2030 study: A Path to Revolutionary Computational Aerosciences, 2014.

[Sokolov 2017] D. Sokolov, N. Ray, L. Untereiner and B. Lévy. *Hexahedral-dominant Meshing.* ACM Trans. Graph., vol. 36, no. 4, 2017.

[Tam 2000] A. Tam, D. Ait-Ali-Yahia, M.P. Robichaud, M. Moore, V. Kozel and W.G. Habashi. *Anisotropic mesh adaptation for 3D flows on structured and unstructured grids.* Comput. Methods Appl. Mech. Engrg., vol. 189, pages 1205–1230, 2000.

[TecPlot Inc.] TecPlot Inc. *TecPlot.* https://www.tecplot.com/.

[Toulorge 2013] Thomas Toulorge, Christophe Geuzaine, Jean-François Remacle and Jonathan Lambrechts. *Robust Untangling of Curvilinear Meshes.* J. Comput. Phys., vol. 254, pages 8–26, December 2013.

[Toulorge 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.

[Tremel 2007] U. Tremel, K.A. Sørensen, S. Hitzel, H. Rieger, O. Hassan and N.P. Weatherill. *Parallel remeshing of unstructured volume grids for CFD applications.* International Journal for Numerical Methods in Fluids, vol. 53, no. 8, pages 1361–1379, 2007.

[Tristano 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.

[Turner 2016] M. Turner, J. Peirò and D. Moxey. *A Variational Framework for High-order Mesh Generation.* Procedia Engineering, vol. 163, no. Supplement C, pages 340 – 352, 2016. 25th International Meshing Roundtable.

[Vallet 2007] M.-G. Vallet, C.-M. Manole, J. Dompierre, S. Dufour and F. Guibault. *Numerical comparison of some Hessian recovery techniques.* Int. J. Numer. Meth. Engng, vol. 72, pages 987–1007, 2007.

[Vanharen 2017] J. Vanharen. *High-order numerical methods for unsteady flows around complex geometries.* Ph.D. Thesis, Université de Toulouse, 2017.

[Verfürth 1996] R. Verfürth. A review of *a posteriori* error estimation and adaptative mesh-refinement techniques. Wiley Teubner Mathematics, New York, 1996.

[Vlachos 2001a] 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.

[Vlachos 2001b] A. Vlachos, J. Peters, C. Boyd and J. L. Mitchell. *Curved PN Triangles.* In Proceedings of the 2001 Symposium on Interactive 3D Graphics, I3D '01, pages 159–166, New York, NY, USA, 2001. ACM.

[Wang 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, no. 41-43, pages 5621–5633, 2006.

[Watson 1981] D.F. Watson. *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes.* Comput. J., vol. 24, no. 2, pages 167–172, 1981.

[Woodward 1984] P. Woodward and P. Colella. *The numerical simulation of two-dimensional fluid flow with strong shocks.* Journal of Computational Physics, vol. 54, no. 1, pages 115 – 173, 1984.

[Xie 2013] Z. Q. Xie, R. Sevilla, O Hassan and K. Morgan. *The Generation of Arbitrary Order Curved Meshes for 3D Finite Element Analysis.* Comput. Mech., vol. 51, no. 3, pages 361–374, March 2013.

[Yano 2012] M. Yano and D.L. Darmofal. *An optimization-based framework for anisotropic simplex mesh adaptation.* J. Comp. Phys., vol. 231, no. 22, pages 7626–7649, 2012.

[Zwanenburg 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.