Numerical Methods and Mesh Adaptation for Reliable RANS Simulations

Victorien Menier

NUMERICAL METHODS AND MESH ADAPTATION FOR RELIABLE RANS SIMULATIONS.

THÈSE

présentée pour obtenir le titre de

DOCTEUR

de l'Université Pierre et Marie Curie, Paris VI

École doctorale : Sciences Mathématiques de Paris Centre

Spécialité : Mathématiques Appliquées

par

Victorien Menier

soutenue le 23 Novembre 2015 devant le jury composé de :

Directeurs:

Frédéric Alauzet	Directeur de recherche	Inria Paris Rocquencourt
Adrien Loseille	Chargé de recherche	Inria Paris Rocquencourt

Rapporteurs:

Bruno Koobus	Professeur	Université de Montpellier
Marco Picasso	Professeur	École Polytechnique Fédérale de Lausanne

Examinateurs:

Jean-Frédéric Gerbeau	Directeur de Recherche	Inria Paris Rocquencourt & UPMC
Dave Marcum	Professeur	Mississippi State University
Gilbert Rogé	Ingénieur de Recherche	Dassault Aviation

Résumé

Cette thèse porte sur la prédiction haute-fidélité de phénomènes visqueux turbulents modélisés par les équations Reynolds-Averaged Navier-Stokes (RANS). Si l'adaptation de maillage a été appliquée avec succès aux simulations non-visqueuses comme la prédiction du bang sonique ou la propagation d'explosion, prouver que ces méthodes s'étendent et s'appliquent également aux simulations RANS avec le même succès reste un problème ouvert. Dans ce contexte, cette thèse traite des problématiques relatives aux méthodes numériques (solveur de mécanique des fluides) et aux stratégies d'adaptation de maillage.

Pour les méthodes numériques, nous avons implémenté un modèle de turbulence dans notre solveur et nous avons conduit une étude de vérification et validation en deux et trois dimensions avec comparaisons à l'expérience. Des bons résultats ont été obtenus sur un ensemble de cas tests, notamment sur le calcul de la traînée pour des géométries complexes. Nous avons également amélioré la robustesse et la rapidité de convergence du solveur, grâce à une intégration en temps implicite, et grâce à une procédure d'accélération multigrille.

En ce qui concerne les stratégies d'adaptation de maillage, nous avons couplé les méthodes multigrilles à la boucle d'adaptation dans le but de bénéficier des propriétés de convergence du multigrille, et ainsi, améliorer la robustesse du processus et le temps CPU des simulations. Nous avons également développé un algorithme de génération de maillage en parallèle. Celui-ci permet de générer des maillages anisotropes adaptés d'un milliard d'éléments en moins de 20 minutes sur 120 coeurs de calcul. Enfin, nous avons proposé une procédure pour générer automatiquement des maillages anisotropes adaptés quasi-structurés pour les couches limites.

Abstract

This thesis deals with the high-fidelity prediction of viscous turbulent flows modeled by the Reynolds-Averaged Navier-Stokes (RANS) equations. Although mesh adaptation has been successfully applied to inviscid simulations like sonic boom prediction or blast propagation, demonstrating that these methods are also well-suited for 3D RANS simulations remains a challenge. This thesis addresses research issues that arise in this context, which are related to both numerical methods (flow solver) and mesh adaptation strategies.

For the numerical methods, we have implemented a turbulence model in our in-house flow solver and carried out its verification & validation study. Accurate results were obtained for an extensive set of test cases, including the drag prediction workshop. Additional developments have been done to improve the robustness and the convergence speed of the flow solver. They include the implementation of an implicit time integration and a multigrid acceleration procedure.

As regards mesh adaptation, we have coupled the adaptive process to a multigrid procedure in order to benefit from its convergence properties and thus improve robustness while avoiding wasted computational effort. We also have devised a parallel mesh generation algorithm. We are able to generate anisotropic adapted meshes containing around one billion elements in less than 20min on 120 cores. Finally, we have introduced a procedure to automatically generate anisotropic adapted quasi-structured meshes in boundary layer regions.

Introduction

This thesis deals with Computational Fluid Dynamics (CFD) and more specifically with the issue of the high-fidelity prediction of viscous turbulent flows - modeled by the Reynolds-Averaged Navier-Stokes (RANS) equations - in an adaptive context.

After a brief review of the design process of aircraft, we recall the standard computational pipeline. We emphasize the critical roles of the flow solver and the mesh generation step in this pipeline, which are the core of this thesis. Then, we briefly introduce mesh adaptation and the research issues that occur when viscous turbulent flows are considered. Finally, we give an outline of the thesis and list the main contributions.

Industrial and Scientific Context

Aviation plays an important role in our society, as it supports both freight transportation and private travel. In 2014, airlines transported 3.3 billion passengers and 50 million tons of cargo across a network of almost 50,000 routes [3]. An aircraft is a major investment, as it is expected to fly for around 25 years. Therefore, its conception must take into account a large number of constraints, which have a dramatic impact on the safety, the operating cost, as well as the environmental footprint of the aircraft. For instance, reduced drag (i.e. a better air penetration) means less fuel burnt, and better lift properties (i.e. the force perpendicular to the flow, "what makes an aircraft fly") make it possible to carry more passengers, baggage, cargo or mail, while consuming the same amount of fuel.

These considerations must be addressed during the design phase. This design phase has long consisted in analytical theory calculations, together with a large number of experimentations. A prototype is built and tested in a **wind tunnel** (see Figure 1), and is then redesigned several times, until no more unanticipated test results are observed. Since the early days of aviation, wind tunnels have thus played a critical role in the design process of an aircraft. Major wind-tunnel facilities were constructed both in Europe and in the United States to support the aeronautical revolution of the 20th century [85, 133]. New wind tunnels with increased power and application ranges were designed and constructed up to

the 80s. A 1994 national study of aeronautic R&D facilities conducted in the United States recommended the construction of large wind tunnels at a cost of about $3.2 billion [1]. However, this study was never followed up and the overall trend over the last three decades has been to close wind-tunnel facilities.

The reasons for the decreasing use of wind tunnels are many, and include:

- **The operating and maintenance costs:** a day in a large transonic wind-tunnel costs about $100,000, due to the huge electrical power consumption. Moreover, maintenance costs have kept increasing for the aging wind-tunnel inventory.
- **The rapid advance of Computational Fluid Dynamics (CFD)**: since the 1960s, numerical computations have played an ever more important role in the design process, supported by the increasing computational resources available as well as the development of advanced algorithms [32]. In a 2012 study [63] sponsored by the American National Science Fondation (NSF) and other agencies (including NASA), a panel of experts stated that *"computer simulation is more pervasive today -and having more impact- than any time in the human history"*.

Although wind tunnels are being used less and less, it should be noted that experimentation still plays an important role, particularly in the validation process of CFD codes [146] (see Chapter 4).

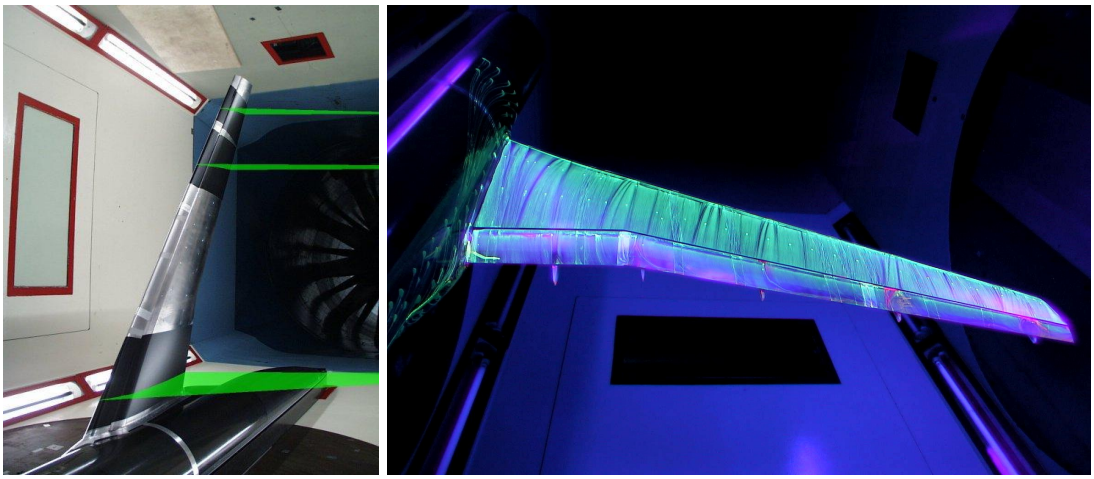


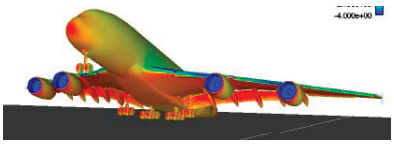
Figure 1: Wind tunnels are tube-shaped facilities, which use powerful fans to move air over a model of aircraft. It provides a lot of information about an aircraft aerodynamics, and is extensively used during the design process. (Image credit: NASA).

CFD [92, 74], which has progressed rapidly over the last four decades, has fundamentally changed the aircraft design process and is partially responsible for the aforementioned reduction in the amount of wind-tunnel testing during the same period. Not only does it enable reductions in ground-based and in-flight testing, but it also provides more physical insight, thus enabling superior designs at reduced cost and risk.

CFD is used to accurately predict the flow around the aircraft geometry, and thus deduce the aircraft features, in terms of fuel consumption, environmental impact, noise, operating cost, etc. It consists in solving numerically partial differential equations (PDEs), such as the Navier-Stokes equations, in the continuum medium framework. As it is impossible to solve continuously the physical equations at every point of the computational domain (i.e. the space around the aircraft for instance), the domain is discretized. It is approximated by a mesh, i.e. a collection of vertices and elements (triangles, tetrahedra, ...). Then, the solution is computed at each vertex or element.

High-fidelity prediction of the flow field using CFD consists in four successive steps: (1) the CAD (Computer Aided Design) model of the geometry is defined, (2) a mesh of the geometry and of the domain around it is generated, (3) a solution (i.e. the flow pressure, density, velocity etc.) is computed at each vertex of the mesh, and (4) the solution is visualized and analyzed. Figure 2 presents the example of the design of the Airbus A380.

(a) Wind tunnel experiments.



(b) CAD modeling, numerical simulations.

(c) Real-life: A380 in take-off configuration.

Figure 2: The design phase of the Airbus A380 is a combination of experiments and simulations (Credit: Airbus).

Note that a mesh is an essential component of the CFD pipeline [112]. A mesh is a discretization of the computational domain, which is composed (for instance) of triangles and edges in 2D, and of tetrahedra and triangles in 3D (Figure 3 presents two examples). During a CFD simulation, the solution is computed at each vertex of the mesh (or at each element, depending on the numerical scheme used). Increasing the density of the mesh (vertices, elements) increases the accuracy of the numerical solution. Moreover, the mesh quality and its adequation with the underlying physics strongly impacts the quality of the numerical solution provided by the numerical flow solver. Consequently, modifying judiciously the features of the mesh may improve the accuracy of the simulation: this is within the scope of mesh adaptation.

Anisotropic mesh adaptation [35, 60, 65, 88, 7] consists in modifying the density and orientation of the discretization in order to increase the accuracy of the prediction. The need for mesh adaptation is motivated by the nature of the physical phenomena involved (boundary layers, shock waves, contact discontinuities, wake, vortices, etc.), which are located in **small regions** of the computational domain, and are **anisotropic** (i.e. directionally dependent). A typical example is the numerical prediction of sonic

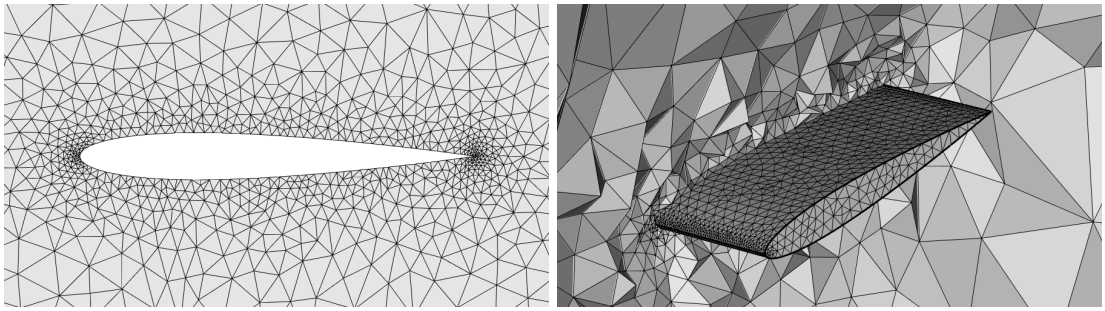


Figure 3: **Examples of meshes**
2D (left) and 3D (right) meshes of a NACA 0012 airfoil.

boom. Such a simulation presents highly anisotropic shock waves (see Figure 4a) along with large variations of the problem scales: they present millimeter (near the aircraft) to kilometer (in the atmosphere) variations. Standard CFD approaches are only able to predict the pressure distribution at one-aircraft-length distance below the jet at most. Beyond that, the signal is lost due to inadequate mesh resolution. Some hybrid procedures [166] consist in coupling CFD for the near-field region to linear propagation for the far-field region (see Figure 4b). But these coupled methods lack precision, since some assumptions must be made for the coupling, as well as the use of isotropic meshes for CFD simulations, leading to a lot of numerical dissipation when propagating the shock wave. By using anisotropic mesh adaptation, a CFD solution is computed on the whole domain (which is a few kilometers high) and the shock waves are accurately predicted from the aircraft to several kilometers below.

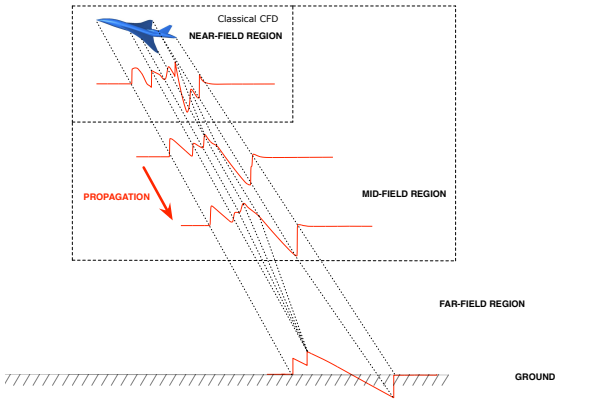
Sonic boom prediction is one example - among others (blast propagation, acoustic waves, etc.) - showing the power of mesh adaptation for **inviscid simulations**. To summarize, it has the ability (i) to substantially reduce the tradeoff between the accuracy of the solution and the number of degrees of freedom, thus reducing the CPU time, (ii) to optimize the numerical scheme dissipation by automatically taking into account the anisotropy of the physics in the mesh generation, and (iii) to reach high-order asymptotic convergence (see [46, 101]) for non-smooth flows.

However, demonstrating that mesh adaptation is also well-suited to 3D Reynolds-Averaged Navier-Stokes (RANS) simulations represents a huge step forward, and this PhD aims at tackling some of the numerous research issues that remain.

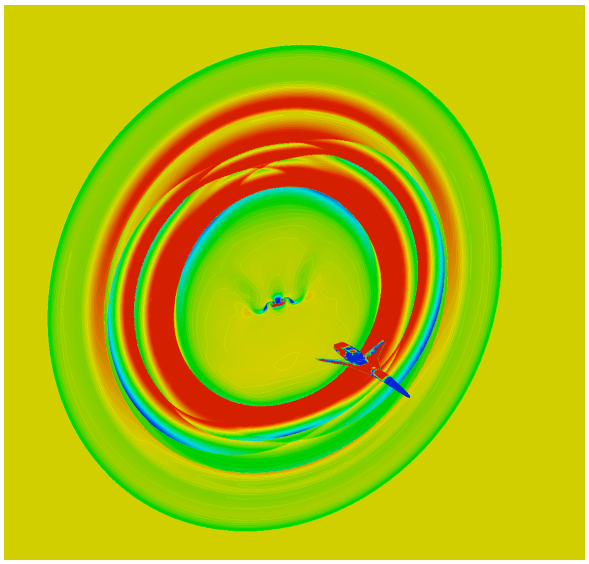
Research issues

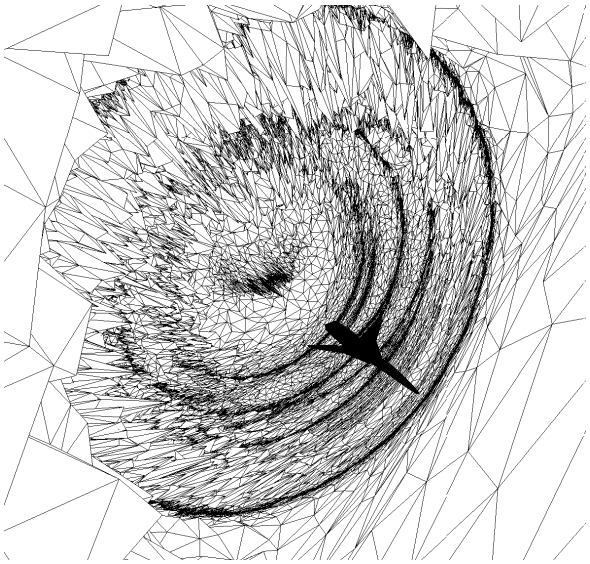
The research issues we are facing are inherent to the complexity of the flows considered by the scientific community, and in particular to the willingness to capture interactions between different physical phenomena. Figure 5a is a recent Schlieren photograph of an aircraft flying at supersonic speed and is a

(a) Mach cone around a F-15. In red: the directions of anisotropy of the shock wave.



(b) Some hybrid procedures consist in coupling different methods for each region.





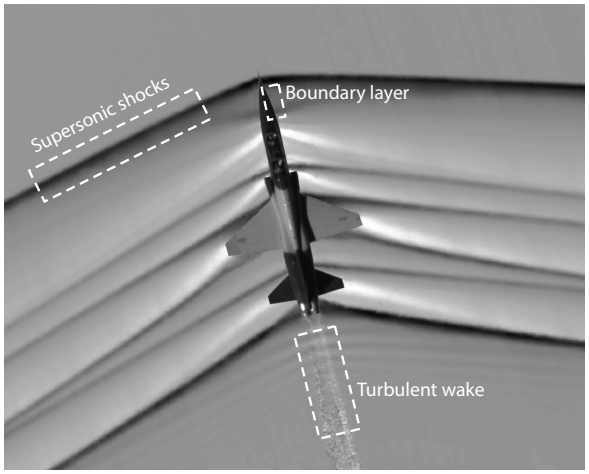
(c) Mesh adaptation result: solution (left) and adapted mesh (right).

Figure 4: **A sonic boom** is an acoustic phenomenon caused by a body moving in athmosphere at a speed exceeding the local speed of sound. As an aircraft exceeds the speed of sound, shock waves are created at its surface and emanate forward, forming Mach cones. These shock waves propagate through the atmosphere notably toward the ground, resulting in an abrupt pressure increase in the ambiant air and causing the typical "boom-boom" heard on the ground. This noise has a major impact on the environment as it impacts a band 60 to 80 km wide, causing annoyance for the population as well as building vibrations. Therefore, there is nowadays substantial economic interest in designing low sonic boom supersonic aircraft and CFD plays an important part in the design process. Mesh adaptation has proved to be a powerful tool for this example [7].

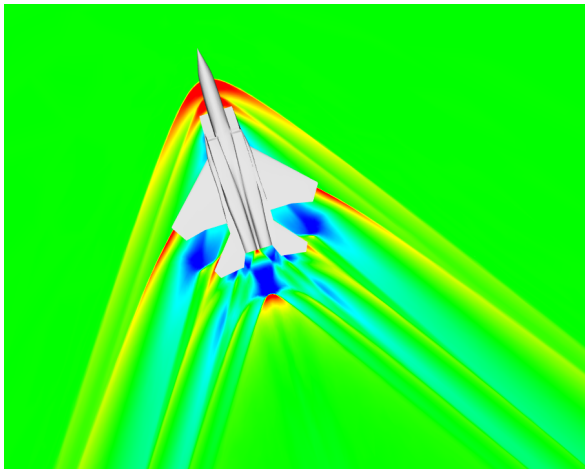
good illustration of the three different phenomena we are particularly interested in: it contains (i) shock waves that propagate through the farfield region, (ii) a turbulent wake, and (iii) viscous boundary layers i.e. strong normal gradient variation in the immediate vicinity of the surface of the aircraft.

As previously mentioned, the current state-of-the-art makes it possible to successfully apply mesh adaptation to inviscid phenomena but many research issues remain for turbulent viscous phenomena modeled by RANS equations. We list some of them below.

- **Solver and meshing software:** there is a need for simultaneous improvements in both the flow



(a) A recent schlieren photograph of a T-38C flying at Mach 1.09 (Credit: NASA).



(b) Result of an **inviscid** mesh adaptation of a F15 flying at Mach 1.8 (in-house simulation).

Figure 5: Illustration of the physical phenomena of interest.

solver and the mesh adaptation strategy.

- **Boundary layer mesh adaptivity:** an accurate prediction of viscous phenomena in near-wall regions (boundary layers) is crucial for the accuracy of the whole solution. These boundary layers require quasi-structured meshes, for which only little work exists in an adaptative context. In 3D, the boundary layer mesh is usually kept unchanged, and fully turbulent adaptive RANS simulations are only carried out in 2D.

- **Computational time and robustness:** the cost (in terms of CPU time) of the flow solver increases greatly for RANS simulations as they require larger meshes. The complexity of the geometry also tends to decrease the robustness (i.e. the probability that it provides an accurate solution) of the flow solver.

Main contributions

The main contributions of this thesis are focused on the development of the flow solver along with the improvement of the mesh generation software.

The numerical flow solver is an essential component of the mesh adaptation procedure, during which a solution is computed at each iteration of the adaptive loop. If one of these solutions is not accurate enough, the next adaptive iterations might certainly be spoiled, leading to a wrong final result. Moreover, the convergence speed (in terms of CPU) of the solver dramatically impacts the total wall clock time of the mesh adaptation process. Keeping in mind that the objective of this PhD is to develop adaptive strategies for RANS simulations, we have implemented the RANS version of our in-house flow solver `Wolf` and we have put a lot of effort into improving the robustness and the convergence speed.

Turbulence modeling. At the beginning of this PhD, only the explicit Euler version of the flow solver had been implemented. We implemented the Spalart-Allmaras turbulence model and adequate boundary conditions. We present the comprehensive Verification & Validation (V&V) study we carried out, which includes comparisons with other well-established CFD flow solvers and experimental data, for a comprehensive set of 2D and 3D test cases.

Improving the convergence and the robustness of the flow solver. In the context of RANS simulations, meshes are larger and flows are more complex, which is why 3D simulations cannot be envisaged without accelerating convergence and improving robustness. To this end, we made the following improvements.

- We have implemented an implicit time integration, which presents a much faster convergence rate compared to an explicit time integration. Using the implicit approach, a linear system is solved at each flow solver iteration and the method used to solve this system is crucial for the global convergence of the simulations in terms of both wall clock time and the accuracy of the solution.
- We have implemented an implicit multigrid method in order to accelerate and improve the convergence of the solving of this linear system.
- Appropriate CFL laws are mandatory to achieve fast convergence in solving non-linear equations, but are too dependent on parameters set by the user. To avoid this issue, we implemented a local (i.e. a CFL value for each vertex) dynamic CFL law. CFL values are set automatically, depending on the evolution of the solution.
- All the new routines were parallelized using a shared-memory approach based on pthreads, using an in-house library that automatically deals with indirect addressing.

As regards mesh adaptation, we list below the contributions to two aforementioned issues: convergence/robustness and boundary layer adaptivity.

Improving the convergence and the robustness of mesh adaptation. We addressed this issue in an adaptive context, by benefiting from multigrid properties in the mesh adaptation process, and developing a distributed parallel mesh generation algorithm.

- **Multigrid methods coupled with mesh adaptation:** we extended the aforementioned multigrid method to an adaptive context, which consists in recycling the adapted meshes generated during the adaptive process to run multigrid flow computations. In this context, interesting convergence properties arising from the multigrid theory make it possible to improve robustness and avoid wasted computational effort.

- **Adaptive parallel mesh generation:** we devised a distributed parallel mesh generation algorithm for small scale parallel architectures (less than 1000 cores) such as are typically found in most R&D units. We were able to generate an anisotropic adapted mesh containing around one billion elements in less than 20 minutes on 120 cores.

Boundary layer adaptivity. We introduced a procedure to automatically generate anisotropic adapted quasi-structured meshes of high-quality. It consists in taking into account the natural alignment and orthogonality of the provided input metric field during the mesh generation process.

Scientific communications

Peer-reviewed proceedings

- **Fast generation of large-size adapted meshes** (best paper award), A. Loseille, V. Menier, F. Alauzet, *24th International Meshing Roundtable*, Austin, TX, USA, October 2015.
- **Multigrid Methods Coupled with Anisotropic Mesh Adaptation**, V. Menier, A. Loseille, F. Alauzet, *The American Institute of Aeronautics and Astronautics (AIAA) SciTech 2015*, Kissimee, FL, USA, January 2015 [124] (DOI: http://dx.doi.org/10.2514/6.2015-2041).
- **3D Parallel Unsteady Mesh Adaptation for the High-Fidelity Prediction of Bubble Motion**, V. Menier, *ESAIM Proceedings*, 2015 [122] (DOI: http://dx.doi.org/10.1051/proc/201550009).
- **CFD Validation and Adaptivity for Viscous Flow Simulations**, V. Menier, A. Loseille, F. Alauzet, *The American Institute of Aeronautics and Astronautics (AIAA) Aviation*, Atlanta, GA, USA, June 2014. [123] (DOI: http://dx.doi.org/10.2514/6.2014-2925).
- **Serial and Parallel Mesh Modification Through a Unique Cavity-Based Primitive**, A. Loseille and V. Menier, *22nd International Meshing Roundtable*, Orlando, FL, USA, October 2013 [108] (DOI: http://dx.doi.org/10.1007/978-3-319-02335-9_30).

Communications

- **Méthodes Multigrilles Couplées à l'Adaptation de Maillage**, V. Menier, *SMAI 2015*, Les Karellis, France, June 2015.
- **Mesh Adaptation for Viscous Simulations**, V. Menier, A. Loseille and F. Alauzet, *ECCOMAS 2014*, Barcelona, Spain, July 2014.
- **(Poster) Adaptation de Maillage en Parallèle** V. Menier, *CANUM 2014*, Carry-Le-Rouët, France, April 2014.

- **(Poster) Parallel Mesh Adaptation**, V. Menier and A. Loseille, *International Meshing Roundtable*, Orlando, FL, USA, October 2013.

Awards

- 2nd Best Poster Award, *CANUM 2014*, Carry-Le-Rouët, France, April 2014.
- Best Student Poster Award and Best Technical Poster Award, *International Meshing Roundtable*, Orlando, FL, USA, October 2013.

Outline

This work is composed of three parts.

Part I deals with metric-based mesh adaptation. In Chapter 1, twenty years of active research is reviewed, useful mathematic and geometric notions are introduced, and our in-house anisotropic mesh generator `AMG` is described. Then, in Chapter 2, we present how we address the fast generation of very large adapted meshes. To this end, we devised a distributed coarse-grained parallel mesh adaptation procedure.

Part II focuses on the contribution to our in-house flow solver `Wolf`, including the implementation of turbulence modeling and a multigrid acceleration procedure. In Chapter 3, we detail the numerical modeling and the parallelization methodology chosen. In Chapter 4 we present the rigorous verification and validation (V&V) study we carried out. The multigrid acceleration procedure is described in Chapter 5.

In Part III, we first present the coupling of multigrid methods with mesh adaptation in Chapter 6. Then, the remaining research issues for adaptive RANS simulations are set out and attempts to address them are presented from the point of view of mesh generation in Chapter 7.

Contents

Part I

Anisotropic Mesh Adaptation

Chapter 1

A review of feature-based anisotropic mesh adaptation for inviscid flows

Contents

1.1 Introduction

The purpose of this chapter is to review research activities in the field of anisotropic mesh adaptation. In particular, we focus on the research issues that have been addressed since the beginning of the 2000s for inviscid flows.

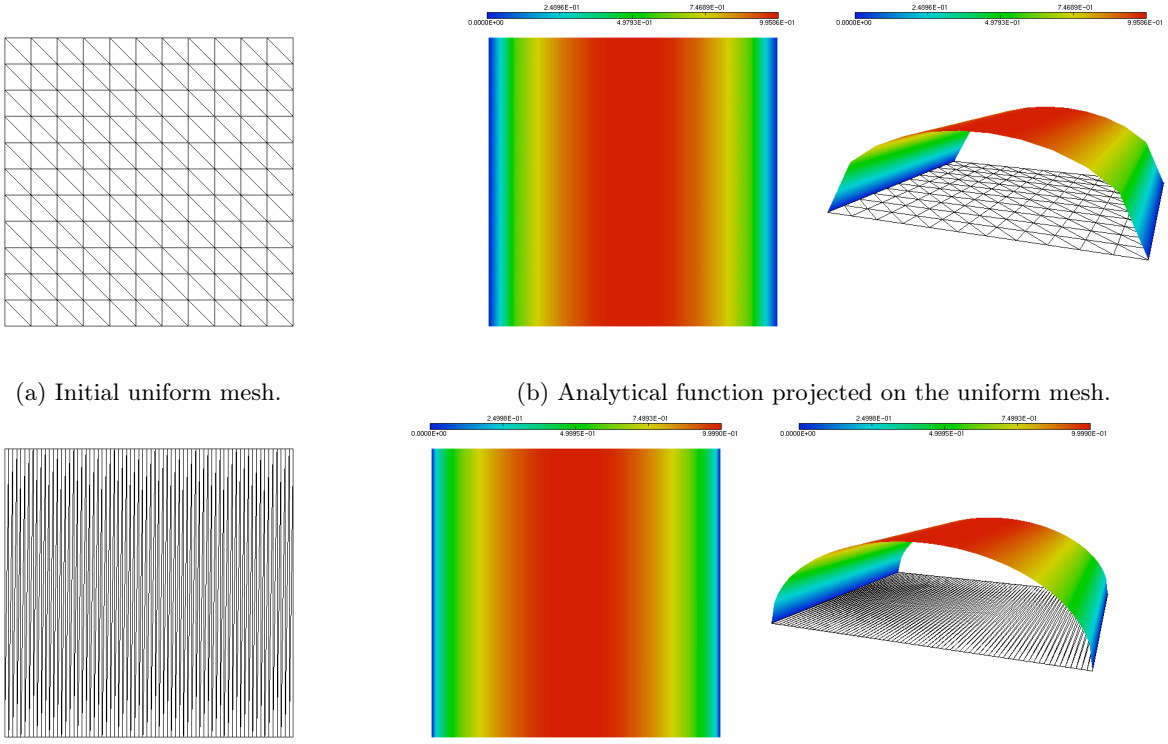
As explained in the introduction, a mesh is a discrete support for the considered numerical methods. Mesh generation is thus an essential part of the computational pipeline: no mesh, no simulation. Moreover, the mesh greatly impacts the efficiency, the stability and the accuracy of numerical methods. One goal of anisotropic mesh adaptation is to generate a mesh that fits the physics and (if possible) the numerical scheme in order to compute the best possible solution at the cheapest computational cost.

The general idea of anisotropic mesh adaptation is to modify the discretization of the computational domain in order to minimize errors induced by the discretization. Some mesh regions are refined, while other regions are coarsened, and stretched mesh elements are generated to follow the natural anisotropy (i.e. when the variation of the solution is directionally dependent) of the physical phenomena. We generally distinguish three kinds of errors: (i) the interpolation error ($\mathbf{u} - \Pi_h \mathbf{u}$), (ii) the implicit error ($\Pi_h \mathbf{u} - \mathbf{u}_h$) and (iii) the approximation error ($\mathbf{u} - \mathbf{u}_h$), where $\mathbf{u}$ is the exact solution, $\mathbf{u}_h$ is the numerical solution provided by the flow solver and Π_h is the linear interpolate of $\mathbf{u}$ on the discretization. In the sequel, we illustrate the main principle of mesh adaptation through the simple example of the control of the interpolation error of an analytical function.

1.1.1 A simple 2D example

Figure 1.1 is a simple 2D analytical example that illustrates how mesh adaptation can reduce the interpolation error. An analytical function f is considered, that presents variation in the $\mathbf{x}$ direction but is constant along the $\mathbf{y}$ direction. The discretization is modified manually in order to take into account the anisotropy of f and thus to improve its representation, i.e. to reduce the interpolation error such as illustrated in Figure 1.2.

We consider two different meshes which both contain 144 vertices: an initial -uniformely sized- discretization (Figure 1.1a) and a manually modified mesh (Figure 1.1c). Obviously, the initial mesh is not optimal, as half its vertices were inserted along the $\mathbf{y}$ direction and thus do not improve the representation of f. In the adapted mesh, however, all the vertices were inserted along the $\mathbf{x}$ direction, which leads to more precision. As a consequence, stretched elements were created along the direction of anisotropy ($\mathbf{y}$). The difference between the two approximations is highlighted in Figures 1.1b and 1.1d. Table 1.1 shows how drastically interpolation errors (in L^1, L^2 and L^∞ norms) are reduced by almost one order of



(a) Initial uniform mesh.

(b) Analytical function projected on the uniform mesh.

(c) Taylored mesh.

(d) Analytical function projected on the taylored mesh.

Figure 1.1: Simple analytical example that illustrates the minimization of the interpolation error using manual mesh modification.

magnitude using the modified mesh compared to the uniform one.

The underlying concepts illustrated by this simple example can be naturally extended to the field of Computational Fluid Dynamics (CFD). Here the discretization was modified manually, which is impossible when considering real life CFD applications. Mesh adaptation automatically performs this modification process, which requires (i) to be able to communicate with an automatic mesh generator and (ii) to measure and quantify mesh size and anisotropy. Details on the adaptive process are provided in this chapter, but first we explain why anisotropic mesh adaptation can have a significant impact for CFD applications.

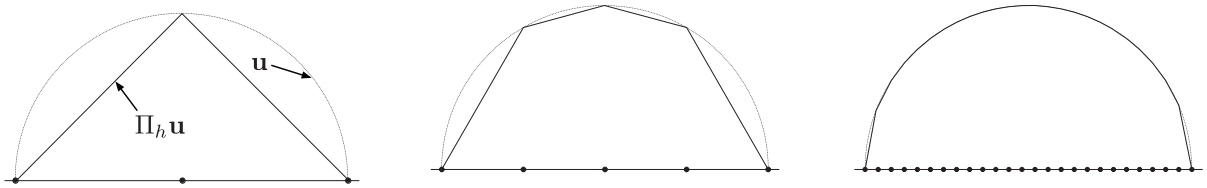


Figure 1.2: Geometrical illustration of the error of interpolation $\mathbf{u} - \Pi_h\mathbf{u}$. The more vertices are used, the better the representation of the circle (dashed line) is.

Mesh	L^1	L^2	L^∞
Uniform	0.029	0.059	0.133
Taylored	0.008	0.005	0.014

Table 1.1: Simple 2D example: interpolation errors on both meshes.

1.1.2 Anisotropic mesh adaptation in CFD

The use of anisotropic mesh adaptation in CFD is motivated by the nature of the flows. In many real-life applications, important physical phenomena take place in **small regions** of the computational domain. Important means that if they are not accurately captured, the accuracy of the whole solution -including larger scale phenomena- may be badly impacted (boundary layers are an obvious example). As this solution is unknown *a priori*, it is impossible to generate a manually taylored mesh that fits the underlying physics. Moreover, some physical phenomena are **anisotropic**. Therefore, uniform meshes are not optimal as all the vertices inserted in the direction of anisotropy may not improve the accuracy in any way, although they may badly impact the CPU time. Thus, it seems natural to take into account the underlying physics during the simulation, in order to improve the tradeoff between accuracy of the solution and computational time.

Mathematically speaking, mesh adaptation aims at generating an optimal mesh to control the accuracy of the numerical solution. Optimal means that the best possible accuracy is achieved for a given mesh size, or equivalently, a mesh of minimal size is generated to reach a given accuracy. Thus, it enables substantial gains in CPU time, memory requirement and storage space. Furthermore, error estimates have the ability to detect physical phenomena and capture their behavior. Meshes are thus automatically adapted in critical regions without any *a priori* knowledge of the problem.

A lot of work has been achieved by the scientific community to be able to apply mesh adaptation to real-life inviscid simulations such as the prediction of the sonic boom. We now provide a short history of mesh adaptation, starting from its early days at the end of the 80s.

1.1.3 1980-2000: a short history of mesh adaptation

Even though the basic idea of splitting edges to fit the numerical solution appeared in the 60s, the proper emergence of the concept of mesh adaptation dates back from the end of the 80s, when Peraire et al. introduced error measures involving directions in 2D [141]. They studied the directional properties of the interpolation error and initiated the idea of generating stretched mesh elements. These slightly anisotropic elements were generated using and advancing front method and had an aspect ratio of approximately $1:5$. Similar approaches have been considered by Selmin and Formaggia [151]. Attempts to extend this idea to three dimensions were published in the early 90's by Löhner [90] and Peraire [140] but results

were almost isotropic.

Almost at the same time, Mavriplis proposed generating stretched elements in two dimensions using a Delaunay approach based on a locally stretched space which was close to the idea of a metric [117]. A year later, George et al. introduced the use of a metric in a Delaunay mesh generator [62]. They noticed that the absolute value of the Hessian of a scalar solution is a metric, and proposed a Delaunay-based approach where edge lengths were computed in the Riemannian metric space. This idea generalized all the previous work.

The idea of metric has then been widely used for 2D anisotropic mesh adaptation since the 90s, see for example the following work [56, 35, 71, 50, 30]. In 1997, Baker presented a state-of-the art and wrote [10]: *"Mesh generation in three dimensions is a difficult enough task in the absence of mesh adaptation and it is only recently that satisfactory three-dimensional mesh generators have become available. [...] . Mesh alteration in three dimensions is therefore a rather perilous procedure that should be under taken with care".*

At the dawn of the 21st century, robust 3D isotropic mesh generators had been developed and the problem of controlling the error of interpolation was well known. However, many research issues remained to be faced to apply mesh adaptation to real-life CFD applications.

1.1.4 Research issues faced at the beginning of the 2000s

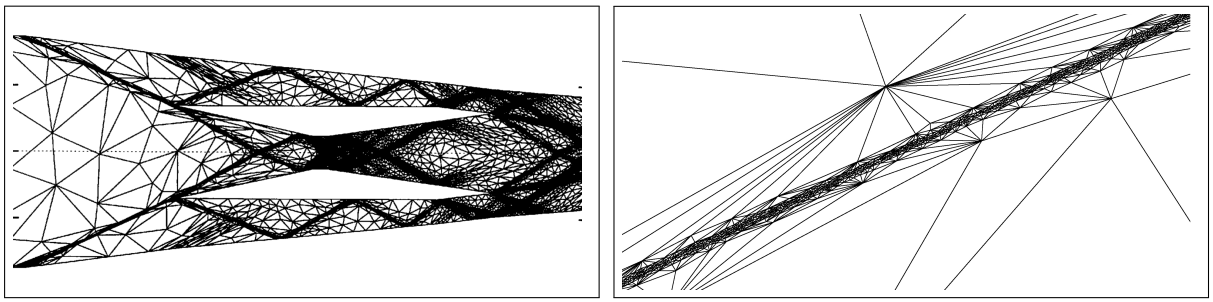
We list below the main research issues that remained to be addressed in order to apply anisotropic mesh adaptation to real-life simulations. These research issues included the loss of anisotropy **(Issue 1)**, the inability to capture all scales of the physics **(Issue 2)** , and the lack of robustness induced by anisotropic meshes **(Issue 3)**.

(Issue 1) Toward the generation of anisotropic meshes

It has been shown that, in the adaptive process, both the error estimate and the numerical scheme were the cause of a loss of anisotropy [35]. Figure 1.3 presents an adapted mesh from a simulation such as was performed at the end of the 90s. This example considers an internal supersonic flow at Mach 3 in a scramjet inlet which was published in 1997 by Castro-Díaz et al. [35]. The error estimate used for this simulation is based on the control of the L^∞ norm of the interpolation error of the local Mach number. The adapted mesh presented seems to present a fair refinement of the shock regions along with anisotropic elements following the shock directions. But if we take a closer look at the shock region, it appears that the mesh elements that seemed anisotropic are in fact isotropic, or at least only slightly stretched. There were two main reasons for this loss of anisotropy:

- In the normal direction to the shock, the size prescribed by the error estimate is much smaller than the smallest size that the remesher can possibly generate.

- In the tangential direction to the shock, the numerical solution presents local oscillations because of the flow solver that does not strictly respect the TVD property. These oscillations are captured by the error estimate (which uses the Hessian as the sensor).



(a) Large view of the final adapted mesh. (b) Close view of the shock region.

Figure 1.3: Anisotropic mesh adaptation of a supersonic scramjet published in 1997 by Castro-Díaz et al. [35].

(Issue 2) Capturing all scales of the physics

Using the L^∞ norm for measuring errors was a major limitation as it ignores the smallest scales of the solution and focuses only on high gradients (shocks, discontinuities, etc.). And yet the flows studied by the CFD community are multiscale, see for example the case of a transonic business jet in Figure 1.4: shock waves must be captured along with small vortices at the extremity of the wings, although these phenomena have very different magnitudes (Figure 1.4b). This research issue was addressed thanks to the recent advances in mesh adaptation, notably the use of the L^p norm. Indeed when using the L^∞ norm, a major constraint is the obligation to prescribe a minimal edge size. To remove this constraint, it became necessary to use a norm that is less sensitive to stiff gradients, like the L^p. Using the L^p norm induces an automatic normalization of the solution field, thus allowing all scales to be captured.

Significant work to propose new more accurate anisotropic error estimates includes the following: a posteriori estimates [142, 54, 20], a priori estimates [55, 75, 98], and goal-oriented estimates for scalar functional outputs [164, 80, 102].

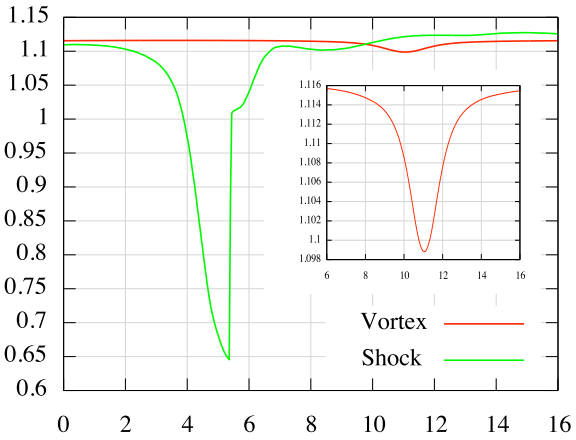
(Issue 3) Robustness

Strong mesh anisotropy caused serious robustness issues for flow solvers and for (re)meshing algorithms.

The behaviour of a flow solver depends greatly on the discretization, and computing on highly stretched mesh elements was -and sometimes remains- challenging.

From the point of view of mesh generation, generating a volume mesh starting from a given anisotropic surface was also a major issue, as most algorithms simply failed during the boundary recovery phase. This difficulty has since been partly solved using local remeshing approaches to adapt the mesh. The idea is to start from an existing mesh and to perform local modifications (such as edge collapses, swaps,

(a) Vorticity in the wake region (*Photo courtesy of Cessna Aircraft Company*).



(b) Shock waves and vortices do not have the same magnitude, but must be captured altogether.

Figure 1.4: Example of a multiscale simulation: a transonic business jet [96].

point insertions/deletions, etc.) iteratively to adapt the mesh, while keeping a valid surface and volume mesh during the whole process. See, for instance, the following work [160, 135, 18, 16, 65, 88].

These robustness considerations are even more crucial in an adaptive context, as it is an iterative process during which several flow solver computations and several remeshing steps are performed. If one stage of this process fails, then the whole simulation collapses.

In the sequel, we present the recent developments in mesh adaptation to address the aforementioned research issues and illustrate them on a 3D simulation.

1.2 Metric definition

To generate anisotropic meshes, one must be able to prescribe at each point of the domain the desired sizes and directions of the final mesh elements. To this end, Riemannian metric spaces are used. The main idea of metric-based mesh adaptation is to generate a so-called *unit mesh* according to a Riemannian metric space, i.e. a mesh whose edges have a size equal to one according to this metric space and whose elements have a unit volume equal to $\sqrt{2}/12$ (in 3D).

In this section, we recall the necessary differential geometry notions. We use the following notations: bold face symbols, such as $\mathbf{a}, \mathbf{b}, \mathbf{u}, \mathbf{v}, \mathbf{x}, \ldots$, denote vectors or points of $\mathbb{R}^3$. Vector coordinates are denoted by $\mathbf{x} = (x_1, x_2, x_3)$. The natural dot product between two vectors $\mathbf{u}$ and $\mathbf{v}$ of $\mathbb{R}^3$ is: $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{3} u_i \, v_i$.

1.2.1 Euclidean metric space

An **Euclidean metric space** $(\mathbb{R}^3, \mathcal{M})$ is a vector space of finite dimension where the dot product is defined by means of a 3×3 symmetric definite positive tensor $\mathcal{M}$:

$$\langle \mathbf{u}, \mathbf{v} \rangle_{\mathcal{M}} = \langle \mathbf{u}, \mathcal{M}\mathbf{v} \rangle = {}^t\mathbf{u}\mathcal{M}\mathbf{v}\,, \quad \text{for } (\mathbf{u}, \mathbf{v}) \in \mathbb{R}^3 \times \mathbb{R}^3\,.$$

In the following, the matrix $\mathcal{M}$ is simply called **a metric tensor** or **a metric**.

The dot product defined by $\mathcal{M}$ makes $\mathbb{R}^3$ become a normed vector space $(\mathbb{R}^3, \|.\|_{\mathcal{M}})$ and a metric vector space $(\mathbb{R}^3, d_{\mathcal{M}}(.,.))$ supplied by the following **norm** and **distance** definitions:

$$\forall \mathbf{u} \in \mathbb{R}^3\,,\ \|\mathbf{u}\|_{\mathcal{M}} = \sqrt{\langle \mathbf{u}, \mathcal{M}\mathbf{u} \rangle} \quad \text{and} \quad \forall (\mathbf{a}, \mathbf{b}) \in \mathbb{R}^3 \times \mathbb{R}^3\,,\ d_{\mathcal{M}}(\mathbf{a}, \mathbf{b}) = \|\mathbf{ab}\|_{\mathcal{M}}.$$

In these spaces, the **length** $\ell_{\mathcal{M}}$ of a segment $\mathbf{ab}$ is given by the distance between its extremities:

$$\ell_{\mathcal{M}}(\mathbf{ab}) = d_{\mathcal{M}}(\mathbf{a}, \mathbf{b}) = \|\mathbf{ab}\|_{\mathcal{M}}. \tag{1.1}$$

We are also able to compute the cross product with respect to metric tensor $\mathcal{M}$. In a Euclidean metric space, volumes and angles are still well defined [114]. These features are of main interest when dealing with meshing. For instance, given a bounded subset K of $\mathbb{R}^3$, the **volume** of K computed with respect to metric tensor $\mathcal{M}$ is:

$$|K|_{\mathcal{M}} = \sqrt{\det \mathcal{M}}\, |K|_{\mathcal{I}_3}\,, \tag{1.2}$$

where $|K|_{\mathcal{I}_3}$ is the Euclidean volume of K. Finally, as metric tensor $\mathcal{M}$ is symmetric, it is diagonalizable in an orthonormal basis:

$$\mathcal{M} = \mathcal{R}\,\Lambda\,{}^t\mathcal{R}\,,$$

where $\mathcal{R}$ is an orthonormal matrix composed of the **eigenvectors** $(\mathbf{v}_i)_{i=1,3}$ of $\mathcal{M}$ verifying ${}^t\mathcal{R}\mathcal{R} = \mathcal{R}{}^t\mathcal{R} = \mathcal{I}_3$. $\Lambda = \mathrm{diag}(\lambda_i)$ is a diagonal matrix composed of the **eigenvalues** of $\mathcal{M}$, denoted $(\lambda_i)_{i=1,3}$, which are strictly positive.

Geometric interpretation. At each point of the domain, we represent the corresponding metric tensor by an ellipsoid defined by the set of points at distance 1 (in the metric space) from that point. In the vicinity $\mathcal{V}(\mathbf{a})$ of point $\mathbf{a}$, the unit ball $\Phi_{\mathcal{M}}(1)$ of $\mathcal{M}$ is defined by:

$$\Phi_{\mathcal{M}}(1) = \left\{ \mathbf{x} \in \mathcal{V}(\mathbf{a}) \mid {}^t(\mathbf{x} - \mathbf{a})\,\mathcal{M}\,(\mathbf{x} - \mathbf{a}) = 1 \right\}.$$

The above relation defines an ellipsoid denoted by $\mathcal{B}_{\mathcal{M}}$ centered at $\mathbf{a}$ with its axes aligned with the eigen directions of $\mathcal{M}$. Sizes along these directions are given by $h_i = \lambda_i^{-\frac{1}{2}}$. This ellipsoid depicted in

Figure 1.5.

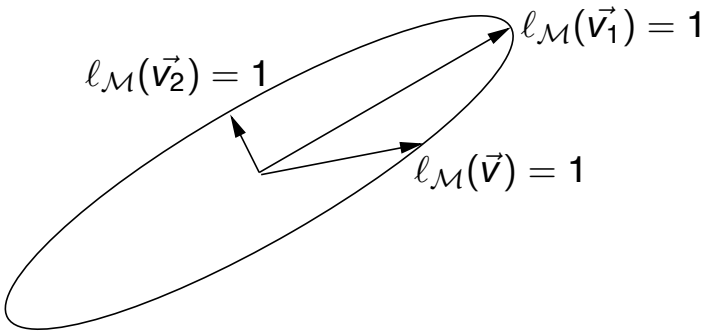


Figure 1.5: Geometric interpretation of the unit ball $\mathcal{B}_{\mathcal{M}}$. $\mathbf{v}_i$ are the eigenvectors of $\mathcal{M}$

1.2.2 Riemannian metric space

In the context of mesh adaptation, we use a **Riemannian metric space** defined by $\mathbf{M} = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$. In this specific case, we only know $\mathcal{M}$ a Riemannian metric and $\Omega \subset \mathbb{R}^3$ a common space of parametrization which is our computational domain. There is no global notion of scalar product. The main interest is that we can extend the notions of length and volume as in Euclidean metric spaces to Riemannian metric spaces which will be used by the mesher in an anisotropic adaptive context.

A mesh generator requires the computation of an edge length that takes into account the variation of the metric along the edge. Using the parametrization $\gamma(t) = \mathbf{a} + t\,\mathbf{ab}$, where $t \in [0, 1]$, the length of an edge $\mathbf{ab}$ according to $\mathcal{M}$ is:

$$\ell_{\mathcal{M}}(\mathbf{ab}) = \int_0^1 \|\gamma'(t)\|_{\mathcal{M}}\, \mathrm{d}t = \int_0^1 \sqrt{{}^t\mathbf{ab}\, \mathcal{M}(\mathbf{a} + t\,\mathbf{ab})\, \mathbf{ab}}\, \mathrm{d}t. \tag{1.3}$$

Figure 1.6 depicts iso-values of segment length from the origin for different Riemannian metric spaces. The iso-values are isotropic for the Euclidean space. They are anisotropic in the case of a Euclidean metric space defined by $\mathbf{M} = \mathcal{M}$. The two principal directions of $\mathcal{M}$ clearly appear. In the case of a Riemannian metric space $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$, all previous symmetries are lost.

The notion of volume is also extended to Riemannian metric spaces. Given a bounded subset K of Ω, the **volume** of K computed with respect to $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ is:

$$|K|_{\mathcal{M}} = \int_K \sqrt{\det \mathcal{M}(\mathbf{x})}\, \mathrm{d}\mathbf{x}\,. \tag{1.4}$$

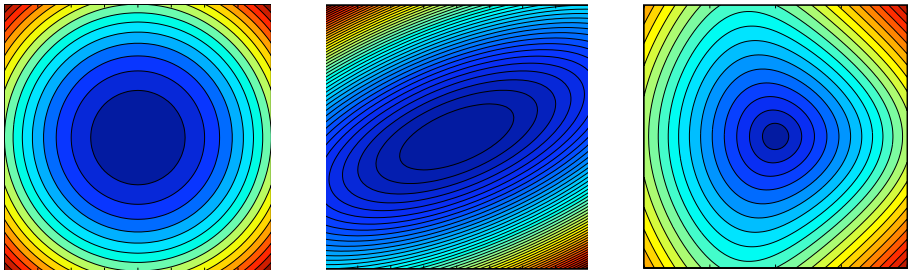


Figure 1.6: Iso-values of the function $f(\mathbf{x}) = \ell_{\mathcal{M}}(\mathbf{ox})$ where $\mathbf{o}$ is the origin for different Riemannian metric spaces. Left, canonical Euclidean space $(\Omega, \mathcal{I}_2)$, middle, Euclidean metric space $(\Omega, \mathcal{M})$ with $\mathcal{M}$ constant and, right, Riemannian metric space $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ with a varying metric tensor field.

1.3 Metric-based mesh generation

This section describes `AMG`, our in-house adaptive mesh generator. In the previous section, Riemannian metric spaces are used to prescribe sizes and orientation at each point of the domain. In this section, we explain how this information is used to generate a mesh that meets these length and orientation requirements. The general idea of metric-based mesh generation is to generate a **unit mesh** in the prescribed Riemannian metric space. Note that many metric-based remeshers exist, see for example [42, 88, 49, 126]. For a more complete description of `AMG`, we refer to [105, 108].

We first recall the central notion of unit mesh and unit element, then we give an overview of the mesh modification operations we perform in order to generate such a unit mesh according to the prescribed metric. All these local mesh modifications are embedded in a single cavity-based mesh operator and formalism.

1.3.1 Unit elements and unit meshes

A tetrahedron K, defined by its list of edges $(\mathbf{e_i})_{i=1..6}$, is **unit** with respect to a metric tensor $\mathcal{M}$ if the lengths of all its edges are unit in metric $\mathcal{M}$:

$$\forall i = 1, ..., 6, \quad \ell_{\mathcal{M}}(\mathbf{e_i}) = 1 \text{ with } \ell_{\mathcal{M}}(\mathbf{e_i}) = \sqrt{{}^t\mathbf{e_i}\,\mathcal{M}\,\mathbf{e_i}}.$$

If K is composed only of unit length edges, then its volume $|K|_{\mathcal{M}}$ in $\mathcal{M}$ is constant equal to:

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

where $|K|$ is its Euclidean volume.

A discrete mesh $\mathcal{H}$ of a domain $\Omega \subset \mathbb{R}^3$ is a **unit mesh** with respect to Riemannian metric space

$(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ if all its elements are quasi-unit. The definition of unity is thus relaxed by taking into account technical constraints imposed by mesh generators. To avoid cycling while analyzing edges lengths, a tetrahedron K defined by its list of edges $(\mathbf{e_i})_{i=1\dots6}$ is said to be quasi-unit if, $\forall i,\ \ell_{\mathcal{M}}(\mathbf{e_i}) \in [\frac{1}{\sqrt{2}}, \sqrt{2}]$, see [61]. The study in [99] shows that several non-regular space filling tetrahedra verify this constraint, which guarantees the existence for constant Riemannian metric space. Unfortunately, this weaker constraint on edges lengths can lead to the generation of quasi-unit elements with a null volume, see [99]. Consequently, controlling only the length of the edges is not sufficient, the volume must also be controlled to relax the notion of unit element. In practice, these two quantities are combined into a quality function:

$$Q_{\mathcal{M}}(K) = \frac{36}{3^{\frac{1}{3}}} \frac{|K|_{\mathcal{M}}^{\frac{2}{3}}}{\sum_{i=1}^{6} \ell_{\mathcal{M}}^2(\mathbf{e_i})} \in [0,1]\,. \tag{1.5}$$

For the perfect regular tetrahedron, whatever the length of its edges, the quality function is equal to 1. For a null volume tetrahedron, $Q_{\mathcal{M}}$ is 0. We deduce the following definition of quasi-unit element, used by mesh generators. A tetrahedron K defined by its list of edges $(\mathbf{e_i})_{i=1\dots6}$ is said to be **quasi-unit** for Riemannian metric space $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ if

$$\forall i \in [1,6], \quad \ell_{\mathcal{M}}(\mathbf{e_i}) \in \left[\frac{1}{\sqrt{2}}, \sqrt{2}\right] \quad \text{and} \quad Q_{\mathcal{M}}(K) \in [\alpha, 1] \ \text{ with } \ \alpha > 0\,, \tag{1.6}$$

where Relations (1.3) and (1.4) are used to evaluate lengths and volumes, respectively. We usually take $\alpha = 0.8$.

1.3.2 Cavity-based operators

A complete mesh generation or mesh adaptation process usually requires a large number of operators: Delaunay insertion, edge-face-element point insertion, edge collapse, point smoothing, face/edge swaps, etc. Independently of the complexity of the geometry, the more 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. In [108], a unique cavity-based operator has been introduced which embeds all the aforementioned operators. This unique operator is used at each step of the process for surface and volume remeshing.

The cavity-based operator is inspired by incremental Delaunay methods [21, 167, 72] where the current mesh $\mathcal{H}_k$ is modified iteratively through sequences of insertion of a point P:

$$\mathcal{H}_{k+1} = \mathcal{H}_k - \mathcal{C}_P + \mathcal{B}_P, \tag{1.7}$$

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 circumcircle 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 $\mathcal{C}_P$. This insertion pattern in

two dimensions is illustrated in Figure 1.7.

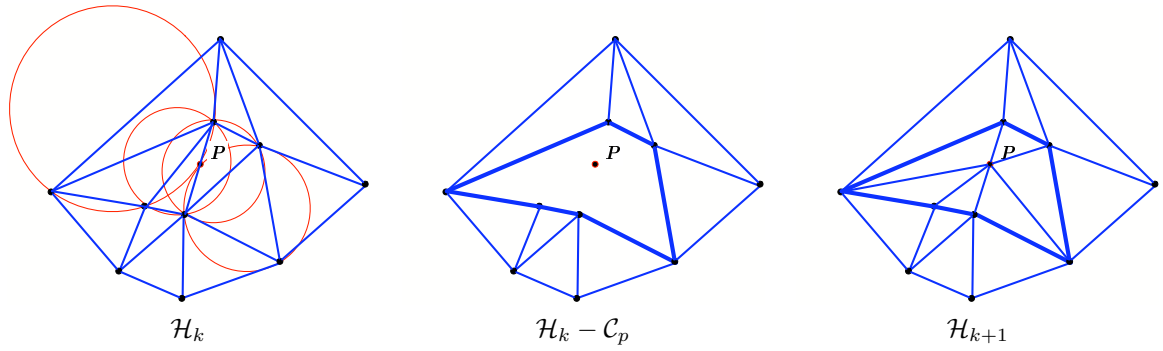


Figure 1.7: Illustration of the 2D incremental Delaunay point insertion given by Relation (1.7).

In the cavity-based framework [108], each mesh modification operator is equivalent to a node (re)insertion inside a cavity. 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 $\mathcal{C}_p$:

$$\mathcal{H}_{k+1} = \mathcal{H}_k - \mathcal{C}_p + \mathcal{R}_P \,, \tag{1.8}$$

where $\mathcal{R}_P$ is the set of elements created in the cavity. Note that 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 a tetrahedron) and $\mathcal{R}_P$ generates only valid elements. Figure 1.8 presents the reinterpretation of three meshing operators with the cavity-based operator.

The use of such local mesh operators addresses the aforementioned robustness issue **(Issue 3)** of the remeshing. In particular, the boundary recovery is only treated during the initial mesh generation, and then a valid surface and volume mesh is kept during the whole process.

1.4 Continuous mesh framework

The previous section emphasized the role of metric tensors and Riemannian metric spaces as useful mathematical tools to prescribe sizes and directions to the remesher. Here, we introduce the concept of a continuous mesh (see [99, 100]), which establishes a duality between the discrete domain and the continuous domain, based on Riemannian metric spaces. In other words, discrete meshes are represented by Riemannian metric spaces, for which powerful mathematical tools are available. Thus, mathematical problems which could not even be considered on discrete meshes can be addressed in this framework. This is particulary useful to derive error estimates and to design suitable metric tensor fields from these error estimates.

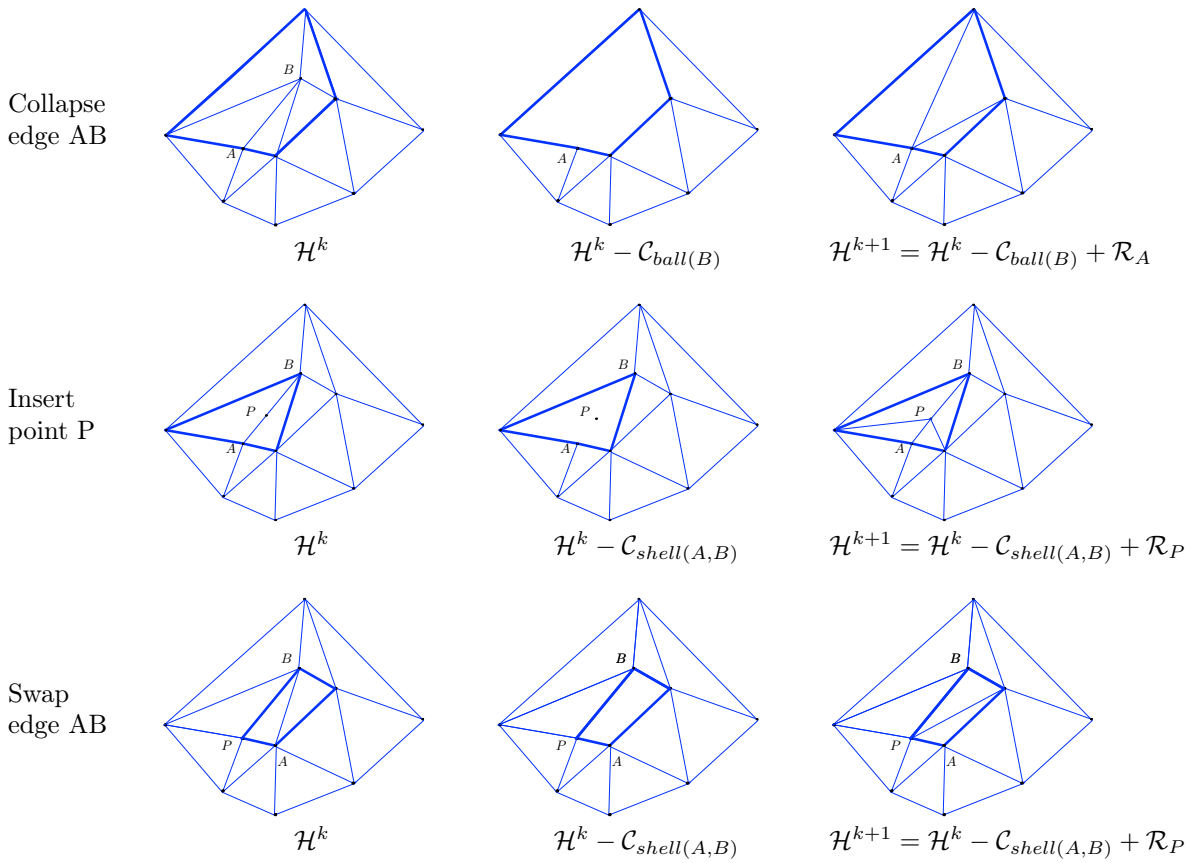


Figure 1.8: Some 2D meshing operators reinterpreted as a cavity-based operator with an appropriate choice of the point to (re)insert and cavity to remesh. From top to bottom, the collapse, insertion and swap operators.

1.4.1 Duality between discrete and continuous entities

The following points out the strong duality between discrete entities, e.g. elements and meshes, and continuous mathematical objects, e.g. metric tensors and Riemannian metric spaces.

Let $\mathcal{M}$ be a metric tensor, there exists a non-empty infinite set of unit elements with respect to $\mathcal{M}$. Conversely, given an element K such that $|K| \neq 0$, there is a **unique metric tensor** $\mathcal{M}$ for which element K is unit with respect to $\mathcal{M}$ (see proof in [99]). The consequence is that the function *unit with respect to* defines classes of equivalences of discrete elements. Thus, in the continuous mesh framework, a metric tensor $\mathcal{M}$ is called a **continuous element**. It is used to model all discrete elements that are unit for $\mathcal{M}$. Geometric quantities associated with a continuous element can be computed.

Similarly, in the continuous mesh framework, a **continuous mesh** of a domain Ω is defined by a collection of continuous elements $\mathbf{M} = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$, *i.e.,* a Riemannian metric space. It is used to model all meshes that are unit for $\mathbf{M}$. The properties of the continuous mesh can be exhibited by rewriting $\mathbf{M}$ in order to distinguish local properties from global ones:

A Riemannian metric space $\mathbf{M} = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ locally is written:

$$\forall \mathbf{x} \in \Omega, \quad \mathcal{M}(\mathbf{x}) = d^{\frac{2}{3}}(\mathbf{x})\, \mathcal{R}(\mathbf{x}) \begin{pmatrix} r_1^{-\frac{2}{3}}(\mathbf{x}) & & \\ & r_2^{-\frac{2}{3}}(\mathbf{x}) & \\ & & r_3^{-\frac{2}{3}}(\mathbf{x}) \end{pmatrix} {}^t\mathcal{R}(\mathbf{x}),$$

where

- density d is equal to: $d = (\lambda_1\lambda_2\lambda_3)^{\frac{1}{2}} = (h_1h_2h_3)^{-1}$, with λ_i the eigenvalues of $\mathcal{M}$
- anisotropic quotients r_i are equal to: $r_i = h_i^3\,(h_1h_2h_3)^{-1}$
- $\mathcal{R}$ is the eigenvectors matrix of $\mathcal{M}$ representing the orientation.

The density d controls only the local level of accuracy of $\mathbf{M}$. Increasing or decreasing d does not change the anisotropic properties or the orientation. The anisotropy property is given by the anisotropic quotients r_i and the orientation by matrix $\mathcal{R}$. We also define the complexity $\mathcal{C}$ of $\mathbf{M}$:

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

This real-value parameter quantifies the level of accuracy of $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$. The correspondence between discrete and continuous entities is summarized in Table 1.2.

Discrete	**Continuous**
Element K	Metric tensor $\mathcal{M}$
Mesh $\mathcal{H}$ of Ω_h	Riemannian metric space $\mathbf{M} = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$
Number of vertices N_v	Complexity $\mathcal{C}(\mathbf{M}) = \int_\Omega \sqrt{\det(\mathcal{M}(\mathbf{x}))}\, d\mathbf{x} = \mathcal{N}$
Linear interpolate $\Pi_h u$	Continuous linear interpolate $\pi_{\mathcal{M}} u$

Table 1.2: Discrete entities and their continuous counterparts.

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

Mesh adaptation consists in finding the mesh $\mathcal{H}$ of a domain Ω that minimizes a given error for a given function u. For the sake of simplicity, we consider here the linear interpolation error $u - \Pi_h u$ controlled in L^p norm and that u is twice continuously differentiable. Note that considering other norms also works [75]. The problem is thus stated in an *a priori* way:

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

(P) is a global combinatorial problem which turns out to be intractable practically. Indeed, this would require the simultaneous optimization of both the mesh topology and the vertices location. Consequently, simpler problems are considered to approximate the solution, and error approximations are performed, that are equivalent to a steepest descent algorithm converging only to a local minimum with poor convergence properties.

This drawback arises because a minimization on a discrete mesh is directly considered. In order to prevent it, we address the resolution of (P) in a continuous setting. Consequently, (P) is recast as a continuous optimization problem where the discrete interpolation error is replaced by the continuous one:

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

where $\pi_{\mathcal{M}}$ is the continuous interpolate defined by:

$$\pi_{\mathcal{M}} u(\mathbf{a}) = u(\mathbf{a}) + \nabla u(\mathbf{a}) + \frac{1}{20} \text{trace}\big(\mathcal{M}(\mathbf{a})^{-\frac{1}{2}} \, |H(\mathbf{a})| \, \mathcal{M}(\mathbf{a})^{-\frac{1}{2}}\big) .$$

In contrast to discrete-based studies, the continuous formulation succeeds in solving the optimal interpolation error problem globally by using calculus of variations.

Optimal continuous mesh. Using the definition of the linear continuous interpolate $\pi_{\mathcal{M}}$, it is then possible to set the well-posed global optimization problem of finding the optimal continuous mesh minimizing the continuous interpolation error in L^p norm:

$$\begin{aligned} \text{Find } \mathbf{M}_{L^p} = \min_{\mathbf{M}} \mathbf{E}_{L^p}(\mathbf{M}) &= \left(\int_\Omega \big(u(\mathbf{x}) - \pi_{\mathcal{M}} u(\mathbf{x})\big)^p \, \mathrm{d}\mathbf{x} \right)^{\frac{1}{p}} \\ &= \left(\int_\Omega \text{trace}\Big(\mathcal{M}(\mathbf{x})^{-\frac{1}{2}} |H_u(\mathbf{x})| \mathcal{M}(\mathbf{x})^{-\frac{1}{2}}\Big)^p \, \mathrm{d}\mathbf{x} \right)^{\frac{1}{p}} , \end{aligned} \tag{1.9}$$

under the constraint $\mathcal{C}(\mathbf{M}) = \displaystyle\int_\Omega d(\mathbf{x}) \, \mathrm{d}\mathbf{x} = \mathcal{N}$.

The constraint on the complexity is added to avoid the trivial solution where all $(h_i)_{i=1,3}$ are zero which provides a null error. Unlike a discrete analysis, this problem can be solved globally by using calculus of variations that is well-defined on the space of continuous meshes. In [100], it is proved that Problem (1.9) admits a unique solution:

$$\mathcal{M}_{L^p}(\mathbf{x}) = \mathcal{N}^{\frac{2}{3}} \left(\int_\Omega \det(|H_u(\bar{\mathbf{x}})|)^{\frac{p}{2p+3}} \, \mathrm{d}\bar{\mathbf{x}} \right)^{-\frac{2}{3}} \det(|H_u(\mathbf{x})|)^{\frac{-1}{2p+3}} \, |H_u(\mathbf{x})| , \tag{1.10}$$

where H_u is the Hessian of u.

1.5 Feature-based anisotropic mesh adaptation for steady flows

This section presents the classic mesh adaptation process for steady flows, which is a fixed-point algorithm where both the solution and the mesh are converged. Starting from an initial -coarse- mesh, adapted mesh generations are performed iteratively in order to progressively capture all the physics, including the smallest scales. Thus, at each stage of this fixed-point algorithm a flow solver computation is performed, followed by an error estimation of the solution and a mesh regeneration. The error estimate used in this process is based on the control of the interpolation error in L^p norm. We consider a **feature-based** (or **Hessian-based**) error estimate (see [160, 135, 142, 54, 18, 88, 65, 40],...) whose goal is to derive the best mesh to compute the characteristics of a given sensor w.

Note that other error estimates exist, such as the **goal-oriented** (adjoint-based) which aims at deriving the best mesh to observe a given output scalar functional $j(w) = (g, w)$ [164, 80, 144, 169, 87, 170].

1.5.1 Mesh adaptation algorithm for numerical simulations

Anisotropic mesh adaptation is a non-linear problem, therefore an iterative procedure is required to solve this problem. For stationary simulations, an adaptive computation is carried out *via* a mesh adaptation loop inside which an algorithmic convergence of the mesh-solution couple is sought. This mesh adaptation loop is described in Algorithm 1 and illustrated in Figure 1.9, where $\mathcal{H}$, $\mathcal{S}$ and $\mathcal{M}$ denote respectively meshes, solutions and metrics.

Initial mesh and solution $(\mathcal{H}_0, \mathcal{S}_0^0)$ and set targeted complexity $\mathcal{N}$.
For $i = 0, n_{adap}$

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

EndFor

Algorithm 1: Mesh Adaptation Loop for Steady Flows

Note that this loop is applied several times for a sequence of given mesh complexities, for instance $\mathcal{N}$, $2\mathcal{N}$, etc. This process is illustrated by Figure 2.1 through the example of a 2D transonic NACA airfoil. Step 1 (solution computation) is detailed in Chapter 3, step 3 (mesh generation) in Section 1.3. For step 4 (interpolation) we refer to [7]. We now focus on step 2, by presenting the Hessian-based error estimate.

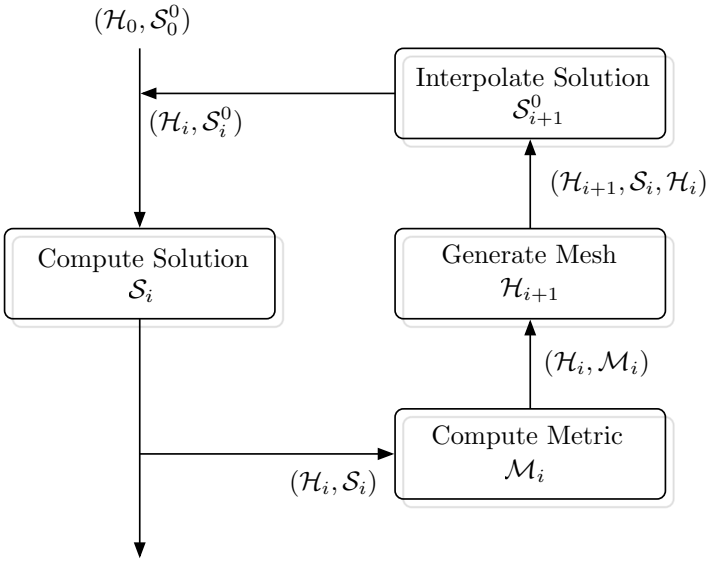
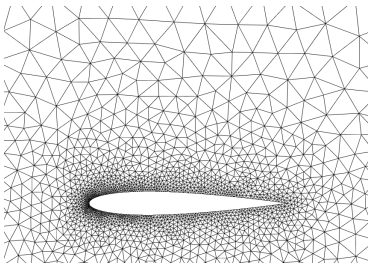
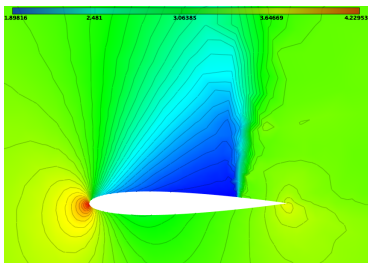


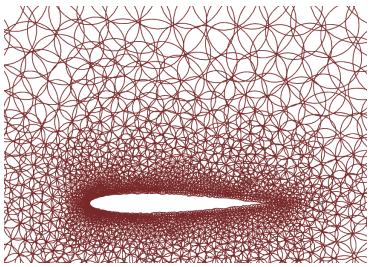
Figure 1.9: Mesh adaptation algorithm.



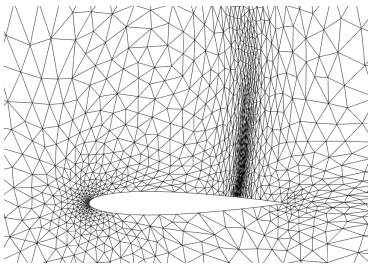
(a) Initial mesh $\mathcal{H}_0$.



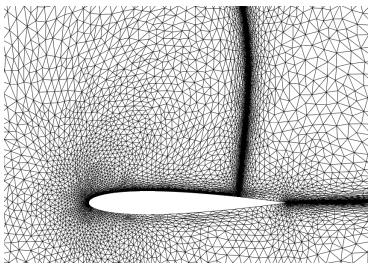
(b) Initial solution $\mathcal{S}_0$.



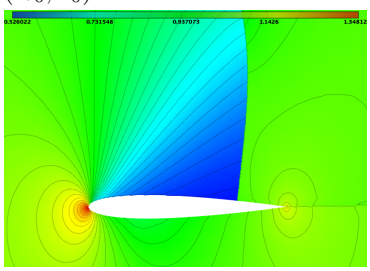
(c) Metric $\mathcal{M}_{L^2,0}$ computed according to an error estimation of $(\mathcal{H}_0, \mathcal{S}_0)$.



(d) Adapted mesh obtained after 3 stages of Algorithm 1 (2 200 vertices).



(e) After 10 stages (22 000 vertices).



(f) Corresponding final solution at stage 10.

Figure 1.10: Illustration of the steady mesh adaptation process (see Algorithm 1).

1.5.2 Hessian-based anisotropic mesh adaptation

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

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

We demonstrate how this problem can be simplified, under some assumptions, to the specification of a mesh that is optimal for some kind of interpolation error.

Controlling the approximation error

We describe how the interpolation theory is applied when only u_h, a piecewise linear approximation of the solution, is known. Indeed, in this particular case, the interpolation error estimate (1.10) cannot be applied directly to u or to u_h.

Let $\bar{V}_h^k$ be the space of piecewise polynomials of degree k (possibly discontinuous) and V_h^k be the space of continuous piecewise polynomials of degree k associated with a given mesh $\mathcal{H}$ of domain Ω_h. We denote by R_h a reconstruction operator applied to numerical approximation u_h. This reconstruction operator can be either a recovery process [172], a hierarchical basis [12], or an operator connected to an a posteriori estimate [76]. We assume that the reconstruction $R_h u_h$ is better than u_h for a given norm $\|.\|$ in the sense that:

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

From the triangle inequality, we deduce:

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

If reconstruction operator R_h has the property: $\Pi_h R_h \phi_h = \phi_h \,,\; \forall \phi_h \in V_h^1$, (meaning that R_h preserves the node value) the approximation error of the solution can be bounded by the interpolation error on the recovered function $R_h u_h$:

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

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

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

In the context of numerical simulations, u_h lies in V_h^1 and its derivatives ∇u_h in $\bar{V}_h^0$. We propose

a reconstruction operator from V_h^1 into V_h^2 based on $\mathbb{P}_2$ Lagrange finite element test functions. As the approximate solution u_h is only known at mesh vertices, we need to reconstruct mid-edge values. To this end, we consider the L^2-projection operator $\mathcal{P} : \bar{V}_h^0 \to V_h^1$ defined by [39]:

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

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

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

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

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

where $\psi_{\mathbf{p}} = \phi_{\mathbf{p}}\,(2\,\phi_{\mathbf{p}} - 1)$ and $\psi_{\mathbf{e}} = 4\,\phi_{\mathbf{p}}\,\phi_{\mathbf{q}}$ are the $\mathbb{P}_2$ Lagrange test functions. This reconstructed function can be rewritten $R_h u_h = u_h + z_h$ and by definition verifies:

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

Therefore, we deduce:

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

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

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

Note that the Hessian of z_h lies in $\bar{V}_h^0$. If nodal values are needed to build $\mathcal{M}_{L^p}$, then the L^2-projection operator can be applied to these Hessians [39]. This recovery procedure is similar to those of [172, 173]. Other reconstruction operators can be applied such as the double L^2-projection, the least square method or the Green formula based approach.

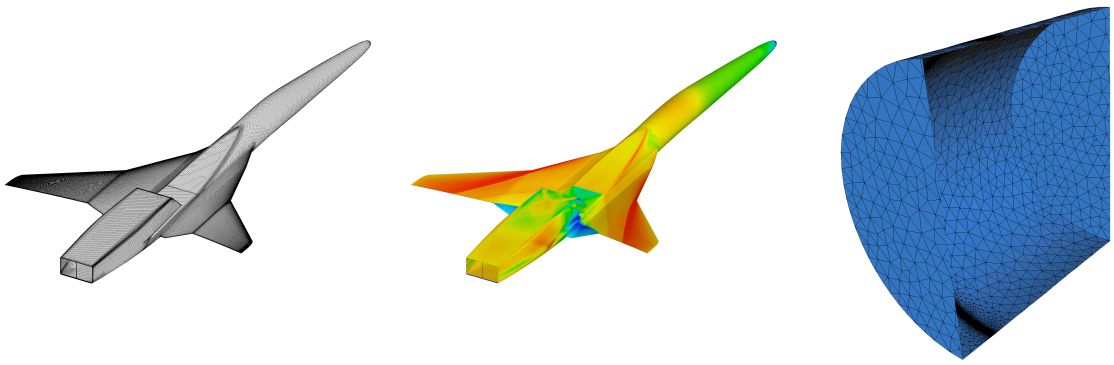
1.6 Application to a supersonic business jet (SSBJ)

In this section, we present an application of anisotropic feature-based mesh adaptation. We consider a supersonic flow around a low-boom-shaped business jet geometry provided by Dassault-Aviation in the

frame of the HISAC European project [73]. We present the results obtained using mesh adaptation [7]. The L^p interpolation error estimate is applied to the numerical solution.

1.6.1 Case description

The SSBJ geometry is depicted in Figure 1.11. It is a complex low-boom design: engines are integrated over the fuselage to minimize the impact of the nacelles on the sonic boom. Moreover, the wing has a double dihedral angle. The first dihedral angle is at the junction of the wing and the fuselage. The second one is where the wing swept angle change. The aircraft's length is 42 meters and it has a wing span of 20 meters. The surface mesh size on the aircraft geometry ranges between 0.2 mm and 12 cm. This already represents a size variation of five orders of magnitude with respect to size of the aircraft. The SSBJ geometry is considered inside a cylindrical computational domain of 2.25 km length and 1.5 km diameter. This represents a scale factor of 10^7 if the size of the domain is compared to the maximal accuracy of the low boom jet surface mesh.



(a) Surface mesh.

(b) Solution (local Mach number).

(c) The 2.5 kilometers cylindrical computational domain: the SSBJ is represented by a little dot !

Figure 1.11: Supersonic SSBJ. Presentation of the low-boom-shaped supersonic business jet geometry.

The jet is flying at cruise with Mach number 1.6, an angle of attack of 3° and an altitude of 45,000 feet. Such supersonic flows involve highly anisotropic physical features. Indeed, as for a body flying at a supersonic speed, each geometric singularity generates a cone-shaped shock wave ; a multitude of conic shock waves are emitted by the aircraft geometry. They generally coalesce around the aircraft and propagate to the ground. The goal, here, is to accurately compute the complete pressure field around the aircraft.

1.6.2 Adaptive results

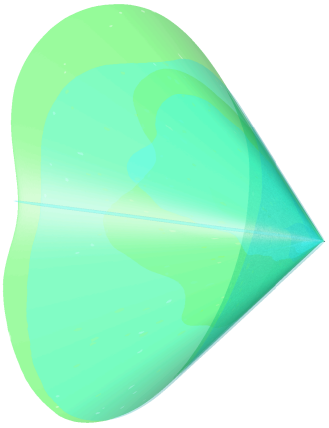
The interpolation error on the local Mach number is controlled in L^2 norm. 32 mesh adaptation iterations are performed. The adaptation loop is split into 4 steps with an increasing complexity specification at each step. Within each step, an adapted mesh at fixed complexity is converged. Final step meshes are used for the computation of the global mesh convergence order. Starting from a coarse uniform mesh containing 772 572 vertices and 3 768 534 tetrahedra, the final adapted mesh contains 9.1 million vertices and 53.9 million tetrahedra. This mesh is illustrated in Figure 1.13. The adapted meshes obtained are highly anisotropic. For the last one, the mean anisotropic ratio is 372 and the mean anisotropic quotient is 49 051. The last quantity signifies that the anisotropy leads to a reduction in mesh complexity by more than four orders of magnitude as compared to an isotropic adapted mesh.

Such adapted meshes considerably enhance the efficiency of the flow solver. In particular, we make the following observations.

- Numerous shock waves (Mach cones) have been accurately computed in the SSBJ near-field, see Figure 1.13 (right).
- All the details and scale of the solution have been captured and are represented in the mesh.
- Mesh refinements along Mach cones have been propagated in the whole computational domain with high accuracy (small size) allowing an accurate flow prediction to be achieved everywhere, Figure 1.12 (left) and 1.13 (left). This result highlights the fact that the numerical dissipation of the flow solver is drastically reduced thanks to anisotropic mesh refinement.
- The global spatial second order of convergence is asymptotically reached for the local Mach number on the sequence of adapted meshes, see Figure 1.12 (right).

1.7 Conclusion

In this chapter, we reviewed previous research in the field of mesh adaptation, with a focus on some of the main research issues faced at the beginning of the 2000s. Addressing these research issues made it possible to successfully apply anisotropic mesh adaptation to the simulation of complex three dimensional **inviscid** flows: sonic boom prediction, blast propagation, acoustic waves etc. It has been established that it has the ability to (i) substantially optimize the tradeoff between accuracy of the solution and the number of degrees of freedom (thus the computational time), (ii) accurately capture all scales of the physical flow by automatically detecting the regions of interests where the mesh needs more resolution, (iii) reduce the numerical scheme dissipation by automatically taking into account the anisotropy of the physics, and (iv) obtain an early mesh convergence: high order asymptotic rate of convergence even for



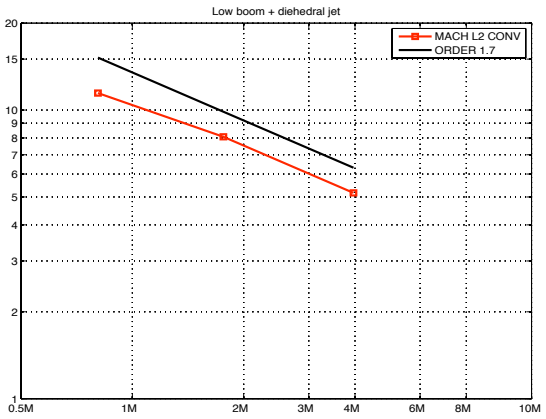


Figure 1.12: Supersonic SSBJ. Left, Mach cone emitted by the SSBJ. The maximal Mach cone diameter is 2.5 km. The solution is accurately propagated in the whole computational domain. Right, global L^2 norm spatial convergence for the local Mach number on a sequence of adapted meshes.

non-regular flows.

In this thesis, we present a coupling of mesh adaptation with multigrid methods, as well as adaptive strategies for viscous simulations. Additional contributions are presented for the two main components of the adaptive loop: mesh generation and solution computation.

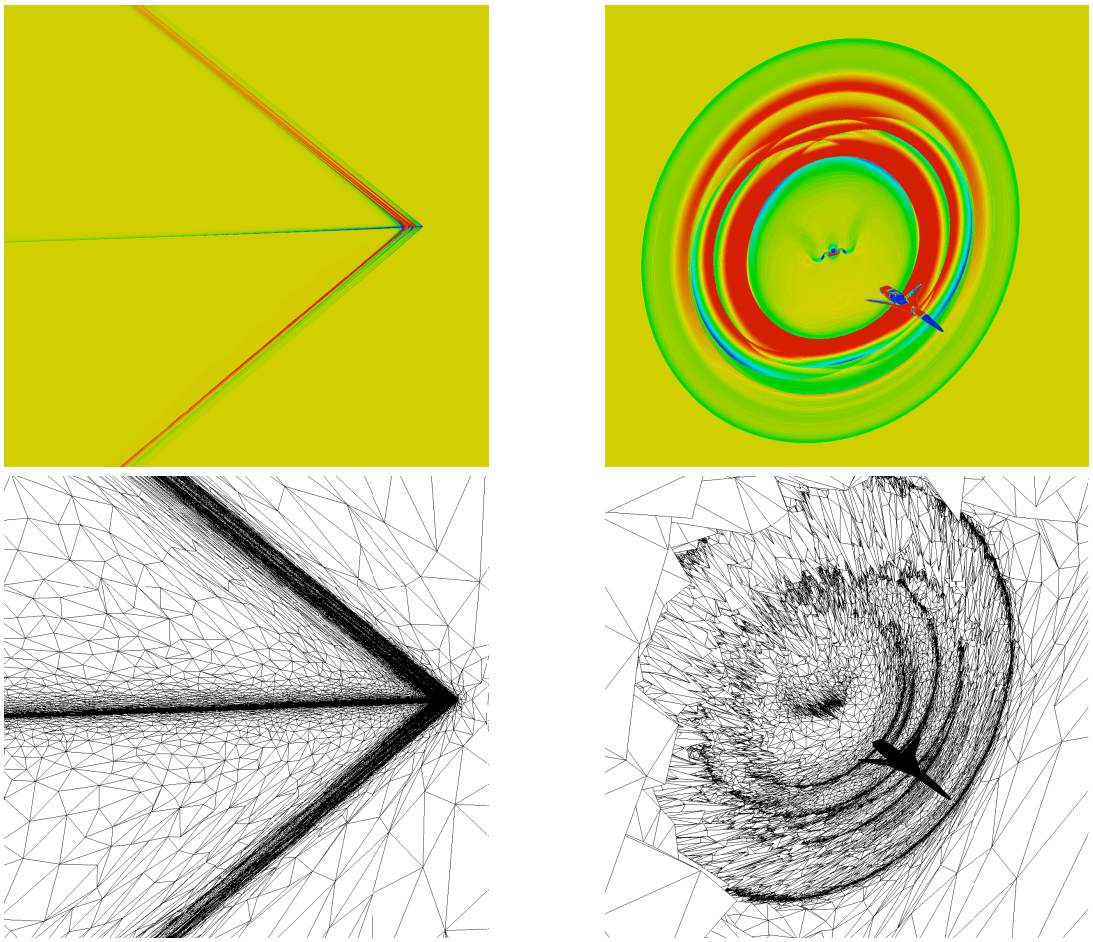


Figure 1.13: Supersonic SSBJ. Views of the local mach number iso-values (top) and the final anisotropic adapted mesh composed of 9.1 million vertices and 53.9 million tetrahedra (bottom) in the symmetry plane (left) and in a plane behind the aircraft orthogonal to the jet path (right).

Chapter 2

Parallel Anisotropic Mesh Adaptation

Contents

In this chapter, we present a parallel strategy, which we devised in order to generate large-size adapted anisotropic meshes ($O(10^8 - 10^9)$ elements), as are required in many scientific computing applications, and in particular in CFD when dealing with complex flows around complex geometries. We target moderate scale parallel computational resources as are typically found in R&D units where the number of cores ranges in $O(10^2 - 10^3)$. Both distributed and shared memory architectures are handled. Our strategy is based on a typical domain splitting algorithm allowing us to remesh the partitions in parallel.

2.1 Introduction

2.1.1 Motivation

As explained in the introduction of this thesis, problems studied by the CFD community may require billions of degrees of freedom to ensure a high-fidelity prediction of the physical phenomena. To satisfy this need, many numerical platforms (numerical solver, solution visualization) have been developed for parallel architectures (distributed or shared-memory). Although some simulations are performed on thousands of processors, recent studies show that the vast majority of R&D applications are run on a daily basis on smaller architectures of less than 1 000 cores [2, 51].

In the computational pipeline, mesh generation or adaptation is a crucial step, as the existence of a mesh (especially when dealing with complex geometries) is a necessary condition to start a simulation. The cost in terms of CPU of the mesh generation step is a major concern when very large meshes are required. This cost must remain low enough in comparison to the solver CPU time to be used in practice. This is particularly true in the context of adaptive simulations, as the remeshing step is repeated at each stage of the classical mesh adaptation loop (see Section 1.5.1 for a description of the adaptive process). To address this issue, we developed a parallel anisotropic mesh adaptation algorithm suitable for the aforementioned small parallel architectures (around 1000 cores). Our target is to generate anisotropic adapted meshes containing around one billion elements in less than 20min on 120 cores.

2.1.2 Research issues

The parallelization of the meshing/remeshing step is a complex problem. We list below the research issues that must be addressed.

Robustness of surface and volume remeshing When considering the coarse-grained strategy, parallel mesh generators or parallel local remeshers generally adapt either the surface or the volume mesh. In [84, 93], the fine surface mesh is unchanged during the parallel meshing process. When anisotropic meshes are used, being able to adapt the surface and the volume into a single thread is necessary in order to increase robustness [104]. However, adapting both the surface and the volume meshes at the same

time induces 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 amount of work. Graph-based techniques tend to minimize the size of the cuts (or integer cost function) which is not the primary intent in remeshing. This becomes even more critical for anisotropic mesh adaptation where refinements have a large variation in the computational domain. Additional developments of graph-based methods are then necessary to work in the anisotropic framework [84]. Domain partitioning also represents one of the main parallel overhead of the method. In particular, general purpose graph-partitioners cannot take into account the different geometrical properties of the sub-domain to be partitioned. Indeed, splitting an initial domain is completely different from partitioning an interface mesh.

Partition remeshing. This is the core component of 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.

Out-of-core meshing. Out-of-core meshing was originally designed to store the parts of the mesh that were completed on disk so as to reduce the memory footprint [8]. 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 used advantageously. On shared memory machines (with 100-200 cores), if the memory used by a thread is bigger that the memory of a socket, then the memory exchange between neighboring sockets implies a huge overhead of the sequential time (when running the procedure with only one thread). This phenomenon is even more critical for NUMA architectures.

2.1.3 State-of-the-art

Parallel mesh generation has been an active field of research [93, 161, 78, 57]. Two main frames of parallelization exist: coarse-grained [47, 84, 93], and fine-grained [57, 155, 37, 134]. A fine-grained parallelization requires directly to implement in parallel all the mesh modification operators at the lowest level: insertion, collapse, swap etc. This usually implies using of specific data structures to handle distributed dynamic meshes, especially for adaptive procedures. 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. This approach has also been extended to adaptive frameworks [47, 84].

Current state-of-art parallel mesh generation approaches [47, 89] for unstructured (and adapted) meshes require thousands of cores (4092-200 000 cores) to generate meshes containing a billion elements. Our scope is to make this size of meshes affordable on smaller parallel architectures (120-480 cores) in an acceptable runtime for a design process (less than 20 min). To this end, we devise an approach based on coarse-grained parallelization.

2.1.4 Our approach

Our procedure is based on standard coarse-grained parallel strategies [94, 84, 93] where the initial domain is split into several sub-domains that are then remeshed in parallel. The interfaces between the partitions are constrained during the meshing phase. Both the volume and the surface mesh are adapted simultaneously and the efficiency of the method is independent of the complexity of the geometry.

The originality of our method lies in the following key features:

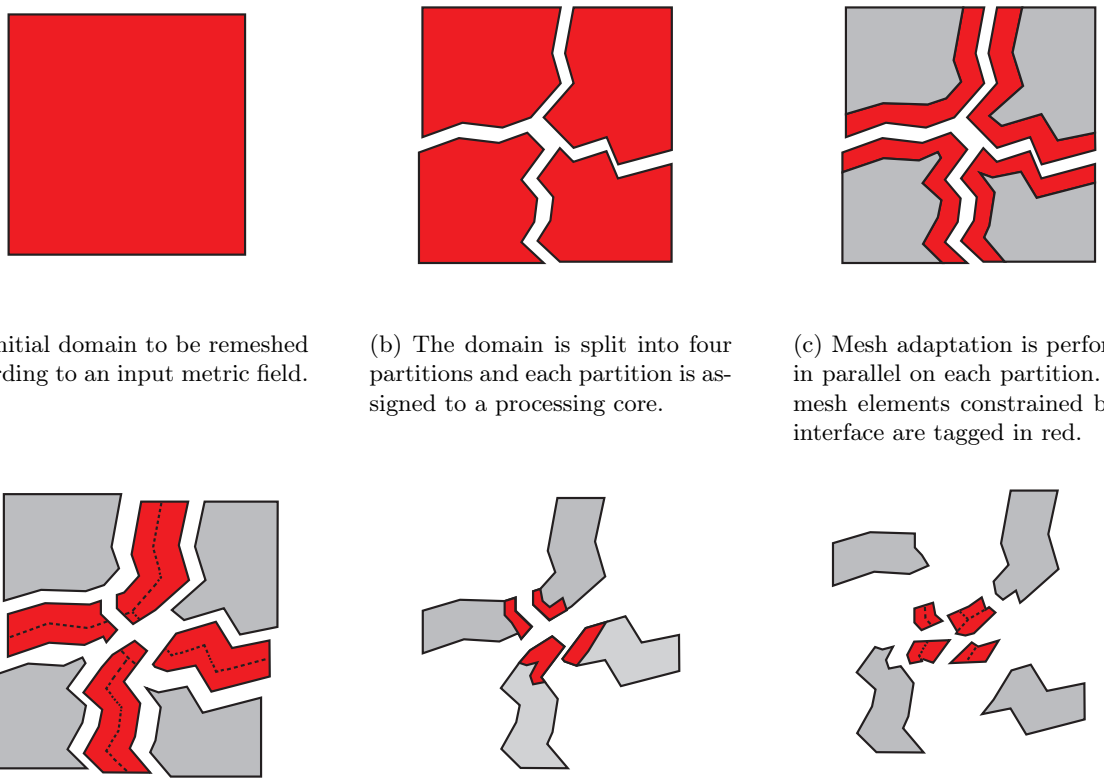
- Metric-based static load-balancing: weights are used to *a priori* equilibrate the work on each sub-domain. This is necessary to efficiently handle non uniform refinements (in terms of sizes and directions).
- Dedicated mesh partitioning techniques to (re)split (complex) interface meshes: we define two distinct partitioning techniques depending on the level of refinement. In particular, we take advantage of the geometry of the mesh at the interface to guarantee that the number of constrained faces are minimized at each step.
- A fast, robust and generic sequential cavity-based mesh modification kernel.
- Out-of-core storing of completed mesh parts to reduce the memory footprint.

Using this approach, we show that we are able to generate (uniform, isotropic and anisotropic) meshes with more than 1 billion tetrahedra in less than 20 minutes on 120 cores.

The chapter is organized as follows. In Section 2.2, we start by giving a overall description of the parallel algorithm and its main steps. Then we detail the key features of the process. Our domain partitioning algorithm is described in Section 2.3 along with load balancing considerations. In Section 2.4, we present how we deal with mesh elements constrained by an interface between two partitions. Finally, numerical examples are given in Section 2.5.

2.2 Overview of the parallel algorithm

We give an overview of the parallel mesh adaptation algorithm. More details on domain partitioning and interface remeshing are provided in the following sections. The method is illustrated through the schematic example of the remeshing of a square on 4 processors, see Figure 2.1.



(a) Initial domain to be remeshed according to an input metric field.

(b) The domain is split into four partitions and each partition is assigned to a processing core.

(c) Mesh adaptation is performed in parallel on each partition. The mesh elements constrained by an interface are tagged in red.

(d) Interface elements are tagged and resplit.

(e) Interface are remeshed in parallel.

(f) New interface elements are extracted and re-split.

Figure 2.1: Illustration of the parallel mesh adaptation algorithm. In red: regions of the computational domain that need to be remeshed (i.e. whose elements are not unit for the input metric). In grey: regions of the domain that have been successfully remeshed.

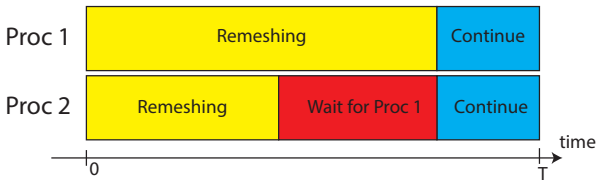
We are given an initial mesh and a metric tensor field (Figure 2.1a). The steps of the algorithm are the following:

1. **Domain partitioning** (Figures 2.1a and 2.1b): the domain is split and each partition is assigned to a processing core. Preserving the load-balancing is crucial in order to avoid performance loss (see Figure 2.2).

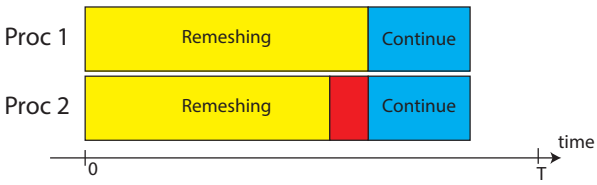
2. **Parallel adaptation** (Figure 2.1c): each processing core performs a mesh adaptation on its assigned part. The elements at the interface between two parts remain unchanged.

3. **Interface elements** (Figure 2.1d): elements constrained by an interface are extracted and re-split in parallel.

4. **Iterate** (Figures 2.1e and 2.1f): we go back to Step 2 and iterate until there are no more elements constrained by an interface (their number is expected to converge toward zero).



(a) Bad load-balancing. Proc 1 spends almost twice as much time remeshing as proc 2. Meanwhile, proc 1 is unused.

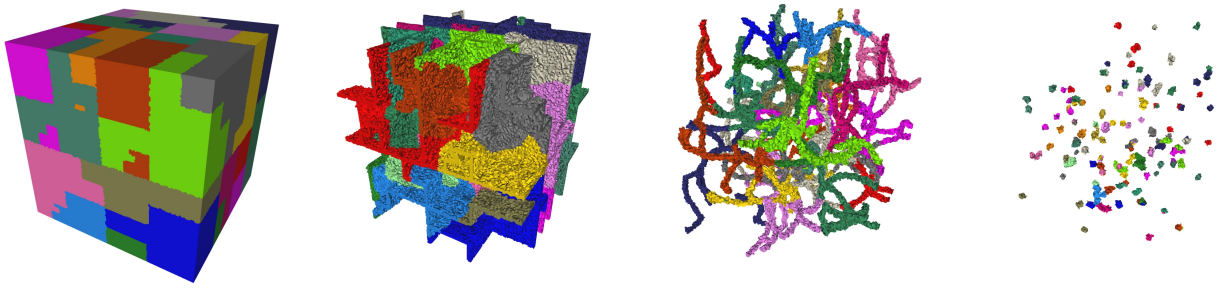


(b) A fairly good load-balancing.

Figure 2.2: The domain partitioning step must preserve the load-balancing.

2.3 Domain partitioning

In the context of parallel remeshing, the domain partitioning method must be fast, require low memory, able to handle domain with many connected components, and effective to balance the remeshing work. Moreover, we should have an efficient partitioning method for several levels of partitions. More precisely, we first - level 1 - split the domain volume. Level 2, we split the interface of the partitions of level 1; the interface 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 decomposition of a cubic domain into 32 partitions is shown in Figure 2.3. We observe that the domain topology varies drastically with the level.



Level 1 Level 2 Level 3 Level 4

Figure 2.3: Recursive partitioning into 32 sub-domains of a cubic domain for a constant work per element. From left to right, level 1, 2, 3 and 4 of partitioning. We observe that the domain topology varies drastically with the level.

2.3.1 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, *i.e.*, there is no communication and the partition interfaces are constrained. The work to be performed depends on the mesh operations used (insertion, collapse, swap, smoothing), the given metric field $\mathcal{M}$ and, also, on the initial mesh $\mathcal{H}$ natural metric field $\mathcal{M}_{\mathcal{H}}$. Indeed, if the initial mesh already satisfies the metric then nothing needs to be done. 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 $\mathcal{M}_K$ which is obtained by solving a simple linear system [99]. And, metric field $\mathcal{M}_{\mathcal{H}}$ is the union of the element metrics $\mathcal{M}_K$.

Isotropic case. Assuming the initial mesh is too coarse and is going to be only refined, the work per element is:

$$wrk_{vol}(K) = r_n \left(\frac{d_{\mathcal{M}_K}}{d_{\mathcal{M}}} - 1 \right),$$

where r_n is a constant defining the cost of the vertex insertion operator in dimension n and $d_{\mathcal{M}} = |K|\sqrt{\det \mathcal{M}}$ is the metric density. For an isotropic metric, metric $\mathcal{M}$ reduces to $h_{\mathcal{M}}^{-2}\mathcal{I}_n$ with $h_{\mathcal{M}}$ the local mesh size, and $d_{\mathcal{M}} = |K|h_{\mathcal{M}}^{-n}$. Thus, we get

$$wrk_{vol}(K) = r_n \left(\frac{h_{\mathcal{M}}^{-n}}{h_{\mathcal{M}_K}^{-n}} - 1 \right).$$

For instance, if element K has a constant size h and we are looking for a final mesh of size $h/2$ then the work is

$$wrk_{vol}(K) = r_n \left(2^n - 1\right).$$

But, this formula is not valid for coarsening. The opposite is required. Hence,

$$wrk_{vol}(K) = c_n \left(\frac{h_{\mathcal{M}_K}^{-n}}{h_{\mathcal{M}}^{-n}} - 1 \right).$$

where c_n is a constant defining the cost of the vertex collapse operator in dimension n. In our case, the local remeshing strategy uses a unique cavity operator for all mesh modifications (see Chapter 1), therefore all mesh modifications have exactly the same cost. We thus set: $r_n = c_n = 1$. Finally, the work per element is evaluated as

$$wrk_{vol}(K) = \max\left(\frac{h_{\mathcal{M}_K}}{h_{\mathcal{M}}}, \frac{h_{\mathcal{M}}}{h_{\mathcal{M}_K}} \right)^n - 1.$$

The previous relation is valid inside the volume. Now, we have to take into account the work necessary to remesh the surface. In the proposed method, each surface mesh modification requires constraint cavity construction, CAD re-projection and geometric topology check. Consequently, surface mesh modifications

are more costly than volume ones. To take into account this extrac-cost, the following modified formula proves to be very effective to produce an even distribution of the work:

$$wrk_{tot}(K) = wrk_{vol}(K) + N_f \times wrk_{vol}(K)\,,$$

where N_f is the number of boundary faces of the element which is most of the time one or two.

Anisotropic case. We cannot directly use the density of the anisotropic metric to define the work per element because the direction associated with each size must be taken into account. Indeed, 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. To consider the directions, a simultaneous reduction of both metrics ($\mathcal{M}$ and $\mathcal{M}_K$) is applied [60]. It provides a common basis $\{\mathbf{e}_i\}_{i=1,n}$ in which both associated matrices are diagonal. Then, the 1D density in each direction of the common basis is considered to define the work per element:

$$wrk_{vol}(K) = \left(\prod_{i=1}^{n} \max \left(\frac{\tilde{h}^i_{\mathcal{M}_K}}{\tilde{h}^i_{\mathcal{M}}} \,,\, \frac{\tilde{h}^i_{\mathcal{M}}}{\tilde{h}^i_{\mathcal{M}_K}} \right) \right) - 1\,,$$

where $\tilde{h}^i_{\mathcal{M}_K}$ (resp. $\tilde{h}^i_{\mathcal{M}}$) is the mesh (resp. metric) size with respect to direction $\mathbf{e}_i$, the i^{th} vector of the common basis.

2.3.2 Partitioning methods

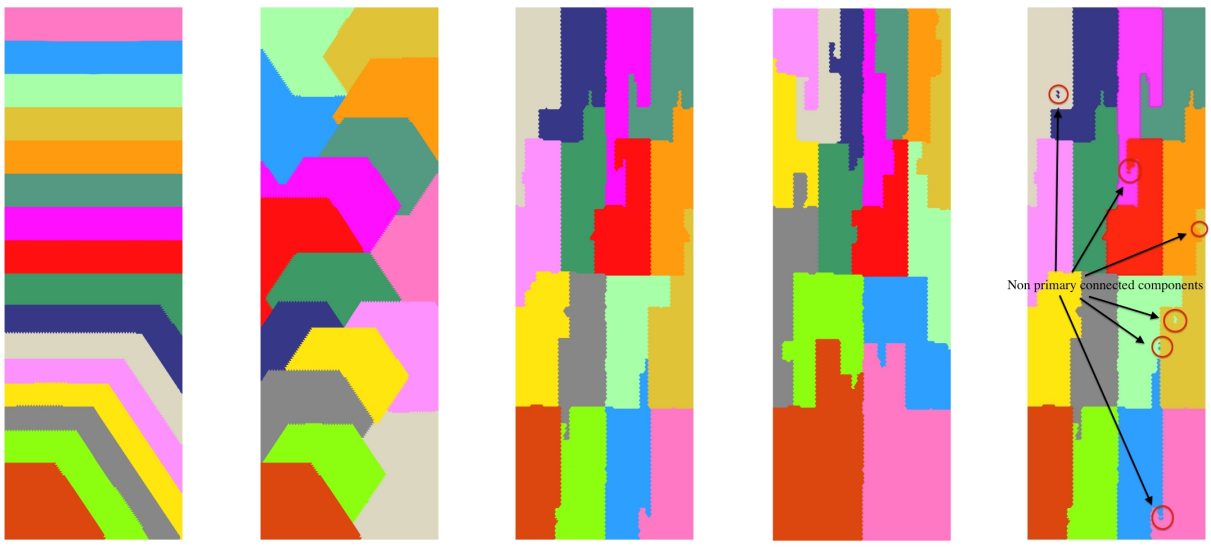
Before using any of the partitioning methods presented below, the mesh vertices are first renumbered using a Hilbert space filling curve based reordering [6]. A Hilbert index (the position on the curve) is associated with each vertex according to its position in space. This operation has a linear complexity and is straightforward to parallelize as there is no dependency. Then, the renumbering is deduced from the Hilbert indices of the vertices. Vertices are sorted using the standard C-library `quicksort`.

The domain partitioning problem can be viewed as a renumbering problem of the elements. In this 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 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 in the choice of the renumbering. Note that, for efficiency purposes, the elements are not explicitly reordered but are only assigned an index or a partition index on the fly.

Now, assuming that 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. This 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 elements ball of vertex 1), then we list the elements sharing vertex 2 (the elements 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 partitions with equal size interface (see Figure 2.4 (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 result of the previous level and thus it will not gather the interfaces of different sub-domains.

Breadth-first search (BFS) partitioning. Here, we start from an element root - generally, element 1 - and we add the neighbor elements of the root first. Then, we move to the next level of neighbors, in other words, we add the neighbor of the neighbors not already assigned. And so on. This splitting of the domain progresses by front. Each time an element is assigned, its non-assigned neighbors are added to a pile. The elements in this pile represent the current front. For level 1 domain, it results in layered partitions which contain only one connected component (see Figure 2.4 (a)) except the last one(s) which could be multi-connected. For level 2 or more domains, this method is able to gather the interfaces of different sub-domains but, as the pile is always growing, the number of connected components grows each time a bifurcation is encountered (see Figure 2.5 (a)). This leads to unbalanced sub-domains after the



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

Figure 2.4: 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-based (c) methods. Picture (d) shows the Hilbert-based partitioning with a linear work function (the work per element increases with y) which has to be compared with picture (c) for a constant work per element. Picture (e) shows the Hilbert-based partitioning before the connected components correction. Several isolated connected components appear. The result after the correction is shown in picture (c).

correction presented in Section 2.3.3.

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 a problem because the resulting partitions are multi-connected, cf. Figure 2.5 (a). One easy way to solve this issue is to reset the pile each time we deal with a new partition. The root of the new partition is the first element of the present pile, all the other elements are removed from the pile. For level 1 domain, it results in more circular (spherical) partitions (see Figure 2.4 (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 Figure 2.5 (c). We observe in Figure 2.3 that the size of the partitions interface mesh reduces at each level.

2.3.3 Correction of connected components

As the interfaces are constrained and not remeshed, the number of connected components per sub-domain should be minimized to maximized the work done by the remeshing strategy. In other words, each partition should have only one connected component if that is possible. All the elements of the same

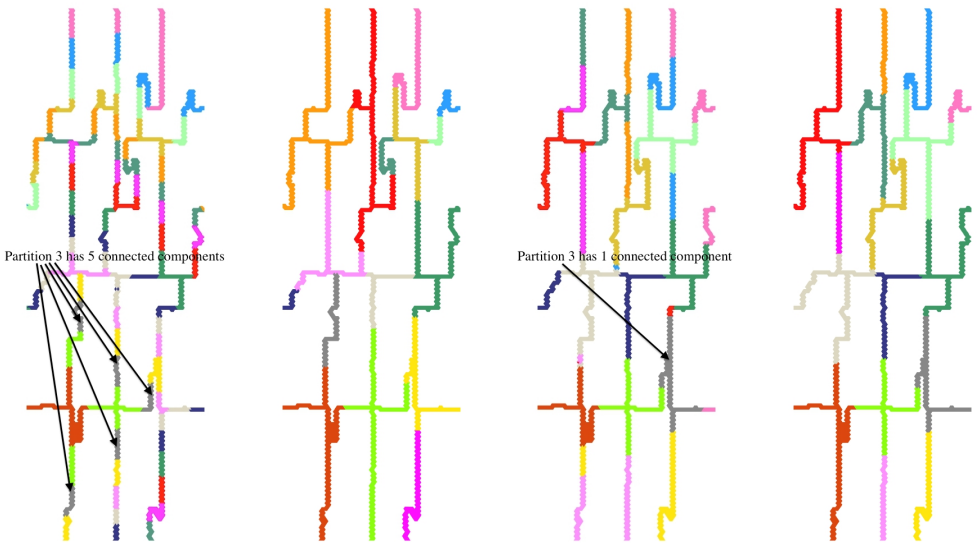


Figure 2.5: Partitioning into 16 sub-domains of a - level 2 - interface mesh of a rectangular domain for a constant work per element. The interface mesh results from the Hilbert-based partitioning of the level 1 domain. Partitions obtained with the BFS method before and after correction are shown in pictures (a) and (b), respectively. Many connected components are created for each partition (a) due to the bifurcations resulting in an unbalanced domain decomposition after correction (b). Partitions obtained with the BFS method with restart before and after correction are shown in pictures (c) and (d), respectively. Just a few isolated small connected components are created leading to a balanced domain decomposition after correction.

connected component are linked by at least one neighboring face.

After the domain splitting, a correction is applied to merge isolated connected components, see Figure 2.4 (e). First, for each sub-domain, 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 least work. The impact of this correction is illustrated in Figure 2.4 from (e) to (c).

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

2.3.4 Efficiency of the method

The domain partitioning methods presented minimize the memory requirement as the only data structures they use are: the elements list, the elements' neighbors list, the elements' partitions indices list and a pile.

They are efficient in terms of CPU time because the elements assignment to a sub-domain is done in one loop over the elements. Then, correcting the connected components 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 level 1, 2 and 3 domains, respectively, where the Hilbert-based partitioning has been used for level 1 domain and the BFS with restart partitioning has been used for level 2 and 3 domains.

2.4 Interface definition

The sequential mesh modification operator we use is `AMG`, our in-house meshing algorithm described in Section 1.3. Note that no specific modification of the sequential algorithm was made to use it in parallel, as it natively takes into account constrained boundary faces (defining interfaces). Moreover, the algorithm can handle non manifold geometries -i.e. configurations where three (or more) faces share the same edge (see Figure 2.6)- which is crucial when dealing with interfaces, although it is a challenging issue because of robustness considerations. In addition, the volume and the surface meshes are adapted simultaneously in order to keep a valid 3D mesh throughout the entire process. This guarantees the robustness of the complete remeshing step.

During the remeshing phase, the set of elements that surrounds the constrained faces defining the

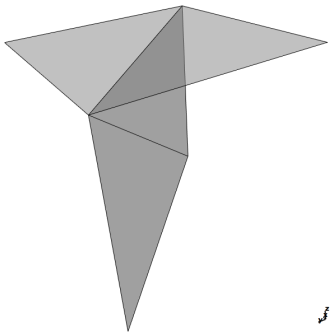
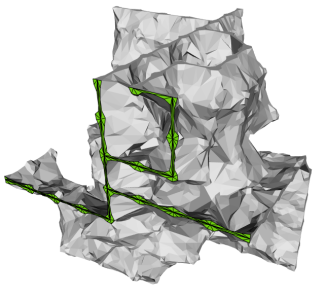
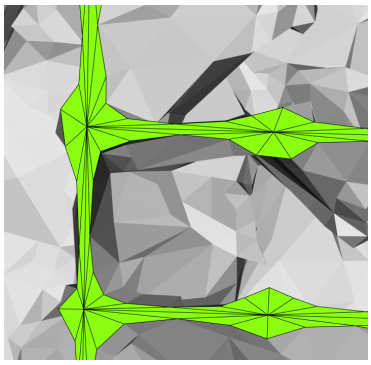


Figure 2.6: Example of a non-manifold configuration (here three faces share an edge).

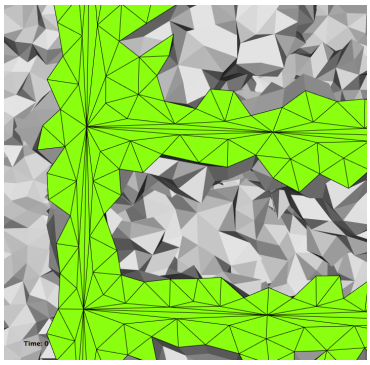
partition are not adapted. So, it is necessary to identify this set of interface elements in order to adapt them during the next level. To do so, a first choice is to introduce only the elements having at least one node on the interface boundary. This choice may work when the size of the mesh on the constrained faces is of the same order as the size imposed in the volume. When large size variation occurs, additional elements need to be part of the new interface volume mesh. An automatic way to find these elements is to add the relevant set of elements of the cavity [97] for each operator (insertion, collapse, etc.). In other words, for each element of the initial interface we compute the set of elements that belong to its cavity (as defined in Section 1.3). If these elements do not already belong to the interface mesh, we add them to it. This is illustrated in Figure 2.7 where a cube 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 Figure 2.7 (a)-(c). Including additional elements based on the cavity defining the relevant mesh modification operator gives additional room to the mesh generator to perform a quality modification, Figure 2.7 (b)-(d).



(a) Large view of the initial interface mesh (with no correction).



(b) Close-up view of the initial interface mesh.



(c) Close-up view of the interface mesh after correction. Cavity elements are added.

Figure 2.7: Definition of the interface mesh: example of a cube.

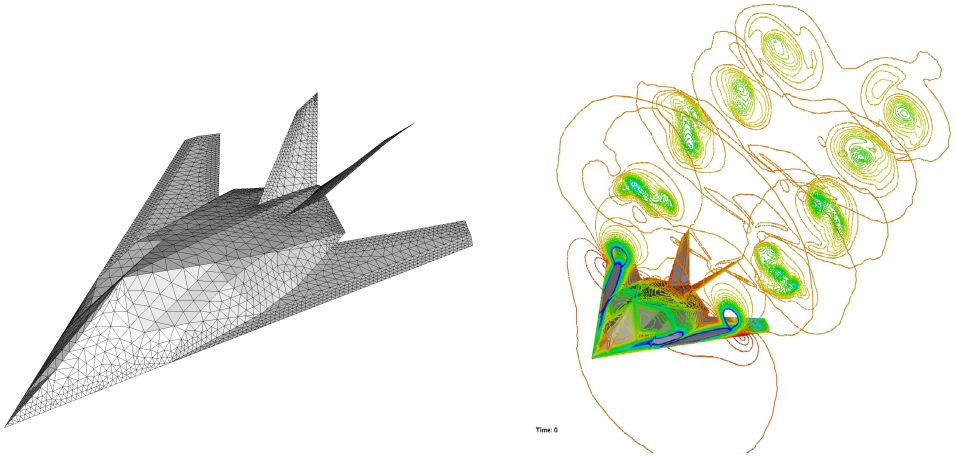


Figure 2.8: F117 test case. Geometry of the f117 aircraft (left) and representation of the vortical flow (right).

2.5 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/collapsed. 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.

Vortical flows on the F117 geometry. This adapted mesh generation is part of an unsteady adaptive simulation performed to accurately capture vortices generated by the delta-shaped wings of the F117 geometry, see Figure 2.8. The initial mesh of the simulation is depicted in Figure 2.9. It contains 1 619 947 vertices, 45740 triangles and 9 710 771 tetrahedra. The final adapted mesh is composed of 83 752 358 vertices, 539 658 triangles and 520 073 940 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 2.1. The maximal memory used per core is 1.25 Gb. The same example on 480 cores is reported in Table 2.2, 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 in the interfaces in the mesh, see Table 2.3 (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.

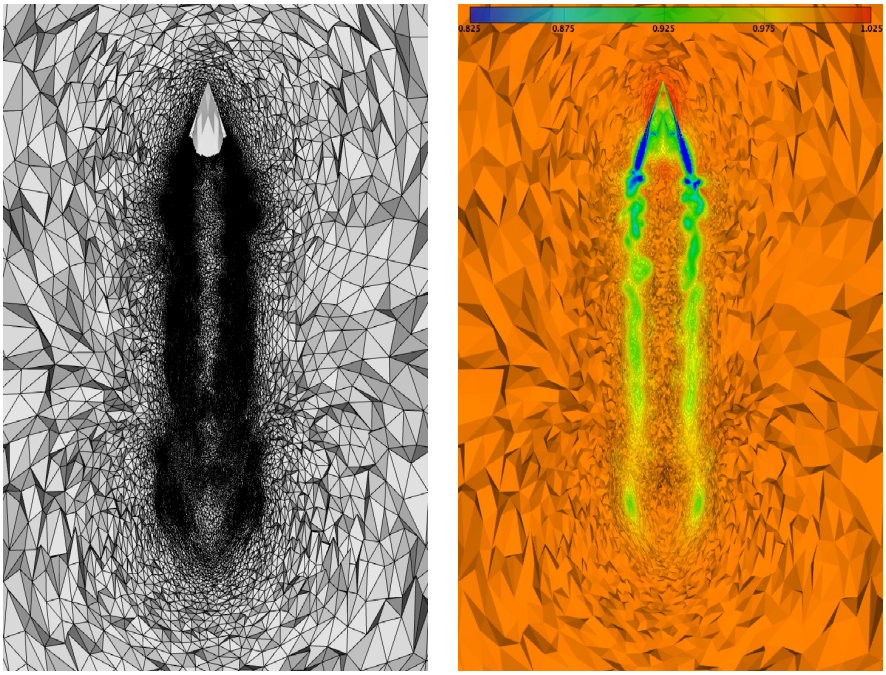


Figure 2.9: F117 test case. Left, top view of the mesh adapted to the local Mach number, and right, local Mach number iso-values.

Iteration	% done	# of tets in interface	# of tets inserted	CPU time (sec.)	# of cores used
1	84%	69 195 431	433 495 495	180.8	120
2	96%	1 692 739	502 706 732	95.0	120
3	99%	1 231 868	518 850 149	35.9	91
4	99%	6459	520 067 586	7.5	7
5	100%	0	520 073 940	1.7	1

Table 2.1: F117 test case on 120 cores. Table gathering the size of the interface, the number of inserted tetrahedra and the CPU time for each iteration.

Iteration	% done	# of tets in interface	# of tets inserted	CPU time (sec.)	# of cores used
1	76%	109 269 782	389 476 861	109.9	480
2	91%	42 836 303	486 695 293	67.0	480
3	98%	5 567 744	525 073 846	28.1	228
4	99%	32292	530 573 260	8.9	30
5	100%	0	530 605 308	2.3	1

Table 2.2: F117 test case on 480 cores. Table gathering the size of the interface, the number of inserted tetrahedra and the CPU time 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 2.3: Number of boundary 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	# of tets in interface	# of tets inserted	CPU time (sec.)	# of cores used
1	84%	89 577 773	919 345 377	577.3	120
2	95%	14 290 245	1 062 994 802	280.7	120
3	97%	1 290 855	1 089 035 610	56.3	120
4	97%	3636	1 090 321 352	8.0	7
5	100 %	0	1 090 324 952	2.1	1

Table 2.4: Tower-Bridge test case on 120 cores. Table gathering the size of the interface, the number of inserted tetrahedra and the CPU time for each iteration.

Iteration	% done	# of tets in interface	# of tets inserted	CPU time (sec.)	# of cores used
1	79%	193 529 057	922 145 088	255.8	480
2	93 %	52 837 674	1 115 428 211	106.7	379
3	96%	4 258 411	1 165 096 167	34.6	282
4	97%	27 095	1 169 283 585	23.0	23
5	100%	0	1 169 310 260	3.9	1

Table 2.5: Tower-Bridge test case on 480 cores. Table gathering the size of the interface, the number of inserted tetrahedra and the CPU time for each iteration.

Blast simulation on 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 drops to 16 min 30 s. The maximal memory used on 120 cores is 1.8Gb and reduces to 1Gb on 480 cores. In Tables 2.3 (right), 2.4 and 2.5, we report the convergence of the process. This example exemplifies the robustness of this approach with complex geometries. A view of the adapted surface mesh is depicted in Figure 2.11.

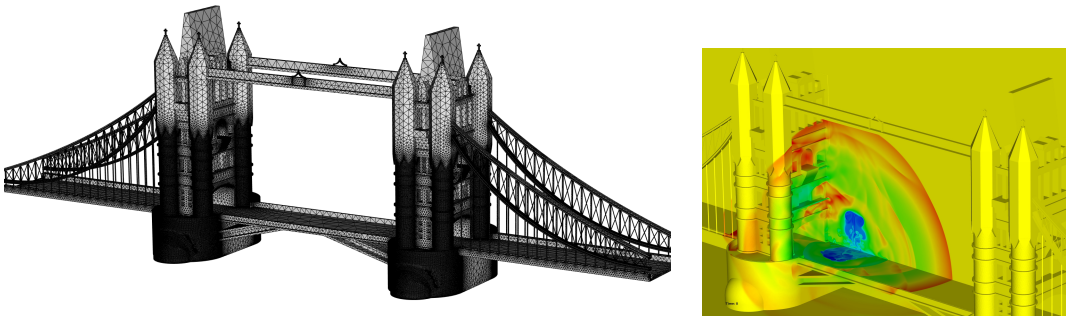


Figure 2.10: Tower-Bridge test case. Initial mesh and geometry (left) and density iso-values of the blast on an adapted mesh (right).

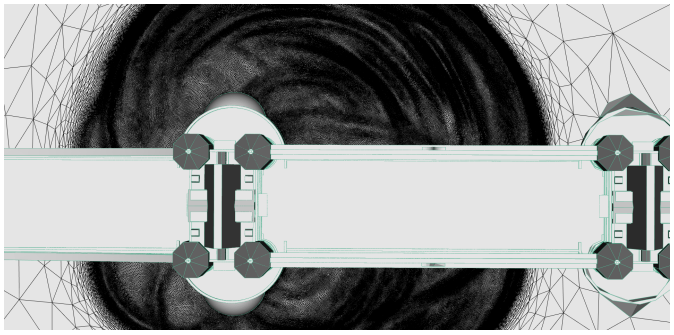


Figure 2.11: Tower-Bridge test case. Upper view of the adapted surface mesh showing the footprint of the blast on the Thames.

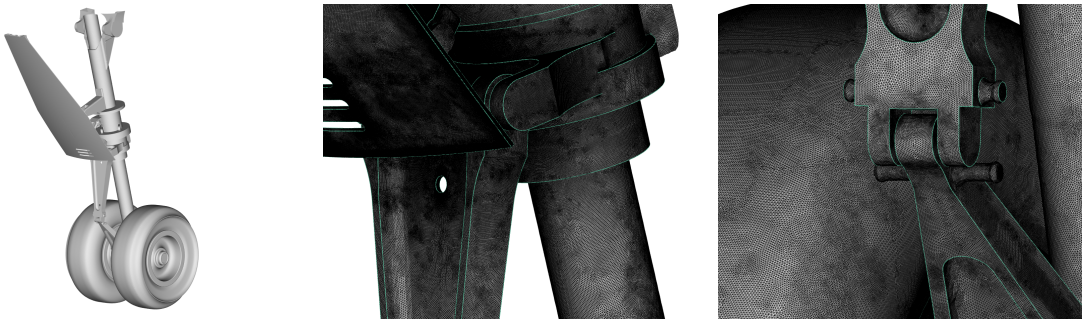


Figure 2.12: Landing gear test case. Geometry of the landing gear (left) and closer view of the surface mesh around some geometrical details (middle and right).

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 and 14 731 068 tetrahedra while the adapted mesh is composed of 184 608 096 vertices 14 431 356 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 (with the initial splitting and the final gathering). This example illustrates the stability of this strategy when the surface mesh contains most of the refinement. Indeed, the surface mesh is composed of more than 7.2 million vertices and 14.4 million triangles. Table 2.6 gathers all the data per iteration on this case. The geometry and closer view on the surface mesh are depicted in Figure 2.12.

Iteration	% done	# of tets in interface	# of tets inserted	CPU time (sec.)	# of cores used
1	84 %	89 718 245	1 009 783 723	487.5	120
2	91 %	16 368 313	1 107 015 758	126.7	120
3	92 %	645 035	1 122 857 778	36.6	87
4	97%	2 351	1 123 488 597	5.6	4
5	100%	0	1 123 490 929	1.7	1

Table 2.6: Landing gear test case on 120 cores. Table gathering the size of the interface, the number of inserted tetrahedra and the CPU time for each iteration.

2.6 Conclusion and future work

An efficient coarse-grained parallel strategy is proposed to generate large-size adaptive meshes. It can handle uniform, isotropic and anisotropic refinements. The volume and the surface meshes are adapted simultaneously and a valid mesh is kept throughout the process. The parallel resources are used to remove the memory impediment of the serial meshing software. Even if the remeshing is the only part of the process completely done in parallel, we still achieve reasonable CPU times. The CPU time for the meshing part ranges from 15 min to 30 min to generate adapted meshes containing 1 billion tetrahedra. The key components of the process are:

- a fast sequential cavity-based remesher that can handle constrained surface and non-manifold geometries during the remeshing,
- specific splitting of the interface mesh ensuring that the number of faces defining the interfaces tends to zero,
- a cavity-based correction of the interface mesh to ensure that enough elements are included in order to favor the success of the needed mesh modification operator at the next level.

Additional developments are needed to further reduce the total CPU time. The current work is directed at recovering the IOs with the remeshing. Indeed, as we use an out-of-core strategy, the final gathering can be partially done at the same time. Then, the partitioning techniques of the interfaces are also being currently extended to work efficiently in a parallel environment as well.

Part II

Contributions to `Wolf`, a 3D RANS Flow Solver

Chapter 3

Mixed finite element-volume MUSCL method

Contents

3.1 Introduction

The development of our in-house CFD flow solver `Wolf` started about ten years ago. The initial motivation was to develop a 2D/3D inviscid flow solver to validate anisotropic mesh adaptation. The choice was made to implement a vertex-centered finite volume scheme with an explicit time integration. It was successfully embedded in the mesh adaptation loop and good results were obtained for industrial cases such as blast propagation [5] or sonic boom prediction [101].

Four years ago, the decision was made to include turbulence modeling to simulate viscous flows, in the hope of extending adaptive methods to cases for which turbulence is an essential feature (such as the drag or the high-lift prediction).

The Navier-Stokes equations are unsteady by nature, which is why a common approach consists in averaging the governing equations of the flow in order to predict its non-fluctuating features using a steady method. In particular, Reynolds-Averaged Navier Stokes (RANS) equations have been widely used for the last 25 years. As the RANS equations are unclosed, a model is necessary to predict turbulent effects on the mean flow. This turbulence model is thus a key feature of solving the RANS equations but is also one of its largest source of uncertainty, as most models are known to be flawed in one way or another. Moreover, turbulence modeling raises convergence issues that are not present in a non-viscous context.

In the immediate vicinity of a viscous wall, highly-stretched quasi-structured meshes are mandatory to accurately capture viscous phenomena in the boundary layer. These boundary layer meshes bring two main difficulties from the point of view of the flow solver. First, the definition of the solver time step dt is homogeneous to the smallest height of the mesh h_{min}. Consequently, a single small-height element in the whole mesh is sufficient to considerably reduce the time step and thus increase the CPU time of the simulation. Knowing that viscous simulations require highly-stretched elements in the immediate vicinity of viscous walls, this is a serious complication compared to inviscid simulations. Second, building a boundary layer mesh in the near-wall regions leads to the generation of very larges meshes, which impact the total wall clock time of the simulations.

We solve the RANS equations using the Spalart-Allmaras one-equation model, which is a standard and well-documented option (other turbulence models will certainly be implemented in the future). The spatial discretization of the governing equations is based on a vertex-centered finite element-finite volume formulation, where the finite volume cells are built on unstructured meshes. Second-order space accuracy is achieved through a piecewise-linear extrapolation based on the Monotonic Upwind Scheme for Conservation Law (MUSCL) procedure with a particular edge-based formulation. Both explicit and implicit approaches are available for the time integration. During implicit simulations, a linearized system is solved at each solver iteration using an approach derived from the Lower-Upper Symmetric Gauss-Seidel

(LU-SGS) introduced by Jameson.

At the beginning of this PhD, only the Euler and laminar Navier-Stokes version of the flow solver had been implemented. We have implemented the turbulence model and we have put a lot of effort into accelerating the convergence and improving robustness, which includes implementating an implicit time integration, appropriate CFL local (i.e. a CFL value for each vertex) dynamic CFL laws. All the new routines were parallelized using a shared-memory approach based on pthreads, using an in-house library that automatically deals with indirect addressing. After each significant modification of `Wolf`, test cases from the verification & validation (V&V) study (presented in Chapter 4) were run.

3.2 Modeling equations

The Reynolds Averaged Navier-Stokes (RANS) system relying to the Spalart-Allmaras model is composed of the compressible Navier-Stokes equations and the standard Spalart-Allmaras equation with no trip.

3.2.1 The compressible Navier-Stokes equations

The compressible Navier-Stokes equations for mass, momentum and energy conservation read:

$$\left\{\begin{array}{rcl} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) & = & 0\,, \\ \dfrac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) + \nabla p & = & \nabla \cdot (\mu \mathcal{T})\,, \\ \dfrac{\partial (\rho e)}{\partial t} + \nabla \cdot ((\rho e + p)\mathbf{u}) & = & \nabla \cdot (\mu \mathcal{T} \mathbf{u}) + \nabla \cdot (\lambda \nabla T)\,, \end{array}\right. \tag{3.1}$$

where ρ denotes the density (kg/m^3), $\mathbf{u}$ the velocity (m/s), e the total energy per mass ($m^2.s^{-2}$), p the pressure (N/m^2), T the temperature (K), μ the laminar dynamic viscosity ($kg/(m.s)$) and λ the laminar conductivity. $\mathcal{T}$ the laminar stress tensor:

$$\mathcal{T} = (\nabla \otimes \mathbf{u} + {}^t\nabla \otimes \mathbf{u}) - \frac{2}{3} \nabla . \mathbf{u}\, \mathbb{I}\,,$$

where (in 3D) $\mathbf{u} = (u, v, w)$ and

$$\nabla . \mathbf{u}\, \mathbb{I} = \begin{pmatrix} u_x + v_y + w_z & 0 & 0 \\ 0 & u_x + v_y + w_z & 0 \\ 0 & 0 & u_x + v_y + w_z \end{pmatrix},$$

where $u_x = \frac{\partial u}{\partial x}$, $u_y = \frac{\partial u}{\partial y}$, $u_z = \frac{\partial u}{\partial z}$ (idem for v and w).

The variation of nondimensionalized laminar dynamic viscosity and conductivity coefficients μ and λ

as a function of a dimensional temperature T is defined by Sutherland's law:

$$\mu \;=\; \mu_\infty \left(\frac{T}{T_\infty}\right)^{\frac{3}{2}} \left(\frac{T_\infty + \mathrm{Su}}{T + \mathrm{Su}}\right) \quad \text{and} \quad \lambda \;=\; \lambda_\infty \left(\frac{T}{T_\infty}\right)^{\frac{3}{2}} \left(\frac{T_\infty + \mathrm{Su}}{T + \mathrm{Su}}\right),$$

where $\mathrm{Su} = 110$ is the Sutherland constant and the index ∞ denotes reference quantities. The relation linking μ and λ is expressed from the Prandtl laminar number:

$$\mathrm{Pr} = \frac{\mu c_p}{\lambda} \quad \text{with} \quad \mathrm{Pr} = 0.72 \quad \text{for (dry) air},$$

where C_p is the specific heat at constant pressure.

3.2.2 Turbulence modeling

In accordance with the standard approach to turbulence modeling based upon the Boussinesq hypothesis [168], the total viscosity is divided into a laminar (or dynamic), μ, and a turbulent, μ_t, component. The dynamic viscosity is usually taken to be a function of the temperature, whereas μ_t is obtained using a turbulence model. Here we chose the Spalart-Allmaras one equation turbulence model [157] given by the following equation:

$$\frac{\partial \tilde{\nu}}{\partial t} + \mathbf{u}\cdot\nabla\tilde{\nu} = c_{b1}[1-f_{t2}]\tilde{S}\tilde{\nu} - \left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right]\left(\frac{\tilde{\nu}}{d}\right)^2 + \frac{1}{\sigma}\left[\nabla\cdot((\nu+\tilde{\nu})\nabla\tilde{\nu}) + c_{b2}\|\nabla\tilde{\nu}\|^2\right] + f_{t1}\Delta\mathbf{u}^2, \quad (3.2)$$

where $\tilde{\nu}$ is the turbulent kinematic viscosity and all the constants are defined below. In the standard model the trip term is being left out, *i.e.,* $f_{t1} = 0$. Moreover, some implementations also ignore the f_{t2} term as it is argued that if the trip is not included, then f_{t2} is not necessary [52]. In `Wolf`, this simplified version has been considered and we prefer to write it under the following form, which is more appropriate for its discretization with the finite element/finite volume method. Indeed, Equation (3.2) can be decomposed into the following terms:

$$\frac{\partial \rho\tilde{\nu}}{\partial t} + \underbrace{\mathbf{u}\cdot\nabla\rho\tilde{\nu}}_{convection} = \underbrace{c_{b1}\tilde{S}\rho\tilde{\nu}}_{production} \underbrace{- c_{w1}f_w\rho\left(\frac{\tilde{\nu}}{d}\right)^2}_{destruction} + \underbrace{\frac{\rho}{\sigma}\nabla\cdot((\nu+\tilde{\nu})\nabla\tilde{\nu})}_{dissipation} + \underbrace{\frac{c_{b2}\rho}{\sigma}\|\nabla\tilde{\nu}\|^2}_{diffusion}.$$

Notice that this is not a conservative model. If a conservative form of the Spalart-Allmaras is foreseen, we have to consider the variation proposed by Catris and Aupoix [36]. The turbulent eddy viscosity is computed from:

$$\mu_t = \rho\tilde{\nu}f_{v1},$$

where

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \quad \text{and} \quad \chi = \frac{\tilde{\nu}}{\nu} \quad \text{with} \quad \nu = \frac{\mu}{\rho}.$$

Additional definitions are given by the following equations:

$$f_{v2} = 1 - \frac{\chi}{1+\chi f_{v1}} \quad \text{and} \quad \tilde{S} = \Omega + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2} \quad \text{where} \quad \Omega = \|\nabla \times \mathbf{u}\| .$$

d is the distance to nearest wall which is computed for each vertex at the beginning of the simulation. The algorithm used to compute d is described in Appendix B. The set of closure constants for the model is given by

$$\sigma = \frac{2}{3} , \qquad c_{b1} = 0.1355 , \quad c_{b2} = 0.622 , \quad \kappa = 0.41 ,$$
$$c_{w1} = \frac{c_{b1}}{\kappa} + \frac{1+c_{b2}}{\sigma} , \quad c_{w2} = 0.3 , \qquad c_{w3} = 2 , \qquad c_{v1} = 7.1 .$$

Finally, the function f_w is computed as:

$$f_w = g\left(\frac{1+c_{w3}^6}{g^6+c_{w3}^6}\right)^{1/6} \quad \text{with} \quad g = r + c_{w2}\left(r^6 - r\right) \quad \text{and} \quad r = \min\left(\frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2}, 10\right) .$$

3.2.3 Vector form of the RANS system

We write the RANS system in the following (more compact) vector form:

$$W_t + F_1(W)_x + F_2(W)_y + F_3(W)_z = S_1(W)_x + S_2(W)_y + S_3(W)_z + Q(W) ,$$

where $S_i(W)_a = \frac{\partial S_i(W)}{\partial a}$ $(i = 1, 2, 3, a = x, y, z)$ (idem for F). W is the nondimensionalized conservative variables vector:

$$W = (\rho, \rho u, \rho v, \rho w, \rho E, \rho\tilde{\nu})^T .$$

$F(W) = (F_1(W), F_2(W), F_3(W))$ are the convective (Euler) flux functions:

$$\begin{aligned} F_1(W) &= (\rho u,\ \rho u^2 + p,\ \rho uv,\ \rho uw,\ u(\rho E + p), \rho u\tilde{\nu})^T , \\ F_2(W) &= (\rho v,\ \rho uv,\ \rho v^2 + p,\ \rho vw,\ v(\rho E + p), \rho v\tilde{\nu})^T , \\ F_3(W) &= (\rho w,\ \rho uw,\ \rho vw,\ \rho w^2 + p,\ w(\rho E + p), \rho w\tilde{\nu})^T . \end{aligned} \tag{3.3}$$

$S(W) = (S_1(W), S_2(W), S_3(W))$ are the laminar viscous fluxes:

$$\begin{aligned} S_1(W) &= (0,\ \tau_{xx},\ \tau_{xy},\ \tau_{xz},\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + \lambda\mathcal{T}_x)^T, \frac{\rho}{\sigma}(\nu + \tilde{\nu})\tilde{\nu}_x , \\ S_2(W) &= (0,\ \tau_{xy},\ \tau_{yy},\ \tau_{yz},\ u\tau_{xy} + v\tau_{yy} + w\tau_{yz} + \lambda\mathcal{T}_y)^T, \frac{\rho}{\sigma}(\nu + \tilde{\nu})\tilde{\nu}_y , \\ S_3(W) &= (0,\ \tau_{xz},\ \tau_{yz},\ \tau_{zz},\ u\tau_{xz} + v\tau_{yz} + w\tau_{zz} + \lambda\mathcal{T}_z)^T, \frac{\rho}{\sigma}(\nu + \tilde{\nu})\tilde{\nu}_z , \end{aligned} \tag{3.4}$$

where τ_{ij} are the components of laminar stress tensor defined by:

$$\tau_{ij} = \mu \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) - \frac{2}{3} \mu \frac{\partial v_k}{\partial x_k} \delta_{ij} \, .$$

where (v_i, v_j, v_k) are the three components of the velocity and δ_{ij} the Kroneker symbol.

$Q(W)$ are the source terms, i.e. the diffusion, production and destruction terms from the Spalart-Allmaras turbulence model:

$$Q(W) = (0,\ 0,\ 0,\ 0,\ 0,\ \frac{c_{b2}\rho}{\sigma} \|\nabla \tilde{\nu}\|^2 + \rho c_{b1} \tilde{S} \tilde{\nu} + c_{w1} f_w \rho \left(\frac{\tilde{\nu}}{d} \right)^2)^T \, . \tag{3.5}$$

Note that $Q = 0$ in the case of the laminar Navier-Stokes equations, unless additional source terms are added (to take into account gravity, for instance).

3.3 Spatial discretization

The spatial discretization of the fluid equations (3.1) and (3.2) is based on a vertex-centered finite element/finite volume formulation on unstructured meshes. It combines a HLLC upwind scheme [15] to compute the convective fluxes and the Galerkin centered method to evaluate the viscous terms. Second order space accuracy is achieved through a piecewise linear extrapolation based on the Monotonic Upwind Scheme for Conservation Law (MUSCL) procedure [86] which uses a particular edge-based formulation with upwind elements. A specific slope limiter is employed to damp or eliminate spurious oscillations that may occur in the vicinity of discontinuities [44] (see Section 3.3.3).

3.3.1 Finite Volume discretization

Let $\mathcal{H}$ be a mesh of domain Ω, the vertex-centered finite volume formulation consists in associating with each vertex P_i of the mesh a control volume or finite volume cell, denoted C_i. Discretized domain Ω_h (see Figure 3.3) can be written as the union of the elements or the union of the finite volume cells:

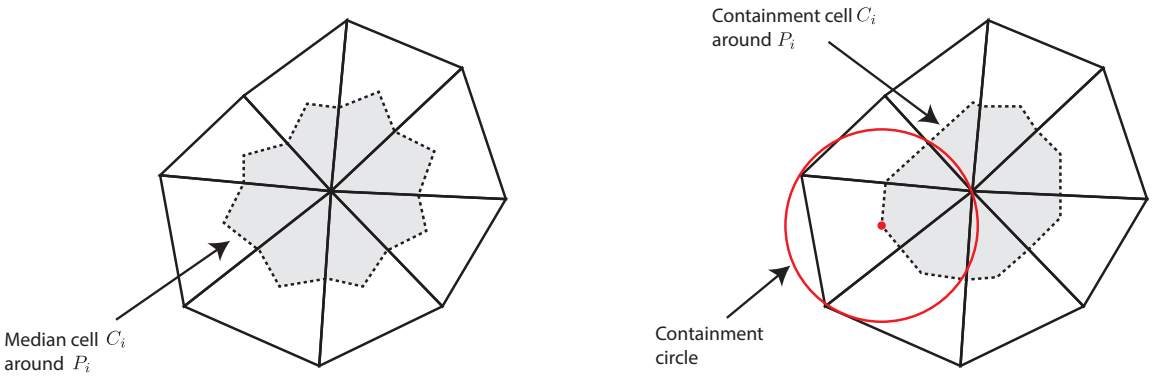
$$\Omega_h = \bigcup_{i=1}^{N_K} K_i = \bigcup_{i=1}^{N_V} C_i \, ,$$

where N_K is the number of elements and N_V the number of vertices.

Note that the dual mesh (composed of cells) il built in a preprocessing step. Consequently, only a simplicial mesh is needed in the input. Several choices are possible to build finite volume cells. In this work, two methods were considered: median cells and containment cells.

Median cells. In 2D, this standard method consists in building cells bounded by segments of medians (so-called median cells), see Figure 3.1a. In 3D, each tetrahedron is split into four hexahedra (one associated to each one of its four vertices). The eight vertices of the hexadron associated to a point P_i are given by: (i) M_i, M_j, M_k, the middle points of the three edges incident to P_i, (ii) Gf_i, Gf_j, Gf_k, the gravity centers of the 3 faces containing P_i, (iii) G, the gravity center of the tetra, and (iv) the vertex P_i considered. The cell C_i associated to vertex P_i is the union of all hexahedra of the tetrahedra surrounding P_i.

These cells enjoy a rather good robustness to distorted meshes, but they are not well-suited to stretched meshes [165]. Viscous simulations require highly stretched boundary layer meshes, which is why the second method is used in this context. For instance, we show in the sequel that we are unable to converge the simulation of the 2D turbulent bump using median cells.



(a) Median cell.

(b) Containment cell.

Figure 3.1: Example of median and containment cells in 2D.

Containment cells. This method, introduced in 2D by Barth [13] and generalized to 3D by Dervieux [64], is well-suited to discretize accurately the flow equations on highly anisotropic quasi-structured meshes (boundary layer meshes). In 3D, it consists in subdividing each tetrahedron into four hexahedra cell around each vertex, see Figure 3.1a. The hexahedron cell vertices associated with vertex P_i are (i) the middle of the three edges issued from P_i, (ii) the containment circle center of the three faces containing P_i, (iii) the containment sphere center of the tetrahedron and (iv) the vertex P_i considered. The containment sphere cells of vertex P_i is the union of all its hexahedra cells. The containment sphere center corresponds to the sphere circumcenter if it falls inside the element.

Comparison of median and containment cells. Figure 3.3 compares the two approaches on a 2D mesh with a quasi-structured region. The same comparison in 3D is shown in Figure 3.4. These examples illustrate how the faces of containment cells are aligned with the flow direction in the presence

of a boundary layer mesh, in contrast to median cells. This is why we prefer containment cells for highly stretched quasi-structured meshes.

We now show the importance of this choice of cells, by running the same simulation using median and containment cells and comparing the results. We take the example of the 2D bump which is part of the verification study presented in Chapter 4 (we refer to Section 4.2.2 for the presentation of the case). As shown in Figure 3.2, we fail to converge the simulation using median cells, despite our efforts in trying to find a set of parameters for which it works. On the other hand, an accurate solution is easily obtained using containment cells.

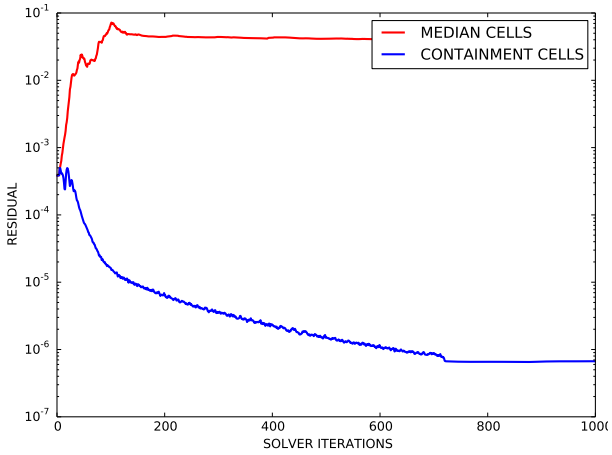


Figure 3.2: 2D turbulent bump: comparison of the residual convergence using the median and the containment cells. For this case, we were unable to converge the computation on median cells (different sets of input parameters have been tried).

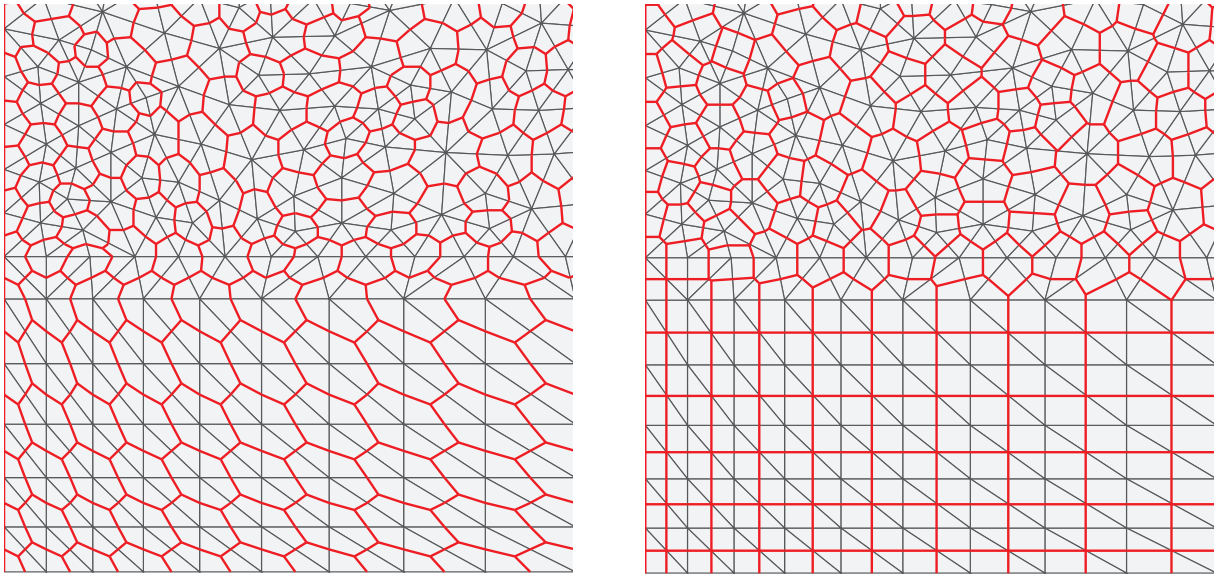
Based on a finite volume formulation, the Reynolds Averaged Navier-Stokes equations are integrated on each finite volume cell C_i (using the Green formula):

$$|C_i|\frac{dW_i}{dt} + \mathbf{F}_i = \mathbf{S}_i + \mathbf{Q}_i\,, \tag{3.6}$$

where W_i is the mean value of the solution W on cell C_i, $\mathbf{F}_i$, $\mathbf{S}_i$ and $\mathbf{Q}_i$ are respectively the numerical convective, viscous and source flux terms:

$$\mathbf{F}_i = \int_{\partial C_i} F(W_i)\cdot\mathbf{n}_i \mathrm{d}\gamma\,, \qquad \mathbf{S}_i = \int_{\partial C_i} S(W_i)\cdot\mathbf{n}_i\,\mathrm{d}\gamma\,, \qquad \mathbf{Q}_i = \int_{C_i} Q(W_i)\,\mathrm{d}\mathbf{x}\,,$$

where $\mathbf{n}_i$ is the outer normal to the finite volume cell surface ∂C_i as depicted in Figure 3.5, F (see Relation 3.3) and S (see Relation 3.4) are respectively the convective and viscous flux functions and Q (see Relation 3.5) the source flux function.



(a) Median cells. (b) Containment cells.

Figure 3.3: Median and containment dual meshes (in red) constructed on a mesh containing a quasi-structured region with a transition to a fully-unstructured one.

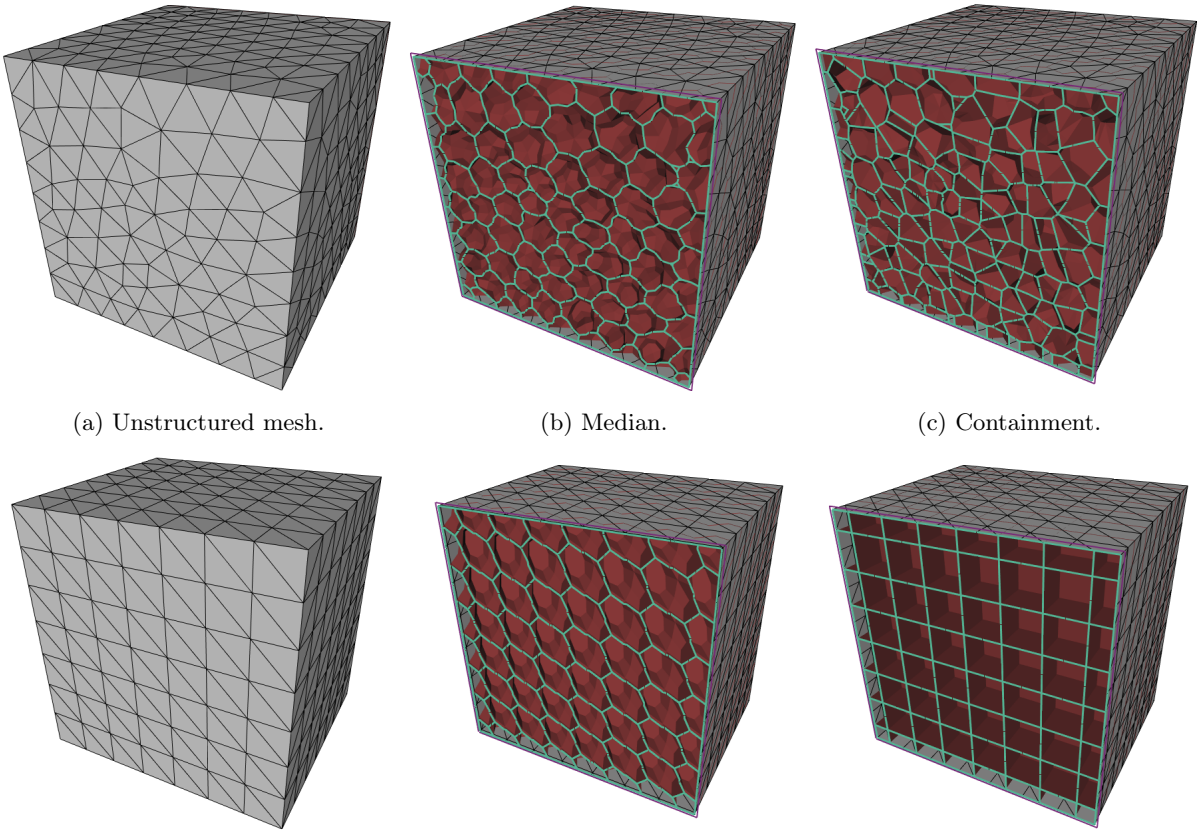
Discretization of the convective terms. The integration of convective fluxes $\mathbf{F}$ of Equation (3.6) is done by decomposing the cell boundary into many facets ∂C_{ij}:

$$\mathbf{F}_i = \sum_{P_j \in \mathcal{V}(P_i)} F|_{\partial C_{ij}} \cdot \int_{\partial C_{ij}} \mathbf{n}_i \, \mathrm{d}\gamma \,,$$

where $\mathcal{V}(P_i)$ is the set of all neighboring vertices linked by an edge to P_i and $F|_{\partial C_{ij}}$ represents the constant value of $F(W)$ at interface ∂C_{ij}. The flow is calculated using a numerical flux function, denoted by Φ_{ij}:

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

where $\mathbf{n}_{ij} = \int_{\partial C_{ij}} \mathbf{n}_i \, \mathrm{d}\gamma$. The numerical flux function approximates the hyperbolic terms on the common boundary ∂C_{ij}. We notice that the computation of the convective fluxes is performed mono-dimensionally in the direction normal to the boundary of the finite volume cell. Therefore, the numerical calculation of the flux function Φ_{ij} at the interface ∂C_{ij} is achieved by the resolution of a one-dimensional Riemann problem in the direction of the normal $\mathbf{n}_{ij}$ by means of an approximate Riemann solver. In this work, the HLLC approximate Riemann solver is used for the mean flow - more details can be found in [15] - and linear advection with upwinding is used for the turbulent variable convection.



(a) Unstructured mesh. (b) Median. (c) Containment.

(d) Quasi-structured mesh. (e) Median. (f) Containment.

Figure 3.4: 3D median and containment dual meshes constructed on a mesh on an unstructured mesh (top) and a quasi-structured mesh (bottom).

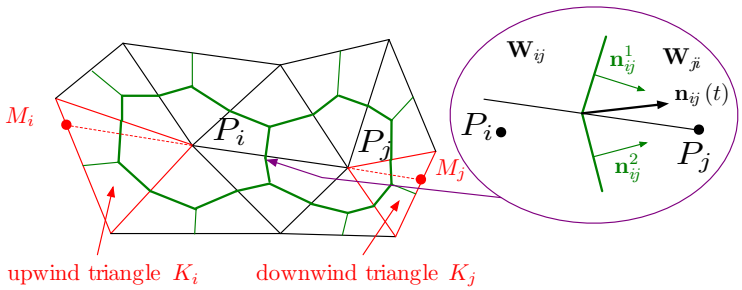


Figure 3.5: Illustration of finite volume cell construction in 2D: two neighbouring cells C_i and C_j and the upwind triangles K_i and K_j associated to edge P_iP_j.

3.3.2 HLLC approximate Riemann solver

The idea of the HLLC flow solver is to consider locally a simplified Riemann problem with two intermediate states depending on the local left and right states. The simplified solution to the Riemann problem consists of a contact wave with a velocity S_M and two acoustic waves, which may be either shocks or expansion fans. The acoustic waves have the smallest and the largest velocities (S_I and S_J, respectively) of all

the waves present in the exact solution. If $S_I > 0$ then the flow is supersonic from left to right and the upwind flux is simply defined from $F(W_l)$ where W_l is the state to the left of the discontinuity. Similarly, if $S_J < 0$ then the flow is supersonic from right to left and the flux is defined from $F(W_j)$ where W_j is the state to the right of the discontinuity. In the more difficult subsonic case when $S_I < 0 < S_J$ we have to calculate $F(W_l^*)$ or $F(W_j^*)$. Consequently, the HLLC flux is given by:

$$\Phi_{ij}^{hllc}(W_l, W_j, \mathbf{n}_{ij}) = \begin{cases} F(W_i)\cdot\mathbf{n}_{ij} & \text{if } S_I > 0 \\ F(W_i^*)\cdot\mathbf{n}_{ij} & \text{if } S_I \le 0 < S_M \\ F(W_j^*)\cdot\mathbf{n}_{ij} & \text{if } S_M \le 0 \le S_J \\ F(W_j)\cdot\mathbf{n}_{ij} & \text{if } S_J < 0 \end{cases} .$$

W_i^* and W_j^* are evaluated as follows. We denote by $\eta = \mathbf{u}\cdot\mathbf{n}$. Assuming that $\eta^* = \eta_i^* = \eta_j^* = S_M$, the following evaluations are proposed [15] (the subscripts $_i$ and $_j$ are omitted for clarity):

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

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

$$S_M = \frac{\rho_j\eta_j(S_J - \eta_j) - \rho_i\eta_i(S_I - \eta_i) + p_i - p_j}{\rho_j(S_J - \eta_j) - \rho_i(S_I - \eta_i)}\,,$$

and the acoustic wave speeds based on the Roe average:

$$S_I = \min(\eta_i - c_i, \tilde{\eta} - \tilde{c}) \quad \text{and} \quad S_J = \max(\eta_j + c_j, \tilde{\eta} + \tilde{c})\,.$$

With such waves velocities, the approximate HLLC Riemann solver has the following properties. It automatically (i) satisfies the entropy inequality, (ii) resolves isolated contacts exactly, (iii) resolves isolated shocks exactly, and (iv) preserves positivity.

Linear convection. The turbulent variable $\tilde{\nu}$ is linearly convected:

$$\Phi_{ij}^{\rho\tilde{\nu}}(W_i, W_j, \mathbf{n}_{ij}) = \begin{cases} \eta\,\rho\tilde{\nu}_i & \text{if} \quad \eta > 0 \\ \eta\,\rho\tilde{\nu}_j & \text{otherwise} \end{cases} \quad \text{where} \quad \eta = \frac{1}{2}\left(\mathbf{u}_i\cdot\mathbf{n}_{ij} + \mathbf{u}_j\cdot\mathbf{n}_{ij}\right) .$$

3.3.3 2nd-order accurate version

The MUSCL type reconstruction method has been designed to increase the order of accuracy of the scheme [86]. The idea is to use extrapolated values W_{ij} and W_{ji} instead of W_i and W_j at the interface

∂C_{ij} to evaluate the flux. The numerical flux becomes:

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

where W_{ij} and W_{ji} are linearly extrapolated as:

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

In contrast to the original MUSCL approach, the approximate "slopes" $(\nabla W)_{ij}$ and $(\nabla W)_{ji}$ are defined for any edge and obtained using a combination of centered, upwind and nodal gradients.

The centered gradient, which is related to edge P_iP_j, is implicitly defined along edge P_iP_j by the relation:

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

Upwind and downwind gradients, which are also related to edge P_iP_j, are computed according to the definition of upwind and downwind tetrahedra of edge P_iP_j. These tetrahedra are respectively denoted K_{ij} and K_{ji}. K_{ij} (resp. K_{ji}) is the unique tetrahedron of the ball of P_i (resp. P_j) the opposite face of which is crossed by the line defined by the edge P_iP_j, see Figure 3.6.

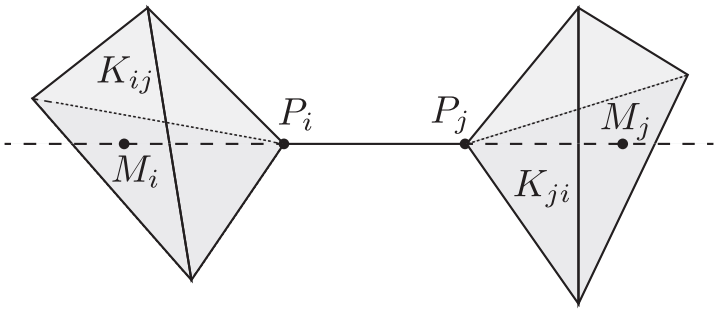


Figure 3.6: Downwind K_{ij} and upwind K_{ji} tetrahedra associated to edge P_iP_j.

Upwind and downwind gradients are then defined for vertices P_i and P_j as:

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

where $(\nabla W)|_K = \sum_{P\in K} W_P \nabla \phi_P|_K$ is the P_1-Galerkin gradient on tetrahedron K. Parametrized nodal gradients are built by introducing the β-scheme:

$$\begin{aligned}
(\nabla W)_{ij} \cdot \overrightarrow{P_iP_j} &= (1-\beta)(\nabla W)^C_{ij} \cdot \overrightarrow{P_iP_j} + \beta\,(\nabla W)^U_{ij} \cdot \overrightarrow{P_iP_j} \\
(\nabla W)_{ji} \cdot \overrightarrow{P_iP_j} &= (1-\beta)(\nabla W)^C_{ij} \cdot \overrightarrow{P_iP_j} + \beta\,(\nabla W)^D_{ij} \cdot \overrightarrow{P_iP_j} \,,
\end{aligned}$$

where $\beta \in [0,1]$ is a parameter controlling the amount of upwinding. For instance, the scheme is centered

for $\beta = 0$ and fully upwind for $\beta = 1$.

Fourth-order numerical dissipation: V4-scheme. The most accurate β-scheme is obtained for $\beta = 1/3$. Indeed, it can be demonstrated that this scheme is third-order for the two-dimensional linear advection on structured triangular meshes. On unstructured meshes, a second-order scheme with a fourth-order numerical dissipation is obtained. These high-order gradients are given by:

$$\begin{aligned}(\nabla W)_{ij}^{V4} \cdot \overrightarrow{P_iP_j} &= \frac{2}{3}(\nabla W)_{ij}^{C} \cdot \overrightarrow{P_iP_j} + \frac{1}{3}(\nabla W)_{ij}^{U} \cdot \overrightarrow{P_iP_j} \\ (\nabla W)_{ji}^{V4} \cdot \overrightarrow{P_iP_j} &= \frac{2}{3}(\nabla W)_{ji}^{C} \cdot \overrightarrow{P_iP_j} + \frac{1}{3}(\nabla W)_{ij}^{D} \cdot \overrightarrow{P_iP_j}\,.\end{aligned}$$

Sixth-order numerical dissipation: V6-scheme. An even less dissipative scheme has been proposed [45]. It is a more complex linear combination of gradients using centered, upwind and nodal P_1-Galerkin gradients. The nodal P_1-Galerkin gradient of P_i is related to cell C_i and is computed by averaging the gradients of all the tetrahedra containing vertex P_i:

$$(\nabla W)_{P_i} = \frac{1}{4\,|C_i|}\sum_{K\in C_i} |K|(\nabla W)|_K\,.$$

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

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

where $(\nabla W)_{M_{i,j}}$ is the gradient at the points $M_{i,j}$ intersecting the line defined by P_iP_j and upwind-downwind tetrahedra. These gradients are computed by linear interpolation of the nodal gradients of faces containing M_i and M_j, see Figure 3.6.

Limiter function. The aforementioned MUSCL schemes are not monotone and can be a source of spurious oscillations. These oscillations can affect the accuracy of the final solution or simply end the computation because (for instance) of negative pressures. A widely used technique for addressing this issue is to guarantee the TVD property of the scheme -first established in the 1D case by Harten et al. [69]-, which ensures that the extrapolated values W_{ij} and W_{ji} are not erronate. To guarantee the TVD property, limiting functions are coupled with the previous high-order gradient evaluations. The gradient is substituted by a limited gradient denoted $(\nabla W)_{ij}^{lim}$. The choice of the limiting function is crucial as it directly affects the convergence of the simulation. In this work, we use the three-entries

limiter introduced by Dervieux which is a generalization of the Superbee limiter [44]:

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

and we use:

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

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

3.3.4 Discretization of the viscous terms

In `Wolf`, we discretize the viscous terms using the finite element method (FEM):

$$
\mathbf{S}_i \;=\; \sum_{P_j \in \mathcal{V}(P_i)} \int_{\partial C_{ij}} S(W_i) \cdot \mathbf{n}\, \mathrm{d}\gamma + BT
$$

where ∂C_{ij} is the common interface between cells C_i and C_j, and BT stands for the boundary terms. Let ϕ_i be the $\mathbb{P}_1$ finite element basis function associated with vertex P_i, we have: $\int_K \nabla\phi_i \,\mathrm{d}\mathbf{x} = -\int_{\partial C_i \cap K} \mathbf{n}\,\mathrm{d}\gamma$ and if we assume that $S(W_i)$ (which comes from a gradient) is constant on element K, then we obtain:

$$
\sum_{P_j \in \mathcal{V}(P_i)} \int_{\partial C_{ij}} S(W_i) \cdot \mathbf{n}\, \mathrm{d}\gamma \;=\; -\sum_{K \ni P_i} \int_K S(W_i)|_K \cdot \nabla\phi_i \,\mathrm{d}\mathbf{x}\,.,
$$

The effective computation of the previous integral then leads to the computation of integrals of the following form:

$$
\int_K \nabla\phi_i \,\nabla\phi_j \,\mathrm{d}\mathbf{x} \;=\; |K|\, \nabla\phi_i|_K \,\nabla\phi_j|_K.
$$

In this expression, $\nabla\phi_i|_K$ is the constant gradient of basis function ϕ_i associated with vertex P_i. This discretization is justified because the characteristic times associated with the diffusive terms are larger than the characteristic times associated with the hyperbolic (convective) terms. We now apply the FEM formulation to all convected variables that are averaged on the element and we easily verify that the components of the (Cauchy) stress tensor $S(W)$ are constant on each element K. For instance, the term $u\,\tau_{xy}$ of the (Cauchy) stress tensor reads:

$$
(u\,\tau_{xy})|_K = u|_K\, \mu|_K \sum_{\mathbf{p}_i \in K} \left(u_i \frac{\partial\phi_i|_K}{\partial y} + v_i \frac{\partial\phi_i|_K}{\partial x} \right)\,,
$$

where $u|_K$ is the averaged value of u on the mesh element K (idem for μ). The other terms are computed analogously.

Discretization of the Spalart-Allmaras dissipation term. The Spalart-Allmaras dissipation term is also discretized with the FEM:

$$\Phi^{SA}_{visc,K}(W_i, W_j, W_k, W_l) = |K| \, \frac{1}{\sigma}\rho_i\Big((\nu|_K + \tilde{\nu}|_K) \, \nabla\tilde{\nu}|_K \cdot \nabla\phi_i|_K\Big) \, .$$

Discretization of the source terms. Finally, the Spalart-Allmaras source terms (diffusion, production and destruction) are discretized by simple integration on each vertex cell:

$$\mathbf{Q}_i = |C_i| Q(W_i) \, ,$$

where $|C_i|$ is the volume of the vertex cell.

3.4 Boundary conditions

System (3.1) is closed using a set of appropriate boundary conditions. For the flow simulations presented in this thesis, three boundary conditions were used. Slip boundary conditions are imposed for bodies when the flow is considered inviscid or for symmetry. For viscous flow, no slip boundary conditions are considered for bodies. And finally, we used Steger-Warming flux to set up free-stream (external flow) conditions.

3.4.1 Free-stream condition

This condition imposes a free-stream uniform flow from the infinite. It is applied when we have a boundary Γ_∞ for which the infinite constant state W_∞ is prescribed:

$$W_\infty = (\rho_\infty, \, (\rho\mathbf{u})_\infty, \, (\rho E)_\infty, (\rho\tilde{\nu})_{farfield})^T \quad \text{and} \quad \tilde{\nu}_{farfield} \in [3\nu_\infty \, , \, 5\nu_\infty] \, .$$

This state enables upwind fluxes at the infinite to be computed. The considered boundary fluxes are built from a decomposition following the characteristics' values. We consider the Steger-Warming flux which is completely upwind on solution W_i:

$$\Phi_{\infty,Fac}(W_i) = A^+(W_i, \mathbf{n_{Fac}})W_i + A^-(W_i, \mathbf{n_{Fac}})W_\infty \quad \text{where} \quad A^+ = \frac{|A| + A}{2} \quad \text{and} \quad A^- = \frac{|A| - A}{2} \, ,$$

where Fac are boundary faces with normals $\mathbf{n}_{Fac}$.

3.4.2 Slip condition

For this boundary condition, we impose weakly $\mathbf{u} \cdot \mathbf{n} = 0$, which is done by imposing the following boundary flux:

$$\Phi_{slip,Fac}(W_i) = \sum_{Fac \ni P_i} \int_{\partial C_i \cap Fac} F_{slip}(W_i) \cdot \mathbf{n}_{Fac} \, \mathrm{d}\gamma \quad \text{with} \quad F_{slip}(W_i) \cdot \mathbf{n}_{Fac} = (0,\ p_i \, \mathbf{n}_{Fac},\ 0)^t \, .$$

According to [9], the slip boundary conditions for the turbulent equations are different depending on wether a wall or a symmetry plane is considered:

$$\tilde{\nu}_{slipwall} = 0 \quad \text{and} \quad \frac{\partial \tilde{\nu}_{symmetry}}{\partial \mathbf{n}} = 0 \, .$$

3.4.3 No slip condition

Adiabatic conditions are considered, therefore only a null velocity is strongly imposed for this boundary condition: $\mathbf{u} = 0$. The turbulent variable is also strongly imposed to zero: $\tilde{\nu}_{noslip} = 0$.

3.5 Time integration

Once the equations have been discretized in space, a set of ordinary differential equations in time is obtained. There are two ways for integrating this set of ODE in time: either using an explicit or an implicit method. Although all the simulations presented in this thesis were run using an implicit time integration, this section describes both approaches.

3.5.1 Explicit time integration

For an explicit time discretization, the semi-discretized RANS system becomes:

$$\frac{|C_i|}{\delta t_i^n} \delta W_i \ = \ - \mathbf{F}_i^n + \mathbf{S}_i^n + \mathbf{Q}_i^n \, ,$$

where $\delta W_i = W_i^{n+1} - W_i^n$. We recall that $\mathbf{F}_1$, $\mathbf{S}_i$ and $\mathbf{Q}_i$ are respectively the convective, viscous and source numerical flux terms defined in Section 3.3. Explicit time stepping algorithm are used by means of a strong-stability-preserving (SSP) Runge-Kutta scheme [156, 158].

Time step computation. The local time step δt is computed at each vertex:

$$\delta t = \mathrm{CFL} \, \frac{h^2}{h \left(c + \|\mathbf{u}\| \right) + 2 \, \dfrac{\gamma}{\rho} \left(\dfrac{\mu}{\mathrm{Pr}} + \dfrac{\mu_t}{\mathrm{Pr}_t} \right)}$$

where CFL is the Courant-Friedrichs-Lewy condition [43], which links links the smallest altitude of the mesh to the maximal time step. Pr= 0.72 and $\mathrm{Pr}_t = 0.9$ are the Prandtl and the turbulent Prandtl constants, μ and μ_t are the (molecular dynamic) viscosity and the turbulent (molecular dynamic) viscosity, $\mathbf{u}$ is the velocity, c is the speed of sound, and h is the smallest height of the elements surrounding the vertex considered.

3.5.2 Implicit time integration

For an implicit time integration, the discretization of the partial derivative in time is:

$$\frac{|C_i|}{\delta t_i^n}\delta W_i \;=\; -\mathbf{F}_i^{n+1} + \mathbf{S}_i^{n+1} + \mathbf{Q}_i^{n+1}\,, \tag{3.7}$$

where $\delta W_i = W_i^{n+1} - W_i^n$, which, after linearization of the RHS, becomes:

$$\left(\frac{|C_i|}{\delta t_i^n} I_d + \frac{\partial \mathbf{F}_i^n}{\partial W_i} - \frac{\partial \mathbf{S}_i^n}{\partial W_i} - \frac{\partial \mathbf{Q}_i^n}{\partial W_i}\right)\delta W_i + \sum_{j\in\mathcal{V}(i)}\left(\frac{\partial \mathbf{F}_i^n}{\partial W_j} - \frac{\partial \mathbf{S}_i^n}{\partial W_j} - \frac{\partial \mathbf{Q}_i^n}{\partial W_j}\right)\delta W_j = -\mathbf{F}_i^n + \mathbf{S}_i^n + \mathbf{Q}_i^n\,.$$

where $j \in \mathcal{V}(i)$ is the set of vertices connected to vertex i by an edge. The first term of the LHS contributes to the diagonal of the matrix and the second term of the LHS (*i.e.,* the sum) contributes to extra-diagonal terms on line i of the matrix. We now describe each term of the matrix.

Inviscid flux Jacobian. We recall that $\mathbf{F}_i^{n+1} = \sum_{j\in\mathcal{V}(i)} \Phi_{ij}^{hllc}(W_i^{n+1}, W_j^{n+1}, \mathbf{n}_{ij})$. The linearization of the convective flux term reads:

$$\begin{aligned}\Phi_{ij}^{hllc}(W_i^{n+1}, W_j^{n+1}, \mathbf{n}_{ij}) \;&=\; \Phi_{ij}^{hllc}(W_i^n, W_j^n, \mathbf{n}_{ij}) \\ &+\; \left.\frac{\partial \Phi_{ij}^{hllc}(W_i^n, W_j^n, \mathbf{n}_{ij})}{\partial W_i}\delta W_i\right\}\text{(A)} \\ &+\; \left.\frac{\partial \Phi_{ij}^{hllc}(W_i^n, W_j^n, \mathbf{n}_{ij})}{\partial W_j}\delta W_j\right\}\text{(B)}\,.\end{aligned}$$

Term (A) contributes to matrix diagonal $D(i,i)$ and Term (B) contributes to matrix upper part $U(i,j)$ (here we assume that $i<j$). As $\Phi_{ji}^{hllc} = -\Phi_{ij}^{hllc}$, minus Term (B) contributes to matrix diagonal $D(j,j)$ and minus Term (A) contributes to matrix lower part $L(j,i)$.

Viscous flux Jacobian. Let $K = (P_i, P_j, P_k, P_l)$, we now linearize the viscous flux terms $\mathbf{S}_i^n$:

$$\mathbf{S}_i^n = -\sum_{K\ni P_i} \Phi_{ij,K}^{visc}(W_i^{n+1}, W_j^{n+1}, W_k^{n+1}, W_l^{n+1})$$

. It reads:

$$\begin{aligned}\Phi^{visc}_{ij,K}(W_i^{n+1},W_j^{n+1},W_k^{n+1},W_l^{n+1}) &= \Phi_{visc,K}(W_i^n,W_j^n,W_k^n,W_l^n)\\ &+ \left.\frac{\partial\Phi_{visc,K}}{\partial W_i}(W_i^n,W_j^n,W_k^n,W_l^n)\delta W_i\right\}\text{(A)}\\ &+ \left.\frac{\partial\Phi_{visc,K}}{\partial W_j}(W_i^n,W_j^n,W_k^n,W_l^n)\delta W_j\right\}\text{(B)}\\ &+ \left.\frac{\partial\Phi_{visc,K}}{\partial W_k}(W_i^n,W_j^n,W_k^n,W_l^n)\delta W_k\right\}\text{(C)}\\ &+ \left.\frac{\partial\Phi_{visc,K}}{\partial W_l}(W_i^n,W_j^n,W_k^n,W_l^n)\delta W_l\right\}\text{(D)}\,.\end{aligned}$$

Term (A) contributes to the matrix diagonal, while Terms (B), (C), (D) contribute to the matrix extra-diagonal.

Boundary conditions Jacobian. The linearization of the boundary conditions term reads:

$$\Phi_{bc,Fac}(W_i^{n+1},\mathbf{n}_{Fac}) = \Phi_{bc,Fac}(W_i^n,\mathbf{n}_{Fac}) + \frac{\Phi_{bc,Fac}}{\partial W_i}(W_i^n,\mathbf{n}_{Fac})\delta W_i\,,$$

which contributes to the matrix diagonal.

Source terms Jacobians. The source terms are the sum of production ($\mathcal{P}$), destruction ($\mathcal{D}$) and diffusion terms ($\mathcal{V}$), which only contribute to the diagonal:

$$\frac{\partial\mathbf{Q}_i^n}{\partial\tilde{\nu}_i} = \frac{\partial\mathcal{P}_i^n}{\partial\tilde{\nu}_i} + \frac{\partial\mathcal{D}_i^n}{\partial\tilde{\nu}_i} + \frac{\partial\mathcal{V}_i^n}{\partial\tilde{\nu}_i}\,.$$

We chose a full linearization of these terms, which we detail in Appendix C.

3.6 Newton's method

The linearized system obtained in the previous Section is written in vector form:

$$\mathbf{A}^n\,\delta W^n = \mathbf{R}^n \tag{3.8}$$

$$\text{where}\quad \mathbf{R}^n = -\mathbf{F}^n + \mathbf{S}^n + \mathbf{Q}^n\,,\quad \mathbf{A}^n = \frac{|C|}{\delta t^n}\mathbf{I} - \frac{\partial\mathbf{R}^n}{\partial\mathbf{W}}\,,\quad \text{and}\quad \delta\mathbf{W}^n = \mathbf{W}^{n+1} - \mathbf{W}^n\,.$$

This linear system is solved at each flow solver iteration using an iterative Newton method. In practice, we ask the user to provide a maximal number k_{max} of iterations of the Newton method and a targetted order of magnitude by which the residual of the system must be decreased. The iteration is stopped when this targetted residual is reached.

To solve the non-linear system, we follow the approach based on Lower-Upper Symmetric Gauss-Seidel

(LU-SGS) implicit solver initially introduced by Jameson [79] and fully developed by Sharov et al. and Luo et al. [110, 111, 154, 153]. The Newton method can be the LU-SGS approximate factorization or the SGS relaxation or the GMRES method with LUSGS or SGS as a preconditioner. The LU-SGS and SGS are very attractive because they use an edge-based data structure which can be efficiently parallelized with p-threads [6, 153]. From our experience, we have made the following - crucial - choices to solve the compressible Navier-Stokes equations.

Converging the Newton method is important for the global convergence of the Navier-Stokes non-linear problem. Hence, an iterative method, such as SGS or GMRES+LUSGS or GMRES+SGS[1], is required. Usually, the Newton method iterates until the residual of the linear system is reduced by two orders of magnitude (i.e. 0.01).

The choice of the renumbering also impacts strongly the convergence of the non-linear system. While Hilbert-type (space filling curve) renumbering is very efficient for cache misses and memory contention [6, 153], Breadth-first search renumbering proves to be more effective for the convergence of the implicit method and the overall efficiency. For more details about renumbering methods, we refer to Chapter 2.

Luo et al. [110, 111, 153] proposed using a simplified flux function - a Rusanov approximate Riemann solver for the convective terms and the operator spectral radius for the viscous terms - to compute Jacobians while keeping the complex flux function for the right-hand side term. However, we observed that this modification slows down the convergence of the whole process. We found it very advantageous to fully differentiate the HLLC approximate Riemann solver [15], the FEM viscous terms and the Spalart-Allmaras source terms [4] as presented in the previous section.

To achieve high efficiency, automation, and robustness in the resolution of the non-linear system of algebraic equations to steady-state, it is mandatory to have a clever strategy to specify the time step. This is done by coupling a local under-relaxation coefficient, and local CFL.

In this work, we have considered the symmetric Gauss-Seidel (SGS) relaxation. This linear system can be re-written:

$$(\mathbf{D}+\mathbf{L})\mathbf{D}^{-1}(\mathbf{D}+\mathbf{U})\,\delta\mathbf{W}^n = \mathbf{R}^n + (\mathbf{L}\mathbf{D}^{-1}\mathbf{U})\,\delta\mathbf{W}^n$$

The following approximate system is used:

$$(\mathbf{D}+\mathbf{L})\mathbf{D}^{-1}(\mathbf{D}+\mathbf{U})\,\delta\mathbf{W}^n = \mathbf{R}^n\,.$$

Matrix $(\mathbf{D}+\mathbf{L})\mathbf{D}^{-1}(\mathbf{D}+\mathbf{U})$ can inverted in two sweeps which correspond to the LU-SGS approximate factorization:

$$\begin{aligned} \text{Forward sweep:} \quad & (\mathbf{D}+\mathbf{L})\,\delta\mathbf{W}^* &&= \mathbf{R} \\ \text{Backward sweep:} \quad & (\mathbf{D}+\mathbf{U})\,\delta\mathbf{W} &&= \mathbf{D}\,\delta\mathbf{W}^*\,. \end{aligned}$$

[1] In comparison, the LU-SGS method works well for the compressible Euler equation.

These sweeps are written point-wise:

$$\begin{aligned} \delta W_i^* &= D_{ii}^{-1}\Big(R_i - \sum_{j\in\mathcal{L}(i)} L_{ij}\delta W_j^*\Big) \\ \delta W_i &= \delta W_i^* - D_{ii}^{-1}\sum_{j\in\mathcal{U}(i)} U_{ij}\delta W_j^* \,. \end{aligned}$$

where $\mathcal{L}(i)$ (resp. $\mathcal{U}(i)$) is the set of vertices with an index lower (resp. upper) than i. The lower and upper parts can be stored or not (i.e., matrix-free) as a choice between efficiency or memory requirements.

In the SGS relaxation, we first zero the unknown: $\delta W^0 = 0$. Then, k_{max} sub-iterations are made using forward and backward sweeps:

$$\begin{aligned} (\mathbf{D}+\mathbf{L})\,\delta\mathbf{W}^{k+1/2} &= \mathbf{R} - \mathbf{U}\,\delta\mathbf{W}^{k} \\ (\mathbf{D}+\mathbf{U})\,\delta\mathbf{W}^{k+1} &= \mathbf{R} - \mathbf{L}\,\delta\mathbf{W}^{k+1/2} \,. \end{aligned}$$

or rewritten point-wise:

$$\begin{aligned} \delta W_i^{k+1/2} &= D_{ii}^{-1}\Big(R_i - \sum_{j\in\mathcal{L}(i)} L_{ij}\delta W_j^{k+1/2} - \sum_{j\in\mathcal{U}(i)} U_{ij}\delta W_j^{k}\Big) \\ \delta W_i^{k+1} &= D_{ii}^{-1}\Big(R_i - \sum_{j\in\mathcal{U}(i)} U_{ij}\delta W_j^{k+1} - \sum_{j\in\mathcal{L}(i)} L_{ij}\delta W_j^{k+1/2}\Big) \,. \end{aligned}$$

For one sub-iteration, the SGS method is equivalent to the LU-SGS method.

Note that the convergence of the Newton method is crucial for the global convergence of the simulation, as shown in Figure 3.7 through the example of a subsonic flow computed over a 3D NACA 0012 airfoil (for more details about the simulation, see Section 5.4.1). It presents the residual convergence of the simulation in terms of solver iterations, for different prescribed maximal numbers of SGS iterations. It reveals that no less than 25 SGS iterations are necessary for the simulation to successfully converge.

In some cases, iterating the Newton method does not help to decrease the residual of the linear system. This can be due to the stiffness of the problem or simply because the linear system has already converged by its maximum order of magnitude, in which case the targetted residual was not set properly. In order to avoid costly SGS iterations which will not impact the final solution, we stop iterating when stagnations of the residual of the linear system are detected.

Let Res_i be the residual of the linear system at the current SGS iteration i and Res_{i-1} the residual

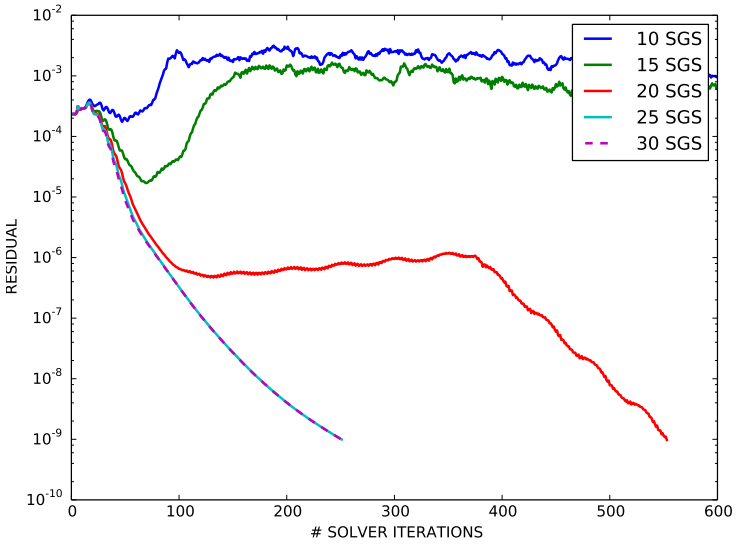


Figure 3.7: 3D subsonic NACA 0012 : residual convergence of the simulation in terms of flow solver iterations for different prescribed maximal number of SGS iterations. No less than 25 SGS iterations are necessary to ensure the global convergence.

at the previous one. We consider that the residual is stagnating from iteration $i-1$ to i, if

$$|Res_i - Res_{i-1}| \quad < \quad \epsilon Res_{i-1}\,,$$

where we use $\epsilon = 10^{-3}$. We chose to stop iterating the Newton method once we detect three stagnations.

3.7 CFL laws

Many CFL laws exist in the literature - linear, geometric, residual based, etc. - but these laws generally require parameters that are difficult to establish optimally because they depend on the flow considered, the geometry and the size of the mesh. In other words, they are too dependent on parameters set by the user. However, they are mandatory to achieve fast convergence in solving non-linear equations.

To avoid this issue, Luke et al. proposed a new approach [109] based on bounding the primitive variables, ρ, p and T, variations at each time step. More precisely, we initially allowed the maximal time step at each vertex, then this local time step is truncated such that the change in ρ, p and T are below a user given percentage η. But, the change in primitive variables during a given interval of time has to be estimated. A way to accomplish this is to solve an explicit time-integration step to describe a functional relationship between time and the primitive variables. Notice that this is done before assembling the matrix and the truncated local time step is used to compute the mass matrix.

This method achieves a maximal efficiency as each vertex is progressing at its own optimal time step. But, that choice is made from an estimation before the resolution of the linear system, thus there is no guarantee that the Newton method will converge.

Another approach has been proposed by Burgess and Glasby [29] which couples an under-relaxation coefficient and dynamic CFL. Here, the solution is analyzed at each step of the Newton method (after solving the linear system) and before updating the solution. First, the change in primitive variables, ρ and p, is again controlled by a user given percentage η and defines an under-relaxation coefficient ω^n at each step of the process. This global coefficient is then applied to the solution evolution: $W^{n+1} = W^n + \omega^n \delta W$. Then, the CFL value is updated depending on that under-relaxation coefficient:

$$CFL^{n+1} = \begin{cases} 0.1\, CFL^n & \text{if} \quad \omega^n < 0.1 \\ CFL^n & \text{if} \quad 0.1 \leq \omega^n < 1 \\ \alpha CFL^n + \beta & \text{if} \quad \omega^n = 1 \end{cases}$$

where we choose $\alpha = 1$ and $\beta = 1$ for a linear increase or $\alpha = 2$ and $\beta = 0$ for a geometric increase. This adaptive CFL, thus time step, is attractive because it is based on the behavior of the Newton method. To improve the robustness of the method even more, they propose setting the solution update to zero when the value of ω^n is less than 0.1, *i.e.,* $\omega^n = 0$.

This approach is extremely robust because if the Newton method diverges, the current step is cancelled and the time step, via the CFL, is automatically reduced. But the criterium considered is global and hence one bad vertex in the mesh can kill the overall efficiency by not allowing the CFL to grow.

In `Wolf`, we consider a hybrid method having the efficiency of the first method and the robustness of the second one. We proceed exactly like the second approach but the under-relaxation coefficient is set locally, *i.e.,* vertex-wise, and each vertex is supplied with its own CFL coefficient which evolves with respect to its own under-relaxation coefficient. Thus, we have a local time step and a local CFL for each vertex.

3.8 Shared memory optimization

This section presents the parallel optimization of `Wolf` using an OpenMP-like approach designed for shared memory architectures of up to 100 cores. This optimization is achieved using the LP3 library [116] developed at Gamma3.

3.8.1 The LP3 library

The LP3 is based on posix standard threads (posix-threads [132] are also known as pthreads) thus taking advantage of multi-core chips and shared memory architectures. This library is specifically designed for algorithms dealing with unstructured meshes. Using the LP3 requires only little knowledge about parallelization and it has a slight impact on the code, as the only modifications needed are at the loop level. In a typical flow solver, there are two kinds of loops to be parallelized: without and with indirect addressing.

The first one presents no possible memory concurrency between two threads launched at the same time. This is the case of a loop over triangles to compute its barycentric coordinates for instance. For this kind of loop, the LP3 library is quite similar to OpenMP.

The second kind of loop can present memory concurrency when run in parallel and thus requires a more subtle management of cache-line overwrite, which we describe in the sequel.

3.8.2 Dealing with indirect addressing

We illustrate how indirect addressing is managed by the LP3 library through the simple example of the computation of the ball area of each vertex of a 2D triangular mesh. In other words, for each vertex we want to compute the sum of the areas of all the triangles it belongs to. To do so, one must loop over the triangles of the mesh and add each triangles' area to its three vertices.

We consider that the following vertex, triangle and mesh structures are defined:

```
typedef struct
{
    double Coordinates[2];
    double BallArea;
}VerStruct;
```

```
typedef struct
{
    VerStruct *Ver[3];
    double Area;
}TriStruct;
```

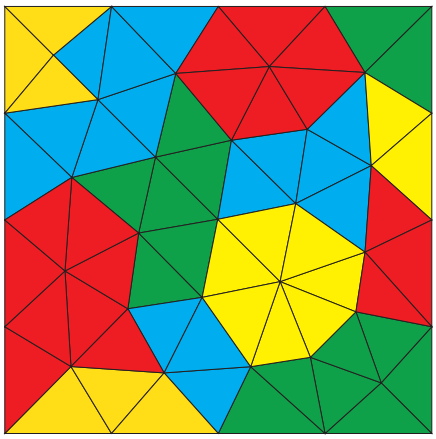
```
typedef struct
{
  int NVer, NTri;
  VerStruct VerTab[NVer];
  TriStruct TriTab[NTri];
}MeshStruct;
```

Here is an example of a serial code for computing each vertex ball:

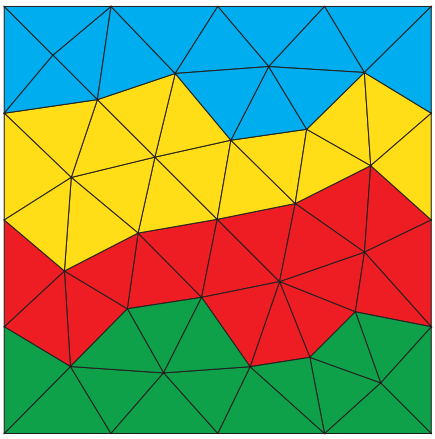
```
for (iTri=1; iTri<=Mesh->NTri; iTri++) {
  tri = &Mesh->TriTab[iTri];
  for (j=0; j<3; j++) {
    ver = tri->Ver[j];
    ver->BallArea += tri->Area;
  }
}
```

This loop over the triangles presents memory dependency when executed in parallel, as if two neighboring triangles are accessed on two different threads simultaneously, a common vertex of theirs might be written on at the same time, causing a memory access conflict and thus a wrong final result. Note that when run in parallel using OpenMP, a duplication of the memory is performed to prevent such cache-line overwrite, which might be costly in terms of both memory and CPU time. The LP3 library does not require any memory overhead, as the triangle table is divided into independent small sub-blocks that can be treated simultaneously. Moreover, the table is divided into more sub-blocks than the thread number (around 64 times more) to allow a dynamic scheduling: as a thread completes its task, it is dynamically reassigned to another sub-block.

Figure 3.8 shows the importance of mesh renumbering in creating the sub-blocks. We consider a triangular mesh and we want to compute the ball volume of all its vertices in parallel using 2 threads. To do so, the triangle table is divided into 4 sub-blocks, each one of them corresponding to a color. In Figure 3.8a, the partitioning does not make it possible to execute two sub-blocks simultaneously as each sub-block (∼color) has a common frontier with the three other sub-blocks. In Figure 3.8b however, two independent sets of sub-blocks were created, so that it is possible to run the blue and the red blocks at the same time, as well as the yellow and the green together.



(a) A bad partitioning: each block presents memory concurrencies with the three others.



(b) A good partitioning: the blue and the red blocks can be run simultaneously, as well as the yellow and the green.

Figure 3.8: Illustration of memory concurrencies through the parallel computation of the vertices' ball volume using 2 threads. A loop is performed over the triangles and a value is written on their vertices. The mesh is divided into 4 sub-blocks (∼color). Two blocks can not be run simultaneously if they present memory concurrencies, i.e. if two triangles from two different blocks share a common vertex.

In practice, the LP3 creates independent sub-blocks by reordering the mesh using Hilbert space-filling curves [149] which is a very fast method. In Chapter 2, a description of mesh partitioning methods is given, including this Hilbert method.

We now present the slight source code modifications necessary to parallelize a loop presenting memory dependencies using the LP3 library. The first step consists in declaring these dependencies. In the case of the vertices' ball volume computation, there exists a dependency between triangles and their vertices:

```
BeginDependency(Msh->TriTab,Msh->Ver);
for (iTri=1; iTri<=NbrTri; iTri++) {
  tri = &Msh->TriTab[iTri];
  for (j=0; j<3; j++) {
    AddDependency(iTri, tri->Ver[j]);
  }
}
```

```
EndDependency(Msh->TriTab,Msh->Ver);
```

Then, the previous serial loop over triangles becomes:

```
Solve(Msh->TriTab,iBeg,iEnd) {
for (iTri=iBeg; iTri<=iEnd; iTri++) {
  // What is inside the loop remains unchanged
  tri = &Mesh->TriTab[iTri];
  for (j=0; j<3; j++) {
    ver = tri->Ver[j];
    ver->BallArea += tri->Area;
} } }
```

3.8.3 Example of timings

We analyse the performance of the LP3 library on the two different computers presented in Table 3.1. The case we consider is an unsteady spherical blast computed on a mesh with 2 173 612 vertices and 13 037 975 tetrahedra. 180 flow solver iterations were performed. Note that I/Os are not included in the timings presented. **NB:** This simulation employs an explicit time integration.

Computer 1	Computer 2
• 2 chips: Xeon E5-2670 10 cores 2.5 GHz • Hyper-threading • Both chips are connected by 2 QPI links with a speed of 16 GB/s • 64 GB RAM	• 4 chips: Xeon E7-4850 10 cores 2 GHz • Hyper-threading • All chips are connected to all by 1 QPI link with a speed of 16 GB/s • 1 TB RAM

Table 3.1: Two computers used for the performance analysis.

Timings. The timings are presented in Table 3.2 (computer 1) and Table 3.3 (computer 2). We make two observations: (i) the speed-up is excellent up to 1 chip with 10 cores and good for 2 chips, and (ii) the speed-up drops for more than two chips, due to memory access speed.

Nbr. cores	Serial	1 HT	2 HT	4 HT	8 HT	10 HT	20 HT
Timings (sec.)	1,054	878	423	236	133	112	71
Speed-up	1.0	1.2	2.5	4.5	7.9	9.4	14.9

Table 3.2: Timings and speed-up observed using computer 1 (HT stands for hyper-threading).

Nbr. cores	Serial	1 HT	2 HT	4 HT	8 HT	10 HT	20 HT	40 HT
Timings (sec.)	2,072	1,506	759	393	228	193	121	117
Speed-up	1.0	1.4	2.8	5.2	9.1	10.7	17.1	17.7

Table 3.3: Timings and speed-up observed using computer 2 (HT stands for hyper-threading).

3.9 Conclusion

In this chapter we introduced the numerical choices we made in `Wolf` and presented some of the main related work that has been achieved during this PhD. It includes the implementation of turbulence modeling, as well as improvements in the robustness and the convergence speed.

Chapter 4

Verification and Validation of the Flow Solver

Contents

4.1 Introduction

This chapter presents a set of test cases for the verification and validation (V&V) [146] of the flow solver `Wolf`. V&V studies of two and three-dimensional problems are presented within the context of turbulent flows modeled by the Reynolds-averaged Navier-Stokes (RANS) equations, for a wide range of Mach numbers and geometrical configurations. Various test cases were studied, from a simple subsonic flat plate to more complex configurations such as geometries from the Drag Prediction Workshop.

4.1.1 About V&V

CFD software that solves the RANS equations has been widely used for the last twenty-five years. It is used not only for basic research in fluid dynamics, but is also intensively employed for the analysis and design processes in many industries worldwide, including aerospace, petroleum exploration, power generation, etc.

As the RANS equations are unclosed, a model is necessary to predict turbulent effects on the mean flow, through the Reynolds stress terms (more details are provided in the description of our flow solver, see Chapter 3). This turbulence model is thus a key feature of solving the RANS equations, but is also one of its largest sources of uncertainty, as most models are known to be flawed in one way or another. For instance, RANS models in CFD are known to be reliable for predicting attached flow, but many of them remain inaccurate when computing flows involving separation. Despite its associated uncertainties, RANS turbulence modeling has proved its industrial-readiness in the aerospace field. Confidence in its results was made possible by an important step of the development and the implementation of a turbulence model: its verification and validation study.

Verification ensures that a turbulence model was implemented correctly, i.e. as intended according to the equations and the boundary conditions. Its objective is to detect and correct bugs in the implementation. This verification step is usually done either through the use of manufactured solutions, or through meticulous comparisons with other flow solvers. In the sequel, we only present code to code comparisons.

Validation is performed after the verification step. Its objective is to establish the 'goodness' of a model, i.e. to assess its ability to represent different types of flow physics. Validation thus involves a large number of test-case comparisons, including comparisons with other codes and experiments.

4.1.2 V&V resources

Carrying out a V&V study requires data from experiments as well as results from other codes. Over the years, numerous workshops have focused on providing such data. They have proved to be valuable

resources when it comes to identifying strengths and weaknesses of turbulence models for particular problems of interest. We list below the resources we used for our V&V study.

The European Research Community on Flow, Turbulence, and Combustion (ERCOFTAC) has sponsored numerous workshops since 1991 on "Refined Turbulence Modeling". Many of these workshops are well documented on the ERCOFTAC website, including experimental data from a large set of test cases.

The Drag Prediction Workshop (DPW) series is organized by members of a working group of the AIAA commitee and is open to participants worldwide. The objective of the workshop is to assess state-of-the-art computational methods for aircraft force and moment prediction. There have been five editions starting from the first one in 2001 to the last one in 2012. Previous to each edition, an industry-relevant configuration is provided to the participants along with a series of meshes of different sizes and experimental data. A set of simulations is required and results such as the drag polar or the pressure coefficient are compared.

The NASA Langley Research Center Turbulence Modeling Resource (TMR) Website is a central resource for turbulence model V&V. It provides precise definitions of the commonly used turbulence models, and a set of test cases including grids, experimental data and results from other CFD codes (especially from CFL3D [83] and FUN3D [17]).

4.1.3 Numerical method

The numerical method used is the same for almost all the following V&V test cases (unless otherwise specified). We used the second-order HLLC flow solver with the V6 numerical dissipation scheme and the Dervieux Limiter. The Spalart-Allmaras turbulence model was used (one equation model, no trip term). For the implicit time integration, we aim to decrease the residual of the linear system by two orders of magnitude at each solver iteration. The maximal number of SGS iterations is set to 20. Gradients are recovered using a weighted least-square approach [14].

4.2 Verification test cases

This set of test cases aims at ensuring that the turbulence model was implemented correctly (that there are no bugs in the code, for instance).

4.2.1 Turbulent flat plate

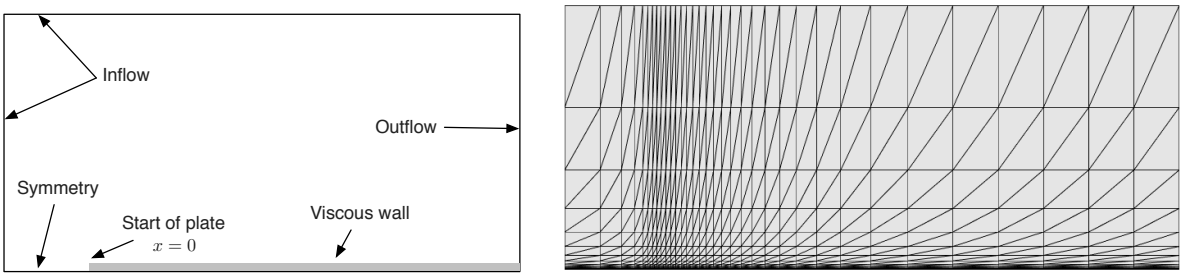
Description. The flow over a flat plate with zero pressure gradient is considered. The free-stream conditions are summarized in Table 4.1. Note that the Mach number ($M_\infty = 0.2$) is low enough for the

flow to be considered as "essentially" incompressible, but the test case remains a compressible verification case and has been compared to other compressible codes.

M_∞	Re_L	T_∞	α
0.2	$5M$	$300K$	0°

Table 4.1: Turbulent flat plate free-stream conditions.

Computational Domain. The rectangular computational domain is depicted in Figure 4.1a. Five structured grids (ranging from the finest 545 x 385 to the coarsest 35 x 25) were downloaded, and converted to unstructured meshes (decomposed into triangles) (see Figure 4.1b). The average $y+$ range is from 0.1 (i.e. minimum wall-normal spacing $y = 5 \times 10^{-7}$) for the finest mesh to $y+ = 1.7$ for the coarsest, which remains reasonably fine.

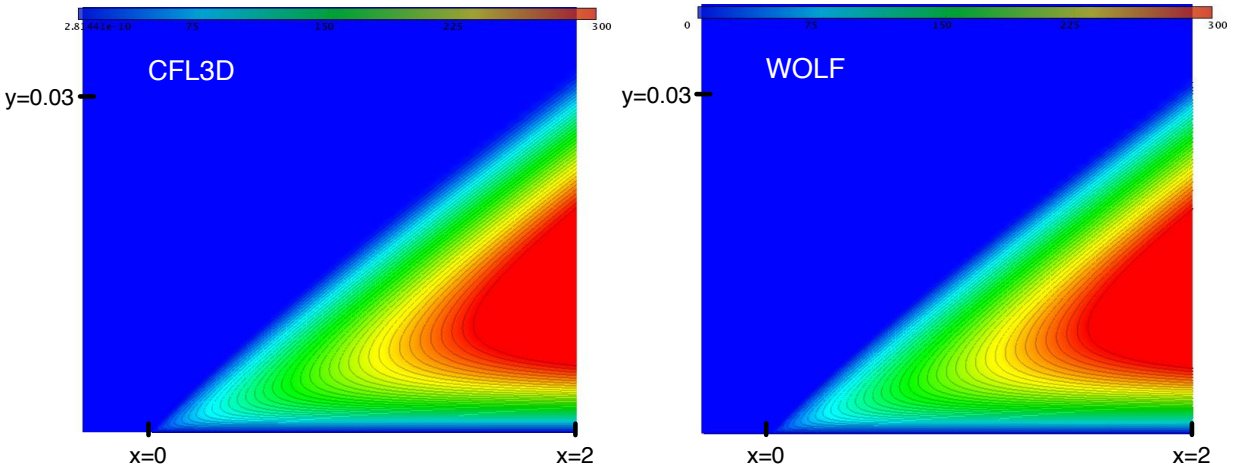


(a) Boundary conditions.

(b) Mesh of the domain (coarsest mesh).

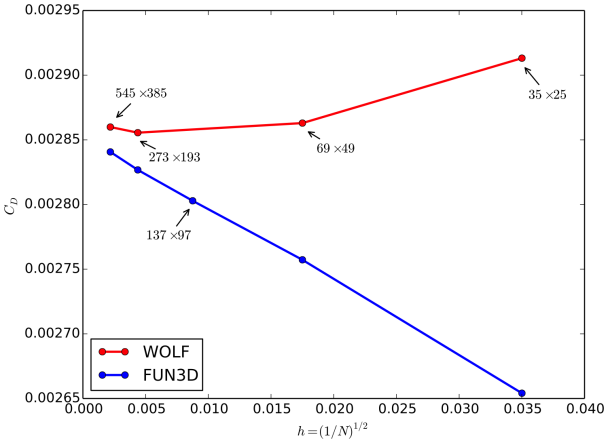
Figure 4.1: Turbulent flat plate: computational domain.

Results. The results from `Wolf` were compared to those from FUN3D (and CFL3D for the eddy viscosity contours) and are in good agreement. The following plots were observed: drag convergence (Figure 4.3a), convergence of the skin friction coefficient (C_f) at $x = 0.97$ (Figure 4.3b), skin friction coefficient (Figure 4.3c), eddy viscosity contours (Figure 4.2), maximum nondimensional eddy viscosity as a function of x (Figure 4.3d), nondimensional eddy viscosity at $x = 0.97$ (Figure 4.3e), velocity profiles at $x = 0.97$ and $x = 1.90$ (Figure 4.3f).

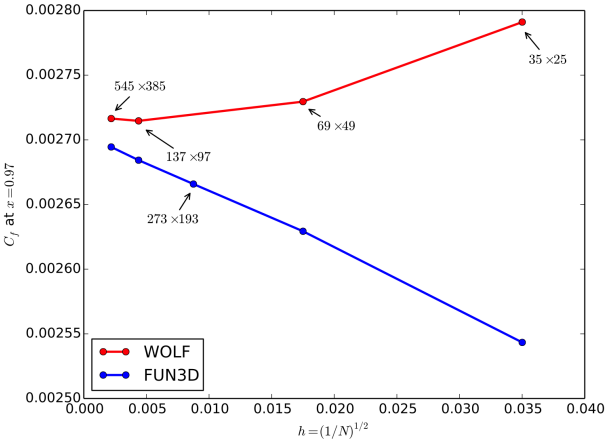


(a) CFL3D. (b) WOLF.

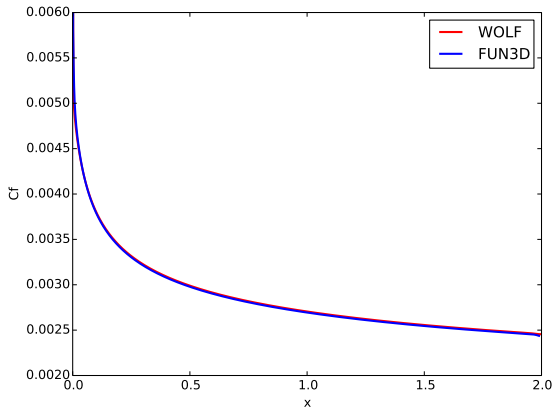
Figure 4.2: Turbulent flat plate: Eddy viscosity contours. The x-coordinate was scaled so that $x_{sca} = \frac{1}{50}x$.



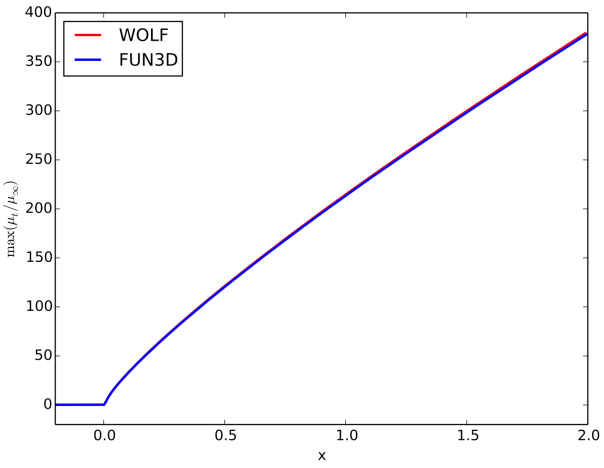
(a) Turbulent flat plate: Drag convergence.



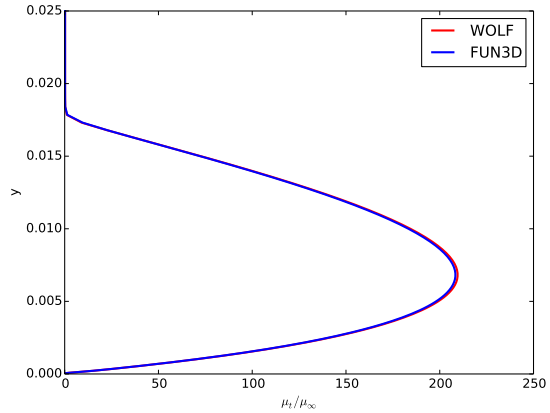
(b) Turbulent flat plate: convergence of the skin friction coefficient (C_f) at $x = 0.97$.



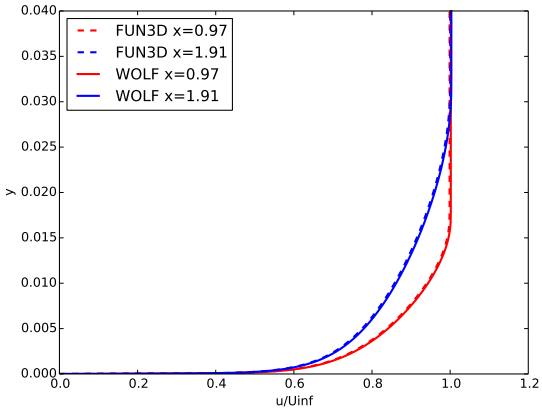
(c) Turbulent flat plate: Surface skin friction coefficient on finest mesh (545×385).



(d) Turbulent flat plate: Maximum nondimensional eddy viscosity as a function of x.



(e) Turbulent flat plate: Nondimensional eddy viscosity at x=0.97.



(f) Turbulent flat plate: Velocity profiles.

Figure 4.3: 2D turbulent flat plate.

4.2.2 2D bump in a Channel

Description. Flow over a bump in a channel is useful for verification purposes, as in contrast to the flat plate it introduces non-zero pressure gradients on the wall. A description of this test case is presented in Figure 4.4, and the freestream conditions are summarized in Table 4.2.

M_∞	Re_L	T_∞	α
0.2	$3M$	$300K$	$0°$

Table 4.2: 2D bump in a channel: free-stream conditions.

The definition of the bump is given by:

$$y = \begin{cases} 0.05 \times \sin^4(\frac{\pi x}{0.9} - \frac{\pi}{3}), & \text{for } 0.3 \leq x \leq 1.2 \\ 0, & \text{for } 0 \leq x < 1.2 \text{ or } 1.2 < x \leq 1.5\,. \end{cases}$$

Computational Domain. Five structured grids (ranging from the finest 1409×641 to the coarsest 89×41) were downloaded, and converted to unstructured meshes (decomposed into triangles) (see Figure 4.5).

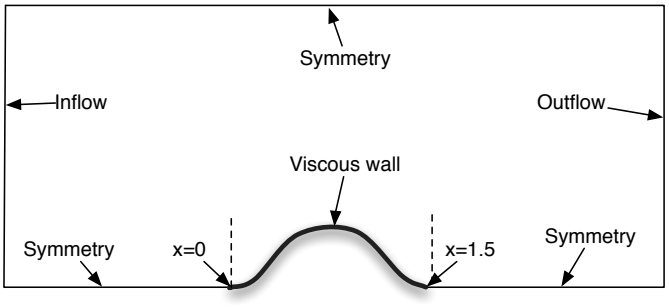


Figure 4.4: 2D bump: Test case description.

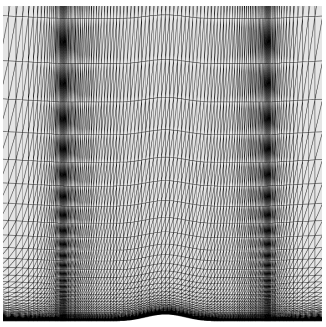
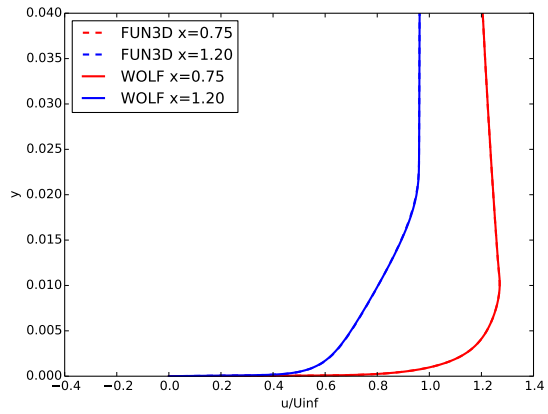
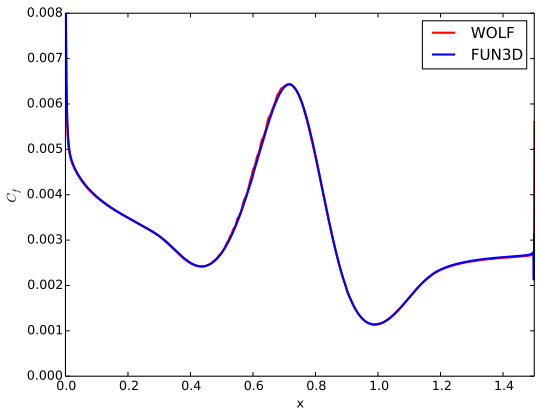


Figure 4.5: Mesh of the 2D bump: close-up view of the bump (177×81 mesh).

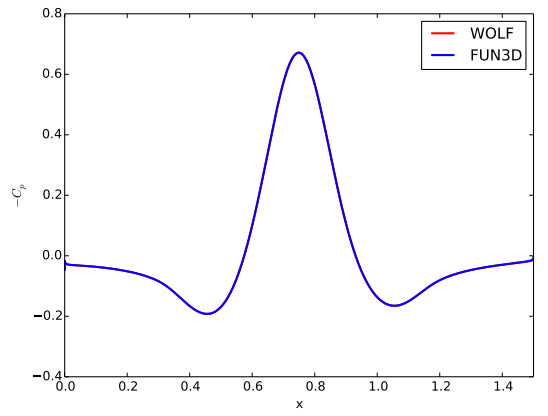
Results. The results from `Wolf` are in good agreement with FUN3D: velocity profiles at $x = 0.75$ and $x = 1.20$ (Figure 4.6a), skin friction coefficient C_f (Figure 4.6b), pressure coefficient C_p (Figure 4.6c), eddy viscosity at $x = 0.75$ (Figure 4.6d), maximal nondimensional eddy viscosity as a function of x (Figure 4.6f).



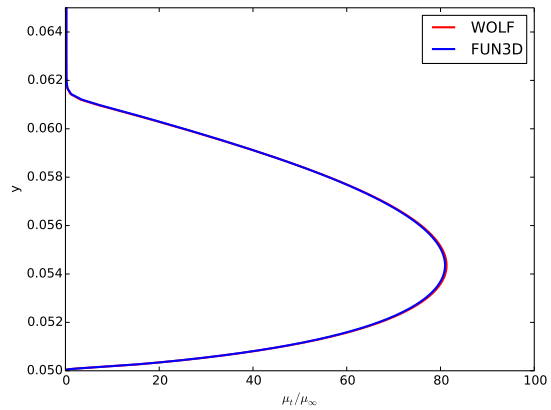
(a) Velocity profiles at $x = 0.75$ and $x = 1.20$.



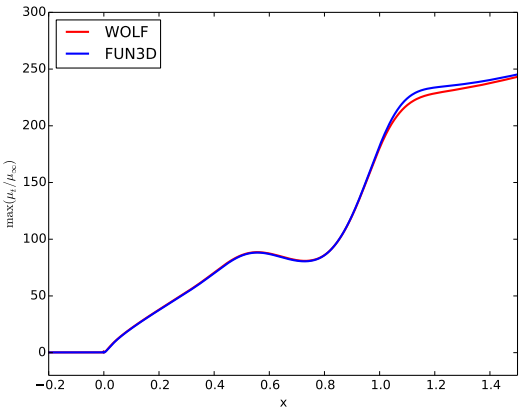
(b) Skin friction coefficient C_f.



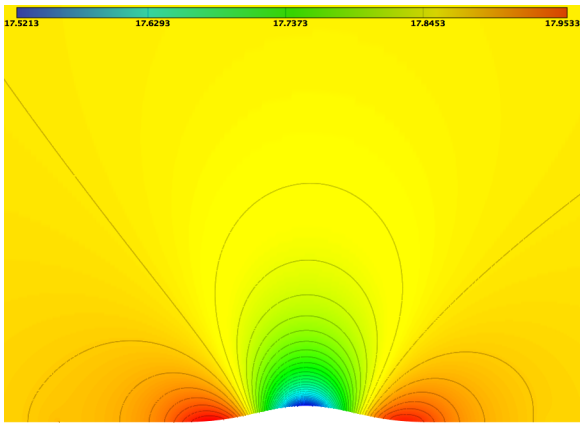
(c) Pressure coefficient C_p.



(d) Nondimensional eddy viscosity at $x = 0.75$.



(e) Maximal nondimensional eddy viscosity as a function of x.



(f) Solution (pressure).

Figure 4.6: 2D bump-in-channel verification.

4.3 Validation test cases

The following validation test cases seek to assess the solver's ability to reproduce the physics. It differs from verification, which seeks to establish that a model has been implemented correctly.

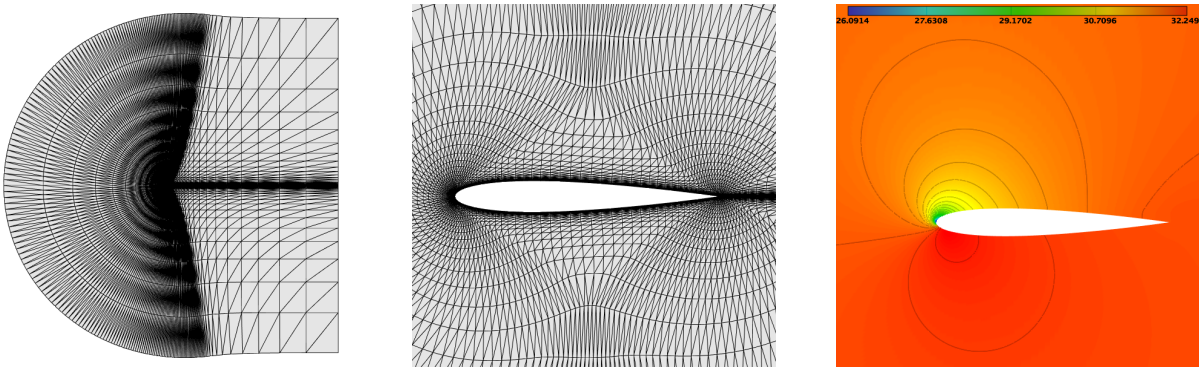
4.3.1 2D NACA 0012 airfoil

Description. The NACA 0012 airfoil has been tested in most wind tunnels in the world and is widely used for validation purposes. A turbulent flow (the freestream conditions are summarized in Table 4.3) is applied and results are compared to FUN3D for three different angles of attack α : 0°, 10° and 15°.

M_∞	Re_L	T_∞	α
0.15	$6M$	$300K$	0°, 10°, 15°

Table 4.3: NACA 0012: free-stream conditions ($L = 1$ is the airfoil chord).

Computational Domain. Five structured grids (ranging from the finest 1793×513 to the coarsest 113×33) were downloaded, and converted to unstructured meshes (decomposed into triangles) (see Figure 4.7).



(a) Mesh of the domain. (b) Close-up view of the airfoil. (c) Solution for $\alpha = 15°$ (pressure).

Figure 4.7: Mesh of the NACA 0012 airfoil.

Results. For each angle of attack (0, 10 and 15°), we compare the pressure coefficient C_p and the skin friction coefficient C_f. Results are presented in Figure 4.8 and are in good agreement with FUN3D.

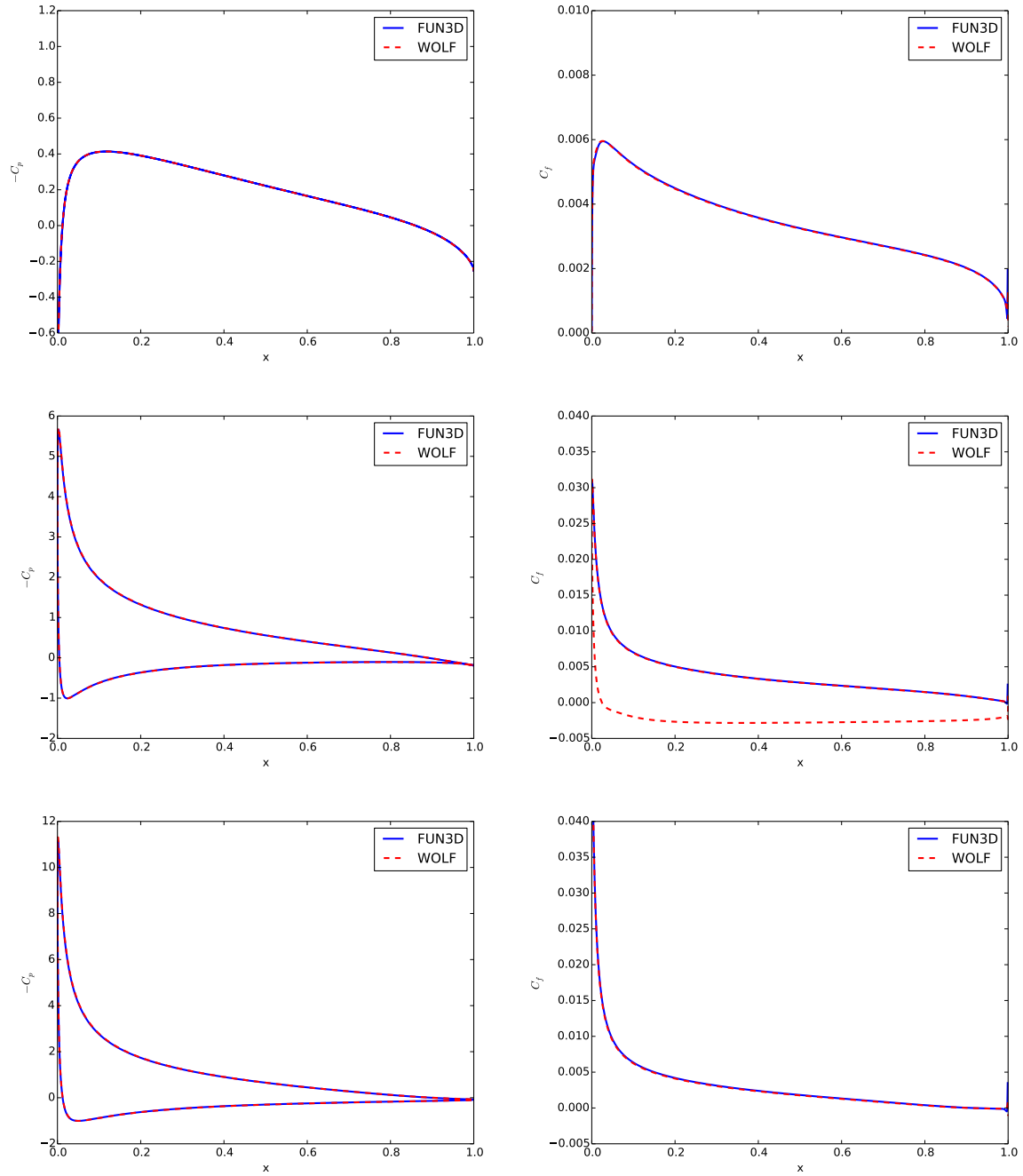


Figure 4.8: NACA0012: Pressure coefficient C_p (left column) and skin friction coefficient C_f (right column) for $\alpha = 0, 10, 15°$ (first, second and third line, resp.). Comparison to FUN3D (C_p and C_f).

4.3.2 2D transonic RAE2822

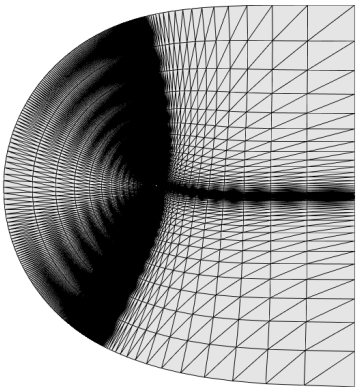
Description. We study a transonic flow around the RAE2822 airfoil. Two cases from the HAGARD report [41] are studied: case 6 and case 9 which are summarized in Table 4.4.

Case	M_∞	Re_L	T_∞	α
6	0.725	$6.5M$	$300K$	2.92°
9	0.730	$6.5M$	$300K$	3.19°

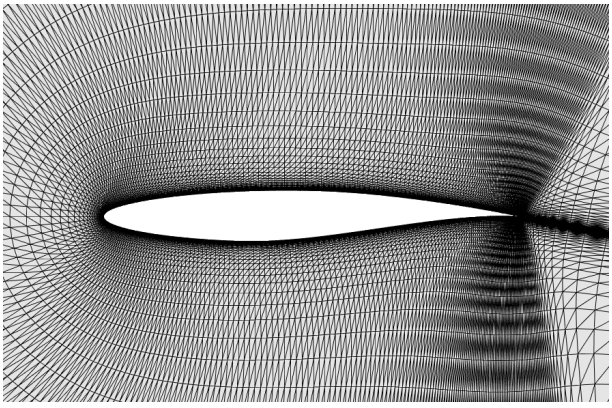
Table 4.4: 2D RAE 2822: free-stream conditions for case 6 and 9.

Mesh. The mesh used (see Figures 4.9a and 4.9b) was downloaded from the NPARC alliance test cases database [31] and decomposed into triangles. It contains 23 952 vertices and 47 104 triangles.

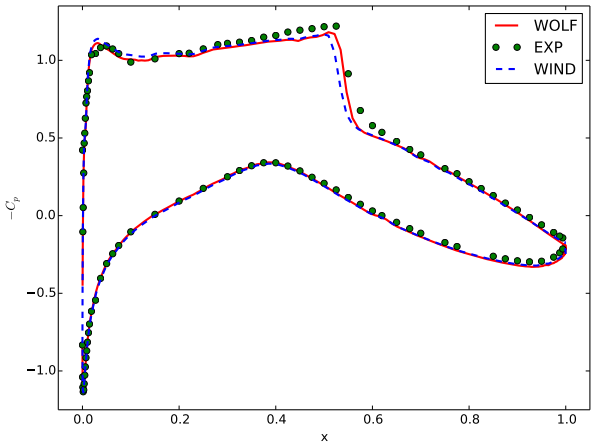
Results. We compare the pressure coefficient on the airfoil from Wolf, Wind and experimental data. The results are presented in Figure 4.9c and Mach isolines are depicted in Figure 4.9d.



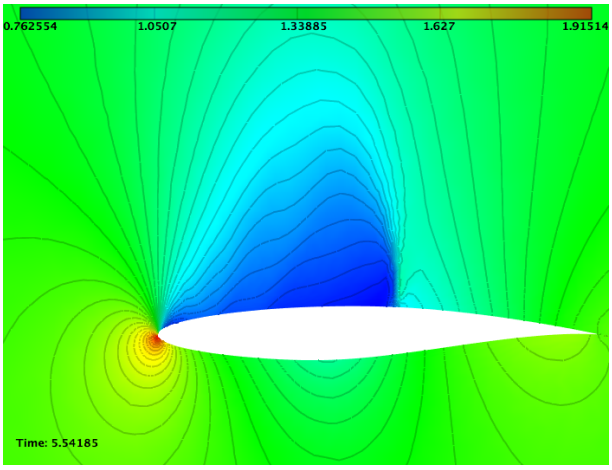
(a) Mesh of the domain.



(b) Close-up view of the airfoil.



(c) Pressure coefficient.



(d) Solution (pressure).

Figure 4.9: RAE2822.

4.3.3 2D backward-facing step

Description. As described in Figure 4.10, a turbulent boundary layer encounters a sudden back step, causing flow separation. The freestream conditions are summarized in Table 4.5. Note that the back pressure was adjusted in order to achieve a Mach number $M = 0.128$.

M_∞	Re_L	T_∞	α
0.128	36000	$298K$	$0°$

Table 4.5: 2D step: free-stream conditions ($L = 1$).

Computational Domain. Five structured grids (ranging from the finest to the coarsest) were downloaded, and converted to unstructured meshes (decomposed into triangles) (see Figure 4.11).

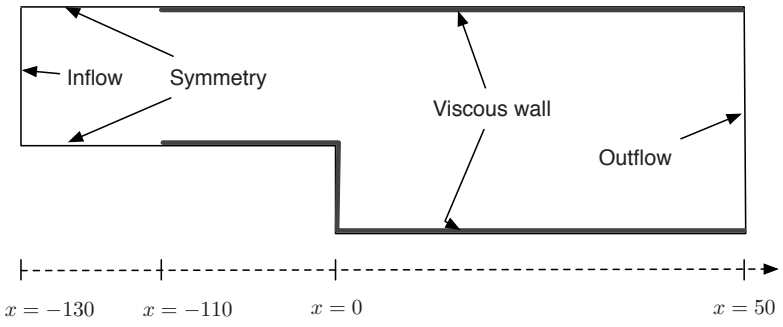


Figure 4.10: 2D step: Test case description.

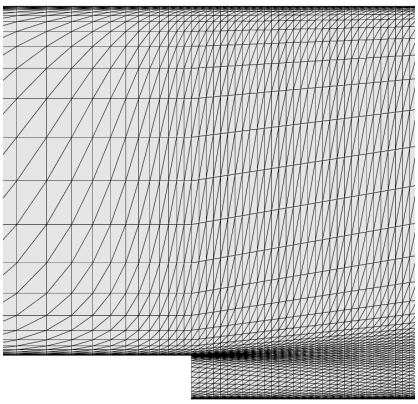


Figure 4.11: Mesh of the 2D step: close-up view of the step (coarsest mesh).

Results. The results from `Wolf` are in good agreement with those from FUN3D: skin friction coefficient C_f (Figure 4.13a), pressure coefficient C_p (Figure 4.13b), velocity profile at $x = -4$ (Figure 4.13e), velocity profiles downstream of the step (Figure 4.13c), turbulent shear stress (Figure 4.13d). The solution is shown in Figure 4.12.

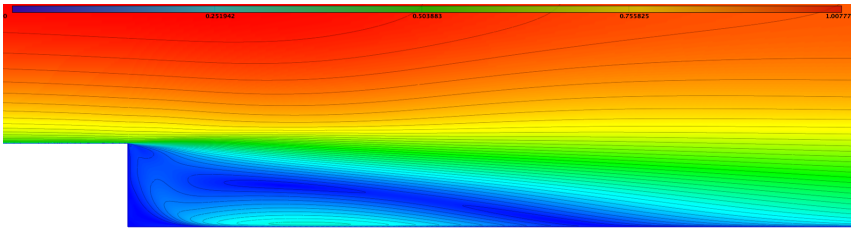
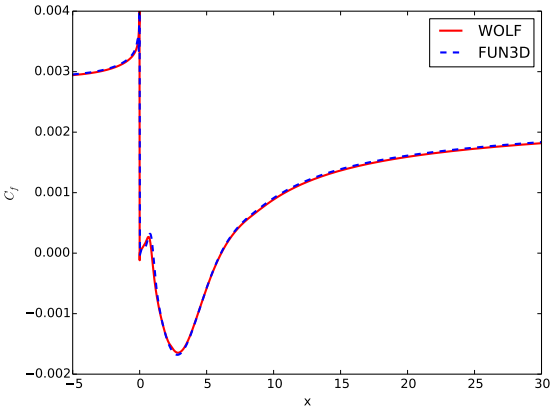
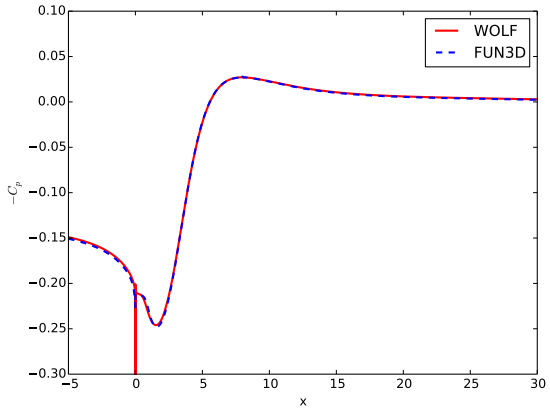


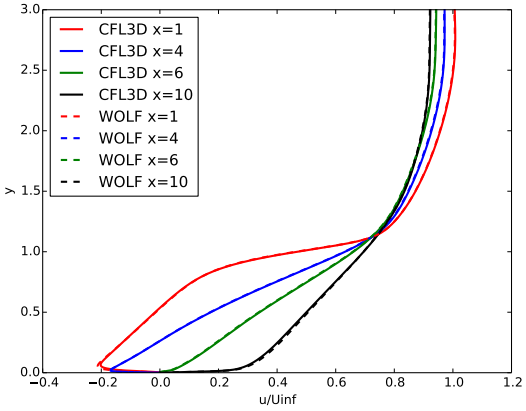
Figure 4.12: 2D step : Mapping of the velocity (finest mesh).



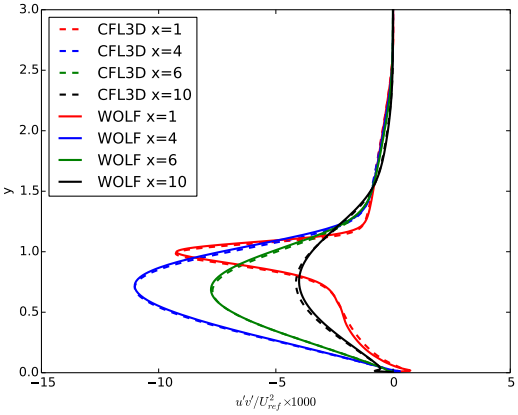
(a) Skin friction coefficient C_f.



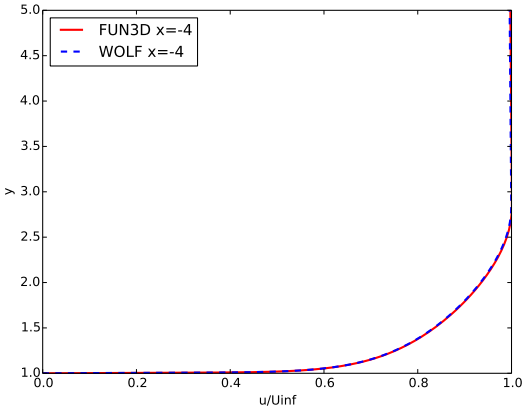
(b) Pressure coefficient C_p.



(c) Velocity profiles downstream of the step.



(d) Turbulent shear stress.



(e) Velocity profile at $x = -4$.

Figure 4.13: 2D backward-facing step.

4.3.4 2D airfoil near-wake

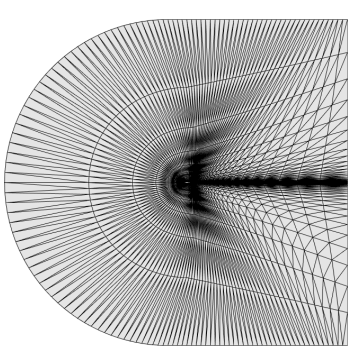
Description. We consider the near-wake airfoil from Nakamaya [130], which belongs to the set of cases of the TMR website. The freestream conditions are summarized in Table 4.6.

M_∞	Re_L	T_∞	α
0.088	$1.2M$	$300K$	$0°$

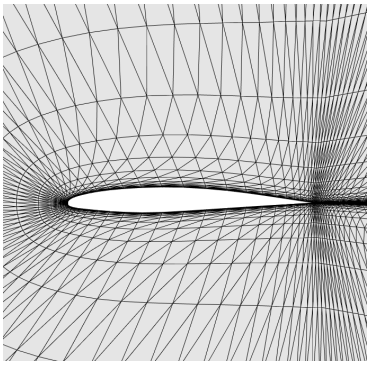
Table 4.6: Airfoil near-wake: free-stream conditions ($L = 1$ is the airfoil chord).

Computational Domain. Five structured grids (ranging from the coarsest containing 3 486 vertices to the finest containing 862 176 vertices) were downloaded, and converted to unstructured meshes (decomposed into triangles) (see Figures 4.14a and 4.14b).

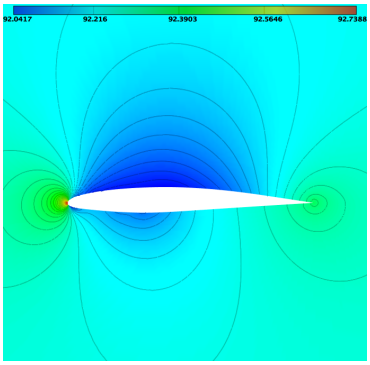
Results. A view of the solution is shown in Figure 4.14c. We compare velocity profiles (Figure 4.14d) and turbulent shear stress profiles (Figure 4.14e) to CFL3D and experimental measurements (at several locations). The velocity profiles are slightly in better agreement with the experiment than CFL3D is. However, our prediction of the turbulent shear stress is not as good.



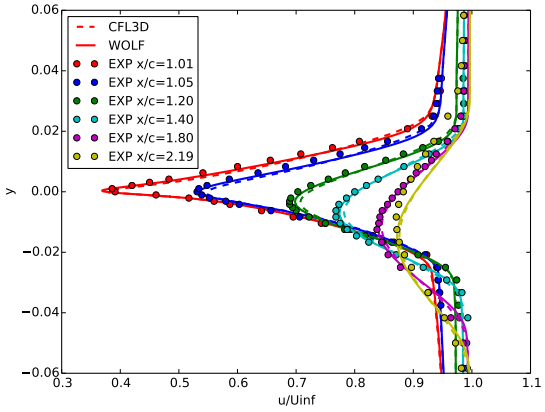
(a) Mesh of the domain (coarsest mesh).



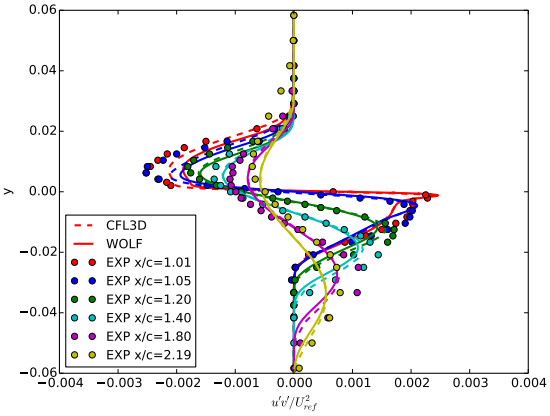
(b) Close-up view of the airfoil (coarsest mesh).



(c) Pressure (finest mesh).



(d) Velocity profiles.



(e) Turbulent shear stress.

Figure 4.14: 2D airfoil near-wake.

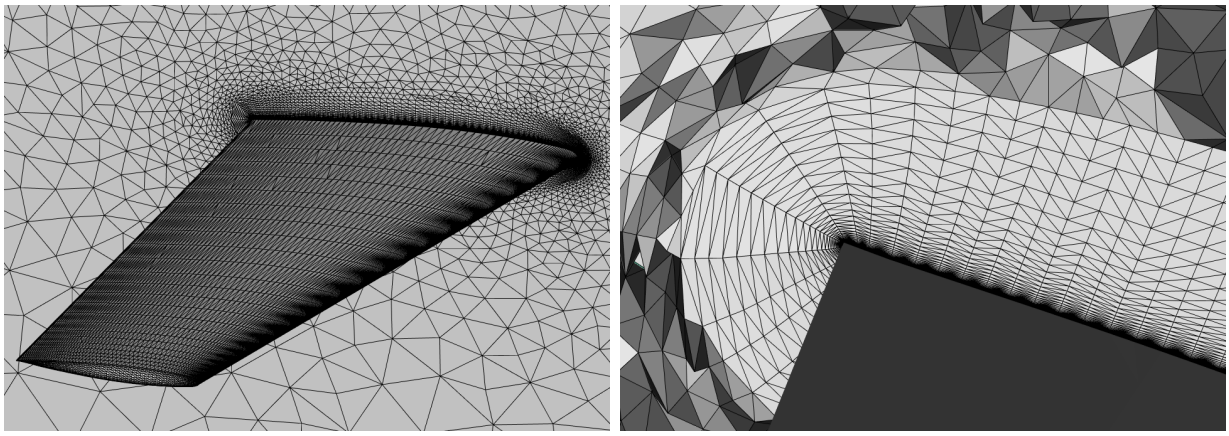
4.3.5 ONERA M6 wing

Description. The ONERA M6 wing has become a classic validation case, thanks to the simplicity of its geometry combined with the complexity of the physics observed when a transonic flow is applied over it. The flow configuration we chose is documented in the AGARD report [150]. The freestream conditions are summarized in Table 4.7.

M_∞	Re_L	T_∞	α
0.8395	$11.72M$	$255.55K$	$3.06°$

Table 4.7: ONERA M6 wing: free-stream conditions ($L = 0.64607$).

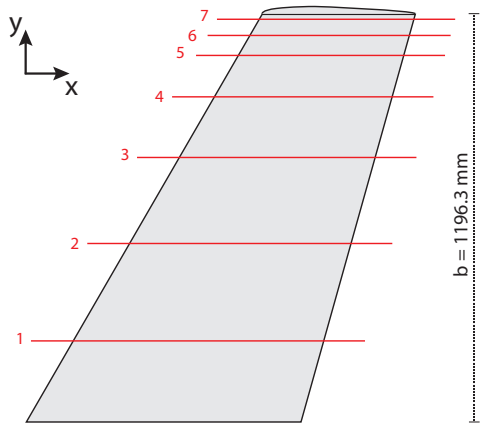
Computational Domain. The mesh used for the computation is depicted in Figure 4.15 (surface mesh). It was generated using our in-house software.



(a) Surface mesh. (b) Boundary layer mesh (cut in the volume).

Figure 4.15: Mesh of the ONERA M6 wing.

Results. The pressure coefficient C_p is in good agreement with experimental data, see Figure 4.17. It was compared along the extraction lines depicted in Figure 4.16. **NB:** The mesh we used is different from the one used for the WIND computation.



#	y/b
1	0.20
2	0.44
3	0.65
4	0.80
5	0.90
6	0.95
7	0.99

Figure 4.16: M6 wing: Location of the seven extraction lines for the pressure coefficient.

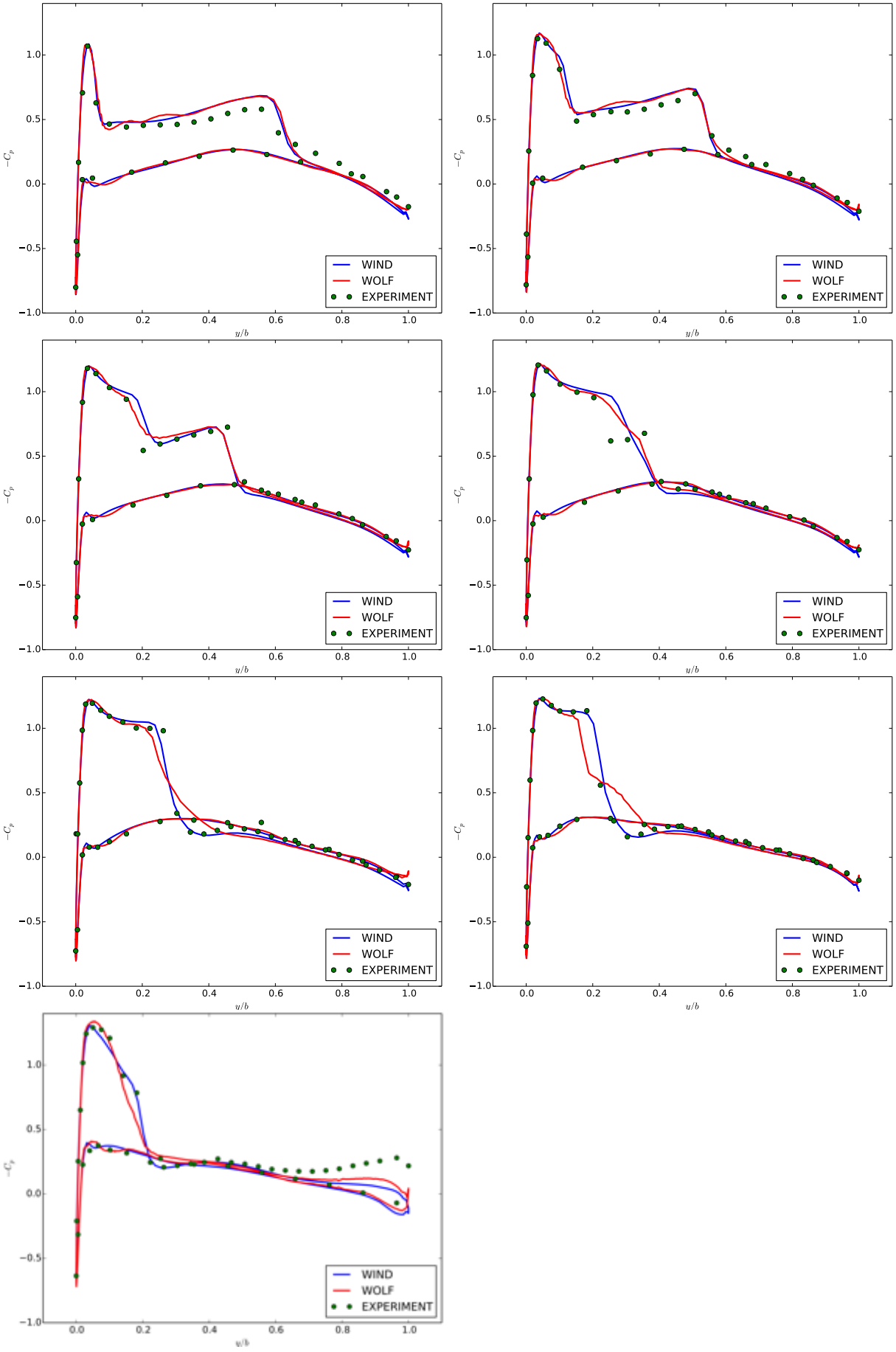


Figure 4.17: ONERA M6 Wing : Pressure coefficients at the 6 extraction lines presented in Figure 4.16.

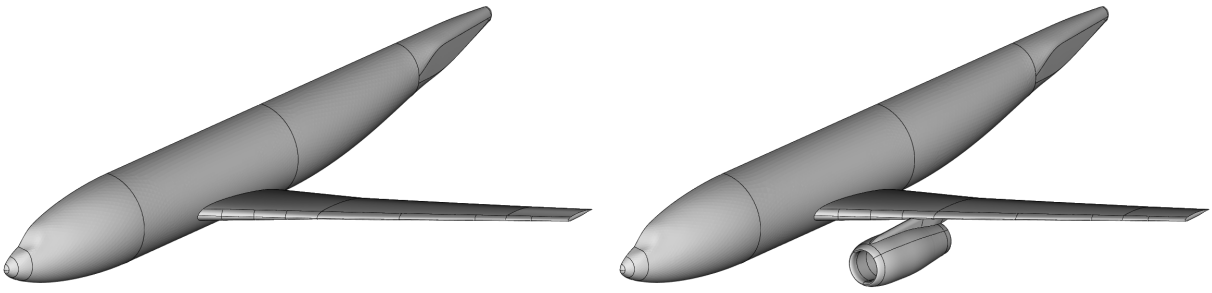
4.3.6 2nd Drag Prediction Workshop

Description. The objective of the Drag Prediction Workshop (DPW) series is to assess state-of-the-art computational methods for aircraft force and moment prediction. This section presents the results we obtained for the 2nd DPW. Two industry-relevant geometries are provided by the workshop organizers: a wing-body (WB) and a wing-body-nacelle (WBN) configuration, see Figure 4.18. The freestream conditions are summarized in Table 4.8.

M_∞	Re_L	T_∞
0.75	$3M$	$288.15K$

Table 4.8: DPW2: free-stream conditions ($L = 0.1412m$).

Numerical method. For this test case, a fourth order numerical dissipation was used (instead of the V6 scheme).



(a) Wing Body configuration. (b) Wing Body Nacelle configuration.

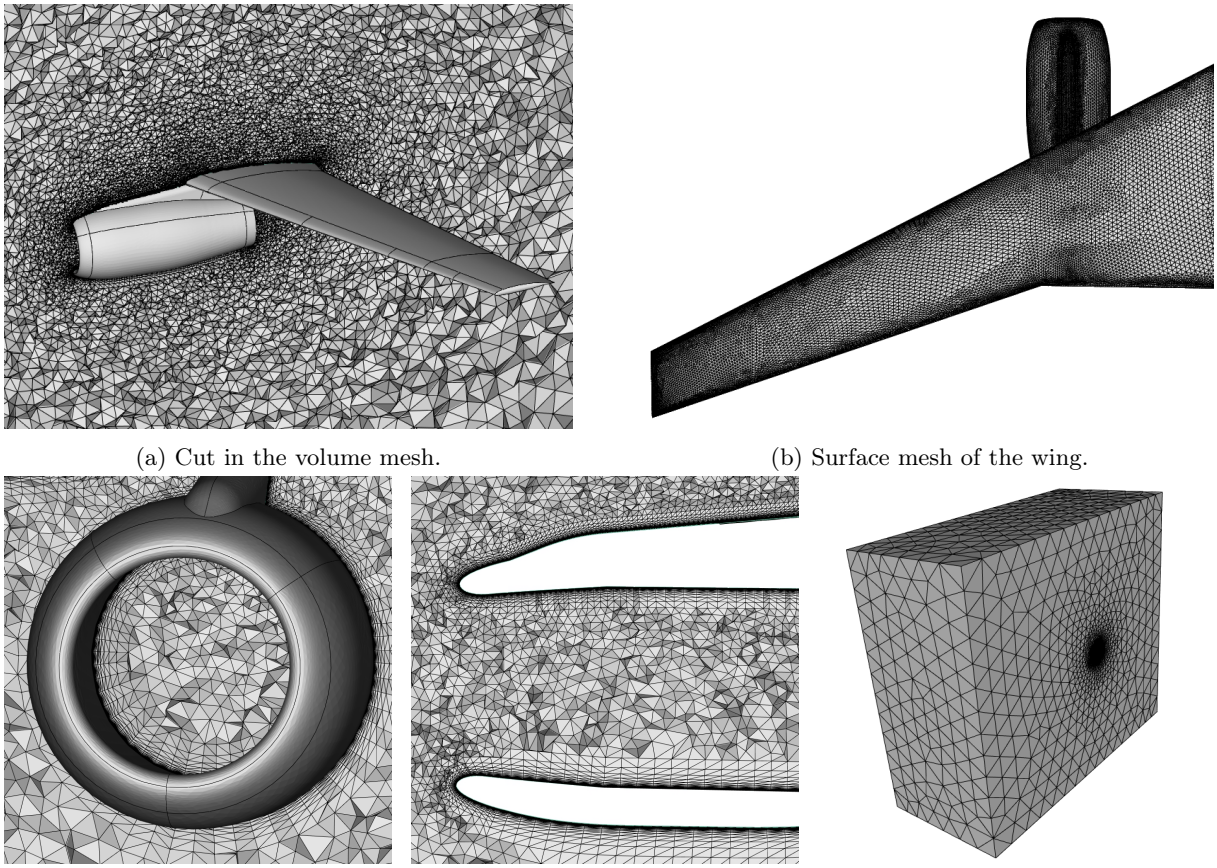
Figure 4.18: DPW2 geometries.

Meshes. We used five unstructured meshes that we downloaded from the workshop's website: three for the WB and two for the WBN configuration, see Table 4.9. Views of the coarse mesh of the Wing/-Body/Nacelle configuration are presented in Figure 4.19.

Configuration	Coarse	Medium	Fine
Wing/Body			
# Points	246 020	675 946	1 984 343
WB + Nacelle			
# Points	1 827 470	4 751 207	×

Table 4.9: Meshes used for the DPW2 case.

Results. We present the drag polar study that we carried out, as well as the pressure coefficient extractions we obtained for iso-lift simulations.



(a) Cut in the volume mesh.

(b) Surface mesh of the wing.

(c) Nacelle.

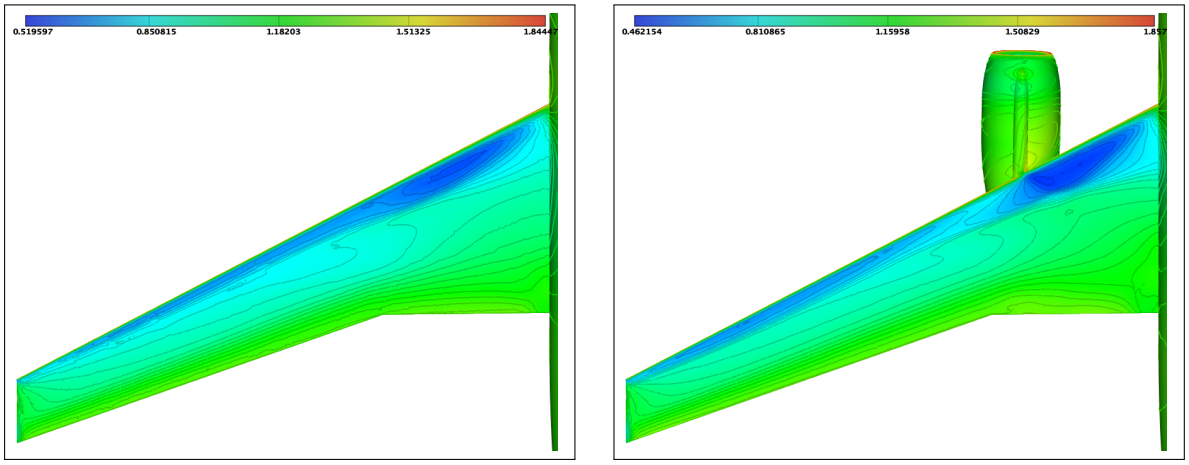
(d) Cut through the nacelle.

(e) Domain.

Figure 4.19: DPW2: Coarse mesh of the Wing/Body/Nacelle configuration.

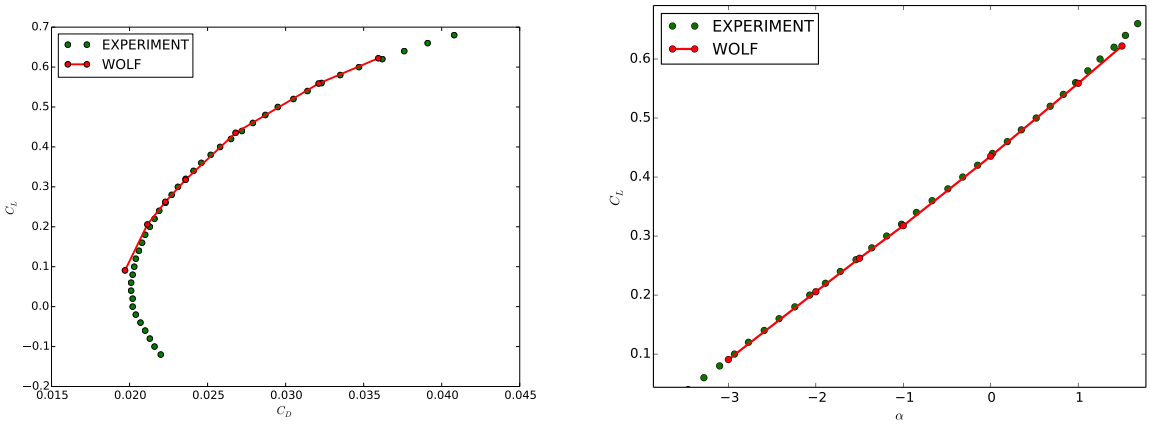
- **Drag polar study.** This consists in running several simulations using different angles of attack. Here, seven angles α are set: $\alpha = -3°, -2°, -1.5°, -1°, 0°, 1°, 1.5°$. This leads to fourteen simulations (seven for each configuration), performed using the fine mesh for the WB configuration and the medium mesh for the WBN configuration. The results are presented in Figures 4.21 and 4.22 and fit the experimental values.

- **Iso-lift simulations:** the angle of attack is set, so that the lift coefficient C_L is equal to 0.500 ± 0.001. The angles of attack leading to this lift value were obtained thanks to the drag polar study. For both configurations, we compare extractions of the pressure coefficient C_p along the eight lines depicted in Figure 4.23. The extractions are shown in Figure 4.24 (WB) and 4.25 (WBN). The prediction of the pressure coefficient along these lines is in good agreement with experimental data. Views of the solutions are shown in Figure 4.20.



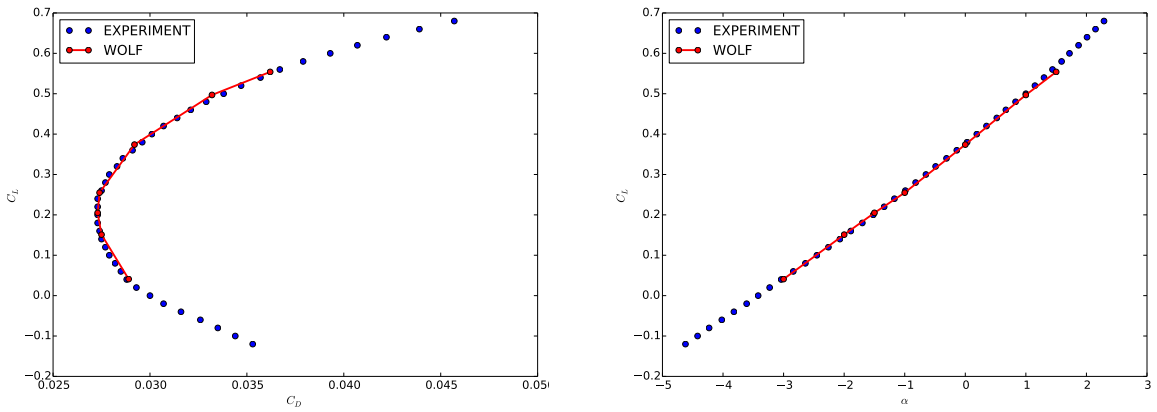
(a) Wing Body configuration. (b) Wing Body Nacelle configuration.

Figure 4.20: DPW2 solutions (pressure).



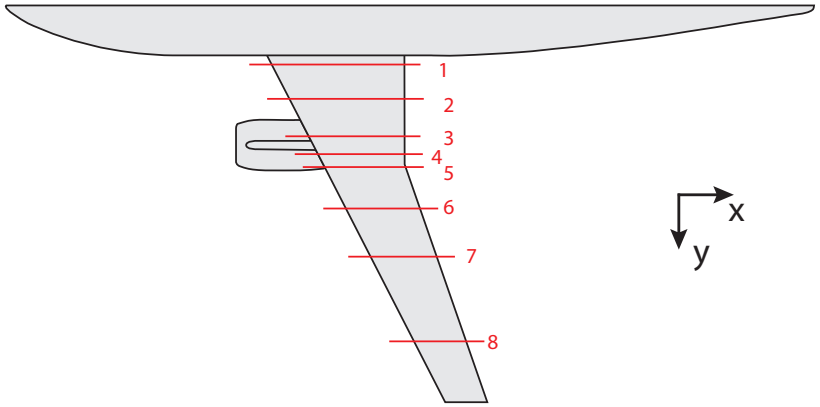
(a) Sweep. (b) Polar.

Figure 4.21: DPW2: Results of the drag polar study for the Wing/Body condiguration.



(a) Sweep. (b) Polar.

Figure 4.22: DPW2: Results of the drag polar study for the Wing/Body/Nacelle condiguration.



#	y/b
1	0.15
2	0.239
3	0.331
4	0.377
5	0.411
6	0.514
7	0.638
8	0.847

Figure 4.23: Drag Prediction Workshop: Location of the 8 extraction lines. b refers to the wing span.

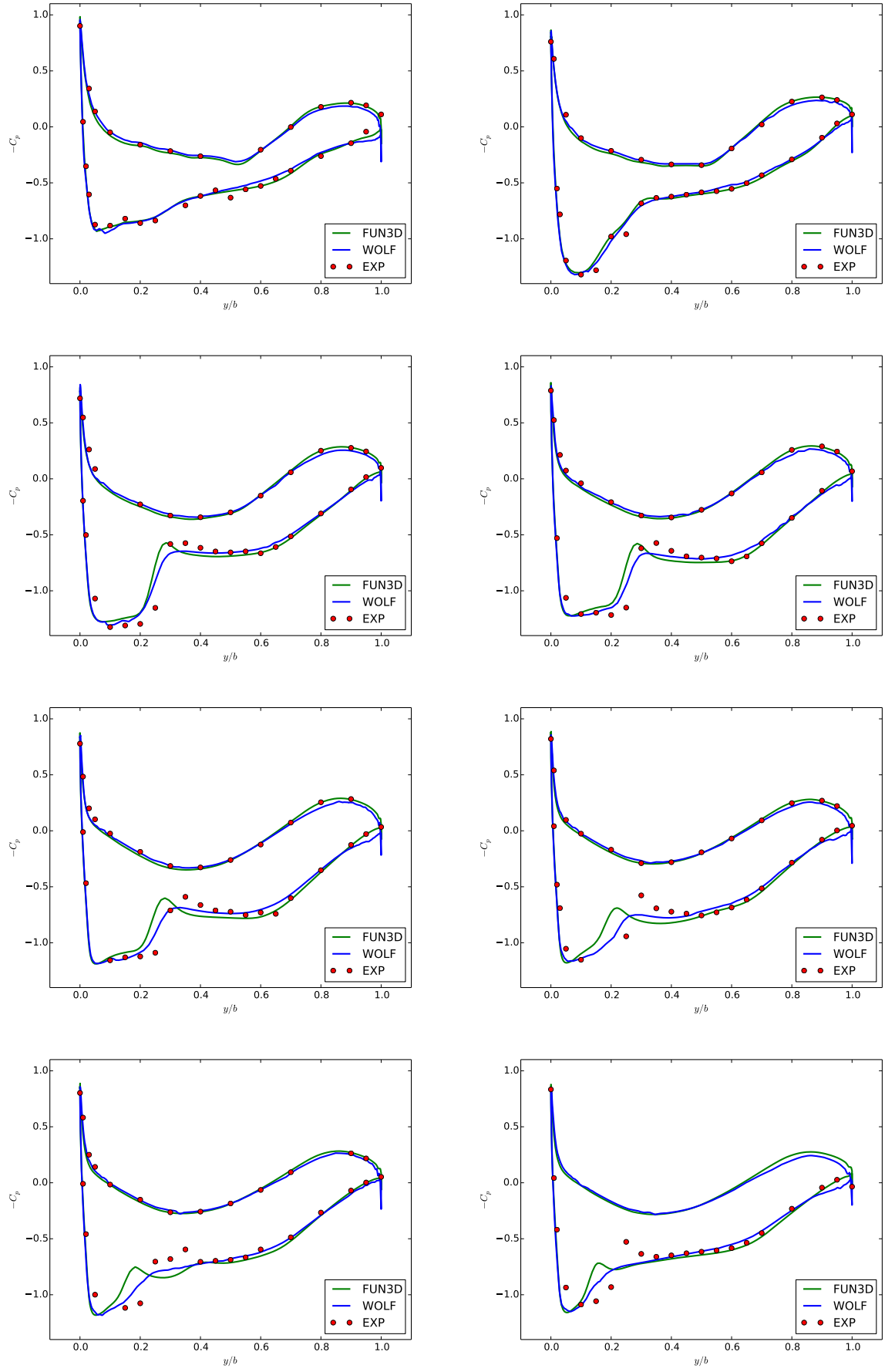


Figure 4.24: DPW2 validation : Wing Body Configuration (sections from 1 to 8).

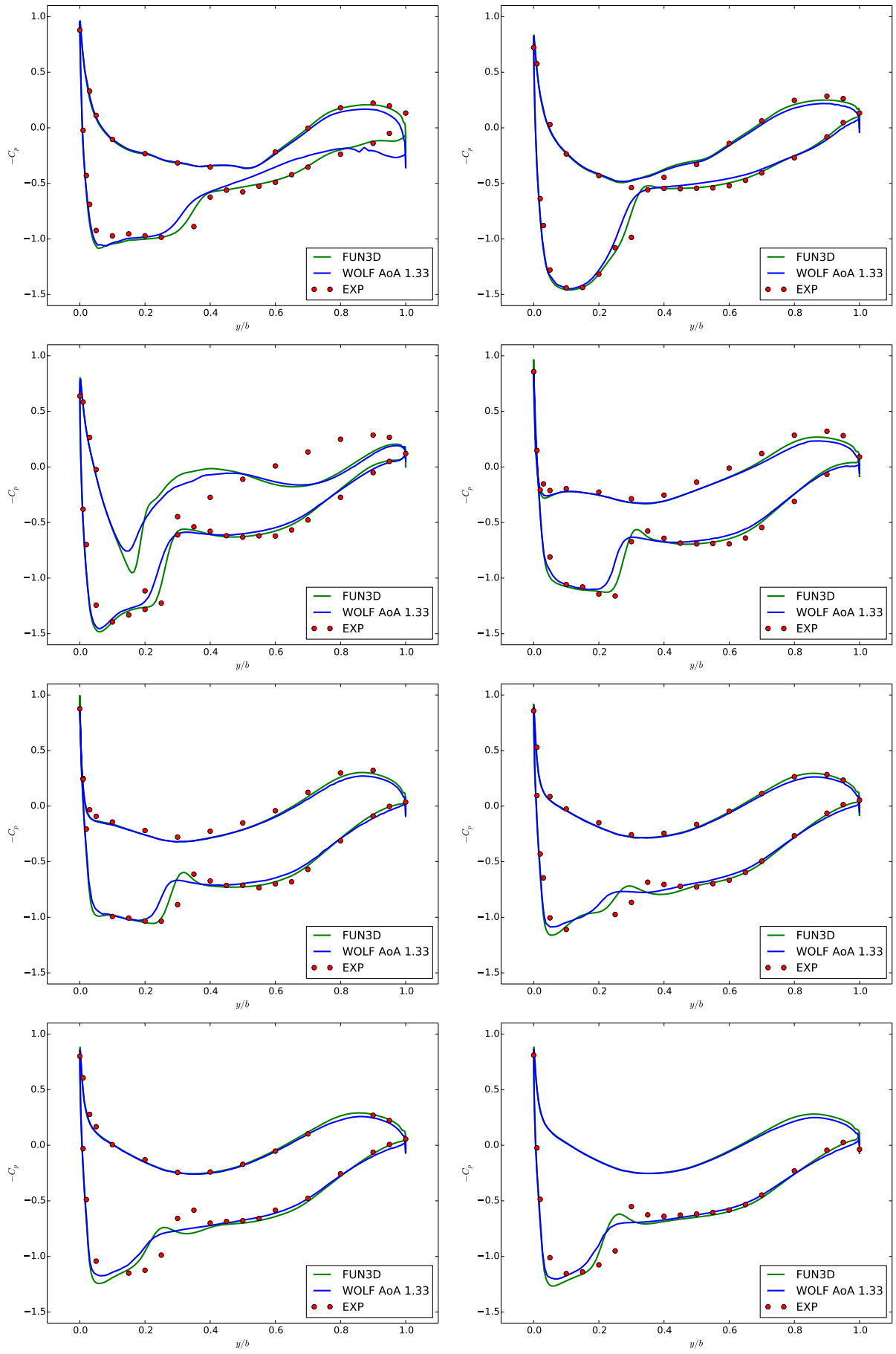


Figure 4.25: DPW2 validation : Wing Body Pylon Nacelle Configuration (sections from 1 to 8).

4.4 Conclusion

In this work, we have carried out a comprehensive V&V study for the RANS flow solver. The validation cases considered span a range of flow regimes pertinent to applications in aeronautics. The results obtained are in good agreement with experimental data, as well as with results from other well-established numerical flow solvers.

Chapter 5

Multigrid acceleration

Contents

5.1 Introduction

In this chapter, we describe the implementation of an implicit multigrid procedure in `Wolf`. As explained in Chapter 3, we employ an implicit time integration, which leads to solving a linear system at each time step. We emphasize the crucial impact of the convergence of the Newton method (used for solving this system) on the success and the wall clock time of the simulation.

5.1.1 Research issues

In this context, we identified two main research issues. The first one is that although classical iterative approaches are suitable for rapidly damping high frequency error components on a given grid, low frequency error components remain and are responsible for the slow convergence, and thus dramatically impact the total wall clock time [118]. The second one is the parameter dependency. Running a CFD simulation is far from just pushing a button, as the user must provide a large set of appropriate parameters: flux reconstruction method, gradient recovery technique, CFL law, limiter function, etc. The choice of this set of parameters is crucial for the success of the simulation and strongly depends on the case. As a consequence, there are some cases where the user might prefer to use more 'secure' parameters (a lower maximal CFL value, for instance), even though it increases the computational time: a slow convergence is better than no solution.

Multigrid methods are commonly used to address these issues. The basic idea behind all multigrid strategies is to accelerate the solution of a set of fine grid equations by computing corrections on coarser grids.

5.1.2 State-of-the-art

Multigrid methods have been widely used for algebraic problems since their original development over thirty years ago [25, 27]. Interest in these methods has since become even greater, thanks to their ability to efficiently solve problems arising from partial differential equations.

Multigrid simulations require a sequence of coarse grid levels, whose generation can be classified into three main categories. The simplest manner to generate coarser meshes is to build a hierarchical set of embedded meshes, which presents serious limitations, one of which is that bad quality elements are created during the process because the grid hierarchy is built starting from the coarser mesh. These elements badly impact the numerical solution [34]. Another method is the volume agglomeration technique, which consists in agglomerating the finite volume cells of the dual mesh [58, 81]. The third approach is the generation of non-nested unstructured coarse meshes [66, 120, 129]. Specific anisotropic coarsening strategies are used for some stiff problems (shock waves, boundary layers, etc.), for which anisotropy causes a breakdown in efficiency [24, 58, 119, 120, 125].

Multigrid methods have been implemented within many well-established numerical flow solvers. Within NSU3D, a line-Jacobi solver is used as a smoother for an agglomeration multigrid solver [121]. Over the past few years, ONERA has been working on extending the multiblock structured solver elsA to hybrid grid configurations, in which an agglomeration multigrid algorithm is embedded [33]. The Stanford solver SU^2 also contains an agglomeration multigrid implementation [136]. A multigrid methodology has been recently developed in the NASA solver FUN3D, which includes both regular and agglomerated coarse meshes [48].

5.1.3 Our approach and contributions

In our multigrid strategy, we generate a set of coarser meshes prior to the simulation, using a non-nested, unstructured coarsening method (isotropic or anisotropic). We chose this coarsening technique (instead of the commonly used agglomeration method) to take advantage of our well-established in-house meshing software, and also because we had in mind to couple multigrid to mesh adaptation (this work in presented in Chapter 6). We improved the coarse grid generation process, by taking into account the preservation of the geometric approximation of the underlying surface.

5.2 Multigrid acceleration in `Wolf`

As detailed in Chapter 3, we use an implicit time integration which implies solving a linear system at each solver iteration. In this chapter, we consider the compressible Euler equations given by System (3.1), where $\mu = \lambda = 0$. The spatial discretization, the implicit time integration as well as the Newton method used for solving the linearized system do not significantly differ from Navier-Stokes (presented in Chapter 3). In the sequel, we describe the multigrid procedure for accelerating the Newton method.

We consider a sequence of N meshes that are generated prior to the simulation (details on coarse mesh generation are provided in Section 5.3):

$$\mathcal{H}_h, \mathcal{H}_{2h}, \mathcal{H}_{4h}, \dots, \mathcal{H}_{2Nh}\,,$$

where $\mathcal{H}_h$ is the initial (and finest) mesh, and $(\mathcal{H}_{2ih})_{(i=1,N)}$ are the coarsened versions of $\mathcal{H}_h$. The linear system obtained in Section 3.5.2 reads (the notations were changed for the sake of clarity):

$$A_h \delta u_h = F_h\,. \tag{5.1}$$

where A_h is the matrix of the linearized system built on $\mathcal{H}_h$ and F_h is the right-hand-side (RHS). The residual of System (5.1) is given by

$$r_h = A_h \delta u_h - F_h\,.$$

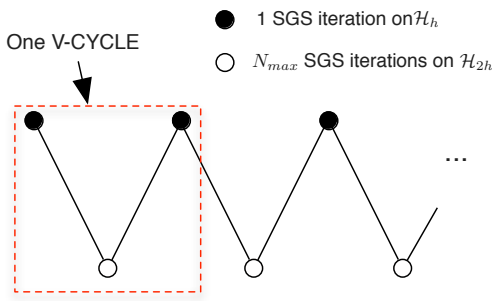


Figure 5.1: Bigrid V-cycle

In the case of a single-grid computation, we have seen that k_{max} (or less) SGS sub-iterations are performed on $\mathcal{H}_h$ in order to reduce r_h by a desired order of magnitude. In the case of a multigrid computation, n_{max} multigrid cycles are performed. One multigrid cycle consists in (i) performing one SGS sub-iteration on $\mathcal{H}_h$, (ii) computing a correction using the coarser meshes, and (iii) adding the computed correction to the solution on $\mathcal{H}_h$. The way this correction is computed depends on the number of coarse meshes involved and on the type of the multigrid cycle used.

Although one multigrid cycle is more costly in terms of CPU (than one single-grid sub-iteration), the number of cycles required to reach the targeted residual is expected to be smaller, thanks to the corrections. Note that the smaller the number of vertices of the coarsest mesh is, the quicker the correction is computed. Moreover, coarser meshes have a strong smoothing property, which increases robustness, i.e. using a multigrid procedure makes it possible to reduce the residual by some orders of magnitude that could not be reached using one single mesh. To summarize, we use a multigrid cycle to compute a correction at each sub-iteration of the Newton method, in order to (i) increase the convergence speed, while (ii) improving the robustness.

We now describe the three different types of multigrid cycles we use to compute corrections: the V-cycle, the W-cycle, and the F-cycle. We start by explaining the case of the two-grid V-cycle, which only requires one coarser mesh. Then, the three types of cycles are introduced in the general case of N meshes.

5.2.1 Two-grid V-cycle

The two-grid V-cycle requires a mesh $\mathcal{H}_h$ and a coarser mesh $\mathcal{H}_{2h}$. We suppose that N time steps were performed by the flow solver. Let

$$A_h \delta u_h = F_h$$

be the linear system obtained after the N-th time step. In order to accelerate the convergence of the Newton method to solve this linear system, a given number of multigrid cycles can be performed. The bigrid V-cycle (see Figure 5.1) consists in computing a correction by performing several SGS iterations on $\mathcal{H}_{2h}$.

A_h, the matrix of the linearized system, was built on $\mathcal{H}_h$ as explained in Chapter 3 (see Section 3.5.2). A_{2h} was built in a similar way on $\mathcal{H}_{2h}$ after $\mathcal{S}_h$, the solution obtained on $\mathcal{H}_h$ after N time steps, was linearly interpolated to $\mathcal{H}_{2h}$. Starting from an initial solution δu_h^0, a pre-smoothing is performed on $\mathcal{H}_h$, i.e. one SGS iteration. Note that when the multigrid cycle is the first of the current time step, δu_h^0 is set to $\mathbf{0}$, and otherwise δu_h^0 is the solution of the previous cycle. Let δu_h^1 be the solution obtained after the pre-smoothing, the residual is computed:

$$r_h = A_h \delta u_h^1 - F_h$$

and is restricted to $\mathcal{H}_{2h}$. The restriction operator $R_{h\to 2h}$ first consists in locating each vertex P_h of $\mathcal{H}_h$ in $\mathcal{H}_{2h}$, i.e. identifying the element $K_{2h} = (P_{2h}(i))_{i=0,3}$ of $\mathcal{H}_{2h}$ containing P_h. Then, the restricted residual is summed to the vertices of K_{2h}:

$$R_{h\to 2h}(r_h)(P_{2h}(i)) \quad += \quad \beta_i \times r_h(P_h) \quad \text{for} \quad i = 0, 3\,,$$

where β_i is the barycentric coordinate of P_h in K_{2h} associated to $P_{2h}(i)$.

The correction c_{2h} is then computed on $\mathcal{H}_{2h}$ by using $R_{h\to 2h}(r_h)$ as the RHS. The initial correction is set to 0 : $c_{2h}^0 = \mathbf{0}$ and a given number of SGS iterations is performed:

$$A_{2h} c_{2h}^0 = R_{h\to 2h}(r_h) \tag{5.2}$$

Then, the resulting correction c_{2h}^1 is linearly interpolated to $\mathcal{H}_h$ and added to the solution:

$$\delta u_h^2 = \delta u_h^1 + I_{2h\to h}(c_{2h}^1)\,.$$

So, at each time step of the flow solver, the corrections added after each multigrid cycle are expected to improve the convergence of the Newton method, and thus to improve the convergence of the whole simulation. The number of multigrid cycles required to reach the targeted residual of the linear system is expected to be smaller than the required number of SGS iterations on $\mathcal{H}_h$ in the single-grid case. At a given solver time step, the best convergence of the Newton method is reached using an ideal bigrid V-cycle.

Ideal Bigrid V-cycle. The ideal bigrid V-cycle consists in performing a large number of SGS iterations on $\mathcal{H}_{2h}$ in order to obtain a fully converged correction c^1_{2h}. This ideal bigrid V-cycle is obviously too costly in terms of CPU to be used for real-life simulations, but since the linear system is converged to its maximum on $\mathcal{H}_{2h}$, it provides the best correction that can be obtained using a multigrid cycle. The number of iterations needed by an ideal bigrid V-cycle to converge the linear system on the finest mesh $\mathcal{H}_h$ can thus be targeted when using another multigrid cycle (using more mesh levels). In other words, a "good" multigrid cycle aims at requiring as few iterations as the ideal bigrid cycle to decrease the residual on $\mathcal{H}_h$, while being less costly in terms of CPU thanks to the use of more coarser mesh levels.

5.2.2 N-grid V-Cycle, W-Cycle and F-Cycle

The N-grid V-Cycle is simply the extension of the bigrid V-cycle to N grids. Only one SGS iteration is performed on $\mathcal{H}_{2h}$ and the residual

$$r^1_{2h} = A_{2h}c^1_{2h} - r_{2h}$$

is computed and restricted to $\mathcal{H}_{4h}$. $R_{2h\to 4h}$ is then used as the RHS to compute a correction c^1_{4h} on $\mathcal{H}_{4h}$. This is how a correction is computed on each coarse mesh. Once on the coarsest mesh $\mathcal{H}_{2Nh}$, not one but several SGS iterations are used to compute c^1_{2Nh}, which is not costly in terms of CPU due to the low number of vertices. Then, the correction of the coarsest mesh is interpolated and added to the correction of the second coarsest mesh and so on. In the end, the final correction containing all the contributions of the coarser meshes is interpolated on the finest mesh and added to δu^1_h. A post-smoothing (i.e. one SGS iteration) can be performed on each level i after $I_{2(i+1)h\to 2ih}(c_{2(i+1)h})$ has been added to c_{2ih}.

Other types of multigrid cycles may be used, such as the W- and the F-cycle. The structures of these three cycles is depicted in Figure 5.2 for the case of four grids, and a 5-grid W-cycle is depicted in Figure 5.3.

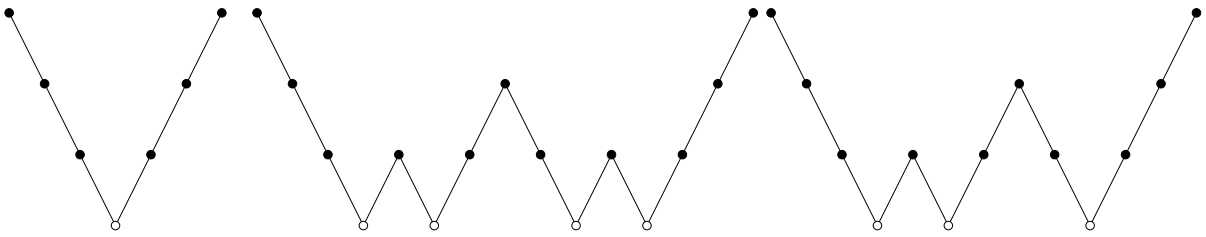


Figure 5.2: Four-grid methods : V-cycle, W-cycle and F-cycle (● : 1 smoothing SGS iteration, ○ : Several SGS iterations)

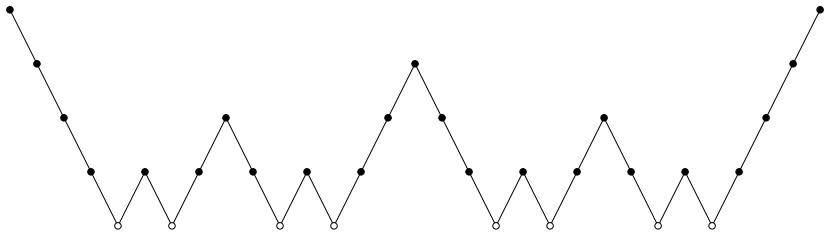
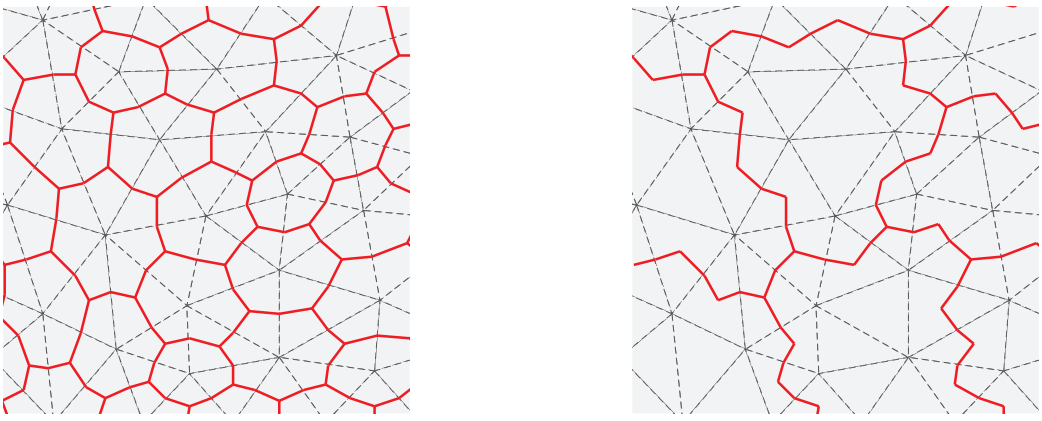


Figure 5.3: A five-grid method: W-cycle (• : 1 smoothing SGS iteration, ∘ : Several SGS iterations)

5.3 Generation of coarse grids

This section describes how we generate a hierarchical set of coarser meshes, with consideration for the preservation of a good geometric approximation. Starting from an initial finest mesh $\mathcal{H}_h$ (whose representative edge size is h), we want to generate coarser meshes $\mathcal{H}_{2h}$, $\mathcal{H}_{4h}$, $\mathcal{H}_{8h}$ etc. suitable for a multigrid computation. First, we briefly review commonly used coarsening methods, which can be classified into three main categories: agglomeration, nested meshes and non-nested meshes.

Agglomeration techniques. These methods consist in agglomerating the finite volume cells of the dual mesh [58, 81] (as illustrated in Figure 5.4), while maintaining the quality of the finer grid in the agglomerated levels as much as possible. Although these techniques provide good results, we chose the geometric multigrid technique, which consists in generating a coarser mesh, whose edges are twice as large as the finer one (the dual mesh is then built on this new discretization).

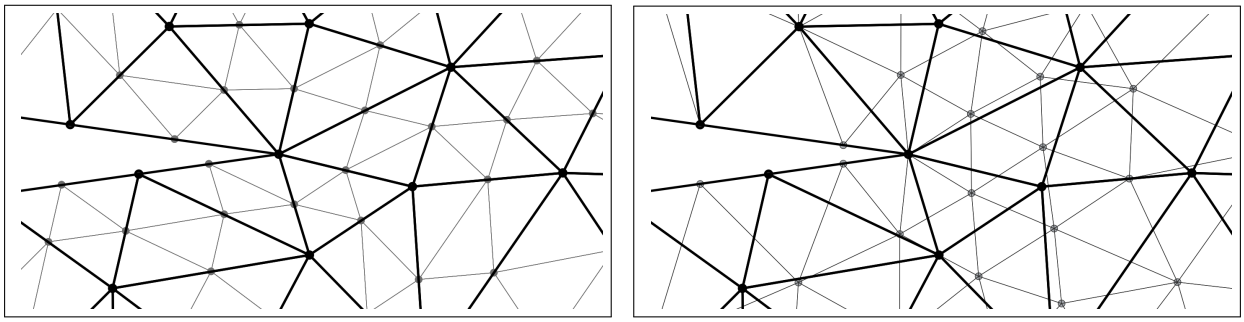


(a) Initial dual mesh. (b) Agglomerated dual mesh.

Figure 5.4: Illustration of agglomeration techniques. The dual mesh (in red) is built on the initial mesh (dashed lines) and cells are merged.

Remark. *The choice for this coarsening method (rather than agglomeration) is motivated by two reasons. First, we want to benefit from all our in-house meshing software developed over the years. Second, we want to couple multigrid algorithms with adaptive methods (see Chapter 6), which consists in using adapted meshes as coarser meshes.*

Generation of nested meshes. Generating embedded meshes is a simple geometric way to build a hierarchical nested set of coarse meshes for multigrid methods, such as depicted in Figure 5.5. It consists in first generating an initial coarse mesh, which is then refined by element subdivision. This method has a major drawback, since the quality of the meshes generated decreases as they are refined. Indeed, as element subdivisions are iteratively performed, patterns corresponding to the coarsest mesh elements appear. These patterns may influence the computation, as they can act as artificial internal boundaries [34], as illustrated in Figure 5.6.



(a) $\mathcal{H}_h$ and $\mathcal{H}_{2h}$ are nested. (b) Non-nested meshes.

Figure 5.5: Comparison between nested and non-nested meshes. The coarse mesh $\mathcal{H}_{2h}$ (in black) and the finer mesh $\mathcal{H}_h$ (in grey) are juxtaposed for visualization purposes.

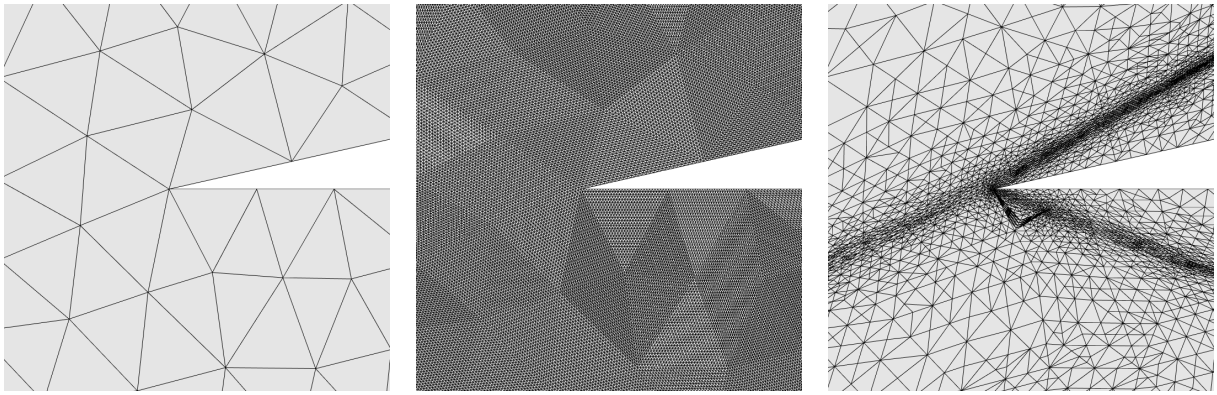


Figure 5.6: Close-up views of three nested meshes of a 2D scramjet. Left: the initial coarse mesh. Middle: all the edges of the mesh were refined. Right: only some edges were refined. Both nested meshes (middle and right) contain elements of bad quality due to the constraints from the initial discretization.

Unstructured non-nested meshes. This is the method we chose. The metric field representing the finer mesh is scaled and the coarser mesh is generated according to this new scaled metric, without necessarily generating nested elements. This method ensures a good geometric approximation and a good mesh quality in every coarse levels. Note that stiff problems (shock waves, boundary layers, etc.) can cause a breakdown in the efficiency of multigrid methods [119], due to high anisotropy. This is faced using

specific anisotropic coarsening strategies [24, 58, 119, 120, 125]. We now describe this last approach, in both isotropic and anisotropic cases.

5.3.1 Isotropic and anisotropic scaling of the metric

The first step of mesh coarsening is to compute the metric $\mathcal{M}_{geo}(\mathcal{H}_h)$, for which $\mathcal{H}_h$ is unit (see Chapter 1). A geometric representation of $\mathcal{M}_{geo}$ at a vertex is depicted in Figure 5.7: it contains representative information on the sizes and anisotropy of the elements of $\mathcal{H}_h$. First we describe how $\mathcal{M}_{geo}(\mathcal{H}_h)$ is computed, then how we multiply it by a scaling factor c to generate coarser meshes.

Remark. *If available, we prefer to use the metric provided as an output by the (re)mesher used to generate the initial fine mesh, instead of re-computing the metric from the discretization. The reason for that is the possible presence of bad quality elements due to the inability of the (re)mesher to meet the metric requirements. This way, we do not depend on the initial discretization but only on the initial desired continuous metric. So, irrelevant edge sizes are not propagated to the coarser levels.*

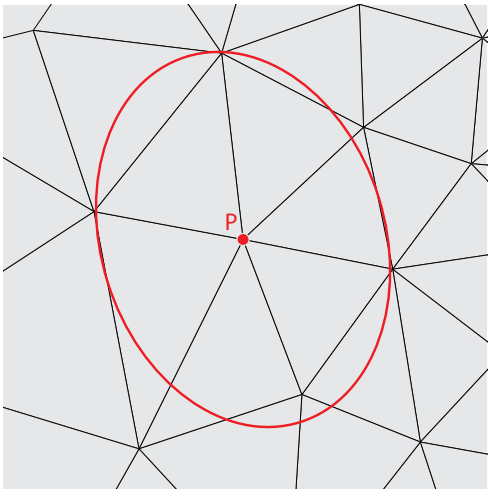


Figure 5.7: In red: geometric representation of $\mathcal{M}_{geo}$ around P.

Computing the geometric metric. If the aforementioned metric is not available, we compute the geometric of the mesh using the initial discretization. Given a mesh element $K = (\mathbf{e_i})_{i=1..n(n+1)/2}$ such that its volume is positive, we can show that there is only one metric $\mathcal{M}$ such that K is unit according to $\mathcal{M}$. To do so, one must solve the following linear system:

$$(S) \begin{cases} \ell^2_{\mathcal{M}}(\mathbf{e_1}) = 1 \\ \vdots \\ \ell^2_{\mathcal{M}}(\mathbf{e_6}) = 1\,. \end{cases} \tag{5.3}$$

The determinant of (S) being equal to the volume $|K|_{I_3} \neq 0$, there exists a unique solution. The algorithm for computing $\mathcal{M}_{geo}$ consists in two steps:

1. For each mesh element K, compute $\mathcal{M}_{geo,K}$ by solving system (5.3).
2. The metric at the elements is projected onto the vertices of the mesh using an averaging weight using the volume of elements:

$$\mathcal{M}_{geo,P} = \exp\left(\frac{\sum_{P\in K} |K|_{I_3} \ln(\mathcal{M}_K)}{\sum_{P\in K} |K|_{I_3}}\right).$$

Scaling of the geometric metric. A breakdown in efficiency of multigrid methods can be observed when dealing with high anisotropy, such as boundary layer mesh elements or stretched elements in shock directions. In order to prevent this breakdown in efficiency, existing isotropic coarsening techniques [66, 120] were extended to the anisotropic case [24, 58, 119, 120, 125]. We introduce both cases (which are illustrated in Figure 5.8).

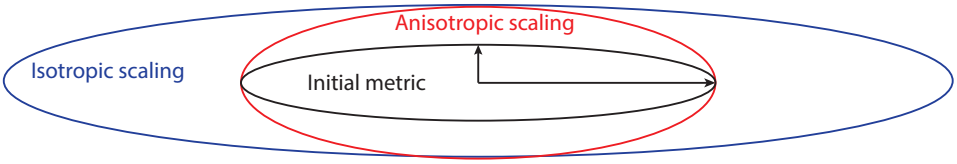


Figure 5.8: Geometric illustration of the isotropic and the anisotropic scaling of the initial metric.

Let $\mathcal{M}$ be the metric tensor of $\mathcal{M}_{geo}(\mathcal{H}_h)$ associated to a vertex P, and $\lambda_1, \lambda_2, \lambda_2$ its eigenvalues, and h_1, h_2, h_3 the corresponding sizes ($h_i = (\lambda_i)^{-1/2}$). We denote by c the scaling factor: $c = 2$ for $\mathcal{H}_{2h}$, 4 for $\mathcal{H}_{4h}$ etc.

- **Isotropic coarsening:** The same scaling factor is applied to all directions of the metric:

$$h_i^{new} = c \times h_i\,,$$

- **Anisotropic coarsening:** The general idea is to coarsen the mesh only in the directions that are perpendicular to a direction of anisotropy. This coarsening leads to increasing the smallest size of each element until it is isotropic, then the isotropic scaling is applied. To do so, only the scaling of the geometrical metric $\mathcal{M}_{geo}(\mathcal{H}_h)$ differs from the isotropic case.

  We chose to compute the anisotropic scaled metric tensor field of $\mathcal{M}_{geo}(\mathcal{H}_h)$. We start by ordering the sizes:

$$h_1 \leq h_2 \leq h_3\,.$$

  Then, the new coarsened sizes are computed:

1. $h_1^{new} = ch_1$
2. $h_2^{new} = \max(h_2, \min(ch_2, h_1^{new}))$
3. $h_3^{new} = \max(h_3, \min(ch_3, h_2^{new}))$.

Applied to isotropic elements, the anisotropic coarsening is equivalent to the isotropic coarsening. Applied to anisotropic elements, however, only the directions where the mesh size is minimal are scaled.

Once the metric has been scaled, the coarser mesh is generated using the anisotropic remesher described in Chapter 1. An example of coarsening is depicted in Figure 5.9.

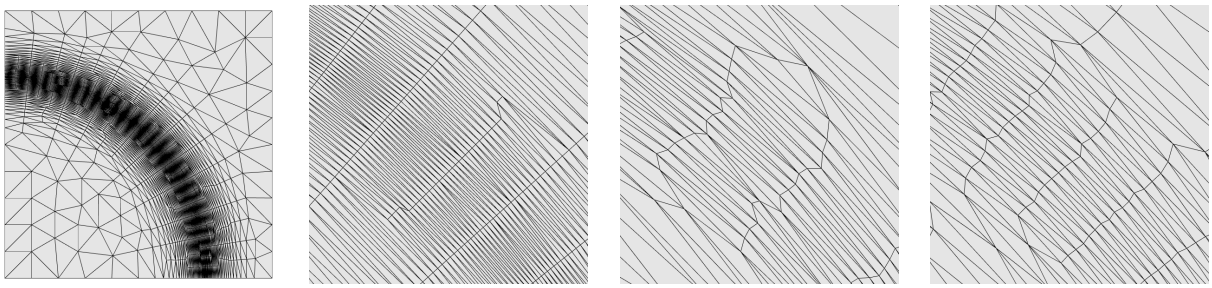
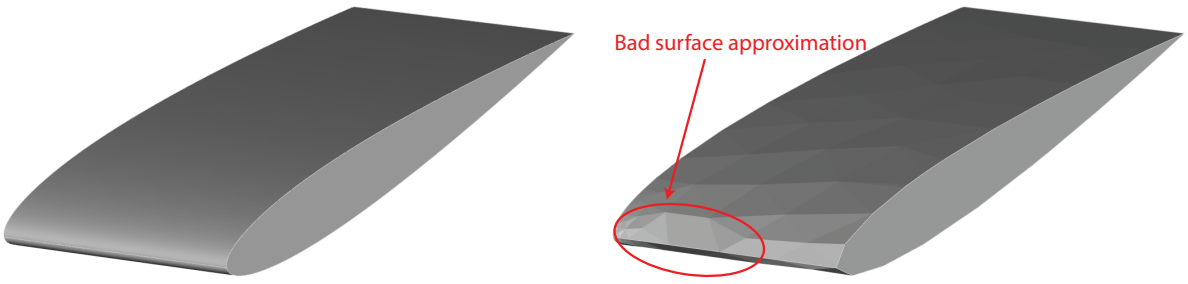


Figure 5.9: Example of mesh coarsening. From left to right: initial mesh (large and close-up views), isotropic coarsening and anisotropic coarsening.

5.3.2 Preserving the geometric approximation

This section presents how we enhance the coarsening procedure in order to preserve the geometric approximation. In Figure 5.10 for instance, an initial mesh of a 3D airfoil is coarsened using an isotropic scaling of the initial geometric metric.



(a) Initial surface. (b) Surface of the coarse mesh.

Figure 5.10: 3D airfoil: a basic coarsening does not preserve the geometric approximation.

Starting from an initial fine mesh $\mathcal{H}_h$, we list below the different steps of the enhanced method.

1. Compute the scaled metric $\mathcal{M}_{2h}$, as described in Section 5.3 (iso- or anisotropic coarsening).

2. Compute a metric $\mathcal{M}_{surf}$ that preserves the geometric approximation.

3. Intersect $\mathcal{M}_{2h}$ and $\mathcal{M}_{surf}$.

4. Generate $\mathcal{H}_{2h}$ using the intersected metric.

Surface metric computation

We describe the construction of $\mathcal{M}_{surf}$, the surface metric that preserves the geometric approximation. It consists in two steps: first we compute a quadric model of the surface, then we compute its principal curvatures and use them to build $\mathcal{M}_{surf}$.

Quadric surface model. Following the approach described in [59], we compute a quadric surface model around each surface vertex $\mathbf{P}_i$. First, a normal vector $\mathbf{n}_i$ and orthogonal tangent vectors $(\mathbf{u}_i, \mathbf{v}_i)$ are assigned to each $\mathbf{P}_i$. Then, the topological neighbors $\mathbf{P}_j$ of $\mathbf{P}_i$ are mapped onto the local orthonormal Frenet frame $(\mathbf{u}_i, \mathbf{v}_i, \mathbf{n}_i)$ centered in $\mathbf{P}_i$. We denote by $(u_j, v_j, \sigma_j) = ({}^t\mathbf{P}_j.\mathbf{u}_i, {}^t\mathbf{P}_j.\mathbf{v}_i, {}^t\mathbf{P}_j.\mathbf{n}_i)$ the new coordinates of vertex $\mathbf{P}_j$. $\mathbf{P}_i$ is set as the new origin, so $(u_i, v_i, \sigma_i) = (0, 0, 0)$. We compute the quadric surface using the following least square approximation:

$$\sigma(u, v) = au^2 + bv^2 + cuv, \text{ where } (a, b, c) \in \mathbb{R}^3. \tag{5.4}$$

The least square problem gives the solution minimizing

$$\min_{(a,b,c)} \sum_{j \in \mathcal{V}(\mathbf{P}_i)} |\sigma_j - \sigma(u_j, v_j)|^2,$$

where $\mathcal{V}(\mathbf{P}_i)$ is the set of all neighboring vertices of $\mathbf{P}_i$. Note that 3 neighbors points are necessary to recover the surface model.

In order to add more information to the surface model construction, mid-edge points $\mathbf{P}_m$ are recovered from the following quadratic formula:

$$\begin{aligned} &\mathbf{P}_m = (1-t)^2(1+2t)\mathbf{x}_1 + t(1-t)^2\mathbf{r}_1 + t^2(3-2t)\mathbf{x}_2 - t^2(1-t)\mathbf{r}_2, \text{ with} \\ &\mathbf{r}_i = \|\mathbf{e}\|_2 \frac{\mathbf{n}_i \times (\mathbf{e} \times \mathbf{n}_i)}{\|\mathbf{n}_i \times (\mathbf{e} \times \mathbf{n}_i)\|_2} \text{ and } t \in [0, 1], \end{aligned} \tag{5.5}$$

where $\mathbf{e}$ is an edge issued from $\mathbf{P}_i$ and $\mathbf{P}_j$ a neighbor of $\mathbf{P}_i$. Finally, let d be the number of neighbors of

$\mathbf{P}_i$, we solve the following linear system that involves the d neighbors and the d mid-points:

$$A\,X \;=\; B \iff \begin{pmatrix} u_1^2 & v_1^2 & u_1 v_1 \\ \vdots & \vdots & \vdots \\ u_d^2 & v_d^2 & u_d v_d \\ u_{\frac{1}{2}}^2 & v_{\frac{1}{2}}^2 & u_{\frac{1}{2}} v_{\frac{1}{2}} \\ \vdots & \vdots & \vdots \\ u_{\frac{d}{2}}^2 & v_{\frac{d}{2}}^2 & u_{\frac{d}{2}} v_{\frac{d}{2}} \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} \sigma_1 \\ \vdots \\ \sigma_d \\ \sigma_{\frac{1}{2}} \\ \vdots \\ \sigma_{\frac{d}{2}} \end{pmatrix},$$

where (u_m, v_m, σ_m) are mid-points local coordinates recovered using (5.5). The least square formulation consists in solving ${}^tA\,A \;=\; {}^tA\,B$, which gives the quadric surface approximation. From this quadric surface, we now compute the local principal curvatures and use them to construct $\mathcal{M}_{surf}$.

Computation of the surface metric. Once the quadric surface is locally defined at each mesh vertex P, it makes it possible to compute its local principal curvatures κ_1 and κ_2, as well as its principal directions $\mathcal{D}(\mathbf{P})$. The principal curvatures at a point $\mathbf{P}$ make it possible to characterize the local behavior of the surface: $\mathbf{P}$ is elliptic if $\kappa_1\kappa_2 > 0$, hyperbolic if $\kappa_1\kappa_2 < 0$, and parabolic if $\kappa_1\kappa_2 = 0$.

This local information about the surface is used to compute $\mathcal{M}_{surf}$, a geometric metric that preserves the point characteristics. $\mathcal{M}_{surf}$ is constructed in the tangent plane of the surface mesh. It is defined by a matrix of the form:

$$\mathcal{M}_{surf}(\mathbf{P}) = {}^t\mathcal{D}(\mathbf{P}) \begin{pmatrix} \frac{1}{\alpha^2 \rho_1^2(\mathbf{P})} & 0 & 0 \\ 0 & \frac{1}{\beta^2 \rho_2^2(\mathbf{P})} & 0 \\ 0 & 0 & \lambda \end{pmatrix} \mathcal{D}(\mathbf{P}),$$

where $\mathcal{D}(\mathbf{P})$ are the principal directions at P, $\rho_1 = 1/\kappa_1$, $\rho_2 = 1/\kappa_2$ are the main radii of curvature, α and β are appropriate coefficients, and $\lambda \in \mathbb{R}$ provides an anisotropic (curvature-based) control of the geometry.

The local size of this metric is proportional to the principal radii of curvature. Let ϵ be a parameter provided by the user, that bounds the gap between any mesh element and the underlying surface. Setting a constant ϵ leads for instance to fixing:

$$\alpha = 2\sqrt{\epsilon(2-\epsilon)}$$

and to defining:

$$\beta = 2\sqrt{\epsilon\frac{\rho_1}{\rho_2}(2-\epsilon\frac{\rho_1}{\rho_2})}\,.$$

5.4 Multigrid validation

This section presents the validation of the multigrid procedure, using three test cases: 2D transonic NACA 0012, 3D subsonic NACA 0012 and 3D transonic WBT configuration.

First, we make sure that multigrid corrections accelerate the convergence of the Newton method for the resolution of the linear system at each time step, and that it has an impact on the convergence of the whole simulation. Finally, we carried out a parameter dependency study.

Remark. *It is important to distinguish the two kinds of iterations: (i) iteration of the Newton method, which corresponds to an SGS iteration in single-grid, and to a multigrid cycle in multigrid, and (ii) iteration of the flow solver, which corresponds to a time step. In the sequel, we refer to (i) as a 'Newton iteration', and to (ii) as 'time step'.*

Acceleration of the Newton method. To analyze the benefits of the multigrid strategy on the Newton method, a solution is "almost" converged on the finest mesh by performing N time steps using an adequate CFL law. The evolution of the residual of the solving of the linear system obtained at time step N is then compared for the single-grid method, the ideal bigrid, and the three aforementioned cycles (V-, W- and F-cycles, using various numbers of mesh levels). At time step N, a high CFL is prescribed in order to evaluate the robustness of each method.

Impact on the whole simulation. Starting from an initial uniform state, the convergence of the solution in terms of CPU and the number of solver iterations is compared.

Parameter depency study. The idea is to launch a set of simulations with different input parameters (maximal CFL value for instance) and to evaluate the dependency of the solver on these parameters. In particular, we compare their impact on the global convergence of the simulation for both single and multigrid simulations.

5.4.1 Description of the test cases

2D transonic NACA 0012. We consider a transonic flow (Mach number $M = 0.8$, angle of attack $\alpha = 1.25$) around a NACA 0012 geometry. We used the four meshes presented in Figure 5.11 for the multigrid computations. This series of meshes was generated using an isotropic scaling of the metric field representing the finest mesh, see Section 5.3. A view of the solution is depicted in Figure 5.12.

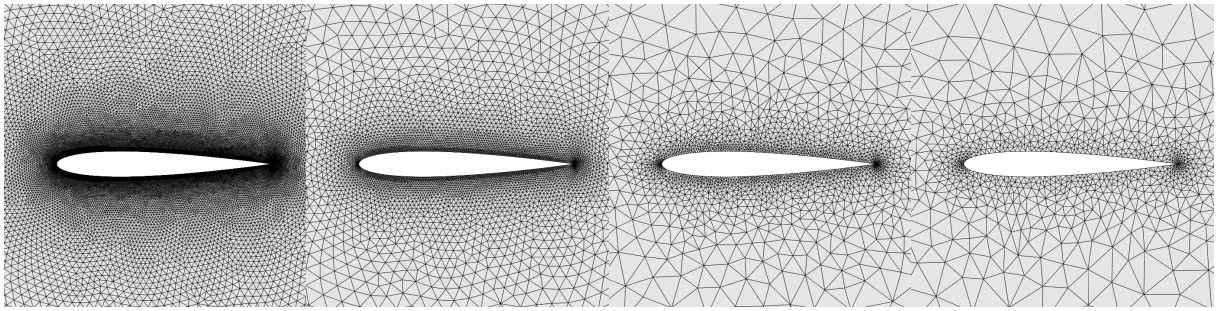


Figure 5.11: Close-up views of the four meshes used during the multigrid computations of the 2D transonic NACA. Number of vertices, from left to right: 29 024, 7 379, 2 499 and 1 305.

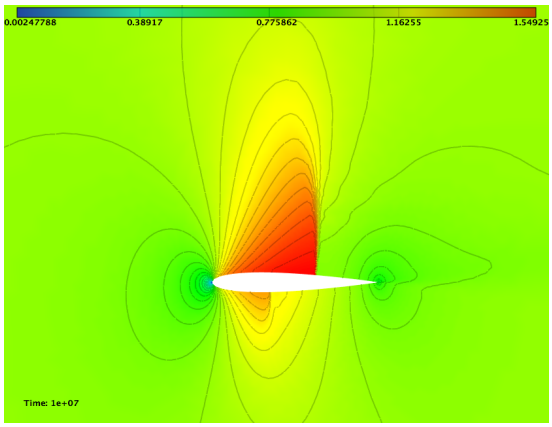


Figure 5.12: 2D transonic NACA 0012: pressure.

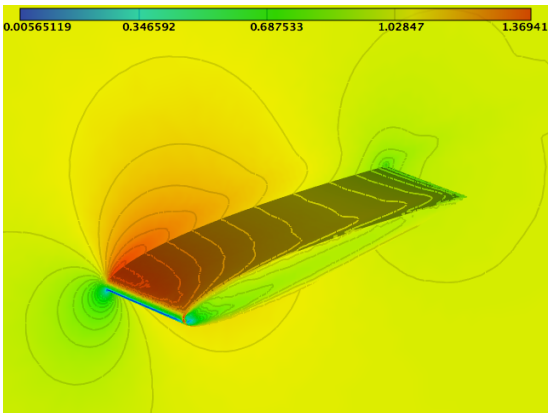


Figure 5.13: 3D subsonic NACA 0012: Solution (velocity).

3D subsonic NACA 0012. This geometry was generated by extruding the 2D airfoil along a linear path. A subsonic flow is prescribed (Mach $M = 0.4$, angle of attack $\alpha = 4°$). The series of meshes used is presented in Figure 5.14, and the solution in Figure 5.13.

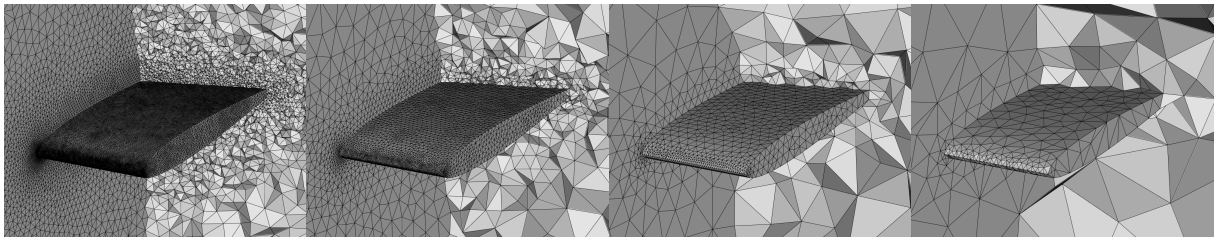


Figure 5.14: 3D subsonic NACA 0012 : Cuts in the volumes of the four meshes used during the multigrid computations. Number of vertices, from left to right: 271 311, 34 452, 4 940, 1 007.

3D Wing Body Tails (WBT) configuration. This industrial configuration (see Figure 5.22) includes a wing, a body and two tails (horizontal and vertical). A transonic flow is computed, using prescribed Mach number $M = 0.8$ and an angle of attack $\alpha = 1°$. Three meshes used for the multigrid

computations are presented in Figure 5.15.

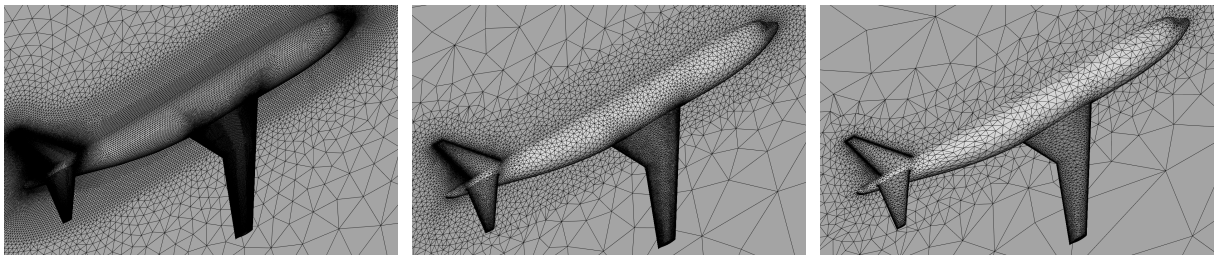


Figure 5.15: 3D WBT configuration : The three meshes used for multigrid simulations.

5.4.2 Resolution of the Linear System

The multigrid acceleration for solving the linear system is compared for the 2D transonic NACA 0012 airfoil test case.

Method. A solution (see Figure 5.12) is computed on the finest mesh by performing 120 time steps at $\text{CFL}_{max} = 10$. Then, the resulting solution (which is 'almost' converged) is used as a restart solution and a time step at CFL = 1000 is performed in order to compare the convergence of the linear system in terms of the number of iterations and in terms of wall clock time. This convergence is compared for the different methods: single-grid, ideal bigrid, and the three aforementioned cycles (V-, W- and F-cycles).

Results. Figure 5.16 presents the convergence rates obtained using one single-grid, an ideal bigrid, and 3 V-cycles (using 3, 4 and 5 meshes). All the multigrid methods manage to decrease the initial residual by twelve orders of magnitude, while in the same CPU time interval, the single-grid computation fails to decrease it by one order due to the high CFL. As expected, the ideal bigrid shows the fastest convergence in terms of the number of iterations but is also the slowest method in terms of CPU. Figure 5.17 presents a comparison of the three different 4 grid cycles used (V, W and F). Although both the W-cycle and the F-cycle are really close to the ideal bigrid in terms of the number of iterations, they are slower than the V-cycle in terms of CPU. To summarize, the fastest convergence for the transonic NACA is the 4-grid V-cycle in terms of CPU, and the 4-grid F-cycle in terms of the number of iterations.

5.4.3 Impact on the Whole Simulation

The evolution of the residual after each time step is compared for the single-grid method and several multigrid cycles. Starting from the uniform solution, the number of time steps needed to reach a targeted residual is compared for three test cases : 2D transonic NACA 0012, 3D subsonic NACA 0012 and 3D transonic WBT configuration.

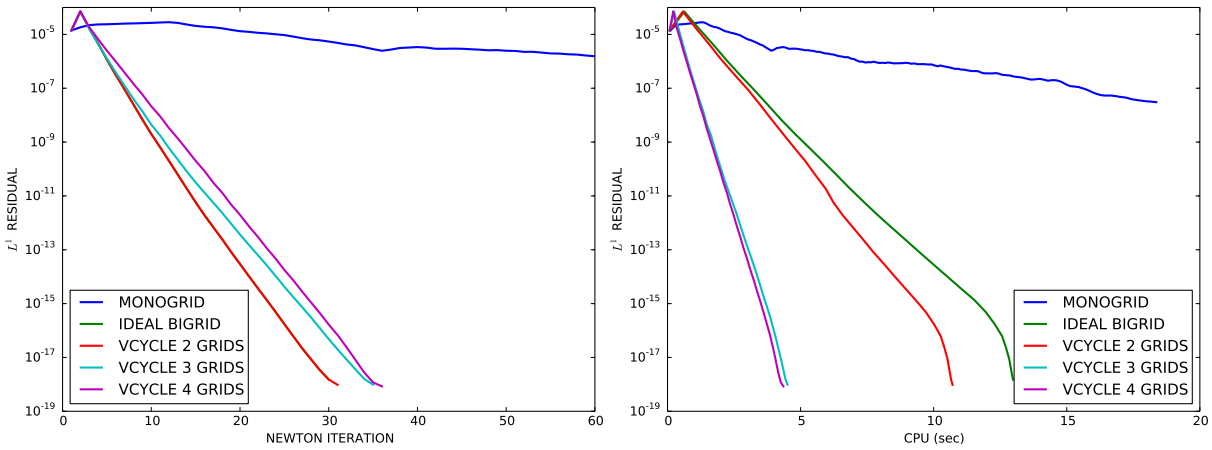


Figure 5.16: Transonic NACA: Comparison of the V-cycles for the convergence of the Newton method after 120 time steps.

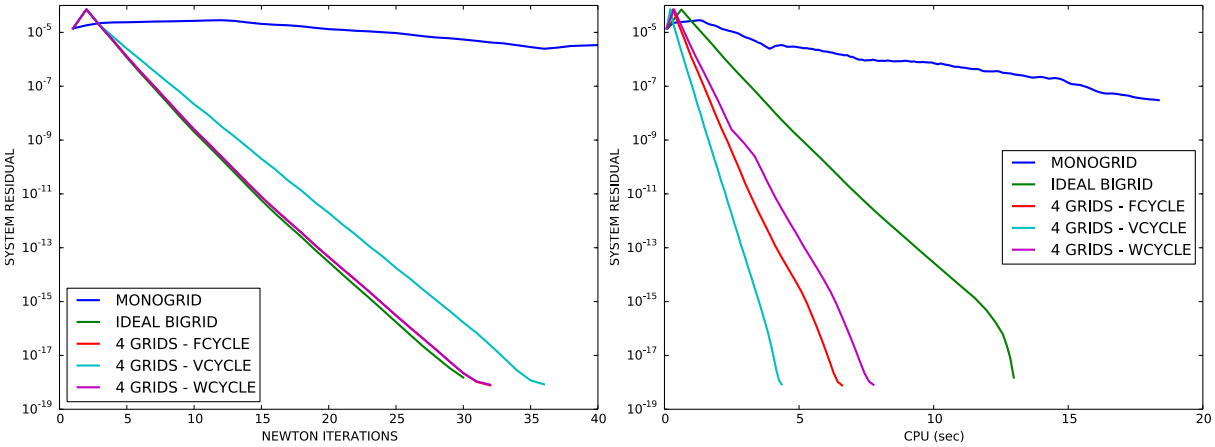


Figure 5.17: Transonic NACA: Comparison of the V, the W and the F cycles (using 4 grid levels) for the convergence of the Newton method after 120 time steps.

2D transonic NACA 0012. This time, the convergence of the residual in terms of the number of time steps is considered. Figure 5.18 presents a comparison between the single-grid and the multigrid methods. Multigrid methods improve the convergence rate in terms of both the number of iterations and wall clock time. As regards the number of iterations, the best convergence rate is obtained in the single-grid case by performing 40 SGS sub-iterations, and in the multigrid case using a 3-grid V-cycle. The fastest methods in terms of CPU are the 3-grid and 4-grid V-cycles.

3D subsonic NACA 0012. This case is interesting because the convergence of the residual of the whole simulation greatly depends on the Newton method. As shown in Figure 5.19, no fewer than 25 SGS sub-iterations are required in the single-grid case to reduce the residual of the whole simulation by the desired order of magnitude (10^{-9} is the target). Figure 5.21 shows that only one V-cycle is enough, and that performing two cycles is enough to obtain an optimal residual convergence in terms of the

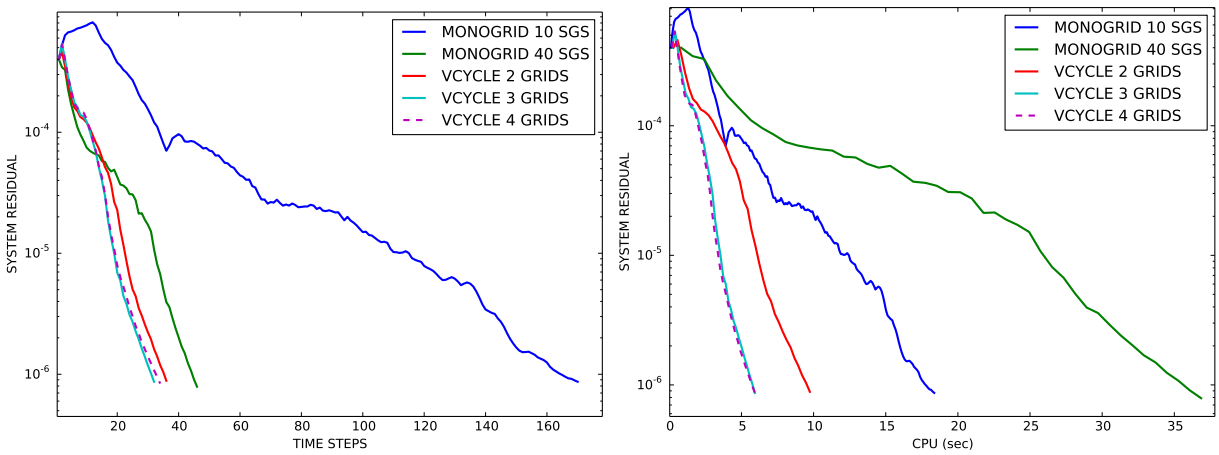


Figure 5.18: 2D transonic NACA: Convergence of the residual of the whole simulation. Left: number of time steps. Right: wall clock time (sec).

number of iterations, i.e. performing more than two cycles does not help to increase the convergence rate. Figure 5.21 also presents a comparison between the most efficient single-grid method (i.e. 25 SGS sub-iterations), and the multigrid. As concerns the number of iterations, the residual convergence of the optimal single-grid method (25 SGS sub-iterations) and the optimal multigrid method (2 V-cycles) are identical. The wall clock time, however, drops from 6m50s in the single grid case to 1m54s using one V-cycle.

3D transonic WBT configuration. The results are presented in Figure 5.23. For this simulation, a dynamic CFL law was prescribed, using 1 000 as the maximal CFL value. But for $CFL_{max} = 1000$, the single-grid approach failed (green plot) and we had to reduce it to $CFL_{max} = 100$. The multigrid approach appears to be more robust, as it converges using $CFL_{max} = 1000$. Moreover, a significant gain in CPU is obtained, as the total CPU time drops from 16 min to 4 min.

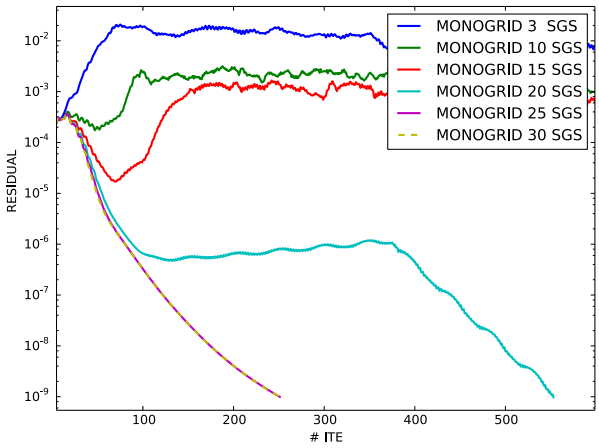


Figure 5.19: 3D subsonic NACA 0012: At least 25 SGS sub-iterations are required in the single-grid case to converge the residual of the whole simulation (#ITE refers to the number of solver time steps).

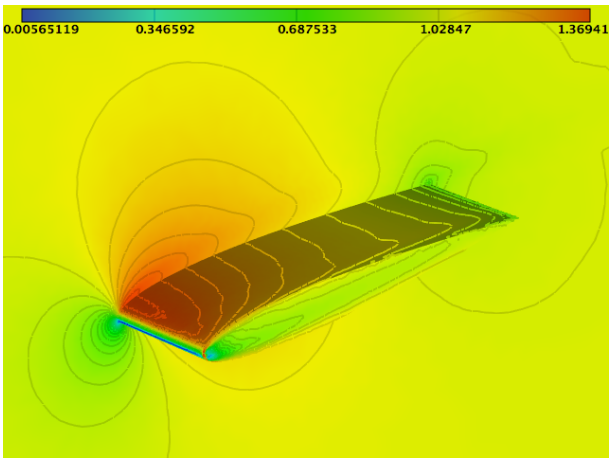


Figure 5.20: 3D subsonic NACA 0012: Solution (velocity).

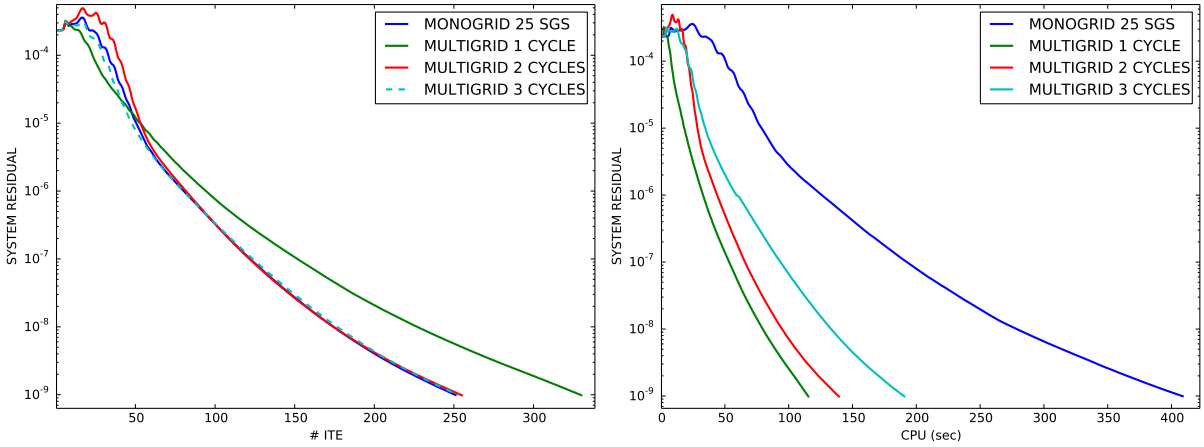


Figure 5.21: 3D subsonic NACA 0012 : Convergence of the residual of the whole simulation. Left: number of time steps. Right: wall clock time (sec).

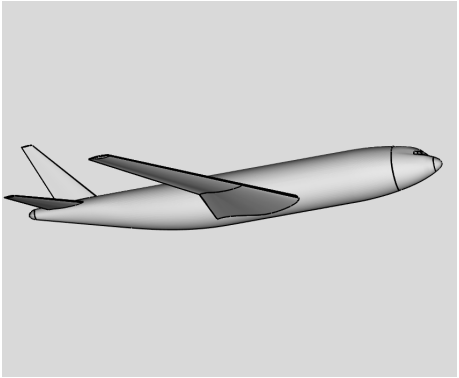


Figure 5.22: 3D wing body tails (WBT) configuration.

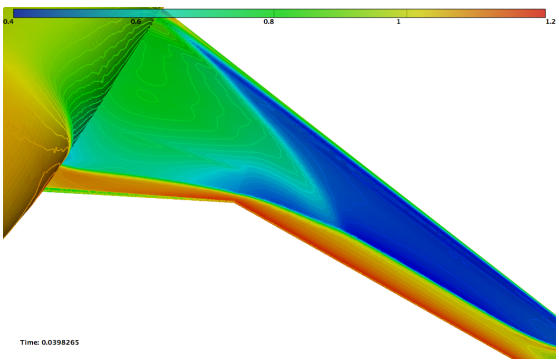


Figure 5.23: Solution computed on the finest WBT mesh using a 3-grid V-cycle.

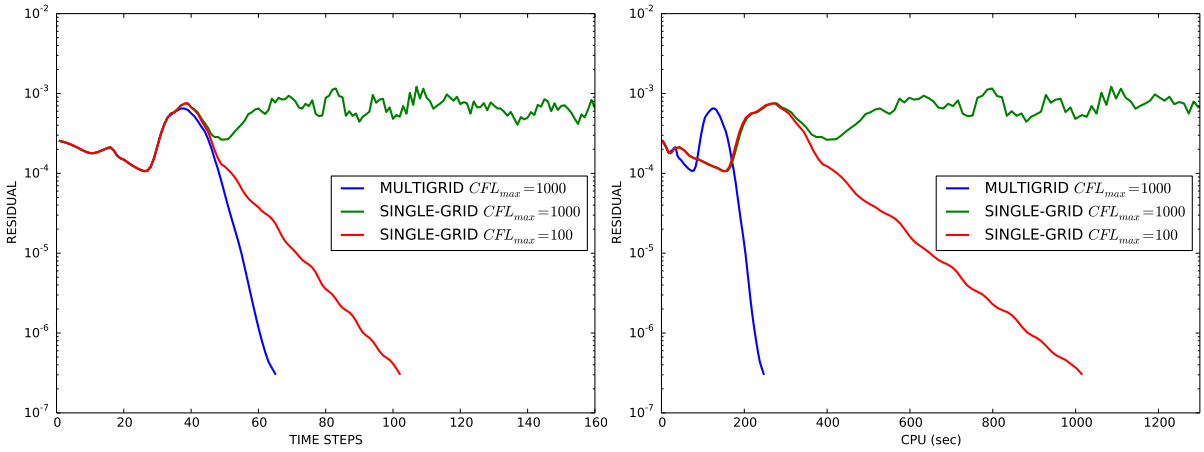


Figure 5.24: 3D WBT : Residual convergence in terms of the number of time steps (left) and wall clock time (right). Comparison between a monogrid computation and a 3-grid V-cycle.

5.4.4 Parameter dependency study

We carried out a parameter dependency study of multigrid using the 2D transonic NACA 0012 airfoil case described in the previous section, in order to point out the robustness of the multigrid method characterized by its independence to the user settings. This study consists in comparing the results of multigrid and monogrid methods for a set of simulations run with different input parameters. Here, the two chosen input parameters are the prescribed maximal CFL value (see Section 3.7) and the maximal number k_{max} of Newton iterations (at each time step). We recall that one Newton iteration consists in one SGS iteration for a single-grid computation, and in one MG cycle for a multigrid one. The output parameters we compare are the global residual convergence of the simulation and the total CPU time.

Both single-grid and multigrid simulations were run using the same parameters (except for the aforementioned input parameters of interest) of the numerical model, including a numerical dissipation of fourth order (V4 scheme, see Section 3.3.3), and the Dervieux limiter. The chosen CFL law is local at the vertices and dynamic, as described in Chapter 3. It is geometric: the local CFL value is multiplied by two at each solver iteration (if the under-relaxation allows it, see Section 3.7), which is high and might cause convergence issues in spite of the under relaxation coefficient. A maximal number of 30 SGS sub-iterations is set for the single-grid simulations, and 3 V-cycles are set for the multigrid ones. 64 simulations were run and are summarized in Table 5.1.

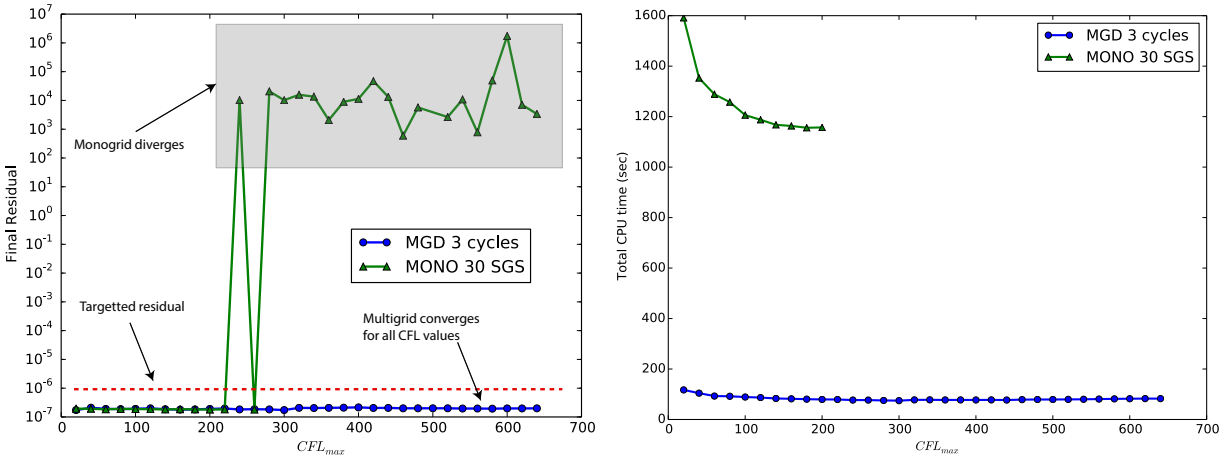
Method	# SGS/Cycles	CFL_{max}
Monogrid	30	20 to 640 (every 20)
Multigrid (4 levels)	3	

Table 5.1: Summary of the 64 simulations run for the parameter dependency study.

Figure 5.25 presents the results of the parameter dependency study. For each input CFL_{max}, we compare the final residual and the total CPU, for both single-grid and multigrid. The targeted final residual is reached by multigrid simulations for all CFL_{max} values, whereas single-grid simulations fail to converge for $CFL_{max} > 220$. The total CPU time is divided by ten using multigrid.

Remark. *The main reason why single-grid simulations fail to converge for* $CFL_{max} > 220$ *is the aggressive CFL law (we double the local CFL value at each iteration, whereas the coefficient is usually set between* 1.1 *and* 1.5*).*

This numerical example shows the robustness of multigrid compared to single-grid, with respect to the input parameters set by the user.



(a) Final residual wrt CFL_{max}. The targeted final residual was set to 10^{-6}.

(b) Final CPU comparison wrt CFL_{max}. NB: the single-grid CPU is not plotted for $CFL_{max} > 220$, as it failed to converge.

Figure 5.25: 2D transonic NACA 0012: Results of the parameter dependency study. Note that one point of the curve corresponds to one simulation (launched with the corresponding maximal CFL value).

5.5 Conclusion

We presented the implementation of an implicit multigrid procedure for inviscid flows, as well as its validation study on subsonic and transonic cases. From the experience we have gained, the V-cycle appeared to be the most efficient method. A significant improvement in both the convergence speed and robustness was observed.

In Chapter 6, we aim at extending this multigrid approach to an adaptive context. A coupling of a multigrid algorithm with the mesh adaptation procedure is presented.

Part III

Adaptive Multigrid and RANS

Chapter 6

Multigrid Strategies Coupled with Anisotropic Mesh Adaptation

Contents

6.1 Introduction

This chapter presents an attempt to couple mesh adaptation with a full multigrid (FMG) algorithm [162]. This FMG process consists in computing a solution on a sequence of hierarchical grids starting from the coarsest level. The solution is interpolated from one level to the next and multigrid simulations are performed at each level using the coarser grids (single-grid at level 1, 2-grid at level 2, 3-grid at level 3 etc.). The FMG algorithm has an interesting theoretical convergence property. It states that, if the residual of the flow computation is fully converged at level 1, decreasing the residual by one order of magnitude at stages 2, 3, etc., is enough to ensure the convergence of the global process. We want to benefit from this convergence property in an adaptive context, by coupling FMG with mesh adaptation. This coupling consists, at a given stage of the adaptation loop, at recycling the previously adapted meshes as coarse grid levels.

Thanks to this coupling, we want to improve the robustness and the rapidity of the adaptive process. As regards robustness, we want to avoid cases where the process fails due to an inadequate choice of input parameters. This choice of parameters (CFL law, maximal number of iterations of the Newton method, flux computation method, etc.) is even more critical in an adaptive context, because of the greater number of flow computations launched and because of the (anisotropic) adapted mesh. If the flow computation fails at one stage of the adaptive loop, the next stages are spoiled and the whole process fails. It is therefore complex to choose the set of parameters that will provide the best of the flow solver (in terms of accuracy of the solution and CPU time), while ensuring the convergence of the global process. The FMG theoretical property provides guarantees on the convergence, and thus reduce the parameter dependency of the adaptive procedure.

A lot of CPU time can be saved thanks to this reduced parameter dependency. In particular, we are interested in the targeted residual set for the flow computation run at each stage of the adaptation loop. Indeed, there is no guarantee that decreasing the residual by a given order of magnitude will ensure the global convergence. In this context, we want to benefit of the convergence properties arising from the multigrid theory.

The idea of coupling adaptivity with FMG is not new, see for instance [23, 148, 22, 11, 162] or more recently [131, 127]. These adaptive strategies are most frequently based on mesh refinement by local division of mesh elements. The resulting adapted meshes are nested, which, as explained in Chapter 5, has a major drawback since the quality of the meshes decreases as they are refined. In [26], a coupling of FMG with metric-based mesh adaptation is studied for the Poisson problem.

In the sequel, the FMG algorithm to be coupled with the classical mesh adaptation process is introduced along with its validation study. Then the coupling is detailed, and numerical results are presented in Section 6.4.

6.2 Full Multigrid (FMG) algorithm

The FMG algorithm [67, 162], consists in combining the classic multigrid approach with a nested iteration. A solution is computed on the coarsest mesh at stage 1, the second coarsest mesh at stage 2... and on the finest mesh at the final stage. From one stage to the next, the solution is linearly interpolated and used as a restart solution by the flow solver for the next computation. At each stage i ($i >= 2$), the coarser meshes from the previous stages are used to run a multigrid simulation. According to [67], an optimal $\mathcal{O}(N)$ complexity is obtained for N unknowns.

A typical successful FMG computation provides a solution at the end of each stage, that is as accurate as the fully iteratively converged solution on the same grid level. The standard theory [67] states that this is obtained thanks to a fixed number of iterations in each FMG stage. According to [128], if the solution is fully converged at stage 1, then it is sufficient to converge the solution by one order of magnitude at stages 2, 3, etc., in order to achieve the global convergence on the finest mesh. In other words, fully converging the solution at every stage would not improve the residual on the finest mesh (and would be more CPU comsuming). In [34], this property is validated using 2D compressible Navier-Stokes simulations. In [128] and [34], it is however found that, for some calculations, failing scenarii arise, that is, the FMG sequence does not succeed in providing an accurate solution, which is why in our coupling with adaptivity, we chose to decrease the residual by two orders rather than one order, to be sure to achieve the global convergence on adapted meshes.

6.2.1 Description

We describe the FMG algorithm using the example of four meshes (see Figure 6.1):

$$\mathcal{H}_h, \mathcal{H}_{2h}, \mathcal{H}_{4h}, \mathcal{H}_{8h}\,,$$

where $\mathcal{H}_h$ is the finest mesh and $\mathcal{H}_{8h}$ the coarsest. The FMG algorithm is the following (see Figure 6.1):

1. On $\mathcal{H}_{8h}$: starting from a uniform solution $\mathcal{S}^0_{8h}$, a solution $\mathcal{S}_{8h}$ is computed using a single-grid method. $\mathcal{S}_{8h}$ is then interpolated to $\mathcal{H}_{4h}$.

2. On $\mathcal{H}_{4h}$: the interpolated of $\mathcal{S}_{8h}$ is used as a restart solution by the flow solver : $\mathcal{S}^0_{4h} = I_{8h\to 4h}(\mathcal{S}_{8h})$. A two-grid multigrid simulation is then performed on $\mathcal{H}_{4h}$ using $\mathcal{H}_{8h}$ as the coarse mesh. $\mathcal{S}_{4h}$ is then interpolated to $\mathcal{H}_{2h}$.

3. On $\mathcal{H}_{2h}$: $\mathcal{S}^0_{2h} = I_{4h\to 2h}(\mathcal{S}_{4h})$ is used as a restart solution and a 3-grid multigrid simulation is performed on $\mathcal{H}_{2h}$ using $\mathcal{H}_{4h}$ and $\mathcal{H}_{8h}$ as coarser meshes.

4. On $\mathcal{H}_h$: $\mathcal{S}^0_h = I_{2h\to h}(\mathcal{S}_{2h})$ and a 4-grid multigrid simulation is performed.

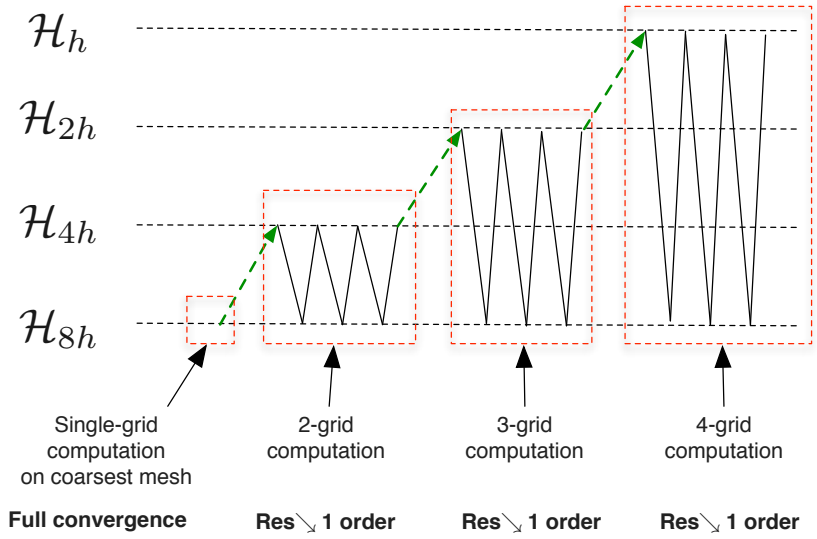


Figure 6.1: Description of the FMG algorithm.
⇢ : linear interpolation of the solution

This algorithm extends naturally to N meshes.

We validate the FMG algorithm using the 3D transonic WBT configuration and we verify the stated convergence property using the 2D transonic NACA (both cases were introduced in Chapter 5).

6.2.2 FMG validation

We validated the FMG algorithm using two examples.

- 3D transonic WBT configuration: we compare the CPU time obtained using the FMG algorithm to the time obtained with a regular 3 grid V-cycle on the finest mesh. We verify that the lambda shock on the wing is correctly captured using FMG.
- 2D transonic NACA 0012: we verify the FMG theory by comparing the results of FMG and a multi-level single-grid approach. A set of simulations was run, and for each of them a different order of magnitude is prescribed, by which the residual must be decreased in the finer levels.

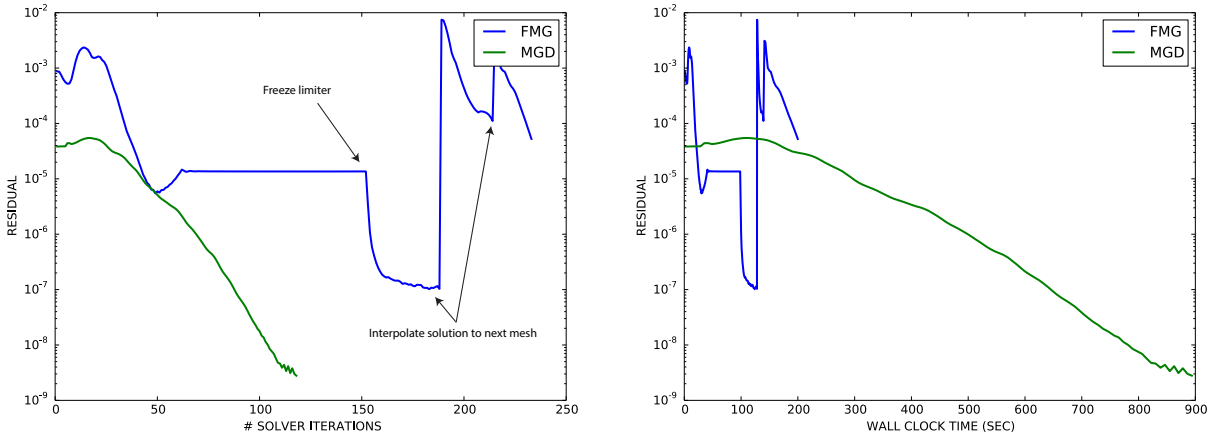
3D transonic WBT configuration

We performed two simulations using a sequence of three meshes.

- FMG simulation: the solution on the coarsest mesh is converged by 4 orders of magnitude. Then by one order of magnitude on the two other meshes. 30 SGS iterations are set for the initial single-grid computation, and 3 V-cycles for the multigrid ones.

- Multigrid simulation on the finest mesh: a 3-grid V-cycle is performed and the residual is decreased by 4 orders of magnitude.

Results. Figure 6.2a shows the convergence in terms of the number of iterations. The first part of the blue curve (from iteration 1 to 188) corresponds to the coarse mesh computation. From iterations 60 to 150, we notice a limit cycle that is due to oscillations of the limiter function. We automatically detected this limit cycle and froze the limiter around iteration 150 in order to keep converging to the targeted residual. On the two next mesh levels, the residual is decreased by one order of magnitude. It takes fewer solver iterations than the classical multigrid to converge, but most of the FMG iterations are performed on coarser levels, which saves a lot of computational time, as shown in Figure 6.2: the FMG algorithm is almost twice as fast as the classic multigrid approach. The final FMG solution is depicted in Figure 6.3, the lambda shock was accurately captured.



(a) Convergence in terms of flow solver iterations.

(b) Convergence in terms of the number of CPU time.

Figure 6.2: 3D WBT configuration: comparison of the convergence.

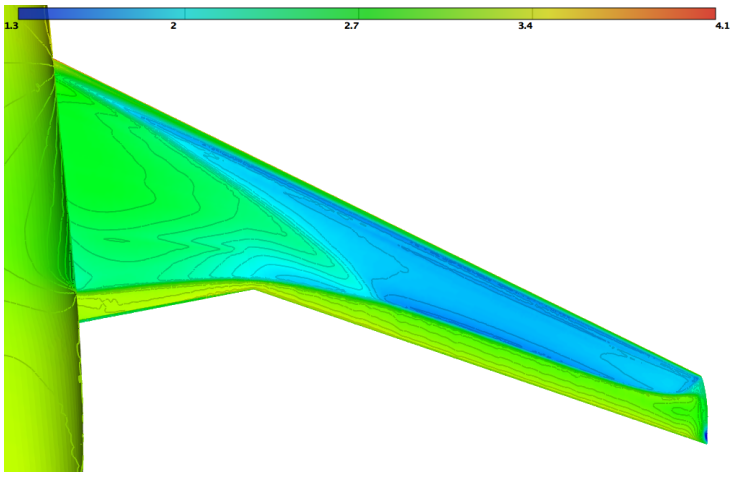


Figure 6.3: 3D WBT configuration: final solution (FMG algorithm).

Verification of the FMG theory

We want to verify or invalidate the FMG theory, according to which converging the residual by one order of magnitude at each FMG stage is sufficient to ensure the global convergence. To this end, several FMG and single-grid simulations were performed on the 2D transonic NACA case (using a set of four meshes):

- **FMG simulations:** V-cycles are used to converge the Newton method (the maximal number of cycles is set to 3). The solution is fully converged on the coarsest mesh, then on the next mesh levels the residual is decreased by an order of magnitude provided as an input.
- **Multi-level single grid simulations:** a solution is computed on each mesh level starting from the coarsest one. The solution is interpolated from one stage to the next but in contrast to the FMG algorithm, only single-grid computations are performed. Note that this is quite similar to the mesh adaptation loop, except that the meshes are not adapted. The solution is fully converged on the coarsest mesh, whereas the residual is decreased by an input order of magnitude on the next levels. A maximal number of 30 SGS iterations is set to converge the Newton method.

We compare the results obtained by the two methods, using different values for one of the input parameters: the order of magnitude by which the residual is decreased at each stage (except on the coarsest level where the solution is fully converged every time). The simulations are summarized in Table 6.1.

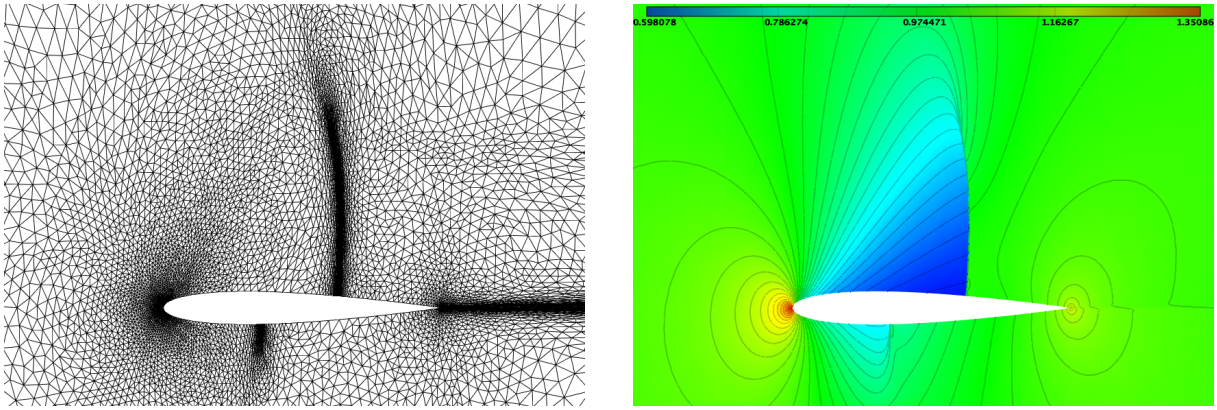
Method	# SGS/Cycles	Order of magnitude
FMG	3	0.1 to 0.0001
Single-grid	30	

Table 6.1: Summary of the simulations run.

We made comparisons in terms of the total CPU time and accuracy of the solution: a spatial L^1 error is computed using the adapted couple mesh/solution depicted in Figure 6.4, which is the final result of a 5 stage mesh adaptation. The reference mesh contains 22 000 vertices and provides an accurate prediction of the shock region.

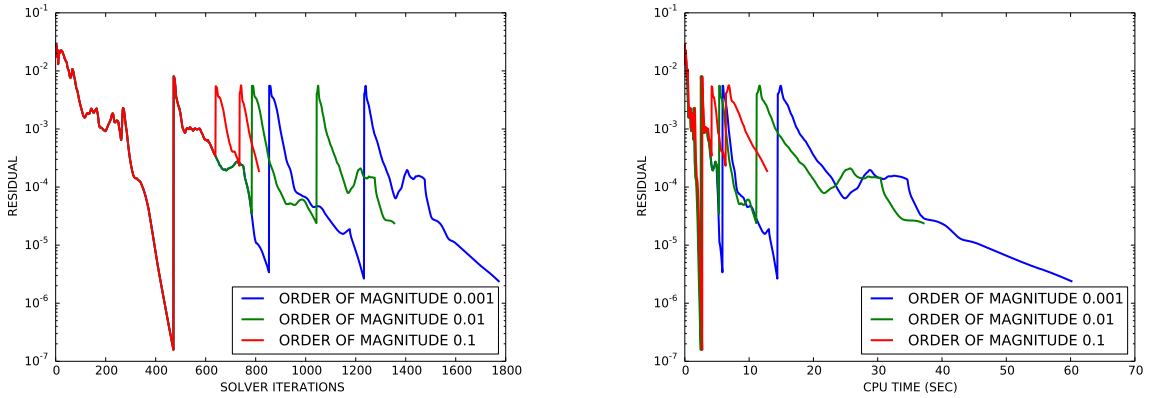
Results. Figure 6.5 presents the residual convergence of some selected input orders of magnitude: 0.1, 0.01 and 0.001. As expected, the convergence of the first mesh level (coarsest) is identical for all three simulations, because a full convergence is always prescribed for the coarsest mesh. Then, the more we decrease the residual during the next stages, the more costly it becomes in terms of both the number of iterations and CPU time.

Although iterating less on the finer levels leads to a gain in CPU, we want to make sure it does not badly impact the accuracy of the final solution. To do so, we compare final spatial errors, see Figure 6.6a. We made two main observations. First, for a given input order of magnitude, the final



(a) Adapted mesh. (b) Corresponding solution.

Figure 6.4: FMG validation: Reference mesh/solution for the 2D transonic NACA.



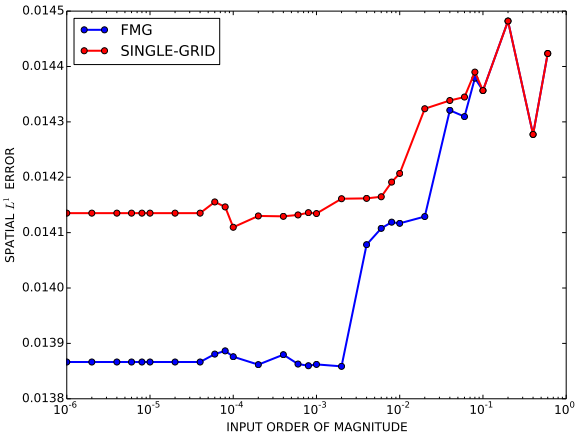
(a) Convergence in terms of the number of iterations. (b) Convergence in terms of CPU time.

Figure 6.5: Comparison of the residual convergence for some input orders of magnitude.

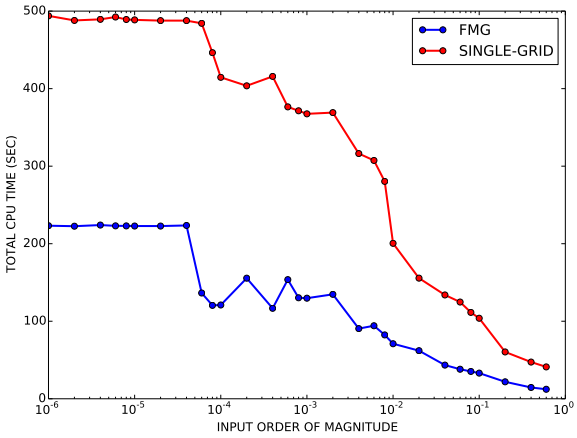
solution obtained using the FMG algorithm is always more accurate than the solution obtained using the multi-level single-grid method. Second, the maximal accuracy reached using the single-grid method is obtained by decreasing the residual by $2e^{-3}$.

Remark. *Although the FMG theory states that one order of magnitude is enough to ensure the global convergence, this example tends to show that setting two orders is safer. This is what we do in the coupling with adaptivity presented below.*

Figure 6.6b shows the corresponding CPU timings. As confirmed by Figure 6.5b, the more we decrease the residual at each stage, the more costly it is in terms of CPU time. For each input order of magnitude, using multigrid leads to a significative acceleration of the computational time.



(a) Comparison of the final spatial L^1 error.



(b) Comparison of the total CPU time.

Figure 6.6: NACA 2D : comparison of the final spatial error and the total CPU time. Each point corresponds to a simulation (run with a different input order of magnitude).

6.3 FMG algorithm coupled with adaptivity

6.3.1 Description

There are many similarities between the FMG algorithm and the mesh adaptation loop described in Chapter 1. In both cases, we start from an initial coarse mesh and the complexity of the current mesh is increased at each stage. From one stage to the next, a solution is interpolated and used as a restart solution by the flow solver.

The coupling between the two methods consists in modifying the solution computation step in the classical mesh adaptation loop. Instead of a single-grid computation, a i-grid multigrid computation is performed at stage i, using the previously adapted meshes as coarser meshes. The coupling is described in Algorithm 2 and illustrated in Figure 6.7, where $\mathcal{H}$, $\mathcal{S}$ and $\mathcal{M}$ denote respectively meshes, solutions and metrics.

Remark. *Algorithm 2 was simplified for the sake of clarity. In practice, we perform a given number of sub-iterations for each mesh complexity (usually 3 to 5 sub-iterations). Only the meshes generated at a final sub-iteration are used as coarse-grid levels.*

6.3.2 Why couple the two methods?

An example of how mesh adaptation can benefit from multigrid is presented in Figure 6.8. It considers a transonic flow (Mach 0.8, angle of attack $\alpha = 1°$) over a Falcon business jet geometry using four mesh levels $\mathcal{H}_h$, $\mathcal{H}_{2h}$, $\mathcal{H}_{4h}$ and $\mathcal{H}_{8h}$ (see Figure 6.8a). Two simulations are compared:

1. FMG algorithm (plotted in green): the solution is interpolated from one stage to the next and the

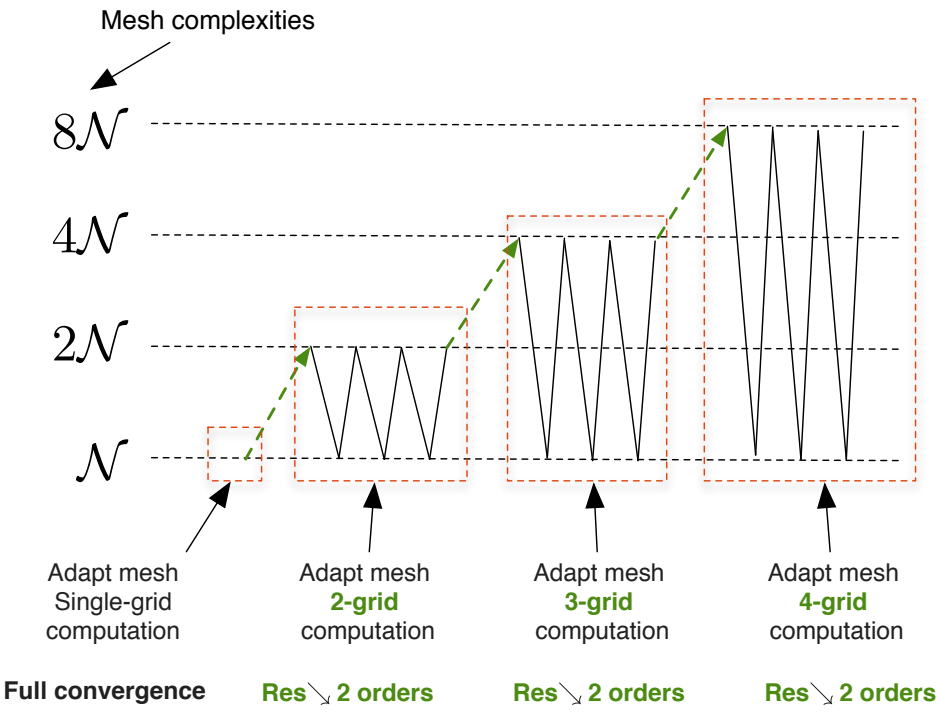


Figure 6.7: Description of the adaptive FMG algorithm. Note that for each mesh complexity, a given number of adaptive sub-iterations are performed.

Initial mesh and solution $(\mathcal{H}_0, \mathcal{S}_0^0)$ and set targeted complexity $\mathcal{N}$.
For $i = 0, n_{cpx}$

1. If $i = 0$, compute solution from pair $(\mathcal{H}_0, \mathcal{S}_0^0)$. Full residual convergence.

   If $i > 0$, compute solution from pair $(\mathcal{H}_i, \mathcal{S}_i^0)$ **using an i-grid computation and** $(\mathcal{H}_j)_{j=0,\dots,i-1}$ **as coarse meshes.** Decrease the residual by two orders of magnitude.

   If $i = n_{adap}$ break;

2. $(\mathcal{M}_{L^p,i})$ = Compute metric $\mathcal{M}_{L^p}$ according to selected error estimate from $(\mathcal{H}_i, \mathcal{S}_i)$;
3. $(\mathcal{H}_{i+1})$ = Generate a new adapted mesh from pair $(\mathcal{H}_i, \mathcal{M}_{L^p,i})$;
4. $(\mathcal{S}_{i+1}^0)$ = Interpolate new initial solution from $(\mathcal{H}_{i+1}, \mathcal{H}_i, \mathcal{S}_i)$;

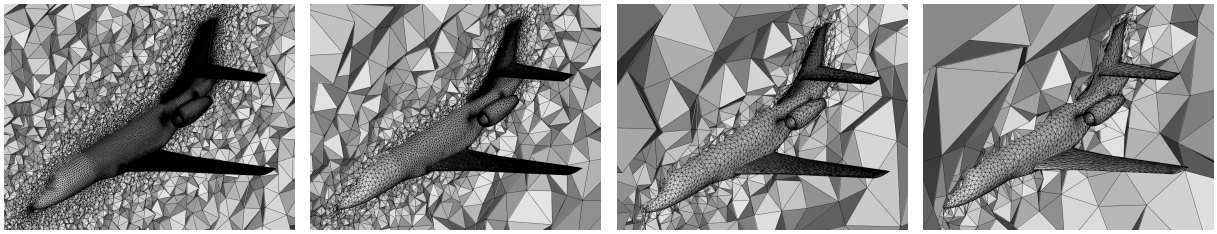
EndFor

Algorithm 2: Adaptive FMG algorithm (n_{cpx} is the number of prescribed compexities: for instance $\mathcal{N}, 2\mathcal{N}, 4\mathcal{N}$ etc.).

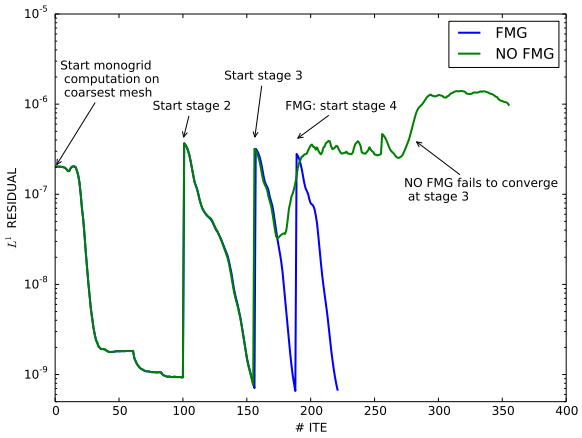
coarser levels are used to run multigrid simulations.

2. For the second simulation (plotted in blue), the solution is converged on each mesh level starting from the coarsest one. The solution is interpolated from one stage to the next but only single-grid simulations are performed (in contrast to FMG). This is similar to the mesh adaptation loop.

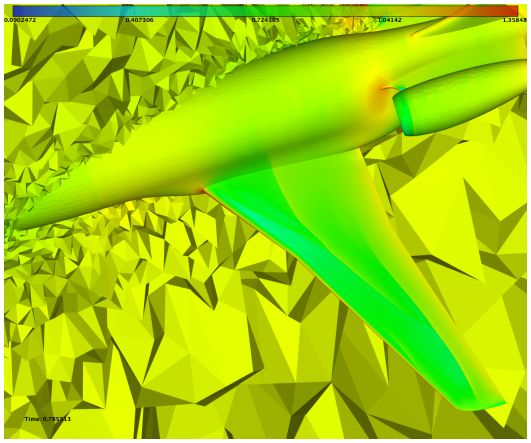
For both simulations, a sixth-order numerical dissipation was prescribed and the Dervieux limiter was used. The maximal number of Newton iterations set is 3 for the FMG algorithm and 20 for the second



(a) The sequence of four meshes used.



(b) Residual convergence in terms of time iterations. Blue: classical FMG algorithm. Green: multigrid computations were replaced by single-grid computations in the FMG algorithm.



(c) Solution (density).

Figure 6.8: FMG results for the case of the transonic Falcon.

computation.

The final solution using FMG is shown in Figure 6.8b and the residual convergence for both simulations in Figure 6.8c. It shows that simulation 2 fails to converge at stage 3, in contrast to the FMG algorithm which provides a converged solution at every stage. This is an eloquent example, because the second simulation is similar to the mesh adaptation loop, the only difference being that here the meshes are not adapted from an error estimation of the solution. It illustrates how multigrid can improve the robustness of the adaptive process.

6.4 Numerical results

Two cases were considered to validate the adaptive FMG algorithm: the 3D subsonic NACA 0012 airfoil and the 3D transonic WBT configurations. For each case, two simulations were performed:

- A classical mesh adaptation process: the prescribed mesh complexity was increased at each stage, and monogrid simulations were run. The residual of the solution was fully converged at each stage.
- An adaptive FMG algorithm: the same mesh complexities were prescribed. The residual of the solution was fully converged for the lowest mesh complexity (i.e. stage 1), and then reduced by two

orders of magnitude for the other complexities.

In both cases, the adaptive multigrid algorithm showed a significant reduction in the total wall clock time of the simulation. It was ensured that both methods converged to same final solution. To do so, one more stage of the classical mesh adaptation loop was performed using a higher mesh complexity, and the resulting couple mesh/solution was then used as a reference solution to compute spatial errors. Both algorithms showed the same mesh convergence.

6.4.1 3D subsonic NACA 0012

The first mesh adaptation considers the 3D subsonic NACA 0012 case that was introduced in Section 5.4.3. Six stages of the classical mesh adaptation loop and the adaptive FMG alorithm were performed, using mesh complexities leading approximately to the following numbers of vertices:

$$8\,000,\ 16\,000,\ 32\,000,\ 64\,000,\ 128\,000,\ 256\,000\,.$$

For each mesh complexity, three sub-iterations in the adaptation loop were performed. The residual of the solution was fully converged to 10^{-9} at each stage of the classical adaptation. A slope limiter was used in order to avoid spurious oscillations [4]. A freeze of this limiter is activated in case the limiter itself is oscillating. During the adaptive FMG algorithm, the residual was fully converged at stage 1 (which does not differ from the classical algorithm), and then it was reduced by two orders of magnitude at stages 2, 3, etc. The residual convergence of both simulations in terms of wall clock time is presented in Figure 6.9. The total wall clock time of the simulation is dramatically improved: 13min9s for the adaptive FMG method, and 2h25min for the classical adaptation algorithm. As shown in Figure 6.10, the mesh convergence observed for both methods are similar. The reference couple mesh/solution was computed using one more step in the classical adaptation loop at a mesh complexity leading to $\sim 512\,000$ vertices. The final adapted mesh and the solution are depicted in Figures 6.11 and 6.12.

6.4.2 3D transonic WBT Configuration

The second example considers the 3D transonic WBT. Four stages of the classical mesh adaptation loop and of the adaptive FMG algorithm were performed. The prescribed mesh complexities (corresponding to each stage) lead approximately to the following numbers of vertices:

$$140\,000,\quad 210\,000,\quad 340\,000,\quad 620\,000\,.$$

At each stage (i.e. mesh complexity), five sub-iterations in the adaptation loop were performed. A comparison of the residual convergence of both methods in terms of wall clock time is presented in Figure 6.13. The total wall clock time of the simulation is reduced from 1d3h57m for the classical mesh

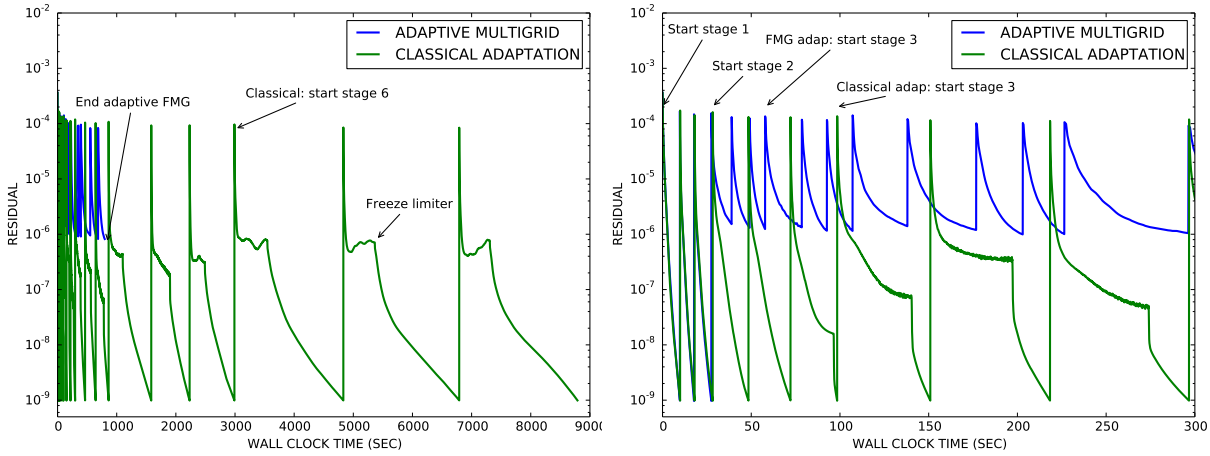


Figure 6.9: 3D subsonic NACA 0012: comparison of the residual convergence in terms of wall clock time. Left: whole simulation. Right: close-up view of the first stages. Note that each stage corresponds to a mesh complexity, and that three sub-iterations were performed for each of them.

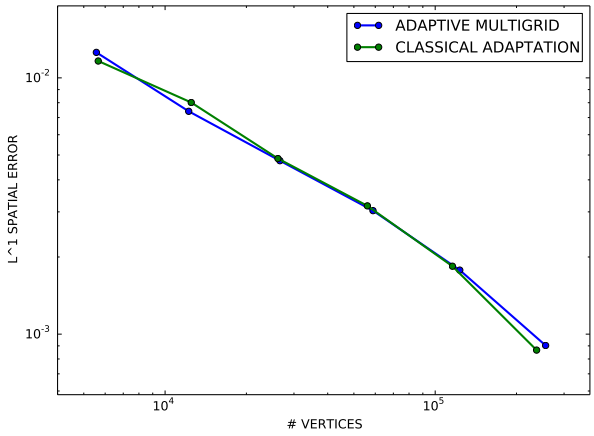


Figure 6.10: 3D subsonic NACA 0012: mesh convergence to the reference solution.

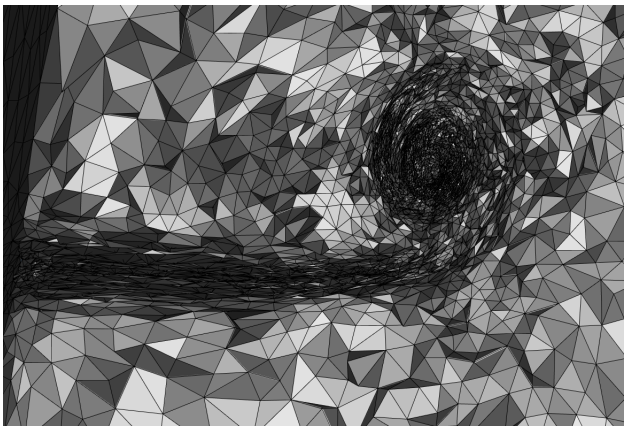


Figure 6.11: 3D subsonic NACA 0012: Cut in the volume of the final adapted mesh.

adaptation loop, to 2h53m for the adaptive FMG algorithm. As shown in Figure 6.14, converging the residual by two orders of magnitude at stages 2, 3 and 4 of the adaptive FMG process does not affect the global convergence to the reference solution. This reference couple mesh/solution was computed by performing one more step of the classical mesh adaptation (∼5M vertices). Various views of the final solution and of the corresponding adapted meshes are depicted in Figures 6.15, 6.16 and 6.17.

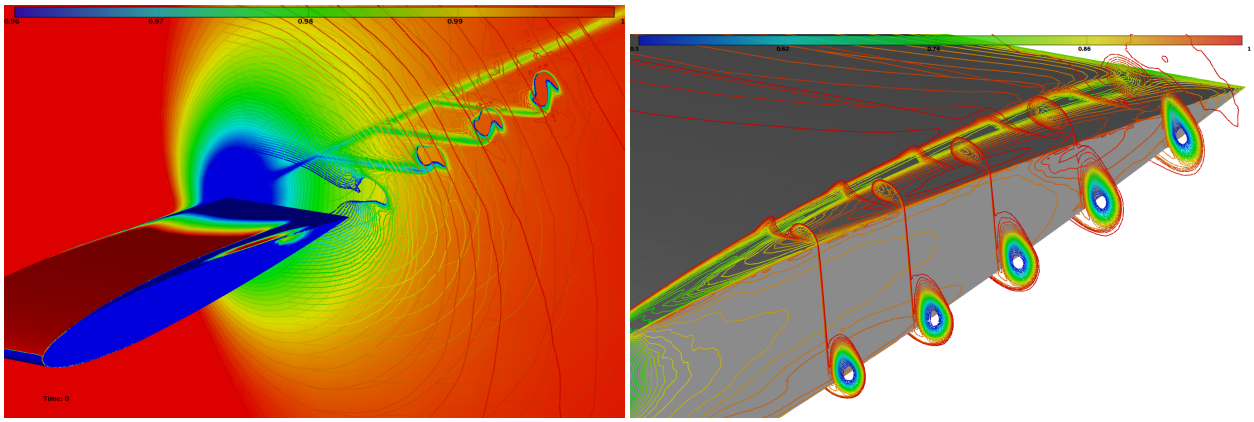


Figure 6.12: 3D subsonic NACA 0012 : velocity isovalues.

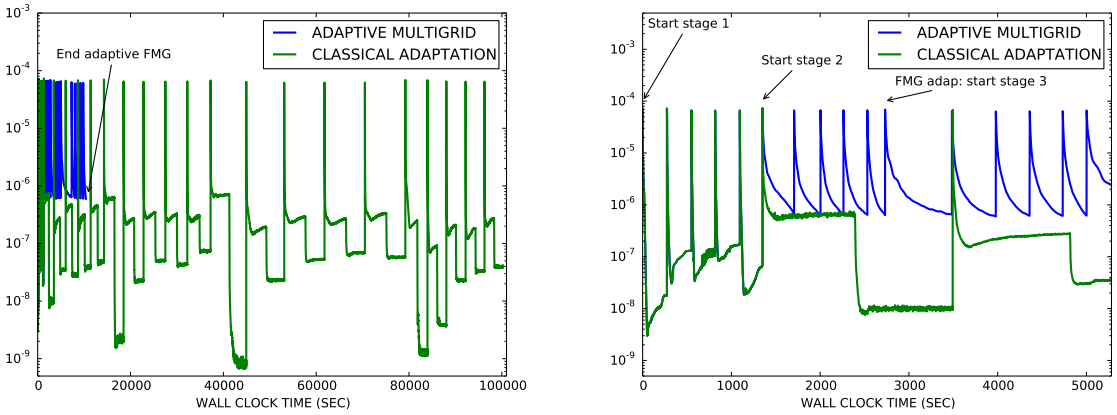


Figure 6.13: 3D transonic WBT: comparison of the residual convergence in terms of wall clock time. Left: whole simulation. Right: close-up view of the first stages. Note that each stage corresponds to a mesh complexity, and that five sub-iterations were performed for each one of them.

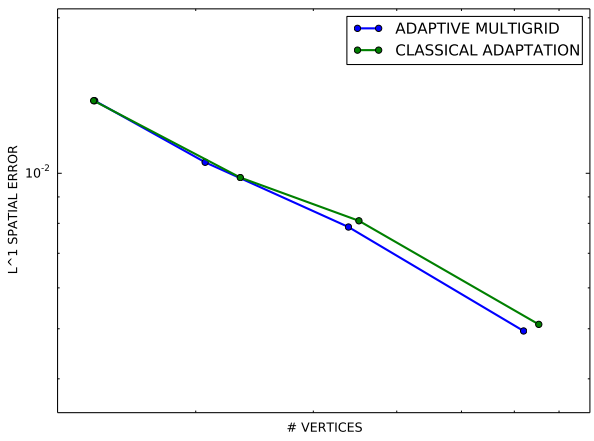


Figure 6.14: 3D transonic WBT: Mesh convergence to the reference solution.

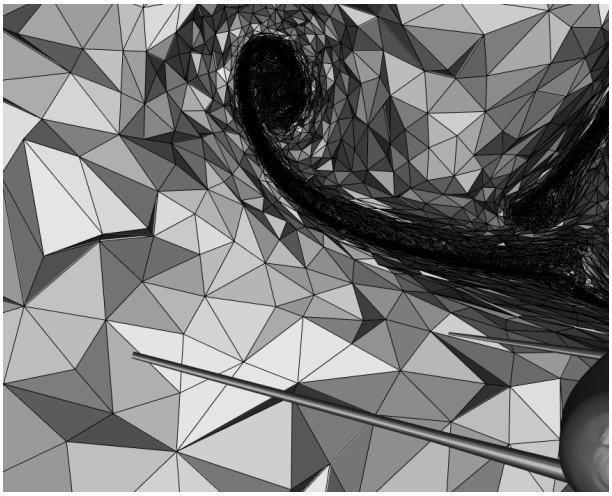


Figure 6.15: Transonic WBT configuration: cut in the trailing vortices region of the final adapted mesh.

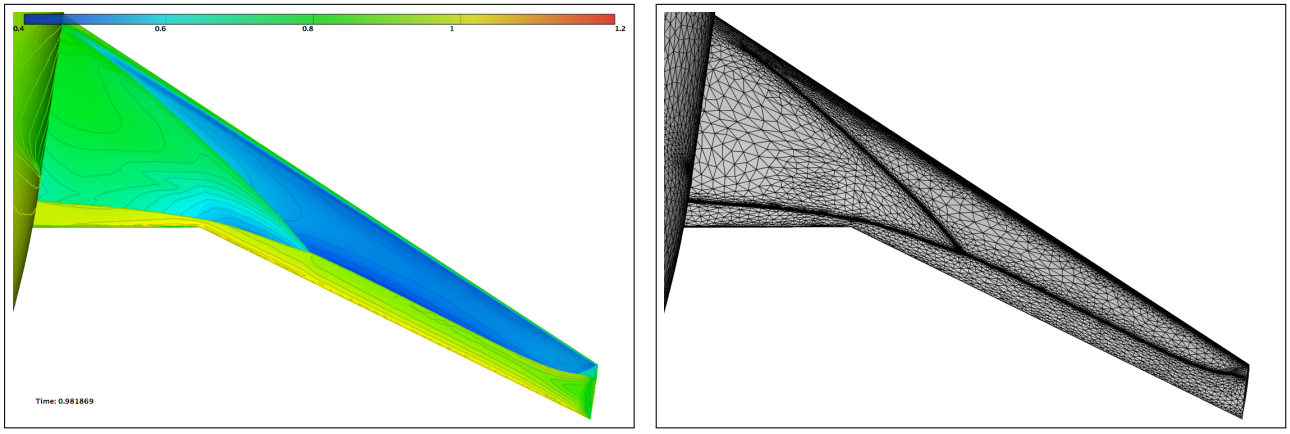


Figure 6.16: Transonic WBT configuration: pressure on the wing and corresponding adapted surface mesh.

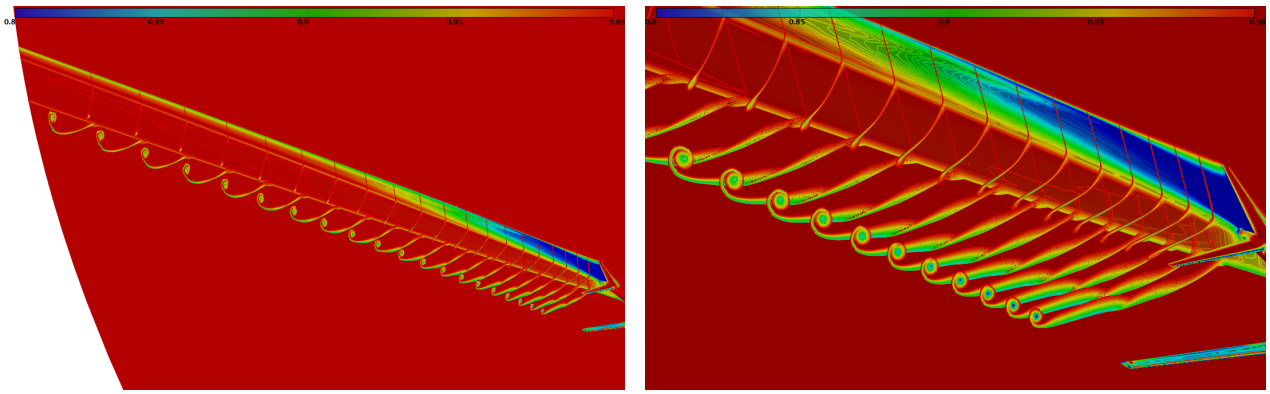


Figure 6.17: Transonic WBT configuration: views of the wing-tip vortices in the wake.

6.5 Conclusion

In this chapter, we presented a coupling of the FMG algorithm with mesh adaptation, which consists in recycling the meshes generated during the adaptive loop to run multigrid flow computations instead of single-grid ones.

First, we described the FMG algorithm along with its interesting theoretical convergence property, which states that if the solution is fully converged at stage 1, then it is sufficient to converge the solution by one order of magnitude at stages 2, 3, etc., in order to achieve global convergence on the finest mesh. We carried out a validation of the FMG algorithm. A significant gain in CPU time was observed for the 3D transonic WBT compared to the classic multigrid approach. The case of the 2D transonic NACA was instructive. In particular, we saw that even though the FMG theory states that one order of magnitude is enough, this example tends to show that two orders are needed, which is what we applied for the coupling with adaptivity.

In the classic (single-grid) mesh adaptation process, there exists no such guarantee on the convergence. In consequence, the order of magnitude by which the residual of the flow computation is decreased at each adaptive iteration is generally established empirically, which sometimes supposes repeating the simulation one or several times using a new input value for the residual convergence (because it failed using the previous values). In this context, one of the motivations for this coupling with FMG is to avoid situations where the user chooses a more secured non optimal input value, causing a significant waste of computational effort.

We validated the coupling using two 3D cases (subsonic NACA and transonic WBT). To this end, we compared the adaptive FMG algorithm to a classic mesh adaptation. The residual was fully converged at each stage of the classic mesh adaptation and decreased by two orders of magnitude at each stage of the coupled process. As a result, significant CPU gains were observed. More importantly, we verified that the mesh convergence is the same for both approaches, i.e. that the final solution obtained using the coupling is as accurate. Note that, although a full convergence at each stage of the classic adaptive process is probably not optimal, searching for the optimal parameter is empirical. This coupling, based on the aforementioned theoretical property, provides a generic guarantee on the global convergence.

Chapter 7

Contributions and Remaining Challenges for 3D RANS Adaptive Simulations

Contents

7.1 Introduction

Anisotropic metric-based mesh adaptation has proved to be a powerful approach for the simulation of three dimensional inviscid flows: sonic boom prediction, blast propagation, acoustic waves, etc. It has been established that it has the ability to (i) substantially optimize the tradeoff between accuracy of the solution and the number of degrees of freedom (thus the computational time), (ii) accurately capture all scales of the physical flow by automatically detecting the regions of interest where the mesh needs more resolution, (iii) reduce the numerical scheme dissipation by automatically taking into account the anisotropy of the physics, and (iv) obtain an early mesh convergence: high order asymptotic rate of convergence even for discontinuous flows. Proving that the same benefits hold for RANS simulations remains an open question as many new research issues appear. We give an overview of them and show how we can address some of these issues in 2D and 3D.

When dealing with viscous flows, it is important to distinguish 2D simulations from 3D ones. In 2D, a lot of studies exist on how to perform fully unstructured adaptive simulations. In most cases, the error estimate is the core of the study [53, 70, 102, 137, 170]. Goal-oriented estimates are usually derived for an accurate prediction of the skin friction coefficient or velocity profiles. The size of the 2D meshes (in terms of the number of elements) allows classical adaptive meshing strategies to handle the required level of anisotropy. In addition, the geometries considered remain simple (flat plate, multi-element airfoil, etc.), which makes generating a quasi-structured or unstructured boundary layer mesh simpler than in the 3D case. In 3D, boundary layer mesh generation is a field of research by itself [19, 91, 143]. To deal with this difficulty, most 3D studies keep a frozen boundary layer mesh during the adaptive process [70, 137] and focus on the error estimate. In [138], an extension to goal-oriented estimates is provided for turbulent flows. In [70], an extension to *a posteriori* H^1 estimates is applied to the set of RANS equations. The advantage of these approaches is that they keep a fully converged flow in the viscous layers without having to handle the very high level of anisotropy $O(1:10^6)$ that appears near the body. These approaches also avoid the major difficulty of generating anisotropic surface meshes, especially for complex geometries. In addition, a frozen boundary layer prevents issues with the convergence of the flow solver near a viscous body by keeping well-shaped elements. A few attempts have been tried to generate fully unstructured adaptive meshes for viscous simulations. In [106], a boundary layer/shock interaction is studied with the simple Baldwin-Lomax turbulent model. The viscous body remains a flat plate to avoid any surface remeshing issues. Although this approach provides some insights on the flow (location of the recirculation bubble for instance), it fails to provide quantitive information such as the lift or drag value. In [126], a remeshing strategy to generate anisotropic meshes with only edge-based primitives is discussed. The authors show that the way the surface metric is handled during the remeshing near a viscous body can lead to erroneous results. Special care is thus needed to make sure that every operation on a metric (such as interpolation, intersection or length computation) is correct. Note that some Navier-Stokes studies exist for laminar flows at lower Reynolds numbers [138]. If this flow regime is of great help to validate error

estimates, numerical schemes or remeshing strategies, it is far from the complexity of targeted industrial applications such as the high-fidelity prediction of drag or lift at high Reynolds numbers. The interest of the community for these flow configurations -proved for instance by the many AIAA workshops [147, 163]- is now motivated by the increasing maturity of flow solvers and the increasing complexity and fidelity of the geometry.

7.1.1 Contributions

In this chapter, we do not claim to provide a final answer to the adaptation of turbulent RANS simulations. We do, however, provide some discussions and contributions that should help to design robust and efficient adaptive strategies. To reach this goal, complex (and maybe years of) developments for each component (flow solver, error estimate, remeshing) are still needed. However, the component that needs the most developments remains the meshing step, on which we focus in the sequel.

We first provide an additional proof of concept that fully unstructured adaptive meshes can be used to accurately predict viscous phenomena on 2D examples. To do so, we first extend the (one-field) multi-scale metric to the many-field multi-scale metric to take into account several solution fields for adaptation. 2D validation examples (chosen among those introduced in Chapter 4) are revisited using anisotropic adaptation. The goal is to provide a simple but robust error estimate, and also to verify that the numerical schemes and implementation choices described in the previous chapters can support fully-anisotropic meshes in the boundary layer. Then, we introduce metric-aligned and metric-orthogonal meshing strategies that make it possible to generate automatically the highly anisotropic quasi-structured mesh elements required in the boundary layer. For the 3D case, we discuss the remaining challenges for the mesh generation step and present some preliminary results for the metric-aligned approach both on surface and volume mesh generation.

7.2 Multi-field multi-scale error estimates for RANS

Error estimates are generally sought within the goal-oriented [102, 164, 170], norm-oriented [103] or entropy-variable [53] frameworks. In these cases, numerical schemes along with continuous and discretized PDEs (equations of state), are taken into account in the analysis, leading to some (guaranteed) error bounds. However, they usually require having an adjoint solver, and anisotropic estimates are harder to derive. Here, we prefer to focus on simple geometric error estimates that depend only on the numerical solution provided by the flow solver. This kind of estimate is complementary to the aforementioned estimates for their ease of use. In addition, for second order schemes, the anisotropy is naturally contained in the second order derivatives of the sensor, see Chapter 1. However, unlike in inviscid cases where one sensor usually contains sufficient information to drive the adaptation (Mach number for supersonic studies, density field for blast, pressure for acoustic, etc.), it is more difficult to derive a single sensor for

RANS simulation. Even for simple cases where the velocity field drives the simulation, we observe that under resolving other fields (density, eddy viscosity) leads to wrong predictions of velocity profiles or skin friction coefficient. Consequently, each field should be sufficiently resolved to provide reliable results. We discuss how to derive a unique metric with multi-field as input. Additional choices are given for the turbulent model. We also derive a gradient-based metric for validation purposes.

Multi-field multi-scale error estimates. When a single field is used as the sensor, the procedure to adapt the mesh consists in generating a sequence of meshes at fixed complexity $\mathcal{N}$ for $\mathcal{M}_{L^p}$. Then, $\mathcal{N}$ is increased until a user-acceptable level of accuracy is reached. It has been proved that the interpolation error with respect to the sensor field is optimal for the considered norm. However, when multiple fields are considered, the normalization with the same complexity leads to a non-balanced level of error. In order to overcome this drawback, the normalization of the metric of each field is done for the same level of error. Starting from the estimate of the optimal interpolation error of the sensor:

$$\varepsilon \;=\; \|u - \Pi_h u\|_{L^p(\Omega_h)} = 3\mathcal{N}^{-\frac{2}{3}} \left(\int_\Omega \det\left(|H_u(\mathbf{x})|\right)^{\frac{p}{2p+3}} \mathrm{d}\mathbf{x} \right)^{\frac{2p+3}{3p}},$$

where u is the sensor, we can scale the complexity to fit the given level of error. Starting from the local optimal L^p normalization:

$$\mathcal{M}_p(u) = \det(|H_u|)^{-\frac{1}{2p+3}} |H_u|, \qquad \text{with} \qquad \mathcal{C}_p = \int_\Omega \sqrt{\det(\mathcal{M}_p(u))}\mathrm{d}\mathbf{x}\,,$$

the final optimal L^p metric providing error ε is :

$$\mathcal{M}^{\varepsilon}_{L^p}(u) = \frac{3}{\varepsilon}\,\mathcal{C}_p^{\frac{3}{p}}\,\mathcal{M}_p(u).$$

Finally, the multiple metric fields are intersected and the normalization of the complexity is performed only on this single final metric. In the following examples, we consider the density, the norm of velocity, the pressure and the eddy-viscosity of the turbulence model as the combination of sensors.

For the turbulent model, we add one more sensor to the previous one. We perform the adaptation using the linear interpolate of the sum of the production, destruction and diffusion terms of the Spalart-Allmaras model (see Chapter 3). This is motivated by Laplacian *a priori* error estimates [26] where the adaptation on the second member (or observation) is needed to accurately capture the solution (in our case, the dissipation term).

Gradient-based metric. The turbulent boundary layer is usually divided into 3 parts: sub-layer, viscous layer and outer-layer. In the sub-layer, the velocity profile is empirically linear. Consequently, the size provided by a control of the interpolation error on the velocity interpolation should provide a maximal size while practical considerations require a smooth gradation in the normal direction. In order

to enforce this feature automatically, we consider a tailored metric that controls the size according to the gradients of the velocity. This metric is inherited from H^1 error estimate [55]. If $\Phi = \|\mathbf{u}\|_{\mathbf{2}}$, then the metric is assembled by integrating the gradient computed at the neighboring elements of a point. So the metric is defined point-wise, its upper part is:

$$\mathcal{M}_G(\Phi) = \frac{1}{|S|} \begin{pmatrix} \int_S \left(\frac{\partial \Phi}{\partial x}\right)^2 & \int_S \frac{\partial \Phi}{\partial x}\frac{\partial \Phi}{\partial y} & \int_S \frac{\partial \Phi}{\partial x}\frac{\partial \Phi}{\partial z} \\ & \int_S \left(\frac{\partial \Phi}{\partial y}\right)^2 & \int_S \frac{\partial \Phi}{\partial y}\frac{\partial \Phi}{\partial z} \\ & & \int_S \left(\frac{\partial \Phi}{\partial z}\right)^2 \end{pmatrix},$$

where the integrals are approximated by using the natural basis $(\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$ of $\mathbb{R}^3$:

$$\int_S \left(\frac{\partial \Phi}{\partial x}\right)^2 = \sum_K |K| \left(\nabla_K \Phi \,.\, \mathbf{e}_x\right)^2,$$

and where the sum is over the elements surrounding the point under consideration. $|S|$ denotes the sum of the volume of these elements.

If this metric is not optimal to control the interpolation error, we use it to validate the previous metric for the flat plate example. Indeed, this metric ensures a fine control of the gradients in the boundary layer.

7.3 Metric-aligned and metric-orthogonal mesh generation

The difficulty to generate completely adaptive boundary layer mesh is due to the incompatibility of standard techniques to comply with a metric size prescriptions. Indeed, standard boundary-layer mesh generation techniques generate elements with an advancing layer/normal type process [77, 91, 113, 107, 143] so the sizing in the volume mesh depends only on the surface mesh. In contrast, standard anisotropic mesh generation strategies fail to generate quasi-structured elements as are typically required in the boundary layer. Consequently, the quest of anisotropic mesh generation with locally structured elements is of main interest to improve the mesh quality in such regions. But, it remains an open problem with only a small number of previous attempts in 2D and 3D [82, 152].

In this section, we show that the quality of anisotropic unstructured meshes can be improved using a metric-aligned or a metric-orthogonal strategy [97, 115]. The metric-aligned method consists in generating unit regular (equilateral) elements in metric space that are aligned with the metric field eigenvectors, see Figure 7.1 (left). This method improves the angles distribution in physical space of the resulting adapted mesh which is of main interest in CFD computation to capture shock waves, contact discontinuities, ... The metric-orthogonal method consists in generating unit orthogonal (right-angled) elements in metric space that are aligned with the metric field eigenvectors, see Figure 7.1 (right). This method improves the

angles distribution in physical space of the resulting adapted mesh and also generates quasi-structured anisotropic adapted meshes. This seems to be a promising approach to generate adapted quasi-structured mesh in boundary layer and wake regions.

In the following, we describe the strategy to generate metric-aligned and metric-orthogonal meshes. Then, we illustrate the gain in quality on the remeshing of the wing-body configuration of the second drag prediction workshop.

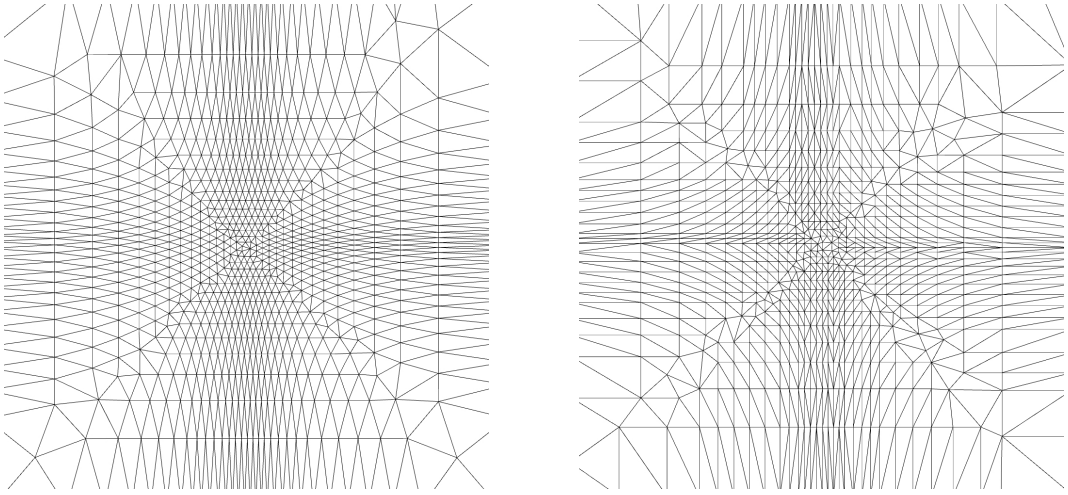


Figure 7.1: Illustration of alignment strategies on a simple analytic function. Left, metric-aligned anisotropic mesh adaptation. Right, metric-orthogonal anisotropic mesh adaptation.

7.3.1 Overall strategy

As we want to force the alignment of the edges, standard local remeshing approaches based on a set of classical operators (insertion, collapse, swap, etc.) as in [104, 126], seem to be unsuitable to be used for this purpose as they iteratively modify the mesh with no specific ordering. Frontal methods [95] have, however, been used to generate high-quality isotropic meshes but with little success for anisotropic mesh generation because the front is marching into a space that has not yet been as meshed. This work combines 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 $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$. Unlike fully frontal mesh generation techniques, where a front of points/elements is used to fill the ungridded computational domain, the points are inserted in an *empty volume mesh*, as in [113]. Here, 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 done 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 algorithms [10]. However, instead of starting the process from the empty mesh generated by

the mesh generation process, we use the fast coarsening cavity-based operator defined in [97].

Starting from a provided input 3D valid volume mesh supplied with a metric discrete field, the overall procedure to generate a metric-aligned/orthogonal anisotropic adapted mesh is composed of the following steps:

- Store the initial mesh-metric couple as background information
- Adapt the surface mesh in the standard way [104]
- Generate an *empty volume mesh* with the fast collapse operator based on the cavity operator, see [97] and Chapter 1
- Propose, filter and insert points with a frontal algorithm and the cavity-based insertion operator
- Optimize the final mesh quality using local reconnection.

Now, we describe more precisely the frontal algorithm and how new points are proposed to force the quasi-structured aspect of the mesh.

7.3.2 An advancing front point creation strategy

A frontal approach is used to enforce alignment and orthogonality with respect to the metric field. In [115], the front is defined by a list of faces, but in this work we consider a front composed of vertices. From a practical point of view, the new points are proposed by vertices and not by faces. Given a point $\mathbf{x}_0$ and its metric $\mathcal{M}_0$ of the current front with eigenvectors $(\mathbf{u}_{0,i})_{i=1,3}$ and eigenvalues $(\lambda_{0,i})_{i=1,3}$, six points are proposed:

$$\mathbf{x}_{i\pm} = \mathbf{x_0} \pm \lambda_{0,i}^{-\frac{1}{2}} \mathbf{u}_{0,i}. \tag{7.1}$$

When the metric is isotropic, we force the eigenvectors to be aligned with the natural axis of $\mathbb{R}^3$. Once proposed, we have to check that these new points are inside the current volume domain by using a simple mesh localization algorithm. This check is also performed on the background mesh. The back mesh localization also provides the metric $\mathcal{M}_i$ of $\mathbf{x}_i$. But, this does not take into account the metric variation in terms of sizes and directions.

Improved direction and length. In order to improve the direction of the proposed points, a four-step Runge-Kutta-like algorithm is considered to give an initial guess of new point $\mathbf{x}_i$. Let us give the

algorithm for one of the six points proposed by $\mathbf{x}_0$:

$$\begin{aligned}
\mathbf{x}_1 &= \mathbf{x_0} + \frac{1}{4}\,\lambda_{0,i}^{-\frac{1}{2}}\,\mathbf{u}_{0,i} \\
\mathbf{x}_2 &= \mathbf{x_1} + \frac{1}{4}\,\lambda_{1,j_1}^{-\frac{1}{2}}\,\mathbf{u}_{1,j_1} \quad \text{where } j_1 = \text{argmax}_{k=1,3}\mathbf{u}_{1,k}.\mathbf{u}_{0,i} \\
\mathbf{x}_3 &= \mathbf{x_2} + \frac{1}{4}\,\lambda_{2,j_2}^{-\frac{1}{2}}\,\mathbf{u}_{2,j_2} \quad \text{where } j_2 = \text{argmax}_{k=1,3}\mathbf{u}_{2,k}.\mathbf{u}_{1,j_1} \\
\mathbf{x}_4 &= \mathbf{x_3} + \frac{1}{4}\,\lambda_{3,j_3}^{-\frac{1}{2}}\,\mathbf{u}_{3,j_3} \quad \text{where } j_3 = \text{argmax}_{k=1,3}\mathbf{u}_{3,k}.\mathbf{u}_{2,j_2}
\end{aligned}$$

The vector $\mathbf{u} = \mathbf{x}_0\mathbf{x}_4$ provide the optimal direction where new point $\mathbf{x}_i$ is proposed. Now, we need to find the final position of $\mathbf{x}_i$ on the line $(\mathbf{x}_0\mathbf{x}_4)$ such that

$$\ell_{\mathcal{M}}(\mathbf{x}_0\mathbf{x}_i) = \int_0^1 \sqrt{{}^t\mathbf{x}_0\mathbf{x}_i\;\mathcal{M}(\mathbf{x}_0 + t\,\mathbf{x}_0\mathbf{x}_i)\;\mathbf{x}_0\mathbf{x}_i}\;\mathrm{d}t = 1\,.$$

The procedure is based on a dichotomy along the line $(\mathbf{x}_)\mathbf{x}_4)$. Note that we need to iterate because we interpolate the metric from the background mesh.

Front definition and update. The initial front of points is given by the list of the surface points. Each point of the front proposes new points to be inserted following the above process. This list of new points is then filtered in order to suppress insertion points that are too close in metric space, see the next section. The filtering process gives the final 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.

7.3.3 Anisotropic filtering

By using the previous point creation procedure, neighboring points in the front can generate similar points, it is thus important to filter out the points that are too close in metric space. To do so, we use an octree of points. Each octant can contain up to 10 points before being subdivided. Initially, the octree contains only the surface points (that are constrained and define the initial front). The rejection test is based on the Riemannian length computation, see Chapter 1.

To validate the insertion of a point, we first check the length between all the points that are in the octant containing the point to be inserted. If no rejection occurs, then the current octree 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 along with its metric.

7.3.4 Numerical illustration

The previous procedure is introduced in 3D for the sake of simplicity, but the same approach was developed in 2D and for surface mesh adaptation. For the surface case, the main modification consists in working

in a local Frenet frame of the surface. The initialization of the front is then based on the mesh of the edges instead of the surface mesh.

In this example, we consider the DPW2 configuration (see Chapter 4). The multi-field multi-scale metric is computed on the converged flow solution on the coarse mesh. We then generate the adapted surface mesh resulting from this metric with the standard approach and with the metric-aligned one, see Figure 7.2. Note that the multi-field multi-scale metric imposes an anisotropic ratio of around 10^5 on the surface mesh. The mesh adapted using the standard approach is composed of 57 921 vertices and 115 838 triangles, while the metric-aligned mesh is composed of 65 608 vertices and 131 212 triangles. The discrepancy in the number of nodes is explained by the fact that the metric-aligned follows the direction of minimal sizes. Consequently, more points are inserted in these directions. This metric tends to generate two boundary layer surface meshes at the wing-fuselage junction, see Figure 7.2 (right). The qualities of the meshes in the metric are equivalent, only the angle of the triangles are improved for the metric-aligned version. For each method, more than 94% of the edges have a unit length (in $[\frac{1}{\sqrt{2}}, \sqrt{2}]$), and the percentage of edges having a length in $[0.90, 1.11]$ is 53% for the metric-aligned and 38.91% for the standard approach.

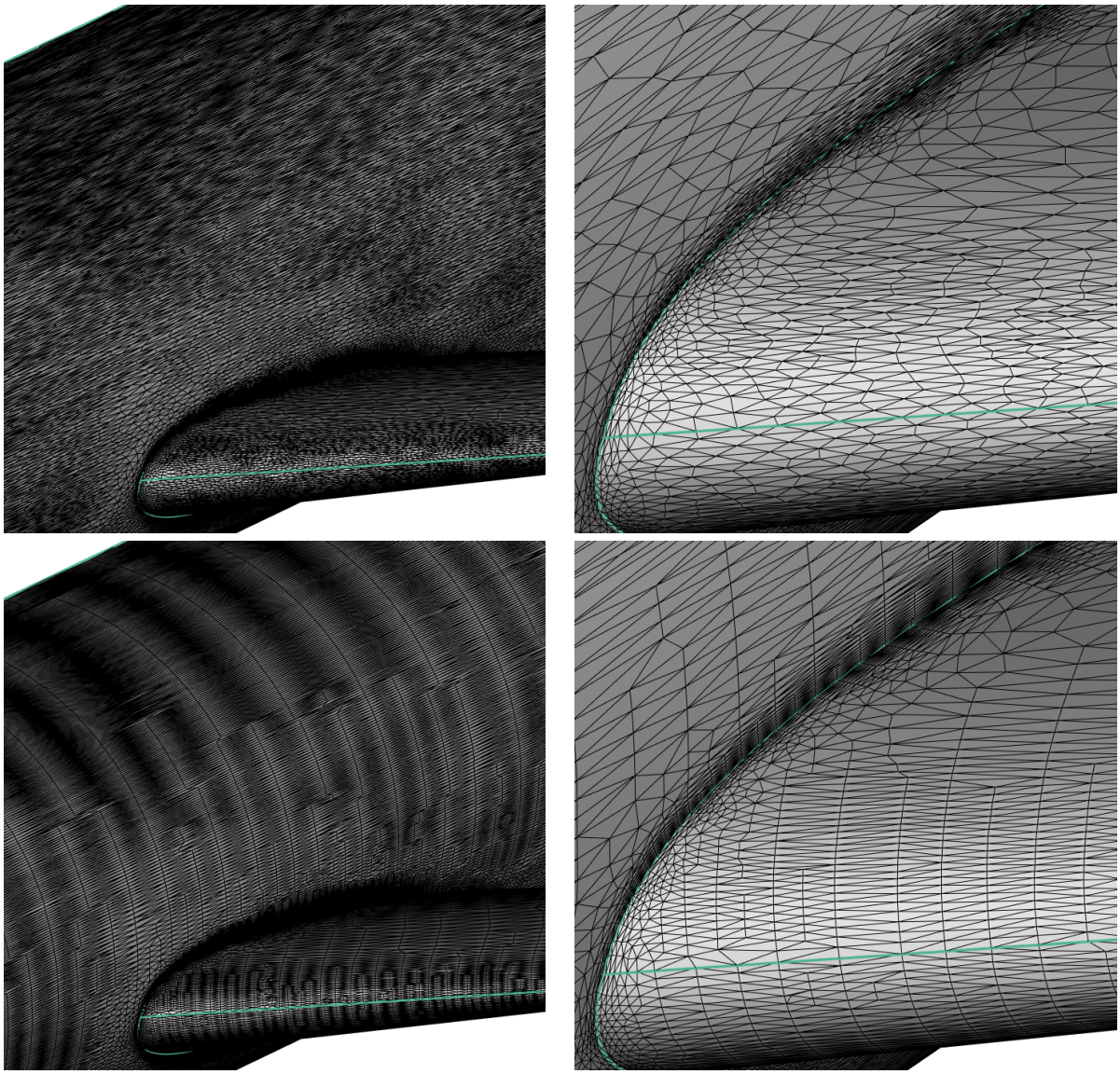


Figure 7.2: DPW2 test case: Generation of two adapted surface meshes with the standard approach (top) and the metric-aligned approach (bottom).

7.4 Numerical results

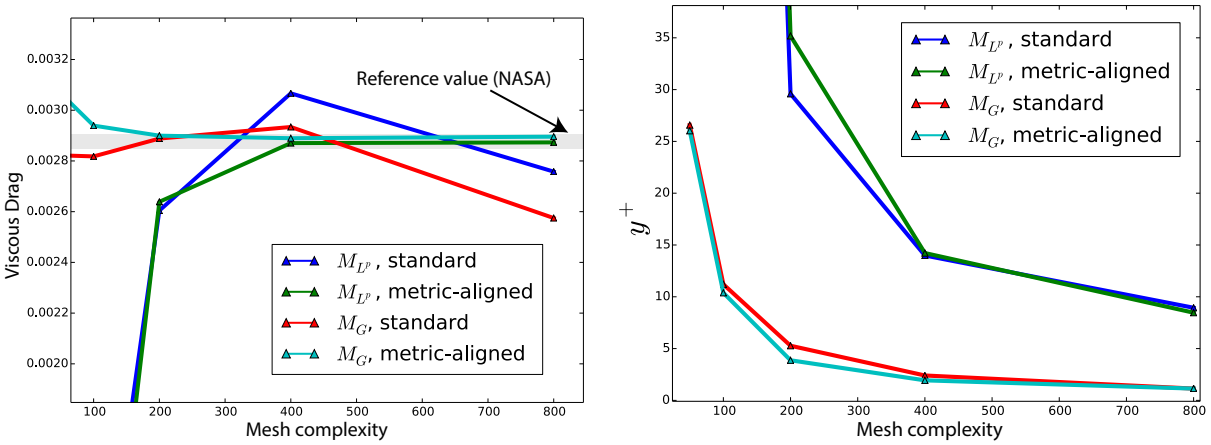
We revisit the flat plate, backward-facing step and RAE 2822 airfoil validation examples of Chapter 4 with the metric-aligned and the multi-field multi-scale metric. For each case, we observe the convergence of the skin friction coefficient along with the resolution of the mesh in the boundary layer y^+.

7.4.1 2D Turbulent flat plate

We consider the turbulent flat plate described in Section 4.2.1. The two aforementioned metrics ($\mathcal{M}_{L^p}$ and $\mathcal{M}_G$) are compared along with the two mesh generation methods: standard and metric-aligned. For all four mesh adaptation processes, 36 iterations of the adaptive loop were performed and the prescribed complexity $\mathcal{N}$ was increased every four iterations, ranging from 50 to 800. The final numbers of vertices are reported in Tables 7.1, 7.2, 7.3, and 7.4.

Results. Scaled close-up views of the four final adapted meshes are presented in Figure 7.4. For each metric, the difference between the two generated meshes (which are both discrete representations of this continuous metric) is pointed out: the standard approach generates a fully unstructured mesh with no-alignment, whereas the metric-aligned produced a quasi-structured boundary layer mesh. For the unstructured version, the level of anisotropy is reduced. The sub and viscous-layer are well represented in the metric-aligned adapted mesh where the accuracy in the boundary layer is diffused for the fully unstructured mesh, 7.4 (bottom). This observation is confirmed by the prediction of the velocity profiles (Figure 7.5) and the prediction of the viscous drag value (Figure 7.3a). Regardless which error estimate is used, the metric-aligned point insertion provides more accurate results.

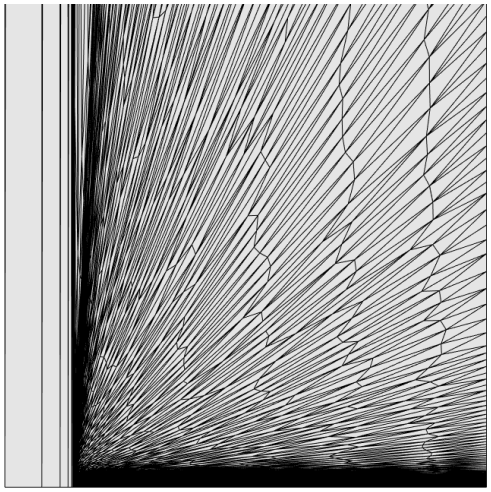
As predicted, the gradient-based metric reaches faster sizes around $y^+ < 1$ than the multi-scale metric, such as shown in Figure 7.3b. However, for each case, y^+ decreases at each time the complexity increases.



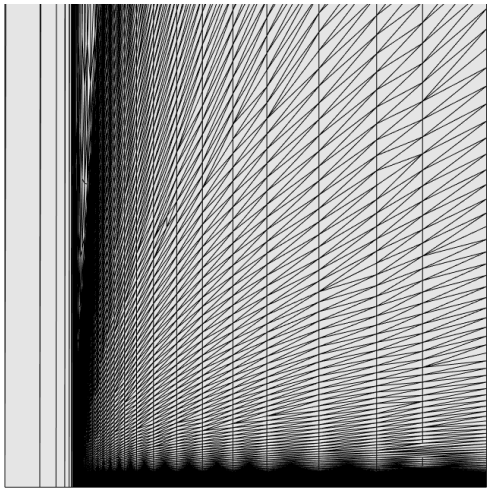
(a) Viscous drag prediction.

(b) y^+ value.

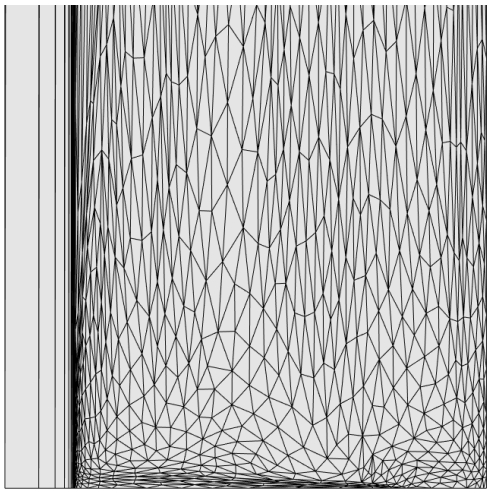
Figure 7.3: Convergence of the viscous drag prediction and the y^+ value.



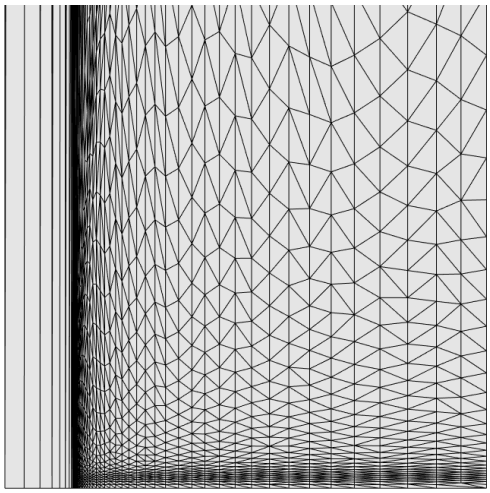
(a) $\mathcal{M}_G$ metric, standard mesh generation.



(b) $\mathcal{M}_G$ metric, metric-aligned mesh generation.



(c) $\mathcal{M}_{L^p}$ metric, standard mesh generation.



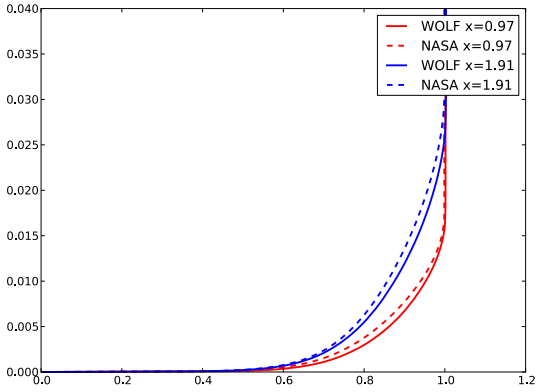
(d) $\mathcal{M}_{L^p}$ metric, metric-aligned mesh generation.

Figure 7.4: Turbulent flat-plate: Close-up views of the meshes (scaled by a factor of 1000 in the y direction).

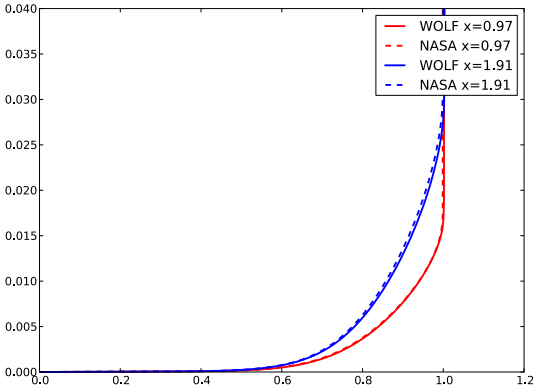
7.4.2 2D backward-facing step

We compared the two metrics for the case of the backward-facing step introduced in Section 4.3.3. This time, only the metric-aligned meshing method was used. The two following simulations were run:

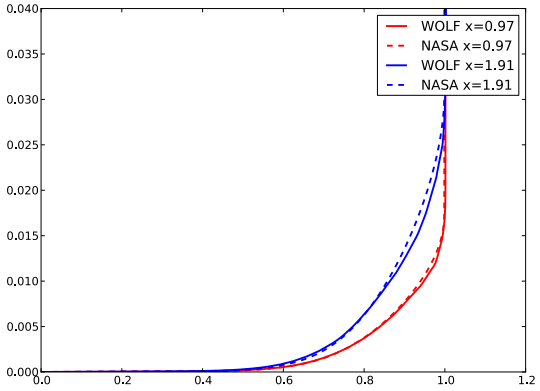
- Metric $\mathcal{M}_G$: 24 adaptive loops were performed, using a mesh complexity $\mathcal{N}$ ranging from 3 000 to 96 000. The corresponding numbers of vertices are presented in Table 7.5.
- Metric $\mathcal{M}_{L^p}$: 36 adaptive loops were run, using a $\mathcal{N}$ ranging from 3 000 to 768 000. These additional adaptive iterations were performed in order to reach the same y^+ value. The corresponding numbers of vertices are presented in Table 7.6.



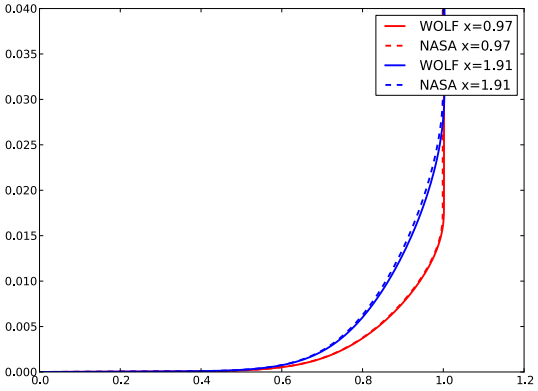
(a) $\mathcal{M}_G$ metric, standard mesh generation.



(b) $\mathcal{M}_G$ metric, metric-aligned mesh generation.



(c) $\mathcal{M}_{L^p}$ metric, standard mesh generation.



(d) $\mathcal{M}_{L^p}$ metric, metric-aligned mesh generation.

Figure 7.5: Turbulent flat-plate: Comparison of the velocity profiles for $\mathcal{M}_G$ metric and $\mathcal{M}_{L^p}$ metric using the standard mesh generation and the metric-aligned.

Results. As shown in Tables 7.5 and 7.6, here also the gradient-based metric presents a faster convergence of the y^+ value to $y^+ < 1$, compared to the multi-scale metric. Close-up views of the meshes generated using the $\mathcal{M}_{L^p}$ for the complexities 24 000 to 36 000 are depicted in Figure 7.6, showing how the viscous layer is progressively refined. The final adapted meshes generated using the $\mathcal{M}_G$ and $\mathcal{M}_{L^p}$ metrics and the corresponding solutions are compared in Figure 7.7. As shown in Figure 7.7, an accurate prediction of the velocity profiles is ensured using the $\mathcal{M}_{L^p}$ metric for the complexity 24 000.

7.4.3 2D RAE 2822

This test case (introduced in Section 4.3.2) is used to validate the adaptation strategy on a curved geometry. Here, we only consider the multi-field multi-scale ($\mathcal{M}_{L^p}$) metric and the metric-aligned mesh generation. 32 adaptive iterations were performed, for a complexity $\mathcal{N}$ ranging from 6 000 to 48 000 (the complexity is increased every four iterations). The corresponding numbers of vertices are given in Table 7.7.

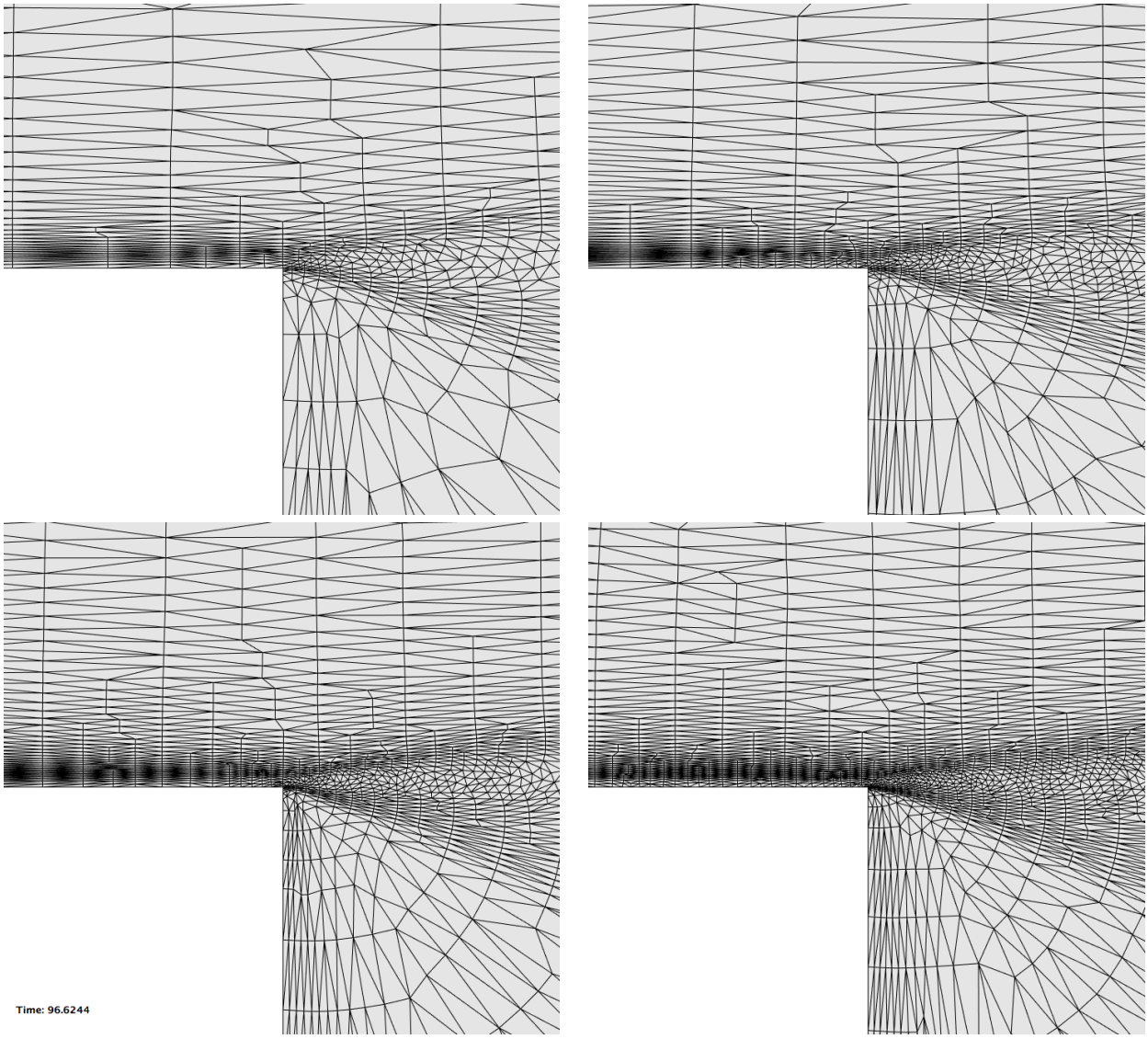


Figure 7.6: Backward facing step: mesh resolution around the corner for the complexity 24 K, to 36 K for the multi-field multi-scale metric from left to right, top to bottom.

Complexity	# vertices	# triangles	y^+	C_f
50	125	212	26.5731	2.821985e-03
100	202	348	11.2054	2.817947e-03
200	400	713	5.27362	2.888213e-03
400	779	1419	2.40335	2.933894e-03
800	1677	3107	1.13789	2.574812e-03

Table 7.1: Flat plate, metric $\mathcal{M}_G$, standard.

Complexity	# vertices	# triangles	y^+	C_f
50	171	303	26.0255	3.066430e-03
100	451	838	10.3843	2.939279e-03
200	993	1885	3.86989	2.899726e-03
400	2081	4003	1.93433	2.889562e-03
800	3695	7129	1.13789	2.895685e-03

Table 7.2: Flat plate, metric $\mathcal{M}_G$, metric-aligned.

Complexity	# vertices	# triangles	y^+	C_f
50	89	148	231.307	4.048006e-04
100	176	299	202.187	7.689662e-04
200	349	640	29.5913	2.604312e-03
400	668	1242	13.9867	3.066622e-03
800	1506	2906	8.94044	2.757870e-03

Table 7.3: Flat plate, metric $\mathcal{M}_{L^p}$, standard.

Complexity	# vertices	# triangles	y^+	C_f
50	100	166	381.001	2.303758e-04
100	229	411	225.159	4.168281e-04
200	458	847	35.179	2.638882e-03
400	921	1744	14.2213	2.870383e-03
800	1742	3343	8.46453	2.873137e-03

Table 7.4: Flat plate, metric $\mathcal{M}_{L^p}$, metric-aligned.

Complexity	# vertices	# triangles	y^+	C_f
3 000	7 328	14 313	5.20736	8.652049e-01
6 000	11 214	21 959	2.22989	8.771854e-01
12 000	19 106	37 601	0.816076	8.774204e-01
24 000	31 889	63 001	0.440237	8.952100e-01
48 000	38 472	76 128	0.426837	8.778391e-01
96 000	39 695	78 556	0.468967	8.825638e-01

Table 7.5: Backward-facing step: metric $\mathcal{M}_G$: number of vertices and values of y^+ and C_f obtained.

Results. The y^+ value decreases as the mesh complexity increases, as shown in Table 7.7. Views of the final adapted mesh are depicted in Figure 7.9, showing how the wake and the shock (in the near-wall region) are captured. The corresponding solution fields (pressure, density, velocity and turbulent viscosity) are presented in Figure 7.10.

Complexity	# vertices	# triangles	y^+	C_f
3 000	4 948	9 584	37.9324	7.957966e-01
6 000	11 566	2 2701	9.37787	8.741370e-01
12 000	24 124	47 573	5.91782	8.733467e-01
24 000	46 379	91 722	5.83077	8.737102e-01
48 000	88 017	174 379	3.80343	8.738171e-01
96 000	164 157	325 670	1.61118	8.732349e-01
192 000	301 733	599 070	1.70682	8.728550e-01
384 000	556 805	1 105 037	1.13617	8.693202e-01
768 000	1 047 896	2 072 419	0.969889	8.400578e-01

Table 7.6: Backward facing step: metric $\mathcal{M}_{L^p}$: number of vertices and values of y^+ and C_f obtained.

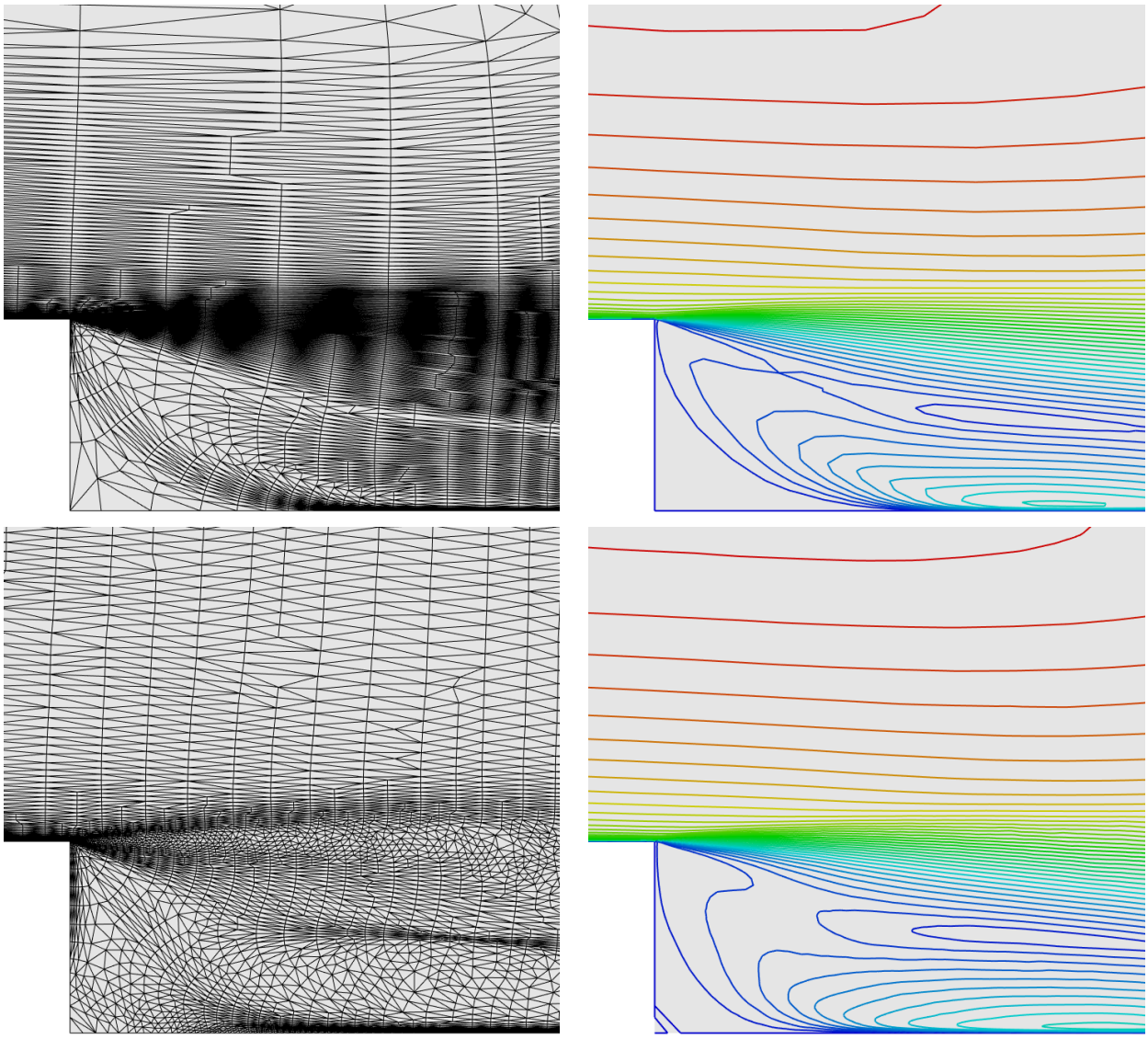


Figure 7.7: Backward facing step: Left, comparison of the meshes generated from $\mathcal{M}_G$ (top) and $\mathcal{M}_{L^p}$ (bottom) near the corner at complexity 24 000. Right, iso-values of the norm of the velocity.

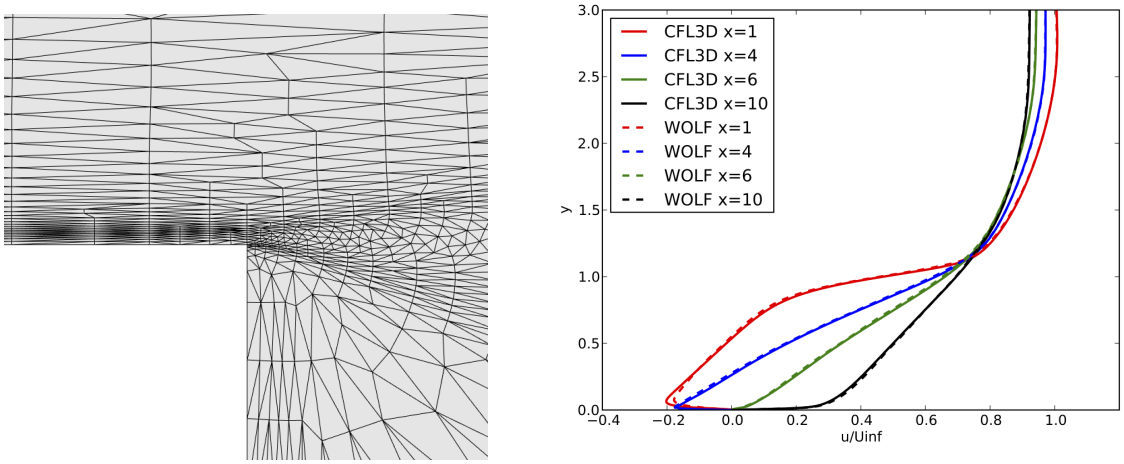


Figure 7.8: Backward facing step, metric $\mathcal{M}_{L^p}$: Left, mesh at complexity 24 000, and comparison to CFL3D (whose solution was computed on the uniform reference mesh) of the velocity profiles.

Complexity	# vertices	# triangles	y^+	C_f
6 400	12 582	24 648	34.8636	3.692674e-04
12 000	22 984	45 292	20.3588	6.569288e-04
24 000	46 155	91 168	6.4739	1.859531e-03
48 000	90 074	178 260	5.11535	2.124196e-03

Table 7.7: RAE 2822, metric $\mathcal{M}_{L^p}$ and metric-aligned : number of vertices and values of y^+ and C_f obtained.

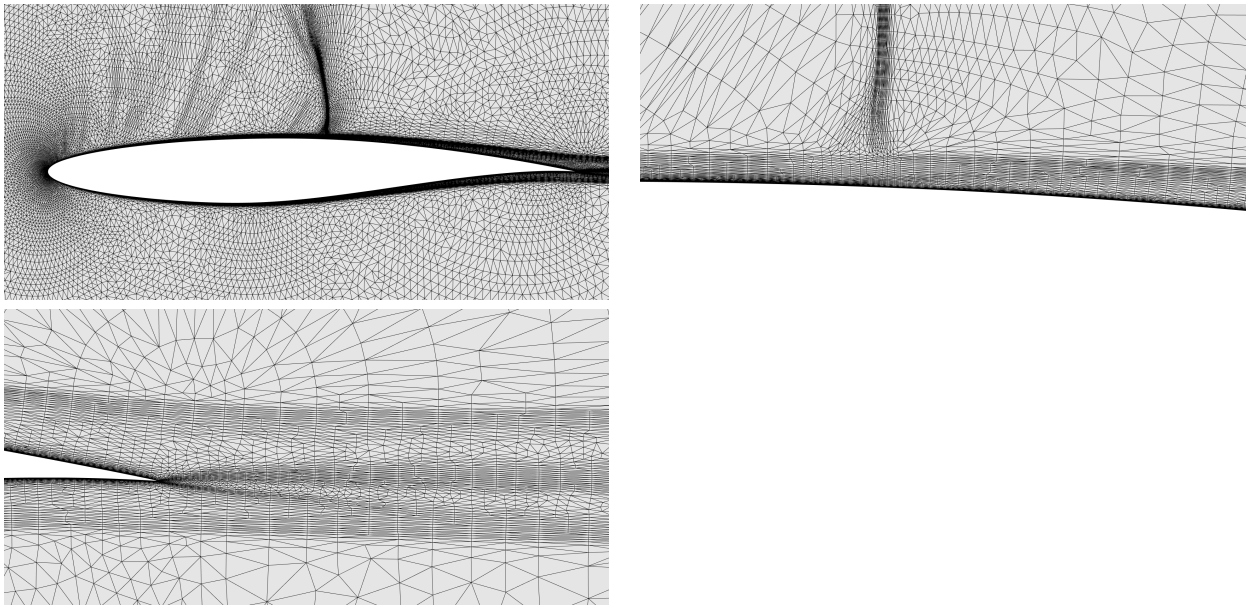
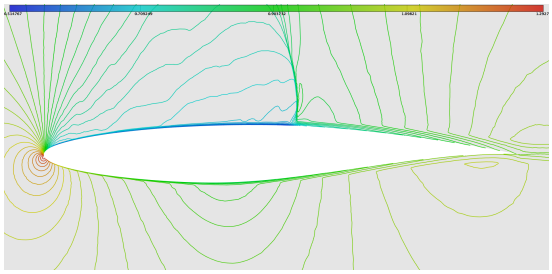
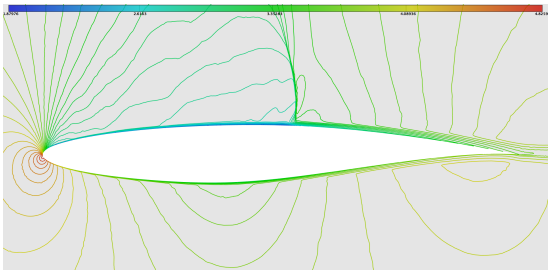


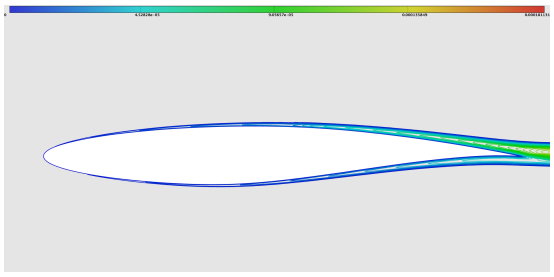
Figure 7.9: RAE 2822: Views of the final adapted mesh using the $\mathcal{M}_G$ metric and the metric-aligned meshing method.



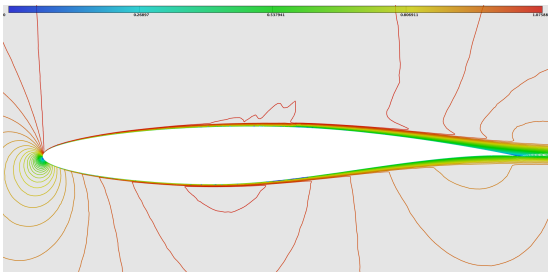
(a) Density.



(b) Pressure.



(c) Turbulent viscosity.



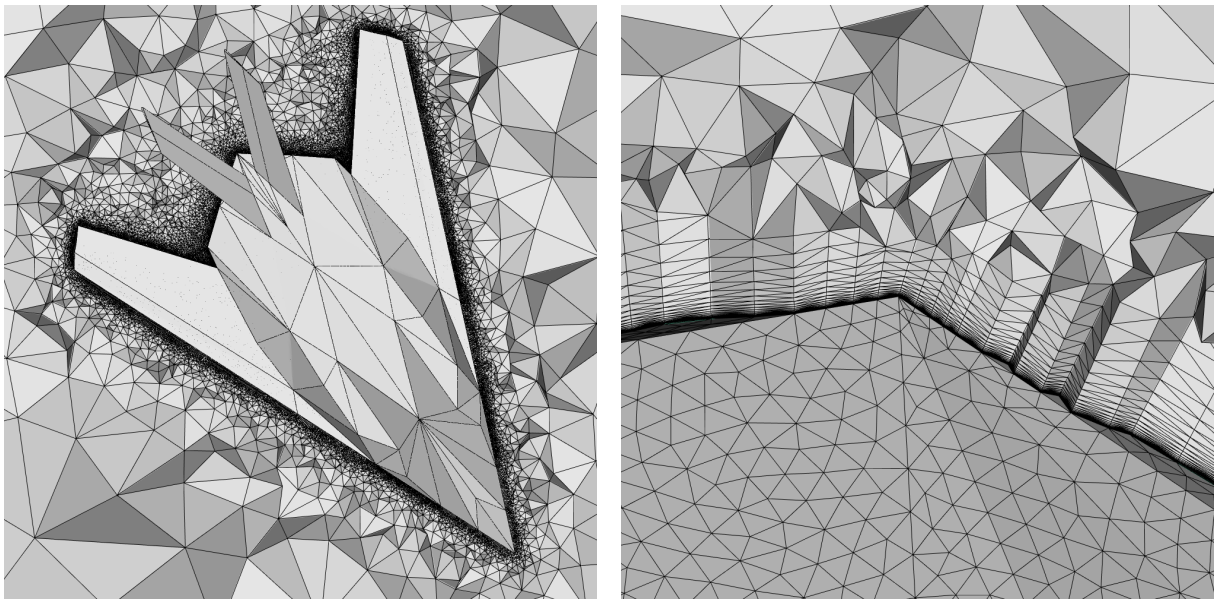
(d) Velocity.

Figure 7.10: RAE 2822: isolines of the solution on the final adapted mesh.

7.4.4 3D F117

In this section, we present our first attempt of adaptive RANS simulation in 3D. In order to prevent issues related to surface and CAD reprojection, we used a F117 geometry that we modified, so that every surface patch becomes piecewise linear. The simulation is run at Mach $M = 0.8$ with an angle of attack $\alpha = 7°$. The Reynolds number is set to $5M$ based on the half-wing span of the aircraft. The temperature is set to $300K$. Views of the initial uniform mesh are presented in Figure 7.11, and the corresponding solution (pressure) is shown in Figure 7.12.

We have performed two adaptive RANS simulations: a mesh adaptation of the surface mesh, and a mesh adaptation of the volume mesh. For both these simulations, the multi-field multi-scale error estimate $\mathcal{M}_{L^p}$ is employed, together with the metric-aligned point insertion strategy.



(a) Cut in the volume mesh. (b) Close-up view of the boundary layer mesh.

Figure 7.11: F117: Initial uniform mesh.

Results. The final adapted surface mesh is depicted in Figure 7.13. We were able to capture transonic features of the flow using anisotropic surface elements, while ensuring a good mesh quality. Also, quasi-structured elements were automatically created around the leading and the trailing edge of the wing. The final adapted volume mesh is shown in Figure 7.15. The solution computed using this adapted volume mesh is presented in Figure 7.16. Using this approach, quasi-structured volume elements were created both in the immediate vicinity of the aircraft and in the wake region, where the metric provided is anisotropic and aligned with the flow.

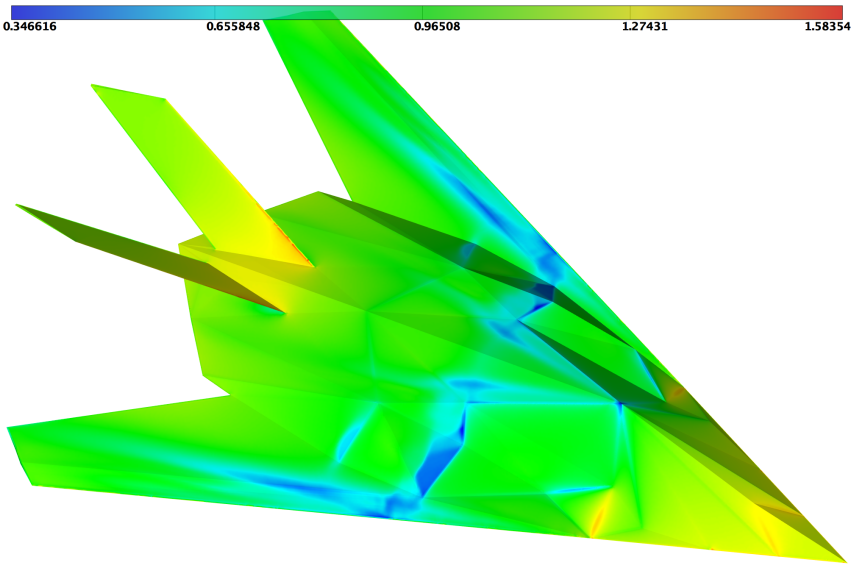
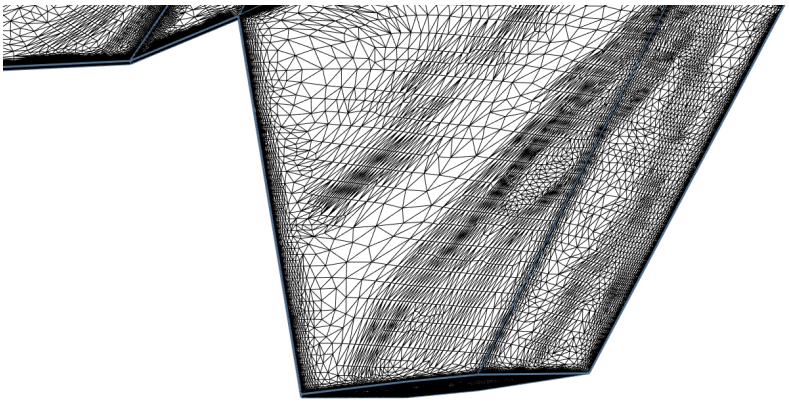
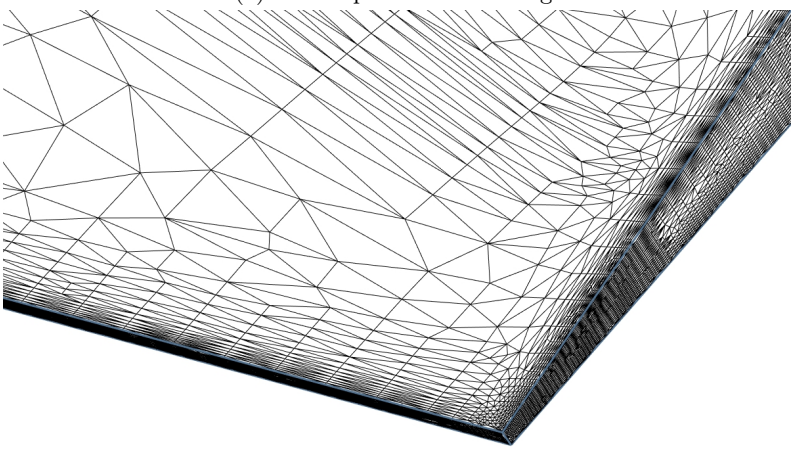


Figure 7.12: F117: Solution (pressure) computed using the initial uniform mesh.

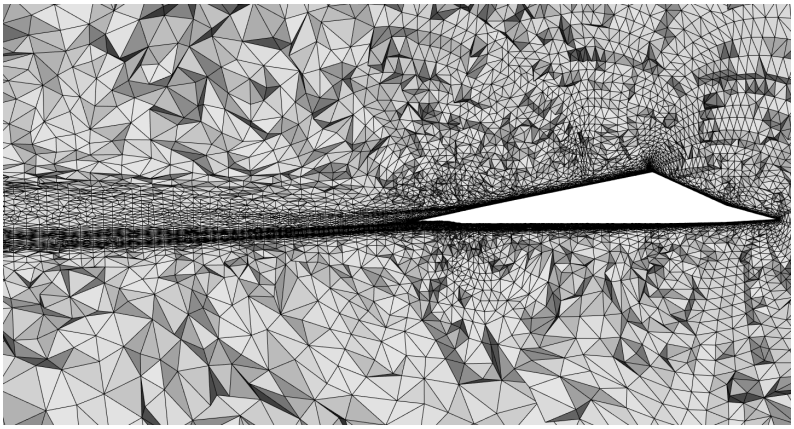


(a) Close-up view of the wing.

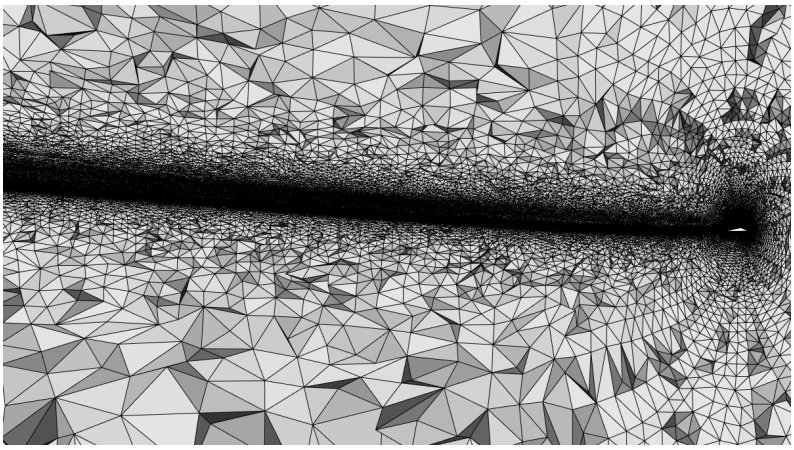


(b) Close-up view of trailing edge.

Figure 7.13: F117: Final adapted surface mesh.



(a) Cut in the volume mesh.

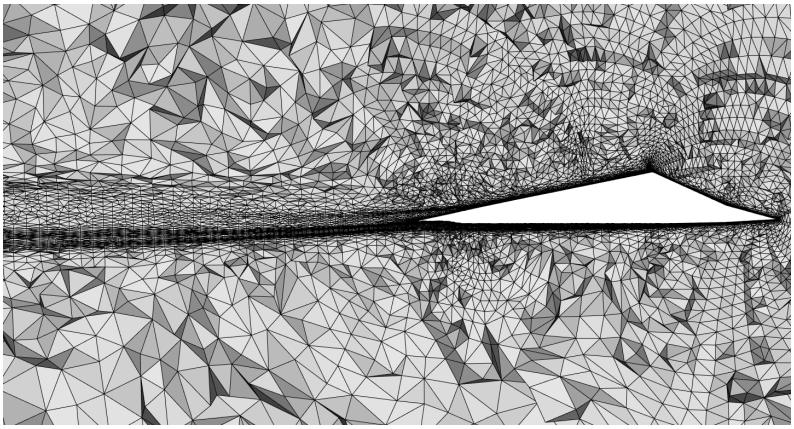


(b) Large view of the wake region.

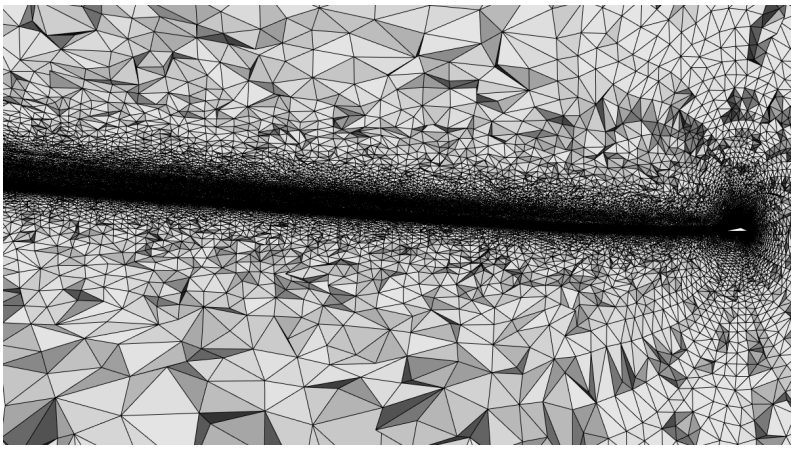
Figure 7.14: F117: Final adapted volume mesh.

7.5 Conclusion and remaining challenges

The extension of multi-scale mesh adaptation to viscous flows is discussed in 2D. Even on fully unstructured meshes, the mixed finite element/volume approach described in Chapter 3 allows us to obtain an accurate prediction of the boundary layer. Even if the multi-field multi-scale ($\mathcal{M}_{L^p}$) metric does not make it possible to control explicitly the size of the first layer and the growth rate of the boundary-layer mesh, $y^+ < 1$ is quickly reached during the adaptive procedure, *i.e.,* for an affordable complexity with respect to the structured reference meshes. However, the quality of the results along with the speed of the resolution are greatly improved by using metric-aligned or metric-orthogonal strategy to generate the adaptive mesh. Indeed, such an approach can handle an arbitrary high level of anisotropy while generating high-quality elements. This strategy is based on an ordered advancing-point algorithm where the vertices are first created and then inserted in a second step. A first example with this approach is provided for a wing-body configuration. The error estimate implies a strong anisotropy of the order of $O(1 - 10^5)$ on the surface mesh. Then, the main difficulty is to maintain a good surface approximation while conforming to the level of anisotropy of the metric.



(a) Cut in the volume mesh.

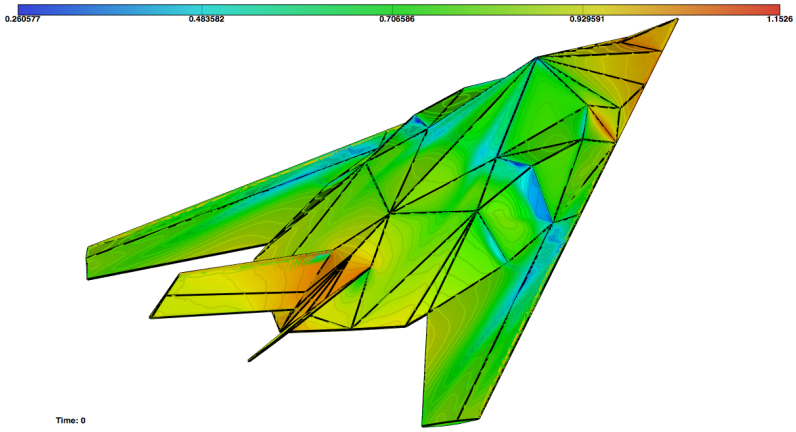


(b) Large view of the wake region.

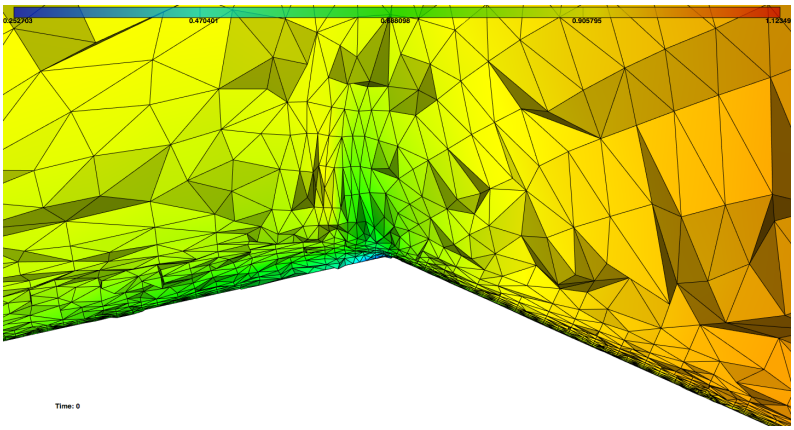
Figure 7.15: F117: Final adapted volume mesh.

Consequently, fully adaptive viscous flow simulations in 3D for complex geometries remain a challenging research issue. The high variations of the flow solution coupled with the complexity of the geometry point out the weaknesses of each component of the adaptive loop. When the multi-scale multi-field error estimate is applied to a 3D configuration, the resulting estimates show that typical tailored meshes are highly under-resolved on the surface even at low complexity. For instance, a natural boundary layer mesh appears at the wing-fuselage junction, whereas standard meshes have a uniform large spacing. To conclude, we give an non-exhaustive list of the remaining difficulties.

Surface remeshing and CAD-projection. When a local remeshing approach is used, each modification of the surface mesh is also performed on the volume to ensure that the validity of the mesh is maintained. If a boundary layer mesh exits from initial or previous iterations, its presence may constrained the projection on the new surface point. For instance, to ensure the validity of the mesh, several layers must be removed locally, see Figure 7.17. Consequently, simple edge-based operators [104] (edge collapse or edge insertion) are not sufficient to handle such kind of configurations.



(a) Pressure on the surface of the aircraft.

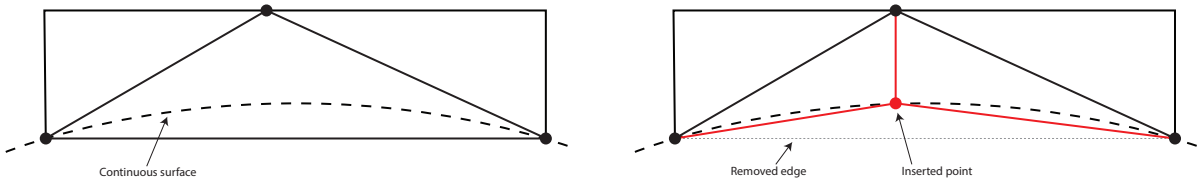


(b) Close-up view of the near-wall region.

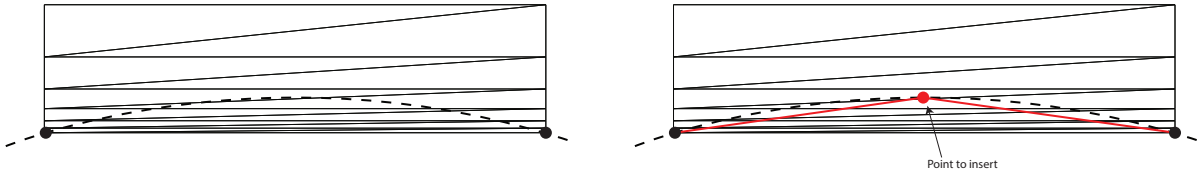
Figure 7.16: F117: Final solution computed using the adapted volume mesh.

As mentioned above, near-wall regions require strong anisotropy in the normal plane to the surface. This normal anisotropy makes it difficult to adapt the underlying surface mesh. It can also cause an inaccurate computation of the surface metric. Even more difficulties appear in presence of anisotropic surface elements. Note that this projection issue is crucial for RANS simulations, as the prediction of viscous forces strongly depends on the quality of the surface approximation and on gradient evaluation. Even a small alteration of the surface definition may drastically spoil the numerical solution. We give an illustration of the impact of using the CAD to project the point in the geometry. In Figure 7.18, we consider a geometric Falcon geometry provided by Dassault-Aviation. We consider a discrete surface mesh of composed of 631 000 vertices and 1 263 932 triangles. We display the faceted mesh the surface mesh before and after projection on the CAD data. For each pictures, the projected and unprojected mesh are depicted. For this example, EGADS [68] based on top of OpenCascade is used to query the CAD.

Error estimates for RANS simulations. If there exist theoretical developments for goal-oriented error estimates for RANS equations [170, 171], by their very nature, they allow the user only to control one scalar output functional (such as the lift and the drag coefficients). Consequently, there is no guarantee of obtaining a fully converging flow field, *i.e.,* for all the flow variables. It therefore seems necessary to extend norm oriented approaches or entropy variables to RANS equations.



(a) Isotropic case : initial mesh (left) and modified mesh (right).



(b) Viscous case. The anisotropy in the immediate vicinity of the surface makes it difficult to refine it.

Figure 7.17: Illustration of one difficulty of surface reprojection when boundary layer meshes are involved.

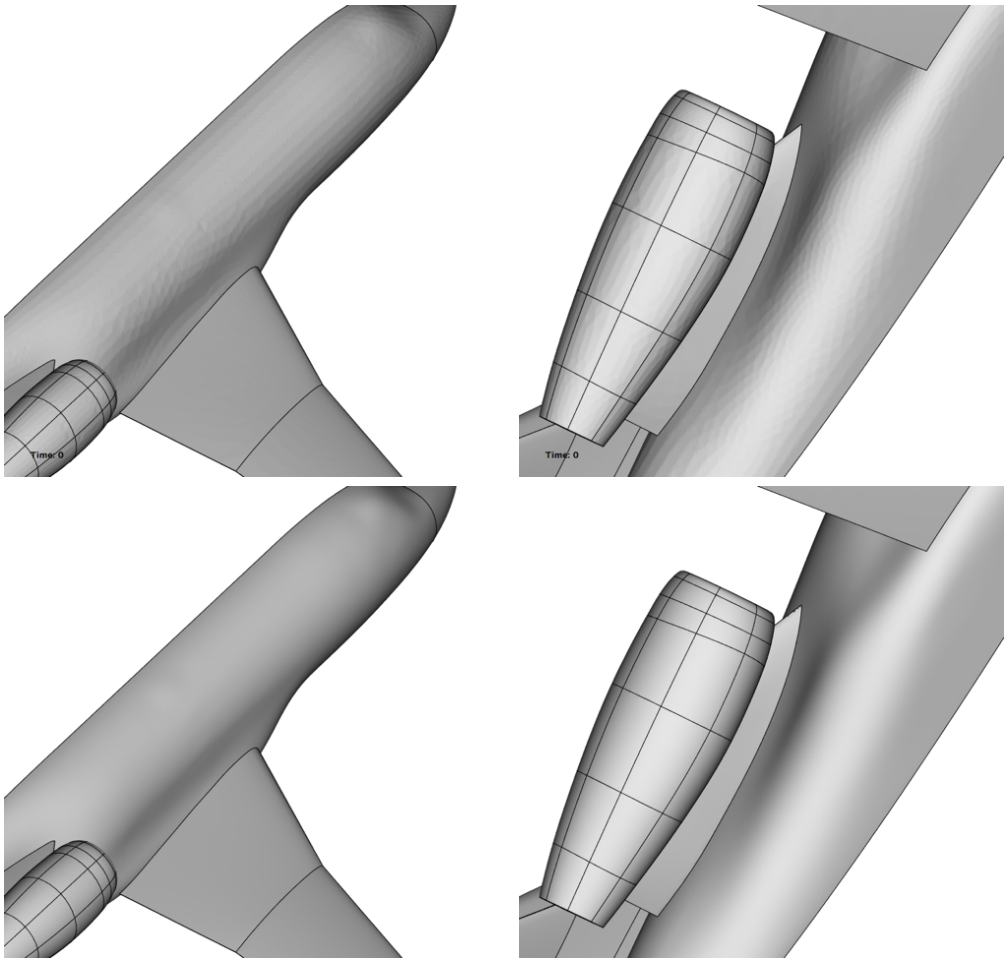


Figure 7.18: CAD projection: Falcon case: Illustration of the surface approximation without (top) and with (bottom) projecting the mesh to the CAD.

Conclusion

In this thesis, we have presented our contributions to some of the research issues that remain regarding 3D RANS adaptive simulations. These contributions include work on both numerical methods (flow solver) and mesh adaptation strategies.

The numerical flow solver is an essential component of the mesh adaptation procedure, during which a solution is computed at each iteration of the adaptive loop. First, we have implemented the Spalart-Allmaras turbulence model and carried out a rigorous verification & validation study, which consists in comparing our numerical solution to experimental data and other well-established numerical solvers. Accurate results were obtained for a representative set of cases encountered in aeronautics, including the drag prediction workshop.

In the context of RANS simulations, meshes are larger and flows are more complex, which is why 3D simulations cannot be envisaged without accelerating convergence and improving robustness. To this end, we have implemented an implicit time integration and accelerated the convergence of the linear system (solved at each solver iteration) thanks to a multigrid procedure. The validation of this implicit multigrid procedure has shown a significant improvement in robustness and the convergence rate of the flow solver. Appropriate CFL laws are mandatory to achieve fast convergence in solving non-linear equations, but are too dependent on parameters set by the user. To avoid this issue, we implemented a local (i.e. a CFL value for each vertex) dynamic CFL law. All the new routines were parallelized using a shared-memory approach based on pthreads, using an in-house library that automatically deals with indirect addressing.

As regards mesh adaptation, we also have improved the robustness and the rapidity of the adaptive process, in order to deal with the increased complexity induced by RANS simulations. We have extended the full multigrid (FMG) algorithm to an adaptive context in order to benefit from its interesting convergence properties. The validation study of the adaptive FMG algorithm on 3D cases has confirmed the FMG theory, leading to increased robustness and a reduction in the computational effort. We have also devised a distributed parallel mesh generation algorithm for small scale parallel architectures (less than 1000 cores) such as are typically found in most R&D units. We have been able to generate anisotropic adapted meshes containing around one billion elements in less than 20 minutes on 120 cores. Finally,

we have worked on adaptive meshing strategies for viscous near-wall regions (boundary layers). We introduced a procedure to automatically generate anisotropic adapted quasi-structured meshes of high-quality, which consists in taking into account the natural alignment and orthogonality of the provided input metric field during the mesh generation process.

Appendix

Appendix A

Side Project: 3D Parallel Anisotropic Unsteady Mesh Adaptation for the High-Fidelity Prediction of Bubble Motion

Anisotropic unsteady mesh adaptation is applied to the high-fidelity prediction of bubble motion, which has applications in the framework of safety evaluations for nuclear reactors. A prescribed advection of the bubble is performed, which lacks any physical sense but is representative of the reality and makes it possible to precisely measure the diffusion caused by the numerical model. The model is described, and results are presented in 2D and 3D, with comparisons in terms of mesh convergence, CPU time, and propagation of the interface.

Introduction

In this chapter, the high-fidelity prediction of the propagation of an extremely thin interface is addressed from the meshing point of view. Applications exist in the framework of safety evaluations for nuclear reactors, in which gas bubbles may appear in the liquid phase. In this context, the meshing strategy used must deal with the discontinuities of most variables through the bubble's interface. Here, the contribution of anisotropic unsteady mesh adaptation to this issue is discussed, which aims at increasing the accuracy of the solution while decreasing the CPU time of the simulation, by dividing the physical time frame considered into sub-intervals for each of which an anisotropic mesh is generated according to size and directional constraints.

In order to measure how well the numerical model predicts bubble motion, the Kothe-Rider test [145] is performed. An initial sphere is linearly advected, governed by a velocity field $\vec{v}(x, y, z, t)$ of period T, and starting from $t = 0$. Due to the periodicity of the velocity field, the bubble is expected to recover its original position (i.e. the sphere) at each $t \in \frac{T}{2}\mathbb{N}$. Although this bubble advection lacks any physical sense, it is representative of the reality, and makes it possible to precisely estimate the numerical error due to the meshing strategy at each $t \in \frac{T}{2}\mathbb{N}$.

Several studies of numerical methods for propagating an extremely thin interface have been carried out. Adaptive Mesh Refinement (AMR) techniques have been used [139, 159] as well as level set methods coupled with anisotropic mesh adaptation [38, 28].

A.1 Numerical Model

This Section describes the numerical model used for performing the Kothe-Rider test, including the advection solver, as well as the steady and unsteady mesh adaptation algorithms.

Using anisotropic mesh adaptation for predicting bubble motion is motivated by the features of this physical phenomena, which (i) is concentrated in a small area of the computational domain, (ii) is anisotropic, and (iii) is time-dependent. Therefore, uniform meshes - i.e. meshes whose edges size is constant in the domain - are not optimal in terms of both sizes and directions. Mesh adaptation, however, provides a way to control the accuracy of the numerical solution by modifying the domain discretization according to size and directional constraints. For instance, unstructured Hessian-based mesh adaptation has already proved its efficiency to improve the solution accuracy while decreasing the problem complexity (i.e. the number of degrees of freedom).

A.1.1 Advection Solver

In this Section, the advection solver used for performing the Kothe-Rider test case is described. It is based on our in-house flow solver `Wolf`. The advection equation is the following:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0, \tag{A.1}$$

where ρ is the density (see Figure A.1) and $\vec{v}(x, y, z, t)$ a velocity field.

The spatial discretization of Eq. A.1 is based on a vertex-centered finite volume formulation on unstructured meshes. Let $\mathcal{H} = (K_i)$ be a mesh of a domain Ω, the vertex-centered finite volume formulation consists in associating to each vertex P_i of the mesh a control volume or finite volume cell, denoted C_i.

The discretized domain Ω_h can be written as the union of the mesh elements or the union of the finite volume cells:

$$\Omega_h = \bigcup_{i=1}^{N_T} K_i = \bigcup_{i=1}^{N_S} C_i \,.$$

The dual finite volume cells used for the bubble motion are the classical median cells, as depicted in Figure A.2.

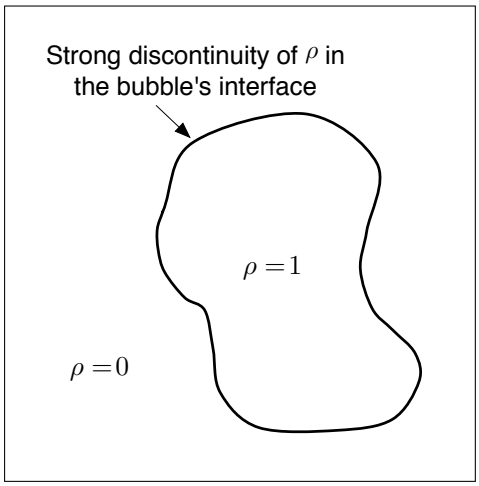


Figure A.1: The density ρ is discontinued through the bubble's interface.

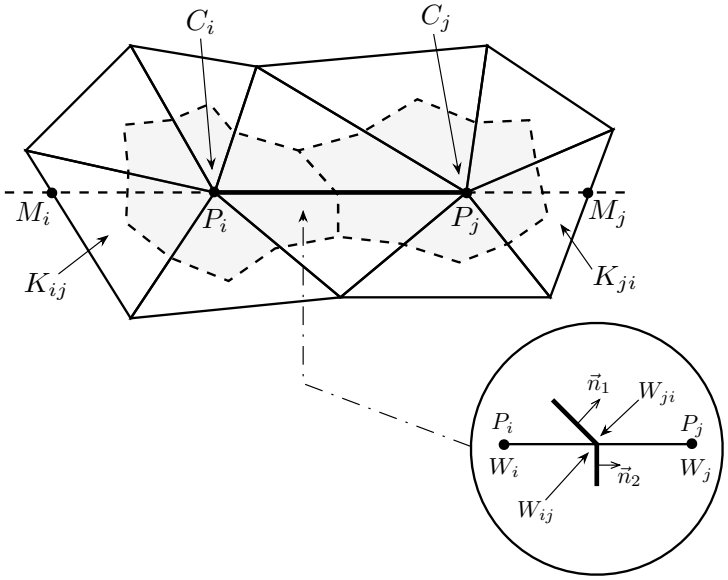


Figure A.2: Illustration of two finite volume control cells C_i and C_j around two vertices P_i and P_j.

Second order space accuracy is achieved through a piecewise linear interpolation based on the Monotonic Upwind Scheme for Conservation Law (MUSCL) procedure with a particular edge-based formulation with upwind elements. The advection flux through an edge $\overrightarrow{P_iP_j} = \vec{e}_{ij}$ (see Figure A.2) is given by:

$$\Phi_{ij}^{\mathtt{adv}} = \begin{cases} v_{moy}.\rho_i.\,\|\vec{e}_{ij}\| & \text{if} \quad v_{moy} > 0 \\ v_{moy}.\rho_j.\,\|\vec{e}_{ij}\| & \text{else.} \end{cases} \tag{A.2}$$

where ρ_i is the solution at the vertex P_i,

$$v_{moy} = \frac{1}{2}(v_i + v_j),$$

$$\begin{cases} v_i &= \vec{v}(P_i,t).\vec{n} \\ v_j &= \vec{v}(P_j,t).\vec{n} \end{cases},$$

and $\vec{n}$ is the edge's normal vector.

The MUSCL type reconstruction method is used in order to increase the order of accuracy of the scheme [86]. The idea is to use extrapolated values ρ_{ij} and ρ_{ji} of ρ at the interface ∂C_{ij} to evaluate the flux. The following approximation is performed:

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

ρ_{ij} and ρ_{ji} which are linearly interpolated as:

$$\rho_{ij} \;=\; \rho_i + \frac{1}{2}\,(\nabla\rho)_{ij} \cdot \overrightarrow{P_iP_j} \qquad \text{and} \qquad \rho_{ji} \;=\; \rho_j + \frac{1}{2}\,(\nabla\rho)_{ji} \cdot \overrightarrow{P_jP_i} \,,$$

where, in contrast to the original MUSCL approach, the approximate "slopes" $(\nabla\rho)_{ij}$ and $(\nabla\rho)_{ji}$ are defined for any edge and obtained using a combination of centered, upwind and nodal gradients.

The centered gradient related to an edge P_iP_j, is defined as:

$$(\nabla\rho)^C_{ij} \cdot \overrightarrow{P_iP_j} = \rho_j - \rho_i \,.$$

Upwind and downwind gradients are computed according to the definition of upstream and downstream tetrahedra of an edge P_iP_j. These tetrahedra are respectively denoted K_{ij} and K_{ji}. K_{ij} (resp. K_{ji}) is the unique tetrahedron of the ball of P_i (resp. P_j) the opposite face of which is crossed by the line defined by the edge P_iP_j. Upwind and downwind gradients are then defined for vertices P_i and P_j as:

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

where $(\nabla\rho)|_K = \sum_{P\in K} \rho_P \nabla\phi_P|_K$ is the P_1-Galerkin gradient on tetrahedron K. Parametrized nodal gradients are built using the β-scheme:

$$\begin{aligned} (\nabla\rho)_{ij} &= (1-\beta)(\nabla\rho)^C_{ij} + \beta\,(\nabla\rho)^U_{ij} \\ (\nabla\rho)_{ji} &= (1-\beta)(\nabla\rho)^C_{ij} + \beta\,(\nabla\rho)^D_{ij} \,, \end{aligned}$$

where $\beta \in [0,1]$ is a parameter controlling the amount of upwinding. For instance, the scheme is centered for $\beta = 0$ and fully upwind for $\beta = 1$.

Bubble motion is predicted using a V4-scheme, obtained for $\beta = 1/3$. It can be demonstrated that this scheme is third-order for the two-dimensional linear advection on structured triangular meshes. On

unstructured meshes, a second-order scheme with a fourth-order numerical dissipation is obtained. High-order gradients are given by:

$$\begin{aligned}(\nabla\rho)_{ij}^{V4} &= \frac{2}{3}(\nabla\rho)_{ij}^{C} + \frac{1}{3}(\nabla\rho)_{ij}^{U} \\ (\nabla\rho)_{ji}^{V4} &= \frac{2}{3}(\nabla\rho)_{ji}^{C} + \frac{1}{3}(\nabla\rho)_{ij}^{D}\,.\end{aligned}$$

The parallelization of the solver is based on posix standard threads (pthreads) taking advantage of multi-core chips and shared memory architectures supported by most platforms. Loops running over tables and structures featuring direct or indirect memory accesses take up a large part of the total CPU time when dealing with meshes, and are easily parallelized with pthreads (see Chapter 3).

A.1.2 Steady Mesh Adaptation

The general idea of mesh adaptation is to modify the discretization of the computational domain according to size and directional constraints, in order to minimize a given error criterion, and thus improve the adequation with the underlying physics. Mesh adaptation has proved its efficiency in improving the tradeoff between computational time and accuracy of the solution. Figure A.3 presents the example of mesh adaptation for the recovery of a bubble's interface in the steady case. The adaptation is performed on the density variable ($\rho = 1$ inside the bubble and 0 outside), so that the mesh is refined close to the interface and coarsened elsewhere. As the solution varies dramatically in the normal direction to the interface and does not vary in the tangential direction, stretched elements aligned to the direction of anisotropy are created.

Isotropic mesh adaptation simply relies on the prescription of a scalar size field. Anisotropic mesh adaptation, however, must control the sizes along prescribed directions. To this end, we use the unit-mesh concept in the continuous mesh framework. The main idea is to generate a uniform mesh with respect to a Riemannian metric space rather than to the Euclidian space. According to the continuous mesh framework, any mesh can be represented by a continuous Riemannian metric field $\mathcal{M}$. The link between a continuous and a discrete mesh is based on the concept of unit-mesh: a mesh is unit according to $\mathcal{M}$, if all its edges have a length $l_{\mathcal{M}}(e)$ in the metric approximately equal to 1 and if its elements K have a volume $|K|_{\mathcal{M}}$ in the metric approximately equal to $\sqrt{2}/12$. More formally, a metric tensor $\mathcal{M}$ in $\mathbb{R}^n$ is a $n \times n$ symmetric definite positive matrix. The scalar product of two vectors $\vec{u}$ and $\vec{v}$ in $\mathbb{R}^n$ according to $\mathcal{M}$ is defined as:

$$\langle \vec{u}, \vec{v} \rangle_{\mathcal{M}} = \langle \vec{u}, \mathcal{M}\vec{v} \rangle = {}^t\vec{u}\mathcal{M}\vec{v} \in \mathbb{R}.$$

So, the associated norm of a vector in $\mathbb{R}^n$ is defined as:

$$\|\vec{u}\|_{\mathcal{M}} = \sqrt{\langle \vec{u}, \vec{u} \rangle_{\mathcal{M}}} = \sqrt{{}^t\vec{u}\mathcal{M}\vec{u}},$$

which measures the length of the vector $\vec{u}$ in the metric $\mathcal{M}$. Thus, the length of an edge $e = AB$ according to $\mathcal{M}$ is defined:

$$l_{\mathcal{M}}(e) = \int_0^1 \sqrt{{}^t AB \mathcal{M}((1-t)A + tB)AB}. \tag{A.3}$$

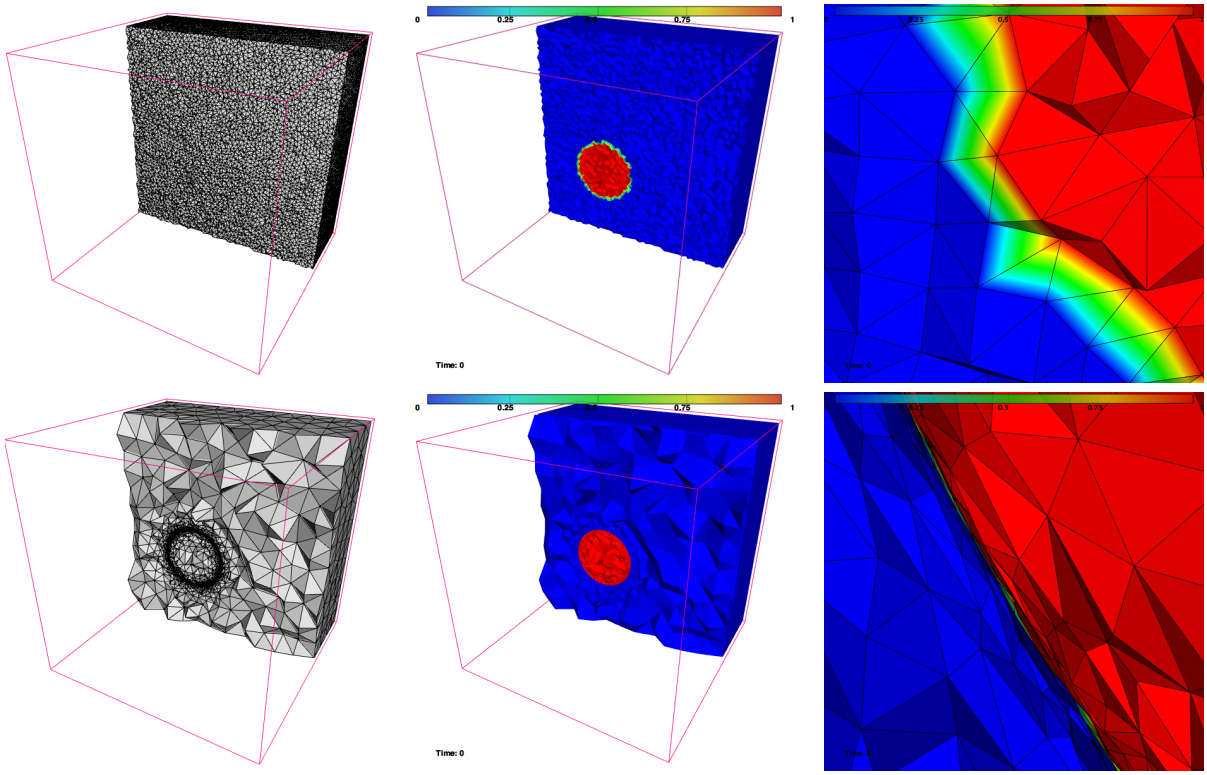


Figure A.3: Steady mesh adaptation at time $t = 0$. Top: cuts in the initial uniform mesh. Bottom: cuts in the final adapted mesh which is refined at the bubble's interface.

Mesh adaptation consists in generating a mesh that is unit according to a Riemanninan metric field obtained from an error estimation of the solution. We now describe the mesh adaptation process in the steady case. We start from an initial (coarse and non-adapted) mesh $\mathcal{H}_0$. The four main stages of the process are the following: (i) solution computation on $\mathcal{H}_0$ ($\mathcal{S}_0^0$), (ii) metric computation ($\mathcal{M}_0$), (iii) mesh generation ($\mathcal{H}_1$) using the sizes and directions provided by $\mathcal{M}_0$, (iv) solution interpolation to $\mathcal{H}_1$.

As presented in Figure A.4, these four stages are repeated several times and at each iteration, the complexity of the generated meshes is increased. It makes it possible to converge both the mesh and the solution to an optimal state and to capture accurately physical phenomena. The classical steady mesh adaptation scheme is a fixed point algorithm: the algorithm stops when there is no variation of the couple mesh/solution from one iteration to the next. More details on each stage are now provided.

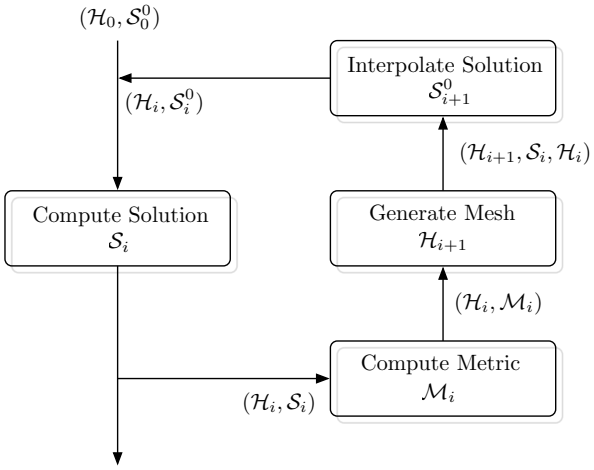


Figure A.4: The mesh adaptation loop

(i) Solution computation We use the advection solver described in Section A.1.1 to compute the density variable ρ at each vertex of the mesh.

(ii) Metric computation. We chose to minimize the $\mathbb{P}^1$ interpolation error $\rho - \Pi_h \rho$.

(iii) Mesh generation. The results presented in this paper were achieved using our in-house remesher `AMG` [108] (described in Chapter 3). The general idea is to perform iteratively simple mesh modifications such as vertex insertions/removal, edge swaps/collapses etc., in order to generate unit mesh elements. All the aforementioned operations are performed using a single mesh operator based on cavity remeshing [108].

(iv) Solution interpolation. The interpolation operator used verifies the properties of mass conservation, $\mathbb{P}^1$-exactness (order 2) and maximum principle, which are achieved through local mesh intersections and quadrature formulae.

A.1.3 Unsteady Mesh Adaptation

The random progression of the bubble's interface in the computational domain is a time-dependent problem, which makes the steady mesh adaptation algorithm inadequate. Indeed, the mesh generated for the time $t = 0$ would lead to a strong error in both time and space for the rest of the time frame. A non-optimal approach would be to perform a steady mesh adaptation at each time step, but this would be too costly in terms of CPU. In order to minimize the number of mesh adaptations, an unsteady mesh

adaptation algorithm is used [5], which consists in dividing the physical time frame considered into sub-intervals and generating an adapted mesh for each one of them.

The unsteady mesh adaptation scheme is derived from the classical steady mesh adaptation algorithm. It consists of two steps: the main (classical) adaptation loop, and an internal loop in which a transient fixed point problem is solved (see Figure A.5). Let us consider the simulation of bubble motion from time $t = 0$ to $t = T$. First, the time period $[0, T]$ is divided in N sub-intervals $[t, t + \Delta t]$. At each iteration of the main adaptation loop is considered a time period $[t, t + \Delta t]$ in which the solution evolves. For instance during the i-th ($i \in [1, N]$) main iteration, a mesh $\mathcal{H}_i$ is generated that is suitable for times $t \in [(i-1)\Delta t, i\Delta t]$. This sub-interval mesh is generated via the internal loop: at each internal iteration j, a metric $\mathcal{M}_{(i,j)}$ is computed that takes into account the solution progression in the sub-interval and a mesh $\mathcal{H}_{(i,j+1)}$ is generated according to $\mathcal{M}_{(i,j)}$. The final solution $\mathcal{S}_{(i,j+1)}$ of the period (i.e. at time $t = i\Delta t$) is computed and compared to the solution of the previous internal iteration $\mathcal{S}_{(i,j)}$ in order to assess the convergence of the internal loop. Let ϵ be a given parameter, the internal transient fixed point algorithm is iterated until:

$$\frac{\left\|\mathcal{S}_{(i,j+1)} - \mathcal{S}_{(i,j)}\right\|_{L^1(\Omega)}}{\left\|\mathcal{S}_{(i,j+1)}\right\|_{L^1(\Omega)}} \leq \epsilon \quad ,$$

where Ω is the computational domain.

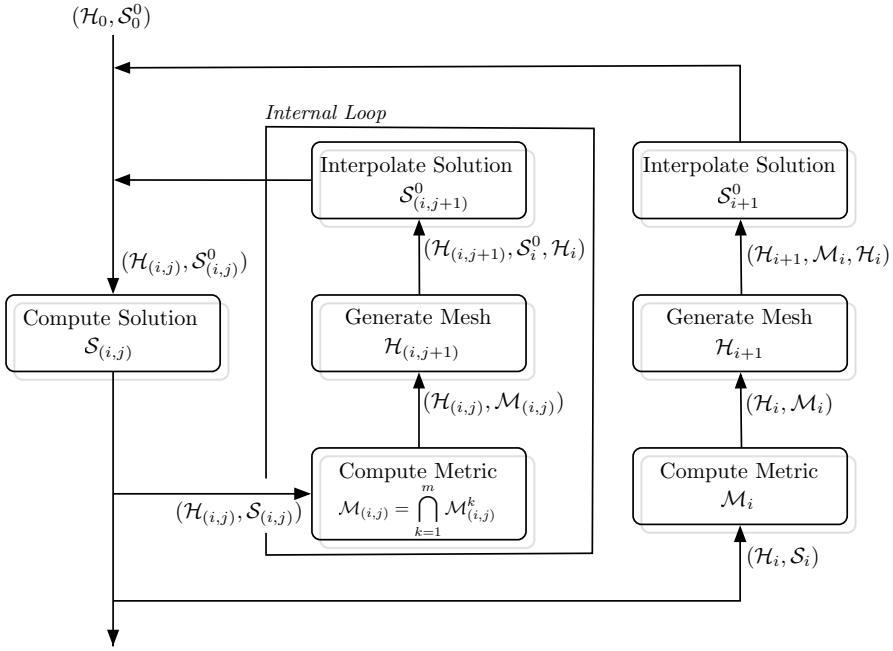


Figure A.5: The mesh adaptation loop

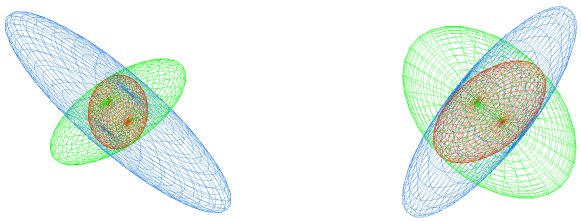


Figure A.6: Two illustrations of the metric intersection procedure. $\mathcal{M}_1$ (in blue) is intersected with $\mathcal{M}_2$ (in blue). The resulting metric $\mathcal{M}_1 \cap \mathcal{M}_2$ is in red.

Computing $\mathcal{M}_{(i,j)}$ At the jth internal iteration of the main iteration i, a metric intersection in time procedure is used to compute $\mathcal{M}_{(i,j)}$, the metric field that takes into account the evolution of the solution in the ith sub-interval $[t, t+\Delta t]$ (i.e. $[(i-1)\Delta t, i\Delta t]$). All the intermediate solutions between $(i-1)\Delta t$ and $i\Delta t$ must be considered to mesh suitably all this region so as to control the error of the solution throughout the time sub-interval. So, $\mathcal{M}_{(i,j)}$ is the intersection of m intermediate metrics:

$$\mathcal{M}_{(i,j)} = \bigcap_{k=1}^{m} \mathcal{M}^k_{(i,j)},$$

where $\cap$ is the metric intersection defined above and $\mathcal{M}^k_{(i,j)}$ is the kth intermediate metric of the sub-interval $[(i-1)\Delta t, i\Delta t]$. The number of intermediate metrics m is given as an input of the algorithm.

Definition of metric intersection $\cap$ Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two metrics of eigenvalues (λ_i) and (μ_i) resp. $(i = 1, 3)$. Let $\mathcal{P} = (e_1, e_2, e_3)$ be the matrix whose columns are formed by the eigenvectors of $\mathcal{N} = \mathcal{M}_1^{-1}\mathcal{M}_2$. The intersection of two metrics $\mathcal{M}_1$ and $\mathcal{M}_2$ is given by:

$$\mathcal{M}_1 \cap \mathcal{M}_2 = {}^t\, \mathcal{M}_1^{1/2} \overline{\mathcal{M}_{1\cap 2}} \mathcal{M}_1^{1/2}$$

where

$$\overline{\mathcal{M}_{1\cap 2}} = \mathcal{P} \begin{pmatrix} \max(\lambda_1, 1) & 0 & 0 \\ 0 & \max(\lambda_2, 1) & 0 \\ 0 & 0 & \max(\lambda_3, 1) \end{pmatrix} {}^t\mathcal{P}$$

Geometrically speaking, a metric intersection is depicted in Figure A.6.

A.2 Numerical Results

The Kothe-Rider test [145] is performed in order to measure the impact of the meshing strategy in accurately predicting bubble motion. In 3D, an initial sphere is linearly advected in a cubic computational domain according to a periodic velocity field of period T. Due to this periodicity, the bubble moves

forward from $t = 0$ to $t = \frac{T}{4}$, and backward from $t = \frac{T}{4}$ to $t = \frac{T}{2}$. So, it is expected to recover its original position (i.e. the sphere) at each $t \in \frac{T}{2}\mathbb{N}$ (see Figure A.7). Although the Kothe-Rider test case lacks any physical sense, it is representative of the reality and makes it possible to measure the diffusion due to the numerical model by estimating the spatial error at each $t \in \frac{T}{2}\mathbb{N}$ (see Figure A.7).

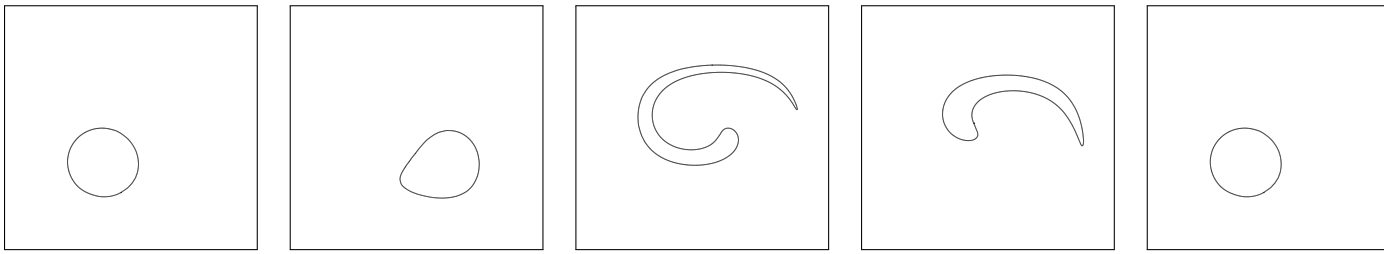


Figure A.7: The 2D bubble's interface at several times from $t = 0$ to $t = \frac{T}{2}$ (from left to right: $t = 0, \frac{T}{16}, \frac{T}{4}, \frac{3T}{8}, \frac{T}{2}$). A spatial error is computed at each $t \in \frac{T}{2}\mathbb{N}$ to evaluate the diffusion due to the numerical model.

A.2.1 Results in 2D

In 2D, the computational domain is $\Omega = [0, 1] \times [0, 1]$ and the velocity field is the following:

$$\begin{cases} u(x, y, t) &= -\sin^2(\pi x)\sin(2\pi y)\cos(2\pi \frac{t}{T}) \\ v(x, y, t) &= \sin(2\pi x)\sin^2(\pi y)\cos(2\pi \frac{t}{T}) \end{cases} \tag{A.4}$$

The initial bubble's radius is $R = 0.15$ and it is centered in $(x_0, y_0) = (0.50, 0.75)$. The velocity field period is $T = 6$ (see Eq. A.4). The bubble was chosen to be advected from time $t = 0$ to $t = 10\frac{T}{2}$ and a spatial error is computed at each $t \in \frac{T}{2}\mathbb{N}$:

$$\epsilon = \sum_i |C_i| \, |\rho_{i,\texttt{exact}} - \rho_{i,h}|$$

where $|C_i|$ is the area of the finite volume cell associated to vertex P_i, $\rho_{i,\texttt{exact}}$ is the exact solution at vertex P_i and $\rho_{i,h}$ is the computed solution.

A total of 9 simulations were run: 5 using uniform meshes and 4 unsteady mesh adaptations. These 9 simulations are summarized in Table A.1. The final error at time $t = 10\frac{T}{2}$ was computed for each simulation, see Figure A.11. It shows that a 2nd order mesh convergence is achieved for adapted meshes (1st order for uniform meshes). The total CPU time of each simulation presented in Figure A.12 shows that the final spatial error observed in 6 hours using a uniform mesh of 1 Million vertices can be achieved in 20 minutes using mesh adaptation. Moreover, it would take 12 days and 22 hours for an uniform mesh

to reach the final spatial error observed after the mesh adaptation which took 2 hours and 30 minutes.

All the 2D simulations were run on a two 2.93 GHz Quad-Core Intel Xeon chips with 24Gb of RAM. The time frame $[0, 10\frac{T}{2}]$ was divided into 20 sub-intervals. At each main iteration in the unsteady adaptation loop, the 20 mesh generations were performed in parallel (8 cores, one process per core at the time).

Adapted meshes alongside with the corresponding solutions are depicted in Figure A.9 and close-up views of the meshes in Figure A.10. Mappings of the density for several meshes and at several physical times are depicted in Figure A.13. A comparison of the final bubble's interface at time $t = 10\frac{T}{2}$ is given in Figure A.8.

# ver	type	#procs	Total CPU	Spatial L^1 error at $t = 10\frac{T}{2}$
10k	uniform	4	57s	8.50e-02
50k	uniform	8	6m5s	4.74e-02
100k	uniform	8	13m58s	3.68e-02
500k	uniform	8	2h11m	1.99e-02
1M	uniform	8	6h0m	1.50e-02
24k	adapted	8	6m39s	2.73e-02
37k	adapted	8	16m35s	1.57e-02
50k	adapted	8	50m42s	1.05e-02
92k	adapted	8	2h30m	4.89e-03

Table A.1: Summary of the 9 simulations for the Kothe-Rider test in 2D. For the mesh adaptations, the given number of vertices corresponds to the mesh for time $t = 10\frac{T}{2}$.

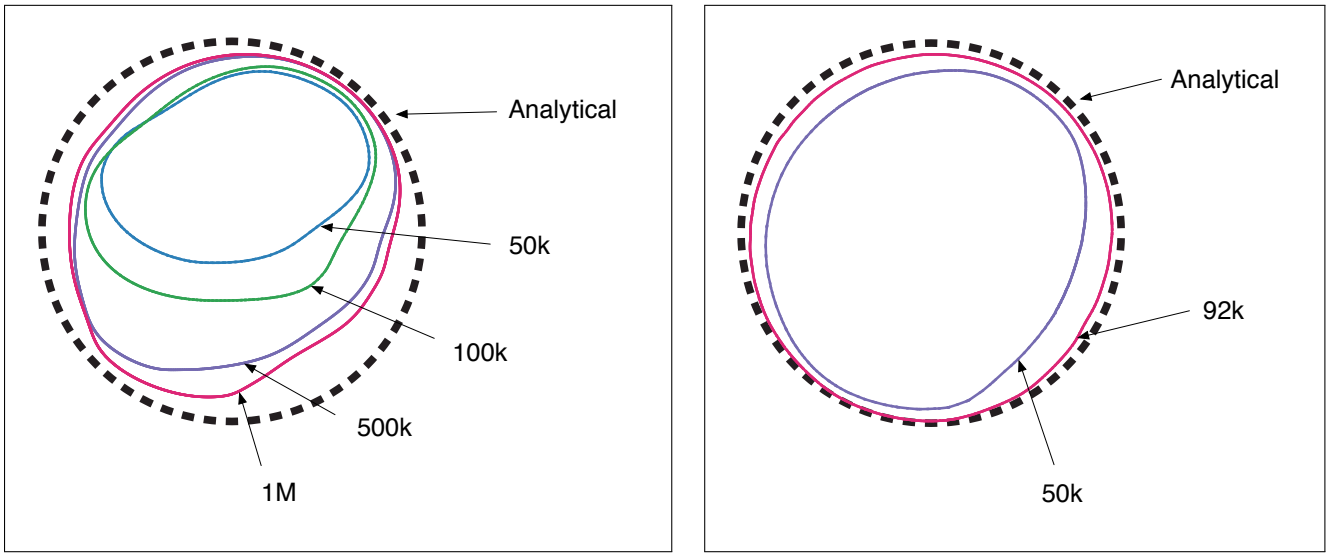


Figure A.8: Comparison of the bubble's interface at $t = 10\frac{T}{2}$. Left: uniform meshes. Right: unsteady mesh adaptation.

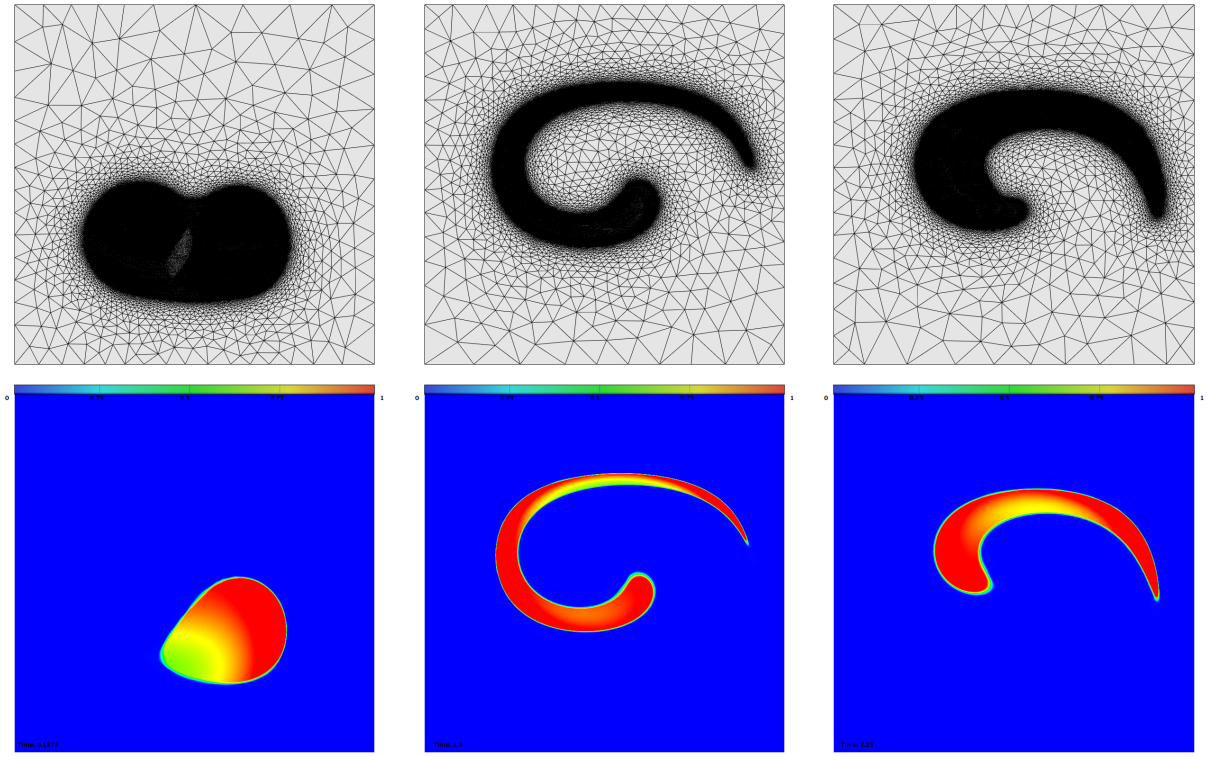


Figure A.9: Adapted meshes and the corresponding densities at times $t = \frac{T}{16}, \frac{T}{4}$ and $\frac{3T}{8}$.

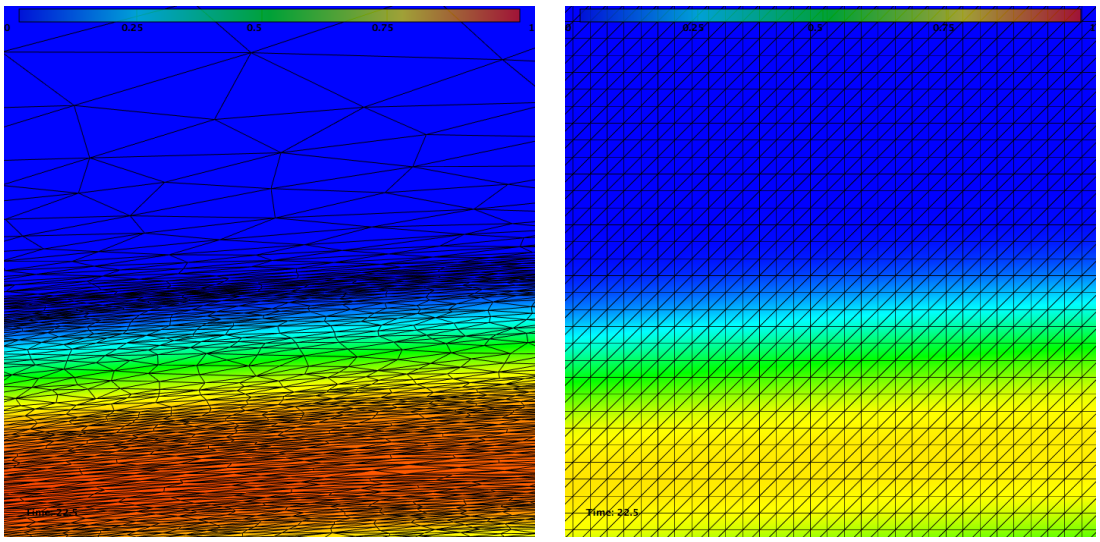


Figure A.10: Close-up views of meshes around the bubble's interface at time $t = 7\frac{T}{2} + \frac{T}{4}$. Left: adapted anisotropic mesh, 50k vertices. Right: uniform mesh, 1M vertices.

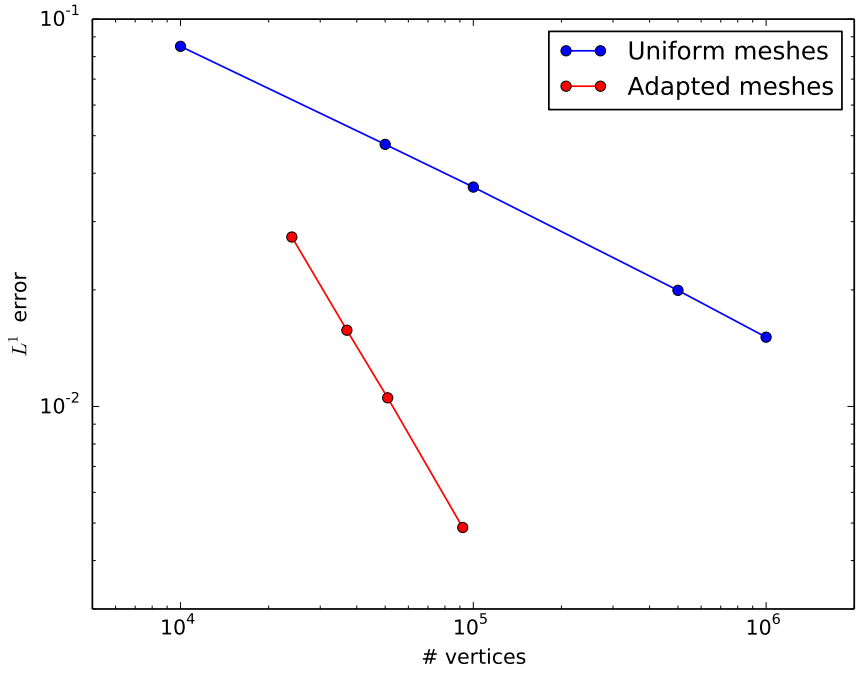


Figure A.11: Comparison of the spatial L^1 error in 2D at time $t = 10\frac{T}{2}$ vs number of vertices.

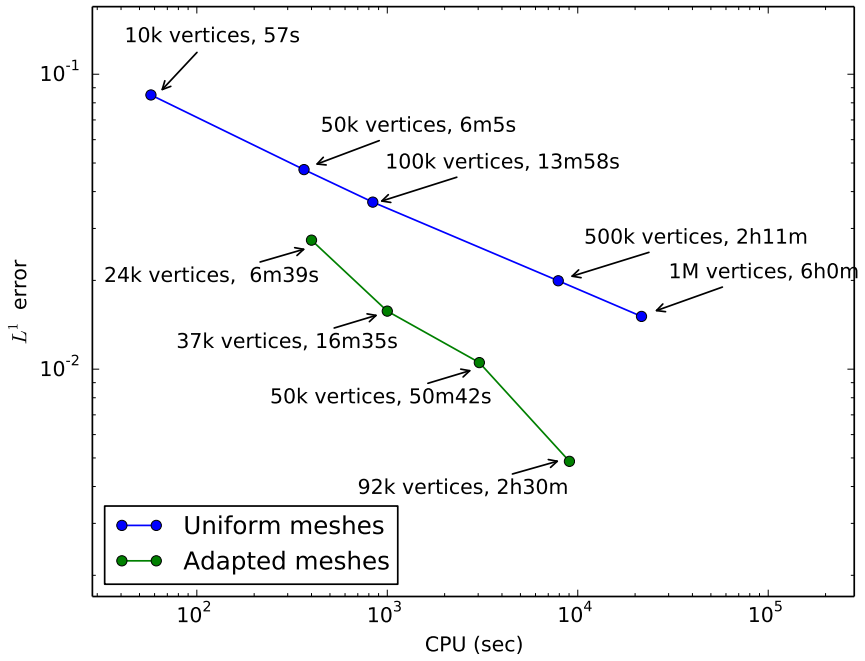
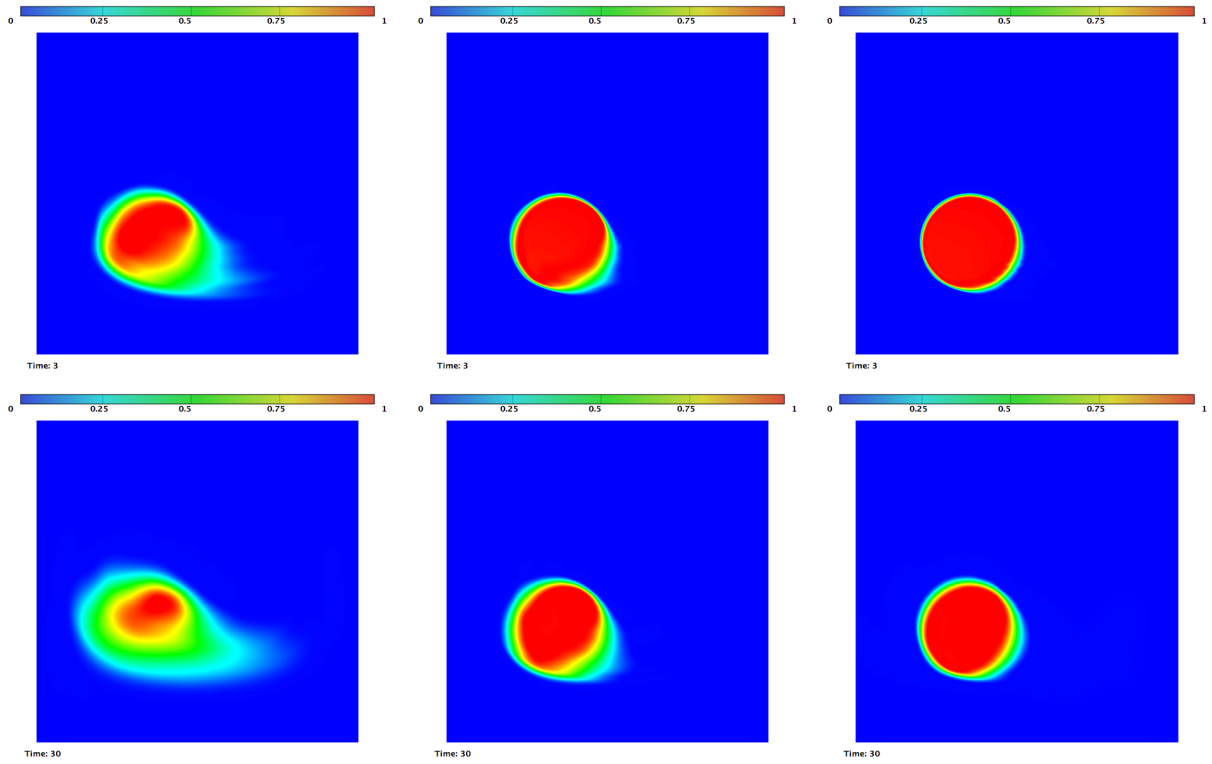


Figure A.12: Spatial L^1 error in 2D at time $t = 10\frac{T}{2}$ vs the total CPU time of each simulation.



$50k$ vertices (uniform) $1000k$ vertices (uniform) $50k$ vertices (adapted)

Figure A.13: Comparison of the mapping of the density at times $t = \frac{T}{2}$ (top row) and $t = 10\frac{T}{2}$ (bottow row). Three computations are compared: two uniform meshes of 50k vertices and 1M vertices (1st and 2nd column resp.) and adapted meshes ($\sim 50k$ vertices each, 3rd column).

A.2.2 Results in 3D

In 3D, the bubble is advected in a cubic computational domain $\Omega = [0,1] \times [0,1] \times [0,1]$. The advection is governed by the following time-periodic velocity field:

$$\begin{cases} u(x,y,z,t) &= 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)\cos(2\pi\frac{t}{T}) \\ v(x,y,z,t) &= -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z)\cos(2\pi\frac{t}{T}) \\ w(x,y,z,t) &= -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)\cos(2\pi\frac{t}{T}) \end{cases} \tag{A.5}$$

The initial bubble's radius is $R = 0.15$ and it is centered in $(x_0, y_0, z_0) = (0.35, 0.35, 0.35)$. The velocity field's period is $T = 6$. The bubble was chosen to be advected from time $t = 0$ to $t = 2\frac{T}{2}$ and a spatial error is computed at $t = \frac{T}{2}$ and $t = T$.

Four simulations were run using uniform meshes containing from 125k to 32M vertices. One mesh adaptation was run with an increasing mesh complexity at each iteration in the main loop (see Section A.1.3): from ∼50k vertices for the first iteration to ∼350k vertices for the last one. A summary of

the simulations in 3D is presented in Table A.2. Note that the CPU timing given for i-th iteration of the main loop includes the timings of the iterations from the first to the ith.

The 3D simulations were run on four 2.00GHz ten-core Intel Xeon chips with 3Tb of RAM. The time frame $[0, T]$ was divided into 64 sub-intervals and mesh generations were performed on 32 cores (one process per core at the time).

A spatial L^1 error ϵ was computed at time $t = T$:

$$\epsilon = \sum_i |C_i| \, |\rho_{i,\texttt{exact}} - \rho_{i,h}|$$

where $|C_i|$ is the volume of the finite volume cell associated to vertex P_i, $\rho_{i,\texttt{exact}}$ is the exact solution at vertex P_i and $\rho_{i,h}$ is the computed solution. The mesh convergence is presented in Figure A.16, and the CPU timings in Figure A.17. One would need a uniform mesh of approximately 10^{12} vertices to reach the final spatial error obtained using unsteady mesh adaptation (349k vertices), which would require years of computation using this numerical model.

The bubble's interface (iso-value $\rho = 0.95$) is depicted for several physical times in Figure A.14 for the uniform case (32 Million vertices), and in Figure A.15 for the adapted case (349k vertices). The interface's conservation is significantly improved using mesh adaptation. Several views of the adapted meshes are given in Figure A.18. Views of the uniform mesh containing 4 Million vertices are depicted in Figure A.19.

# ver	type	#procs	Total CPU	L^1 error ($t = T$)
125k	uniform	8	87s	2.43e-02
500k	uniform	16	7m55s	2.11e-02
4M	uniform	32	1h23m	1.49e-02
32M	uniform	32	13h0m	9.92e-03
51k	adapted (1st ite)	32	36m	1.60e-02
93k	adapted (2nd ite)	32	2h15m	7.86e-03
187k	adapted (3rd ite)	32	7h8m	4.74e-03
234k	adapted (4th ite)	32	15h50m	3.17e-03
292k	adapted (5th ite)	32	1d3h13m	2.41e-03
349k	adapted (6th ite)	32	1d6h48m	2.01e-03

Table A.2: Summary of the simulations for the Kothe-Rider test in 3D. For the mesh adaptations, the given number of vertices corresponds to the mesh for the time $t = T$.

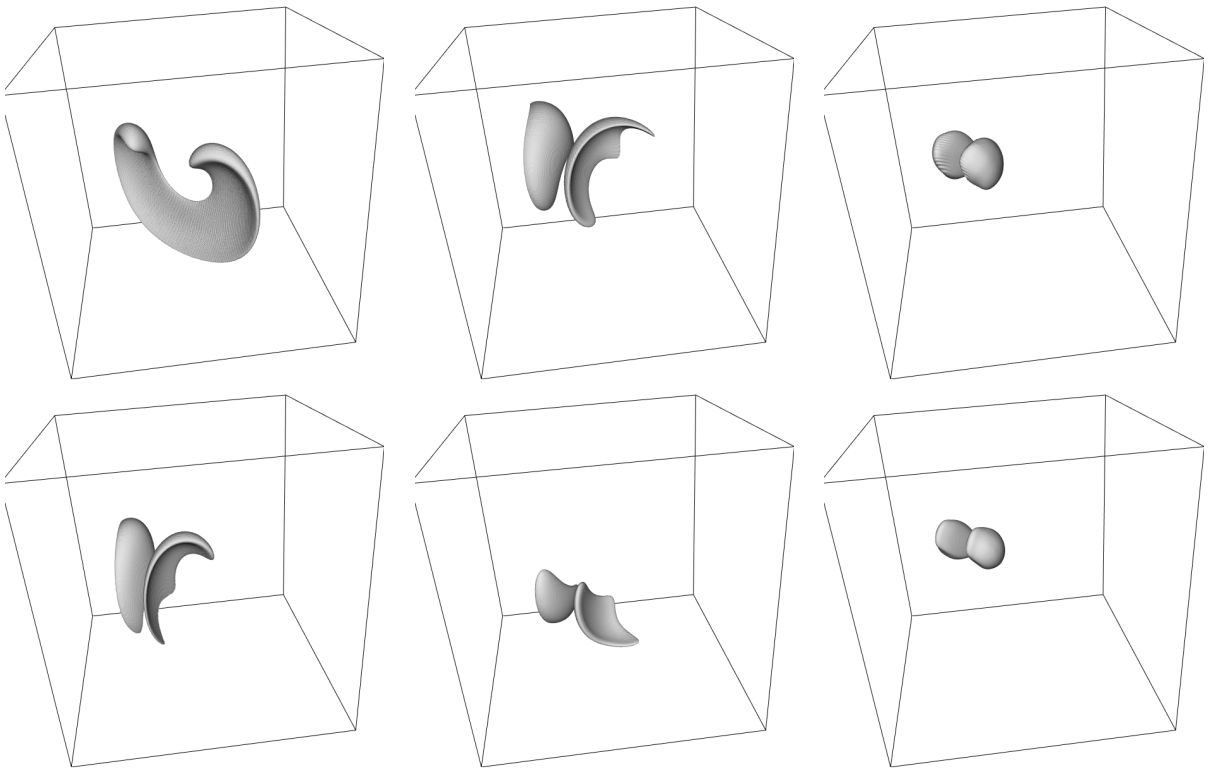


Figure A.14: Result of the computation on the uniform mesh containing 32 Million vertices. The bubble's interface (iso-surface $\rho = 0.95$) at times $t = \frac{T}{8}, \frac{T}{4}, \frac{T}{2}, \frac{3T}{4}, \frac{7T}{8}$ and T.

A.3 Conclusion

Anisotropic mesh adaptation was compared to uniform meshes for the high-fidelity prediction of bubble motion. Mesh convergence is dramatically improved for the Kothe-Rider test case using mesh adaptation. In 2D, the final spatial error observed in 6 hours with an uniform mesh of 1M vertices can be achieved in 20 minutes using mesh adaptation. Moreover, it would take 12 days and 22 hours for uniform meshes to reach the final spatial error observed after the mesh adaptation which took 2 hours and 30 minutes. In 3D, a uniform mesh of 10^{12} vertices would give the same accuracy as obtained in 1 day and 6 hours using an adapted mesh of 349k vertices, which would then take years of computation using the same numerical model.

The results obtained with mesh adaptation could be improved in several ways, including (i) using a level-set method, (ii) generating metric-aligned adapted meshes (see Chapter 7) in order to better handle strong anisotropy in the interface, and (iii) optimizing the number of sub-intervals during the adaptation as well as the prescribed number of vertices, which can dramatically impact the total CPU time of the simulation.

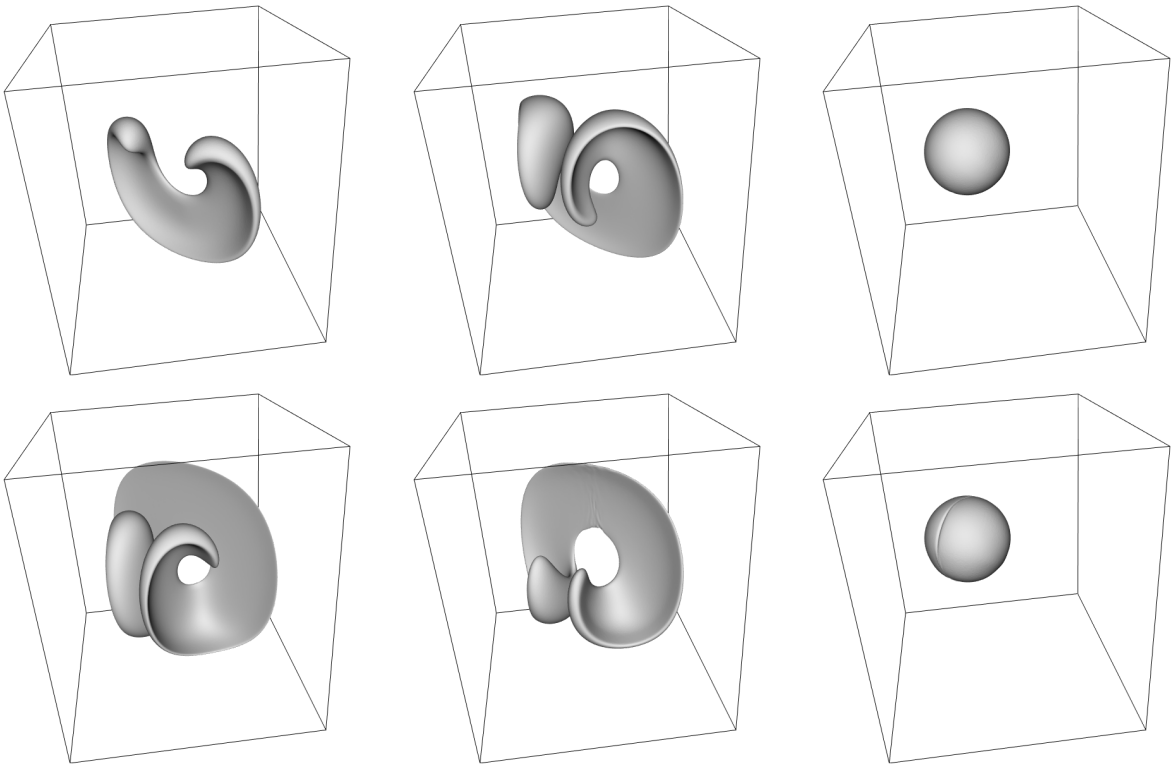


Figure A.15: Result of the final 3D mesh adaptation. The bubble's interface (iso-surface $\rho = 0.95$) at times $t = \frac{T}{8}, \frac{T}{4}, \frac{T}{2}, \frac{3T}{4}, \frac{7T}{8}$ and T.

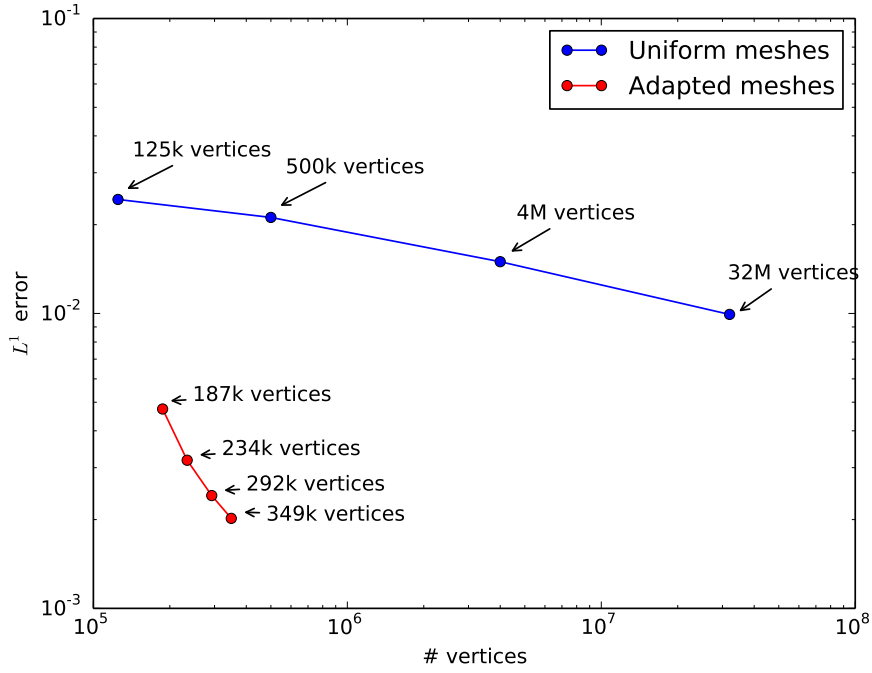


Figure A.16: Comparison of the spatial L^1 error in 3D at time $t = T$ vs number of vertices.

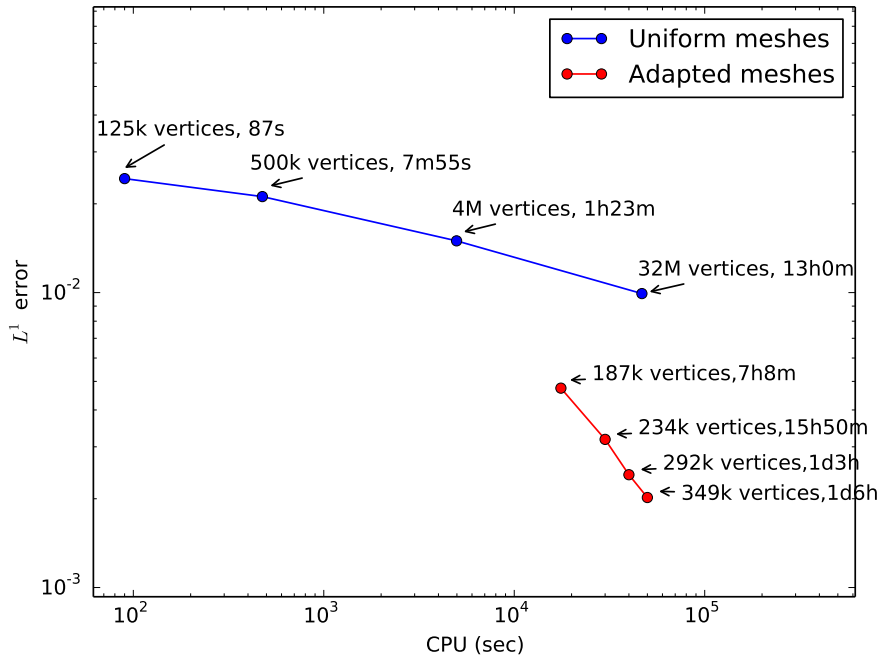


Figure A.17: Spatial L^1 error in 3D at time $t = T$ vs the total CPU time of each simulation.

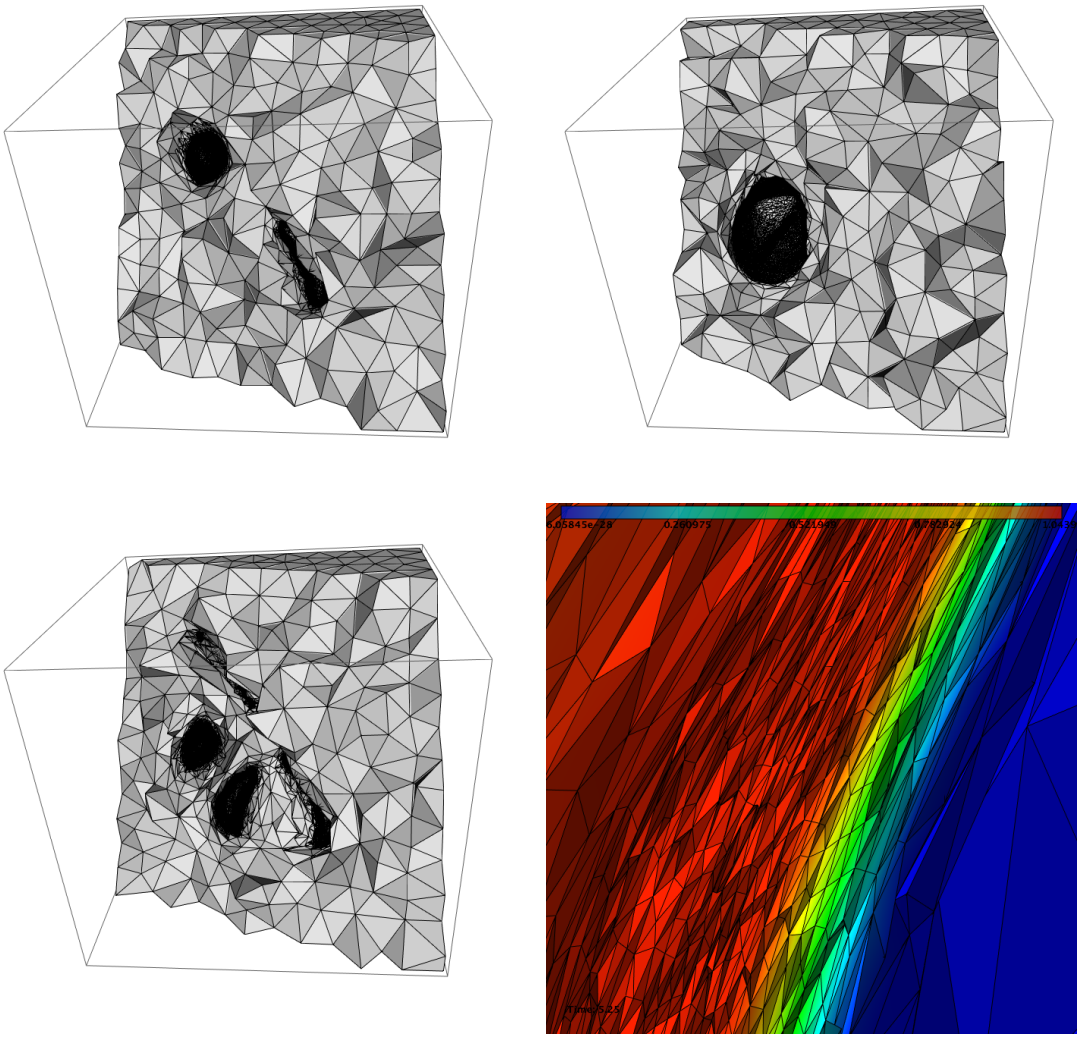


Figure A.18: Adapted meshes for the 3d bubble motion. Top: cuts in the volume at times $t = \frac{T}{8}$ and $\frac{T}{2}$. Bottom: cuts in the volume at time $t = \frac{3T}{4}$ (right: close-up view of the bubble's interface).

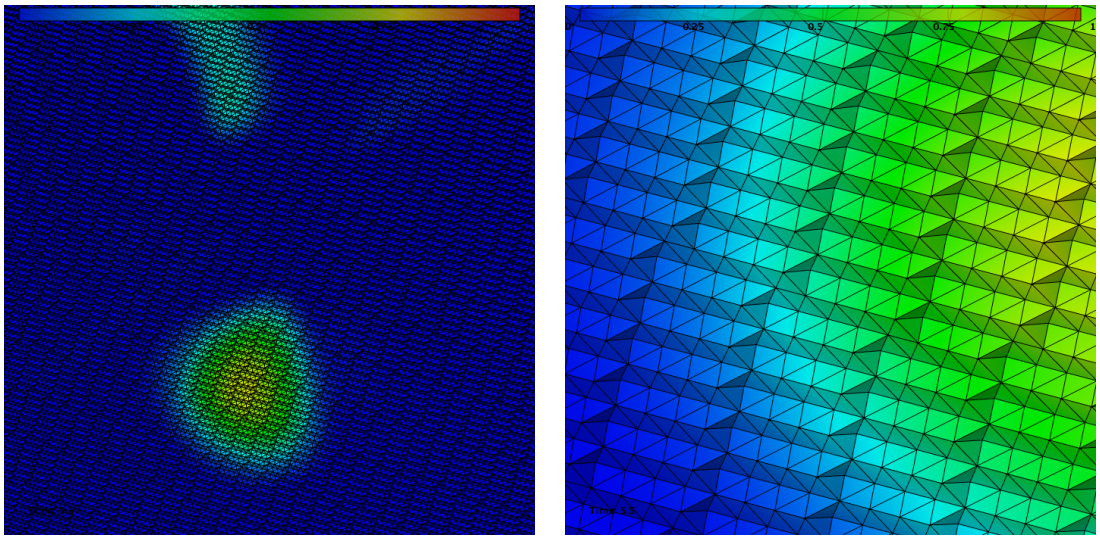


Figure A.19: Cut in the volume of the 4M vertices uniform mesh with its solution at time $t = 0.91T$ (two close-up views).

Appendix B

Computation of the Distance Function

The Spalart-Allmaras turbulence model requires the distance to a viscous wall in order to estimate the turbulent viscosity. A brute force computation of the shortest distance from any volume vertex to the boundary would obviously be too costly in terms of CPU, as it would lead to a complexity of the order of $\mathcal{O}(N_v \times N_b)$, where N_v is the number of points in the volume and N_b the number of points on the boundary. We chose to construct the distance function using (i) point-surrounding points information, (ii) a heap list for the points, and (iii) faces-surrounding points information. This algorithm is described in [92] and consists in two parts: surface initialization and volume treatment. An example of distance function computed around a 3D wing/body/nacelle configuration is shown in Figure B.1.

Surface initialization. This part consists in looping over the boundary faces from which the distance is to be computed. All their vertices are marked and added (once for each) in the heap list. The key used is the vertex' distance to the wall (i.e. 0).

```
Tag all points as unmarked ;
for all boundary faces do
    for all points of the face do
        if The point is not marked then
            Mark the point using the face number;
            Introduce the point in the heap list using its distance to the wall (i.e. 0) as key.
        end
    end
end
```

Algorithm 3: Surface initialization.

Volume treatment. We now compute the distance to wall for each volume vertex. The general idea is to loop over the aforementioned heap list until it's empty.

```
while the heap list is not empty do
    Find the point ipmin with the smallest distance to wall (i.e. the one on top of the heap list) ;
    for all points ip surrounding ipmin do
        if the wall distance of ip is unknown then
            Starting from the face used to mark ipmin, find the shortest distance to the wall ;
            Mark ip with the current face and update its shortest distance ;
            for all points jp surrounding ip do
                if the wall distance of jp is unknown then
                    Introduce jp into the heap list using the distance of ip as key ;
                    Mark jp
                end
            end
        end
    end
end
```

Algorithm 4: Volume treatment.

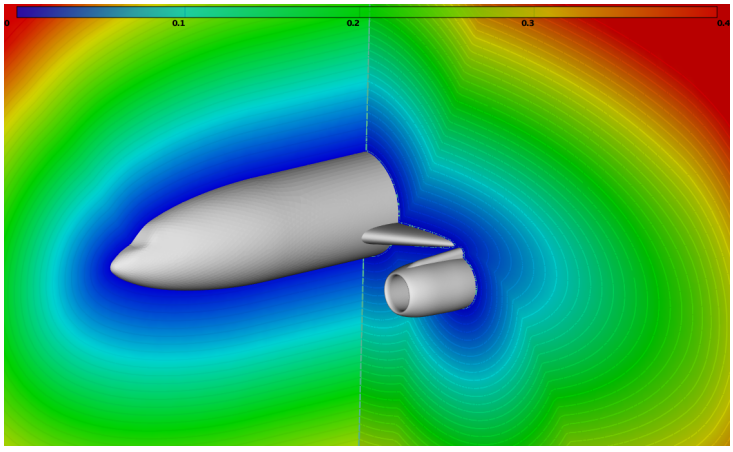


Figure B.1: Mapping of the distance function around a wing/body/nacelle configuration (mesh from the 2nd drag prediction workshop).

Appendix C

Full Linearization of the Source Terms

We detail the full linearization of the source term introduced in Section 3.3.4. We recall that the source terms are the sum of production ($\mathcal{P}$), destruction ($\mathcal{D}$) and diffusion terms ($\mathcal{V}$), which only contribute to the diagonal:

$$\frac{\partial \mathbf{Q}_i^n}{\partial \tilde{\nu}_i} = \frac{\partial \mathcal{P}_i^n}{\partial \tilde{\nu}_i} + \frac{\partial \mathcal{D}_i^n}{\partial \tilde{\nu}_i} + \frac{\partial \mathcal{V}_i^n}{\partial \tilde{\nu}_i} \,.$$

Production term. The full linearization of the production term $\mathcal{P}$ reads:

$$\frac{\partial \mathcal{P}_i^n}{\partial \tilde{\nu}_i} = c_{b_1} \rho_i \left(\tilde{S} + \tilde{\nu}_i \frac{\partial \tilde{S}}{\partial \tilde{\nu}_i} \right)$$

Seeing that $\dfrac{\partial \chi}{\partial \tilde{\nu}_i} = \dfrac{1}{\nu_i}$, we have:

$$\begin{aligned}
\frac{\partial \tilde{S}}{\partial \tilde{\nu}_i} &= \frac{f_{v2}}{\kappa^2 d^2} + \frac{\tilde{\nu}_i}{\kappa^2 d^2} \frac{\partial f_{v2}}{\partial \tilde{\nu}_i} \\
\frac{\partial f_{v2}}{\partial \tilde{\nu}_i} &= \frac{-\dfrac{\partial \chi}{\partial \tilde{\nu}_i}(1+\chi f_{v1}) + \chi \dfrac{\partial \chi f_{v1}}{\partial \tilde{\nu}_i}}{(1+\chi f_{v1})^2} = \frac{-\dfrac{\partial \chi}{\partial \tilde{\nu}_i} + \chi^2 \dfrac{\partial f_{v1}}{\partial \tilde{\nu}_i}}{(1+\chi f_{v1})^2} = \frac{-\dfrac{1}{\nu} + \chi^2 \dfrac{\partial f_{v1}}{\partial \tilde{\nu}_i}}{(1+\chi f_{v1})^2} \\
\frac{\partial f_{v1}}{\partial \tilde{\nu}_i} &= \frac{3\chi^2 \dfrac{\partial \chi}{\partial \tilde{\nu}_i}(\chi^3 + c_{v1}^3) - \chi^3 \dfrac{\partial (\chi^3 + c_{v1}^3)}{\partial \tilde{\nu}_i}}{(\chi^3 + c_{v1}^3)^2} = \frac{3\chi^2 c_{v1}^3 \dfrac{\partial \chi}{\partial \tilde{\nu}_i}}{(\chi^3 + c_{v1}^3)^2} = \frac{3\chi^2 c_{v1}^3}{\nu_i \, (\chi^3 + c_{v1}^3)^2} \,.
\end{aligned}$$

We also give the differentiation with respect to $\rho\tilde{\nu}$:

$$\frac{\partial \mathcal{P}_i^n}{\partial \rho \tilde{\nu}_i} = c_{b_1} \left(\tilde{S} + \rho \tilde{\nu}_i \frac{\partial \tilde{S}}{\partial \rho \tilde{\nu}_i} \right)$$

where (knowing that $\frac{\partial \chi}{\partial \rho\tilde{\nu}_i} = \frac{1}{\rho\nu_i}$):

$$\frac{\partial \tilde{S}}{\partial \rho\tilde{\nu}_i} = \frac{f_{v2}}{\rho_i\kappa^2 d^2} + \frac{\tilde{\nu}_i}{\kappa^2 d^2}\frac{\partial f_{v2}}{\partial \rho\tilde{\nu}_i}\,, \qquad \frac{\partial f_{v2}}{\partial \rho\tilde{\nu}_i} = \frac{-\frac{1}{\rho\nu} + \chi^2\frac{\partial f_{v1}}{\partial \rho\tilde{\nu}_i}}{(1+\chi f_{v1})^2}\,, \qquad \frac{\partial f_{v1}}{\partial \rho\tilde{\nu}_i} = \frac{3\chi^2 c_{v1}^3}{\rho\nu_i\,(\chi^3 + c_{v1}^3)^2}\,.$$

Destruction term. As regards the destruction term $\mathcal{D}$, we have to differentiate f_w:

$$\frac{\partial \mathcal{D}_i^n}{\partial \tilde{\nu}_i} = \frac{c_{w_1}\rho_i}{d_i^2}\left(2\,f_w\tilde{\nu}_i + \tilde{\nu}_i^2\frac{\partial f_w}{\partial \tilde{\nu}_i}\right) = \frac{c_{w_1}\rho_i\tilde{\nu}_i}{d_i^2}\left(2\,f_w + \tilde{\nu}_i\frac{\partial f_w}{\partial \tilde{\nu}_i}\right)$$

We denote $G_{lim} = \dfrac{1+c_{w3}^6}{g^6+c_{w3}^6}$ and we have:

$$\begin{aligned}
\frac{\partial G_{lim}}{\partial \tilde{\nu}_i} &= \frac{-6g^5\frac{\partial g}{\partial \tilde{\nu}_i}(1+c_{w3}^6)}{(g^6+c_{w3}^6)^2} \\
\frac{\partial f_w}{\partial \tilde{\nu}_i} &= G_{lim}^{\frac{1}{6}}\frac{\partial g}{\partial \tilde{\nu}_i} + g\frac{1}{6}\frac{\partial G_{lim}}{\partial \tilde{\nu}_i}G_{lim}^{\frac{1}{6}}G_{lim}^{-1} \;=\; G_{lim}^{\frac{1}{6}}\frac{\partial g}{\partial \tilde{\nu}_i}\left(1 - \frac{g^6}{g^6+c_{w3}^6}\right) \\
\frac{\partial g}{\partial \tilde{\nu}_i} &= \left(1 + c_{w2}\left(6r^5 - 1\right)\right)\frac{\partial r}{\partial \tilde{\nu}_i} \\
\frac{\partial r}{\partial \tilde{\nu}_i} &= \frac{1}{\kappa^2 d^2}\,\frac{\tilde{S} - \tilde{\nu}_i\frac{\partial \tilde{S}}{\partial \tilde{\nu}_i}}{\tilde{S}^2} \;=\; \frac{\tilde{S} - \tilde{\nu}_i\frac{\partial \tilde{S}}{\partial \tilde{\nu}_i}}{\tilde{S}^2\kappa^2 d^2}\,.
\end{aligned}$$

We also give the differentiation with respect to $\rho\tilde{\nu}$:

$$\frac{\partial \mathcal{D}_i^n}{\partial \rho\tilde{\nu}_i} = \frac{c_{w_1}}{d_i^2}\left(2\,f_w\tilde{\nu}_i + \rho_i\tilde{\nu}_i^2\frac{\partial f_w}{\partial \rho\tilde{\nu}_i}\right) = \frac{c_{w_1}\tilde{\nu}_i}{d_i^2}\left(2\,f_w + \rho\tilde{\nu}_i\frac{\partial f_w}{\partial \rho\tilde{\nu}_i}\right)$$

with

$$\begin{aligned}
\frac{\partial G_{lim}}{\partial \rho\tilde{\nu}_i} &= \frac{-6g^5\frac{\partial g}{\partial \rho\tilde{\nu}_i}(1+c_{w3}^6)}{(g^6+c_{w3}^6)^2} & \frac{\partial f_w}{\partial \rho\tilde{\nu}_i} &= G_{lim}^{\frac{1}{6}}\frac{\partial g}{\partial \rho\tilde{\nu}_i}\left(1 - \frac{g^6}{g^6+c_{w3}^6}\right) \\
\frac{\partial g}{\partial \rho\tilde{\nu}_i} &= \left(1 + c_{w2}\left(6r^5 - 1\right)\right)\frac{\partial r}{\partial \rho\tilde{\nu}_i} & \frac{\partial r}{\partial \rho\tilde{\nu}_i} &= \frac{\tilde{S} - \rho\tilde{\nu}_i\frac{\partial \tilde{S}}{\partial \rho\tilde{\nu}_i}}{\rho_i\tilde{S}^2\kappa^2 d^2}\,.
\end{aligned}$$

Diffusion term. Here, the differentiation reads:

$$\frac{\partial \mathcal{V}_i^n}{\partial \tilde{\nu}_i} = \frac{c_{b2}\rho_i}{\sigma}\frac{\partial\left(||\nabla\tilde{\nu}_i||^2\right)}{\partial \tilde{\nu}_i} = \frac{2c_{b2}\rho_i}{\sigma}\frac{\partial \nabla\tilde{\nu}_i}{\partial \tilde{\nu}_i}\cdot\nabla\tilde{\nu}_i$$

Assuming the derivatives commute, we have: $\dfrac{\partial \nabla\tilde{\nu}_i}{\partial \tilde{\nu}_i} = \nabla\dfrac{\partial \tilde{\nu}_i}{\partial \tilde{\nu}_i} = \nabla 1 = 0$, thus there is no contribution from the diffusion term:

$$\frac{\partial \mathcal{V}_i^n}{\partial \tilde{\nu}_i} = 0\,.$$

Bibliography

[1] *Aeronautical Facilities: Assessing the National Plan for Aeronautical Ground Test Facilities*. The National Academies Press, Washington, DC, 1994.

[2] The ubercloud HPC experiment: Compendium of case studies, 2013.

[3] *IATA Annual Report*. Number 70. International Air Transport Association, 2014.

[4] F. Alauzet. `Wolf` user guide. Internal report, Inria, 2015.

[5] F. Alauzet, P .J. Frey, P.-L. George, and B. Mohammadi. 3D transient fixed point mesh adaptation for time-dependent problems: Application to CFD simulations. *J. Comp. Phys.*, 222:592–623, 2007.

[6] F. Alauzet and A. Loseille. On the use of space filling curves for parallel anisotropic mesh adaptation. In *18th International Meshing Roundtable*, 2009.

[7] F. Alauzet and A. Loseille. High order sonic boom modeling by adaptive methods. *J. Comp. Phys.*, 229:561–593, 2010.

[8] A. Alleaume, L. Francez, M. Loriot, and N. Maman. Automatic tetrahedral out-of-core meshing. In *16th International Meshing Roundtable*, 2008.

[9] S.R. Allmaras, F.T. Johnson, and P.R. Spalart. Modifications and clarifications for the implementation of the Spalart-Allamas turbulence model. In *7th International Conference on Computational Fluid Dynamics*, Big Island, HI, USA, Jul 2012.

[10] T. J. Baker. Mesh adaptation strategies for problems in fluid dynamics. *Finite Elements in Analysis and Design*, 25(3–4):243 – 273, 1997.

[11] R.E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations: Users' Guide 8.0*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics, 1998.

[12] R.E. Bank and R.K. Smith. A posteriori error estimate based on hierarchical bases. *SIAM J. Numer. Anal.*, 30:921–935, 1993.

[13] T. J. Barth. Aspects of unstructured grids and finite-volume solvers for the euler and navier-stokes equations. VKI Lecture Series 1994-05, 1994.

[14] T.J. Barth. A 3D least-squares upwind euler solver for unstructured meshes. In *13th International Conference on Numerical Methods in Fluid Dynamics*, volume 414 of *Lecture Notes in Physics*, pages 240–244. 1993.

[15] P. Batten, M.A. Leschziner, and U.C. Goldberge. Average-state Jacobians and implicit methods for compressible viscous and turbulent flows. *J. Comp. Phys.*, 137:38–78, 1997.

[16] Y. Belhamadia, A. Fortin, and E. Chamberland. Three-dimensional anisotropic mesh adaptation for phase change problems. *J. Comp. Phys.*, 201(2):753 – 770, 2004.

[17] R. T. Biedron, J. M. Derlaga, P. A. Gnoffo, D. P. Hammond, W. T. Jones, B. Kleb, E. M. Lee-Rausch, E. J. Nielsen, M. A. Park, C. L. Rumsey, J. L. Thomas, and W. A. Wood. FUN3D manual: 12.4. NASA TM-2014-218179, Langley Research Center, March 2014.

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

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

[20] Y. Bourgault, M. Picasso, F. Alauzet, and A. Loseille. On the use of anisotropic error estimators for the adaptative solution of 3D inviscid compressible flows. *Int. J. Numer. Meth. Fluids*, 59:47–74, 2009.

[21] A. Bowyer. Computing dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.

[22] M. E. Braaten and S. D. Connell. Three-dimensional unstructured adaptive multigrid scheme for the Navier-Stokes equations. *AIAA Journal*, (2):281–290, 1996.

[23] A. Brandt. Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems. In Henri Cabannes and Roger Temam, editors, *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, volume 18 of *Lecture Notes in Physics*, pages 82–89. Springer Berlin Heidelberg, 1973.

[24] A. Brandt and O. Livne. *Multigrid Techniques*. Society for Industrial and Applied Mathematics, 2011.

[25] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In *Sparsity and Its Applications*. Cambridge University Press, 1983.

[26] G. Brèthes, O. Allain, and A. Dervieux. A mesh-adaptive metric-based Full-Multigrid for the Poisson problem. *International Journal for Numerical Methods in Fluids*, 2015.

[27] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge. Adaptive algebraic multigrid. *SIAM Journal on Scientific Computing*, 27(4):1261–1286, 2006.

[28] C. Bui, C. Dapogny, and P. Frey. An accurate anisotropic adaptation method for solving the level set advection equation. *International Journal for Numerical Methods in Fluids*, 70(7):899–922, 2012.

[29] N.K. Burgess and R.S. Glasby. Advances in numerical methods for CREATE-AV analysis tools. AIAA Paper 2014-0417, 2014.

[30] G. C. Buscaglia and E. A. Dari. Anisotropic mesh optimization and its application in adaptivity. *Int. J. Numer. Meth. Engng*, 40:4119–4136, 1997.

[31] R. H. Bush, G. D. Power, and C. E. Towne. WIND: The production flow solver of the NPARC alliance. AIAA Paper 98–935, 1998.

[32] D. M. Bushnell. SCALING: Wind tunnel to flight. *Annual Review of Fluid Mechanics*, 38(1):111–128, 2006.

[33] M.C. Le Pape M. de la Llave Plata V. Couaillier C. Marmignon, B. Cantaloube and M. Gazaix. Development of an agglomeration multigrid technique in the hybrid solver elsA-H. 2013.

[34] G. Carré and A. Dervieux. On the application of FMG to variational approximation of flow problems. *International Journal of Computational Fluid Dynamics*, 12(2):99–117, 1999.

[35] M. J. Castro-Díaz, F. Hecht, B. Mohammadi, and O. Pironneau. Anisotropic unstructured mesh adaptation for flow simulations. *Int. J. Numer. Meth. Fluids*, 25:475–491, 1997.

[36] S. Catris and B. Aupoix. Density corrections for turbulence models. *Aerospace Science and Technology*, 4:1–11, 2000.

[37] A.N. Chernikov and N.P. Chrisochoides. A template for developing next generation parallel delaunay refinement methods. *Finite Elements in Analysis and Design*, 46(1–2):96 – 113, 2010.

[38] R. Claisse, V. Ducrot, and P. Frey. Level sets and anisotropic mesh adaptation, 2008.

[39] Ph. Clément. Approximation by finite element functions using local regularization. *Revue Française d'Automatique, Informatique et Recherche Opérationnelle*, R-2:77–84, 1975.

[40] G. Compère, E. Marchandise, and J.-F. Remacle. Transient adaptivity applied to two-phase incompressible flows. *J. Comp. Phys.*, 227:1923–1942, 2007.

[41] P.H. Cook, C.P. Firmin, A. McDonald, and Royal Aircraft Establishment. *Aerofoil RAE 2822: Pressure Distributions, and Boundary Layer and Wake Measurements.* Technical memorandum / Royal Aircraft Establishment. RAE, 1977.

[42] T. Coupez. Génération de maillages et adaptation de maillage par optimisation locale. *Revue Européenne des Éléments Finis*, 9:403–423, 2000.

[43] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100(1):32–74, 1928.

[44] P.-H. Cournède, B. Koobus, and A. Dervieux. Positivity statements for a Mixed-Element-Volume scheme on fixed and moving grids. *Europ. J. Comp. Mech.*, 15(7-8):767–798, 2006.

[45] C. Debiez and A. Dervieux. Mixed-Element-Volume MUSCL methods with weak viscosity for steady and unsteady flow calculations. *Comput. & Fluids*, 29:89–118, 2000.

[46] A. Dervieux, D. Leservoisier, P.-L. George, and Y. Coudiere. About theoretical and practical impact of mesh adaptations on approximation of functions and of solution of PDE. *Int. J. Numer. Meth. Fluids*, 43:507–516, 2003.

[47] H. Digonnet, L. Silva, and T. Coupez. Massively parallel computation on anisotropic meshes. In *6th International Conference on Adaptive Modeling and Simulation*, pages 199–211, Lisbon, Portugal, June 2013.

[48] B. Diskin and H. Nishikawa. Evaluation of multigrid solutions for turbulent flows. AIAA Paper 2014-0082, 2014.

[49] C. Dobrzynski and P. J. Frey. Anisotropic delaunay mesh adaptation for unsteady simulations. In *Proc. of 17th Int. Meshing Rountable*, pages 177–194. Springer, 2008.

[50] J. Dompierre, M.-G. Vallet, M. Fortin, Y. Bourgault, and W. G. Habashi. Anisotropic mesh adaptation: towards a solver and user independent cfd. AIAA Paper 1997-0861, 1997.

[51] J. Dongarra and M. A. Heroux. Toward a new metric for ranking high performance computing systems. SANDIA Report 2013-4744, 2013.

[52] L. Eca, M. Hoekstra, A. Hay, and D. Pelletier. Verification of RANS solvers with manufactured solutions. *Engineering with Computers*, 23(4), 2007.

[53] K. J. Fidkowski and P. L. Roe. An entropy adjoint approach to mesh refinement. *SIAM J. Sci. Comput.*, 32(3):1261–1287, 2010.

[54] L. Formaggia, S. Micheletti, and S. Perotto. Anisotropic mesh adaptation in computational fluid dynamics: Application to the advection-diffusion-reaction and the Stokes problems. *Appl. Numer. Math.*, 51:511–533, 2004.

[55] L. Formaggia and S. Perotto. New anisotropic a prioiri error estimate. *Numer. Math.*, 89:641–667, 2001.

[56] M. Fortin, M.-G. Vallet, J. Dompierre, Y. Bourgault, and W. G. Habashi. Anisotropic mesh adaptation: theory, validation and applications. In *Proc. of ECCOMAS CFD*, 1996.

[57] P. Foteinos and N.P. Chrisochoides. Dynamic parallel 3D delaunay triangulation. In *20th International Meshing Roundtable*, 2012.

[58] J. Francescatto and A. Dervieux. A Semi-Coarsening Strategy for Unstructured MG with Agglomeration. *Inria Research* R*eport* , RR-2950, 1996.

[59] P. J. Frey. About surface remeshing. In *9th International Meshing Roundtable*, 2000.

[60] P. J. Frey and F. Alauzet. Anisotropic mesh adaptation for CFD computations. *Comput. Methods Appl. Mech. Engrg.*, 194(48-49):5068–5082, 2005.

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

[62] P.L. George, F. Hecht, and M.G. Vallet. Creation of internal points in Voronoi's type method. control adaptation. *Advances in Engineering Software and Workstations*, 13(5–6):303 – 312, 1991.

[63] S. C. Glotzer, S. Kim, P. T. Cumings, A. Deshmukh, M. Head-Gordon, G. Karniadakis, L. Petzold, C. Sagui, , and M. Shinozuka. *International Assessment of Research and Development In Simulation-Based Engineering and Science.* World Technology Evaluation 44, 2012.

[64] N. Gourvitch, G. Rogé, I. Abalakin, A. Dervieux, and T. Kozubskaya. A tetrahedral-based super convergent scheme for aeroacoustics. Inria Research Report RR-5212, INRIA, 2004.

[65] C. Gruau and T. Coupez. 3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric. *Comput. Methods Appl. Mech. Engrg.*, 194(48-49):4951–4976, 2005.

[66] H. Guillard. Node-nested multi-grid with delaunay coarsening. *Inria Research* R*eport* , RR-1898, 1993.

[67] W. Hackbusch. *Multi-grid methods and applications.* Number 4 in Computational mathematics. Springer-Verlag, New York, 1985.

[68] R. Haimes and J. Dannenhoffer. The engineering sketch pad: A solid-modeling, feature-based, web-enabled system for building parametric geometry. AIAA Paper 2013-3073, 2013.

[69] A. Harten, P. D. Lax, and B. Van Leer. On upstream differencing and godunov-type schemes for hyperbolic conservation laws. In M.Yousuff Hussaini, Bram van Leer, and John Van Rosendale, editors, *Upwind and High-Resolution Schemes*, pages 53–79. Springer Berlin Heidelberg, 1997.

[70] W. Hassan and M. Picasso. An anisotropic adaptive finite element algorithm for transonic viscous flows around a wing. *Comput. & Fluids*, 111(1):33–45, 2015.

[71] F. Hecht and B. Mohammadi. Mesh adaptation by metric control for multi-scale phenomena and turbulence. AIAA Paper 97-0859, 1997.

[72] F. Hermeline. Triangulation automatique d'un polyèdre en dimension n. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 16(3):211–242, 1982.

[73] N. Héron, F. Coulouvrat, F. Dagrau, G. Rogé, and Z. Johan. HISAC midterm overview of sonic boom issues. In *Proceedings of the 19th international congress on acoustics-ICA*, Madrid, Spain, 2007.

[74] C. Hirsch. *Numerical Computation of Internal and External Flows (Second Edition)*. Butterworth-Heinemann, Oxford, second edition edition, 2007.

[75] W. Huang. Metric tensors for anisotropic mesh generation. *J. Comp. Phys.*, 204:633–665, 2005.

[76] W. Huang, L. Kamenski, and X. Li. A new anisotropic mesh adaptation method based upon hierarchical a posteriori error estimates. *J. Comp. Phys.*, 229:2179–2198, 2010.

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

[78] 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. Comp. in Sim.*, 75(5-6):200–209, 2007.

[79] A. Jameson and S. Yoon. Lower-Upper implicit schemes with multiple grids for the Euler equations. *AIAA Journal*, 25(7):929–935, 1987.

[80] W.T. Jones, E.J. Nielsen, and M.A. Park. Validation of 3D adjoint based error estimation and mesh adaptation for sonic boom reduction. AIAA Paper 2006-1150, 2006.

[81] B. Koobus, M.-H. Lallemand, and A. Dervieux. Unstructured volume-agglomeration MG : solution of the Poisson equation. Research Report RR-1946, 1993.

[82] J. Krause and P.L. George. Construction d'un maillage 3D anisotrope localement structuré. Rr-4834, INRIA, 2003. (in French).

[83] S. L. Krist, R. T. Biedron, and C. L. Rumsey. *CFL3D user's manual (version 5.0).* National Aeronautics and Space Administration, Langley Research Center, 1998.

[84] 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.

[85] J.L. Lee. *Into the Wind: A History of the American Wind Tunnel, 1896-1941.* 2001.

[86] B. Van Leer. Towards the ultimate conservative difference scheme i. The quest of monotonicity. *Lecture notes in physics*, 18:163–168, 1973.

[87] T. Leicht and R. Hartmann. Error estimation and anisotropic mesh refinement for 3D laminar aerodynamic flow simulations. *J. Comput. Phys.*, 2010.

[88] X. L. Li, M. S. Shephard, and M. W. Beall. 3D anisotropic mesh adaptation by mesh modification. *Comput. Methods Appl. Mech. Engrg.*, 194(48-49):4915–4950, 2005.

[89] A. Lintermann, S. Schlimpert, J.H. Grimmen, C. Günther, M. Meinke, and W. Schröder. Massively parallel grid generation on {HPC} systems. *Comput. Methods Appl. Mech. Engrg.*, 277(0):131 – 153, 2014.

[90] R. Löhner. Adaptive remeshing for transient problems. *Comput. Methods Appl. Mech. Engrg.*, 75:195–214, 1989.

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

[92] R. Löhner. *Applied CFD techniques.* Wiley, New-York, 2001.

[93] R. Löhner. A 2nd generation parallel advancing front grid generator. In *21st International Meshing Roundtable*, 2013.

[94] R. Löhner, J. Camberos, and M. Merriam. Parallel unstructured grid generation. *Comput. Methods Appl. Mech. Engrg.*, 95(3):343 – 357, 1992.

[95] R. Löhner and P. Parikh. Three-dimensional grid generation by the advancing front method. *Int. J. Numer. Meth. Fluids*, 9:1135–1149, 1988.

[96] A. Loseille. *Anisotropic Multi-Scale and Goal-Oriented Mesh Adaptation. Application to High-Fidelity Sonic Boom Prediction.* PhD thesis, Université Pierre et Marie Curie, Paris VI, 2008.

[97] A. Loseille. Metric-orthogonal anisotropic mesh generation. In *23rd International Meshing Roundtable*, 2014.

[98] A. Loseille and F. Alauzet. Optimal 3D highly anisotropic mesh adaptation based on the continuous mesh framework. In *18th International Meshing Roundtable*, pages 575–594. Springer, 2009.

[99] A. Loseille and F. Alauzet. Continuous mesh framework. Part I: well-posed continuous interpolation error. *SIAM J. Numer. Anal.*, 49(1):38–60, 2011.

[100] A. Loseille and F. Alauzet. Continuous mesh framework. Part II: validations and applications. *SIAM J. Numer. Anal.*, 49(1):61–86, 2011.

[101] A. Loseille, F. Alauzet, A. Dervieux, and P. J. Frey. Achievement of second order mesh convergence for discontinuous flows with adapted unstructured mesh adaptation. AIAA Paper 07-4186, 2007.

[102] A. Loseille, A. Dervieux, and F. Alauzet. A 3D goal-oriented anisotropic mesh adaptation applied to inviscid flows in aeronautics. AIAA Paper 2010-1067, 2010.

[103] A. Loseille, A. Dervieux, and F. Alauzet. Anisotropic norm-oriented mesh adaptation for compressible flows. *AIAA Paper* , 2015-2037, 2015.

[104] A. Loseille and R. Löhner. On 3D anisotropic local remeshing for surface, volume and boundary layers. In *18th International Meshing Roundtable*, pages 611–630. Springer, 2009.

[105] A. Loseille and R. Löhner. Adaptive anisotropic simulations in aerodynamics. AIAA Paper 2010-169, 2010.

[106] A. Loseille and R. Löhner. Boundary layer mesh generation and adaptivity. AIAA Paper 2011-894, 2011.

[107] A. Loseille and R. Löhner. Boundary layer mesh generation and adaptivity. In *49th AIAA Aerospace Sciences Meeting*, AIAA Paper 2011-894, Orlando, FL, USA, Jan 2011.

[108] A. Loseille and V. Menier. Serial and parallel mesh modification through a unique cavity-based primitive. In *22nd International Meshing Roundtable*, 2013.

[109] E. Luke, X.-L. Tong, J. Wu, L. Tang, and P. Cinnella. A step towards shape shifting algorithms: Reacting flow simulations using generalized grids. AIAA Paper 2001-0897, 2001.

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

[111] H. Luo, J.D. Baum, and R. Löhner. An accurate, fast, matrix-free implicit method for computing unsteady flows on unstructured grids. *Comput. & Fluids*, 30:137–159, 2001.

[112] D. Marcum. Adaptive unstructured grid generation for viscous flow applications. *AIAA Journal*, 34(8):2440–2443, 1996.

[113] D. Marcum. Unstructured grid generation using automatic point insertion and local reconnection. In *The Handbook of Grid Generation*, Edited by J.F. Thompson, B. Soni, and N.P. Weatherill, chapter 18, pages 1–31. CRC Press, 1998.

[114] D. Marcum and F. Alauzet. Unstructured mesh generation using advancing layers and metric-based transition. AIAA Paper 2013-2710, 2013.

[115] D. Marcum and F. Alauzet. Aligned metric-based anisotropic solution adaptive mesh generation. In *23rd International Meshing Roundtable*, 2014.

[116] L. Maréchal. The LP3 library: A parallelization framework for numerical simulation. Technical report, INRIA, 2010.

[117] D. J. Mavriplis. Adaptive mesh generation for viscous flows using delaunay triangulation. *J. Comput. Phys.*, 90(2):271–291, September 1990.

[118] D. J. Mavriplis. Multigrid techniques for unstructured meshes. Technical report, in VKI Lecture Series VKI-LS, 1995.

[119] D. J. Mavriplis. Multigrid strategies for viscous flow solvers on anisotropic unstructured meshes, 1998.

[120] D. J. Mavriplis. Adaptive meshing techniques for viscous flow calculations on mixed element unstructured meshes. *International Journal for Numerical Methods in Fluids*, 34(2):93–111, 2000.

[121] D. J. Mavriplis and K. Mani. Unstructured mesh solution techniques using the NSU3D solver. AIAA Paper 2014-81, 2014.

[122] V. Menier. 3D parallel anisotropic unsteady mesh adaptation for the high-fidelity prediction of bubble motion. *ESAIM Proceedings*, 50:169–188, 2015.

[123] V. Menier, A. Loseille, and F. Alauzet. CFD validation and adaptivity for viscous flow simulations. AIAA Paper 2014–2925, 2014.

[124] V. Menier, A. Loseille, and F. Alauzet. Multigrid strategies coupled with anisotropic mesh adaptation. AIAA Paper 2015–2041, 2015.

[125] Y. Mesri, H. Guillard, and T. Coupez. Automatic coarsening of three dimensional anisotropic unstructured meshes for multigrid applications. *Applied Mathematics and Computation*, 218(21):pages 10500–10509, April 2012.

[126] T. Michal and J. Krakos. Anisotropic mesh adaptation through edge primitive operations. AIAA Paper 2011-0159, 2011.

[127] W. F. Mitchell. The hp-multigrid method applied to hp-adaptive refinement of triangular grids. *Numerical Linear Algebra with Applications*, 17(2-3):211–228, 2010.

[128] E. Morano and A. Dervieux. Steady relaxation methods for unstructured multigrid euler and Navier-Stokes solutions. *International Journal of Computational Fluid Dynamics*, 5(3-4):137–167, 1995.

[129] J.-D. Müller. Anisotropic adaptation and multigrid for hybrid grids. *International Journal for Numerical Methods in Fluids*, 40(3-4):445–455, 2002.

[130] A. Nakayama. Characteristics of the flow around conventional and supercritical airfoils. *J. Fluid Mech.*, 160:155–179, 1985.

[131] C. R. Nastase and D. J.. Mavriplis. A parallel hp-multigrid solver for three-dimensional discontinuous galerkin discretizations of the euler equations. AIAA Paper 512, 2007.

[132] B. Nichols, D. Buttlar, and J. P. Farrell. *Pthreads Programming.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1996.

[133] National Research Council (U.S.). Committee on Assessment of National Aeronautical Wind Tunnel Facilities, R. Smelt, National Research Council (U.S.). Aeronautics, Space Engineering Board, National Research Council (U.S.). Commission on Engineering, and Technical Systems. *Review of Aeronautical Wind Tunnel Facilities.* National Academy Press, 1988.

[134] C. Özturan, H.L. deCougny, M.S. Shephard, and J.E. Flaherty. Parallel adaptive mesh refinement and redistribution on distributed memory computers. *Comput. Methods Appl. Mech. Engrg.*, 119(1–2):123 – 137, 1994.

[135] C. C Pain, A. P. Umpleby, 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.*, 190:3771–3796, 2001.

[136] F. Palacios, J. Alonso, K. Duraisamy, M. Colonno, J. Hicken, A. Aranake, A. Campos, S. Copeland, T. Economon, A. Lonkar, T. Lukaczyk, and T. Taylor. Stanford university unstructured (su^2): An open-source integrated computational environment for multi-physics simulation and design. AIAA Paper 2013-287, 2013.

[137] M. A. Park and J.-R. Carlson. Turbulent output-based anisotropic adaptation. AIAA Paper 2010–168, 2010.

[138] M. A. Park, A. Loseille, J. A. Krakos, and T. Michal. Comparing anisotropic output-based grid adaptation methods by decomposition. AIAA Paper 2015–2292, 2015.

[139] Y Penel, A. Mekkas, S. Dellacherie, J. Ryan, and M. Borrel. Application of an amr strategy to an abstract bubble vibration model. *AIAA Paper* , 2009.

[140] J. Peraire, J. Peiró, and K. Morgan. Adaptive remeshing for three-dimensional compressible flow computations. *J. Comp. Phys.*, 103:269–285, 1992.

[141] J. Peraire, M. Vahdati, K. Morgan, and O.C. Zienkiewicz. Adaptive remeshing for compressible flow computations. *J. Comp. Phys.*, 72:449–466, 1987.

[142] M. Picasso. An anisotropic error indicator based on Zienkiewicz-Zhu error estimator: Application to elliptic and parabolic problems. *SIAM J. Sci. Comp.*, 24(4):1328–1355, 2003.

[143] S. Pirzadeh. Viscous unstructured three dimensional grids by the advancing-layers method. *AIAA Paper* , 1994-0417, 1994.

[144] P.W. Power, C.C. Pain, M.D. Piggott, F. Fang, G.J. Gorman, A.P. Umpleby, and A.J.H. Goddard. Adjoint a posteriori error measures for anisotropic mesh optimization. *Computers & Mathematics with Applications*, 52:1213–1242, 2006.

[145] W. J. Rider and D. B. Kothe. Reconstructing volume tracking. *J. Comp. Phys.*, 141(2):112–152, 1998.

[146] C. L. Rumsey. Turbulence modeling verification and validation. AIAA Paper 2014–201, 2014.

[147] C. L. Rumsey. *Turbulence Modeling Verification and Validation (Invited)*. American Institute of Aeronautics and Astronautics, 2015/09/07 2014.

[148] U. Rüde. On the v-cycle of the fully adaptive multigrid method. In W. Hackbusch and G. Wittum, editors, *Adaptive Methods — Algorithms, Theory and Applications*, Notes on Numerical Fluid Mechanics (NNFM), pages 251–260. Vieweg+Teubner Verlag, 1994.

[149] H. Sagan. *Space-filling curves*. Universitext Series. Springer-Verlag, 1994.

[150] V. Schmitt and F. Charpin. Pressure distributions on the ONERA-M6-wing at transonic mach numbers. Technical report, Office National d'Etudes et Recherches Aerospatiales, 1979.

[151] V. Selmin and L. Formaggia. Simulation of hypersonic flows on unstructured grids. *International Journal for Numerical Methods in Engineering*, 34(2):569–606, 1992.

[152] M. Sharbatdar and C. O. P. Gooch. Anisotropic mesh adaptation: recovering quasi-structured meshes. *51th AIAA Aerospace Sciences Meeting*, Jan 2013.

[153] D. Sharov, H. Luo, J.D. Baum, and R. Löhner. Implementation of unstructured grid GMRES+LU-SGS method on shared-memory, cache-based parallel computers. *AIAA Paper* , 2000-0927, 2000.

[154] D. Sharov and K. Nakahashi. Reordering of hybrid unstructured grids for Lower-Upper Symmetric Gauss-Seidel computations. *AIAA Journal*, 36(1):484–486, 1997.

[155] M.S. Shephard, C. Smith, and J.E. Kolb. Bringing HPC to engineering innovation. *Comput. in Science Eng.*, 15(1):16–25, Jan 2013.

[156] C. W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *J. Comput. Phys.*, 77:439–471, 1988.

[157] P. R. Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. *AIAA Paper* , 92-0439, 1992.

[158] R. J. Spiteri and S. J. Ruuth. A new class of optimal high-order strong-stability-preserving time discretization methods. *SIAM J. Numer. Anal.*, 40(2):469–491, 2002.

[159] A. Talpaert. Analysis of the efficiency and relevance of the berger-rigoutsos and the livne cluster creation algorithms for patch-based AMR in the case of thin flagged areas. In *Poster presented at the 42nd CANUM*, 2014.

[160] 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.*, 189:1205–1230, 2000.

[161] U. Tremel, K.A. Sørensen, S. Hitzel, H. Rieger, O. Hassan, and N.P. Weatherill. Parallel remeshing of unstructured volume grids for CFD applications. *Int. J. Numer. Meth. Fluids*, 53(8):1361–1379, 2007.

[162] U. Trottenberg and A. Schuller. *Multigrid*. Academic Press, Inc., Orlando, FL, USA, 2001.

[163] J. Vassberg. Summary of the fourth drag prediction workshop. AIAA Paper 2009-4320, 2009.

[164] D. A. Venditti and D. L. Darmofal. Anisotropic grid adaptation for functional outputs of viscous flows. AIAA Paper 2003–3845, 2003.

[165] C. Viozat, C. Held, K. Mer, and A. Dervieux. On Vertex-Centered Unstructured Finite-Volume Methods for Stretched Anisotropic Triangulations. Technical Report RR-3464, INRIA, July 1998.

[166] K. A. Waithe. Application of USM3D for sonic boom prediction by utilizing a hybrid procedure. 2008.

[167] D. F. Watson. Computing the n-dimensional delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981.

[168] D.C. Wilcox. *Turbulence Modeling for CFD*. DCW Industries, Incorporated, 1994.

[169] M. Wintzer, M. Nemec, and M. J. Aftosmis. Adjoint-based adaptive mesh refinement for sonic boom prediction. AIAA Paper 2008-6593, 2008.

[170] M. Yano and D. L. Darmofal. An optimization-based framework for anisotropic simplex mesh adaptation. *J. Comp. Phys.*, 231(22):7626–7649, 2012.

[171] M. Yano, J.M. Modisette, and D.L. Darmofal. The importance of mesh adaptation for higher-order discretizations of aerodynamics flows. Number 2011-3852, 2011.

[172] O.C. Zienkiewicz and J.Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique. *Int. J. Numer. Meth. Engng*, 33(7):1331–1364, 1992.

[173] O.C. Zienkiewicz and J.Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. Part 2: Error estimates and adaptivity. *Int. J. Numer. Meth. Engng*, 33(7):1365–1380, 1992.