Time-accurate anisotropic mesh adaptation for three-dimensional moving mesh problems

Nicolas Barral

THESE DE DOCTORAT DE L'UNIVERSITE PIERRE ET MARIE CURIE

Spécialité: MATHEMATIQUES APPLIQUEES

Ecole Doctorale de Mathématiques de Paris Centre - ED 386

Présentée par

Nicolas Barral

Pour obtenir le grade de

Docteur de l'Université Pierre et Marie Curie

Time-accurate anisotropic mesh adaptation for three-dimensional moving mesh problems

soutenue le 27 Novembre 2015

devant le jury composé de:

Rapporteurs :	Prof.	Jean-Francois Remacle	-	Université Catholique de Louvain
	Dr.	Michel Visonneau	-	CNRS - Ecole Centrale de Nantes
Directeur de thèse :	Dr.	Frédéric Alauzet	-	INRIA Rocquencourt
Examinateurs :	Prof.	Thierry Coupez	-	Ecole Centrale de Nantes
	Dr.	Alain Dervieux	-	INRIA Sophia-Antipolis
	Dr.	Miguel Á. Fernández	-	INRIA / UPMC Paris VI
	Dr.	Paul Louis George	-	INRIA Rocquencourt
	Prof.	David L. Marcum	-	Mississippi State University

Remerciements

Je tiens à remercier tout d'abord Frédéric Alauzet, mon directeur de thèse, qui m'a donné la possibilité de découvrir le monde de la recherche en me proposant ce sujet. Je lui suis très reconnaissant de m'avoir fait partager son savoir et son expérience avec patience et disponibilité.

Je suis particulièrement reconnaissant envers Paul Louis George, responsable du projet Gamma, pour son encadrement strict mais juste dès mon arrivée dans le projet, et pour m'avoir patiemment transmis une petite part de son savoir (qui est grand...).

Je suis très honoré que Messieurs Jean-François Remacle et Michel Visonneau aient accepté de rapporter mon travail et je les en remercie. Je suis également reconnaissant envers Messieurs Thierry Coupez, Alain Dervieux, Miguel Fernández, Paul Louis George et Dave Marcum pour leur participation à mon jury.

Un grand merci à tous les membres du projet Gamma que je vois tous les jours depuis cinq ans, pour les discussions intéressantes avec certains, et les bons moments de détente avec d'autres, les deux ensembles n'étant pas d'intersection nulle: Adrien - qui mériterait une phrase à lui tout seul si mon répertoire de synonymes était plus étoffé-, Loïc, Dave, Houman, Patrick, Victorien, Alexis, Olivier, Loïc sans oublier Maryse pour son aide précieuse. Mention spéciale à Julien, Alex et Eleonore, pour m'avoir supporté dans le même bureau. Merci aussi à Stéphanie, Bruno et Estelle, ainsi qu'à Géraldine, sans son travail approfondi cette thèse n'aurait pas été bien loin. Mes remerciements à tous les gens des autres équipes et bâtiments croisés plus ou moins longuement ici et là.

Je remercie mes amis, qui auront supporté mes sautes d'humeur et que j'ai parfois injustement négligés, Antoine, Camille, Cédric, Jeanne, Lindsay, Maïssane, Marion, Mélanie, Nicolas, Paco, Sibylle, tous ceux d'IRC, Dominique, Benjamin, Florence, Cyprien, Fabien, Valentin, sans oublier tous ceux que j'oublie et qui ne m'en voudront pas.

Je voudrais également remercier la Principauté de Monaco, et la Direction de l'Education Nationale, de la Jeunesse et des Sports, pour leur soutien.

Enfin, je veux remercier ma famille pour son soutien sans faille tout au long de la thèse, malgré les épreuves douloureuses qu'elle traverse, à côté desquelles la convergence d'un résidu ou la réjection d'un papier sont bien peu de choses.

Contents

Résumé

Les simulations dépendant du temps sont devenues une nécessité pour l'industrie, et notamment celles mettant en oeuvre des géométries mobiles, avec mouvement imposé ou avec des interactions de type fluide-structures. Toutefois, de nombreuses difficultés subsistent qui empêchent d'effectuer de telles simulations de manière quotidienne à une précision acceptable. Parmi ces difficultés, les géométries mobiles, en particulier celles présentant des grands déplacements, ne sont en général pas traitées de façon satisfaisante, aussi bien du point de vue du temps de calcul que de la précision des algorithmes. L'adaptation de maillage anisotropes basée sur les métriques a prouvé son efficacité pour améliorer la précision des calculs stationnaires pour un coût en temps maîtrisé. Mais son extension aux problèmes dépendant du temps est loin d'être évidente, et l'ajout de géométries mobiles complique encore plus le problème. Dans cette thèse, on s'intéresse à divers éléments des simulations instationnaires en géométrie mobile, avec l'objectif global de les rendre automatiques et d'en améliorer les performances. Dans la continuité de [Olivier 2011a], des contributions sont apportées à tous les éléments de la chaîne, algorithme de bouger de maillage, solveur Arbitrairement Langrangien Eulerien (ALE) et adaptation instationnaire, conduisant à la mise au point d'un algorithme d'adaptation de maillage pour des simulations en géométrie mobile.

On considère un algorithme de bouger de maillage fondé sur des changements de connectivité (*swaps*) pour préserver la qualité du maillage. Une nouvelle méthode de calcul de la déformation du maillage, par interpolation directe (méthode IDW), a été ajoutée, et comparée à la méthode préexistante par résolution d'une EDP d'élasticité linéaire. Ces deux méthodes présentent toutes deux des performances excellentes, ce qui prouve la robustesse de l'algorithme pour gérer des maillages mobiles sans remaillage global, y compris avec de grands déplacements. Dans cette thèse, on aborde également les corps déformables avec des couches limites ou des grandes variations de volume.

Cet algorithme de bouger de maillage a été conçu pour être utilisé avec un solveur ALE, que l'on présente en détail. Une discrétisation par Volumes Finis est effectuée, avec un solveur de Riemann approché de type HLLC, et une reconstruction des gradients de type MUSCL pour obtenir un ordre 2 en espace. La discrétisation en temps repose sur une analyse fine de la Loi de Conservation Géométrique (GCL), qui permet de construire des schémas de Runge-Kutta à plusieurs pas d'ordres élevés. Le calcul de certaines quantités géométriques en trois dimensions est clarifié. Les changements de connectivité du maillage sont traités par une interpolation linéaire entre deux pas de temps du solveur.

Ce solveur a été implémenté en trois dimensions, parallélisé et validé sur plusieurs cas. Ces derniers ont confirmé les ordres de convergence attendus. En particulier, l'influence des swaps a été étudiée, et il en découle que le faible nombre de ces opérations rend leur influence négligeable sur la convergence et la précision des solutions. D'autres exemples tri-dimensionnels sont présentés dans le champ de la Mécanique des Fluides Numérique (*CFD*), plus complexes en terme de flux étudiés, de géométries et de déplacements des géométries, avec des déplacements imposés ou dérivant d'un couplage Fluide-Structure. Ils prouvent l'efficacité de l'approche retenue, qui permet de simuler des problèmes en maillage mobile complexes dans un temps

raisonnable sans jamais remailler globalement.

Des progrès ont été réalisés depuis le premier algorithme d'adaptation instationnaire de point fixe dans [Alauzet 2003a]. Dans cette thèse, une nouvelle version de l'algorithme de point fixe est utilisée, fondée sur une nouvelle analyse de l'erreur espace-temps dans le cadre de la théorie du maillage continu. Théorie et expérimentation ont permis de clarifier et valider certains points cruciaux du processus, notament le choix d'une Hessienne moyennée essentielle à l'algorithme. Grâce à une parallélisation efficace de type p-threads, dont on fournit une analyse, plusieurs cas d'adaptation instationnaire en 3D ont été simulés avec une précision difficilement concevable il y a encore quelques années.

Finalement, toutes ces briques ont été assemblées pour conduire à un algorithme d'adaptation pour simuler des phénomènes physiques instationnaires en maillage mobile utilisant l'algorithme de bouger de maillage avec changements de connectivité. La métrique ALE qui avait été présentée dans [Olivier 2011a] et permettait de bouger un maillage en un maillage adapté est intégrée à une analyse d'erreur fondée sur l'approche du maillage continu. Cela conduit à une correction de la métrique optimale dans l'algorithme de point fixe. En plus de l'utilisation de cette nouvelle métrique, d'autres ajouts ont été effecutés par rapport à l'algorithme standard pour utiliser des maillages mobiles. En particulier la métrique sur laquelle les optimisations de maillage de l'algorithme de bouger sont effectuées est modifiée. Finalement, plusieurs exemples sont présentés, qui confirment que l'adaptation de maillage anisotrope permet un gain de précision en un temps raisonnable pour les simulations en maillage mobile complexes.

Conventions

Simplicial mesh

Let n be the dimension of the physical space and let $\Omega \subset \mathbb{R}^n$ be the non-discretized physical domain. Ω is an affine space. The canonical basis of its vectorial space is noted $(\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n)$ in general, but notations $(\mathbf{e}_x, \mathbf{e}_y)$ and $(\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$ can also be used in two and three dimensions. Vectors of $\mathbb{R}^n$ are noted in bold font. The coordinate vector of a point of Ω is generally noted $\mathbf{x} = (x_1, \ldots, x_n)$.

The boundary of Ω, noted $\partial\Omega$, is discretized using simplicial elements the vertices of which are located on $\partial\Omega$. In two dimensions, $\partial\Omega$ is discretized with segments while in three dimensions, boundary surfaces $\partial\Omega$ are represented by triangles. The discretized boundary is noted $\partial\Omega_h$ and Ω_h denotes the sub-domain of $\mathbb{R}^n$ having $\partial\Omega_h$ as boundary.

Building a mesh of Ω_h consists in finding a set of *simplicial* elements - triangles in two dimensions, tetrahedra in three dimensions - noted $\mathcal{H}$, satisfying the following properties:

- *Non-degenerescence*: Each simplicial element K of $\mathcal{H}$ is non-degenerated (no flat elements of null area or volume),
- *Covering*: $\Omega_h = \bigcup_{K \in \mathcal{H}} K$,
- *Non-overlapping*: The intersection of the interior of two different elements of $\mathcal{H}$ is empty:

$$\mathring{K}_i \cap \mathring{K}_j = \varnothing, \forall\, K_i,\, K_j \in \mathcal{H},\, i \neq j\,.$$

- *Conformity*: The intersection of two elements is either a vertex, an edge or a face (in three dimensions) or is empty.

The *conformity* hypothesis is adopted here, because the meshes will be used with a Finite-Volume solver which requires the enforcement of this constraint. Besides, it facilitates the handling of data structures on the solver side and also enables to save a consequent amount of CPU time.
Finally, a mesh is said to be *uniform* if all its elements are almost regular (equilateral) and have the same size h.

Notations and orientation conventions

The following notations will be used in this thesis: K is an element of the mesh $\mathcal{H}$, P_i is the vertex of $\mathcal{H}$ having i as global index, $\mathbf{e}_{ij}$ is the edge linking P_i to P_j. The number of elements, vertices and edges are noted N_t, N_v and N_e, respectively.

The vertices and the edges of an element K are also numbered locally. Vertex numbering inside each element is done in a counter-clockwise (or trigonometric) manner, which enables

to compute edges/faces outward normals in a systematic way. This numbering, as well as unit outward normals $\mathbf{n}$ and edges/faces orientations are shown for triangles and tetrahedra in Figure 1. Non-normalized normals will be noted $\boldsymbol{\eta}$.

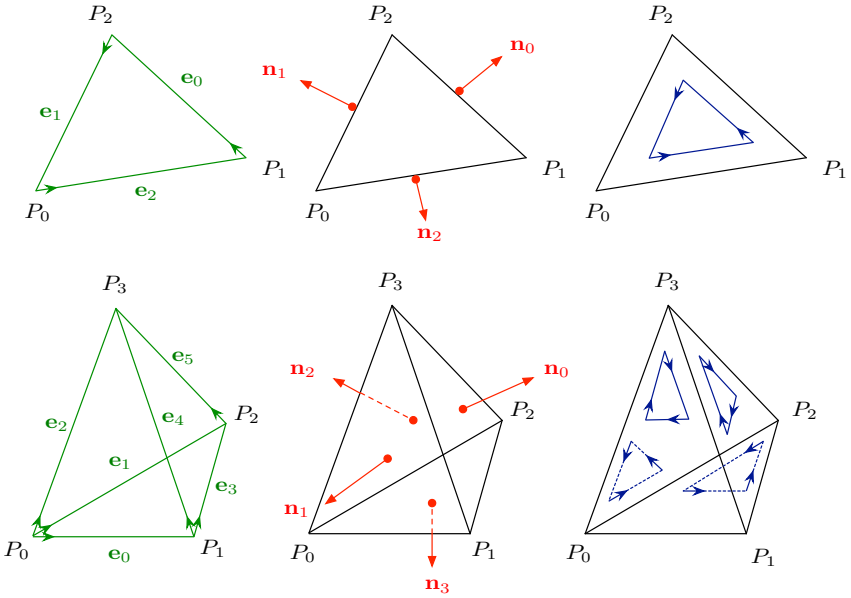

Figure 1: Conventions in a simplicial element K in two (top) and three (bottom) dimensions. Conventions for vertex numbering, edge numbering and orientation (left), unit normal numbering and orientation (middle) and face(s) orientation (right).

Introduction

In 1953, the first civil numerical simulation, also know as Fermi-Pasta-Ulam-Tsingou experiment, focused on a unidimensional system with 64 masses coupled with springs (*i.e.* 64 vertices). In sixty years, numerical simulation has come a long way, together with the development of computing resources. As it became possible to model and simulate more complex problems, industry started to take advantage of its potentialities. In particular, the aeronautic industry quickly saw a way to avoid costly full-size experiments, and has, together with military applications, been a driving force in the development of numerical simulation.

Nowadays, computers with hundreds of thousands of cores are available, and sophisticated numerical models enable running complex multiphysics simulations, involving fluid mechanics, structural mechanics, thermodynamics and electromagnetism. Most of the fields of scientific research resort to numerical simulation, from physics to social sciences through engineering and medicine, with more or less success. However, as the simulation capabilities grow, so do the requirements of both industries and researchers, and a whole set of problems have arisen. Notably, the increase in computational power allows scientists to not content themselves with steady approximations of the physical phenomena, and to study the effects of time-dependent parameters.

In this thesis, we focus on Computational Fluid Dynamics (CFD), *i.e.* the science of simulating numerically fluid flows, and its applications to aeronautic, although the results presented can be used in a vast field of research and industry. The theme of this thesis is *unsteady simulations*, and in particular *simulations with moving boundaries*. This subject is tackled from the points of view of *meshes* and *mesh adaptation*.

Problem

Most real-life phenomena are unsteady, but scientists are still far from running simulations close to real life on a daily basis. Problems arising from unsteady simulations are of multiple kind. To be used by industry, the results of numerical simulations have to be trusted, and the certification of the *accuracy* of the solutions is harder than for steady problems. Instabilities impact more the solution, and it is difficult to control the error in time at long temporal scales. Unsteady simulations are intrinsically longer than steady simulations, due to restrictions on the solvers time steps, and it is all the more difficult to control the error over time at a reasonable CPU time cost. Solutions to *efficiency* issues have to be found. Unsteady simulations often involve the interactions of one or several bodies evolving in one or several fluids, which require considering Fluid-Structure Interaction coupling techniques. Very often, moving boundaries have to be dealt with, which is often done by *moving the mesh*. This proves to be complex, especially for large displacements and in three dimensions, where meshing techniques are still evolving.

State of the art

Mesh adaptation has proved over the past twenty-five years its ability to increase significantly the accuracy of numerical simulations for a reasonable CPU time cost. A large number of different approaches have been considered, for various applications, from aerodynamics to biomechanics and magnetodynamics. *Metric-based mesh adaptation* has notably resulted in considerable gain in terms of accuracy and CPU time for steady problems [Hecht 1997, Bottasso 2004, Gruau 2005, Loseille 2010a, Guégan 2010]. A first attempt at extending it to unsteady simulations was carried out in [Alauzet 2003a], which was promising. This work was carried on in [Olivier 2011a], in which more solid theoretic bases were brought to support the method, and in which metric-based mesh adaptation was used for the first time on two dimensional moving meshes.

Objective

The works presented in this thesis follow on from [Olivier 2011a]. I focused on two aspects of mesh adaptation for moving mesh problems: on the one hand, the handling of moving meshes themselves, and on the other hand metric based mesh adaptation for unsteady simulations. First, I aimed at improving the strategy for simulations with moving meshes developed in [Olivier 2011a, Alauzet 2014a], based on a connectivity-change moving mesh algorithm coupled to an ALE solver. Variants to the algorithm needed to be considered, as well as the implementation and validation of the solver in 3D. Regarding unsteady simulations, the objective was to integrate the latest theoretic updates to the adaptation algorithm, both for fixed and moving meshes, and to extend to 3D the work of [Olivier 2011a] on adaptation for moving meshes.

Numerical context

The present work was conducted in the GAMMA3 research group at INRIA. Most numerical choices naturally stem from those made in the group, to benefit from both the considerable experience and the valuable software of its members.

We consider *unstructured meshes*, *i.e.* meshes made up of triangles (in 2D) or tetrahedra (in 3D). This is due to the fact that it is easier to mesh complex geometries with simplexes, resulting in several fully automatic unstructured three-dimensional meshing codes, and to the flexibility of such elements enabling anisotropic mesh adaptations.

Anisotropic *metric-based* mesh adaptation techniques are considered, which present several advantages compared to other adaptation techniques. First, it allows us to handle anisotropic meshes, which are of great value in mesh adaptation because they enable a more optimal distribution of the mesh vertices, and because the orientation of the elements is accorded to the physical phenomena at stake. Moreover, the theory of metric-based adaptation is well established, and several research groups contribute to its development. In particular, the recent advent of the continuous mesh framework [Loseille 2011a, Loseille 2011b] was a major breakthrough in this field, opening the way to a more rigorous analysis of error optimization problems. Finally, several research works establish the ability of metric-based mesh adaptation to handle complex steady simulations in three dimensions, which is promising for unsteady problems.

Regarding moving-boundary meshes, the *body-fitted approach* has been preferred to immersed boundaries and Chimera (overset) methods. We consider a unique body-fitted mesh, that follows the moving geometry in time, because it enables an accurate treatment of boundaries, unlike immersed boundaries methods where the boundary is not explicitly meshed, and because there is no interpolation step unlike Chimera methods where the solutions on the different sub-grids have to be transferred to one another, resulting in spoiling interpolation error. From the solver point of view, the classic *Arbitrary-Lagrangian-Eulerian (ALE)* framework is used, that enables writing the physical equations on a mesh moving with an arbitrary movement.

Finally, application examples have been chosen in the field of CFD, and only mono-fluid problems modeled by the inviscid compressible Euler equations are considered. Driven by industrial concern, most of the simulations are three dimensional.

Main contributions

During this thesis, I updated the existing algorithms both for moving mesh handling and for unsteady mesh adaptation. I implemented these updates and tested them on a wide variety of test cases. Notably, I extended the previous works of [Olivier 2011a] to 3D.

- I added a new mesh deformation method (the Inverse Distance Weighted (IDW) method) to our moving mesh code, and compared it to the elasticity-like method already implemented.
- I partially implemented and parallelized the ALE solver from [Olivier 2011a] in 3D, validated it on academic test cases and ran complex 3D ALE simulation both with imposed motion and FSI.
- I updated the unsteady adaptation algorithm with the latest theoretical results, using the continuous mesh framework to derive a better expression for the averaged "Hessian-metric", and ran 3D simulations of unprecedented size thanks to an efficient parallelization of the algorithm.
- I updated the error analysis from [Olivier 2011a] for adaptative moving mesh problems, implemented the resulting algorithm, and ran several 3D adaptive simulations with moving boundaries.

Outline

This thesis is structured in two parts, that cover the two aspects of adaptation for moving mesh problems.

In the first part (Part I), aspects linked to the moving mesh computations are considered. First, we focus on the moving mesh strategy used in the whole thesis, describing every aspects of the algorithm, and analyzing the performance of updates to this algorithm. It is notably based on mesh connectivity changes (edge/face swaps), to preserve the quality of the mesh throughout large displacements without remeshing. This algorithm was designed to run ALE simulations on the moving meshes, and an ALE solver was actually implemented in 3D. It is described in details, from the formulation of the equations and the choices of numerical methods to implementation and parallelization considerations. Validation examples of the solver are presented, followed by

more complex CFD cases coupling the ALE solver and the connectivity-change moving mesh algorithm, proving the efficiency of the approach in 3D.

The second part (Part II) deals with mesh adaptation, from the basics of metric-based adaptation to adaptation with moving meshes, through unsteady simulation on fixed meshes. All the theoretical background necessary for the rest of the part is recalled, notably the continuous mesh framework and multiscale mesh adaptation. The continuous mesh framework enabled a new analysis of the space-time error for unsteady simulations, which is presented, and leads to a new version of the unsteady adaptation algorithm. Practical aspects of the implementation of this algorithm are reviewed, and new 3D examples are given. This algorithm is extended to moving meshes: the ALE metric introduced in [Olivier 2011a] is plugged into the unsteady error analysis, and the resulting adaptation algorithm is described. Finally, examples of 3D adaptative simulations with moving mesh are exhibited, that validate the whole approach followed in this thesis.

Scientific communications

Journal articles

- **Three-dimensional large displacement CFD simulations using a connectivity-change moving mesh approach**, N. Barral and F. Alauzet, Journal of Computational Physics, submitted.

- **Construction and geometric validity (positive Jacobian) of serendipity Lagrange finite elements, theory and practical guidance**, P.L. George, H. Borouchaki and N. Barral, Mathematical Modelling and Numerical Analysis, submitted.

- **Geometric validity (positive Jacobian) of high-order Lagrange finite elements, theory and practical guidance**, P.L. George, H. Borouchaki and N. Barral, Engineering with computers, accepted.

Proceedings with peer review

- **Metric-Based Anisotropic Mesh Adaptation for Three-Dimensional Time-Dependent Problems Involving Moving Geometries**, N. Barral, F. Alauzet and A. Loseille, 53th AIAA Aerospace Sciences Meeting, AIAA Paper 2015-2039, Kissimmee, FL, USA, Jan 2015.

- **Two mesh deformation methods coupled with a changing-connectivity moving mesh method for CFD Applications**, N. Barral, E. Luke and F. Alauzet, Proc. of the 23th International Meshing Roundtable, Procedia Engineering, vol 82, pp. 213-227, London, England, 2014.

- **Large displacement body-fitted FSI simulations using a mesh-connectivity-change moving mesh strategy**, N. Barral and F. Alauzet, 44th AIAA Fluid Dynamics Conference, AIAA Paper 2014-2773, Atlanta, GA, USA, June 2014.

Communications

- **Anisotropic Error Estimates for Adapted Dynamic Meshes**, N. Barral and F. Alauzet, ADMOS 2015, Nantes, France, June 2015.
- **Large displacement simulations with an efficient mesh-connectivity-change moving mesh strategy**, N. Barral and F. Alauzet, ECCOMAS 2014, Barcelona, Spain, July 2014.
- **Parallel time-accurate anisotropic mesh adaptation for time-dependent problems**, N. Barral and F. Alauzet, ECCOMAS 2014, Barcelona, Spain, July 2014.

Research reports

- **Construction et validation des éléments Serendip associés à un carreau de degré arbitraire**, P.L George, H. Borouchaki and N. Barral, INRIA RR-8572, 2014.
- **Construction et validation des éléments réduits associés à un carreau simplicial de degré arbitraire**, P.L George, H. Borouchaki and N. Barral, INRIA RR-8571, 2014.

Talks and seminars

- **Adaptation de maillages non structurés pour des problèmes instationnaires, et maillage en géométrie mobile**, N. Barral, Groupe de travail "Analyse Numérique et EDP", Ecole Centrale Paris, November 2014.
- **Du réel au numérique : la science des maillages**, P.L. George and N. Barral, Pint of Science, May 2015.

Awards

- **IMR Meshing Contest Award**, 23th International Meshing Roundtable, London, October 2014.

Part I

3D FSI moving mesh simulations

Introduction

Fluid-structure interaction (FSI) simulations are required for a wide variety of subjects, from the simulation of jellyfish [Etienne 2010] to the releasing of a missile [Murman 2003, Hassan 2007], through the simulation of aortic valves [Astorino 2009], cardiovascular systems [Formaggia 2009, Gerbeau 2014], ship propellers [Compère 2010], tuning forks [Froehle 2014], wind turbines [Bazilevs 2011], 2D airbag deployment and balloon inflation [Saksono 2007]. Other problems such as icing [Tong 2014, Pendenza 2014] or blast studies [Baum 1996] do not involve FSI but share a lot of common problems with FSI simulations. The recent development of computing capacities has made it possible to run increasingly complex simulations where moving bodies interact with an ambient fluid in an unsteady way. However, engineers are still far from performing such simulations on a daily basis, largely due to the difficulty of handling the moving meshes induced by the moving geometries.

When the displacement of the geometry is small enough, slightly deforming the original mesh [Batina 1990, Degand 2002] can generally be acceptable. But when large deformation of the boundaries is considered, the mesh quickly becomes distorted and the numerical error due to this distortion quickly becomes too great, until the elements of the mesh finally become invalid, and the simulation has to be stopped. Specific strategies need to be developed to deal with large displacement moving boundary problems. In the case of FSI problems, another difficulty arises from the fact that the displacement of the boundaries is by definition an unknown, and the deformation of the mesh cannot be imposed *a priori*. Addressing the issue of the mesh movement cannot be separated from addressing the issue of the solver that will compute on these moving meshes. Depending on the strategy employed to deal with the movement, specific numerical methods must be designed to take into account the displacement of the mesh.

The question this part focuses on is: how can we efficiently move the mesh for large displacement 3D FSI simulations and what numerical schemes need to be associated to such a strategy ?

Three main approaches to address the mesh movement problem can be found in the literature. The first approach consists in having a single body-fitted mesh [Baum 1994, Hassan 2007], and moving it along with the moving boundaries. The mesh may thus undergoe large deformation. The second approach is the Chimera (or overset) method [Benek 1985], in which each moving body has its own body-fitted sub-mesh, and the sub-meshes move rigidly together with their body and can overlap one another. Finally the embedded boundary approach [Bruchon 2009, Löhner 2001] uses meshes that are not body-fitted at all: the bodies are embedded in a fixed grid, and techniques such as level-sets are used to recover their moving boundaries. All three approaches have their own strengths and weaknesses. For meshing purposes and to fit our mesh adaptation context, we focus on body-fitted approaches with one single mesh. Our approach was introduced in [Alauzet 2014a] and was enhanced during this thesis.

In some works, a mesh motion that is directly adapted to the physical phenomena in question is applied, using for instance either so-called Moving Mesh PDEs [Huang 2010b] or a Monge-Ampère equation [Chacón 2011]. More generally, these approaches come under the *r-adaptation*

category (see Section 4.1.3). However interesting these approaches may be, they still seem to be time consuming, especially in 3D, due to the solution of a non-linear equation, and it is unsure whether they can handle complex 3D geometries. Therefore we prefer to prescribe arbitrary movements to the mesh, and use our mesh adaptation framework [Loseille 2011a] when necessary, as will be detailed in Part II, Chapter 6.

As regards numerical solvers, we consider a classic framework for moving meshes: the Arbitrary Lagrangian Eulerian (ALE) framework, which is based on a formulation of the equations that takes into account an arbitrary movement of the vertices. This technique was introduced in the 70s in [Hirt 1974, Hughes 1981, Donea 1982]. Since then, so many developments have been made in that field that a complete list of them would not fit in this thesis. However, one may in particular refer to [Nkonga 1994, Baum 1994, Farhat 2001, Formaggia 2004, Mavriplis 2006, Hassan 2007, Hay 2014], which mainly focus on improving temporal schemes for ALE simulations.

To our knowledge, very few examples of ALE solvers coupled with connectivity-change moving mesh techniques can be found in the literature. In [Kucharik 2008] a conservative interpolation is proposed to handle the swaps. In [Guardone 2011, Olivier 2011b] an ALE formulation of the swap operator is built. However, these studies are limited to 2D. Driven by the requirements of industry, we are interested in designing a method that works in 3D. In this thesis, a linear interpolation is carried out after each swap instead of using a specific ALE formulation, and we will evaluate the numerical error due to these swaps.

The goal of this part is to demonstrate that three-dimensional FSI simulations with large displacement can be run efficiently by coupling an ALE solver to our connectivity-change moving mesh strategy. This part is structured as follows. In Chapter 1, the moving mesh strategy used and improved during this thesis is described in detail, and analyzed on purely moving-mesh simulations. In Chapter 2, the ALE solver that was implemented in 3D during this thesis is described, from the formulation of the equations to implementation considerations. Finally Chapter 3 presents a large number of moving-mesh ALE simulations, both academic test cases and more complex CFD simulations, coupling the moving mesh strategy and ALE solver.

Chapter 1

Connectivity-change moving mesh strategy

When dealing with a unique body-fitted moving mesh, two main approaches are used.
The first one is simply to move the mesh for as long as possible, with a constant topology, and remesh when the quality becomes critical. For each remeshing, the simulation has to be stopped, a new mesh must be generated (a whole new mesh or parts of it), new data structures must be created, and the solution must be transferred to the new mesh. This approach can be efficient, especially when small displacements are considered and very few remeshings are necessary, because the solver and the meshing aspects are decoupled. Between two remeshings the simulation is fully ALE and free from interpolation errors due to connectivity changes. However, for larger displacements, the number of remeshings increases to prevent invalid elements from appearing, especially if shearing appears. This can be costly in terms of CPU, but also result in poor accuracy due to the numerous solution transfer steps. This strategy is for instance used in [Murman 2003] with hexaedral Cartesian meshes, or [Löhner 1990a, Hassan 2000, Jayaraman 2000] on unstructured meshes.
The second approach tries to preserve a good mesh quality throughout the simulation by performing local remeshing operations such as vertex insertion, vertex collapse, connectivity changes and vertex displacements. It is used for example in [Dobrzynski 2008, Compère 2010]. The advantage of this method is that it maintains an acceptable mesh quality without needing to stop, remesh and resume the simulation. However, it requires fully dynamic mesh data structures that are permanently updated, which can lead to a loss of CPU efficiency, and the numerous mesh modifications tend to lead to a loss of accuracy due to interpolation errors.

In our case, we want to address the case of large and complex displacements of the boundaries, with potentially highly non-linear trajectories as may occur in FSI simulations. This requires that the moving mesh strategy fulfills a number of properties:

- It must above all preserve the validity of the mesh, and never create elements with a negative volume.
- It must preserve the quality of the elements of the mesh all along the simulation, to ensure a good solution accuracy and reasonable solver time steps.
- It must be efficient in terms of CPU since the mesh must be moved at each solver iteration. It must not take for more than a small fraction of the total simulation time.
- It must be robust, and be able to deal with very small elements, and large shearing constraints.

- It must be able to preserve the anisotropic adaptation if any.

For efficiency purposes, we add that we want to remesh as little as possible, and that most cases should be run without remeshing at all.

Remark 1. *What we call* remeshing *entails: stopping the simulation, calling meshing software to generate either a whole new mesh or parts of it, and interpolate the solution on the new mesh. This is costly because (a) it usually requires the use of an external code, with its own data structures, requiring CPU-expensive data transfers, (b) the meshing step can be long (and might not even complete) in the case of complex geometries, (c) an interpolation step which produces not too much error is long too, especially if the whole mesh or large portions of it are considered. On the other hand, our mesh optimizations (a) are performed on the go, within the numerical solver code, using the same semi-dynamic data-structures, (b) concern only a limited set of elements, and are performed element by element, only if the quality of the element requires them, (c) no global interpolation procedure is required since the number of vertices is constant.*

The previous strategies do not match these requirements, either in terms of preserving the mesh quality or in terms of efficiency for large displacements. That is why we adopt a strategy based on only vertex displacements and connectivity changes. Improvements in the way to compute the displacement of the vertices makes that strategy very efficient, while the connectivity changes makes it robust, as will be demonstrated in Section 1.3. Besides its CPU efficiency, this strategy presents some very interesting features considering numerical simulations:

- Since no vertex insertion or deletion occurs, and no remeshing is performed, a source of large spoiling interpolation errors is removed, and the accuracy of the numerical simulations should be improved. The use of an ALE treatment for connectivity changes would allow the simulation to be fully ALE.

- It is especially efficient with a vertex-centered Finite-Volume approach, since the number of vertices is preserved, the number of dual cells is preserved as well.

- Our unsteady adaptation algorithm, that will be detailed in Part II, Chapter 5, requires a set number of vertices within each adaptation sub-interval to control the error. Our strategy keeps constant the number of vertices and is thus compatible with our adaptation algorithm.

One might wonder why perform connectivity changes (edge/face swaps) in the first place, if they are *a priori* not ALE compliant. First, performing local mesh modifications within the solver is far simpler and more efficient than remeshing globally. Moreover, connectivity changes can be relatively simply interpreted in terms of evanescent cells for purposes of ALE numerical schemes. In many body-fitted moving mesh strategies, a lot of CPU time is dedicated to computing the displacement of the mesh, in order to make it follow the moving boundaries. Thanks to frequent mesh optimizations, the cost of this step is reduced by computing the mesh deformation for a large number of solver time steps (*i.e.* we do it only a few times during the simulation).

In the present chapter, I will first detail the moving mesh algorithm that was used during this thesis (Section 1.1), and present a variant of the algorithm developed during the thesis

(Section 1.2). Then I will present various examples of purely moving mesh simulations to study the efficiency of the algorithm and compare its two variants. The behavior of the algorithm with viscous-like boundary layers is studied in Section 1.4. In these sections, we only consider displacements with no volume variation, or a small one. The case of large volume deformation is tackled at the end of the chapter in Section 1.5.

The works presented in this Chapter were published in [Barral 2014d].

1.1 Our moving mesh algorithm

1.1.1 A two step process

The connectivity-change moving mesh algorithm (MMA) is a two-step process:

- A mesh deformation step, in which a trajectory is assigned to each inner vertex, depending on the displacement of the boundaries.
- A mesh optimization step, in which the trajectories of the vertices are modified, and connectivity changes are performed, to preserve the quality of the mesh.

1.1.2 Mesh deformation step

For body-fitted moving mesh simulations, the whole mesh must be deformed to follow the moving boundaries. During the mesh deformation step, a displacement field is computed for the whole computational domain, given the displacement of its boundaries. Trajectories, or in other words, positions at a future solver time step, can thus be assigned to inner vertices.

Several techniques can be found to compute this displacement field:

- implicit or direct interpolation [de Boer 2007, Luke 2012], in which the displacement of the inner vertices is a weighted average of the displacement of the boundary vertices,
- solving PDEs - the most common of which being Laplacian smoothing [Löhner 1998], Winslow equation [Masters 2015], a spring analogy [Degand 2002] and a linear elasticity analogy [Baker 1999].

It is this last method that we selected at first, due to its robustness in 3D [Yang 2007]. In Section 1.2, a direct interpolation procedure will also be considered.

In the case of the elasticity analogy, the computational domain is assimilated to a soft elastic material, which is deformed by the displacement of its boundaries. More precisely, the inner vertices movement is obtained by solving an *elasticity-like equation* with a $\mathbb{P}_1$ Finite Element Method (FEM):

$$\operatorname{div}(\sigma(\mathcal{E})) = 0\,, \quad \text{with} \quad \mathcal{E} = \frac{\nabla \mathbf{d} + {}^T\nabla \mathbf{d}}{2}\,, \tag{1.1}$$

where σ and $\mathcal{E}$ are respectively the Cauchy stress and strain tensors, and $\mathbf{d}$ is the Lagrangian displacement of the vertices. The Cauchy stress tensor follows Hooke's law for isotropic homogeneous medium, where ν is the Poisson ratio, E the Young modulus of the material and λ, μ

are the Lamé coefficients:

$$\sigma(\mathcal{E}) = \lambda \,\mathrm{trace}(\mathcal{E})\,\mathcal{I}_d + 2\,\mu\,\mathcal{E} \qquad \text{or} \qquad \mathcal{E}(\sigma) = \frac{1+\nu}{E}\sigma - \frac{\nu}{E}\,\mathrm{trace}(\sigma)\,\mathcal{I}_d\,.$$

In our context, ν is typically chosen of the order of 0.48, which corresponds to a very soft, nearly incompressible material [1]. Dirichlet boundary conditions are used and the displacement of vertices located on the domain boundary is strongly enforced in the linear system.

The linear system is then solved with a $\mathbb{P}^1$ Finite Element Method (FEM) by a Conjugate Gradient algorithm coupled with an LU-SGS pre-conditioner (See [Olivier 2011a] for more details).

Basic elasticity-based algorithm

The classic algorithm to move the mesh using the elasticity-based deformation method is simply to solve the elasticity problem at every time step and then move the mesh using the solution of the problem. This strategy is described in Algorithm 1. In this algorithm, $\mathcal{H}$ stands for meshes, $\mathcal{S}$ for solutions and $\mathbf{v}$ for vertex speed.

Algorithm 1 Basic Moving Mesh Algorithm

Input: $\mathcal{H}^0$, $\mathcal{S}^0$

While $(t < T^{end})$

1. δt = Get solver time step $\left(\mathcal{H}^k, \mathcal{S}^k, \mathbf{v}, CFL\right)$
2. Solve mesh deformation: compute vertices trajectories
   (a) $\left\{\mathbf{d}^{body}_{|\partial\Omega_h}\right\}$ = Compute body vertex displacement from current translation speed $\mathbf{v}^{body}$, rotation speed $\boldsymbol{\theta}^{body}$ and acceleration $\mathbf{a}^{body}$ for $[t, t+\delta t]$
   (b) $\mathbf{d}$ = Solve elasticity system $\left(\mathbf{d}^{body}_{|\partial\Omega_h}, \delta t\right)$
   (c) $\mathbf{v}$ = Deduce inner vertices speed from displacement $\mathbf{d}$
3. $\mathcal{S}^{k+1}$ = Solve Equation of State $\left(\mathcal{H}^k, \mathcal{S}^k, \delta t, \mathbf{v}\right)$
4. $\mathcal{H}^{k+1}$ = Move the mesh $\left(\mathcal{H}^k, \delta t, \mathbf{v}\right)$
5. $t = t + \delta t$

EndWhile

The computation of the mesh deformation - here the solution of a linear elasticity problem - is known to be an expensive part of dynamic mesh simulations. Performing it at every solver time step makes it all the more so. In order to reduce significantly the CPU time of that part, this step needs to be enhanced.

[1] Note that the closer to 0.5 ν is, the harder to solve the system is.

Improvements to the mesh deformation steps

Several improvements to the mesh deformation step are brought compared to the basic approach.

Local material properties alteration. First, the linear elasticity analogy allows us to modify the local material properties according to the distortion and efforts born by each element. In particular, small elements are made stiffer than others, which has the advantage of limiting their deformation, and thus avoiding the creation of false elements. To do so, the way the Jacobian of the transformation from the reference element to the current element is accounted for in the FEM matrix assembly is modified as in [Stein 2003]. The classic $\mathbb{P}_1$ FEM formulation of the linear elasticity matrix leads to the evaluation of quantities of the form:

$$\int_K s \frac{\partial \varphi_J}{\partial x_k} \frac{\partial \varphi_I}{\partial x_l} \, \mathrm{d}\mathbf{x} = s \, |K| \, \frac{\partial \varphi_J}{\partial x_k} \frac{\partial \varphi_I}{\partial x_l} \, ,$$

where s is either λ, μ or $\lambda + 2\mu$ and $|K|$ is the volume of tetrahedron K. The above quantity is replaced by:

$$\int_K s \left(\frac{|\hat{K}|}{|K|} \right)^{\chi} \frac{\partial \varphi_J}{\partial x_k} \frac{\partial \varphi_I}{\partial x_l} \, \mathrm{d}\mathbf{x} = s \, |K| \, \left(\frac{|\hat{K}|}{|K|} \right)^{\chi} \frac{\partial \varphi_J}{\partial x_k} \frac{\partial \varphi_I}{\partial x_l} \, ,$$

where $\chi > 0$ is the stiffening power and $\hat{K}$ is the reference element. This technique locally multiplies λ and μ by a factor proportional to $|K|^{-\chi}$. χ determines how stiffer than large elements small elements are. In this work, we chose $\chi = 1$.

Rigidify regions. Some regions of the mesh can also be rigidified, *i.e.* their vertices are all moved with the same displacement. Typically, this is done for a few layers of tiny elements around moving rigid bodies, that are moved together with the body. Rigidifying elements close to complex features of the geometries reduces the risk of creating bad elements in those regions, and also removes stiff elements from the FEM matrix. We can also chose to enclose moving bodies with a complex geometry into boxes or spheres, and rigidify the whole region inside these boxes and spheres: the objects moved are then much simpler.

Elasticity on reduced regions. The elasticity can also be solved on a reduced region: if the domain is very large compared to the size of the moving bodies and the length of the displacement, it is sometimes possible to solve the elasticity problem for a few layers around the moving bodies only. The idea is the same as the previous one, but now it is the far field that is rigidified. It has to be noted that the elastic region has to be large enough to let the moving bodies evolve in it.

Elasticity dedicated mesh. A way to reduce significantly the CPU time of the mesh deformation computation is to consider two meshes: one on which the physical equations are solved, and one on which the elasticity problem is solved. The elasticity-dedicated mesh can be much coarser than the computational one, since a precise elasticity solution is not needed - especially when using mesh optimizations as described below. This strategy could also be mandatory when considering adapted meshes with high aspect-ratio elements, on which the FEM elasticity system could be practically impossible to solve.
More precisely, we consider two meshes with the same boundary mesh, the elasticity-dedicated mesh being coarser than the computational one. The displacement of the boundary vertices

is the same for both meshes (whether it is imposed to the bodies, or whether a FSI motion is considered: in this case, the displacement of the bodies is computed on the computational mesh, then transferred to the elasticity mesh). The elasticity FEM system is then assembled on the coarse mesh, and solved. The solution of the elasticity problem is then transferred to the computational mesh using a $\mathbb{P}^2$-Lagrangian scheme, which is sufficiently accurate considering the intrinsic smoothness of the solution of the above elasticity equation. Then both meshes are moved with the same time step. If mesh optimizations are performed, they are performed on both meshes, so that the elasticity mesh remains valid too. The cost of moving the coarse mesh is shown to be very small compared to the gain in solving the elasticity problem.

Reduced number of solutions. The main improvement to the basic algorithm 1 is that we do not solve an elasticity problem at each FSI solver time step δt. Instead, we consider a larger time step Δt, and the elasticity problem is solved for this time step, and the trajectory of the vertices is kept constant on this whole larger time step. While there is a risk of a less good mesh displacement solution, this strategy is worthy if mesh optimizations are used to preserve the mesh quality, as explained in Section 1.1.3.

High order trajectories. Since the trajectories are computed for a large time step, it is worthwhile to consider vertex paths that are more complex than mere straight lines (linear trajectories). This is especially true if the moving bodies undergo rotation or accelerated movements. The paths of inner vertices is improved providing a constant acceleration $\mathbf{a}$ to each vertex in addition to its speed, which results in an accelerated and curved trajectory. During time frame $[t, t+\Delta t]$, the position $\mathbf{x}$ and the velocity $\mathbf{v}$ of a vertex are updated as follows:

$$\begin{aligned} \mathbf{x}(t+\delta t) &= \mathbf{x}(t) + \delta t\, \mathbf{v}(t) + \frac{\delta t^2}{2}\mathbf{a} \\ \mathbf{v}(t+\delta t) &= \mathbf{v}(t) + \delta t\, \mathbf{a}\,. \end{aligned}$$

Prescribing a velocity vector and an acceleration vector to each vertex requires solving two elasticity systems. For both systems, the same matrix, thus the same pre-conditioner, is considered. Only boundary conditions change. If inner vertex displacement is sought for time frame $[t, t+\Delta t]$, boundary conditions are imposed by the location of the body at time $t+\Delta t/2$ and $t+\Delta t$. These locations are computed using body velocity and acceleration. Note that solving the second linear system is cheaper than solving the first one as a good prediction of the expected solution can be obtained from the solution of the first linear system. Now, to define the trajectory of each vertex, the velocity and acceleration are deduced from evaluated middle and final positions:

$$\begin{aligned} \Delta t\, \mathbf{v}(t) &= -3\,\mathbf{x}(t) + 4\,\mathbf{x}(t+\Delta t/2) - \mathbf{x}(t+\Delta t) \\ \frac{\Delta t^2}{2}\mathbf{a} &= 2\,\mathbf{x}(t) - 4\,\mathbf{x}(t+\Delta t/2) + 2\mathbf{x}(t+\Delta t)\quad . \end{aligned}$$

In this context, it is mandatory to make sure that the mesh remains valid for the whole time frame $[t, t+\Delta t]$, which is done computing by the sign of the volume of the elements all along their path [Alauzet 2014a].

1.1.3 Mesh optimization step

Mesh optimizations are performed regularly to preserve the quality of the mesh. The optimization procedure consists in one or several passes of vertex smoothing and one or several passes of generalized swapping, and is described below.

For 3D adapted meshes, an element's quality is measured in terms of the element's shape by the quality function:

$$Q_{\mathcal{M}}(K) = \frac{\sqrt{3}}{216} \frac{\left(\sum_{i=1}^{6} \ell^2_{\mathcal{M}}(\mathbf{e}_i) \right)^{\frac{3}{2}}}{|K|_{\mathcal{M}}} \quad \in [1, +\infty] \,, \tag{1.2}$$

where $\ell_{\mathcal{M}}(\mathbf{e})$ and $|K|_{\mathcal{M}}$ are the edge lengths and element volume in metric $\mathcal{M}$. Metric $\mathcal{M}$ is a 3×3 symmetric positive definite tensor prescribing element sizes, anisotropy and orientation to the mesh generator (see Chapter 4). $Q_{\mathcal{M}}(K) = 1$ corresponds to a perfectly regular element and $Q_{\mathcal{M}}(K) < 2$ correspond to excellent quality elements, while a high value of $Q_{\mathcal{M}}(K)$ indicates a nearly degenerated element. For non-adapted (uniform) meshes, the identity matrix $\mathcal{I}_3$ is chosen as metric tensor.

Mesh smoothing. The first mesh optimization tool is vertex smoothing which consists in relocating each vertex inside its ball of elements, *i.e.,* the set of elements having P_i as their vertex. For each tetrahedron K_j of the ball of P_i, a new optimal position P_j^{opt} for P_i can be proposed to form a regular tetrahedron:

$$P_j^{opt} = G_j + \sqrt{\frac{2}{3}} \frac{\mathbf{n}_j}{\ell(\mathbf{n}_j)} \,,$$

where F_j is the face of K_j opposite vertex P_i, G_j is the center of gravity of F_j, $\mathbf{n}_j$ is the inward normal to F_j and $\ell(\mathbf{n}_j)$ the length of $\mathbf{n}_j$. The final optimal position P_i^{opt} is computed as a weighted average of all these optimal positions $\{P_j^{opt}\}_{K_j \supset P_i}$, the weight coefficients being the quality of K_j. This way, an element of the ball is all the more dominant if its quality in the original mesh is bad. Finally, the new position is analyzed: if it improves the worst quality of the ball, the vertex is directly moved to its new position.

Edge/face swapping. The second mesh optimization tool to improve mesh quality is generalized swapping/local-reconnection. Let α and β be the two tetrahedra vertices opposite the common face $P_1P_2P_3$. Face swapping consists of suppressing this face and creating the edge $\mathbf{e} = \alpha\beta$. In this case, the two original tetrahedra are deleted and three new tetrahedra are created. This swap is called $2 \to 3$. The reverse operator can also be defined by deleting three tetrahedra sharing such a common edge $\alpha\beta$ and creating two new tetrahedra sharing face $P_1P_2P_3$. This swap is called $3 \to 2$.

A generalization of this operation exists and acts on shells of tetrahedra [Alauzet 2014a, Frey 2008]. For an internal edge $\mathbf{e} = \alpha\beta$, the shell of $\mathbf{e}$ is the set of tetrahedra having $\mathbf{e}$ as common edge. From a shell of size n, a non-planar *pseudo-polygon* formed by n vertices $P_1 ... P_n$ can be defined. Performing a three-dimensional swap of edge $\alpha\beta$ requires (i) suppressing edge $\alpha\beta$ and all tetrahedra of the shell, (ii) defining a triangulation of the pseudo-polygon $P_1 ... P_n$ and (iii) finally creating new tetrahedra by joining each triangle of the pseudo-polygon with

vertices α and β. The number of possible triangulations depends on n the number of vertices of the pseudo-polygon. The different swaps are generally denoted $n \to m$ where m is the number of new tetrahedra. In this work, edge swaps $3 \to 2$, $4 \to 4$, $5 \to 6$, $6 \to 8$ and $7 \to 10$ have been implemented. In our algorithm, swaps are only performed if they improve the quality of the mesh.

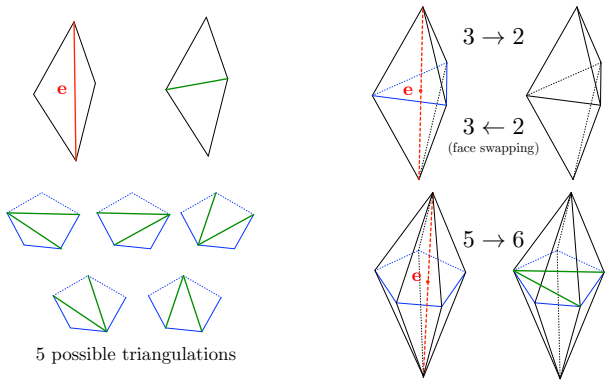

Figure 1.1: Top left, the swap operation in two dimensions. Top right, edge swap of type $3 \to 2$ and face swap $2 \to 3$. Bottom left, the five possible triangulations of the pseudo-polygon for a shell having five elements. Bottom right, an example of $5 \to 6$ edge swap. For all these figures, shells are in black, old edges are in red, new edges in green and the pseudo-polygon is in blue.

1.1.4 Optimization procedure

The optimization procedure is made up of one or several passes of swapping and then one or several passes of smoothing. The number of passes is given by the user as input.

The time step δt^{opt} after which optimizations are performed is defined using a parameter named CFL^{geom}. This parameter tells how many elements a mesh vertex is virtually crossing along his trajectory before optimizations become necessary to preserve the quality of the mesh. It is named from the CFL number, which controls the propagation of the information that can be accepted within one solver time step. In other words, the optimization stage is performed when the mesh "reaches a certain degree of deformation". The optimization time step is given by:

$$\delta t^{opt} = CFL^{geom} \max_{P_i} \frac{h(\mathbf{x}_i)}{\mathbf{v}(\mathbf{x}_i)}, \tag{1.3}$$

where $h(\mathbf{x}_i)$ is the smallest height of all the elements in the ball of vertex P_i, and $\mathbf{v}(\mathbf{x}_i)$ is the speed of vertex P_i.

The swaps are performed according to an analysis of the quality of the elements at the current time and in the future. First, a list of tetrahedra whose current quality is bigger than a certain threshold Q_{swap} is built. This list is then sorted in descending order of the current quality, so that the worst elements are on top of the list. Then the list is processed from top to bottom. For each element, and for each edge/face of the element, all the swaps in the corresponding shell of tetrahedra -among the swap operators implemented - are simulated, and the resulting qualities both at current time and in the future and configurations are analyzed: if they are smaller

than a certain factor c_{swap} times the quality of the current configuration, the configuration is stored temporarily, else it is discarded. Once all the possible swaps for one element have been simulated, if no new configuration has been stored, the mesh remains unchanged, and if one or several new configurations have been stored, the one with the best quality at current time is selected, and the swap is actually performed. The neighboring tetrahedra are then frozen for the remaining of the pass, and a new tetrahedron from the list is analyzed and swapped.

As concerns vertex smoothing, the vertices are processed in the order in which they are stored in memory. For each vertex, the current quality of its ball of tetrahedra is computed. If the maximal quality (*i.e.* worst quality) is below a certain threshold $Q_{smoothing}$, nothing is done. Else, the optimal position of the vertex in its ball is computed, and the new maximal quality corresponding to the position is evaluated. If this maximal quality is below a certain factor $c_{smoothing}$ times the current maximal quality, then the vertex is moved to the optimal position and frozen for the rest of the pass. If the quality resulting from the optimal position is not satisfactory, other positions between the optimal position and the current position are tried. Note that if the resulting configuration is invalid, *i.e* a volume is negative, and the optimization is discarded.

1.1.5 Handling of boundaries

The mesh of the boundaries is moved rigidly, and the vertices are not usually moved on the surface (no displacement in the tangential directions). However, in some cases, such as when a body is moving very close to the bounding box of the domain, it can be useful to move the vertices of the bounding box as well. In this case, we can allow tangential displacement on the boundary. The risk of deforming a curved surface being too great, we only do this for planes aligned with the Cartesian frame. To do so, the displacements along the tangential axes are simply considered as new degrees of freedom. For instance, for a plane (x,y), the displacements along the x-axis and the y-axis are considered as degrees of freedom and are added to the elasticity system. The displacement along the z-axis is still set to 0, and thus is not added to the system.

1.1.6 Algorithm

The overall connectivity-change moving mesh algorithm is described in Algorithm 2, where the different phases described above are put together. When coupled with a flow solver (see Chapter 2), the flow solver is called after the optimization phase. In this algorithm, $\mathcal{H}$ stands for meshes, $\mathcal{S}$ for solutions, Q for quality (see Relation (1.2), $\mathbf{d}_{|\partial\Omega_h}$ for the displacement on the boundary, and $\mathbf{v}$ and $\mathbf{a}$ for speed and acceleration.

1.1.7 Choice of the different parameters for a robust algorithm

In the previous sections, several parameters have been mentioned. It must be stated that the success and robustness of the moving mesh algorithm is due to the choices made for these parameters. They were set experimentally, and several tests were run during this thesis to adjust them.

Algorithm 2 Connectivity-Change Moving Mesh Algorithm with Curved Trajectories

Input: $\mathcal{H}^0$, $\mathcal{S}^0$, Δt^0

While $(t < T^{end})$

1. Solve mesh deformation: compute vertices trajectories
   (a) $\Delta t^0 = \Delta t$
   (b) $\left\{\mathbf{d}_{|\partial\Omega_h}(t+\Delta t/2)\right\}$ = Compute boundary vertices displacement from current translation speed $\mathbf{v}^{body}$, rotation speed $\boldsymbol{\theta}^{body}$ and acceleration $\mathbf{a}^{body}$ for $[t, t+\Delta t/2]$
   $\mathbf{d}(t+\Delta t/2)$ = Solve elasticity system $\left(\mathbf{d}_{|\partial\Omega_h}(t+\Delta t/2),\, \Delta t/2\right)$
   (c) $\left\{\mathbf{d}_{|\partial\Omega_h}(t+\Delta t)\right\}$ = Compute boundary vertices displacement from current translation speed $\mathbf{v}^{body}$, rotation speed $\boldsymbol{\theta}^{body}$ and acceleration $\mathbf{a}^{body}$ for $[t, t+\Delta t]$
   $\mathbf{d}(t+\Delta t)$ = Solve elasticity system $\left(\mathbf{d}_{|\partial\Omega_h}(t+\Delta t),\, \Delta t\right)$
   (d) $\{\mathbf{v}, \mathbf{a}\}$ = Deduce inner vertices speed and acceleration from both displacements $\{\mathbf{d}(t+\Delta t/2), \mathbf{d}(t+\Delta t)\}$
   (e) If predicted mesh motion is invalid then $\Delta t = \Delta t/2^n$ and goto 1.
   Else $T^{els} = t + \Delta t$

2. Moving mesh stage, with mesh optimizations and solver solution
   While $(t < T^{els})$
   (a) δt^{solver} = Get solver time step $\left(\mathcal{H}^k, \mathcal{S}^k, \mathbf{v},\, CFL\right)$
   (b) $\delta t = \min(\delta t^{opt}, \delta t^{solver})$
   (c) If $t > t^{opt}$
      i. $\mathcal{H}^k$ = Swaps optimization $\left(\mathcal{H}^k, Q^{swap}_{target}\right)$
      ii. $\mathbf{v}^{opt}$ = Vertex smoothing $\left(\mathcal{H}^k, Q^{smoothing}_{target},\, Q_{max}\right)$
      iii. δt^{opt} = Get optimization time step $\left(\mathcal{H}^k, \mathbf{v},\, CFL^{geom}\right)$
      iv. $t^{opt} = t + \delta t^{opt}$
   (d) $\mathcal{S}^{k+1}$ = Solve Equation of State $\left(\mathcal{H}^k, \mathcal{S}^k, \delta t, \mathbf{v}, \mathbf{v}^{opt}\right)$
   (e) $\mathcal{H}^{k+1}$ = Move the mesh and update vertex speed $\left(\mathcal{H}^k, \delta t, \mathbf{v}, \mathbf{v}^{opt}, \mathbf{a}\right)$
   (f) Check mesh quality. If too distorted: solve new deformation problem or stop.
   (g) $t = t + \delta t$

   EndWhile

EndWhile

Concerning the elasticity problem, the material properties can be tuned to improve the solution of the problem, as explained in Sections 1.1.2. Notably, the Young modulus is set

to $\lambda = 1$ and the second Lamé's coefficient to $\nu = 0.48$. The local properties are adjusted proportionally to $|K|^{-\chi}$ with $\chi = 1$.

For the optimizations, two parameters come into play: the quality thresholds that triggers optimizations Q_{swap} and $Q_{smoothing}$, and the improvement factors c_{swap} and $c_{smoothing}$. The simulation of the optimizations is triggered if $Q^{current} > Q_{optim}$, and the optimization is actually performed if $Q^{resulting} < c_{optim}\ Q^{current}$. Q_{swap} and $Q_{smoothing}$ are usually set to 2. Concerning the improvement factors, $c_{smoothing}$ is set to 0.99: the smoothing has to increase the quality of at least one percent, whereas c_{swap} is set between 1.2 and 1.8 in 3D. This means that we allow swaps to degrade slightly the quality of one element, because this may help swapping other elements farther away. This is essential to the robustness of the algorithm in 3D, and forbidding any degradation leads to failure of the algorithm in many cases we tried. The side effect is that some elements are swapped in areas far from the moving bodies, and keep uselessly swinging between two states throughout the simulation. To prevent this, we tried to prevent swaps if the velocity of the vertices of the elements is small compared to the maximal velocity (close to the moving bodies). However, the gain in terms of number of swaps was negligible.

1.2 Mesh deformation with Inverse Distance Weighted method

So far, we have considered solving a linear-elasticity-like PDE to compute the mesh deformation. Another big family of methods for that step is interpolation methods, either direct or indirect. These methods compute the displacement of the inner vertices as an algebraic interpolation function of the displacement of the boundary vertices, *i.e.* the volume displacement field is seen as some kind of average of the boundary displacement. The most widespread of these method is known as Radial Basis Functions [de Boer 2007]. But this is an implicit methods, which means that the coefficients of the interpolation function are found solving a linear system the size of the number of vertices. It can thus be time consuming. Explicit interpolation methods on the other hand directly build the interpolation function, but the interpolation functions must be designed with care to handle large displacement and avoid creating false elements.

We chose to add such a direct interpolation method in our code, and to compare it with the elasticity analogy. The method that we chose, the Inverse Distance Weighted (IDW) method, is described at length in [Luke 2012]. In the following, I will recall this method, then I will explain how we plugged the code implementing it into our own code.

1.2.1 IDW method

In the IDW method, the displacement of a volume vertex is computed as a weighted average of the displacements of the boundary vertices, the weight being the inverse of the distance between the volume point and the boundary ones. In other words, the closer a boundary vertex is to the considered volume vertex, the stronger is its influence on the displacement of the volume vertex.

The displacement field in the volume mesh is defined as:

$$\mathbf{d}(\mathbf{r}) = \frac{\sum w_i(\mathbf{r})\mathbf{d}_i(\mathbf{r})}{\sum w_i(\mathbf{r})}\,, \tag{1.4}$$

where $\mathbf{r}$ is the coordinate vector in the original mesh, $\mathbf{d}(\mathbf{r})$ is the displacement field, $\mathbf{d}_i(\mathbf{r})$ is the nodal displacement field around boundary vertex i, and w_i is a weight coefficient computed as explained below. In our case, the nodal displacement field around a boundary vertex is simply the difference of the positions:

$$\mathbf{d}_i(\mathbf{r}) = \mathbf{x}_i(t + \Delta t) - \mathbf{x}_i(t)\,. \tag{1.5}$$

It can take several forms, whether rigid or non rigid displacements of the geometries are considered. Due to the way the algorithm is written, in [Luke 2012], the field is decomposed into a rotation matrix and and a displacement vector:

$$\mathbf{d}_i(\mathbf{r}) = M_i\mathbf{r} + b_i - \mathbf{r}\,, \tag{1.6}$$

where M_i is a rotation matrix that is interpolated from the displacement of the neighboring boundary vertices, and b_i is the displacement vector. In the case of a rigid body, M_i is its rotation matrix and b_i is the translation term.

The weight corresponding to the ith boundary vertex is computed as a function of the inverse of the distance between the volume vertex and the boundary vertex:

$$w_i(\mathbf{r}) = A_i * \left[\left(\frac{L_{def}}{\|\mathbf{r} - \mathbf{r}_i\|} \right)^a + \left(\frac{\alpha L_{def}}{\|\mathbf{r} - \mathbf{r}_i\|} \right)^b \right], \tag{1.7}$$

where $\mathbf{r}_i$ is the position of the boundary vertex, A_i is the area around vertex i, L_{def} is a reference length of the deformation domain, and a and b are given coefficients. We use $a = 3$ et $b = 5$ in 3D, and $a = 2$ et $b = 4$ in 2D. L_{def} can be computed automatically, but in our case it is a parameter given by the user. The α factor controls the relative weights of nearby surface vertices and farther ones, and is computed from a mean displacement:

$$\mathbf{d}_{mean} = \sum_{i=1}^{N} a_i * \mathbf{d}_i(\mathbf{r}_i)\,, \tag{1.8}$$

where $a_i = \frac{A_i}{\sum_{j=1}^{N} A_j}$ is the normalized weight of vertex i. α is then written:

$$\alpha = \frac{\eta}{L_{def}} \max_{i=1}^{N} \|\mathbf{d}_i(\mathbf{r}_i) - \mathbf{d}_{mean}\|\,. \tag{1.9}$$

η is usually set to 5, and α must be greater than 0.1. Further discussion on the choice of the parameters can be found in [Luke 2012].

If the above method is implemented straightforwardly, it results in a quadratic algorithm - or more precisely an algorithm $O(n * m)$ where n is the number of volume vertices and m the number of boundary vertices. In 2D, the number of surface vertices is usually small enough for a naive algorithm to run quite fast. However in 3D, the number of boundary vertices is much larger, and the cost of the naive algorithm is prohibitive. Fortunately, the solution of the mesh deformation problem does not need to be very precise - which is even more true with our connectivity-change moving mesh algorithm - and thus approximations can be made that improve the efficiency of the method. More precisely, when computing the displacement of a

volume vertex, the impact of boundary vertices far from it is limited, and thus their contribution can be simplified.

Based on that observation, a tree-code algorithm was proposed in [Luke 2012], similar to those used for the N-body problem. This is possible because the form of $d(r)$ is similar to gravitational potentials. The principle of the algorithm is to build a tree of the boundary mesh, in the form of a KD-tree, where the leafs are the boundary vertices, and the nodes are groups of neighboring vertices, more or less large depending the depth in the tree. For each set of nodes, the partial sum is approximated by the sum of 4 optimal quad-points. For each volume point, this tree is then descended recursively: the farther the vertex is from a zone of the surface, the less we descend the tree and only the approximate contribution of a node is used. An error evaluation is used to decide when to stop descending the tree.

With this algorithm, the displacement of vertices close to boundaries is very close to the one given by the exact formulation, and the error is somewhat larger for points far from boundaries, where no precise displacement is necessary. Besides being much faster than the naive algorithm, this algorithm preserves the property of being easily parallelizable: once the tree is built, the computation of the displacement of one vertex is independent of the displacement of the other vertices, and they can thus be run in parallel.

Remark 2. *Since the method described in [Luke 2012] is plugged into our algorithm, it is important to notice that our version has some differences with the version described in [Luke 2012]. Notably, the deformation field d is computed between the beginning and the end of a mesh deformation step, while it is computed between the current time and the beginning of the simulation in [Luke 2012]. In our version, the parameter L_{def} needs to be specified by the user. The parameters a, b and η are also specified by the user, although no major difference has been observed when changing them. The impact of these values may appear when dealing with viscous boundary layer meshes.*

The main advantages of this method are the easy parallelization, and the handling of boundary layers: there is no extra-cost to compute the displacement of thin boundary layer vertices, whereas to preserve the structure of the layers, the elasticity approach requires creating very small and rigid elements, resulting in very stiff and hard to solve FEM systems.

1.2.2 Implementation

An MPI code implementing the optimized IDW method was written at Mississippi State University by Prof. Ed. Luke and Eric Collins: `gridMover`, and they allowed us to use this code and plug it into our in-house code `Wolf`.

The code implements the method described in [Luke 2012] rather straightforwardly. Its input is a mesh and a displacement of its boundary, then it computes the weights and rotors for the boundary surface nodes, the reference length and the alpha factor. Then, the KD-tree of the surface mesh is built and filled. Finally, the displacement of the volume vertices is computed descending the tree. The code was parallelized using MPI: the KD-tree is built in parallel, then sent to all the processors, and the volume vertices are split into chunks, each chunk being run independantly on the available processors.

The code was plugged into our code. Our goal was to adapt the method to fit as much as possible into our moving mesh algorithm, so we call the entire `gridMover` code instead of calling our elasticity solution routines. More precisely, everything is done as usual, including the computation of the displacement of the boundary mesh, except that instead of calling the routine that builds the elasticity system and solves it, we call the `gridMover` main routine: Algorithm 2 remains unchanged except that "elasticity system" is replaced by "IDW problem". To do so, several changes to `gridMover` were necessary. At the beginning of the `gridMover` code, we had to add a converter from `Wolf` data structures into `gridMover` structures. The way the boundary displacement is prescribed is modified too. To facilitate the compilation, the `gridMover` code is put in an independent `Wolf` module, and is compiled as an independent dynamic library, that is called from the core `Wolf` code. The main difficulty is that `gridMover` is a MPI code, so `Wolf` had to be converted into an MPI code too. The choice that was made was to run most `Wolf` routines on one processor, which required a delicate handling of data structures

1.3 Examples

We now consider several examples of moving mesh simulations, and study the performance of our connectivity-change moving mesh algorithm, both with the elasticity-like and the IDW methods for the mesh deformation step. The next examples are purely moving mesh simulations and we will focus our analysis on mesh quality criteria. A comparison of the two methods for real ALE simulations will be provided in section 3.2.

In particular, we show that the moving mesh algorithm (MMA) applies successfully to rigid body with large shearing and deformable bodies. In this section, we only address the case of 3D.

1.3.1 Rigid-body examples

Rotation and translation case

The first example seems to be very simple, a cube is moved in translation and rotation in a box, but the strong rotation and acceleration of the cube makes it less easy than expected. The dimension of the box are $15 \times 10 \times 10$, and the side of the cube is 1. The accelerated translation and rotation vectors are:

$$\mathbf{v} = \begin{pmatrix} 0.2\ t \\ 0 \\ 0.05\ t \end{pmatrix} \text{ and } \mathbf{r} = \begin{pmatrix} -10 \\ 30 \\ 10 \end{pmatrix}.$$

The initial mesh has $40,000$ vertices and $220,000$ tetrahedra. The simulation is run until $T^{end} = 10$. For both mesh deformation methods, 10 mesh deformation steps are set, *i.e.* $\Delta t = 1$, and we set $CFL^{geom} = 8$.

Both methods result in excellent quality meshes. Statistics can be found in Table 1.1. Images of the initial and final meshes can be found in Fig. 1.2. However, the IDW method has required significantly more mesh deformation steps than the elasticity-based method, respectively 20 and

	# mesh deform.	# optim.	Q_{end}^{mean}	$1 < Q_{end} < 2$	$Q_{overall}^{worst}$	# swaps
Elasticity	10	53	1.4	98.9%	5.1	97,400
IDW	20	75	1.4	99.2%	6.3	113,042

Table 1.1: Accelerated cube test case. Final mesh statistics and total number of swaps with the changing-connectivity MMA.

10, due to automatic adjustment of the mesh deformation time step when the quality in the future becomes bad. Without these adjustments, the simulation cannot be run until the end, which highlights their importance in some cases. A comparison of the solutions of the mesh deformation problem, see Fig. 1.2 (middle), explains why more solutions of that problem are required with the IDW method. Indeed, the IDW method tends to only move vertices close to the moving body, whereas the elasticity makes more room for the cube ahead of it.

Interpenetrating cylinders

This example is a challenging academic test case with rigid bodies, and has been presented in [Alauzet 2014a]. Two cylinders interpenetrate each other and there is only one layer of elements between both cylinders, *i.e.* internal edges connect both cylinders. The geometry of the domain is shown in Fig. 1.3. To make this test case more complicated, we add rotation to the top cylinder:

$$\theta^{top} = (0,0,5)\,, \mathbf{v}^{top} = (0,0,-0.1) \quad \text{and} \quad \theta^{bottom} = (0,0,0)\,, \mathbf{v}^{bottom} = (0,0,0.1)\,.$$

The initial mesh is composed of 34,567 vertices and 188,847 tetrahedra. The simulation is run until $T^{end} = 100$.

A shear layer appears between the two cylinders, that is only one element wide. Therefore, the swapping optimization has just a little of freedom to act. Moreover, it is obvious that too large Δt will lead to an invalid mesh: such a simulation requires a relatively large number of mesh deformation solutions. For both mesh deformation methods, 50 mesh deformation steps are set, *i.e.* $\Delta t = 2$, and we set $CFL^{geom} = 2$.

The changing-connectivity MMA proves to be extremely robust in both cases, by keeping the mesh worst element quality below 100 and by ensuring a final excellent mean quality of 1.5. Because of the shearing, a large number of swaps - around 1 million - have been performed. For that test case, results with both mesh deformation methods are very similar. However, Fig. 1.4 (center) points out a major difference between elasticity and IDW: elasticity creates rotational displacements in the mesh above and under the cylinder, that push vertices in front, attract vertices behind the moving inner cylinder, and thus move vertices to empty areas instead of crushing them, while the IDW method tends to produce straighter displacements. Table 1.2 provides a comparison of the MMA with the two mesh deformation methods.

Shuttle

The last rigid example is the Space Shuttle ejecting its two Solid Rocket Boosters (SRB) after burnout. SRBs separation occurs just after 2 min into the flight, at an altitude close to 50 km

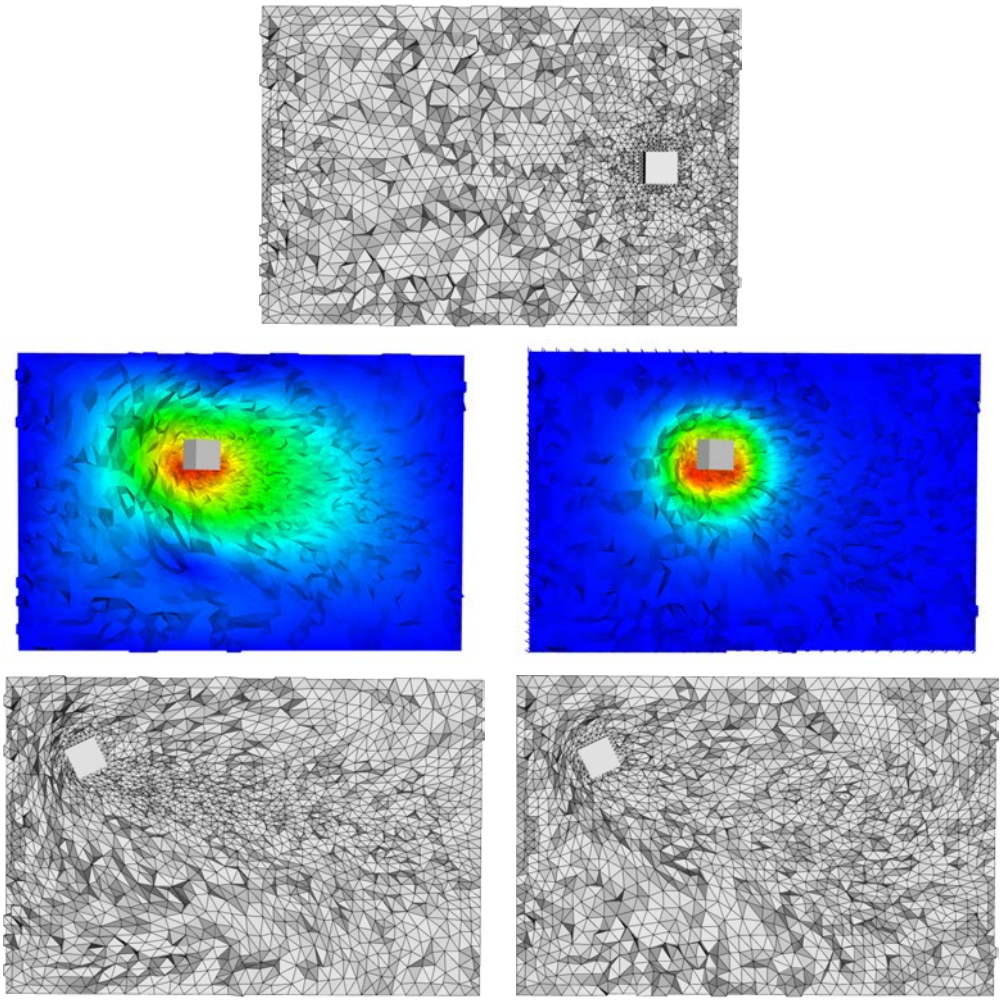

Figure 1.2: Accelerated cube test case. Initial mesh (top), mesh deformation solution at $t = 8$ (center) and final meshes (bottom), with the elasticity-based method (left) and the IDW method (right).

and a Mach number above 4. The objective of such a numerical simulation is to understand the separation dynamics of the SRBs attached to the space shuttle, notably to know if the separating booster motor plume will hit the orbiter wind-shield.

The difficulty of this example is the very complex geometry, see Fig. 1.5, and the large size of the mesh, since we consider an isotropic mesh with $996,599$ vertices and $5,505,036$ tetrahedra. The two SRBs have a parabolic trajectory and they rotate at the same time. The movement of the SRBs together with meshes can be seen in Fig. 1.6.

The simulation is run until $T^{end} = 25$ and Δt is set to 0.1. The density of the mesh in which the SRBs evolve is a lot greater at the beginning than at the end, so the CFL^{geom} parameter

	# mesh deform.	# optim.	Q_{end}^{mean}	$1 < Q_{end} < 2$	$Q_{overall}^{worst}$	# swaps
Elasticity	50	429	1.5	97.1%	79	852,514
IDW	50	570	1.4	98.2%	30	1,408,680

Table 1.2: Interpenetrating cylinders test case. Final mesh statistics and total number of swaps with the changing-connectivity MMA.

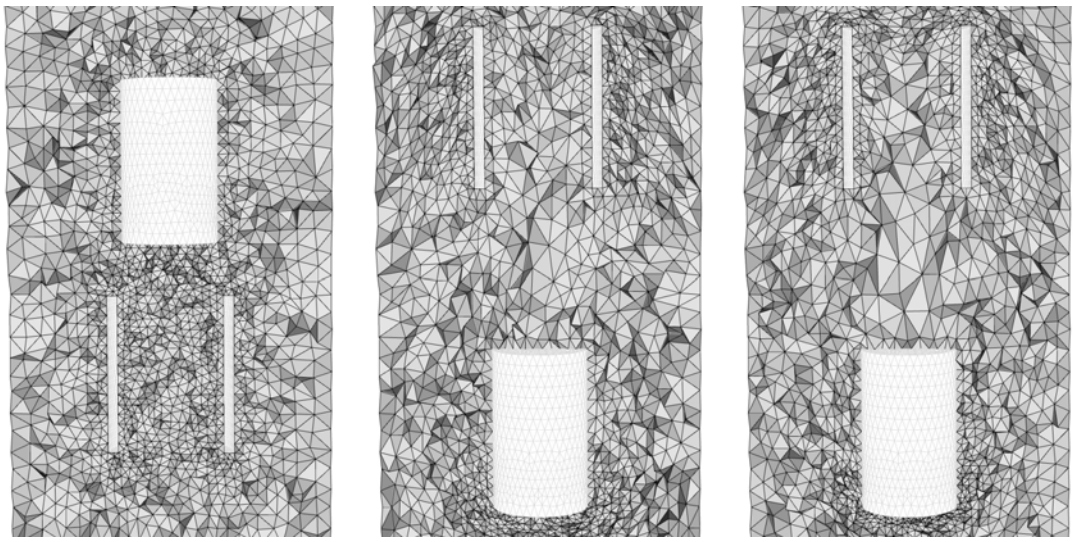

Figure 1.3: Interpenetrating cylinders test case. Initial (left) and final meshes with the elast elasticity-based (center) and IDW (right) methods.

is changed throughout the simulation to speed it up:

$$CFL^{geom} = \begin{cases} 5 & \text{if} \quad t < 3 \\ 10 & \text{if} \quad 3 \leq t < 10 \\ 40 & \text{if} \quad t \geq 10 \end{cases} .$$

Both methods preserve an excellent mesh quality throughout the simulation, and the final statistics for both are equivalent. 250 elasticity solutions were required by the IDW method, and 251 by the elasticity-based method: in the latter case, after the first elasticity solution, the expected future quality of the mesh moved with the prescribed displacement was too bad, so a

	# mesh deform.	# optim.	Q_{end}^{mean}	$1 < Q_{end} < 2$	$Q_{overall}^{worst}$	# swaps
Elasticity	251	943	1.4	99.8%	6.8	4,784,457
IDW	250	957	1.4	99.8%	6.8	4,550,417

Table 1.3: Shuttle test case. Final mesh statistics and total number of swaps with the changing-connectivity MMA.

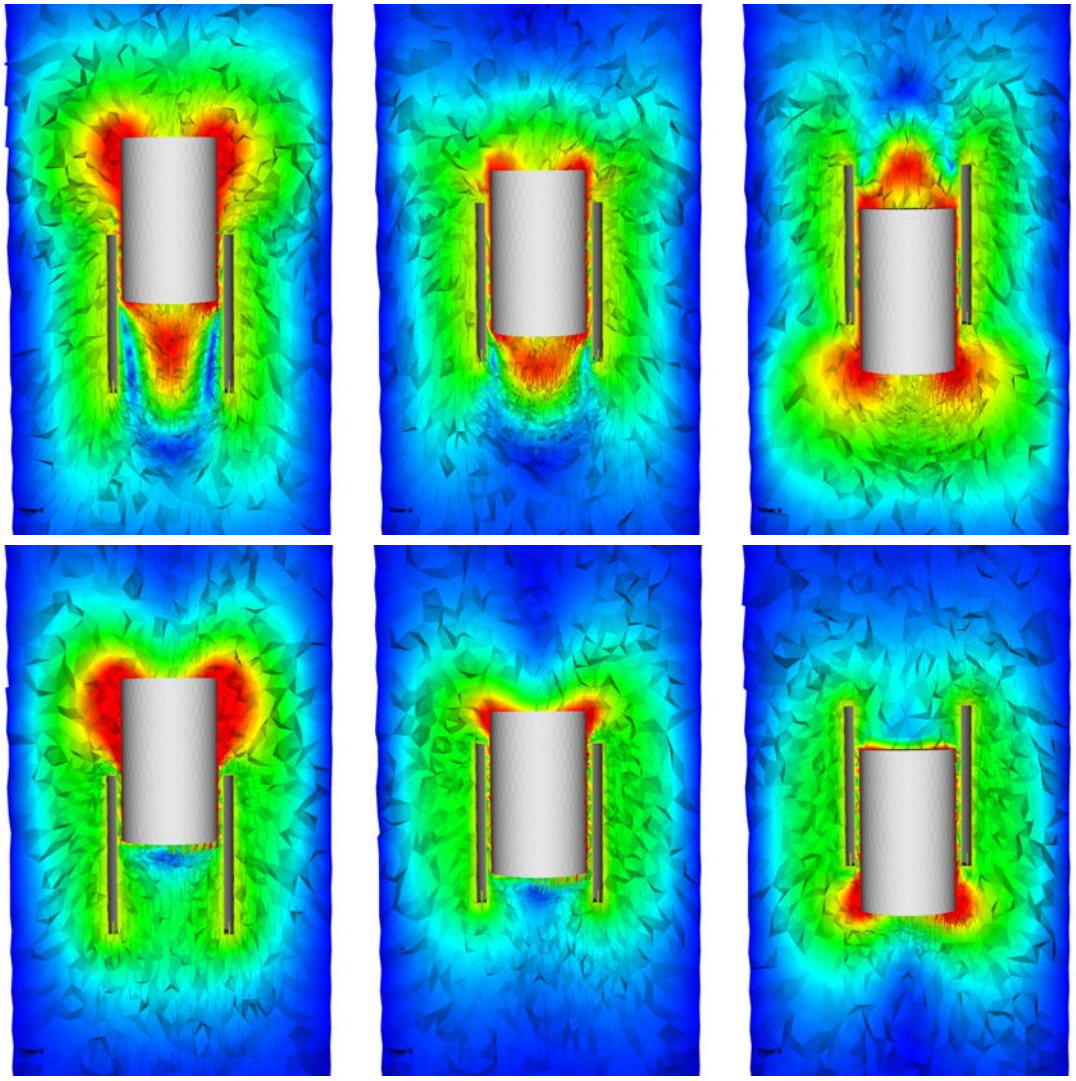

Figure 1.4: Interpenetrating cylinders test case. Evolution of the mesh and solution of the mesh deformation problem with the elasticity-based (top) and IDW (bottom) mesh deformation methods at times 12, 18 and 25.

new elasticity solution was computed for a smaller time step. This way to robustify the code proves its efficiency, since it costs only one elasticity solution and preserves the mesh quality. In both cases, the final average quality is 1.4 which is excellent, and around 4.5 millions of swaps were performed. The meshes and mesh deformation solutions at time $t = 12$ are shown in Fig. 1.7. The palette is the same in both cases, and we can see that the elasticity-based method moves vertices much farther away from the moving bodies than the IDW method. This results in smoothing the trail of the SRBs, while it is very visible above them with the IDW method. Both methods manage to preserve the layers of small elements close to the SRBs, which would be important in a physical computation context. Finally, in terms of CPU, where a difference may be expected, the simulation time is equivalent in both cases, around 2 hours, with the

elasticity code run serially and the IDW code run on 20 cores. The mesh deformation solution only accounts for a small part of the overall algorithm.

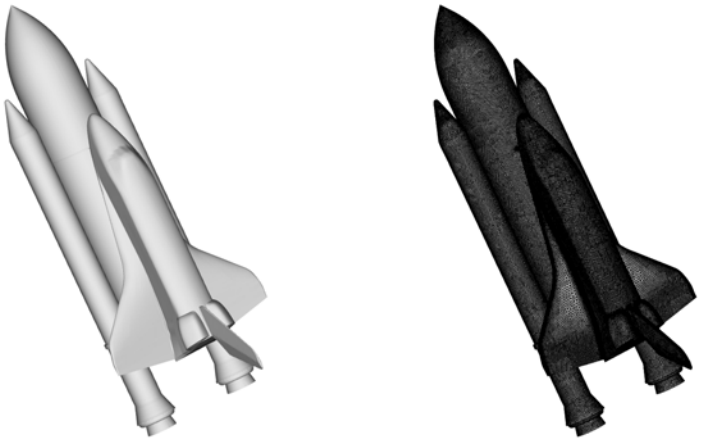

Figure 1.5: Shuttle test case. Geometry and surface mesh in initial state.

1.3.2 Deformable-body examples

Squeezing can

The next test case deals with deformable geometries. It represents a can which is squeezed. The can is a cylinder of length 5 units and diameter 2 units inside a spherical domain of radius 10 units. The can is centered at the origin and is aligned along the z-axis. The simulation runs up to $T^{end} = 1$ and the temporal deformation function is given by:

$$\text{if } |z(0)| < 2 \text{ then } \mathbf{x}(t) = \begin{cases} x(t) &= x(0) \\ y(t) &= \left(\alpha + (1-\alpha)(1-t^2)\right) y(0) \\ z(t) &= z(0) \end{cases}$$

$$\text{with } \alpha = \frac{1}{2.2}\left(1.2 - \cos\left(\frac{\pi}{2}|z(0)|\right)\right) .$$

The can deformation is shown in Fig. 1.8 and 1.9. For this case, the moving mesh simulation is done for the inside and the outside meshes of the can geometry. The inside case mesh is composed of $7,018$ vertices and $36,407$ tetrahedra and the outside mesh contains $43,631$ vertices and $250,152$ tetrahedra. Two mesh deformation steps are initially prescribed for the outside case and four for the inside one. The other simulations parameters are exactly the same for both cases. CFL^{geom} is fixed to 1. Statistics for both cases are given in Table 1.4.

The outside mesh case is known to be problematic for spring-analogy methods because spring stiffness is proportional to the element size, *i.e.,* the smaller an element is, the stiffer it is. Therefore, when the can is squeezed, elements close to the can are elongated in one direction leading to a poor quality mesh for numerical simulation. No vertices come to fill the freed space. As illustrated in Fig. 1.8, the changing-connectivity MMA coupled with both mesh deformation

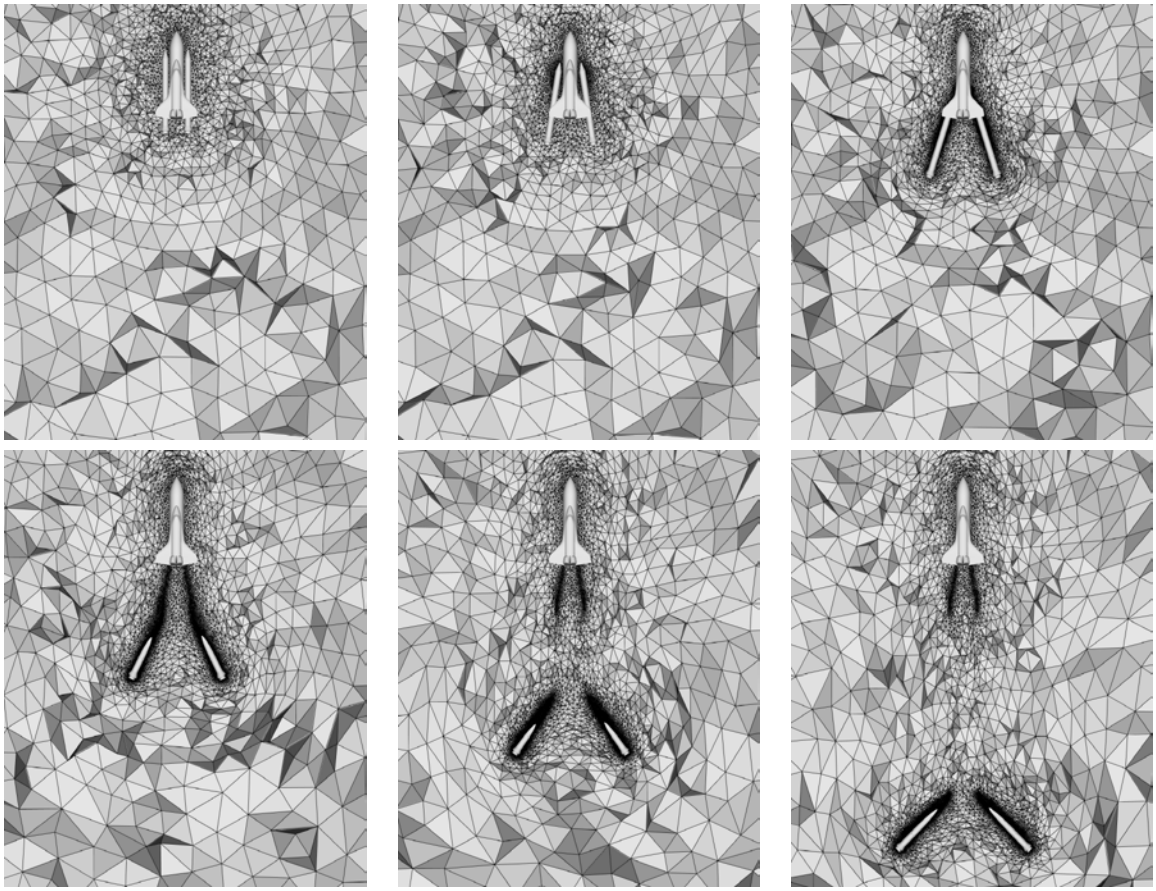

Figure 1.6: Shuttle test case. Snapshots of the ejection of the SRBs at times 0, 5, 10, 15, 20 and 25.

methods behaves properly. In the case of the elasticity-based method, the elasticity solutions move vertices to fill the freed space. The same phenomenon can be observed with the IDW method. In fact, this test case presents no difficulty: it is fast, few swaps are performed and the quality is very good. The elements of worst quality are connected to the squeezed can and are located where the can is crushed. If swaps were performed on the surface, their quality would be improved.

The inside mesh case is also successfully achieved, cf. Fig. 1.9 (bottom). We clearly see that the elasticity methods has moved vertices away from the squeezed region while keeping an excellent mesh quality, whereas the IDW tends to let them crush. For this case, the swap optimization are essential to preserve the quality of the mesh all along the movement. The mesh quality behavior for the inside case is similar to that for the outside one.

Bending beam

This test case has been proposed in [Luke 2012]. It is a general deformation test on a three-dimensional bending beam. The beam is 8 units long, 2 units wide and 0.1 unit thick inside a

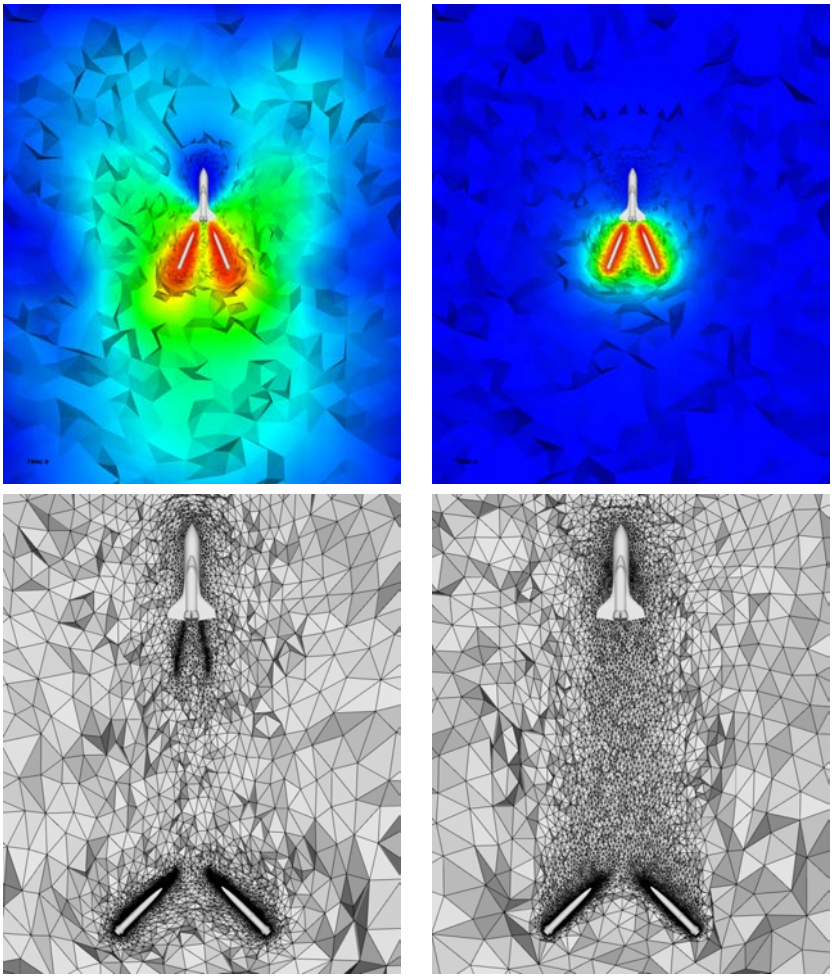

Figure 1.7: Shuttle test case. Mesh deformation solution at time 12 (top) and final meshes at time 25 (bottom) for the elasticity-based method (left) and the IDW method (right).

spherical domain of radius 25 units. The beam is deformed such that the beam center-line is mapped onto a circular arc:

$$\mathbf{x}(t) = \begin{cases} x(t) &= (R - z(0))\sin(\theta) \\ y(t) &= y(0) \\ z(t) &= z(0) + (R - z(0))(1 - \cos(\theta)) \end{cases}$$

$$\text{with } \theta = \frac{x(0)}{R} \text{ and } R = R_{\min} + (R_{\max} - R_{\min})\frac{\mathrm{e}^{-\kappa t} - \mathrm{e}^{-\kappa}}{1 - \mathrm{e}^{-\kappa}}\,.$$

where $R_{\min}$ and $R_{\max}$ are the radii of curvature at $t = 0$ and $t = 1$, respectively. κ is a parameter that controls how rapidly the radius and center of curvature move from the initial to final values. For this test case, $R_{\min} = 2$, $R_{\max} = 500$ and $\kappa = 10$. The initial mesh is composed of $58,315$ vertices and $322,971$ tetrahedra. We set $\Delta t = 0.2$ and $CFL^{geom} = 1$.

Again, this large deformation problem is solved without any difficulty, requiring only a few

	# mesh deform.	# optim.	Q_{end}^{mean}	$1 < Q_{end} < 2$	$Q_{overall}^{worst}$	# swaps
Elasticity (out.)	2	8	1.3	99.5%	12.3	3,360
IDW (out.)	2	14	1.31	99.7%	11	3,950
Elasticity (in.)	6	27	1.5	93.0%	11	16.255
IDW (in.)	4	27	1.48	92.5%	11	11,940

Table 1.4: Squeezing can test case: outside (top) and inside (bottom) of the can. Final mesh statistics and total number of swaps with the changing-connectivity MMA.

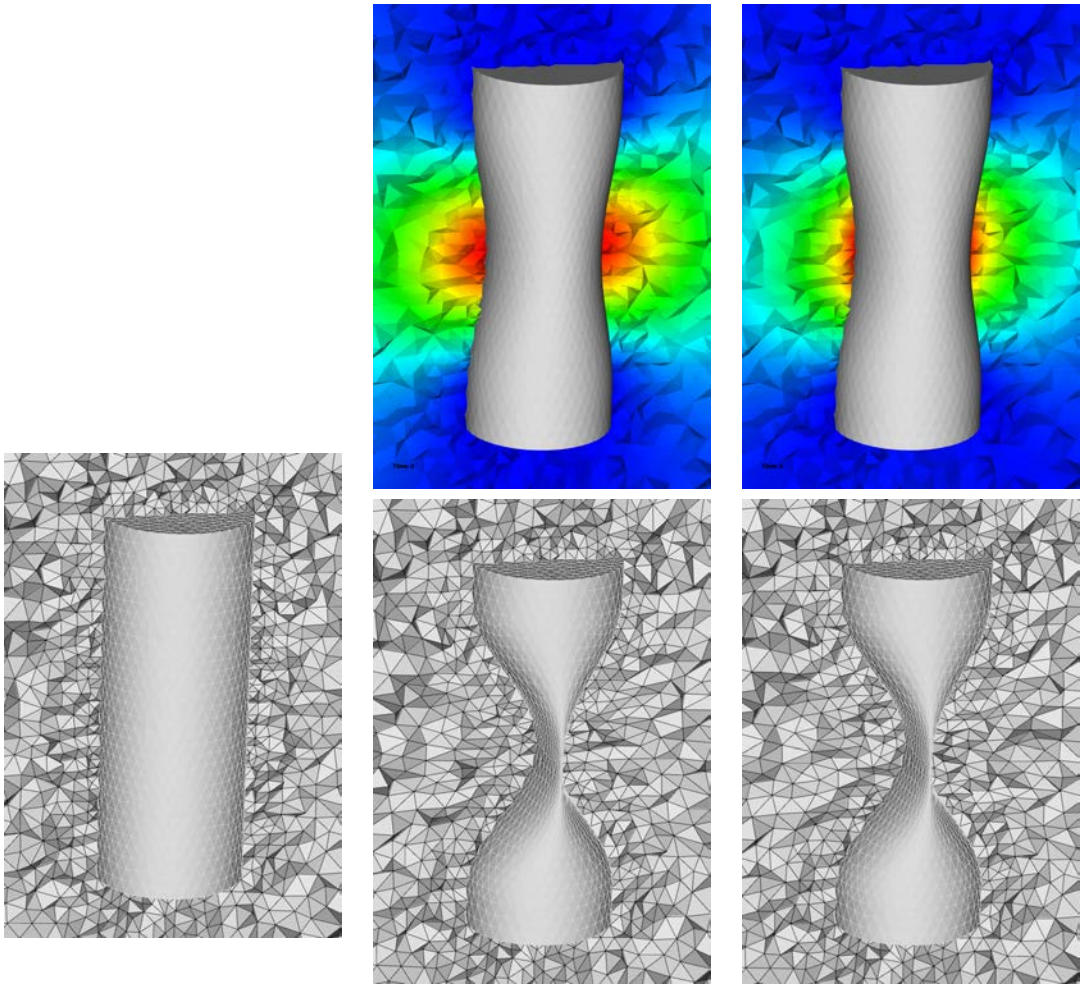

Figure 1.8: Squeezing can test cases: outside mesh. Initial mesh (left), displacement given by the elasticity-based (top left) and IDW (top right) mesh deformation problems at time 0.5, and final meshes with elasticity (bottom left) and (bottom right) IDW.

mesh deformation steps, the final mesh is excellent as shown in Table 1.5 and no skewed element appears inside the mesh. There again, we can see in Fig. 1.10 that the elasticity-based method tends to move points farther from the body than the IDW method, thus requiring significantly

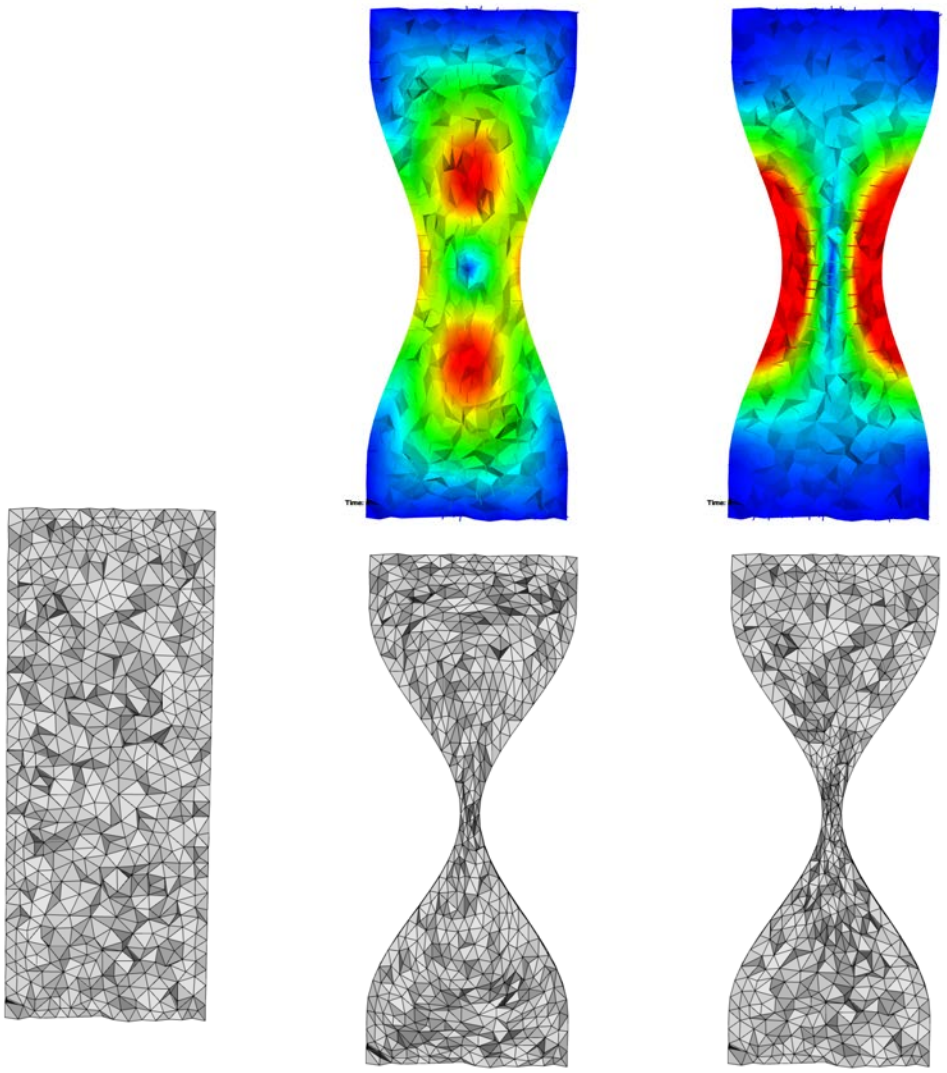

Figure 1.9: Squeezing can test cases, inside mesh. Initial mesh (left), displacement given by the elasticity-based (top left) and IDW (top right) mesh deformation problems at times 0.75, and final meshes with elasticity (bottom left) and (bottom right) IDW.

more swaps. However, the performances of the elasticity-based and IDW methods are very close, cf. Table 1.5.

	# mesh deform.	# optim.	Q_{end}^{mean}	$1 < Q_{end} < 2$	$Q_{overall}^{worst}$	# swaps
Elasticity	5	211	1.3	99.9%	4.4	479, 182
IDW	4	40	1.3	99.8%	5.9	149, 536

Table 1.5: Bending beam test case. Final mesh statistics and total number of swaps with the changing-connectivity MMA.

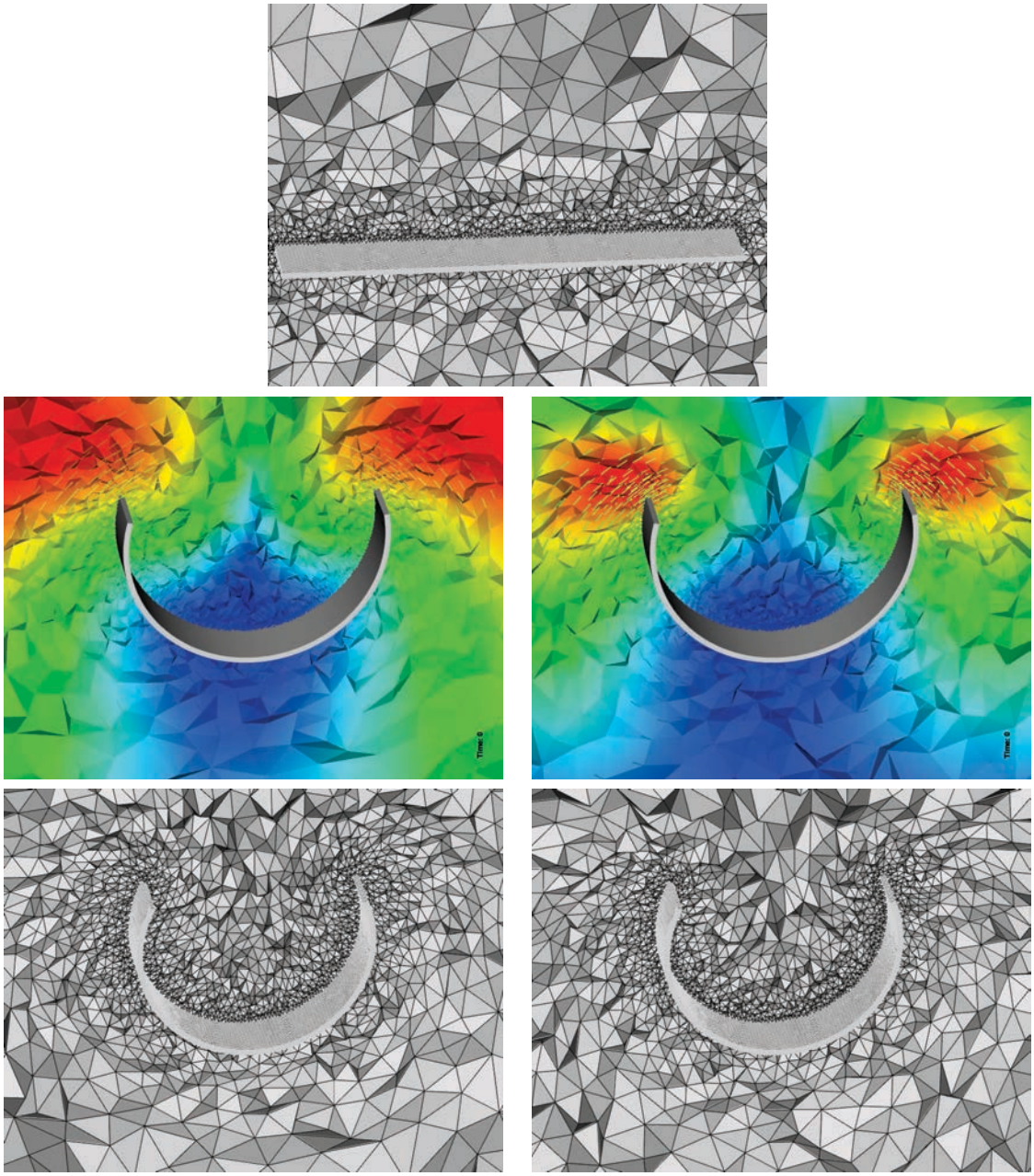

Figure 1.10: Bending beam test case. Top: initial mesh. Center: displacement given by the elasticity-based (left) and IDW (right) mesh deformation problems at time 0.8. Bottom: final meshes with both methods.

1.4 Boundary layers for deformable geometries

Dealing with boundary layer (BL) meshes for rigid bodies is rather easy: the whole mesh layers are rigidified and moved together with the body. However, it is a lot more difficult with deformable bodies, since the mesh structured layers must both follow the deformation of the

body and keep their structure. It was *a priori* not certain that the movement provided by the elasticity or IDW step would be precise enough to preserve that structure. In the sequel we demonstrate that both methods can move one thick layer with a good accuracy, and that such layers can also be moved correctly with the elasticity method.

1.4.1 One boundary layer

For this case, we consider the bending beam studied previously, to which we add one thin structured mesh layer of height a quarter of the height of the beam, see Fig. 1.11. The movement of the beam is the same and no swap or smoothing optimizations are done in the BL. At the end of the simulation, we observe in Fig. 1.11 that the structured aspect of the BL has been well preserved. In Table 1.6, we gather quality indicators depending on different moving mesh parameters. We consider the variation of the distance of the layer vertices to the body, that is expected to remain constant in ideal cases. We can see that taking acceleration into account in the trajectories is very important: indeed, as the movement of the beam is accelerated, the layer under the beam tends to thicken while the layer above it tends to be crushed. We can also see that less IDW steps are required to reach an equivalent mean quality, but the maximal distance variation is less good.

	10 elast. Q.	50 elast. L.	50 elast. Q.	10 IDW Q.
Mean relative $\delta \mathrm{d}x$ (in %)	27	9.9	5.7	6.4
Max relative $\delta \mathrm{d}x$ (in %)	143	66	19	60

Table 1.6: Bending beam with one BL. Mean and maximal relative distance variations $\delta \mathrm{d}x$, in the case of 10 elasticity solutions, 50 elasticity solutions with linear (L.) trajectories (without taking acceleration into account), 50 elasticity solutions with quadratic (Q.) trajectories.

This result is very promising, because the structured aspect of the BL mesh is well preserved. This lets us consider a clever strategy to move BL meshes with deformable geometries.

1.4.2 Several boundary layers

The elasticity mesh deformation method is also actually capable of moving several thinner BL, provided the mesh deformation time step is small enough to handle the displacement of the smallest elements against the body. The simulation was run with a boundary layer containing 10 thin layers (the smallest height is 0.0017 units): the structure inside the layer is still well preserved all along the movement, and the quality of the layer looks good even at the end of the deformation of the beam. Close-ups on the BL meshes are show in Fig. 1.12. We were not able to move these layers with the IDW mesh deformation, which needs to be further investigated.

1.5 Large volume variations

So far, we have only dealt with the case of moving geometries with no or small volume variation. In these case, we have demonstrated that our moving mesh algorithm preserves the quality of

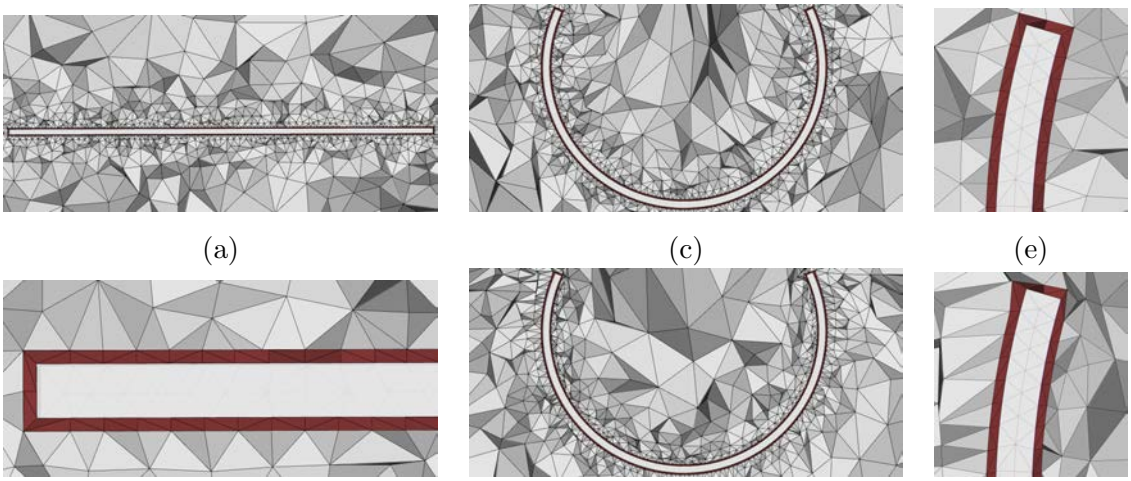

Figure 1.11: One BL test case. Initial mesh (a) and close up on the BL (b), final mesh with elasticity (c) and IDW (d), and close-ups on the final BL meshes with elasticity (e) and IDW (f).

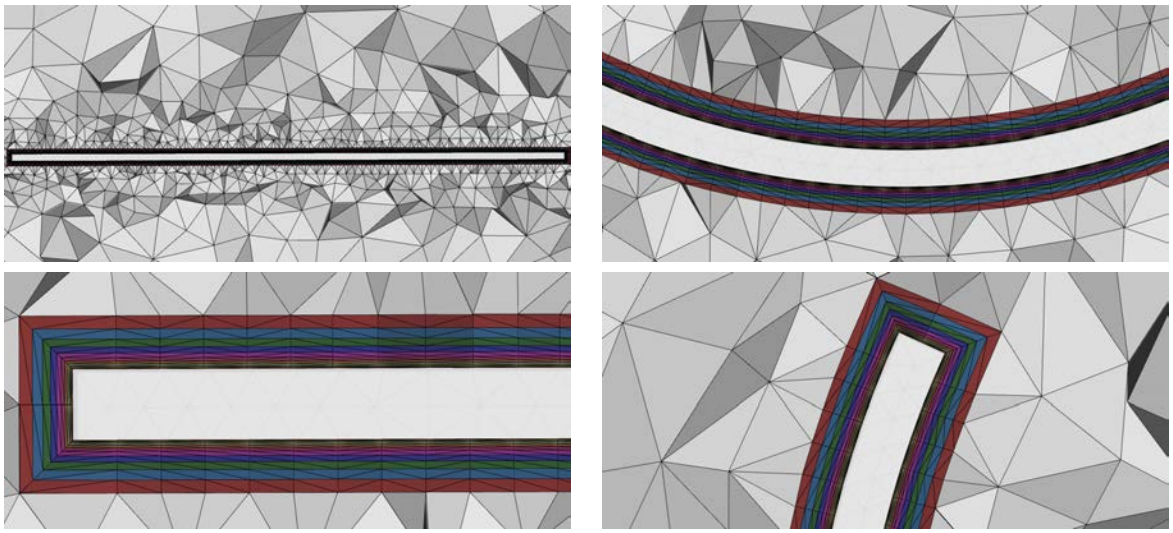

Figure 1.12: Ten BL test case. Initial mesh (left) and close-ups on the final BL mesh (right), with the elasticity-based mesh deformation method.

the mesh without vertex addition, and in particular preserves the size of the elements. However, in the case of large volume variation, the deformation of the mesh is going to result in large variation of the sizes of the elements: if the volume is greatly increased, some of the elements (at least) are going to have a volume that increases greatly too. This is illustrated in Fig. 1.13, where a tube is expanded by a factor 7: in one case, only the vertices of the side of the tube are moved, in the other case, all the vertices inside the tube are moved proportionally to their initial coordinates. In both cases, part or all of the elements are stretched a lot. In some cases,

it might not be a problem (if the accuracy required in the expanding areas is low), but if we want to preserve a certain element size throughout the simulation, some specific strategies need to be considered.

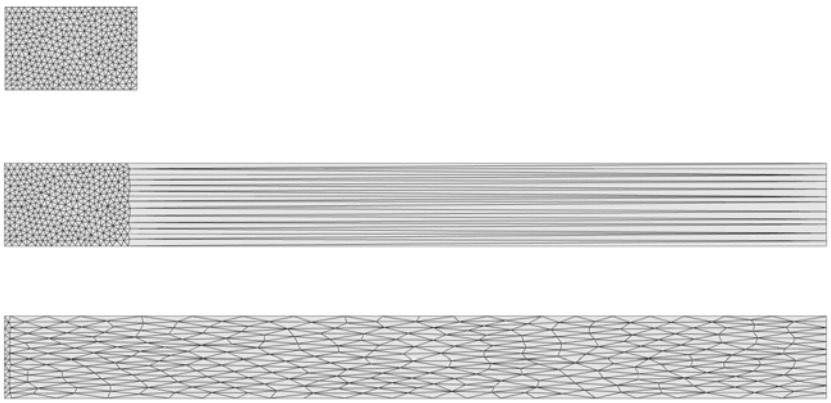

Figure 1.13: Tube in expansion. Top, the initial mesh. Center and bottom, meshes moved, respectively moving only one side of the tube, and moving all the vertices proportionally to their coordinates.

The obvious choice is to mesh the expanding domain with very small elements at the beginning of the simulation, so that they reach the desired size once expanded. However, several issues arise from this simplistic strategy. The first one is that the size of the elements can be very small at the beginning of the simulation if the volume is to vary greatly. This is a problem from the solver point of view, since smaller elements usually require smaller time steps, which may slow the solver down a lot if the elements are too small. The second question is: if the elements do not expand uniformly, what is the optimal size they should be given at the beginning to reach to desired size at the end ? And finally, this strategy is only possible if the displacement of the boundaries is know in advance. In the case of a FSI simulation, where the displacement is *a priori* an unknown of the problem, we cannot predict the volume variation and thus the size of the elements required.

For a better result, the problem has to be seen as a mesh adaptation problem: given a metric at a certain time of the simulation, and given a displacement of the mesh vertices to this time, what should the metric at the beginning of the simulation be, so that the mesh is adapted at time t ? This question will be addressed at length in Part II of this thesis. In what follows, we expose a moving mesh strategy for large volume deformation on one example. This strategy does not use the formalism of metric based mesh adaptation, but it answers to the problematic of controlling the sizes of the mesh throughout the deformation.

1.5.1 Engine piston

An example of such large volume variation simulation is the case of an engine piston. This test case was provided by IFP Energies nouvelles.

The geometry is represented in Fig. 1.14, with the piston in compressed and expanded positions. The piston is a cylindrical chamber whose bottom end can translate downward. The top of the chamber is closed by a mechanical piece with pipes and valves, forming on the left ($x < 0$) the admission system and on the right ($x > 0$) the exhaust system.

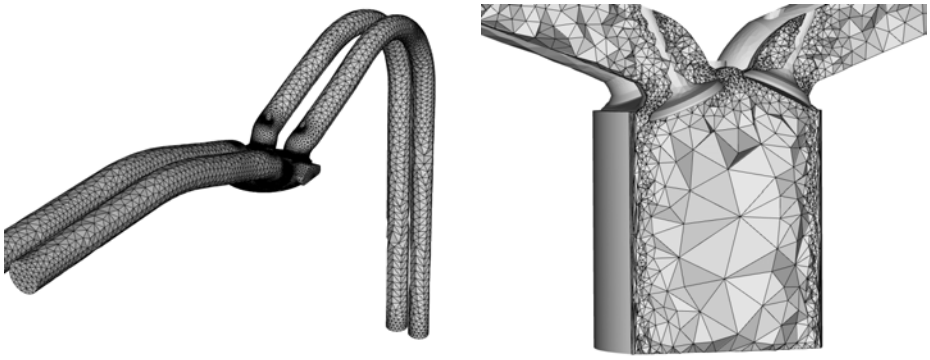

Figure 1.14: Engine piston. Left: provided surface mesh with the piston in upward position (time $t = 0$). Right: cut in the mesh with the piston is in extended position. The valves are visible, the one of the left is half open and the one on the right is closed.

Definition of the geometry displacement

The displacement is given *a priori* for each moving boundary. There are three different displacements: for the piston itself, for the admission valves and for the exhaust valves. The displacement is given as a function of the angle of crank, a given rotation speed allowing to convert times into angles. The initial angle is 2°. Note that, as the displacement of the geometry was prescribed, there seems to be a slight shifting between the valves and the piston.

The height z_{piston} of the bottom side of the piston is given by:

$$\begin{aligned} h &= 0.5\ S\ \sin(\theta) \\ s &= \sqrt{C^2 - h^2} \\ z &= 0.5\ S\ \cos(\theta) + s + 0.5\ S - C \\ z_{piston} &= z - S + z_0\,, \end{aligned} \tag{1.10}$$

where θ is the crank angle in radians, $S = 83.5mm$ is the stroke, $C = 144mm$ is the connecting rod length and z_{piston} is the actual height of the piston. $z_0 = 0.032305$ is a constant used to ensure the consistency between the given geometry and the formulas. All the lengths are given in millimeters.

The position of the valves on their axis is tabulated, the tables being provided by IFPEN. The position at a time t is then interpolated linearly between the two closest tabulated values. Since the axis of the valves was not provided, it had to be computed numerically. Note that the movement is periodic, but is not perfectly synchronous as for real engines.

We consider a four-stroke engine, so a full engine cycle corresponds to a 720° crank angle run: the piston is fully expanded and compressed twice, while the admission and exhaust valves

are opened and closed once for each pair: first the admission valves open to let gas in while the piston is expanded, then the valves are closed, the piston is compressed, when the piston is compressed an explosion makes it expand itself once more, and then the exhaust valves open to let the gas out.

Note that, in the three cases, a rigid displacement is given for the bottom of the object, but this object is connected to a fixed boundary, and no information is given for the displacement of these points. To avoid having expanding elements only at the junction, each moving body is divided into a rigid body and a deformable body: the bottom wall of the piston and its side wall, the bottom part of the valves and their rods, as shown in Fig. 1.15. The rigid parts are moved with the prescribed rigid displacement, whereas we impose a movement to the deformable bodies that is proportional to the distance to the fixed boundary.

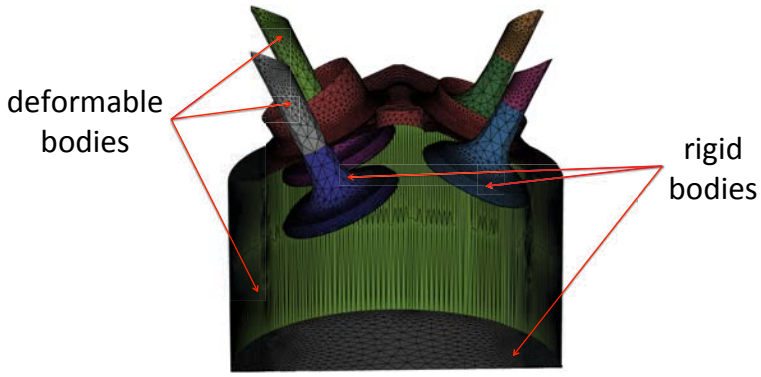

Figure 1.15: Engine piston. The moving bodies (the valves and the piston) are decomposed into a deformable part and a rigid part.

Moving mesh strategy

The moving mesh strategy must be able to handle two aspects: the rigid movement of the valves into the piston, that creates a lot of shearing of the mesh, and the expansion of the piston that results in largely stretched elements. To handle the displacement of the valves, the connectivity-change moving mesh algorithm is very efficient, and preserves the quality of the mesh around the valves. However, as discussed above, it is not enough to handle the large volume variation. To avoid creating too large elements (when the piston is expanded) or too small elements (when the piston is compressed), we have chosen to cut the simulation interval into smaller sub-intervals, where we can more easily control the sizes of the elements. Since the displacement of the piston is non linear, cutting into sub-interval of the same size would not be efficient, so we cut into sub-intervals such that the volume variation is the same. More precisely, on each sub-interval, the volume is at most multiplied or divided by two. To be more generic, the code computes the initial domain volume and the volume variation and exits when it reaches 2 or $\frac{1}{2}$ with respect to the initial volume, or when a change in the volume variation is detected. The piston is then remeshed to control the sizes of the elements. The first idea was to remesh with the initial size h (that is considered as the best for this mesh). However, this would double or be reduced by half the size of the elements by the end of the sub-interval.

To stay closer to the "best" size, we remesh prescribing a size of $\frac{h}{\sqrt{2}}$ or $\sqrt{2}h$ (depending on whether the volume is increasing or decreasing). This way, the sizes of the elements within a sub-interval are expected to stay in the interval $\left[\frac{h}{\sqrt{2}}, \sqrt{2}h\right]$ that is around h.

Some gradation can be added in the volume, to have bigger elements at the center of the chamber. In this case, there is no physical simulation coupled to the moving mesh, so we have to invent an acceptable metric. If a real simulation were run, the metric would be derived from the solution, as explained in Part II.

The volume meshes at different times are showed in Fig. 1.16. One can see that, unlike in Fig. 1.14, the volume of the piston is meshed with regular elements all the time. The remeshings at chosen times avoid the elements to be too stretched, as can be seen in Fig. 1.17, where the mesh at the beginning and at the end of a sub-interval without remeshing is shown.

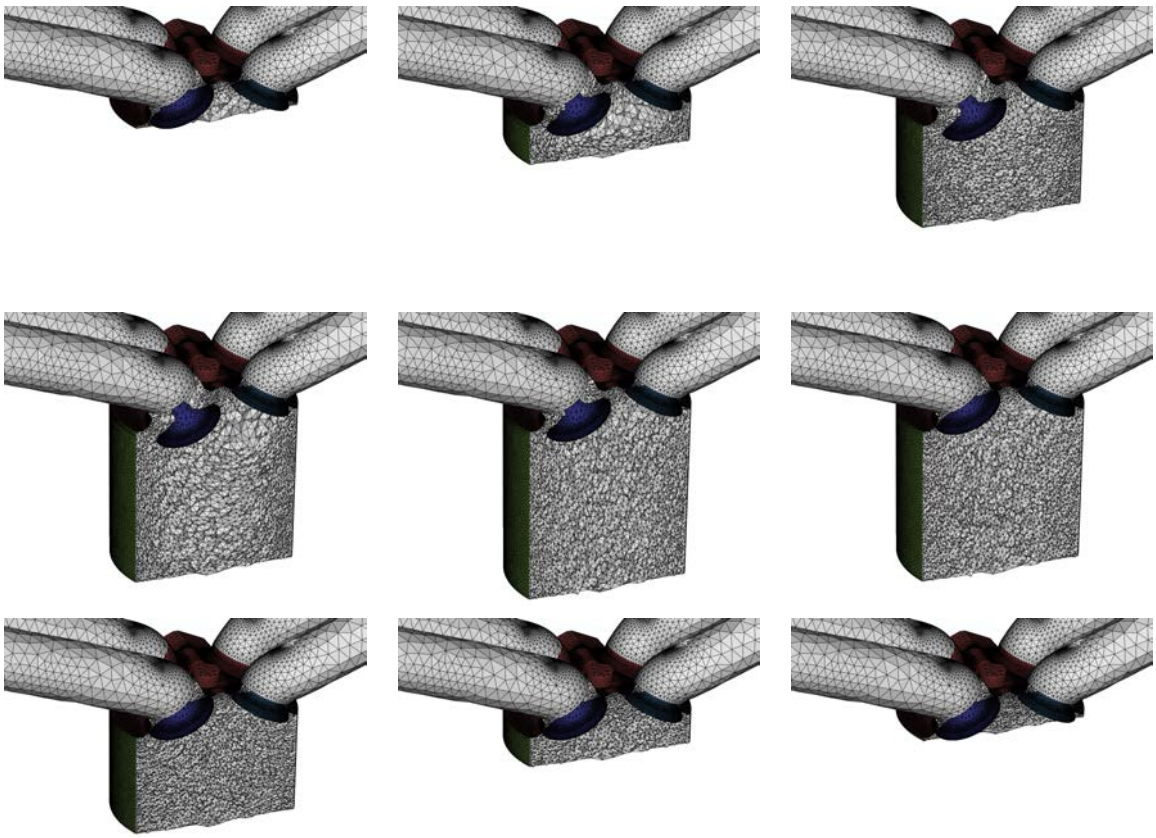

Figure 1.16: Engine piston. Volume mesh at different times of the first crank cycle: $t = 0, 13, 25, 37, 50, 62, 75, 88$ and 100.

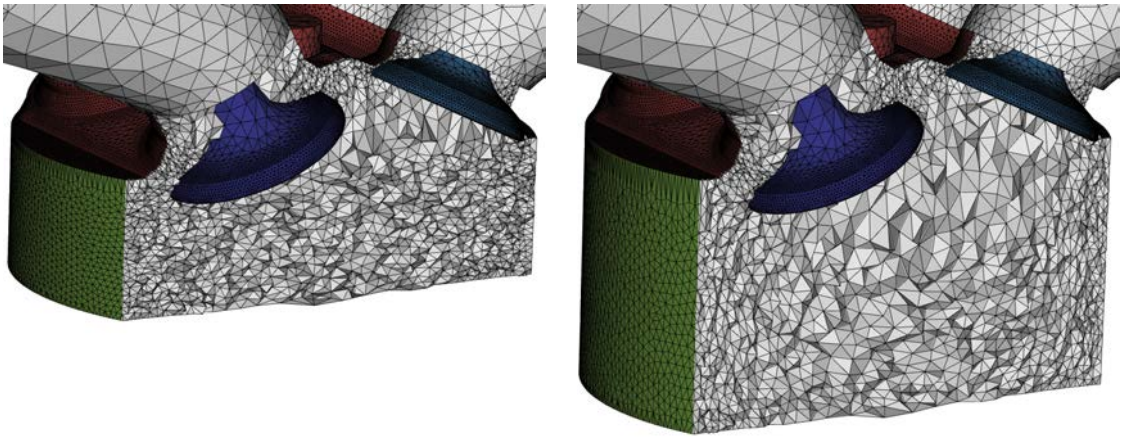

Figure 1.17: Engine piston. Volume mesh at the beginning and at the end of a sub-interval ($t = 14$ and 21), *i.e.* just after (left) and just before (right) a remeshing.

1.6 Conclusion

In this chapter, the connectivity-change moving mesh algorithm used and improved during this thesis was presented. It is based on an elasticity-like mesh deformation step performed for a large time step and mesh optimization, in particular swaps, performed in between to preserve the mesh quality without ever remeshing. A new method to compute the mesh deformation solution was considered, the IDW method. It is a direct interpolation method, in which the movement of inner vertices is computed as a weighted average of the movement of the boundaries. An MPI code implementing this method was plugged into our moving mesh code. A comparison of the two methods was performed on several 3D examples, that demonstrates the efficiency and the robustness of the moving mesh algorithm with both methods. The main difference lies in the fact that the elasticity-like method moves vertices farther away from the moving bodies, which facilitates the movement in the case of sharp shearing, while the IDW method is very rigid around the moving bodies, facilitating the translation of small details. If small differences appear depending on the cases, both methods eventually result in high quality meshes. Finally, the case of boundary layer meshes and of large volume variations was considered, opening the way for specific moving mesh approaches for these cases.

Chapter 2

ALE solver

An Arbitrary Lagrangian Eulerian (ALE) flow solver has been coupled to the moving mesh process described in Algorithm 2 Section 1.1. In this chapter, we discuss in detail the implemented solver, and all the choices that were made from the numerous possibilities available in the literature.

The conventions adopted, notably for the normals, are recalled at the beginning of this thesis (page 3).

The works presented in this Chapter were published in [Barral 2014a].

2.1 Euler equations in the ALE framework

We consider the compressible Euler equations for a Newtonian fluid in their ALE formulation. The ALE formulation allows the equations to take arbitrary motion of the mesh into account. Assuming that the gas is perfect, inviscid and that there is no thermal diffusion, the ALE formulation of the equations is written, for any arbitrary closed volume $C(t)$ of boundary $\partial C(t)$ moved with mesh velocity $\boldsymbol{w}$:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\left(\int_{C(t)} \mathbf{W}\mathrm{d}\mathbf{x}\right) + \int_{\partial C(t)} \left(\mathcal{F}(\mathbf{W}) - \mathbf{W} \otimes \boldsymbol{w}\right) \cdot \mathbf{n}\,\mathrm{d}\mathbf{s} &= \int_{C(t)} \mathbf{F}_{ext}\,\mathrm{d}\mathbf{x} \\ \iff \frac{\mathrm{d}}{\mathrm{d}t}\left(\int_{C(t)} \mathbf{W}\mathrm{d}\mathbf{x}\right) + \int_{\partial C(t)} \left(\mathbf{F}\left(\mathbf{W}\right) - \mathbf{W}(\boldsymbol{w} \cdot \mathbf{n})\right)\,\mathrm{d}\mathbf{s} &= \int_{C(t)} \mathbf{F}_{ext}\,\mathrm{d}\mathbf{x}\,, \end{aligned} \tag{2.1}$$

$$\text{where } \begin{cases} \mathbf{W} = (\rho, \rho\mathbf{u}, \rho e)^T \text{ is the conservative variables vector} \\ \mathcal{F}(\mathbf{W}) = (\rho\mathbf{u},\ \rho u_x\mathbf{u} + p\mathbf{e}_x,\ \rho u_y\mathbf{u} + p\mathbf{e}_y,\ \rho u_z\mathbf{u} + p\mathbf{e}_z,\ \rho\mathbf{u}h) \text{ is the flux tensor} \\ \mathbf{F}\left(\mathbf{W}\right) = \mathcal{F}(\mathbf{W}) \cdot \mathbf{n} = (\rho\eta,\ \rho u_x\eta + pn_x,\ \rho u_y\eta + pn_y,\ \rho u_z\eta + pn_z,\ \rho e\eta + p\eta)^T \\ \mathbf{F}_{ext} = (0,\ \rho\,\mathbf{f}_{ext},\ \rho\mathbf{u} \cdot \mathbf{f}_{ext})^T \text{ is the contribution of the external forces,} \end{cases}$$

and we have noted ρ the density of the fluid, p the pressure, $\mathbf{u} = (u_x, u_y, u_z)$ its Eulerian velocity, $\eta = \mathbf{u} \cdot \mathbf{n}$, $q = \|\mathbf{u}\|$, ε the internal energy per unit mass, $e = 1/2\,q^2 + \varepsilon$ the total energy per unit mass, $h = e + p/\rho$ the enthalpy per unit mass of the flow, $\mathbf{f}_{ext}$ the resultant of the volumic external forces applied on the particle and $\mathbf{n}$ the outward normal to interface $\partial C(t)$ of $C(t)$.

2.2 Spatial discretization

As regards spatial discretization of the solver, we use an edge-based finite-volume approach, with an HLLC Riemann approximate solver and second-order MUSCL gradient reconstruction.

The main difference when translating these schemes from the standard formulation to the ALE formulation is the addition of the mesh velocities in the wave speeds of the Riemann problem.

2.2.1 Edge-based Finite-Volume solver

The domain Ω is discretized by a tetrahedral unstructured mesh $\mathcal{H}$. The vertex-centered Finite Volume formulation consists in associating a control volume denoted $C_i(t)$ with each vertex P_i of the mesh and at each time t. The dual Finite Volume cell mesh is built by the rule of medians. The common boundary $\partial C_{ij}(t) = \partial C_i(t) \cap \partial C_j(t)$ between two neighboring cells $C_i(t)$ and $C_j(t)$ is decomposed into several triangular interface facets. The normal flux $\mathbf{F}_{ij}(t)$ along each cell interface is taken to be constant (not in time but in space), just like the solution $\mathbf{W}_{ij}$ on the interface.
Rewriting System (2.1) for $C(t) = C_i(t)$, we get the following semi-discretization at P_i:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(|C_i(t)|\mathbf{W}_i(t)\right) + \sum_{P_j \in V_i} |\partial C_{ij}(t)|\, \mathbf{\Phi}_{ij}\left(\mathbf{W}_i(t),\, \mathbf{W}_j(t),\, \mathbf{n}_{ij}(t),\, \sigma_{ij}(t)\right) = 0\,, \tag{2.2}$$

- $\mathbf{W}_i(t)$ is the mean value of state $\mathbf{W}$ in cell C_i at time t
- V_i is the set of all neighboring vertices of P_i, *i.e.* the mesh vertices connected connected to P_i by an edge
- $\mathbf{n}_{ij}$ is the outward normalized normal (with respect to cell C_i) of cell interface ∂C_{ij}
- $\mathbf{F}_{ij}(t) = \mathcal{F}(\mathbf{W}_{ij}(t)) \cdot \mathbf{n}_{ij}(t)$ is an approximation of the physical flux through $\partial C_{ij}(t)$[1]
- $\sigma_{ij}(t) = \dfrac{1}{|\partial C_{ij}(t)|} \displaystyle\int_{\partial C_{ij}(t)} \boldsymbol{w}_{ij}(t) \cdot \mathbf{n}_{ij}(t) \mathrm{d}\boldsymbol{s}$ is the normal velocity of cell interface $\partial C_{ij}(t)$
- $\mathbf{\Phi}_{ij}\left(\mathbf{W}_i(t), \mathbf{W}_j(t), \mathbf{n}_{ij}(t), \sigma_{ij}(t)\right) \approx \mathbf{F}_{ij}(t) - \mathbf{W}_{ij}(t)\sigma_{ij}(t)$ is the numerical flux function used to approximate the flux at cell interface $\partial C_{ij}(t)$.

The computation of the convective fluxes is performed mono-dimensionally in the direction normal to each Finite Volume cell interface. Consequently, the numerical evaluation of the flux function $\mathbf{\Phi}_{ij}$ at interface ∂C_{ij} can be achieved by solving, at each time step, a one-dimensional Riemann problem in direction $\mathbf{n}_{ij} = \mathbf{n}$ with initial values $\mathbf{W}_L = \mathbf{W}_i$ on the left of the interface and $\mathbf{W}_R = \mathbf{W}_j$ on the right. The normal speed to the interface is temporarily noted σ for clarity.

2.2.2 HLLC numerical flux

The methodology provided by Batten [Batten 1997] can be extended to the Euler equations in their ALE formulation. The HLLC flux is then described by three waves phase velocities:

$$S_L = \min(\eta_L - c_L, \tilde{\eta} - \tilde{c}) \quad \text{and} \quad S_R = \max(\eta_R + c_R, \tilde{\eta} + \tilde{c})$$
$$S_M = \frac{\rho_R\eta_R(S_R - \eta_R) - \rho_L\eta_L(S_L - \eta_L) + p_L - p_R}{\rho_R(S_R - \eta_R) - \rho_L(S_L - \eta_L)}$$

[1] Our convention is that the flux is positive if it goes in the same direction as the normal. Thus, $\mathbf{F}_{ij} > 0$ means that the flux goes from cell C_i to C_j. If $\sigma_{ij} > 0$, the geometrical flux $-\mathbf{W}_{ij}(t)\sigma_{ij}$ is negative and therefore oriented from C_j to C_i.

and two approximate states:

$$\mathbf{W}_\mathbf{L}^* = \begin{cases} \rho_L^* = \rho_L \dfrac{S_L - \eta_L}{S_L - S_M} \\ p_L^* = p^* = \rho_L \left(\eta_L - S_L\right)\left(\eta_L - S_M\right) + p_L \\ (\rho\mathbf{u})_L^* = \dfrac{\left(S_L - \eta_L\right)\rho\mathbf{u}_L + \left(p^* - p_L\right)\mathbf{n}}{S_L - S_M} \\ (\rho e)_L^* = \dfrac{\left(S_L - \eta_L\right)\rho e_L - p_L\eta_L + p^* S_M}{S_L - S_M} \end{cases}, \quad \mathbf{W}_\mathbf{R}^* = \begin{cases} \rho_R^* = \rho_R \dfrac{S_R - \eta_R}{S_R - S_M} \\ p_R^* = p^* = \rho_R \left(\eta_R - S_R\right)\left(\eta_R - S_M\right) + p_R \\ (\rho\mathbf{u})_R^* = \dfrac{\left(S_R - \eta_R\right)\rho\mathbf{u}_R + \left(p^* - p_R\right)\mathbf{n}}{S_R - S_M} \\ (\rho e)_R^* = \dfrac{\left(S_R - \eta_R\right)\rho e_R - p_R\eta_R + p^* S_M}{S_R - S_M} \end{cases},$$

where $\eta = \mathbf{u}\cdot\mathbf{n}$ is the interface normal velocity and $\tilde{.}$ are Roe average variables [Roe 1981]. The HLLC flux through the interface is finally given by:

$$\mathbf{\Phi}_{\text{HLLC}}(\mathbf{W}_L, \mathbf{W}_R, \mathbf{n}, \sigma) = \begin{cases} \mathbf{F}_L - \sigma\mathbf{W}_L & \text{if } S_L - \sigma > 0 \\ \mathbf{F}_L^* - \sigma\mathbf{W}_L^* & \text{if } S_L - \sigma \leq 0 < S_M - \sigma \\ \mathbf{F}_R^* - \sigma\mathbf{W}_R^* & \text{if } S_M - \sigma \leq 0 \leq S_R - \sigma \\ \mathbf{F}_R - \sigma\mathbf{W}_R & \text{if } S_R - \sigma < 0 \end{cases}.$$

The HLLC approximate 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.

2.2.3 High-order scheme

The previous formulation reaches at best a first-order spatial accuracy. A MUSCL type reconstruction method has been designed to increase the order of accuracy of the scheme. The idea is to use extrapolated values $\mathbf{U}_{ij}$ and $\mathbf{U}_{ji}$ of $\mathbf{U}$ at interface ∂C_{ij} to evaluate the flux, where $\mathbf{U} = (\rho, \mathbf{u}, p)$ is the vector of physical variables. The following approximation is performed: $\mathbf{\Phi}_{ij} = \mathbf{\Phi}(\mathbf{U}_{ij}, \mathbf{U}_{ji}, \mathbf{n}_{ij}, \sigma_{ij})$ with $\mathbf{U}_{ij}$ and $\mathbf{U}_{ji}$ linearly interpolated state values on each side of the interface:

$$\mathbf{U}_{ij} \;=\; \mathbf{U}_i + \frac{1}{2}\left(\nabla\mathbf{U}\right)_{ij}\cdot\mathbf{P}_i\mathbf{P}_j\,, \quad \text{and} \quad \mathbf{U}_{ji} \;= \mathbf{U}_j + \frac{1}{2}\left(\nabla\mathbf{U}\right)_{ji}\cdot\mathbf{P}_i\mathbf{P}_j\,. \tag{2.3}$$

In contrast to the original MUSCL approach, the approximate "slopes" $\left(\nabla\mathbf{U}\right)_{ij}$ and $\left(\nabla\mathbf{U}\right)_{ji}$ are defined for each edge using a combination of centered, upwind and nodal gradients.
The centered gradient related to edge $\mathbf{P}_i\mathbf{P}_j$, is defined implicitly along edge $\mathbf{P}_i\mathbf{P}_j$ via relation:

$$\left(\nabla\mathbf{U}\right)_{ij}^C \cdot \mathbf{P}_i\mathbf{P}_j = \mathbf{U}_j - \mathbf{U}_i\,. \tag{2.4}$$

Upwind and downwind gradients, which are also related to edge $\mathbf{P}_i\mathbf{P}_j$, are computed using the upstream and downstream tetrahedra associated with this edge. These tetrahedra are respectively denoted K_{ij} and K_{ji}. K_{ij} is the unique tetrahedron of the ball of P_i (resp. P_j) whose opposite face is crossed by the straight line prolongating edge $\mathbf{P}_i\mathbf{P}_j$, see Figure 2.1. Upwind and downwind gradients of edge $\mathbf{P}_i\mathbf{P}_j$ are then defined as:

$$\left(\nabla\mathbf{U}\right)_{ij}^U = \left(\nabla\mathbf{U}\right)_{|K_{ij}} \quad \text{and} \quad \left(\nabla\mathbf{U}\right)_{ij}^D = \left(\nabla\mathbf{U}\right)_{|K_{ji}}\,,$$

where

$$\nabla \mathbf{U}_{|_K} = \sum_{P \in K} (\nabla \phi_P \times \mathbf{U}_P)$$

is the $\mathbb{P}^1$-Galerkin gradient on element K, ϕ_P is the basis function associated with P and $\times$ is the Hadamard product (term-by-term product). Parametrized nodal gradients are built by introducing the β-scheme:

$$\begin{aligned} \nabla \mathbf{U}_{ij} \cdot \mathbf{P}_i\mathbf{P}_j &= (1-\beta)\,(\nabla \mathbf{U})^C_{ij} \cdot \mathbf{P}_i\mathbf{P}_j + \beta\,(\nabla \mathbf{U})^U_{ij} \cdot \mathbf{P}_i\mathbf{P}_j, \\ \nabla \mathbf{U}_{ji} \cdot \mathbf{P}_i\mathbf{P}_j &= (1-\beta)\,(\nabla \mathbf{U})^C_{ij} \cdot \mathbf{P}_i\mathbf{P}_j + \beta\,(\nabla \mathbf{U})^D_{ij} \cdot \mathbf{P}_i\mathbf{P}_j\,, \end{aligned}$$

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 $\beta = 1$.

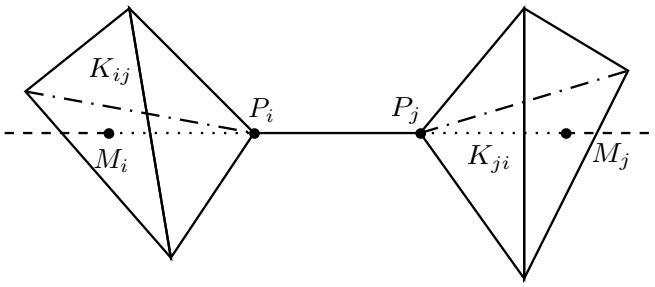

Figure 2.1: Downstream K_{ij} and upstream K_{ji} tetrahedra associated with edge $\mathbf{P}_i\mathbf{P}_j$.

The most accurate β-scheme is obtained for $\beta = 1/3$, also called the V4-scheme. This scheme is third-order for the two-dimensional linear advection problem on structured triangular meshes [Debiez 2000]. In our case, for the non-linear Euler equations on unstructured meshes, a second-order scheme with a fourth-order numerical dissipation is obtained. The high-order gradients are given by:

$$\begin{aligned} (\nabla \mathbf{U})^{V4}_{ij} \cdot \mathbf{P}_i\mathbf{P}_j &= \tfrac{2}{3}\,(\nabla \mathbf{U})^C_{ij} \cdot \mathbf{P}_i\mathbf{P}_j + \tfrac{1}{3}\,(\nabla \mathbf{U})^U_{ij} \cdot \mathbf{P}_i\mathbf{P}_j, \\ (\nabla \mathbf{U})^{V4}_{ji} \cdot \mathbf{P}_i\mathbf{P}_j &= \tfrac{2}{3}\,(\nabla \mathbf{U})^C_{ij} \cdot \mathbf{P}_i\mathbf{P}_j + \tfrac{1}{3}\,(\nabla \mathbf{U})^D_{ij} \cdot \mathbf{P}_i\mathbf{P}_j\,. \end{aligned}$$

2.2.4 Limiter

The previous MUSCL schemes are not monotone. Therefore, limiting functions must be coupled with the previous high-order gradient evaluations to guarantee the Total-Variation-Diminishing (TVD) property of the scheme. To this end, the gradient of Relation (2.3) is replaced by a limited gradient denoted $\left(\nabla \mathbf{U}^{lim}\right)_{ij}$. Here, the three-entry limiter introduced by Dervieux [Cournède 2006], which is a generalization of the SuperBee limiter, will be used :

$$\left(\nabla \mathbf{U}^{lim}\right)_{ij} \cdot \mathbf{P}_i\mathbf{P}_j = L\left(\left(\nabla \mathbf{U}^D\right)_{ij} \cdot \mathbf{P}_i\mathbf{P}_j, \left(\nabla \mathbf{U}^C\right)_{ij} \cdot \mathbf{P}_i\mathbf{P}_j, \left(\nabla \mathbf{U}^{V4}\right)_{ij} \cdot \mathbf{P}_i\mathbf{P}_j\right)$$

$$\text{with: } L(a,b,c) = \begin{cases} 0 & \text{if } ab \le 0 \\ \operatorname{sign}(a)\min(2\,|a|, 2\,|b|, |c|) & \text{otherwise} \end{cases}.$$

2.2.5 Boundary conditions

Boundary conditions are computed vertex-wise. Several types of boundary conditions are considered, but only one, the slipping condition, is applied to moving bodies.

Slipping condition

The ALE slipping boundary flux of vertex P_i reads:

$$\mathbf{\Phi}_{\text{slip}}\left(\mathbf{W}_i, \mathbf{n}_i, \sigma_i\right) = \begin{pmatrix} 0 \\ -p_i \mathbf{n}_i \\ -p_i \sigma_i \end{pmatrix} |\partial C_i|_\Gamma \,, \tag{2.5}$$

where p_i is the vertex pressure, $\mathbf{n}_i = \frac{\sum_{K_j \supset P_i} |K_j| \mathbf{n}_{K_j}}{\sum_{K_j \supset P_i} |K_j|}$ is the mean outward normal of the boundary interface, $|\partial C_i|_\Gamma = \sum_{K_j \supset P_i} \frac{1}{3}|K_j|$ is the interface area and σ_i is the boundary interface celerity, see Section 2.3.5.

Symmetry condition

The flux for symmetry boundary conditions is the same as for the slipping condition.

Free-stream condition

An inflow/external flow condition is also used, to model a flow coming from an infinite known state $\mathbf{W}_\infty$. An upwind Steger-Warning [Steger 1981] flux is used:

$$\mathbf{\Phi}_{\text{free-stream}}\left(\mathbf{W}_i, \mathbf{n}_i\right) = A^+(\mathbf{W}_i, \mathbf{n}_i)\mathbf{W}_i + A^-(\mathbf{W}_i, \mathbf{n}_i)\mathbf{W}_\infty \,, \tag{2.6}$$

where $A^+ = \frac{|A|+A}{2}$ and $A^- = \frac{|A|-A}{2}$ and A is the Jacobian matrix $A = \frac{\partial \mathbf{F}}{\mathbf{W}}$.

Transmitting condition

For this boundary condition we assume that the state behind the boundary is the same as the one inside. Therefore, the flux through the boundary is given by:

$$\mathbf{\Phi}_{\text{transmitting}}\left(\mathbf{W}_i, \mathbf{n}_i\right) = \mathbf{\Phi}_{\text{HLLC}}\left(\mathbf{W}_i, \mathbf{W}_i, \mathbf{n}_i\right) \,. \tag{2.7}$$

2.3 Time discretization

Temporal discretization is a more complex matter, as there is no consensus on some key aspects. We chose an explicit time discretization, which is simpler than implicit discretizations, and seems more natural when dealing with FSI simulations. The most controversial subject is the Geometrical Conservation Law, whose application is a matter of debate, but which can be used to rigorously determine when the geometrical parameters that appear in the fluxes should be computed.

2.3.1 The Geometric Conservation Law

We need to make sure that the movement of the mesh is not responsible for any artificial alteration of the physical phenomena involved, or at least, to make our best from a numerical point of view for the mesh movement to introduce an error of the same order as the one introduced by the numerical scheme. If System (2.1) is written for a constant state, assuming $\mathbf{F}_{ext} = 0$, we get, for any arbitrary closed volume $C = C(t)$:

$$\frac{\mathrm{d}\left(|C(t)|\right)}{\mathrm{d}t} - \int_{\partial C(t)} (\boldsymbol{w} \cdot \mathbf{n})\, \mathrm{d}\boldsymbol{s} = 0\,. \tag{2.8}$$

As the constant state is a solution of the Euler equations, if boundaries transmit the flux towards the outside as it comes, we find a purely geometrical relation inherent to the continuous problem. For any arbitrary closed volume $C = C(t)$ of boundary $\partial C(t)$, this relation is integrated into:

$$|C(t+\delta t)| - |C(t)| = \int_t^{t+\delta t} \int_{\partial C(t)} (\boldsymbol{w} \cdot \mathbf{n})\, \mathrm{d}\boldsymbol{s}\mathrm{d}t, \quad \text{with } t \text{ and } t+\delta t \in [0,T]\,, \tag{2.9}$$

which is usually known as the Geometric Conservation Law (GCL). From a geometric point of view, this relation states that the algebraic variation of the volume of C between two instants equals the algebraic volume swept by its boundary.

The role of the GCL is still controversial, and many papers have tried to analyze its role in ALE simulations. Should Relation (2.8) be satisfied at the discrete level? What are the effects of respecting this law at a discrete level on the consistency, stability and accuracy of the numerical scheme? How can it be enforced at a discrete level? A recent review on the subject can be found in [Etienne 2009]. It has been shown that the GCL is neither a necessary nor a sufficient condition to preserve time accuracy, however violating it can lead to numerical oscillation [Mavriplis 2006]. In [Farhat 2001] the authors show that compliance with the GCL guarantees an accuracy of at least the first order in some conditions. Therefore, most would agree that the GCL should be enforced at the discrete level for a large majority of cases.

2.3.2 Discrete GCL enforcement

A new approach to enforcing the Discrete GCL was proposed in [Mavriplis 2006, Yang 2005, Yang 2007], in which the authors proposed a framework to build ALE high order temporal schemes that reach approximately the design order of accuracy. The originality of this approach consists in precisely defining which ALE parameters are true degrees of freedom and which are not. In contrast to other approaches [Koobus 1999, Lesoinne 1996, Nkonga 2000], they consider that the times and configurations at which the fluxes are evaluated do not constitute a new degree of freedom to be set thanks to the ALE scheme. To maintain the design accuracy of the fixed-mesh temporal integration, the moment at which the geometric parameters, such as the cells' interfaces' normals or the upwind/downwind tetrahedra must be computed, is entirely determined by the intermediate configurations involved in the temporal scheme chosen. The only degree of freedom to be set by enforcing the GCL at the discrete level is σ. Incidentally, it is implicitly stated that $\boldsymbol{w}$ is never involved alone but only hidden in the term $\sigma\|\mathbf{n}\|$ which represents the instantaneous algebraic volume swept.

In practical terms, the interfaces normal speeds are found by simply rewriting the scheme for a constant discrete solution, which leads to a small linear system that is easily invertible by hand. This procedure is detailed in the next section for one Runge-Kutta scheme. Any fixed-mesh explicit RK scheme can be extended to the case of moving meshes thanks to this methodology, and the resulting RK scheme is naturally DGCL. Even if this has not been proven theoretically, the expected temporal order of convergence has also been observed numerically for several schemes designed using this method [Yang 2005].

2.3.3 RK schemes

Runge-Kutta (RK) methods are famous multi-stage methods to integrate ODEs. In the numerical solution of hyperbolic PDEs, notably the Euler equations, the favorite schemes among the huge family of Runge-Kutta schemes are those satisfying the Strong Stability Preserving (SSP) property. A Runge-Kutta scheme is said to be SSP if $|\mathbf{W}^{n+1}| \leq |\mathbf{W}^n|$, $|\cdot|$ being a chosen semi-norm [Shu 1988, Spiteri 2002].

In what follows, we denote by SSPRK(S,P) the S-stage RK scheme of order P. We adopt the following notations:

$$\mathbf{f}_i^s = \sum_{j=1}^{n_i} \mathbf{\Phi}(\mathbf{W}_i^s, \mathbf{W}_j^s, \boldsymbol{\eta}_{ij}^s, \sigma_{ij}^s) \text{ with } \begin{cases} n_i & \text{the number of edges in the ball of vertex } P_i \\ \boldsymbol{\eta}_{ij}^s & \text{the outward } \textbf{non-normalized} \text{ normal to the} \\ & \text{portion of the interface of cell } C_i^s \text{ around} \\ & \text{edge } e_{ij} \\ \sigma_{ij}^s & \text{normal speed of the interface around edge } e_{ij} \text{ of} \\ & \text{cell } C_i^s \ . \end{cases}$$

Superscript notation X^s indicates that the quantity considered is the X obtained at stage s of the Runge-Kutta process. For instance, C_i^s is the cell associated with vertex P_i when the mesh has been moved to its s^{th} Runge-Kutta configuration. Coefficients $(c_s)_{0\leq s\leq S}$ indicate the relative position in time of the current Runge-Kutta configuration: $t^s = t^n + c_s\,\tau$ with $\tau = t^{n+1} - t^n$. Finally, we denote by A_{ij}^s the volume swept by the interface around edge e_{ij} of cell C_i between the initial Runge-Kutta configuration and the s^{th} configuration.

2.3.4 Application to the SSPRK(4,3) scheme

This approach was used, for example, to build the 3rd order 4-step Runge-Kutta scheme [Olivier 2011b], whose Butcher and Shu-Osher representations are given below:

Butcher representation	Shu-Osher representation	$t^s = t^n + c_s\,\tau$
$\mathbf{Y}_i{}^0 = \mathbf{Y}_i{}^n$	$\mathbf{Y}_i{}^0 = \mathbf{Y}_i{}^n$	$c_0 = 0, \quad t^0 = t^n$
$\mathbf{Y}_i{}^1 = \mathbf{Y}_i{}^0 + \frac{\tau}{2}\mathbf{f}_i^0$	$\mathbf{Y}_i{}^1 = \mathbf{Y}_i{}^0 + \frac{\tau}{2}\mathbf{f}_i^0$	$c_1 = \frac{1}{2}, \quad t^1 = t^n + \frac{1}{2}\tau$
$\mathbf{Y}_i{}^2 = \mathbf{Y}_i{}^0 + \frac{\tau}{2}\left(\mathbf{f}_i^0 + \mathbf{f}_i^1\right)$	$\mathbf{Y}_i{}^2 = \mathbf{Y}_i{}^1 + \frac{\tau}{2}\mathbf{f}_i^1$	$c_2 = 1, \quad t^2 = t^{n+1}$
$\mathbf{Y}_i{}^3 = \mathbf{Y}_i{}^0 + \frac{\tau}{6}\left(\mathbf{f}_i^0 + \mathbf{f}_i^1 + \mathbf{f}_i^2\right)$	$\mathbf{Y}_i{}^3 = \frac{2}{3}\mathbf{Y}_i{}^0 + \frac{1}{3}\mathbf{Y}_i{}^2 + \frac{\tau}{6}\mathbf{f}_i^2$	$c_3 = \frac{1}{2}, \quad t^3 = t^n + \frac{1}{2}\tau$
$\mathbf{Y}_i{}^4 = \mathbf{Y}_i{}^0 + \frac{\tau}{2}\left(\frac{1}{3}\mathbf{f}_i^0 + \frac{1}{3}\mathbf{f}_i^1 + \frac{1}{3}\mathbf{f}_i^2 + \mathbf{f}_i^3\right)$	$\mathbf{Y}_i{}^4 = \mathbf{Y}_i{}^3 + \frac{\tau}{2}\mathbf{f}_i^3$	$c_4 = 1, \quad t^4 = t^{n+1}$

For this scheme to be DGCL, it must preserve a constant solution $\mathbf{W}_i = \mathbf{W}_0$, as stated in Section 2.3.1. In this specific case, our conservative variable is $\mathbf{Y}_i = |C_i|\mathbf{W}_0$ and the purely physical fluxes vanish, leading to $\mathbf{f}_i^s = -\mathbf{W}_0 \sum_{j=1}^{n_i} \boldsymbol{\eta}_{ij}^s \sigma_{ij}^s$. Therefore, the scheme reads:

$$
\begin{aligned}
|C_i^0| &= |C_i^n| \\
|C_i^1| - |C_i^0| &= \sum_{j=1}^{n_i} A_{ij}^1 = \frac{\tau}{2}\sum_{j=1}^{n_i} \boldsymbol{\eta}_{ij}^0 \sigma_{ij}^0 \\
|C_i^2| - |C_i^0| &= \sum_{j=1}^{n_i} A_{ij}^2 = \frac{\tau}{2}\sum_{j=1}^{n_i} \left(\boldsymbol{\eta}_{ij}^0 \sigma_{ij}^0 + \boldsymbol{\eta}_{ij}^1 \sigma_{ij}^1\right) \\
|C_i^3| - |C_i^0| &= \sum_{j=1}^{n_i} A_{ij}^3 = \frac{\tau}{6}\sum_{j=1}^{n_i} \left(\boldsymbol{\eta}_{ij}^0 \sigma_{ij}^0 + \boldsymbol{\eta}_{ij}^1 \sigma_{ij}^1 + \boldsymbol{\eta}_{ij}^2 \sigma_{ij}^2\right) \\
|C_i^4| - |C_i^0| &= \sum_{j=1}^{n_i} A_{ij}^4 = \frac{\tau}{2}\sum_{j=1}^{n_i} \left(\frac{1}{3}\boldsymbol{\eta}_{ij}^0 \sigma_{ij}^0 + \frac{1}{3}\boldsymbol{\eta}_{ij}^1 \sigma_{ij}^1 + \frac{1}{3}\boldsymbol{\eta}_{ij}^2 \sigma_{ij}^2 + \boldsymbol{\eta}_{ij}^3 \sigma_{ij}^3\right),
\end{aligned}
$$

where A_{ij}^s is the volume swept by the interface around edge e_{ij} of cell C_i between the initial and the s^{th} Runge-Kutta configuration. A natural necessary condition for the above relations to be satisfied is to have, for each interface around edge e_{ij} of each Finite Volume cell C_i:

$$
\begin{pmatrix} A_{ij}^1 \\ A_{ij}^2 \\ A_{ij}^3 \\ A_{ij}^4 \end{pmatrix} = \tau \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} \boldsymbol{\eta}_{ij}^0 \sigma_{ij}^0 \\ \boldsymbol{\eta}_{ij}^1 \sigma_{ij}^1 \\ \boldsymbol{\eta}_{ij}^2 \sigma_{ij}^2 \\ \boldsymbol{\eta}_{ij}^3 \sigma_{ij}^3 \end{pmatrix} \quad \Leftrightarrow \quad \begin{pmatrix} \boldsymbol{\eta}_{ij}^0 \sigma_{ij}^0 \\ \boldsymbol{\eta}_{ij}^1 \sigma_{ij}^1 \\ \boldsymbol{\eta}_{ij}^2 \sigma_{ij}^2 \\ \boldsymbol{\eta}_{ij}^3 \sigma_{ij}^3 \end{pmatrix} = \frac{1}{\tau} \begin{pmatrix} 2 & 0 & 0 & 0 \\ -2 & 2 & 0 & 0 \\ 0 & -2 & 6 & 0 \\ 0 & 0 & -2 & 2 \end{pmatrix} \begin{pmatrix} A_{ij}^1 \\ A_{ij}^2 \\ A_{ij}^3 \\ A_{ij}^4 \end{pmatrix}. \tag{2.10}
$$

Therefore, the normal speed of the interface around edge e_{ij} of cell C_i must be updated in the Runge-Kutta process as follows :

$$
\sigma_{ij}^0 = \frac{2A_{ij}^1}{\tau\ \boldsymbol{\eta}_{ij}^0}, \quad \sigma_{ij}^1 = \frac{-2A_{ij}^1 + 2A_{ij}^2}{\tau\ \boldsymbol{\eta}_{ij}^1}, \quad \sigma_{ij}^2 = \frac{-2A_{ij}^2 + 6A_{ij}^3}{\tau\ \boldsymbol{\eta}_{ij}^2}, \quad \sigma_{ij}^3 = \frac{-2A_{ij}^3 + 2A_{ij}^4}{\tau\ \boldsymbol{\eta}_{ij}^3}, \tag{2.11}
$$

and the $\boldsymbol{\eta}_{ij}^s$ and A_{ij}^s are computed on the mesh once it has been moved to the s^{th} Runge-Kutta configuration.

2.3.5 Practical computation of the volumes swept

In this section, we detail how the volumes swept by the cells' interfaces are computed. The calculations are only presented in 3D, 2D formulas in our formalism can be found in [Olivier 2011a].

Normal of a moving triangle

The volume swept by a triangle T between two timesteps t^1 and t^2 is approximated by the volume for a *constant straightline* displacement, as proposed in [Nkonga 1994]:

$$A = (t^2 - t^1)\ \boldsymbol{w}\ \tilde{\boldsymbol{\eta}}\,, \tag{2.12}$$

where $\boldsymbol{w}$ is the speed of the center of gravity of the triangle, and $\tilde{\boldsymbol{\eta}}$ is an averaged outward pseudo normal:

$$\tilde{\boldsymbol{\eta}} = \int_{t^1}^{t^2}\int_T \mathbf{n}(t)\mathrm{d}\xi\mathrm{d}t\,, \tag{2.13}$$

where $\mathbf{n}$ is the outward normal of the triangle.

If one writes $\boldsymbol{\mathcal{N}}(\mathbf{x},\mathbf{y}) = \frac{1}{2}(\mathbf{x}_{P_0}\wedge\mathbf{y}_{P_1} + \mathbf{x}_{P_1}\wedge\mathbf{y}_{P_2} + \mathbf{x}_{P_2}\wedge\mathbf{y}_{P_0})$ where $\mathbf{x}$ and $\mathbf{y}$ are two vector fields and P_0, P_1, P_2 are the vertices of triangle T, the pseudo-normal is written:

$$\tilde{\boldsymbol{\eta}}^2(T) = \frac{1}{6}\left(2\boldsymbol{\mathcal{N}}\left(\mathbf{x}(t^2),\mathbf{x}(t^2)\right) + \boldsymbol{\mathcal{N}}\left(\mathbf{x}(t^2),\mathbf{x}(t^1)\right) + \boldsymbol{\mathcal{N}}\left(\mathbf{x}(t^1),\mathbf{x}(t^2)\right) + 2\boldsymbol{\mathcal{N}}\left(\mathbf{x}(t^1),\mathbf{x}(t^1)\right)\right)\,, \tag{2.14}$$

where $\vec{x^1}$ and $\vec{x^2}$ are the coordinate fields at times t^1 and t^2. Thus,

$$\begin{aligned}\tilde{\boldsymbol{\eta}}^2(T) = \frac{1}{12}\big[&2(\mathbf{P_0^2}\wedge\mathbf{P_1^2} + \mathbf{P_1^2}\wedge\mathbf{P_2^2} + \mathbf{P_2^2}\wedge\mathbf{P_0^2})\\ &+ (\mathbf{P_0^2}\wedge\mathbf{P_1^1} + \mathbf{P_1^2}\wedge\mathbf{P_2^1} + \mathbf{P_2^2}\wedge\mathbf{P_0^1})\\ &+ (\mathbf{P_0^1}\wedge\mathbf{P_1^2} + \mathbf{P_1^1}\wedge\mathbf{P_2^2} + \mathbf{P_2^1}\wedge\mathbf{P_0^2})\\ &+ 2(\mathbf{P_0^1}\wedge\mathbf{P_1^1} + \mathbf{P_1^1}\wedge\mathbf{P_2^1} + \mathbf{P_2^1}\wedge\mathbf{P_0^1})\big]\,,\end{aligned}$$

thus, opening the cross products in P_0:

$$\begin{aligned}\tilde{\boldsymbol{\eta}}^2(T) = \tfrac{1}{12}\ \Big[\ &2(\mathbf{P_0^2}\wedge(\mathbf{P_0^2}+\mathbf{P_0^2P_1^2}) + (\mathbf{P_0^2}+\mathbf{P_0^2P_1^2})\wedge(\mathbf{P_0^2}+\mathbf{P_0^2P_2^2})\\ &+(\mathbf{P_0^2}+\mathbf{P_0^2P_2^2})\wedge\mathbf{P_0^2})\\ +\ &(\mathbf{P_0^2}\wedge(\mathbf{P_0^1}+\mathbf{P_0^1P_1^1}) + (\mathbf{P_0^2}+\mathbf{P_0^2P_1^2})\wedge(\mathbf{P_0^1}+\mathbf{P_0^1P_2^1})\\ &+(\mathbf{P_0^2}+\mathbf{P_0^2P_2^2})\wedge\mathbf{P_0^1})\\ +\ &(\mathbf{P_0^1}\wedge(\mathbf{P_0^2}+\mathbf{P_0^2P_1^2}) + (\mathbf{P_0^1}+\mathbf{P_0^1P_1^1})\wedge(\mathbf{P_0^2}+\mathbf{P_0^2P_2^2})\\ &+(\mathbf{P_0^1}+\mathbf{P_0^1P_2^1})\wedge\mathbf{P_0^2})\\ +\ &2(\mathbf{P_0^1}\wedge(\mathbf{P_0^1}+\mathbf{P_0^1P_1^1}) + (\mathbf{P_0^1}+\mathbf{P_0^1P_1^1})\wedge(\mathbf{P_0^1}+\mathbf{P_0^1P_2^1})\\ &+(\mathbf{P_0^1}+\mathbf{P_0^1P_2^1})\wedge\mathbf{P_0^1})\ \Big]\,,\end{aligned}$$

thus:

$$\begin{aligned}\tilde{\boldsymbol{\eta}}^2(T) = \tfrac{1}{12} \Big[\quad & 2(\mathbf{P_0^2} \wedge \mathbf{P_0^2P_1^2} + \mathbf{P_0^2} \wedge \mathbf{P_0^2P_2^2} + \mathbf{P_0^2P_1^2} \wedge \mathbf{P_0^2} + \mathbf{P_0^2P_1^2} \wedge \mathbf{P_0^2P_2^2} \\ & + \mathbf{P_0^2P_2^2} \wedge \mathbf{P_0^2}) \\ + \quad & (3\mathbf{P_0^2} \wedge \mathbf{P_0^1} + \mathbf{P_0^2} \wedge \mathbf{P_0^1P_1^1} + \mathbf{P_0^2} \wedge \mathbf{P_0^1P_2^1} + \mathbf{P_0^2P_1^2} \wedge \mathbf{P_0^1} \\ & + \mathbf{P_0^2P_1^2} \wedge \mathbf{P_0^1P_2^1} + \mathbf{P_0^2P_2^2} \wedge \mathbf{P_0^1}) \\ + \quad & (3\mathbf{P_0^1} \wedge \mathbf{P_0^2} + \mathbf{P_0^1} \wedge \mathbf{P_0^2P_1^2} + \mathbf{P_0^1} \wedge \mathbf{P_0^2P_2^2} + \mathbf{P_0^1P_1^1} \wedge \mathbf{P_0^2} \\ & + \mathbf{P_0^1P_1^1} \wedge \mathbf{P_0^2P_2^2} + \mathbf{P_0^1P_2^1} \wedge \mathbf{P_0^2}) \\ + \quad & 2(\mathbf{P_0^1} \wedge \mathbf{P_0^1P_1^1} + \mathbf{P_0^1} \wedge \mathbf{P_0^1P_2^1} + \mathbf{P_0^1P_1^1} \wedge \mathbf{P_0^1} + \mathbf{P_0^1P_1^1} \wedge \mathbf{P_0^1P_2^1} \\ & + \mathbf{P_0^1P_2^1} \wedge \mathbf{P_0^1}) \quad \Big] .\end{aligned}$$

All simplifications done,

$$\tilde{\boldsymbol{\eta}}^2(T) = \frac{1}{12} \left(2\mathbf{P_0^2P_1^2} \wedge \mathbf{P_0^2P_2^2} + 2\mathbf{P_0^1P_1^1} \wedge \mathbf{P_0^1P_2^1} + \mathbf{P_0^2P_1^2} \wedge \mathbf{P_0^1P_2^1} + \mathbf{P_0^1P_1^1} \wedge \mathbf{P_0^2P_2^2} \right) . \quad (2.15)$$

Let us now use this result to compute the volumes swept by cells' interfaces, made up of several triangles.

Volume swept by inner interfaces

The interface of a finite volume cell is made up of several triangles, connecting the middle of an edge to the center of gravity of a face and the center of gravity of that tetrahedron, see Figure 2.2. The two triangles of the interface sharing one edge within a tetrahedron are coplanar (*i.e.* the middle of an edge, the center of gravity of the two faces neighboring this edge and the center of gravity of tetrahedron are coplanar). The union of these two triangles is called the facet associated to the edge and the tetrahedron.

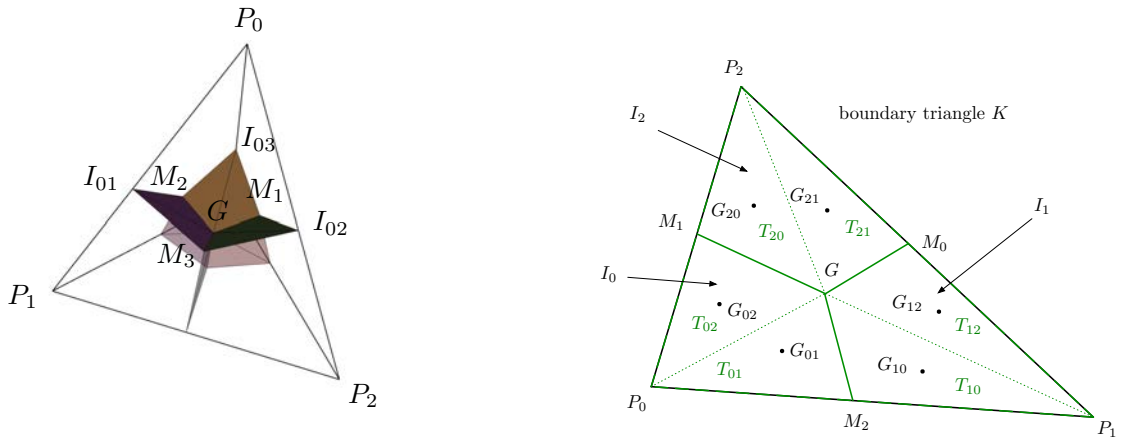

Figure 2.2: Interface of a finite volume cell made up of facets (left) and boundary interface (right).

At configuration $t^s = t^0 + c_s\,\tau$, (with $t^0 = t^n$), the outward non-normalized normal $\boldsymbol{\eta}^s_{ij}$ and the volume swept A^s_{ij} are computed as described in [Nkonga 1994]. As the cell interface is made up of several facets, the total swept volume is the sum of the volumes swept by each facet.

Let us assume that the facet considered is associated with edge $\mathbf{e}_{ij} = P_iP_j$ and belongs to tetrahedron $K = (P_0, P_1, P_2, P_3)$. In what follows, $i \neq j \neq l \neq m \in [\![0,3]\!]$, G denotes the center of gravity of tetrahedron K, M_m denotes the gravity center of face $F_m = (P_i, P_j, P_l)$ of tetrahedron K and M_l the center of gravity of face $F_l = (P_i, P_j, P_m)$. The outward non-normalized normal of the facet is given by:

$$\tilde{\boldsymbol{\eta}}^s_{ij,K} = \frac{1}{4}\mathbf{G^0M^0_m} \wedge \mathbf{G^sM^s_l} + \frac{1}{4}\mathbf{G^sM^s_m} \wedge \mathbf{G^0M^0_l} + \frac{1}{2}\mathbf{G^0M^0_m} \wedge \mathbf{G^0M^0_l} + \frac{1}{2}\mathbf{G^sM^s_m} \wedge \mathbf{G^sM^s_l}\,,$$

where

$$\mathbf{G^0M^0_m} = \frac{1}{12}\left(P^0_i + P^0_j - 3P^0_m + P^0_l\right)\,, \quad \mathbf{G^sM^s_m} = \frac{1}{12}\left(P^s_i + P^s_j - 3P^s_m + P^s_l\right)\,,$$

$$\mathbf{G^0M^0_l} = \frac{1}{12}\left(P^0_i + P^0_j + P^0_m - 3P^0_l\right) \quad \text{and} \quad \mathbf{G^sM^s_l} = \frac{1}{12}\left(P^s_i + P^s_j + P^s_m - 3P^s_l\right)\,.$$

The volume swept by the facet is:

$$A^s_{ij,K} \;=\; c_s\,\tau\,(\boldsymbol{w}_G)^s_{ij,K} \cdot \tilde{\boldsymbol{\eta}}^s_{ij,K}\,, \tag{2.16}$$

with the mean velocity written as:

$$(\boldsymbol{w}_G)^s_{ij,K} = \frac{1}{36\,c_s\,\tau}\left(13\boldsymbol{w}^s_i + 13\boldsymbol{w}^s_j + 5\boldsymbol{w}^s_m + 5\boldsymbol{w}^s_l\right) \quad \text{with} \quad \boldsymbol{w}^s_\xi = P^s_\xi - P^0_\xi\,.$$

Finally, the total volume swept by the interface around edge e_{ij} of cell C_i is obtained by summing over the shell of the edge, *i.e.* all the tetrahedra sharing the edge:

$$A^s_{ij} = \sum_{K \in Shell(i,j)} A^s_{ij,K}\,. \tag{2.17}$$

It is important to understand that normals $\tilde{\boldsymbol{\eta}}^s_{ij,K}$ are pseudo-normals which are used only to compute the volumes swept by the facets. They must not be mistaken for normals to facets taken at t^s, $\boldsymbol{\eta}^s_{ij,K}$, which are used for the computation of ALE fluxes.

Volumes swept by boundary interfaces

The pseudo-normals and swept volumes of boundary faces are computed in a similar way. Let $K = (P_0\,, P_1\,, P_2)$ be a boundary triangle, as in Figure 2.2. Let M_0, M_1 and M_2 be the middles of the edges and G the center of gravity of K. The triangle is made up of three quadrangular finite volume interfaces: $(P_0\,, M_2\,, G\,, M_1)$ associated with cell C_0, $(P_1\,, M_0\,, G\,, M_2)$ with C_1 and $(P_2\,, M_1\,, G\,, M_0)$ with C_2. Each boundary interface I_i is made of two sub-triangles, noted T_{ij} and T_{ik} with $j, k \neq i$.

The volume swept by interface I_i between the initial and current Runge-Kutta configurations is the sum of the volumes swept by its two sub-triangles:

$$A^s_{i,K} = A^s_{T_{ij}} + A^s_{T_{ik}} \;=\; c_s\tau\,\boldsymbol{w}^s_{G_{ij}} \cdot \tilde{\boldsymbol{\eta}}^s_{ij} \;+\; c_s\tau\,\boldsymbol{w}^s_{G_{ik}} \cdot \tilde{\boldsymbol{\eta}}^s_{ik}\,, \tag{2.18}$$

where $\boldsymbol{w}^s_{G_{ij}}$ is the velocity of the center of gravity G_{ij} of triangle T_{ij} and $\tilde{\boldsymbol{\eta}}^s_{ij}$ is the pseudo-normal associated with T_{ij}, computed between the initial and current Runge-Kutta configurations.

The six triangles T_{ij} are coplanar, so their pseudo-normals have the same direction. Moreover, as median cells are used, their pseudo-normals also have the same norm, which is equal to one sixth of the norm of the pseudo-normal to triangle K. This common pseudo-normal is therefore equal to :

$$\tilde{\boldsymbol{\eta}}^s \;=\; \frac{1}{6}\tilde{\boldsymbol{\eta}}^s_K \;=\; \frac{1}{6}\cdot\frac{1}{3}\left[\frac{1}{4}\mathbf{P_0^0P_1^0}\wedge\mathbf{P_0^sP_2^s}+\frac{1}{4}\mathbf{P_0^sP_1^s}\wedge\mathbf{P_0^0P_2^0}+\frac{1}{2}\mathbf{P_0^0P_1^0}\wedge\mathbf{P_0^0P_2^0}+\frac{1}{2}\mathbf{P_0^sP_1^s}\wedge\mathbf{P_0^sP_2^s}\right], \tag{2.19}$$

thus

$$A^s_{i,K} \;=\; c_s\tau\left[\boldsymbol{w}^s_{G_{ij}}+\boldsymbol{w}^s_{G_{ik}}\right]\cdot\tilde{\boldsymbol{\eta}}^s\,, \tag{2.20}$$

where :

$$\boldsymbol{w}^s_{G_{ij}}=\frac{1}{18\,c_s\,\tau}\left(11\boldsymbol{w}^s_i+5\boldsymbol{w}^s_j+2\boldsymbol{w}^s_k\right)\quad\text{with}\quad\boldsymbol{w}^s_\xi=P^s_\xi-P^0_\xi\,.$$

Finally, the total volume swept by the boundary interface is:

$$A^s_i=\sum_{K\in Ball(i)}A^s_{i,K}\,. \tag{2.21}$$

2.3.6 MUSCL approach and RK schemes

As regards spatial accuracy, we have seen that the order of accuracy can be enhanced using the MUSCL technique coupled with a limiter, thereby guaranteeing that this ALE scheme is intrinsically TVD. However, one must determine how to compute upwind/downwind gradients which are necessary for the V4-schemes. Still following [Yang 2005], it is clear that preserving the expected order of accuracy in time imposes that the upwind/downwind elements are computed on the current Runge-Kutta configuration, *i.e.,* on the mesh at t^s.

2.3.7 Computation of the time step

The maximal allowable time step for the numerical scheme is:

$$\tau(P_i)=\frac{h(P_i)}{c_i+\|\mathbf{u}_i-\boldsymbol{w}_i\|} \tag{2.22}$$

where $h(P_i)$ is the smallest height in the ball of vertex P_i, c_i is the speed of sound, $\mathbf{u}_i$ is the Eulerian speed (the speed of the fluid, computed by the solver) and $\boldsymbol{w}_i$ is the speed of the mesh vertex. The global time step is then given by $\tau = CFL\ \min_{P_i}(\tau(P_i))$.

2.3.8 Handling the swaps

An ALE formulation of the swap operator, satisfying the DGCL, was proposed in 2D in [Olivier 2011b], but its extension to 3D is somewhat delicate because it requires to handle 4D geometry, and it has not been carried out yet. Instead of considering an ALE scheme

for the swap operator, our choice in this work is to perform the swaps between two solver iterations, *i.e.* at a fixed time t^n. This consequently means that during the swap phase, the mesh vertices do not move, and thus the swaps do not impact the ALE parameters η and $\boldsymbol{w}$, unlike [Olivier 2011b] where the swaps are performed during the solver iteration, *i.e.* between t^n and t^{n+1}. After each swap step, a linear interpolation is performed to recover the solution. In other words, the solution at the vertices does not change, which is not conservative. The data of the finite volume cells (volume and interface normals) are then updated, together with the topology of the mesh (edges and tetrahedra). This approach however is DGCL compliant, since the constant state is preserved (in fact, any linear state is preserved). A better approach would be to use conservative interpolation [Alauzet 2010b] which is conservative and DGCL compliant and the best would be to develop the ALE formulation of the generalized swap in 3D. In Section 3.1.1, we will quantify the error due to the use of swaps in numerical simulations.

2.4 FSI coupling

The moving boundaries can have an imposed motion, or be driven by fluid-structure interaction. A simple solid mechanics solver is coupled to the flow solver described previously. The chosen approach is the 6-DOF (6 Degrees of Freedom) approach for rigid bodies.

2.4.1 Movement of the geometries

In this work, the bodies are assumed to be rigid, of constant mass and homogeneous, *i.e.,* their mass is uniformly distributed in their volume. The bodies we consider will never break into different parts. Each rigid body B is fully described by:

Physical quantities: its boundary ∂B and its associated inward normal $\mathbf{n}$, its mass m assumed to be constant, its $d \times d$ matrix of inertia $\mathcal{J}_G$ computed at G which is symmetric and depends only on the shape and physical nature of the solid object[2].

Kinematic quantities: the position of its center of gravity $\mathbf{x_G} = (x(t), y(t), z(t))$, its angular displacement vector $\boldsymbol{\theta} = \boldsymbol{\theta}(t)$ and its angular speed vector $\mathrm{d}\boldsymbol{\theta}/\mathrm{d}t$.

$\mathbf{F}_{ext}$ denotes the resultant vector of the external forces applied on B, $\mathbf{M}_G(\mathbf{F}_{ext})$ the kinetic moment of the external forces applied on B computed at G and $\mathbf{g}$ the gravity vector. We assume that the bodies are only submitted to forces of gravity and fluid pressure. The equations for solid dynamics in an inertial frame then read:

$$\left\{ \begin{array}{rcll} m\dfrac{\mathrm{d}^2\mathbf{x_G}}{\mathrm{d}t^2} & = & \mathbf{F}_{ext} & = \displaystyle\int_{\partial B} p(\boldsymbol{s})\mathbf{n}(\boldsymbol{s})\mathrm{d}\boldsymbol{s} + m\mathbf{g} \\ \mathcal{J}_G\dfrac{\mathrm{d}^2\boldsymbol{\theta}}{\mathrm{d}t^2} & = & \mathbf{M}_G(\mathbf{F}_{ext}) & = \displaystyle\int_{\partial B} \big[(\boldsymbol{s} - \mathbf{x_G}) \times p(\boldsymbol{s})\mathbf{n}(\boldsymbol{s})\big]\mathrm{d}\boldsymbol{s}\,. \end{array} \right. \tag{2.23}$$

[2]The moment of inertia relative to an axis of direction $\mathbf{a}$ passing through G where $\mathbf{a}$ is an arbitrary unit vector is given by: $J_{(G\mathbf{a})} = \mathbf{a}^T \mathcal{J}_G \mathbf{a}$

2.4.2 Discretization

The equation governing the position of the center of gravity of the body is easy to solve since it is linear. Its discretization is straightforward. However, the discretization of the second equation, which controls the orientation of the body, is more delicate. Since the matrix of inertia $\mathcal{J}_G$ depends on $\boldsymbol{\theta}$, it is a non-linear second order ODE. The chosen discretization is extensively detailed in [Olivier 2011a] and is based on rewriting of the equations in the frame of the moving body.

2.4.3 Explicit coupling

As the geometry must be moved in accordance with the fluid computation, the same time integration scheme has been taken to integrate the fluid and the solid equations. Therefore, time-advancing of the rigid bodies ODE System is performed using the same RKSSP scheme as the one used to advance the fluid numerical solution. The coupling is loose and explicit as the external forces and moments acting on rigid objects are computed on the current configuration.

2.5 Implementation

2.5.1 Non-dimensionalization

For some aeronautics inspired cases, better convergence of the solver is observed if the physical quantities are non-dimensioned. Solving the non-dimensioned equations is also necessary when considering models scaled to lower dimensions. Details on the non-dimensionalization of the Euler equations are given in [Alauzet 2014b].

When considering FSI problems, it seems crucial to be coherent and non-dimensionalize the solids equations too. To that end, the following reference quantities are defined: a density ρ_∞ a length L_∞, a mass M_∞, an acceleration of gravity g_∞ and a moment of inertia I_∞, such that:

$$\begin{aligned} M_\infty &= \frac{\rho_\infty}{L_\infty^3} \\ g_\infty &= \frac{a_\infty^2}{L_\infty} \\ I_\infty &= M_\infty L_\infty^2 \,, \end{aligned} \tag{2.24}$$

and ρ_∞ and L_∞ are given as input (or computed using aeronautic considerations, as explained in [Alauzet 2014b]). The corresponding non-dimensioned variables $\tilde{x}$ are then written as:

$$\tilde{x} = \frac{x}{x_\infty} \,, \tag{2.25}$$

where x is the regular dimensioned value.

2.5.2 Parallelization

Most parts of the code are parallelized for shared memory multi-core computers with a p-threads approach using a semi-automatic p-thread parallelization library [Maréchal 2016] coupled with a space filling curve renumbering strategy [Alauzet 2009].

The automatic parallelization library cuts the data into small chunks that are compact in terms of memory (because of the renumbering), then uses a dynamic scheduler to allocate the chunks to the threads to limit concurrent memory accesses. In the case of a loop performing the same operation on each entry of a table, the loop is split into many sub-loops. Each sub-loop will perform the same operation (symmetric parallelism) on equally-sized portions of the main table and will be concurrently executed. It is the scheduler's job to make sure that two threads do not write on the same memory location simultaneously. When indirect memory accesses occur in loops, memory write conflicts can still arise. To deal with this issue, an asynchronous parallelization is considered rather than of a classic gather/scatter technique, *i.e.*, each thread writes in its own working array and then the data are merged.

However, with this approach, a subtle management of cache misses and cache-line overwrites is required to avoid drops in scaling factors. Space filling curves exhibit clustering properties very helpful for renumbering algorithms [Sagan 1994]. The Hilbert space filling curve based renumbering strategy is used to map mesh geometric entities, such as vertices, edges, triangles and tetrahedra, into a one dimensional interval. In numerical applications, it can be viewed as a mapping from the computational domain onto the memory of a computer. The local property of the Hilbert space filling curve implies that entities which are neighbors on the memory 1D interval are also neighbors in the computational domain. Therefore, such a renumbering strategy has a significant impact on the efficiency of a code. We can state the following benefits: it reduces the number of cache misses during indirect addressing loops, and it reduces cache-line overwrites during indirect addressing loops, which is fundamental for shared memory parallelism.

2.6 Conclusion

In this chapter, the 3D ALE solver for the compressible Euler equations that was implemented in three dimensions and used during this thesis was presented. It is based on a Finite Volume approach, with an HLLC approximate Riemann solver, and high-order schemes using a MUSCL gradient reconstruction technique. As regards time discretization, multi-step Runge-Kutta schemes are derived from the enforcement of the Geometric Conservation Law. Details on the computation of the geometric parameters are provided. A 6-DOF mechanical model for rigid bodies was considered to add FSI coupling to the fluid solver. This solver was implemented in 3D in the same code as the moving mesh algorithm, and was plugged into it as explained in Algorithm 2. In the following section, we are going to present numerous cases using this ALE solver and the connectivity-change moving mesh method.

Chapter 3

Numerical Examples

In this chapter, the strategy described previously is applied to a variety of test cases, in order to validate the chosen approach and to prove its efficiency. In the first section, we aim at validating the ALE solver, study its convergence both in space and time, and at validating the FSI coupling. Then we discuss the parallel performance of the solver. Finally, we present more complex ALE test cases from the aeronautic industry.

The works presented in this Chapter were published in [Barral 2014a, Barral 2014b].

3.1 Validation of the solver

3.1.1 Static vortex in a rotating mesh

The first test case is used to validate the ALE solver, and study the impact of the moving mesh on the accuracy of the solution. We study the conservation of a static vortex (similar to [Kitamura 2011]) on a rotating mesh.

This case is an extension of a 2D case: we consider an extruded cylinder, and the initial solution is constant along the z axis. The domain is a cylinder of radius 5 and of height 1. The initial solution is defined as follows. Let $r_0 = 1$ be a characteristic radius and the reference state be:

$$\rho_\infty = 1,\ u_{x\infty} = 0,\ u_{y\infty} = 0,\ u_{z\infty} = 0,\ p_\infty = 1\,. \tag{3.1}$$

Let us define the following quantities:

$$D_\infty = \frac{1}{2}\frac{1}{(2\pi)^2}\frac{\rho_\infty}{p_\infty}, \qquad B_\infty = 2r_0^2 - \frac{\gamma-1}{\gamma}D_\infty, \qquad E_\infty = (2r_0^2)^2 - B_\infty^2\,.$$

For a point of cylindrical coordinates (r, θ, z), the enthalpy and speed of the vortex are:

$$H_{\text{vor}} = \frac{\gamma}{\gamma-1}\frac{p_\infty}{\rho_\infty} + \frac{1}{2}(u_{x\infty}^2 + u_{y\infty}^2 + u_{z\infty}^2), \qquad v_{\text{vor}} = \frac{1}{2\pi}\frac{r}{r^2+1}\,.$$

And finally the initial solution is:

$$\begin{cases} u_{x0} &= u_{x\infty} + v_{\text{vor}}\sin\theta \\ u_{y0} &= u_{y\infty} + v_{\text{vor}}\cos\theta \\ u_{z0} &= 0 \\ p_0 &= p_\infty \exp\left(\frac{2D_\infty}{\sqrt{E_\infty}}\ \text{atan}\left(\frac{2r^2+B_\infty}{\sqrt{E_\infty}}\right) - \frac{\pi}{2}\right) \\ \rho_0 &= \dfrac{\frac{\gamma}{\gamma-1}p_0}{H_{\text{vor}} - \frac{1}{2}v_{\text{vor}}}\,. \end{cases} \tag{3.2}$$

This initial configuration results in a cylindrical vortex rotating around the z axis, that remains in a constant state over time (density, pressure and speed are preserved). To avoid errors due to boundary conditions, the exact solution is enforced for every vertex farther than 90% of the cylinder radius (typically just a few layers of elements). The mesh is rotated rigidly around the z axis. Several rotation speeds were considered, all of which led to the same conclusions. Here, we only present the results for an angular speed of 0.34^o per time unit, which corresponds approximately to the speed of the vortex at a radius $r = 5$. We address both space and time convergence.

Space convergence

For the space convergence study, we consider a set of uniform meshes of sizes varying from $10,000$ vertices to 1.8 million vertices, see Table 3.1. The simulation is run until time $t = 540$, which corresponds to half a revolution of the cylinder. Since an exact solution is known, it is easy to compute an error with respect to this exact solution on each mesh. Two cases are compared: in one case the mesh is fixed, and in the other case the mesh is rotated as described previously. Since no mesh optimizations are performed, it allows us to make sure that the extra ALE terms are well computed. The results are shown in Fig. 3.1. On the left graph, we plot the error varying with time for the different meshes, fixed (red) and rotating (blue). We can see that, as expected, the error diminishes as the size of the mesh grows, but more significantly the curves of the error for the moving and static meshes are almost superimposed. This confirms the accuracy of the ALE scheme. To study the spatial order of convergence, the error is plotted with respect to the mesh size at different times on the right-hand graph. We can see that the order of convergence reached is slightly greater than 2, which is coherent with the space scheme used.

	Mesh 1	Mesh 2	Mesh 3	Mesh 4	Mesh 5	Mesh 6	Mesh 7	Mesh 8
# vertices ($\times 10^3$)	12	44	138	256	371	585	965	$1,825$
# tetrahedra ($\times 10^3$)	48	211	691	$1,330$	$1,965$	$3,122$	$5,287$	$10,218$
# time steps	$4,673$	$7,280$	$15,608$	$14,321$	$17,275$	$29,996$	$32,245$	$44,680$

Table 3.1: Test case of the static vortex without swaps. Sizes of the meshes used and number of solver time steps performed.

Time convergence

This study was carried out in 2D for reasons of efficiency. Our goal was to study the impact of the Runge-Kutta schemes used and the CFL parameter. We consider a disc of radius 5, with the same initial solution as previously (Eq. (3.2)) and rotating with the same speed. Three time discretizations are used: a standard first order explicit scheme (SSPRK(1,1)), a five-step second order Runge-Kutta scheme (SSPRK(5,2)) and a four-step third order Runge-Kutta scheme (SSPRK(4,3)) described in Section 2.3.4. Four CFL numbers are set for each case: $CFL_{max}, 3/4 \times CFL_{max}, 1/2 \times CFL_{max}$ and $1/10 \times CFL_{max}$. CFL_{max} is respectively 1, 4 and 2 for the schemes SSPRK(1,1), SSPRK(5,2) and SSPRK(4,3).

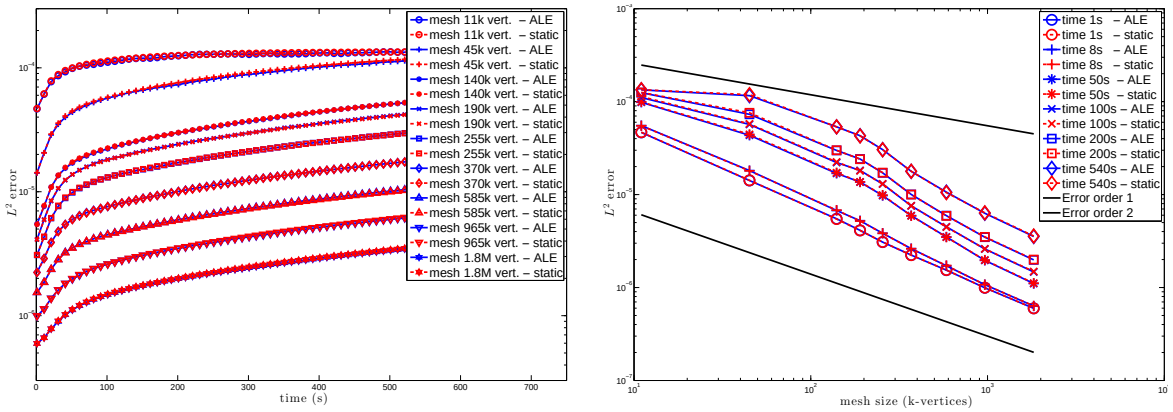

Figure 3.1: Test case of the static vortex (3D). Error curves comparing the simulation on a fixed mesh (red) and on a moving mesh (blue) for several mesh sizes. Left, the evolution of the error over time, right convergence curves with respect to the size of the mesh.

The results are gathered in Fig. 3.2. On the first graph, the error over time is plotted for the different schemes and CFL numbers, while on the second, we plot the error varying with the CFL number at different time steps. On both graphs, we can see that the error decreases with the order of the Runge-Kutta scheme and with the CFL number. Whereas the standard explicit scheme is very sensitive to the time step, the error does not vary much for the SSPRK(4,3) and SSPRK(5,2). What is interesting to notice is that we have to use a CFL number of $0.1 \times CFL_{max}$ for a standard explicit scheme to reach the same level of accuracy as the SSPRK(4,3) scheme with a CFL number at its maximum, or the the SSPRK(5,2) scheme with a CFL number of 0.75 times its maximum. To reach a non-dimensional time equal to 3 with the standard explicit scheme with a CFL of 0.1, 30 time steps are required, whereas only 5 are required for the SSPRK(5,2) scheme with a CFL of 3, and 6 for the SSPRK(4,3) with a CFL of 2. Thus we can go 6 times faster with the high-order Runge-Kutta schemes and still obtain the same solution accuracy, which is obviously interesting.

Influence of swaps

It is important to study the impact of edge/face swaps on the solution accuracy in 3D as we do not a have specific ALE treatment for them. However, it is difficult to set up a meaningful test case with swaps. The number of swaps should be constant (in proportion to the size of the mesh), and they should also be uniformly distributed in both time and space. Moreover, if you force swaps on good quality tetrahedra, the quality of the resulting tetrahedra can degrade significantly and affect substantially the accuracy of the solution. On all the simulations run with the connectivity-change moving mesh strategy, we noticed that, on average, less than one swap per $10,000$ tetrahedra and per time step is performed. To fit this observation, in this example, we swap all the bad elements to preserve the mesh quality every 15 solver time steps. Then, if not enough swaps were performed, we randomly swap elements to reach a number of 1 swap per 666 tetrahedra. Not all swaps are actually performed, because some of them affect the quality of the mesh too drastically, so the number of swaps is not perfectly controlled and

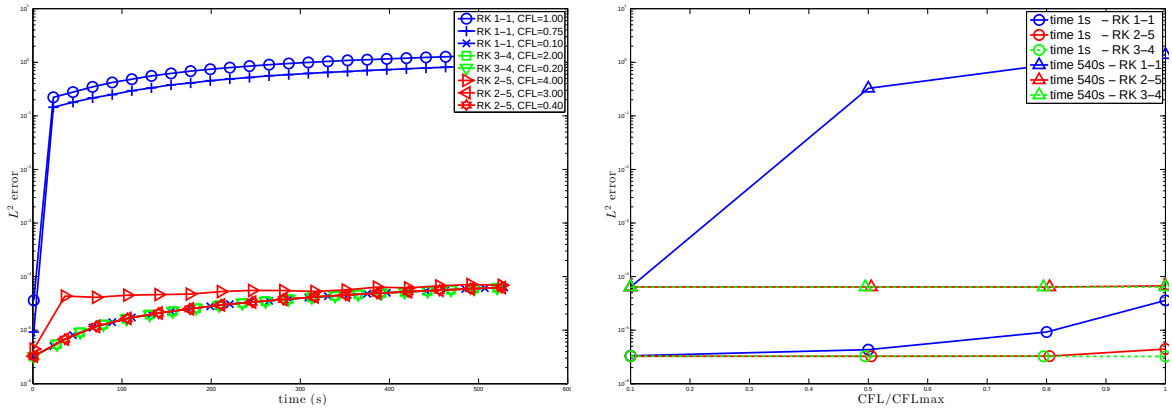

Figure 3.2: Test case of the static vortex (2D). Error curves comparing several temporal schemes: standard explicit Euler scheme (blue), SSPRK(4,3) (green), SSPRK(5,2) (red). Left, evolution of the error over time and right, error at a fixed time varying with the CFL number.

may vary at each mesh optimization step, but this method allows us to have an average number of swaps close to one per 10,000 tetrahedra and per time step for all the meshes, as stated in Table 3.2.

	Mesh 1	Mesh 2	Mesh 3	Mesh 4	Mesh 5	Mesh 6	Mesh 7
# tetrahedra ($\times 10^3$)	48	211	942	1,965	3,122	5,287	10,218
# time steps	2,338	3,647	7,493	8,948	12,040	16,085	21,449
# swaps ($\times 10^3$)	21	114	996	2,319	5,204	10,838	26,719
# swaps/timestep/tet($\times 10^{-4}$)	1.80	1.49	1.41	1.32	1.38	1.27	1.22

Table 3.2: Test case of the static vortex with swaps. Number of swaps for each mesh. On the last line, the number of swaps is close to one swap per time step and per 10,000 tetrahedra for all the meshes.

We again consider 3D meshes, rotating with the same speed of 0.34^o per time unit, and the simulations are run up to 270s. The results are gathered in Fig. 3.3. The error is slightly higher with swaps (in green) than without (in blue and red), however the error curves remain very close to those without swaps. The discrepancy can be explained partly by the quality of the mesh that decreases due to the forced swaps. But the main reason is because no specific treatment is applied to the swaps, so the conservation laws are violated. A conservative treatment is required, such as the 2D ALE formulation of the swap described in [Olivier 2011b], but the 3D version has not yet been developed, or a conservative interpolation such as the one from [Alauzet 2010b]. Even without such a formulation, the error introduced in this example is not so large, and the same second order convergence rate is observed, which validates our approach.

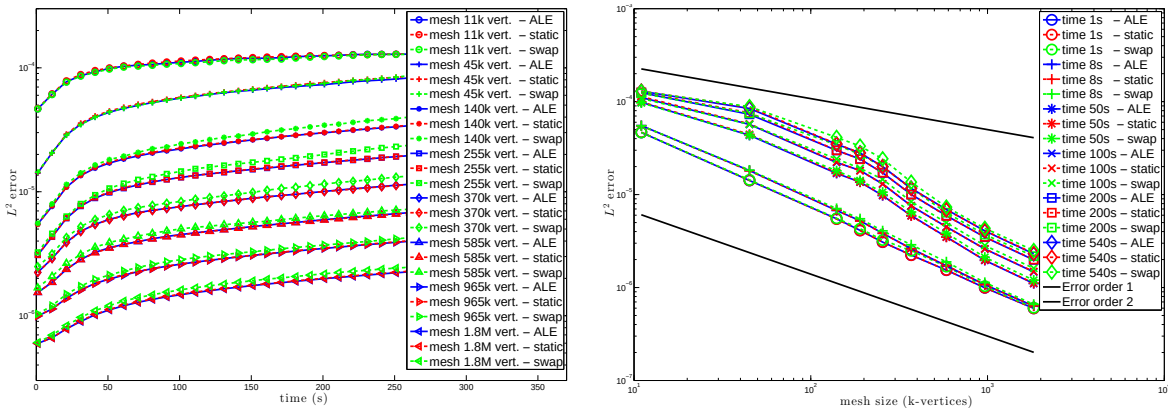

Figure 3.3: Test case of the static vortex (3D). Error curves comparing the simulation on a fixed mesh (red), on a moving mesh without swaps (blue) and on a moving mesh with swaps (green) for several mesh sizes. Left, the evolution of the error over time, right convergence curves with respect to the size of the mesh.

Conclusion

To sum up, this static vortex test case validates our ALE solver. We asymptotically reach an order of convergence of 2 for the spatial error, and we made sure there was no additional error introduced by the ALE terms. As regards temporal convergence, the importance of a high order Runge-Kutta scheme was established. Finally, we showed that edge/face swapping only creates a small error, even if used without some specific conservative treatment. Moreover, swaps are mandatory to preserve the quality of the mesh (and thus the accuracy of the solution) when complex geometric displacements are involved.

3.1.2 Flat plate in free fall

This case is run to validate the rigid body dynamics in the FSI solver. We consider a flat plate of length l, width l and height h. The top and bottom sides are split into four quadrants, see Figure 3.4. Different pressure conditions are imposed on each quadrant, and the plate is in free-fall. The plate is at rest at the beginning of the simulation and we set the pressure to $p_0 = 1$ everywhere on the plate, except on two opposite quadrants where we set it to $p_1 = p_0 + 1 = 2$. The pressure on the body is imposed element-wise throughout the simulation and the pressure of the surrounding fluid is not taken into account. The only other force is gravity. We expect the plate to spin around its corresponding diagonal axis and fall due to gravity.

An exact solution to this problem can be computed, using Newton's second law for the translation motion and the conservation of momentum on the rotation axis. The translation motion is $z(t) = z(0) - \frac{1}{2}gt^2$, where g is the gravity vector norm. The rotation axis (Δ) is obviously $(G, \mathbf{u})$, where G is the center of gravity of the plate and the unit direction vector $\mathbf{u}$ is $\frac{1}{\sqrt{2}}(1, 1, 0)$. The momentum of inertia of the plate on this axis is: $J_\Delta = \frac{1}{12}m(l^2 + h^2)$. For symmetry reasons, we can consider only the extra pressure on the two quadrants with a greater pressure (the constant pressure p_0 cancels itself out everywhere). The moment of the pressure

forces is thus $M_{F/\Delta} = ((\mathbf{GA_1} \wedge \mathbf{P_1}) + (\mathbf{GA_2} \wedge \mathbf{P_2}))\,.\mathbf{u}$, where A_1 and A_2 are the centers of the quadrants considered, $\mathbf{P_1} = \left(\frac{l}{2}\right)^2 \mathbf{e_z}$ and $\mathbf{P_2} = -\left(\frac{l}{2}\right)^2 \mathbf{e_z}$ are the pressure forces. Finally $M_{F/\Delta} = -\frac{l^3}{8}$. The conservation of momentum law on the axis (Δ) is written:

$$J_\Delta \frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = M_{F/\Delta}\,,$$

thus

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = cst = \frac{3l}{2m(l^2+h^2)}\,,$$

and thus

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \frac{3l}{2m(l^2+h^2)}t \quad \text{and} \quad \theta(t) = \frac{1}{2}\frac{3l}{2m(l^2+h^2)}t^2\,.$$

Comparison between the theoretical solutions and computed solutions are shown in Figure 3.5, where it can be seen that both curves are the same, and only diverge slowly over time, due to discrete integration errors.

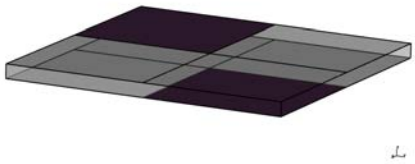

Figure 3.4: Test case of the plate. Initial conditions of pressure on the plate: the pressure is set to 1 everywhere, except on dark red areas where it is set to 2.

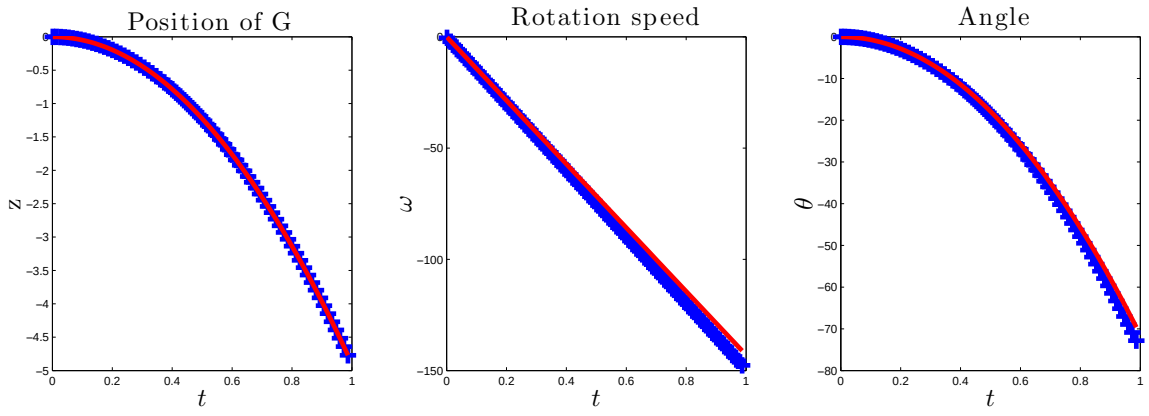

Figure 3.5: Test case of the plate. Comparison between the theoretical values (red) and computed values (blue crosses) of the position of the center of gravity (left), the rotation speed (middle) and the angle (right).

3.1.3 Piston

The third validation test case is an FSI piston case from [Lefrançois 2010]. It is originally a 1D problem, which is extended to 3D. As shown in Fig. 3.6, we consider a gas contained in a 3D cylindrical chamber, closed at one end by a wall, and at the other end by a moving piston of mass m_p and of cross-section A_p. Besides the forces of pressure, the piston is submitted to a restoring force modeled by a spring of rigidity k_p. A natural frequency of the piston can be defined by $f_p = \frac{1}{2\pi}\sqrt{\frac{k_p}{m_p}}$. Let L_0 be the length of the chamber at rest, and $u(t)$ be the displacement of the piston with respect to the position at rest. The gas is initially at rest, and the piston is held in position u_0. We consider that all the variables are uniform on each cross-section of the cylinder (1D assumption).

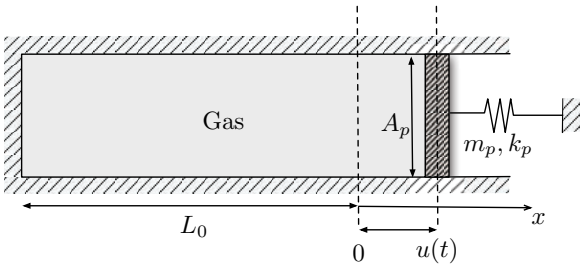

Figure 3.6: Piston test case. 2D representation of the problem.

It is shown in [Lefrançois 2010] that the piston is submitted to a resultant force applied to its center of gravity:

$$F_p = -k_p * u(t) + A_p(p(t) - p(0))\,, \tag{3.3}$$

with $p(t)$ being the pressure on the piston. No effects of gravity are taken into account.

In practice, the domain at $t = 0$ is a rectangular box of dimensions $[-1, 0.2] \times [-0.1, 0.1] \times [-0.1, 0.1]$. The vertical face at $x = 0.2$ is mobile, and all the other faces are fixed, which corresponds to $l_0 = 1$ and $u_0 = 0.2$. The piston rigidity is set to $k_p = 10^7\ N.m^{-1}$, and a variety of piston masses (and thus frequencies) are considered: $m_p = 10\ kg, 100\ kg, 1000\ kg$, corresponding respectively to $f_p = 159\ Hz, 50\ Hz, 16\ Hz$. The other parameters are: $\rho_0 = 1.1612\ kg.m^{-3}$, $p_0 = 10^5\ Pa$, $\mathbf{u}_0 = 0\ m.s^{-1}$. Slipping conditions are imposed on all the boundary faces (fixed and mobile). From a practical point of view, since this is a 1D case extended to 3D, the code of the FSI solver has to be modified to avoid 3D effects: the speeds in the tangential directions are forced to 0 at every time step, as is the rotation matrix. From a moving mesh point of view, it is essential to allow tangential movement of the vertices on the fixed faces of the box.

The quantities of interest are the displacement and the pressure of the piston. For the pressure, we consider the pressure at its center of gravity. These two quantities are plotted in Fig. 3.7. Regarding the piston movement, it is periodic, as expected, with a period $T_p = 1/f_p$ (see Table 3.3). The attenuation of the amplitude due to diffusion is negligible on two periods. As concerns the piston pressure, the results also correspond to [Lefrançois 2010]. We ran the case with several mesh sizes (from 200,000 to one million vertices) and several temporal schemes, and all results are consistent. The pressure fields at a time around 2/5 of the final time are shown in Fig. 3.8 for the three masses considered. We can see that the 1D structure of the

solution is well preserved.

m_p (kg)	10	100	1000
f_p (Hz)	159	50	16
T_p (s)	0.00628	0.0199	0.0628

Table 3.3: Piston test case. Natural frequencies and periods for the different masses used.

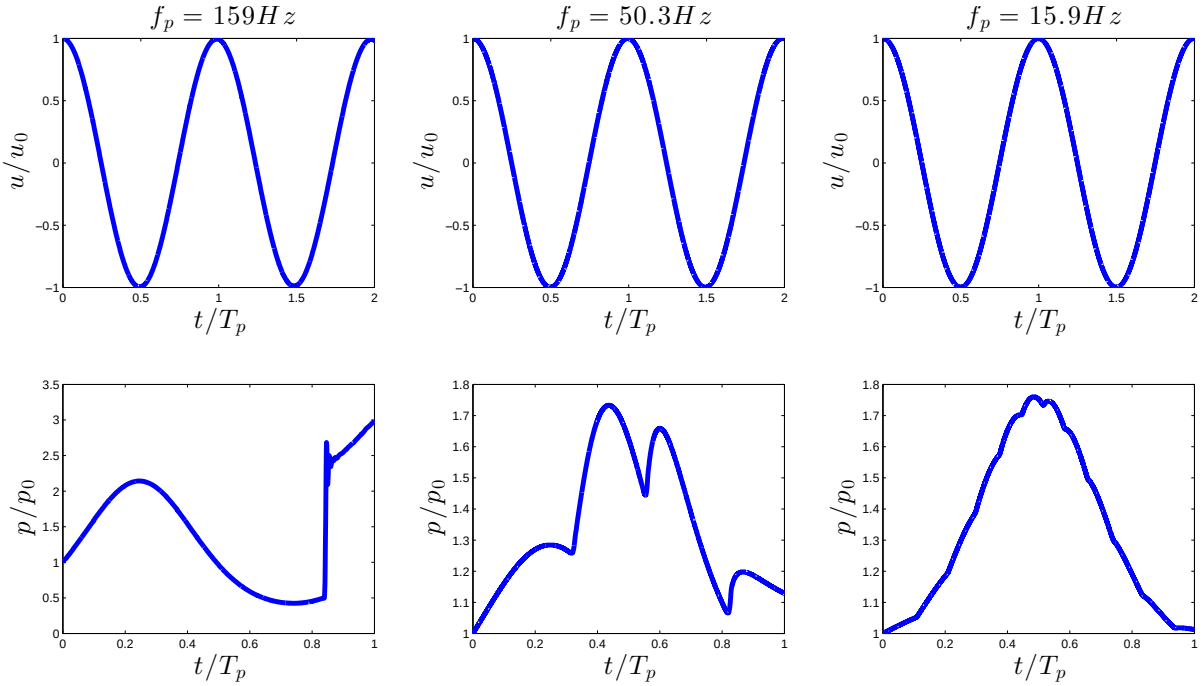

Figure 3.7: Piston test case. Displacement (top) and piston pressure (bottom).

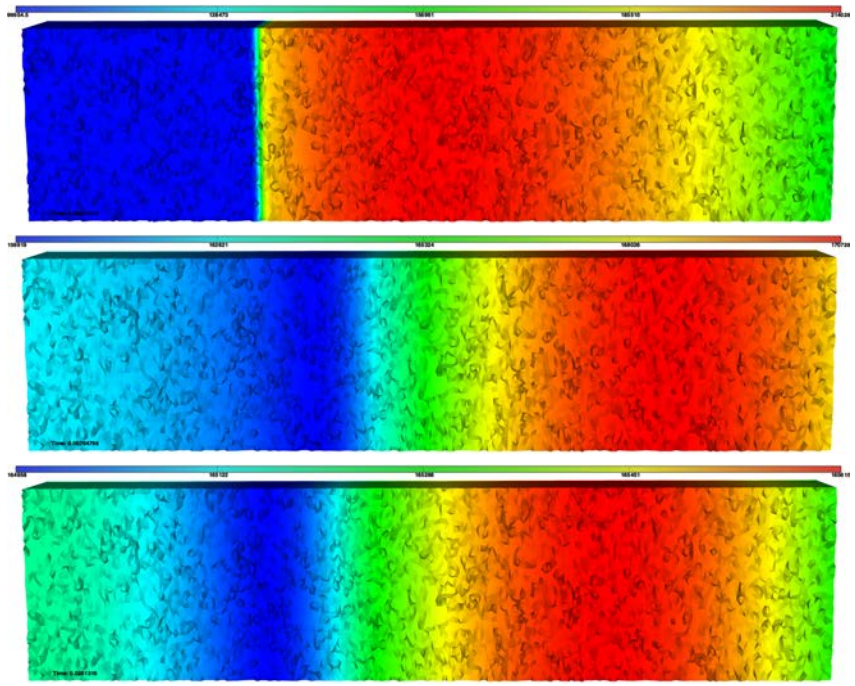

Figure 3.8: Piston test case. Pressure fields at 2/5 of the final time. $m = 10$ and $t = 0.0025$ (top), $m = 100$ and $t = 0.0079$ (center), $m = 1000$ and $t = 0.025$ (bottom). The mobile piston is on the right.

3.2 Numerical examples

3.2.1 AGARD test cases

AGARD test cases are famous test cases for moving geometry problems. The cases are described in detail and experimental data is published in [Landon 1982], which makes it an excellent validation case. Here, we consider two cases, often referred as AGARD CT 1 and AGARD CT 5. Both involve a NACA 0012 airfoil pitching with an angle of attack that is analytically prescribed:

$$\alpha(t) = \alpha_m + \alpha_0 \sin(\kappa t)\,. \tag{3.4}$$

For CT 1, $\alpha_m = 2.89$, $\alpha_0 = 2.41$ and $\kappa = 0.0808$, and for CT 5, $\alpha_m = 0.016$, $\alpha_m = 2.51$ and $\kappa = 0.0814$. The inflow Mach number is 0.6 for CT 1 and 0.755 for CT 5, which correspond respectively to a subsonic and transsonic simulation.

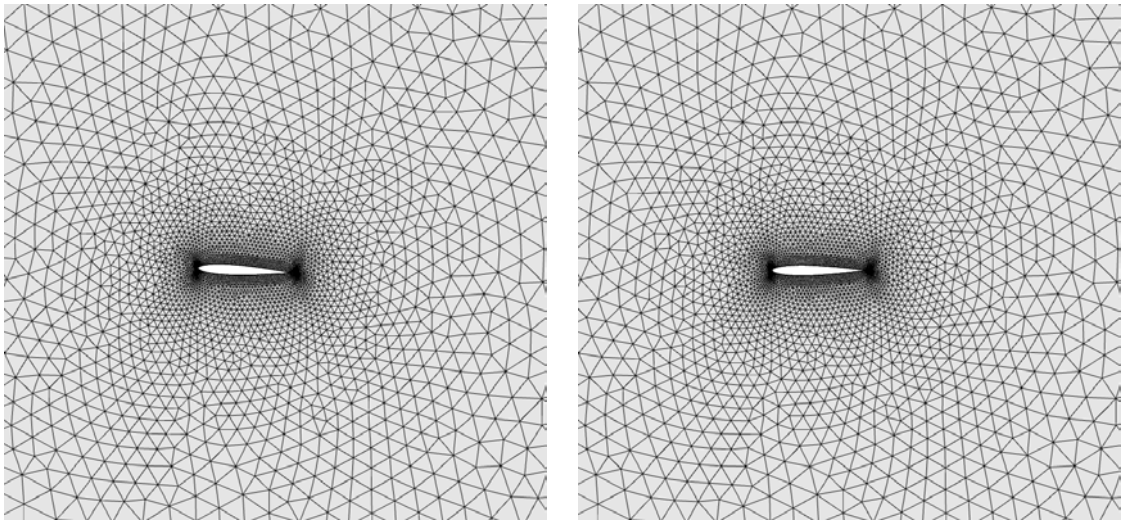

Figure 3.9: Initial meshes for the AGARD CT 1 (left) and CT 5 (right).

The meshes are isotropic and have $4,200$ vertices, see Fig. 3.9. The simulations are run up to times $t_{CT1} = 77.8$ and $t_{CT5} = 77.2$, which correspond to two periods. That way, the flow establishes itself, before reaching its periodic state. Snapshots of the solution at different times of the second period are visible in Fig. 3.10 and 3.11. The two periods can be distinguished in Fig. 3.12 and 3.13, where three aeronautic coefficients are plotted over the angle of attack and over time: the lift, the drag and the pitching moment. On these plots, the available experimental data is also reported, for comparison. We observe that the lift coefficient fits the experimental data with some discrepancies due to the coarseness of the mesh, whereas the pitching moment differs from the experimental data. These results are coherent with other results in the literature, for example [McCullen 2003]: the discrepancies observed are due to the fact that inviscid flows are considered, and the addition of viscosity would improve significantly the accordance of numerical results to experimental data.

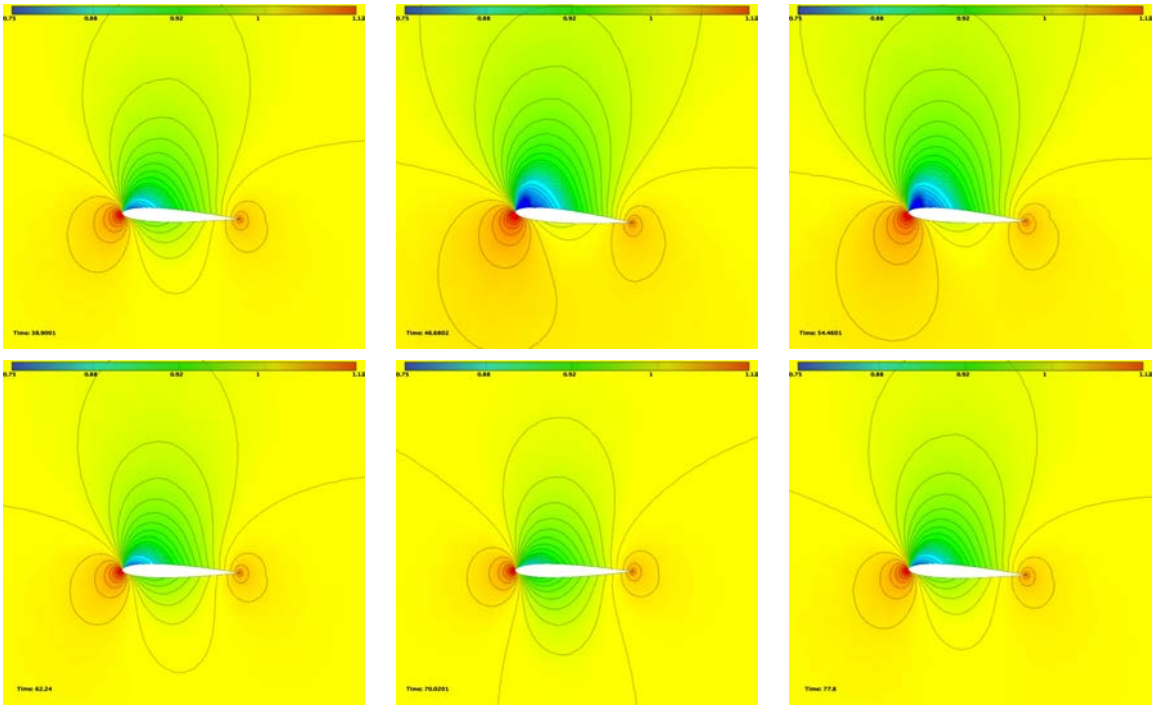

Figure 3.10: AGARD CT 1: Snapshots of the solution at different times of the second period.

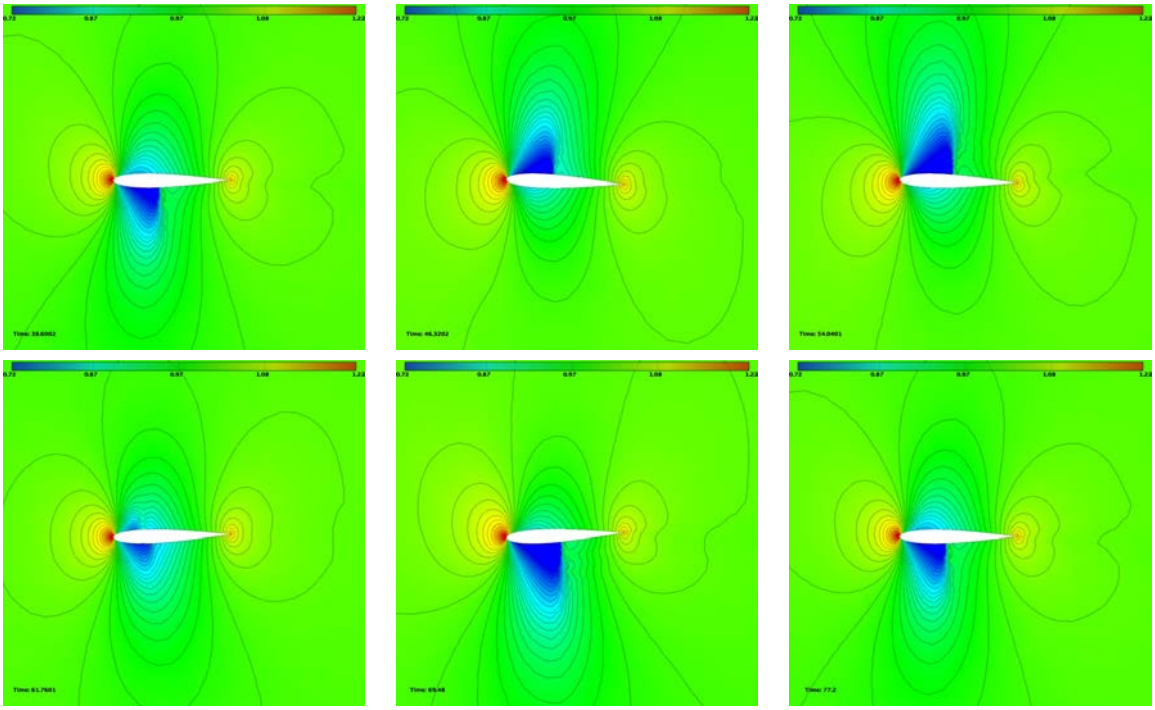

Figure 3.11: AGARD CT 5: Snapshots of the solution at different times of the second period.

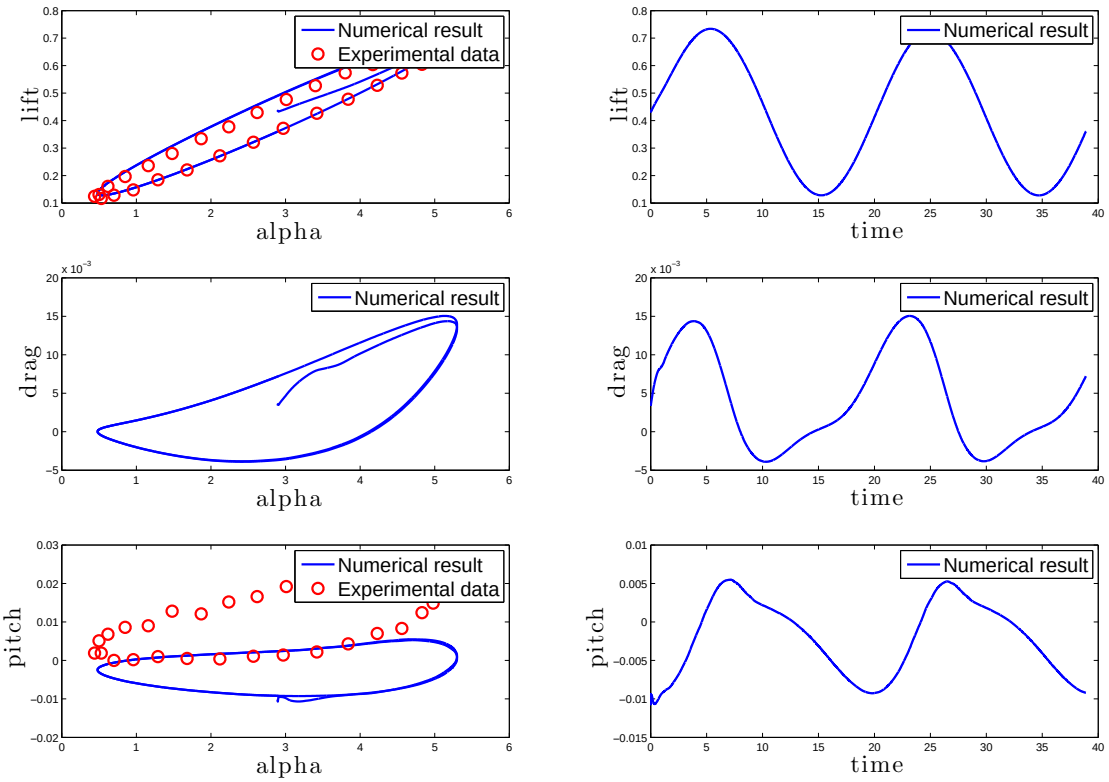

Figure 3.12: AGARD CT case 1.

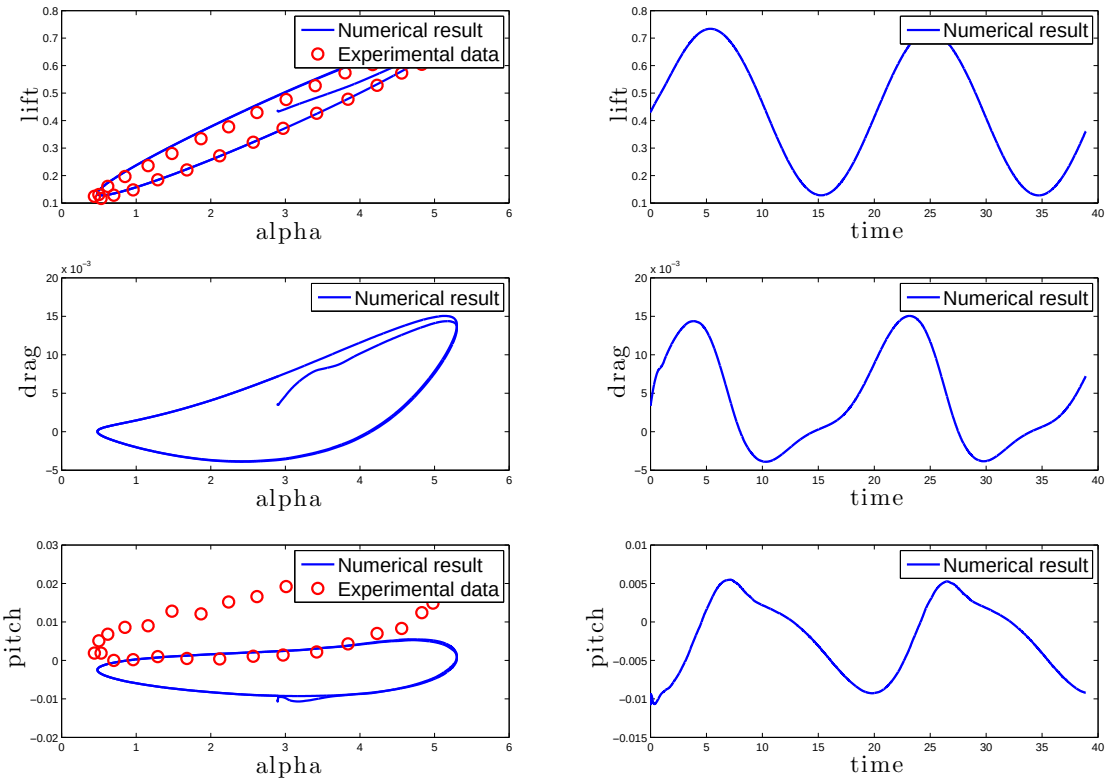

Figure 3.13: AGARD CT case 5.

3.2.2 F117 nosing up

The second example is a subsonic F117 aircraft nosing up, that creates a vortical wake. An inflow of air at Mach 0.4 arrives in front of the aircraft, initially in horizontal position, that noses up, stays up for a while, then noses down. In this example, the aircraft rotates around its center of gravity. Let $T = 1s$ be the characteristic time of the movement and $\theta^{max} = 20^o$ the maximal angle reached, the movement is defined by its angle of rotation, of which the evolution is divided in 7 phases:

$$\theta(t) = \begin{cases} 0 & \text{if} \quad 0 \leq t \leq T/2 \qquad (i) \\ \theta^{max}\left(\frac{2(t-\frac{T}{2})}{T^2}\right) & \text{if} \quad \frac{T}{2} < t \leq T \qquad (ii) \\ \theta^{max}\left(\frac{1}{2} + 2\left(\frac{2\,(t-\frac{T}{2})}{T} - 1\right) - \frac{1}{2}\left(\frac{4\,(t-\frac{T}{2})^2}{T^2} - 1\right)\right) & \text{if} \quad T < t \leq \frac{3T}{2} \qquad (iii) \\ \theta^{max} & \text{if} \quad \frac{3T}{2} < t \leq \frac{7T}{2} \qquad (iv) \\ \theta^{max}\left(1 - \frac{2\,(t-\frac{7T}{2})}{T}\right) & \text{if} \quad \frac{7T}{2} < t \leq 4T \qquad (v) \\ \theta^{max}\left(\frac{1}{2} - 2\left(\frac{2\,(t-\frac{7T}{2})}{T} - 1\right) + \frac{1}{2}\left(\frac{4\,(t-\frac{7T}{2})^2}{T^2} - 1\right)\right) & \text{if} \quad 4T < t \leq \frac{9T}{2} \qquad (vi) \\ 0 & \text{if} \quad \frac{9T}{2} t \leq 5T \qquad (vii)\,. \end{cases}$$

Phases (i) is an initialization phase, during which the flow around the aircraft is established. Phase (ii) and (iii) are respectively phases of accelerated and decelerated ascension. Vortices start to grow behind the aircraft, and they expand during phase (iv), where the aircraft stays in upward position. Phases (v) and (vi) are phases of accelerated and decelerated descent, the vortices start to move away and they slowly disappear in phase (vii). Free-stream conditions are imposed on the faces of the surrounding box, and slipping conditions on the aircraft.

The initial mesh has $501,859$ vertices and $3,012,099$ tetrahedra. The mesh used and snapshots of the solutions are shown in Figures 3.14 and 3.15. A difficulty of this test case is the uneven movement of the aircraft, with strong acceleration and deceleration during the phases of ascension and descent, and the inversion of the acceleration in the middle of these phases. It is necessary to solve enough mesh deformation problems to avoid that the body boundaries cross small elements close to it. Statistics of the moving mesh characteristics are show in Table 3.4. We can see that the two proposed mesh deformation methods produce rather similar results in terms of mesh quality, but the case requires only half as many mesh deformation steps if the IDW method is used. This is probably due to the fact that the displacement of the geometry is relatively small, and most of the vertices of the mesh are located close to the moving body: thus, a method that moves these vertices rigidly predicts better trajectories for them.

	# mesh deform.	# optim.	Q_{end}^{mean}	$1 < Q_{end} < 2$	$Q_{overall}^{worst}$	# swaps
Elasticity	24	66	1.4	99.8%	19	698, 755
IDW	12	37	1.4	99.8%	19	713, 247

Table 3.4: Nosing up F117 test case. Final mesh statistics and total number of swaps with the connectivity-change MMA.

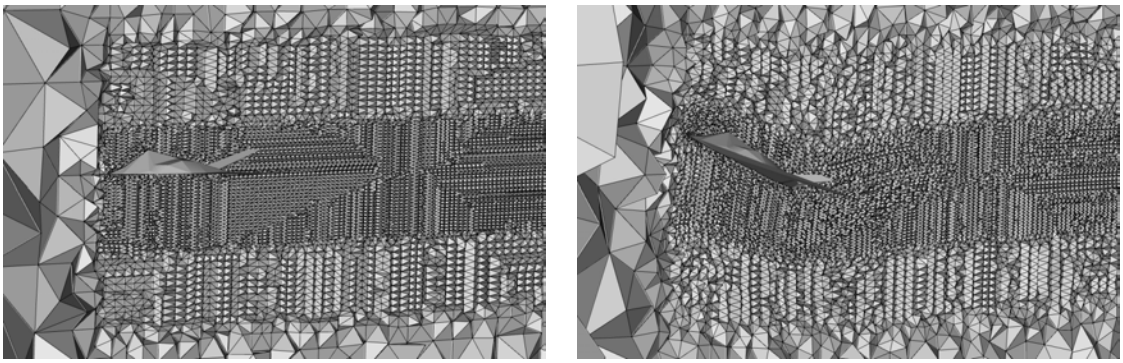

Figure 3.14: Test case of the nosing up F117. Semi-structured meshes in initial horizontal position, and upward position.

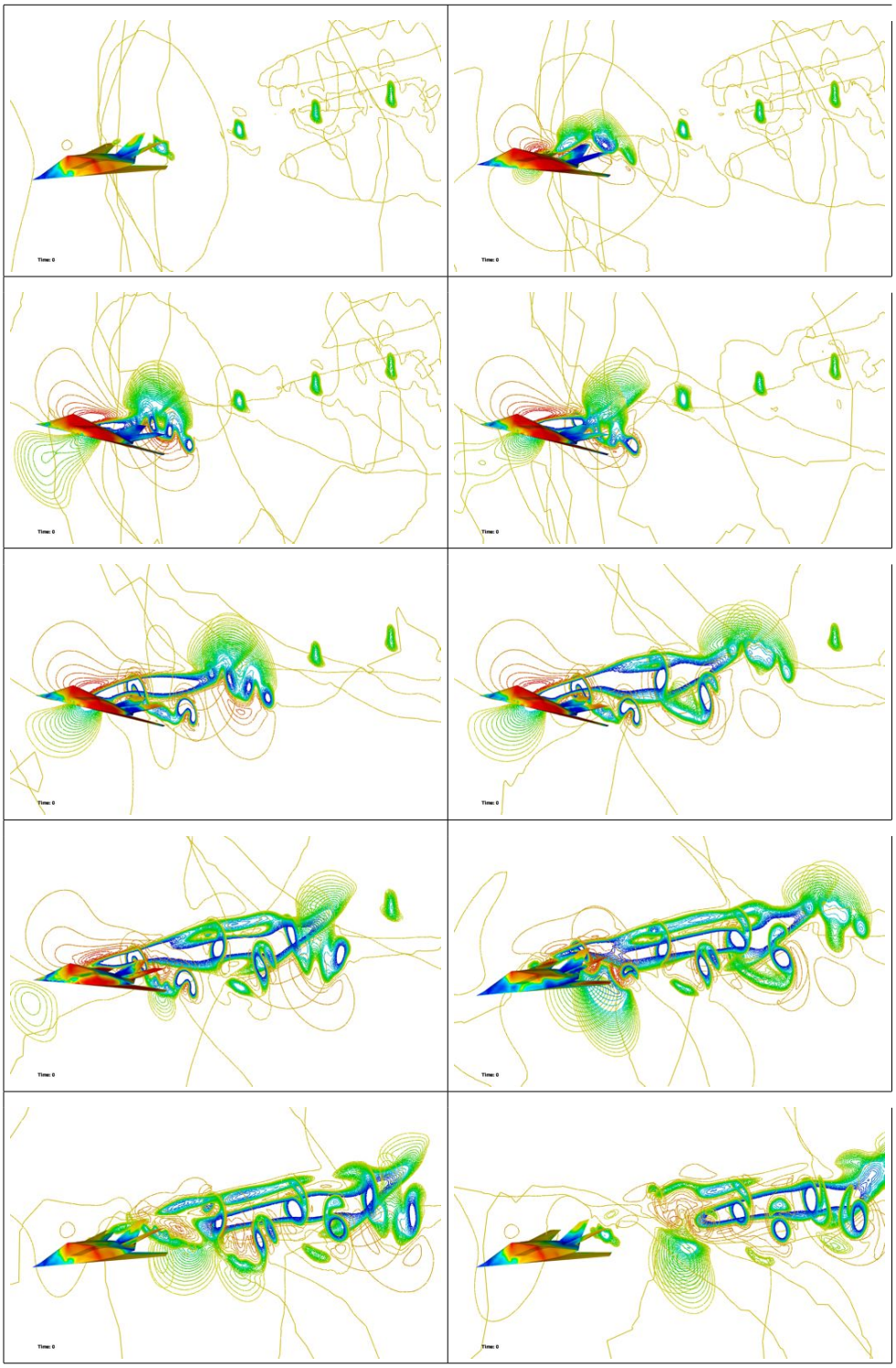

Figure 3.15: Test case of the nosing up F117. Isolines on several cutting planes of the mach field at different time steps ($t = 0.49$, 0.85, 1.22, 1.47, 2.08, 2.82, 3.31, 3.80, 4.28 and 4.90).

3.2.3 Two F117 aircraft crossing flight paths

The third example models two F117 aircraft with crossing flight paths. This problem illustrates well the efficiency of the connectivity-change moving mesh algorithm in handling large displacements of complex geometries without global remeshing. When both aircraft cross each other, the mesh deformation encounters a large shearing due to the opposite flight directions. The connectivity-change mesh deformation algorithm easily handles this complex displacement thanks to the mesh local reconnection. Therefore, the mesh quality remains very good during the whole displacement, without any remeshing step.

The aircraft are imposed the following displacement:

$$\begin{cases} \mathbf{v}_{left} = (1,0,0) \\ \dot{\theta}_{left} = (-1,0,0) \end{cases} \quad \text{and} \quad \begin{cases} \mathbf{v}_{right} = (-1,0,0) \\ \dot{\theta}_{right} = (1,0,0) \end{cases}$$

and the simulation is run until dimensionless time $t = 150$. As concerns the fluid simulation, the aircraft are moved with an imposed motion of translation and rotation at a speed of Mach 0.4, in an initially inert uniform fluid: at $t = 0$ the speed of the air is null everywhere. Transmitting boundary conditions are used on the sides of the surrounding box, while slipping conditions are imposed on the two F117 bodies. After a short phase of initialization, the flow is established when the two F117s pass each other, and the density fields around the aircraft and in their wake interact. Acoustic waves are created in front of the F117s due to the instantaneous setting in motion of the aircraft. This is not realistic, but our aim was to validate our moving mesh approach rather than run a physically realistic simulation. In Fig. 3.16, a zoom on the geometry of the two aircraft is shown. In Fig. 3.18, we show both the moving mesh aspect of the simulation and the flow solution at different time steps.

The mesh is composed of $585,000$ vertices and (initially) 3.5 million tetrahedra. The two methods available were tested for the movement of the mesh. Statistics for both methods are presented in Table 3.5. In both cases, the whole simulation requires 22 elasticity solutions and $1,600$ optimization steps for a total of $2,500,000$ swaps. The final mesh average quality of 1.4 is excellent and we notice that more than 99.7% of the elements have a quality smaller than 2 and only 52 elements have a quality higher than 5. The main difference between both methods lies in the number of swaps, that is almost 5 times bigger for the IDW method. This is due to the fact that with the elasticity-based approach, vertices tend to bypass the moving bodies, while their trajectories are straighter with the IDW, thus creating a lot of shearing. Visualization of a mesh deformation solution just before the aircraft pass each other is telling: the IDW method only moves vertices in the vicinity of the moving bodies, whereas the elasticity method impacts vertices much farther from the bodies, enabling the algorithm to push elements away from the aircraft and fill the gaps behind them.

This simulation was run in a reasonable time: 18 hours were necessary to do $39,000$ time steps on a machine with 20 hyperthreaded i7 cores at 2.5 GHz. The total time required for the solution of the elasticity problems is only 25 minutes for the elasticity method and 5 minutes for the IDW method, which represents only 2.3% (resp. 0.4%) of the total time. The good quality of the mesh ensures an acceptable solution accuracy throughout the simulation. The optimization steps (swapping and smoothing) only take 25 minutes. As for the impact of the swaps, it is difficult to evaluate, since the simulation cannot be run without them. One can

notice that on average, only 66 swaps per solver time step were performed, which is less than 0.0002% of the number of elements of the mesh, and so is likely to have little influence on the solution.

	# mesh deform.	# optim.	Q_{end}^{mean}	$1 < Q_{end} < 2$	$Q_{overall}^{worst}$	# swaps
Elasticity	22	1558	1.4	99.8%	27	$2,576,954$
IDW	22	1693	1.4	99.7%	27	$10,355,416$

Table 3.5: Two F117s test case. Final mesh statistics and total number of swaps with the connectivity-change MMA.

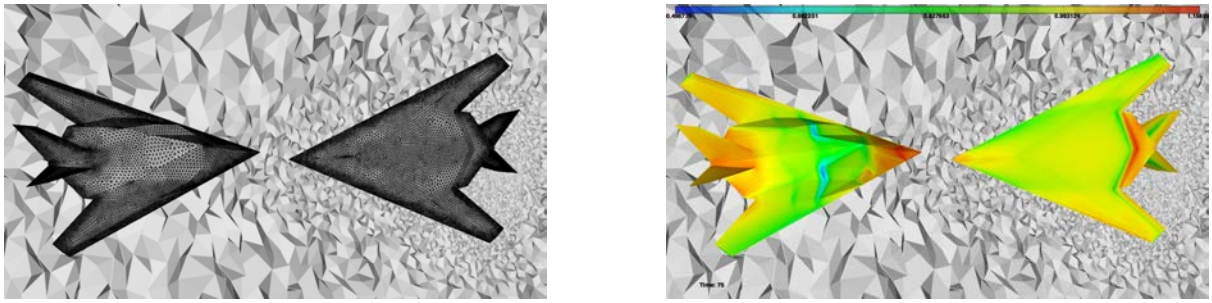

Figure 3.16: Test case of the two F117s. Zoom on the aircraft mesh (left) and solution (right) just before they pass each other.

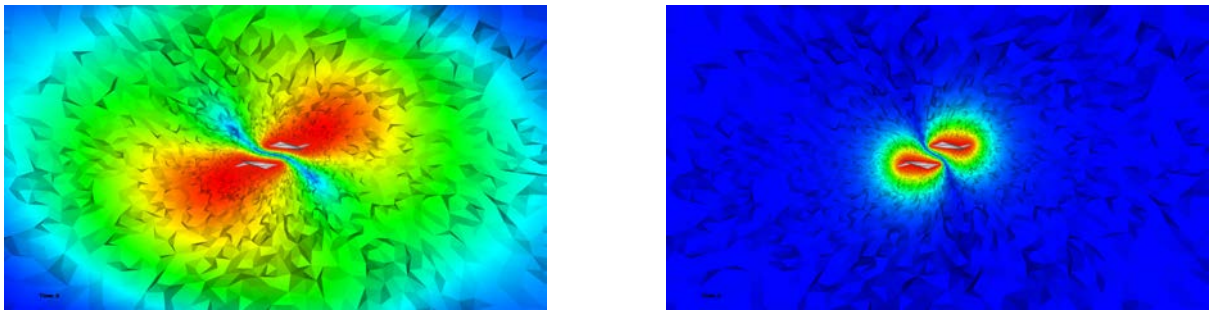

Figure 3.17: Test case of the two F117s. Mesh deformation solution at time $t = 81$ with the linear elasticity method (left) and the IDW method (right).

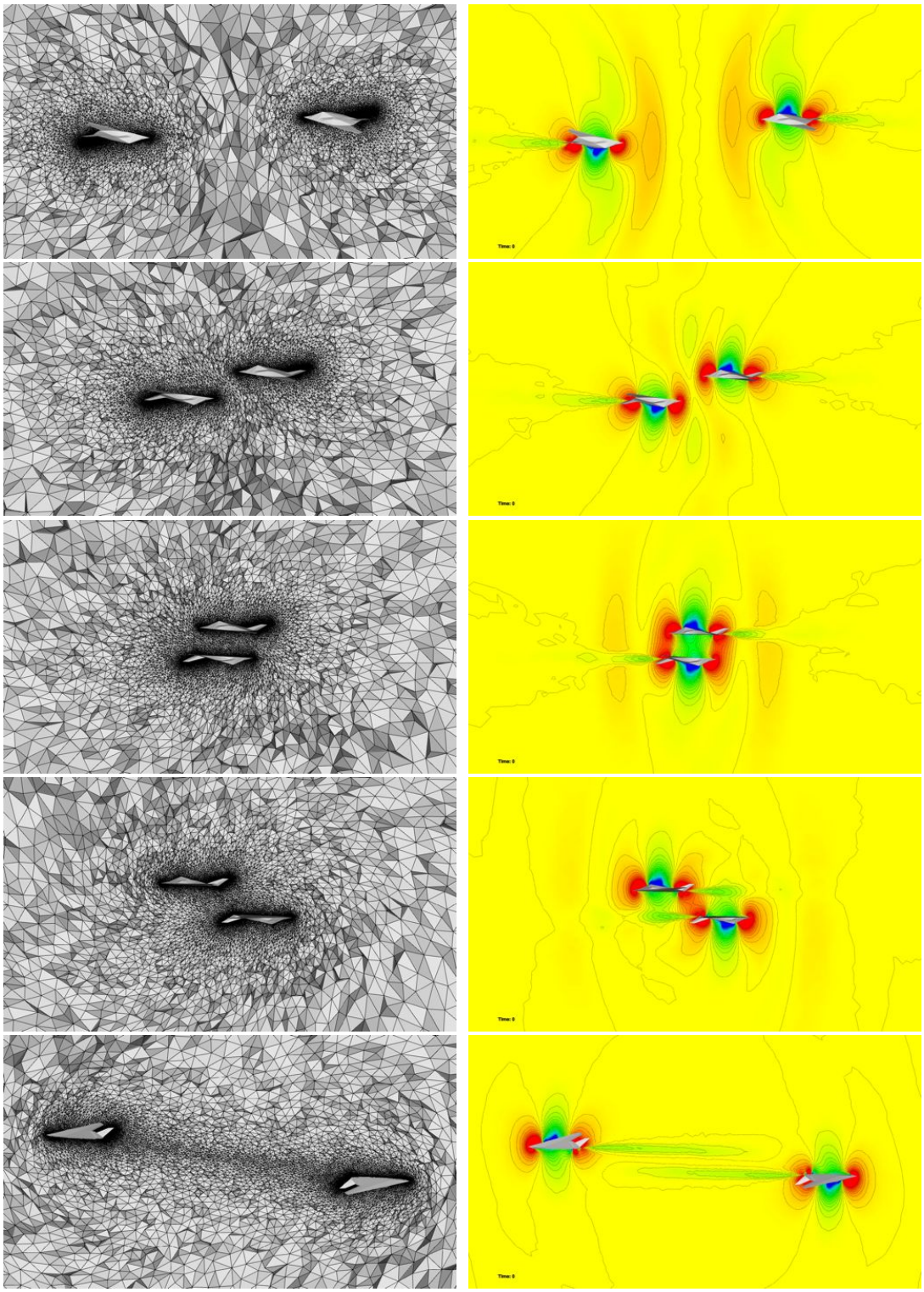

Figure 3.18: Test case of the two F117s. Snapshots of the moving geometries and the mesh (left) and density (right).

3.2.4 Ejected cabin door

The last example is the FSI simulation of the ejection of the door of an over-pressurized aircraft cabin. This is a usual industrial benchmark for aircraft designers, and the aim is to evaluate when the door hinge will yield under cabin pressure. Generally, such experiments are done in a hangar and numerical simulations must be able to predict if the door will impact the observation area.

From a purely moving mesh point of view, the difficulty is that the geometry is anisotropic and rolls over the elements while progressing inside a uniform mesh composed of $965,000$ vertices. Another difficulty lies at the beginning of the movement when the door is ejected from its frame. The gap between the door and the frame is very small, and the mesh is sheared in the interstice, see Figure 3.19. However, no remeshing is needed. The connectivity-change moving mesh strategy is able to get rid of these skewed elements quite rapidly to finally achieve an excellent quality throughout the door displacement.

From the FSI point of view, we present a simplified version of the case – and thus probably not very representative of the actual physics involved. However, we make sure that the solution is physically coherent with the simplified initial conditions. At time $t = 0$, the volume is divided into two parts: outside and inside the cabin, see Figure 3.19. The inside of the cabin and the outside are not insulated, so the pressurized air leaks out of the cabin. The small volume surrounding the door is considered initially as being outside the cabin. The outside of the cabin is initialized to classic atmospheric values at $10,000\ m$ ($\rho = 0.44\ kg.m^{-3}, \mathbf{u} = (0,0,0)\ m.s^{-1}, p = 20\ kPa$). At this altitude, the air of the cabin should be pressurized to simulate an altitude of $2,500m$ ($\rho = 0.96\ kg.m^{-3}, \mathbf{u} = (0,0,0)\ m.s^{-1}, p = 75\ kPa$). To simulate a blast, those values are multiplied by 10 inside the cabin ($\rho = 9.6\ kg.m^{-3}, \mathbf{u} = (0,0,0)\ m.s^{-1}, p = 750\ kPa$). The mass of the door is set to $100\ kg$.

Snapshots of the solution at different time steps are shown in Figure 3.20. We can see features similar to mach diamonds at the rear of the door, due to the high speed of the door. As concerns the door movement, the door is blown away from the cabin as expected. Its trajectory is rather straight, because it is guided between the walls of the cabin at the beginning, like a cannon ball. However, it acquires a slow rotation movement. For the time frame considered ($T_{end} = 0.2s$), gravity has a negligible impact on the trajectory. The mesh is composed of $965,000$ vertices and 5.6 millions of tetrahedra. The whole simulation requires 29 elasticity solutions and 540 optimization steps for a total of $875,000$ swaps. The final mesh average quality of 1.33 is excellent and we notice that 99.95% of the elements have a quality smaller than 2 and only 1 element has a quality higher than 5.

The total time of the simulation is 2 hours on a machine with 20 hyperthreaded i7 cores at 2.5 GHz, for $1,100$ solver time steps, including 43 minutes of elasticity solution. Due to the *a priori* unpredictable trajectory of the door, and to the difficult part when the door is moving in its frame, the number of elasticity steps is slightly higher than with analytic imposed motion. However, the number of elasticity solutions is still small compared to the number of time steps (29 elasticity solutions and $1,122$ solver time steps), and elasticity solutions do not account for more than a third of the total time. The optimization step only takes 3 minutes. Once again, only an average of 0.0001% swaps per tetrahedra and solver time step are performed, which is not a lot.

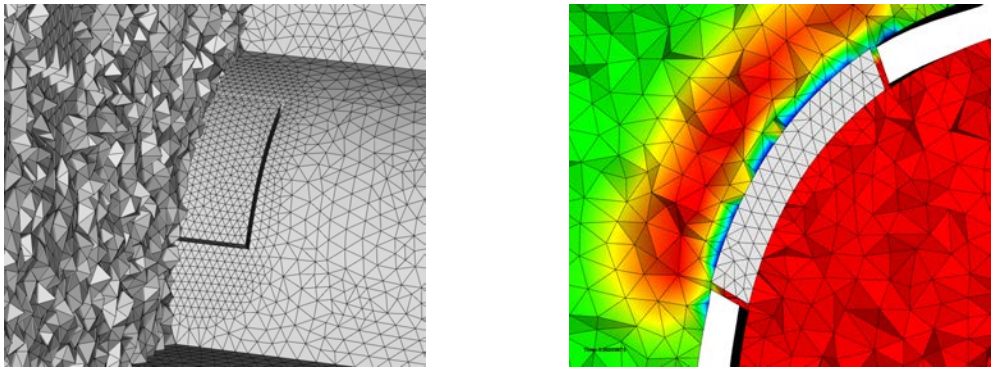

Figure 3.19: Test case of the door ejection. Surface mesh and cut plane just after the beginning of the simulation. Note that the gap between the door and its frame is very small, and meshed with only one layer of elements, which increases the difficulty of the moving mesh problem.

3.3 Parallel performance

The parallelization of the code described in Section 2.5.2 was studied on the problem of the two F117s crossing each other introduced in section 3.2.3. The mesh is made up of 586,610 vertices and 3,416,809 tetrahedra. The simulation is run from time $t = 78$ to time $t = 108$, which corresponds to the interval where the two aircraft actually pass each other. In this interval, 9 elasticity solutions are performed, as well as 317 optimization stages, 941,337 swaps and 9050 solver time steps. The machine considered is a Xeon E5-2670 with two chips with 10 cores at 2.5 GHz each, with hyperthreading capabilities (see Part II Section 5.5 for more details on the machine).

Timings and speed-ups can be found in Table 3.6. First of all, it must be noted that running the case in parallel results in a substantial time-saving. The code scales well up to 4 cores, then scaling factors start to decrease. However, up to 20 cores the speed-up factor increases to finally reach 10. In this case, the hyperthreading extra cores are not useful, the case probably being too small and memory access becoming a bottleneck. To understand these results, a closer look at the parallel performances of each part of the code is necessary.

Detailed timings can be found in Table 3.7. The code was divided into several big parts (underlined). First, the flow solver, Wolf, itself decomposed into the computation of the ALE geometric parameter, the computation of the gradients for the high order scheme, the computation of the fluxes in the volume mesh, the computation of boundary fluxes, the computation of the time step and the advance in time of the solution. The other parts concern the moving of the mesh: the computation of the mesh deformation (the two elasticity solutions), the mesh optimizations (smoothing and swapping), the actual moving of the mesh (update of the positions), and the update of the data structures (edges, tetrahedra, neighbors and parallel structures). As already observed, the predominant part is the solver, and more specifically the computation of the fluxes and the geometric parameters. Whereas the computation of the fluxes scales relatively well, the computation of the geometric parameters scales less well, reaching a speed-up factor of 10 on 20 cores. This can notably be explained by the complexity of this loop (multiple indirections, searches in the data structures) and the dynamic character of some of the data. The parallel data structures and sorting of the data helps improving the efficiency

of the parallelization, but after each mesh optimization stage, the sorting is no more accurate and the parallel efficiency decreases a little. This is partly solved by sorting the data when a parallel efficiency factor provided by the `lplib` reaches a certain threshold. For the other parts of the code, the scaling factors are not great, but they account for a small portion of the total CPU time. The update of the data structure account for a non negligible part of the total time, and it scales up to a certain point. However it is difficult to improve this, notably due to memory access speed. The mesh optimizations are relatively fast, and the parallelization of the computation of the qualities enables some speed-up, even though the optimizations themselves are serial. Note that for some parts, the speed-ups are very bad with hyper-threading, which can probably be explained by memory access bottlenecks and should be analyzed more in depth.

Nbr. cores	1	2	4	8	15	20	20 HT
Timing	40h40	22h35	12h37	7h14	5h05	4h10	4h01
Speed-up	*1*	*1.8*	*3.2*	*5.6*	*8*	*9.8*	*10.1*

Table 3.6: Parallel Wolf timings and speed-up up to 20 cores with hyper-threading.

3.4 Conclusion

The aim of this chapter was to demonstrate that the strategy developed in the two previous chapters, coupling a changing-connectivity moving mesh algorithm and an ALE solver, actually works and allows us to run complex simulation in a reasonable time. This was demonstrated on a wide variety of 3D cases. First academic cases that validate the ALE solver, then more complex CFD cases were run, without ever remeshing. In particular, the influence of swaps on the solution accuracy appears to be negligible. An analysis of parallel timings confirms that the moving mesh part takes little time compared to the solver part, and an efficient parallelization of the solver allows us to save substantial CPU time.

Nbr. cores	1	2	4	8	15	20	20 HT
Wolf (flux)	1065	586	297	152	92	68	59
Speed-up	*1,0*	*1,8*	*3,6*	*7,0*	*11,6*	*15,7*	*18,1*
Wolf (geom. param.)	1085	584	334	192	128	103	92
Speed-up	*1,0*	*1,9*	*3,2*	*5,7*	*8,5*	*10,5*	*11,8*
Wolf (gradients.)	22	12	7,5	4	4	4	9
Speed-up	*1,0*	*1,8*	*2,9*	*5,5*	*5,5*	*5,5*	*2,4*
Wolf (bdry. cdt.)	6	5	2,5	1	1,8	2,5	7,5
Speed-up	*1,0*	*1,2*	*2,4*	*6,0*	*3,3*	*2,4*	*0,8*
Wolf (timestep)	3	1,5	1,5	1,3	1,8	1,7	2
Speed-up	*1,0*	*2,0*	*2,0*	*2,3*	*1,7*	*1,8*	*1,5*
Wolf (adv. time)	18	14	10	8	9	9	12
Speed-up	*1,0*	*1,3*	*1,8*	*2,3*	*2,0*	*2,0*	*1,5*
Wolf (total)	2207	1212	662	368	247	198	190
Speed-up	*1,0*	*1,8*	*3,3*	*6,0*	*8,9*	*11,1*	*11,6*
Mesh deformation	11	12	11	11	11	11	11
Speed-up	*1,0*	*0,9*	*1,0*	*1,0*	*1,0*	*1,0*	*1,0*
Mesh optim.	62	32	20	9	6	5,3	5,7
Speed-up	*1,0*	*1,9*	*3,1*	*6,9*	*10,3*	*11,7*	*10,9*
Move mesh	14	14	14	13,5	15	14	15
Speed-up	*1,0*	*1,0*	*1,0*	*1,0*	*0,9*	*1,0*	*0,9*
Struct. updates	141	83	48	30	23	19	17
Speed-up	*1,0*	*1,7*	*2,9*	*4,7*	*6,1*	*7,4*	*8,3*
Total	**2440**	**1355**	**757**	**434**	**402**	**250**	**241**
Speed-up	***1,0***	***1,8***	***3,2***	***5,6***	***6,1***	***9,8***	***10,1***

Table 3.7: Detailed parallel Wolf timings (in minutes) and speed-up up to 20 cores with hyper-threading.

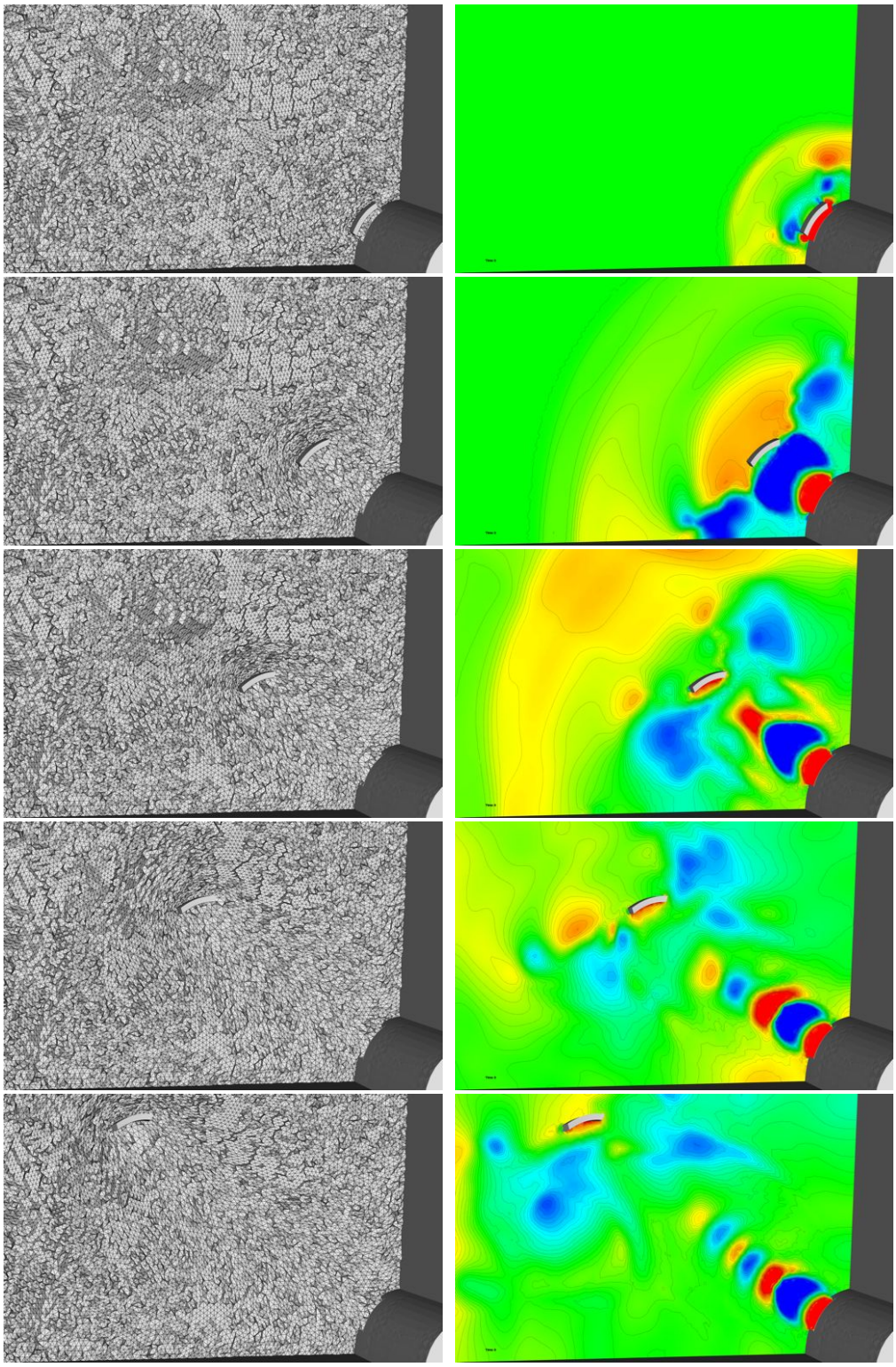

Figure 3.20: Test case of the door ejection. Snapshots of the moving geometry and the mesh (left) and pressure (right).

Conclusion

In this part, we have seen the two aspects of simulations with moving boundaries: the handling of the moving mesh, and its consequences on the solver side. In the first chapter, we have presented a connectivity-change moving mesh algorithm, based on a mesh deformation computed for a large time step, and mesh optimizations performed in between. This algorithm allows us to run large displacement complex moving mesh simulations without remeshing. On this moving meshes, an ALE solver is used, which is described in details in the second chapter. It is based on an HLLC approximate Riemann solver with high order MUSCL-like gradient reconstruction for spatial discretization. Temporal schemes used are high-order multistep Runge-Kutta schemes, built by a local enforcement of the Geometric Conservation Law. This solver was coupled to the moving mesh algorithm to run a large number of simulations in the third chapter. Academic examples are used to validate the solver, in particular in presence of swaps, whereas more complex CFD cases demonstrate the capabilities of the approach.

Following from [Olivier 2011a], several novelties were presented in this part, notably:

- A new mesh deformation method was introduced in the moving mesh algorithm, the Inverse Distance Weighted (IDW) method. It was implemented in the moving mesh code and compared to the elasticity-like method on several cases, where it proved to have comparable performance.
- Moving mesh with boundary layer meshes and large volume variation were addressed, and the results suggest an efficient strategy for these cases.
- The ALE solver was implemented and parallelized in 3D. The formulas for the computation of the geometric parameters were clarified.
- The solver was validated on academic test cases. In particular, the linear interpolation method chosen to handle swaps was shown to have a negligible impact on the solution accuracy.
- ALE simulations were run in 3D, both with imposed motion and Fluid-Structure Interaction.

Perspectives arising from this work concern both the moving mesh aspect and the solver side.

- A strategy for moving boundary layer meshes around deformable geometries naturally arises from what is presented. It needs to be developed and applied on various cases.
- More generally, handling boundary layer meshes would make it possible to run Navier-Stokes simulations.
- Efficient implicit solvers must be considered to speed-up low mach simulations, such as subsonic aeronautic cases and wind turbines.

- More complex FSI simulations are now within range.

Now, there remains to improve the accuracy of these moving mesh simulations without increasing unreasonably their cost, which is performed in the next part using metric-based mesh adaptation.

Part II

Extension of anisotropic metric-based mesh adaptation to moving mesh simulations

Introduction

Simulating complex moving geometries evolving in unsteady flows in three dimensions, such as those presented in the previous part, and which is more and more required by industry, still remains a challenge, because it is very time consuming.

To reduce the CPU time of these simulations while preserving their accuracy, anisotropic metric-based mesh adaptation has already proved its efficiency for steady problems, and appears as a salutary perspective. However, its extension to the unsteady case is not straightforward. These simulations combine the difficulties arising from unsteadiness and geometrical complexity: global time step driven by the mesh smallest altitude, evolution of the phenomena in the whole domain, interpolation spoiling, but also three-dimensional meshing and remeshing issues with an imposed discretized surface. The introduction of moving geometries in this process even raises new difficulties, due to the handling of the mesh movement and the deterioration of its quality, the specific numerical schemes imposed by moving mesh schemes and their restrictions, as well as fluid/structure coupling and contact handling.

Three different approaches dealing with time-dependent mesh adaptation in the literature can be distinguished in the literature: re-meshing methods, fixed-point algorithms and adaptative moving meshes. First, an isotropic mesh is adapted frequently in order to maintain the solution within refined regions and a safety area around critical regions is introduced [Löhner 1990b, Löhner 1992, Rausch 1992, Speares 1997]. Another approach is to use an unsteady mesh adaptation algorithm [de Sampaio 1993, Wu 1990] based on the estimation of the error every n flow solver iterations and global remeshing techniques. If the error is greater than a prescribed threshold, the mesh is re-adapted. A similar strategy is adopted in [Picasso 2003, Picasso 2009], where, at each re-adaptation step, a nearly optimal metric is found by scaling the metrics in every directions depending on directional error factors. More recently, local adaptive remeshing enabling the construction of anisotropic meshes has been considered. In this case [Pain 2001, Remacle 2005, Compère 2008], the mesh is frequently adapted in order to guarantee that the solution always evolves in refined regions. All these approaches involve a large number of adaptations while introducing unquantified errors due to the transfer of the solution from the old mesh to the new one. Moreover, none of them considers the inherent non-linear nature of the mesh adaptation problem: the convergence of the mesh adaptation process is never addressed and therefore obtaining the optimal mesh cannot be expected.

A first answer to these issues has already been proposed in [Alauzet 2007]. It relies on the assumption that the temporal error is always controlled by the spatial error, which is indeed the case when solving a linear advection problem under a CFL condition. A fixed-point algorithm is used to predict the solution and converge the couple mesh/solution. The simulation interval is split into fixed-size sub-intervals on which a unique adapted mesh is used. More recently, mesh adaptation and adaptive time-stepping methods were combined for incompressible unsteady simulations [Coupez 2013]. In that approach, moving boundaries are handled via level-set methods.

In adaptative moving meshes, also called r-adaptation, the mesh is moved at each time step into a mesh adapted to the solution via a mesh equation. A large range of methods share this

common principle, the differences between each lying in the adaptation criterion, the way it influences the mesh movement and the way the mesh movement is taken into account reciprocally in the physical equations. In general, they strongly couple the mesh movement and the physical equation being solved, resulting in methods with no spoiling interpolation errors. Some methods are velocity-based, meaning the mesh position is found through the vertices velocity. They include Lagrangian methods, where the velocity of the mesh is the actual Lagrangian velocity of the fluid [Loubère 2010], Moving Finite Elements, where the mesh celerity is the solution of a minimization problem [Miller 1981, Baines 1994], and methods based on the conservation of the Geometric Conservation Law [Cao 2002]. These methods tend to quickly produce tangled meshes. In location-based methods, the mesh equation directly gives the position of the vertices. The adapted meshes are seen as images of a reference mesh by a coordinate transformation that is the solution of a minimization problem. This problem usually involves a monitor function to specify the sizes of the meshes, which is either heuristic or derived from error estimates. The problems can be based on Laplace [Winslow 1963] or Poisson [Thompson 1983] equations, or harmonic maps [Dvinsky 1991]. The application of the equidistribution principle leads to so called Moving-Mesh PDEs (MMPDEs) [Huang 2010b]. More recently, Monge-Ampère equation was also used to determine the mesh movement [Chacón 2011, Browne 2014]. These approaches seem very time-consuming for now, since the PDEs are solved at every solver time step. Moreover, the solution of the Monge-Ampère equation is known to be very difficult in three dimensions.

In this thesis, we have chosen to work with a fixed-point algorithm following on from [Alauzet 2007]. This is motivated by a reduced CPU consumption with regards to most other methods, by a theoretical background available to control errors both in space and time and by the generic nature of the process: it can work with any physical equation, any solver and any remeshing tool. However, this algorithm is valid only when the space-time interpolation error is controlled in L^∞ norm. Since multiscale mesh adaptation has now proved its efficiency for steady CFD computations [Loseille 2007, Loseille 2011a, Loseille 2011b, Alauzet 2010a], it seems relevant to extend the fixed-point algorithm proposed in [Alauzet 2007] to this framework.

Regarding adaptative strategies for moving mesh simulations, only a few attempts can be found in the literature, among which [Löhner 1989a, Löhner 1990c, Baum 1991, Saksono 2007, Hassan 2007, Compère 2010]. As impressive as they can be, these results nevertheless still suffer from some of the weaknesses described above, *i.e.* mainly very frequent remeshing and spoiling interpolation stages. Very recently, the problem was addressed in [Isola 2015], where the mesh is adapted with local mesh modifications that are integrated directly into the ALE schemes to avoid interpolation errors. However, these local remeshing occur at every solver time step, which can be very costly, and only results in 2D with a small number of vertices were presented.

A two-dimension extension of the fixed point algorithm to moving-geometry problems was introduced in [Alauzet 2011b, Olivier 2011a], in which an effort was made to quantify the impact of the mesh motion on the adaptation process. The goals of the following part of this thesis are first to present updates of the error analysis and the adaptation algorithm for fixed-mesh time-dependent simulations, extending the previous works to multiscale adaptation, and second to extend these updates to adaptative simulations with moving meshes.

This part is structured as follows. In chapter 4, we recall basics of metric-based mesh adaptation, and present all the concepts and tools that will be used in the next chapters. Chapter 5 focuses on time-dependent adaptation for fixed meshes, and important updates of the unsteady adaptation algorithm are presented, both to the theory behind the algorithm and to its practical implementation. Finally, in Chapter 6, we consider mesh adaptation for moving mesh problems: a corrected metric is derived, leading to a modified adaptation algorithm, which is used to run three-dimensional adaptive simulations with moving meshes to validate our approach.

Chapter 4

Basics of metric based mesh adaptation

In this chapter, we are going to set up the notions and formalism necessary for this whole part, defining the metric-based mesh adaptation framework in which our work on unsteady mesh adaptation fits. I am first going to draw up an history of mesh adaptation, then I will discuss the concept of metric, and show how it is used for mesh adaptation. The next section will focus on a powerful framework to model adapted meshes mathematically and compute error estimates. Finally, I will show how all this is used for multiscale mesh adaptation.

4.1 State of the art

Most numerical methods used are based on the subdivision of the spatial domain in polyhedra. This discretization of the space is called a mesh, the polyhedra being the elements of the mesh. The generation of this mesh is an unavoidable step in the numerical simulation workflow. Before it, is the definition of the computational domain, either using analytic functions, such as in CAD (Computer Aided Design), often based on generalizations of Bézier's curves, or using discrete data such as clouds of points. Once the geometry is correctly defined - and it is much harder than it might look - the next step is the **meshing**, in which we are more particularly interested. A mesh of the surface of the domain has to be generated first, based on the prescription of the geometry. Then a volume mesh is generated that is constrained by the surface mesh: the volume is filled with elements, with the constraint of preserving the surface elements given as input.

4.1.1 Meshing status

With the increase of the computational power in the 1980s, together with the arrival of advanced numerical methods, scientists embarked on simulating complex physical phenomena on complex geometries, and started to do three-dimensional simulations. Very quickly, the generation of the mesh became a major issue in the analysis workflow. The need for automated methods to generate meshes became vital, as analysts could spend several months on generating a mesh by hand for a complex geometry. As a result, many books were published on that topic and conferences dedicated to meshing appeared: the annual International Meshing Roundtable, the bi-annual ISGG Conference on Numerical Grid Generation,... Since then, so many works have been carried out on this subject that the list could not fit in this thesis. Many different meshing technologies were considered and developed, with more or less success. Currently, the main approaches for generality and ease of use in CFD are: multiblock grids [Shaw 1992],

adaptive Cartesian grids [Aftosmis 2004] and unstructured mono-element meshes [Coupez 2000, George 1991a, Löhner 1988, Marcum 2000, Weatherill 1994, Yerry 1984].

Pure tetrahedral meshes have now become the most wide-spread technology in engineering compared to other methods, thanks to its ability to mesh automatically complex geometries. Indeed, it is the only method that allows to mesh automatically a wide range of geometries. Other methods still require occasional manual intervention or are only able to handle a small spectrum of geometries, and with some of them, it is usual to spend 6 months on the mesh for complex geometries which is far beyond the expected timeline of a study [Löhner 2001, Puigt 2011]. The exception is simple geometries, such as unit cubes, where Cartesian grids are generally preferred, because they are easy to create and because some numerical schemes are easier to write and more efficient on such discrete support.

As a result, powerful and, by now, mature techniques emerged for tetrahedral mesh generation. Four approaches can be identified: spatial decomposition based methods [Yerry 1984], advancing-front methods [Löhner 1988], Delaunay type methods [George 1991a, Weatherill 1994] and minimal volume principle based methods [Coupez 2000]. Combinations of these approaches have also been considered, see for instance [Marcum 2000] for a coupled advancing front - Delaunay type method. They handle currently most of the cases, even if they still require development to handle the increasing complexity of the geometries and physics considered.

That is why mainly unstructured tetrahedral meshes have been used for mesh adaptation. The other reason for this is the flexibility of the simplex, which allows us to generate anisotropic elements - which is impossible with structured grids - and efficiently modify the mesh density.

4.1.2 History of metric-based mesh adaptation

Adapting the mesh to the simulation run has been considered ever since numerical simulations were run. Since the 1960s, a large number of papers have addressed this topic (Google Scholar finds more than 650,000 results to the query "mesh adaptation"!). However, in most of these works, the adaptation is isotropic and done by successively splitting elements with predefined patterns: a square is split into four squares, a triangle into four triangles, ... It is only after breakthroughs in both fields of error estimates and mesh generation that the idea of anisotropic adapted meshes could emerge.

In 1987, Peraire et al. were the first to point out the directional property of the interpolation error [Peraire 1987], from which they had the idea of generating elements with aspect ratios. They considered a local mapping procedure to generate elongated elements, which, coupled with an advancing front technique, allowed them to generate slightly anisotropic meshes, *i.e.,* a 1:5 ratio. After them, Löhner [Löhner 1989b], Selmin and Formaggia [Selmin 1992] considered similar approaches. Löhner [Löhner 1989b] and Peraire et al. [Peraire 1992] extended the method to 3D, but the meshes considered were almost isotropic. In 1994, Zienkiewicz and Wu [Zienkiewicz 1994] stated that: *"Unfortunately the amount of elongation which can be used in a typical mesh generation by such mapping is small..."* , although they recognized some success with this approach.

Simultaneously, Mavriplis [Mavriplis 1990] considered the meshing point of view to match aeronautical simulations requirements: he generated high aspect ratio elements in boundary

layers and wake regions using a Delaunay approach in a locally stretched space.

The idea of metric was close, and a year later, George and al. introduced the use of metrics in a 2D Delaunay mesher to generate anisotropic meshes. This work synthesized the previous ones: on the one hand, they pointed out that the absolute value of the Hessian of a solution is a metric, and on the other hand they used the metric field induced by the Hessian to compute lengths and areas in the mesher. They proposed to generate a mesh that would be uniform and isotropic in the metric space, which would be adapted and anisotropic in the physical space.

Since then, the concept of metric has been widely exploited 2D anisotropic mesh adaptation. In the late 1990s, the following papers can be referenced among many others: [Fortin 1996, Castro-Díaz 1997, Hecht 1997, Dompierre 1997, Buscaglia 1997]. In 1997, Baker gave a state-of-the-art [Baker 1997] and wrote: *"Mesh generation in three dimensions is 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"*. 3D mesh generation has indeed to face new pathologies, and the very existence of the mesh is not guaranteed, which makes 3D anisotropic mesh adaptation even more complicated.

Meshing blocking points have been partly solved through the use of local remeshing approach: instead of directly generating an adapted mesh, an existing mesh is modified to be adapted (and the existence problem is no more a problem). In the early 2000s, the first results with truly 3D anisotropic mesh adaptation for applications were published [Tam 2000, Pain 2001, Bottasso 2004, Belhamadia 2004, Gruau 2005, Li 2005, Alauzet 2006a]. In the meantime, a lot of work has been done in the field of error estimates, and new more accurate anisotropic error estimates have been proposed: a posteriori estimates [Picasso 2003, Formaggia 2004, Bourgault 2009], a priori estimates [Formaggia 2001, Huang 2005, Alauzet 2006b, Loseille 2009a], and goal-oriented estimates for functional outputs [Venditti 2003, Jones 2006, Loseille 2010b, Alauzet 2011a].

Thanks to its **generality** and **modularity**, metric-based mesh adaptation has been used in various research fields: for example, it has been applied successfully in three-dimensions to the sonic boom simulation [Alauzet 2010a], multi-fluid flows [Compère 2007, Guégan 2010], blast problems [Alauzet 2007], Stefan problems [Belhamadia 2004], metal forming processes [Bruchon 2009], cardiac activity [Southern 2010] as well as ocean modelling [Piggott 2009] or ship hulls [Wackers 2014]. It has also been used with various numerical methods, among which the Finite Volume [Alauzet 2010a, Hay 2007], Finite Element [Allain 2009], Stabilized Finite Element [Bruchon 2009] and Discontinuous Galerkin Finite Element [Remacle 2005, Lacasse 2007] methods. In [Wackers 2012], the authors apply metric-based adaptation methods to semi-structured hexahedral meshes. In all these cases, large improvements in terms of accuracy and CPU performances have been established. There are currently many adaptive software using Riemannian metrics. Let us cite `AFLR2D` [Marcum 2014], `BAMG` [Hecht 1998] and `BL2D` [Laug 2003] in two dimensions, `YAMS` [Frey 2001] for discrete surface mesh adaptation and `Feflo.a` [Loseille 2009b], `Forge3d` [Coupez 2000], `FUN3D` [Jones 2006], `EPIC` [Michal 2012], GAMMANIC3D [George 2003], `MadLib` [Compère 2010], `MeshAdap` [Li 2005], `MMG3D` [Dobrzynski 2008], `MOM3D` [Tam 2000], `TANGO` [Bottasso 2004] and `LibAdaptivity` [Pain 2001] in three-dimensions. It is worth mentioning that all these codes have arisen from different mesh generation methods. The method used in [George 2003, Hecht 1998]

is based on a global constrained Delaunay kernel. In [Laug 2003], the Delaunay method and the frontal approaches are coupled. [Frey 2001, Loseille 2009b, Compère 2010, Dobrzynski 2008] are based on local mesh modifications. [Coupez 2000] is based on the minimal volume principle. Nowadays, metric-based mesh adaptation has become a mature field of research which has now proved its relevance for steady industrial problems. For instance in [Alauzet 2010a], the authors report a mean anisotropic ratio of 1:400 and a mean anisotropic quotient of 50 000 for adapted meshes containing more than 50 millions tetrahedra. Computations involving several millions of tetrahedra can now be considered on a daily basis, with moderate investments.

4.1.3 Other mesh adaptation approaches

Deriving from the pioneer works in mesh adaptation, other techniques have been developed and are worth mentioning: h-, p- and r- refinement. h-refinement is the enrichment of the mesh by splitting elements, p-refinement is the enrichment of the functional space associated to the mesh in the context of Finite Element Methods, and r-refinement consists in moving the vertices of the mesh with a constant number of vertices.

h-refinement is usually associated with Cartesian grids [Berger 1984], where the squares or cubes are split in four or eight. h stands for the size of the elements of the mesh that is modified by this method. It can also be applied to unstructured tetrahedral meshes, as done in [Löhner 1992]. It is still used for a wide variety of problems where structured grids are required - either due to intrinsic physics of the problems or to the solver used.

p-refinement was introduced in the 1980s [Babuska 1981]. It is used in the context of FEM methods, and consists in increasing the order of the polynomials (hence the p) element by element. It is often used in combination with h-refinement, resulting in the hp-refinement method [Babuska 1994].

Finally, r-refinement consists in relocating (hence the r) the vertices of the mesh to move them closer to the areas of interest of the solution. References [Miller 1981, Baines 1994, Huang 2010b] can be pointed out for introduction to these methods. Authors sometimes call this process "generation of adaptive meshes", although it is not really generation of a mesh but the modification of a pre-existing mesh, usually a Cartesian mesh. These methods present a specific interest, because they can easily be adapted to the case of simulations with moving boundaries and moving meshes.

4.2 Principle of metric-based adaptation

Adapting a mesh means changing the sizes of its elements to optimize a certain quantity. These sizes need to be represented with a mathematical tool, that needs to be easily derived from error models, and that is convenient for the meshing software. One way to represent size charts are so called monitor functions [Huang 2010a]. In this work, we will consider metrics, and more generally Riemannian metric spaces, that are convenient to define the fundamental concept of unit mesh. The question is then how to compute such metrics that lead to adapted meshes, which we will explain with the basic steady adaptation algorithm.

4.2.1 Euclidian and Riemannian metric spaces

Euclidian space

We consider the vector space $\mathbb{R}^n$, typically $n = 2$ or 3 in our case. A **scalar product** is a Symmetric Positive Definite (SPD) form. This form can be represented by a SPD matrix $\mathcal{M} = (m_{ij})_{1 \leq i,j \leq n}$, which is called a *metric tensor* or simply *metric*. The scalar product is then written:

$$\begin{array}{rccl} (\cdot\,,\cdot)_{\mathcal{M}}: & \mathbb{R}^n \times \mathbb{R}^n & \longrightarrow & \mathbb{R}^+ \\ & (\mathbf{u}, \mathbf{v}) & \longmapsto & (\mathbf{u},\,\mathbf{v})_{\mathcal{M}} \;=\; \mathbf{u}^T\,\mathcal{M}\,\mathbf{v} \;=\; \displaystyle\sum_{j=1}^{n}\sum_{i=1}^{n} m_{ij} u_i v_i\,. \end{array} \tag{4.1}$$

In the simple case were $\mathcal{M} = \mathcal{I}$ where $\mathcal{I}$ is the identity matrix, the scalar product is the canonical Euclidian dot product. A vector space with a scalar product is called an *Euclidian space*.

The scalar product is useful to compute lengths and angles in the Euclidian space. We can define:

- a distance:

$$\begin{array}{rccl} d_{\mathcal{M}}: & \Omega \times \Omega & \longrightarrow & \mathbb{R}^+ \\ & (P,\,Q) & \longmapsto & d_{\mathcal{M}}(P,\,Q) = \sqrt{\mathbf{PQ}^T\,\mathcal{M}\,\mathbf{PQ}}\,, \end{array} \tag{4.2}$$

  which induces a metric space structure on the vector space,

- and a norm:

$$\begin{array}{rccl} ||\cdot||_{\mathcal{M}}: & \mathbb{R}^n & \longrightarrow & \mathbb{R}^+ \\ & \mathbf{u} & \longmapsto & ||\mathbf{u}||_{\mathcal{M}} = \sqrt{\mathbf{u}^T\,\mathcal{M}\,\mathbf{u}}\,. \end{array} \tag{4.3}$$

From these distance and norm, we deduce the **classic geometrical quantities**:

- the length of an edge $\mathbf{e}$ is given by:

$$\ell_{\mathcal{M}}(\mathbf{e}) = \sqrt{\mathbf{e}^T\,\mathcal{M}\,\mathbf{e}}\,, \tag{4.4}$$

- the angle between two vectors $\mathbf{v_1}$ and $\mathbf{v_2}$ is the unique real number $\theta \in [0, \pi]$ such that:

$$\cos\theta = \frac{(\mathbf{v_1},\,\mathbf{v_2})_{\mathcal{M}}}{||\mathbf{v_1}||_{\mathcal{M}}\,||\mathbf{v_2}||_{\mathcal{M}}} \tag{4.5}$$

- the volume of element K is:

$$|K|_{\mathcal{M}} = \sqrt{\det\mathcal{M}}\,|K|_{\mathcal{I}}\,. \tag{4.6}$$

A very useful result on metrics is that $\mathcal{M}$ is **diagonalizable in an orthonormal basis**:

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

$$\text{where} \begin{cases} \Lambda \;= \text{diag}\,(\lambda_1,\,\ldots,\,\lambda_n) \text{ is the diagonal matrix made of the eigenvalues of } \mathcal{M}\,, \\ \mathcal{R} \;= (\,\mathbf{r_1}\,|\,\mathbf{r_2}\,|\,\ldots\,|\mathbf{r_n}\,)^T \text{ is the unitary matrix (\textit{i.e.} } \mathcal{R}^T\mathcal{R} = \mathcal{I}) \\ \quad\;\; \text{made of the eigenvectors of } \mathcal{M}\,. \end{cases} \tag{4.7}$$

We will very often use the **geometric representation** of a metric tensor. In the vicinity $\mathcal{V}(\mathbf{a})$ of point $\mathbf{a}$, the set of points that are at distance ε of $\mathbf{a}$, is given by:

$$\Phi_{\mathcal{M}}(\varepsilon) = \{\mathbf{x} \in \mathcal{V}(\mathbf{a}) \mid d_{\mathcal{M}}(\mathbf{x}, \mathbf{a}) = \varepsilon\}. \tag{4.8}$$

This relation defines an ellipsoid of center $\mathbf{a}$ and whose axes are aligned with the eigen directions r_i of $\mathcal{M}$. The set $\Phi_{\mathcal{M}}(1)$ is called the **unit ball** of $\mathcal{M}$, and the sizes of its axes are $h_i = \lambda_i^{-\frac{1}{2}}$. This unit ball is depicted in Figure 4.1 in two and three dimensions.

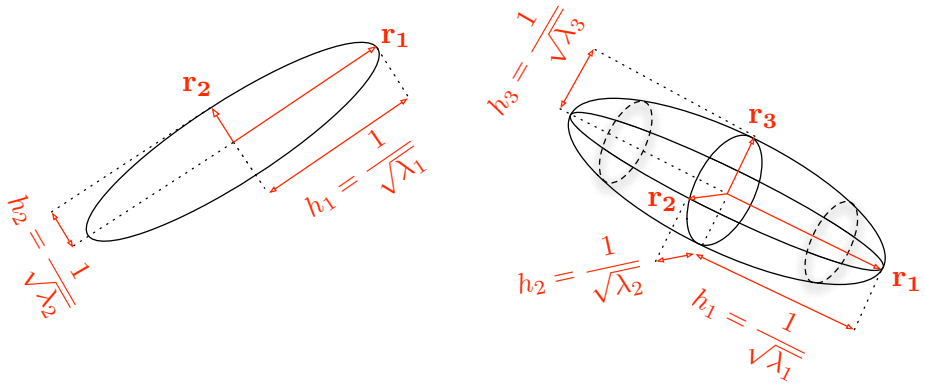

Figure 4.1: Unit balls associated with metric $\mathcal{M} = \mathcal{R}\,\Lambda\,\mathcal{R}^T$ in two and three dimensions.

A metric tensor $\mathcal{M}$ provides another useful information: the application that maps the unit ball associated with $\mathcal{I}$ to the unit ball associated with $\mathcal{M}$. The matrix of this transformation in canonical basis $(\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_n})$ is:

$$\mathcal{M}^{-\frac{1}{2}} = \mathcal{R}\,\Lambda^{-\frac{1}{2}}\,\mathcal{R}^T, \quad \text{where } \Lambda^{-\frac{1}{2}} = \text{diag}\left(h_1 = \lambda_1^{-\frac{1}{2}}, \ldots, h_n = \lambda_n^{-\frac{1}{2}}\right). \tag{4.9}$$

This **natural mapping** corresponds to the change of basis from the canonical basis to the orthonormal diagonalization basis of $\mathcal{M}$ and is depicted in Figure 4.2.

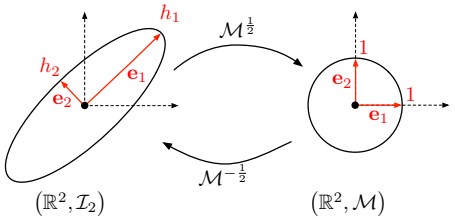

Figure 4.2: Natural mapping $\mathcal{M}^{-\frac{1}{2}}$ associated with metric $\mathcal{M}$ in two dimensions. It sends the unit ball of $\mathcal{I}_2$ onto the unit ball of $\mathcal{M}$.

Riemannian metric space

The scalar product of Euclidian spaces is the same on the whole space, which means that the distance definition is the same for each point of the space (the unit ball is the same everywhere).

However, when used to generate adapted meshes with different element sizes in the domain, it is convenient to have a distance that depends on the position of the space. We now consider a set of SPD tensors $\mathbf{M} = (\mathcal{M}(P))_{P\in\Omega}$, also called metric tensor field, defined on the whole domain $\Omega \subset \mathbb{R}^n$. Locally at point P, $\mathcal{M}(P)$ induces a scalar product on $\mathbb{R}^n \times \mathbb{R}^n$. The vector space, with this new structure, is called a **Riemannian metric space**. In this thesis, we will use the same notation $\mathcal{M}$ to speak of the metric field and of the metric tensor at a given point. Notation $\mathbf{M}$ will only be used if the distinction is necessary for pedagogical purposes.

Remark 3. *Unlike usual Riemannian spaces, there is no notion of manifold in Riemannian metric spaces. However, Riemannian metric spaces can be assimilated to functions representing Cartesian surfaces, and the metric tensor defined for a point of the space is a scalar product on the tangent plane for that point. Figure 4.3 gives an example of a Cartesian surface associated with a Riemannian metric space. This Riemannian metric space is pictured by drawing the unit ball of the metric at some points of the domain. A link with differential geometry is proposed in [Olivier 2011a].*

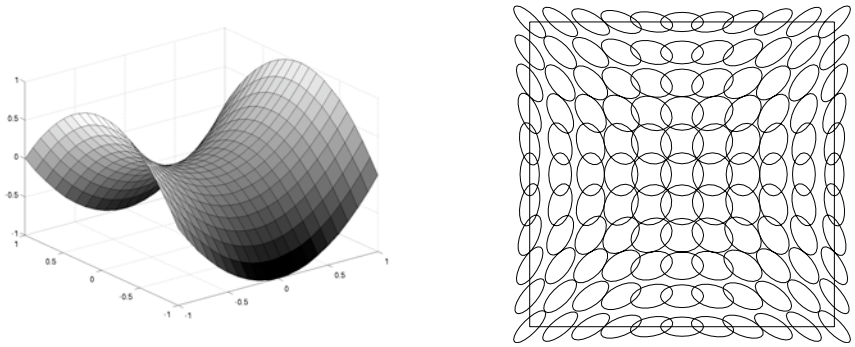

Figure 4.3: Left, example of a Cartesian surface embedded in $\mathbb{R}^3$. Right, geometric visualization of a Riemannian metric space $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in[0,1]\times[0,1]}$ associated with this surface. At some points $\mathbf{x}$ of the domain, the unit ball of $\mathcal{M}(\mathbf{x})$ is drawn.

There is no global notion of scalar product in a Riemannian metric space, thus no global distance or norm. However, we can extend the notions of length, angle and volume from the Euclidian space case. To do so, we have to consider the variation of the metric between two points. The following **geometrical quantities** are defined:

- the length of edge $\mathbf{e} = \mathbf{PQ}$ parametrized by $\gamma : t \in [0,1] \longmapsto P + t\,\mathbf{PQ}$ is computed with the following formula:

$$\ell_{\mathcal{M}}(\mathbf{e}) = \int_0^1 ||\gamma'(t)||_{\mathcal{M}(\gamma(t))}\,\mathrm{d}t = \int_0^1 \sqrt{\mathbf{PQ}^T\,\mathcal{M}(P + t\,\mathbf{PQ})\,\mathbf{PQ}}\,\mathrm{d}t\,, \tag{4.10}$$

- the angle between two vectors $\mathbf{v_1} = \mathbf{PQ}_1$ and $\mathbf{v_2} = \mathbf{PQ}_2$ is the unique real $\theta \in [0,\pi]$ such that:

$$\cos\theta = \frac{(\mathbf{v_1},\,\mathbf{v}_2)_{\mathcal{M}(P)}}{||\mathbf{v}_1||_{\mathcal{M}(P)}\,||\mathbf{v}_2||_{\mathcal{M}(P)}}\,, \tag{4.11}$$

- the volume of element K with respect to $\mathbf{M}$ is more difficult to apprehend. Indeed, due to metric variations, element K, as seen with respect to metric field $\mathbf{M}$ is generally curved: it is not a simplex anymore and normally, its volume should be computed with an integration formula:

$$|K|_{\mathcal{M}} = \int_K \sqrt{\det \mathcal{M}(\mathbf{x})}\, d\mathbf{x}\,. \tag{4.12}$$

  However, this volume can be approximated at first order:

$$|K|_{\mathcal{M}} \approx |K|_{\mathcal{I}} \sqrt{\det \mathcal{M}(G_K)}, \quad \text{where } G_K \text{ is the barycenter of } K. \tag{4.13}$$

4.2.2 Unit mesh

Metric fields, as defined in the previous section, allow us to modify locally geometric quantities such as lengths and volumes as well as to provide a local orientation (prescribing two privileged axes). They are thus a good way, both elegant and mathematically efficient, to prescribe sizes and orientations for the elements of an anisotropic adapted mesh. The main idea of metric-based mesh adaptation, introduced for the first time in [George 1991b], is to use directly a Riemannian metric space within the mesher to compute the necessary geometrical quantities, and to generate a *unit mesh* with respect to this Riemannian metric space.

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

$$\forall i = 1, ..., 6, \;\; \ell_{\mathcal{M}}(\mathbf{e_i}) = 1 \text{ with } \ell_{\mathcal{M}}(\mathbf{e_i}) = \sqrt{\mathbf{e_i}^T \mathcal{M}\, \mathbf{e_i}}. \tag{4.14}$$

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

$$|K|_{\mathcal{M}} = \frac{\sqrt{2}}{12} \;\text{ and }\; |K| = \frac{\sqrt{2}}{12}(\det(\mathcal{M}))^{-\frac{1}{2}}, \tag{4.15}$$

where $|K|$ is its Euclidean volume.

Although this definition is very simple, things are more complicated when it comes to defining a **unit mesh**. *Stricto sensu*, a unit mesh is mesh whose edges are all unit with respect to the prescribed metric field. However, the existence of a mesh made of unit elements is not assured. For example, in the simplest case of $\mathcal{M}(P) = \mathcal{I}(P)$ for each point P, *i.e.* the canonical Euclidian space, it is well known that $\mathbb{R}^3$ cannot be filled with regular tetrahedra (that are unit with respect to the identity metric). So the constraint on the sizes of the edges has to be relaxed, but this can lead to meshes with very bad elements (flat elements), so we have to add a constraint on the volume of the elements, through a quality function.

We have to introduce the notion of **quasi unit element**. A tetrahedron is said to be quasi unit with respect to a metric field $\mathcal{M}$ if its edges are close to unit, *i.e.* $\forall i,\; \ell_{\mathcal{M}}(\mathbf{e_i}) \in [\frac{1}{\sqrt{2}}, \sqrt{2}]$. To avoid elements with a null volume (see [Loseille 2011a]), we have to add a constraint on the volume, which is achieved through a quality function:

$$Q_{\mathcal{M}}(K) = \frac{\sqrt{3}}{216} \frac{\left(\sum_{i=1}^{6} \ell^2_{\mathcal{M}}(\mathbf{e_i})\right)^{\frac{3}{2}}}{|K|_{\mathcal{M}}} \in [1, +\infty]\,. \tag{4.16}$$

For the regular tetrahedron, the quality function is equal to 1, whereas it tends to $+\infty$ for a null volume tetrahedron. So an element close to a perfectly unit element has a quality close to 1. This leads to the following definition. A tetrahedron K defined by its list of edges $(\mathbf{e_i})_{i=1...6}$ is said **quasi-unit** for Riemannian metric space $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ if

$$\forall i \in [1,6], \quad \ell_{\mathcal{M}}(\mathbf{e_i}) \in \left[\frac{1}{\sqrt{2}}, \sqrt{2}\right] \quad \text{and} \quad Q_{\mathcal{M}}(K) \in [1,\alpha] \quad \text{with} \quad \alpha > 1\,, \tag{4.17}$$

The definition of unit mesh consequently becomes: a **unit mesh** with respect to a Riemannian metric space $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ is a mesh made of quasi-unit elements.

In practice, it is this definition that is used in meshing software. For any kind of desired mesh (uniform, adapted isotropic, adapted anisotropic), the mesh generator will generate a mesh that is unit with respect to the prescribed metric space. The resulting mesh is uniform in the metric space while it is adapted in the canonical Euclidian space. This is illustrated in Figure 4.4.

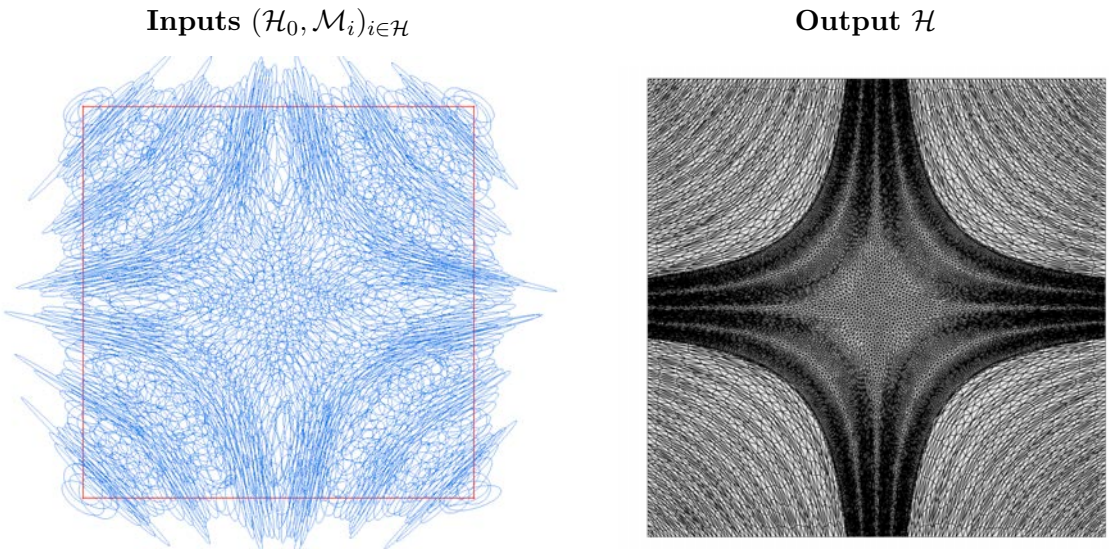

Figure 4.4: Metric-based mesh generation. Left, specified Riemannian metric space. Right, unit mesh in the prescribed Riemannian metric space which becomes adapted anisotropic in the Euclidean space.

4.2.3 Operations on metrics

The main advantage of using metrics is the existence of well-posed operations on metrics, that have a direct geometric interpretation with unit ellipsoids. The most useful ones are metric intersection and metric interpolation.

Metric intersection

When several metrics are specified at a point of the domain, all these metric tensors must be reduced to a single one due to mesh generation concerns. The **metric intersection** consists

in keeping the most restrictive size constraint in all directions imposed by this set of metrics. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two metric tensors given at a point. The intersection of $\mathcal{M}_1$ and $\mathcal{M}_2$ is the metric tensor $\mathcal{M}_{1\cap 2}$ that prescribes the largest possible size under the constraint that the size in each direction is always smaller than the sizes prescribed by $\mathcal{M}_1$ and $\mathcal{M}_2$. From a geometric point of view, the intersection between two metrics is not the intersection between two ellipsoids as their geometric intersection is not an ellipsoid, but it is the largest ellipsoid representing $\mathcal{M}_{1\cap 2}$ included in the geometric intersection of the ellipsoids associated with $\mathcal{M}_1$ and $\mathcal{M}_2$, cf. Figure 4.5, left.

The ellipsoid (metric) verifying this property is obtained by using simultaneous reduction of the quadratic forms associated with the two metrics. During this thesis, I showed that the definition of the metrics intersection used so far [Alauzet 2003a] did not work in some cases, especially when the two metrics are identical, and I proposed a new definition, that is equivalent to the one proposed in [McKenzie 2009].

Simultaneous reduction theory for two quadratic forms, one of which being positive definite, tells that there exist a common basis that is orthonormal for the positive definite form and orthogonal for the other. In other words, there is a basis in which the matrix of the first form is the identity and in which the matrix of the other form is diagonal. We place ourselves in this basis in two steps.

First, let us consider $\mathcal{M}_1^{-1/2}$ (which exists because $\mathcal{M}_1$ is positive definite): $\mathcal{M}_1 = \mathcal{P}\text{diag}(\lambda_1,\lambda_2,\lambda_3)\mathcal{P}^T \Rightarrow \mathcal{M}_1^{-1/2} = \mathcal{P}\text{diag}(1/\sqrt{\lambda_1},1/\sqrt{\lambda_2},1/\sqrt{\lambda_3})\mathcal{P}^T)$. This matrix represents a mapping that sends metric $\mathcal{M}_1$ onto the identity.

Let :

$$\begin{aligned}\overline{\mathcal{M}_1} &= \mathcal{M}_1^{-1/2^T}\mathcal{M}_1\mathcal{M}_1^{-1/2} = \mathcal{I}\,,\\ \overline{\mathcal{M}_2} &= \mathcal{M}_1^{-1/2^T}\mathcal{M}_2\mathcal{M}_1^{-1/2}\,.\end{aligned}$$

We diagonalize $\overline{\mathcal{M}_2}$, noting $\mathcal{P}$ the matrix whose columns are the eigenvectors:

$$\begin{aligned}\overline{\mathcal{M}_2} &= \mathcal{P}\begin{pmatrix}\lambda_1 & 0 & 0\\ 0 & \lambda_2 & 0\\ 0 & 0 & \lambda_3\end{pmatrix}\mathcal{P}^T\,,\\ \overline{\mathcal{M}_1} &= \mathcal{P}\,\mathcal{I}\mathcal{P}^T\,.\end{aligned}$$

The two metrics are then intersected in this latter basis:

$$\overline{\mathcal{M}_{1\cap 2}} = \mathcal{P}\begin{pmatrix}\max(\lambda_1,1) & 0 & 0\\ 0 & \max(\lambda_2,1) & 0\\ 0 & 0 & \max(\lambda_3,1)\end{pmatrix}\mathcal{P}^T\,.$$

So finally:

$$\mathcal{M}_{1\cap 2} = \mathcal{M}_1^{1/2^T}\,\overline{\mathcal{M}_{1\cap 2}}\,\mathcal{M}_1^{1/2}\,. \tag{4.18}$$

Remark 4. *The intersection operation is not commutative. Consequently, when more than two metrics are intersected, the result depends on the order of intersection. In this case, the*

resulting intersected metric is not anymore optimal. If, we seek for the largest ellipsoid included in the geometric intersection region of several (> 2) metrics, the John ellipsoid can be used after solving an optimization problem [Loseille 2008].

Metric interpolation

In practice, the metric field is only known discretely at mesh vertices. The definition of an interpolation procedure on metrics is therefore mandatory to be able to compute the metric at any point of the domain. For instance, the computation of the volume of an element with Relation (4.12) is generally performed using quadrature points to approximate the integral. It thus requires the computation of some interpolated metrics inside the considered element. Figure 4.5 illustrates metric interpolation along a segment, for which the initial data are the endpoints metrics.

Several interpolation schemes have been proposed in [Alauzet 2003b, Alauzet 2003a] which are based on the simultaneous reduction. The main drawback of these approaches is that the interpolation operation is not commutative, and the result depends on the order in which the operations are performed when more than two metrics are involved. Moreover, some useful properties such as the maximum principle are not verified by these schemes. A consistent operational framework, the log-Euclidian framework, was introduced in [Arsigny 2006], which we use to design an interpolation scheme.

We first define the notion of metric logarithm and exponential:

$$\ln(\mathcal{M}) := \mathcal{R}\ln(\Lambda)\mathcal{R}^T \quad \text{and} \quad \exp(\mathcal{M}) := \mathcal{R}\exp(\Lambda)\mathcal{R}^T\,,$$

where $\ln(\Lambda) = \mathrm{diag}(\ln(\lambda_i))$ and $\exp(\Lambda) = \mathrm{diag}(\exp(\lambda_i))$. We can now define the **logarithmic addition** $\oplus$ and the *logarithmic scalar multiplication* $\odot$:

$$\begin{aligned} \mathcal{M}_1 \oplus \mathcal{M}_2 &:= \exp\left(\ln(\mathcal{M}_1) + \ln(\mathcal{M}_2)\right) \\ \alpha \odot \mathcal{M} &:= \exp\left(\alpha.\ln(\mathcal{M})\right) = \mathcal{M}^\alpha\,. \end{aligned}$$

The logarithmic addition is commutative and coincides with matrix multiplication whenever the two tensors $\mathcal{M}_1$ and $\mathcal{M}_2$ commute in the matrix sense. The space of metric tensors, supplied with the logarithmic addition $\oplus$ and the logarithmic scalar multiplication $\odot$ is a vector space.

We use the linear interpolation operator derived from the log-Euclidean framework. Let $(\mathbf{x}_i)_{i=1\ldots k}$ be a set of vertices and $(\mathcal{M}(\mathbf{x}_i))_{i=1\ldots k}$ their associated metrics. Then, for a point $\mathbf{x}$ of the domain such that:

$$\mathbf{x} = \sum_{i=1}^{k} \alpha_i.\mathbf{x}_i \quad \text{with} \quad \sum_{i=1}^{k} \alpha_i = 1\,,$$

the interpolated metric is defined by:

$$\mathcal{M}(\mathbf{x}) = \bigoplus_{i=1}^{k} \alpha_i \odot \mathcal{M}(\mathbf{x}_i) = \exp\left(\sum_{i=1}^{k} \alpha_i\, \ln(\mathcal{M}(\mathbf{x}_i))\right)\,. \tag{4.19}$$

This interpolation is commutative. Moreover, it has been demonstrated in [Arsigny 2006] that this interpolation preserves the maximum principle, *i.e.,* for an edge $\mathbf{pq}$ with endpoints metrics $\mathcal{M}(\mathbf{p})$ and $\mathcal{M}(\mathbf{q})$ such that $\det(\mathcal{M}(\mathbf{p})) < \det(\mathcal{M}(\mathbf{q}))$ then we have $\det(\mathcal{M}(\mathbf{p})) < \det(\mathcal{M}(\mathbf{p} + t\,\mathbf{pq})) < \det(\mathcal{M}(\mathbf{q}))$ for all $t \in [0,1]$.

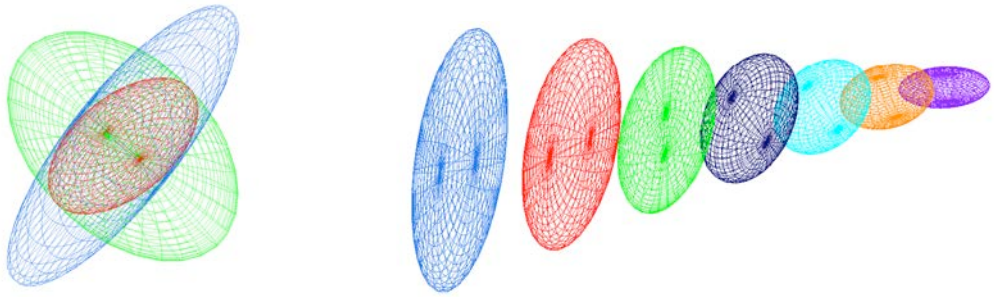

Figure 4.5: *Left, view illustrating the metric intersection procedure with the simultaneous reduction in 3D. In red, the resulting metric of the intersection of the blue and green metrics. Right, metric interpolation along a segment where the endpoints metrics are the blue and violet ones.*

4.2.4 The non-linear adaptation loop

So far, we have seen fundamental mathematical concepts used in metric-based mesh adaptation. Let us now see how they are involved in mesh adaptation.

Steady algorithm

The basic principle of metric-based mesh adaptation is: from an initial solution on an initial mesh, an error estimate is computed (based on an error estimate model), from which a metric field is computed, which is used to generate a mesh adapted to the solution. However, the mesh will be adapted to the numerical solution on the initial mesh, but not necessarily to the physical solution: if a new numerical solution is computed on the adapted mesh, new physical features may appear, to which the mesh is not adapted. This why the mesh adaptation problem is intrinsically a **non-linear** problem.

Therefore, an **iterative process** is necessary to converge the mesh/solution couple, that generates a sequence of consecutively adapted meshes. In the steady state case, there is only one mesh and one solution. The algorithm is illustrated in Figure 4.6. As input is given a pair of initial mesh and solution: $(\mathcal{H}^0, \mathcal{S}^0)$. (Note that the first solution can be the uniform solution or any partly converged solution). A first solution $\mathcal{S}^1$ is computed, from which a metric field $\mathcal{M}^1$ is deduced as will be explained further (Section 4.4), and a new mesh $\mathcal{H}^1$ is generated using this metric field. Solution $\mathcal{S}^1$ is then transferred to $\mathcal{H}^1$ (see Section 4.2.4). The new pair $(\mathcal{H}^1, \mathcal{S}^1)$ is then used as input for a new iteration of the algorithm.

The process ends when the solution stops evolving, for a given number of vertices, or formally speaking when an error threshold is reached. It is because the number of vertices is fixed that the process eventually converges: at some point, the vertices are optimally distributed, and only increasing the number of vertices could reduce the error. In practice, the number of vertices is given approximately through the notion of mesh complexity (see Section 4.3), and the number of iterations of the algorithm is set in advance to reach a good level of convergence (typically five iterations are performed).

In the following sections, we review several important steps of the adaptation algorithm.

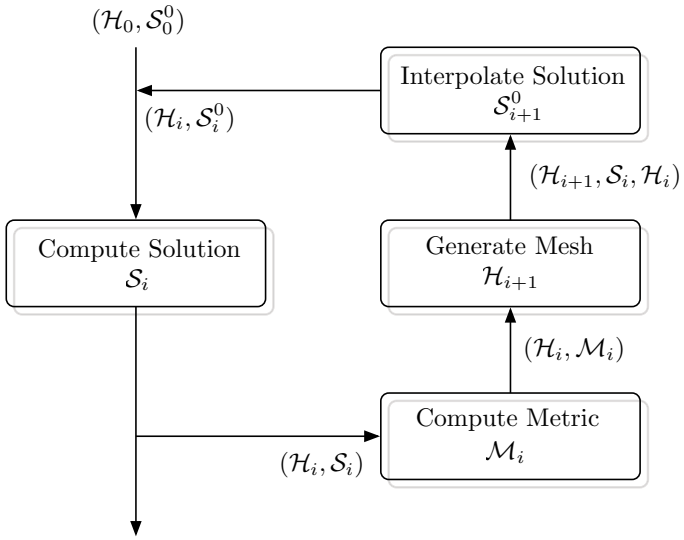

Figure 4.6: The non-linear steady adaptation loop. i is the adaptive loop iteration index, $\mathcal{H}^i$, $\mathcal{S}^i$, $\mathcal{S}_0^i$ and $\mathbf{M}_i$ denote the mesh, the solution, the initial solution and the metric field at iteration i, respectively.

Error estimates

The metric field is computed so that the mesh generated will minimize an error. This error is defined by an error estimate model, about which a lot will be said in the next chapters of this thesis. For now, let us just say that the error that we control is the interpolation error, under the assumption that the global error is controlled by the interpolation error, which is true for order two elliptic problems (See Céa's Lemma [Ciarlet 1991]). Several models were considered over time, but they all are based on the computation of the Hessian of the solution. It is indeed the variation of the first order derivatives that give us enough information on the features of the solution, notably on the directions of the features. It also has to be noted that Hessian matrices are $n \times n$ matrices, n being the space dimension, that are easily converted into metrics.

Choice of the sensor

The adaptation is performed on a sensor function, which is a function of the solution, and on which the modeled error is computed. Usually, one field of the solution vector is selected as sensor (density, momentum or pressure in the case of the Euler equations), but a combination of fields is also possible. The choice of this sensor is very important in the adaptation process, since it influences greatly the final solution. This problematic can be illustrated with the classic case of Sod shock tube (see Section 5.3): all the phenomena of the solution are not visible on some fields, for instance, the contact discontinuity cannot be observed on pressure curves. If the pressure was chosen as adaptation sensor, the area of the contact discontinuity would be meshed with large elements, leading to a loss of accuracy of the solution.

Gradient and Hessian recovery techniques

Since Hessians are the basis of the computation of metrics, it is important to compute them with enough accuracy. From a non smooth piecewise $\mathbb{P}^1$ function u_h, the chosen method must be able to recover a smooth Hessian field. Two main approaches coexist: a double L^2 projection method, and a weighted least square method. The second method is detailed in [Menier 2015], and is better in particular in the case of boundary layer meshes. The first one is the one generally used in this thesis, and works as follows.

Nodal gradients. Let $\mathbb{P}^0$ and $\mathbb{P}^1$ are the set of polynomials of degree respectively 0 (constant polynomials) and 1. Let K be a tetrahedron and $(P_i)_{i=0\dots3}$ its vertices. Let $\mathbf{x} \in K$. u_h is written:

$$u_h(\mathbf{x}) = \sum_{j=0}^{3} u_h(P_j)\, \varphi_j(\mathbf{x})\,,$$

where $(\varphi_j)_j$ are the $\mathbb{P}^1$ shape functions, given by:

$$\nabla_{\boldsymbol{x}}\varphi_0|_K = \frac{1}{6|K|}\overline{\boldsymbol{\eta_0}}, \quad \nabla_{\boldsymbol{x}}\varphi_1|_K = \frac{1}{6|K|}\overline{\boldsymbol{\eta_1}}, \quad \nabla_{\boldsymbol{x}}\varphi_2|_K = \frac{1}{6|K|}\overline{\boldsymbol{\eta_2}}, \quad \nabla_{\boldsymbol{x}}\varphi_3|_K = \frac{1}{6|K|}\overline{\boldsymbol{\eta_3}}\,,$$

where η_i denotes the outward normal of the face opposite to vertex P_i (see conventions on page 3).

The expression is differentiated term by term:

$$\nabla u_h|_K = \sum_{j=0}^{3} u_h(P_j)\, \nabla_{\mathbf{x}}\varphi_j|_K\,.$$

As ∇u_h is not defined at the vertices of the mesh, the nodal gradients are recovered from the gradients to the elements using a L^2 local projection and Clément's interpolation operator.

Let P_i be a vertex of $\mathcal{H}$. The stencil of the shape function φ_i is the ball of vertices around P_i, $\mathcal{B}(P_i)$. The following spaces are introduced:

$$\begin{aligned} V_h^0 &= \{v \in L^2(\Omega) \;\big|\; v_{|K} \in \mathbb{P}^0 \quad \forall K \in \mathcal{H}\} \\ V_h^1 &= \{v \in \mathcal{C}^0(\Omega) \;\big|\; v_{|K} \in \mathbb{P}^1 \quad \forall K \in \mathcal{H}\}\,. \end{aligned}$$

For $v \in L^2(\Omega)$, we set $\Pi_{L^2} v \in V_h^0$:

$$\forall\, \mathcal{B}(P_i) \subset \mathcal{H}\,, \quad \begin{cases} (\Pi_{L^2} v)_{|\mathcal{B}(P_i)} \in \mathbb{P}^0 \\ \int_{\mathcal{B}(P_i)} (\Pi_{L^2} v - v)\; w = 0\,, \;\; \forall w \in \mathbb{P}^0\,. \end{cases}$$

Clement's operator $\Pi_c : V_h^0 \longrightarrow V_h^1$ is defined as follows:

$$\Pi_c v = \sum_{i=0}^{n} \Pi_{L^2} v(P_i)\, \varphi_i\,.$$

With Clément's operator, we can recover nodal gradients from the $\nabla u_h \in \mathbb{P}^0$. For each $\mathcal{B}(P_i) \subset \mathcal{H}$ we can in particular take $v = 1 \in \mathbb{P}^0$ as test function:

$$\begin{aligned}
\int_{\mathcal{B}(P_i)} (\Pi_{L^2}(\nabla u_h) - \nabla u_h) = 0 \quad &\Longleftrightarrow \quad \int_{\mathcal{B}(P_i)} \Pi_{L^2}(\nabla u_h) = \int_{\mathcal{B}(P_i)} \nabla u_h \\
&\Longleftrightarrow \quad |\mathcal{B}(P_i)|\, \Pi_{L^2}(\nabla u_h)_{|\mathcal{B}(P_i)} = \sum_{K_j \in \mathcal{B}(P_i)} \int_{K_j} \nabla u_h \\
&\Longleftrightarrow \quad \Pi_{L^2}(\nabla u_h)_{|\mathcal{B}(P_i)} = \frac{\sum\limits_{K_j \in \mathcal{B}(P_i)} |K_j|\, \nabla u_{h|K_j}}{\sum\limits_{K_j \in \mathcal{B}(P_i)} |K_j|} ,
\end{aligned}$$

where $|K|$ is the volume of element K . For each vertex P_i, we can write:

$$\nabla_R\, u_h(P_i) = \frac{\sum\limits_{K_j \in \mathcal{B}(P_i)} |K_j|\, \nabla u_{h|K_j}}{\sum\limits_{K_j \in \mathcal{B}(P_i)} |K_j|} .$$

This procedure comes to recovering the gradient as an average of the gradients at the elements, weighted by the volume of the elements.

We thus have defined values to the vertices for the gradient, and consequently, thanks to Clément's operator, a piecewise $\mathbb{P}^1$ gradient on the mesh.

Hessian recovery. To recover the Hessian matrix by means of a double L^2 projection, the same procedure is applied once more, but using the gradient as input. Several methods are available. We chose to perform a double L^2 projection, which means applying the previous procedure once again but on the gradients.

Interpolation step

In the case of steady simulations, after each mesh adaptation iteration, the solution is not known on the newly generated mesh. Computing the solution from scratch being too costly in terms of CPU time, we prefer to transfer it from the old mesh to the new one, and restart the simulation from this already partly converged state.

The classic approach involves two steps: first the localization of the vertices of the new mesh in the background mesh, and then the application of an interpolation scheme. The localization step is described in several references [Frey 2008, Alauzet 2010b]. As concerns interpolation schemes, the easiest is the classic $\mathbb{P}^1$ interpolation, which is written:

$$u(P) = \sum_{i=0}^{3} \beta_i(P)\, u(Q_i) , \tag{4.20}$$

where P is a vertex of the new mesh that has been located in tetrahedron $K = [Q_0, Q_1, Q_2, Q_3]$ of the background mesh and β_i are the barycentric coordinates of $\mathbf{p}$ with respect to K. Higher order interpolation schemes can be devised, for instance using $\mathbb{P}^2$ Lagrange test functions. However these schemes do not conserve the mass.

For the adaptation of unsteady simulations, we will see later that this step is even more important because a large number of interpolations are performed, spoiling the accuracy of the solution. In the context of the resolution by a second order numerical scheme of a PDE system of conservation laws, such as the compressible Euler system, the interpolation method seems to have to satisfy the following properties to obtain a consistent mesh adaptation scheme: (i) mass conservation, (ii) $\mathbb{P}_1$ exactness preserving the second order of the adaptive strategy and (iii) verify the maximum principle.

A new conservation scheme was devised to match these criteria [Alauzet 2010b]. The mass conservation property of the interpolation operator is achieved by local mesh intersections, *i.e.,* intersections are performed at the element level. The use of mesh intersection to build a conservative interpolation process seems natural for unconnected meshes (*i.e.* meshes that are not nested). The idea is to find, for each element of the new mesh, its geometric intersection with all the elements of the background mesh it overlaps and to mesh this geometric intersection with simplices. We are then able to use a Gauss quadrature formula to exactly compute the mass which has been locally transferred.

High-order accuracy is obtained through the reconstruction of the gradient of the solution from the discrete data and the use of some Taylor formulas. Unfortunately, this high-order interpolation can lead to a loss of monotonicity. The maximum principle is recovered by correcting the interpolated solution in a conservative manner. Finally, the solution values at vertices are reconstructed from this piecewise linear by element discontinuous representation of the solution. The algorithm is summarized in Algorithm 3.

Algorithm 3 Conservative Interpolation Process

Piecewise linear (continuous or discontinuous) representation of the solution on $\mathcal{H}_{back}$

1. For all elements $K_{back} \in \mathcal{H}_{back}$, compute solution mass $m_{K_{back}}$ and gradient $\nabla_{K_{back}}$
2. For all elements $K_{new} \in \mathcal{H}_{new}$, recover solution mass $m_{K_{new}}$ and gradient $\nabla_{K_{new}}$:
   (a) compute the intersection of K_{new} with all $K^i_{back} \in \mathcal{H}_{back}$ it overlaps
   (b) mesh the intersection polygon/polyhedron of each couple of elements (K_{new}, K^i_{back})
   (c) compute $m_{K_{new}}$ and $\nabla_{K_{new}}$ using Gauss quadrature formulas

   $\Longrightarrow$ a piecewise linear discontinuous representation of the mass on $\mathcal{H}_{new}$ is obtained
3. Correct the gradient to enforce the maximum principle
4. Set the solution values to vertices by an averaging procedure.

Figure 4.7 points out the superiority of the $\mathbb{P}_1$-conservative solution transfer (right) compared with the classic $\mathbb{P}_1$ interpolation (left) on an adaptive blast simulation in two dimensions. For both simulations, all parameters are the same except for the solution transfer stage. This figure shows the final solution obtained with 70 mesh adaptations, *i.e.,* a total of 70 solution transfers. The diffusion and the error introduced by the classic $\mathbb{P}_1$ solution transfer clearly spoils the solution accuracy while the solution remains very accurate with the $\mathbb{P}_1$-conservative operator.

As regards CPU time overhead, it is minor in 2D but a major issue in 3D. Fortunately, the procedure is easily parallelized and scales very well, and the CPU overhead becomes admissible with the parallelization.

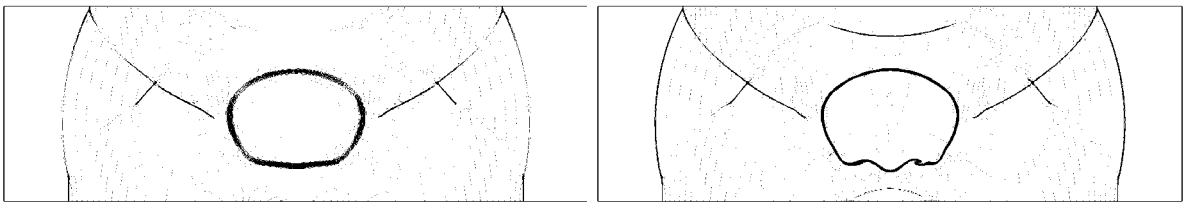

Figure 4.7: Final result of a blast adaptive simulation with 70 mesh adaptations. Left, using a classic $\mathbb{P}_1$ solution transfer. Right, using the $\mathbb{P}_1$-conservative solution transfer.

Mesh gradation

The size field provided by the metric field can have huge variations or be very irregular, especially since they are computed from a numerical solution with sharp features (discontinuities or steep gradients). This may result in poor quality anisotropic meshes. To generate high quality anisotropic meshes, the metric field needs to be smoothed: the sizes prescribed are bound to avoid too large variations. Several methods have been proposed [Borouchaki 1998, Alauzet 2010b]. This procedure should not impact the accuracy of the solution, since the elements can only have a smaller size than the one prescribed by the error model. However, if the procedure is not applied correctly, it may have a noticeable impact on the mesh sizes and anisotropy, and thus the step is worth mentioning.

4.3 The continuous mesh framework

In [Loseille 2008, Loseille 2011a, Loseille 2011b] the concept of **continuous mesh** was introduced, that has become central in our way to consider metric-based mesh adaptation. We have seen that Riemannian metric spaces are a useful tool to generate adapted meshes, but the continuous mesh framework brings them to another stage, and establishes a duality between the discrete domain and the continuous domain which is based of Riemannian metric fields. This duality enables a practical mathematical representation of adapted meshes, and allows us to use powerful tool from calculus of variations. This is particularly useful to derive appropriate error estimates, and find meshes that are optimal with respect to the error model.

The results for this section are presented in 3D, although they are easily extended to nD.

4.3.1 Duality discrete-continous: a new formalism

The key notion of unit element, defined in section 4.2.2, is once again fundamental to draw a correspondence between the discrete and the continuous domain. It is actually used to define classes of equivalence of discrete elements: 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

$|K| \neq 0$, then there is a unique metric tensor $\mathcal{M}$ for which element K is unit with respect to $\mathcal{M}$.

Consequently, a discrete element can be viewed as a discrete representative of an equivalence class formed by all the unit elements of a metric $\mathcal{M}$. Figure 4.8 depicts some unit elements with respect to a metric tensor, which is geometrically represented by its unit-ball. $\mathcal{M}$ denotes the class of equivalence of all the elements which are unit with respect to $\mathcal{M}$ and is called **continuous element**.

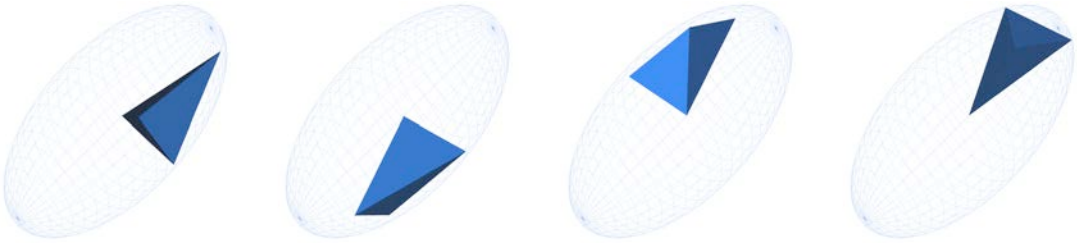

Figure 4.8: Several unit elements with respect to a metric tensor in 3D.

All the discrete representatives of a continuous element $\mathcal{M}$ share some common properties, called invariants, which justify the use of this equivalence relation. They connect the geometric properties of the discrete elements to the algebraic properties of metric $\mathcal{M}$. The two main invariants are:

- the edges $\mathbf{e_i}$ of any unit element K with respect to metric $\mathcal{M}$ are unit in $\mathcal{M}$:

$$\forall\,(\mathbf{e}_i, \mathbf{e}_j), \qquad \mathbf{e}_i^T\,\mathcal{M}\,\mathbf{e}_i = 1, \tag{4.21}$$

- conservation of the Euclidean volume for any unit element K with respect to metric $\mathcal{M}$

$$|K| = \frac{\sqrt{3}}{4}\,\det(\mathcal{M}^{-\frac{1}{2}}) \text{ in 2D} \quad \text{and} \quad |K| = \frac{\sqrt{2}}{12}\,\det(\mathcal{M}^{-\frac{1}{2}}) \text{ in 3D.} \tag{4.22}$$

This local relation of equivalence concerns elements, and needs to be somehow extended to whole meshes. Intuitively, the notion of Riemann metric space is going to play that role. The main difficulty is to take into account the variation of the function $\mathbf{x} \mapsto \mathcal{M}(\mathbf{x})$. The analysis can be simplified if $\mathbf{M}$ is rewritten as follows, separating its local and global properties. A Riemannian metric space $\mathbf{M} = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ locally writes:

$$\forall \mathbf{x} \in \Omega, \quad \mathcal{M}(\mathbf{x}) = d^{\frac{2}{3}}(\mathbf{x})\,\mathcal{R}(\mathbf{x}) \begin{pmatrix} r_1^{-\frac{2}{3}}(\mathbf{x}) & & \\ & r_2^{-\frac{2}{3}}(\mathbf{x}) & \\ & & r_3^{-\frac{2}{3}}(\mathbf{x}) \end{pmatrix} \mathcal{R}^T(\mathbf{x}), \tag{4.23}$$

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 it does not change the anisotropic properties or the orientation), while the anisotropy is given by the anisotropic quotients and the orientation by matrix $\mathcal{R}$. The notion of complexity $\mathcal{C}$ of $\mathbf{M}$ can also be defined:

$$\mathcal{C}(\mathbf{M}) = \int_\Omega d(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_\Omega \sqrt{\det(\mathcal{M}(\mathbf{x}))}\,\mathrm{d}\mathbf{x}. \tag{4.24}$$

This quantifies the global level of accuracy of $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$.

From this formulation of a Riemannian metric field arises a duality between meshes and Riemannian metric spaces. This duality is justified by the strict analogy between the following discrete and continuous notions: orientation vs. $\mathcal{R}$, stretching vs. r_i, size vs. d and number of vertices vs. $\mathcal{C}(\mathbf{M})$. However, the class of discrete meshes represented by $\mathbf{M}$ is complex to describe. Indeed, we have seen that a unit mesh cannot be defined as a set of strictly unit elements, but the notion of quasi-unit elements has to be invoked once again. The following definition is adopted: in the continuous mesh framework, a **continuous mesh** of a domain Ω is defined by a set of continuous elements $\mathbf{M} = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$, *i.e.,* a Riemannian metric space. It represents all the meshes that are unit for $\mathbf{M}$.

4.3.2 Continuous linear interpolation

The model that we have just defined is used to derive error estimates. The goal was to avoid to use upper bounds of the interpolation error to derive adapted meshes, as it is usually done in studies on the interpolation error, and to be able to compute the error for any function on any continuous mesh.

Let $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ be a continuous mesh of a domain Ω and let u be a non linear function which is assumed to be only twice continuously differentiable. We seek a well-posed definition of the continuous linear interpolation error $\|u - \pi_{\mathcal{M}}u\|_{L^1(\Omega)}$ related to a continuous mesh $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ which implies a well-posed definition of a linear continuous interpolate $\pi_{\mathcal{M}}u$. More precisely, we would like the continuous linear interpolation error to be a reliable mathematical model of $\|u - \Pi_h u\|_{L^1(\Omega_h)}$ where Π_h is defined by a mesh $\mathcal{H}$ of a discretized domain Ω_h which is a unit mesh with respect to $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$. The interpolation error is first derived locally (on an element) for a quadratic function, then is extended to a global definition (for any point of the domain).

Local continuous interpolate

Let us first consider a quadratic function u defined on a domain $\Omega \subset \mathbb{R}^3$, and a continuous element $\mathcal{M}$. For all unit elements K with respect to $\mathcal{M}$, the interpolation error of u in L^1 norm does not depend on the element shape and is only a function of the Hessian H_u of u and of continuous element $\mathcal{M}$.

- In 3D, for all unit elements K for $\mathcal{M}$, the following equality holds:

$$\|u - \Pi_h u\|_{L^1(K)} = \frac{\sqrt{2}}{240}\,\det(\mathcal{M}^{-\frac{1}{2}})\,\mathrm{trace}(\mathcal{M}^{-\frac{1}{2}}\,H_u\,\mathcal{M}^{-\frac{1}{2}}). \tag{4.25}$$

- In 2D, for all unit elements K for $\mathcal{M}$, the following equality holds:

$$\|u - \Pi_h u\|_{L^1(K)} = \frac{\sqrt{3}}{64} \det(\mathcal{M}^{-\frac{1}{2}}) \operatorname{trace}(\mathcal{M}^{-\frac{1}{2}} H_u \mathcal{M}^{-\frac{1}{2}}).$$

For all the discrete elements that are unit with respect to $\mathcal{M}$, the interpolation error is the same, and is only based on continuous quantities (metric and Hessian). This last remark is important, since it shows metrics contain all the information required to compute the interpolation error, and are thus well adapted for anisotropic control of this error.

Global continuous interpolate

To define a global continuous linear interpolate, the problem is once again how to move from an expression valid for *one continuous element* to an expression for the case in which *the metric varies point-wise*. Let us now suppose now that the continuous mesh $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ is varying and that the function u is no more quadratic but only twice continuously differentiable. Equality (4.25) does not hold anymore, but all the terms of the right-hand-side $\mathcal{M}$ and H are still well defined continuously.

In the vicinity of $\mathbf{a}$, u_Q is the quadratic approximation of smooth function u and $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ reduces to $\mathcal{M}(\mathbf{a})$ in the tangent space. There exists a unique function $\pi_{\mathcal{M}}$ such that:

$$\forall \mathbf{a} \in \Omega\,, \quad |u - \pi_{\mathcal{M}} u|(\mathbf{a}) \;=\; \frac{\|u_Q - \Pi_h u_Q\|_{L^1(K)}}{|K|} \;=\; \frac{1}{20} \operatorname{trace}\big(\mathcal{M}(\mathbf{a})^{-\frac{1}{2}} \,|H(\mathbf{a})|\, \mathcal{M}(\mathbf{a})^{-\frac{1}{2}}\big)\,, \tag{4.26}$$

for every K unit element with respect to $\mathcal{M}(\mathbf{a})$.

This result underlines another discrete-continuous duality by pointing out a continuous counterpart of the interpolation error. For this reason, the following formalism was adopted: $\pi_{\mathcal{M}}$ is called *continuous linear interpolate* and $|u - \pi_{\mathcal{M}} u|$ represents the continuous dual of the interpolation error. From a practical point of view, we deduce the following analogy. Given a unit mesh $\mathcal{H}$ of a domain Ω_h with respect to a continuous mesh $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$, the global interpolation error is:

$$\|u - \Pi_h u\|_{L^1(\Omega_h)} = \sum_{K\in\mathcal{H}} \|u - \Pi_h u\|_{L^1(K)}. \tag{4.27}$$

In the continuous case, the discrete summation becomes an integral:

$$\|u - \pi_{\mathcal{M}} u\|_{L^1(\Omega)} = \int_\Omega |u - \pi_{\mathcal{M}} u|(\mathbf{x})\, \mathrm{d}\mathbf{x}. \tag{4.28}$$

There is no global guarantee on the reliability of the continuous interpolation error given by Relation (4.28), and in particular, there is no *a priori* relationship between (4.27) and (4.28). The only guarantee is the local equivalence given by Equation (4.26). However, the local guarantee becomes global when the mesh is unit with respect to a constant metric tensor and when the function is quadratic. In the latter case, by neglecting error due to the boundary discretization, we have the equality:

$$\|u - \Pi_h u\|_{L^1(\Omega_h)} = \|u - \pi_{\mathcal{M}} u\|_{L^1(\Omega)}\,, \tag{4.29}$$

for all unit meshes $\mathcal{H}$ with respect to $(\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$.

Several examples are given in [Loseille 2011b], both analytic and numerical, that confirm the validity of this analogy. They show that the model is accurate and the equivalence (4.27)≈(4.28) is observed even for non quadratic functions and non-constant continuous meshes, and that the error due to the fact that the mesh generator generates edges with length not strictly equal to one is negligible. In particular, the range for the lengths of the edges given in Section 4.2.2 ensures reliable numerical results.

4.3.3 Summary

We have presented a framework that draws a correspondence between the discrete domain and the continuous domain. This framework is summarized in Table 4.1.

DISCRETE	CONTINUOUS
Element K	Metric tensor $\mathcal{M}$
Element volume $\lvert K\rvert$	$d^{-1} = \sqrt{\det(\mathcal{M}^{-1})}$
Mesh $\mathcal{H}$ of Ω_h	Riemannian metric space $\mathbf{M}(\mathbf{x}) = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$
Number of vertices N_v	Complexity $\mathcal{C}(\mathcal{M}) = \int_\Omega d(\mathbf{x})\,\mathrm{d}\mathbf{x}$
$\mathbb{P}^1$ interpolate Π_h	$\mathbb{P}^1$-continuous interpolate $\pi_\mathcal{M}$
Local element-wise interpolation error $e_h(K) = \lvert\lvert u - \Pi_h u\rvert\rvert_K$ $e_h(K) = \frac{\lvert K\rvert}{40}\sum_{i=1}^{6} \mathbf{e}_i^T \lvert H_u\rvert \mathbf{e}_i$ (for u quadratic)	Local point-wise interpolation error $e(\mathbf{x}) = (u - \pi_\mathcal{M} u)(\mathbf{x})$ $e(\mathbf{x}) = \frac{1}{20}\,\mathrm{trace}\big(\mathcal{M}(\mathbf{x})^{-\frac{1}{2}} \lvert H(\mathbf{x})\rvert \mathcal{M}(\mathbf{x})^{-\frac{1}{2}}\big)$
Global interpolation error $\sum_{K\in\mathcal{H}} \lvert\lvert u - \Pi_h u\rvert\rvert_K$	Global interpolation error $\int_{\mathbf{x}\in\Omega} e(\mathbf{x})\,\mathrm{d}\mathbf{x}$

Table 4.1: The continuous mesh model draws a correspondence between the discrete domain and the continuous domain.

4.4 Multiscale mesh adaptation

In the previous section, we have presented the continuous mesh model, on which is based a powerful framework to handle mathematically adapted meshes. It reinforces the use of metrics that was made previously, since it establishes a duality between metric fields and adapted meshes, and it allows us to work with metrics instead of discrete meshes.

In the following section, we explain how this framework is used for mesh adaptation.

4.4.1 Optimal control of the interpolation error and optimal meshes

Mesh adaptation consists in finding the mesh that minimizes a certain error on a domain, for a certain sensor function. As already mentioned previously, the error we consider is the interpolation error, that we control in L^p norm - depending on the choice of p, different aspects of the solution are captured, as will be seen later. The problem is stated *a priori*:

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

(P) is too complex to be solved directly, since the unknown, *i.e.* the mesh, is made up of vertices and their topology, which is far too many degrees of freedom. Besides, several optimal meshes can be found for one sensor function (think about merely swapping an element), so the problem is ill-posed. On the other hand, it is possible to show that the problem moved to the continuous domain is well posed, and can be solved using calculus of variations. What is more, the continuous formulation of the adaptation problem uses a global point of view, while usual methods focus on a local analysis of the error. The reformulated problem is:

$$\text{Find } \mathbf{M}_{opt} \text{ having a complexity of } N \text{ such that } \mathbf{E}_{L^p}(\mathbf{M}_{opt}) = \min_{\mathbf{M}} \|u - \pi_{\mathcal{M}} u\|_{L^p(\Omega)}. \qquad (4.30)$$

Using the definition of the linear continuous interpolate $\pi_{\mathcal{M}}$ given by Equation (4.26), the well-posed global optimization problem of finding the optimal continuous mesh minimizing the continuous interpolation error in L^p norm can be established:

$$\begin{aligned} \text{Find } \mathbf{M}_{L^p} = \min_{\mathbf{M}} \mathbf{E}_{L^p}(\mathbf{M}) &= \left(\int_\Omega \left(u(\mathbf{x}) - \pi_{\mathcal{M}} u(\mathbf{x}) \right)^p \mathrm{d}\mathbf{x} \right)^{\frac{1}{p}} \\ &= \left(\int_\Omega \mathrm{trace}\left(\mathcal{M}(\mathbf{x})^{-\frac{1}{2}} |H_u(\mathbf{x})| \mathcal{M}(\mathbf{x})^{-\frac{1}{2}} \right)^p \mathrm{d}\mathbf{x} \right)^{\frac{1}{p}}, \end{aligned} \qquad (4.31)$$

under the constraint $\mathcal{C}(\mathbf{M}) = \displaystyle\int_\Omega d(\mathbf{x})\, \mathrm{d}\mathbf{x} = N$. The constraint on the complexity notably avoids the trivial solution where all $(h_i)_{i=1,3}$ are zero which provides a null error.

Contrary to a discrete analysis, this problem can be solved globally by using a calculus of variations that is well-defined on the space of continuous meshes. In [Loseille 2011b], it is proved that Problem (4.31) admits a unique solution. In addition, the following properties hold. Let u be a twice continuously differentiable function defined on $\Omega \subset \mathbb{R}^3$, H_u its Hessian, the optimal

continuous mesh $\mathbf{M}_{L^p}(u)$ minimizing Problem (4.31) reads locally:

$$\mathcal{M}_{L^p}(\mathbf{x}) = N^{\frac{2}{3}} \left(\int_\Omega \det(|H_u(\bar{\mathbf{x}})|)^{\frac{p}{2p+3}} \, \mathrm{d}\bar{\mathbf{x}} \right)^{-\frac{2}{3}} \det(|H_u(\mathbf{x})|)^{\frac{-1}{2p+3}} \, |H_u(\mathbf{x})| \, . \tag{4.32}$$

It verifies the following properties:

- $\mathbf{M}_{L^p}(u)$ is unique
- $\mathbf{M}_{L^p}(u)$ is locally aligned with the eigenvectors basis of H_u and has the same anisotropic quotients as H_u
- $\mathbf{M}_{L^p}(u)$ provides an optimal explicit bound of the interpolation error in L^p norm:

$$\|u - \pi_{\mathcal{M}_{L^p}} u\|_{L^p(\Omega)} = 3 \, N^{-\frac{2}{3}} \left(\int_\Omega \det \left(|H_u| \right)^{\frac{p}{2p+3}} \right)^{\frac{2p+3}{3p}} . \tag{4.33}$$

- For a sequence of continuous meshes having an increasing complexity with the same orientation and anisotropic quotients $(\mathbf{M}^N_{L^p}(u))_{N=1\ldots\infty}$, the asymptotic order of convergence verifies:

$$\|u - \pi_{\mathcal{M}^N_{L^p}} u\|_{L^p(\Omega)} \leq \frac{Cst}{N^{2/3}}. \tag{4.34}$$

  Relation (4.34) points out a global second order of mesh convergence.

Controlling the approximation error

In practice, we do not know the whole solution, but only the solution at the vertices of the mesh. Let us now 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 is not applied directly to u nor to u_h.

Let $\bar{V}^k_h$ be the space of piecewise polynomials of degree k (possibly discontinuous) and V^k_h 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 [Zienkiewicz 1992a], a hierarchical basis [Bank 1993], or an operator connected to an a posteriori estimate [Huang 2010a]. We assume that the reconstruction $R_h u_h$ is more accurate than u_h for a given norm $\|.\|$ in the sense that:

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

From the triangle inequality, we deduce:

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

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

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

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

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

Remark 5. *It is important to note that $\mathcal{M}_{L^p}$ defined by Relation (4.32) applied to $R_h u_h$ does not allow us to generate an optimal adapted mesh to control the approximation error $\|u - u_h\|$. If all assumptions are verified, this analysis states that such generated adapted meshes control the approximation error (we have only an upper bound).*

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. This operator was already used for gradients and Hessian matrices reconstruction in Section 4.2.4 As approximate solution u_h is only known at mesh vertices, we need to reconstruct mid-edge values. To this end, we consider the L^2-projection operator $\mathcal{P} : \bar{V}_h^0 \to V_h^1$ defined by [Clément 1975]:

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

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

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

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

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

where $\psi_{\mathbf{p}} = \phi_{\mathbf{p}}\,(2\,\phi_{\mathbf{p}} - 1)$ and $\psi_{\mathbf{e}} = 4\,\phi_{\mathbf{p}}\,\phi_{\mathbf{q}}$ are the $\mathbb{P}_2$ 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:

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

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, as explained in Section 4.2.4. This recovery procedure is similar to the ones of [Zienkiewicz 1992a, Zienkiewicz 1992b]. Other reconstruction operators can be applied such as the double L^2-projection, the least square method or eventually the Green formula based approach [Frey 2005].

4.4.2 Control of the error in L^p norm

The classic analysis for mesh adaptation [Alauzet 2003a, Castro-Díaz 1997] was performed in L^∞ norm. However this resulted in a loss of anisotropy of the adapted meshes in some cases [Loseille 2007]. Indeed, let us consider the Heaviside function with a $\delta > 0$ step: $u(x) = \delta$ if $x > 0$ and $u(x) = 0$ if $x \leq 0$, on the segment $[-1, 1]$. Given a uniform mesh of this domain with size parameter h, then we can demonstrate that the interpolation error in L^p norm converges at order $O(h^{\frac{1}{p}})$ and thus the spatial convergence order is $O(h^{\frac{1}{p}})$. When p tends to infinity, the expected convergence order is $O(1)$. In other words, the L^∞ norm will never converge in presence of discontinuities. In practice, thanks to the prescription of a minimal size, the algorithm does not diverge. An L^∞ error estimate is thus not suitable for solutions with discontinuities. Consequently, L^p strategies become of main interest in the case of discontinuous solutions.

Another problem with the use of L^∞ error estimates is that a control of the interpolation error in L^∞ does not capture the small-scale features of the solution. Several modifications of the L^∞ interpolation error estimate have been considered [Castro-Díaz 1997, Löhner 1990b, Frey 2005] to overcome this issue. However, such local normalizations are only slightly more sensitive and the control of the interpolation error remains in L^∞ norm. Another interest of the optimal metric in L^p norm lies in its ability to catch physical phenomena of the solution which are of different scales. To illustrate this, we copy here the analytical example given in [Loseille 2011b]. We consider a function f_1 which is a smooth function involvingvariations of small and large amplitudes. The function is defined as follows:

$$f_1(x,y) = \begin{cases} 0.01\sin(50xy) & \text{if} \quad xy \leq \dfrac{\pi}{50}, \\ \sin(50xy) & \text{else if} \quad xy \leq 2\dfrac{\pi}{50}, \\ 0.01\sin(50xy) & \text{elsewhere.} \end{cases}$$

This function is composed of variations having a unit amplitude along with small variations having an amplitude of 0.01. This feature is illustrated in Figure 4.9 (top left) where a cut through the line $y = 0$ is depicted.

The mesh adaptation process based on the control of the interpolation error is analyzed for the L^1, L^2 and L^4 norms. Figure 4.9 shows adapted meshes composed of almost 7 000 vertices for each norm. We observe that the small amplitude waves regions are better captured when using a L^p norm with a lower p (L^1 is the best) whereas the L^4 norm ignores small amplitudes regions and clearly refines more large amplitudes areas. This behavior is due to the term $\det |H_u|^{\frac{-1}{2p+n}}$ in Relation (4.32) which gives more sensitivity to lower p norm. It illustrates that controlling the interpolation error in L^p norm with a small value of p enables to capture all the scales of the solution.

4.5 Conclusion

In this chapter, we have presented the fundamental concepts of metric based mesh adaptation. It lies on the concept of Riemannian metric field, that allows us to compute distances varying

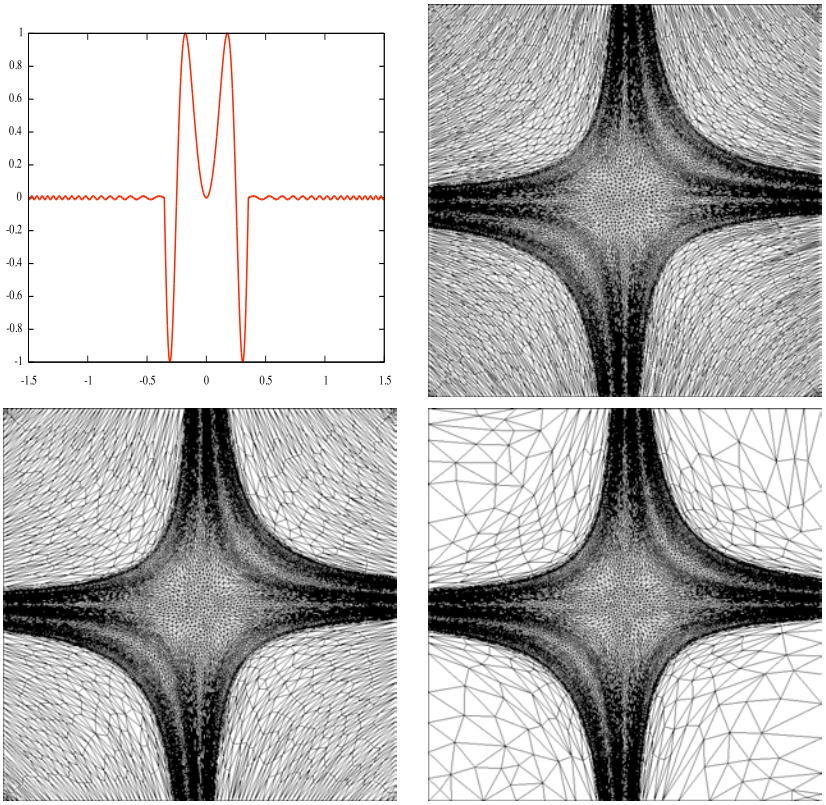

Figure 4.9: Top left, representation of function f_1 along the cut line $y = 0$. Optimal adapted meshes for norms L^1 (top right), L^2 (bottom left) and L^4 (bottom right). Each mesh is composed of about 7 000 vertices.

on the computational domain. An adapted mesh with respect to a metric field is a unit mesh, *i.e* a mesh whose edges have lengths close to 1 in the considered metric field. Adapting a mesh to a solution means minimizing the error of the solution on the mesh. Several error models can be considered, in this work they are based on the interpolation error. This error model is used to compute the metric fields from which the adapted mesh is generated. An iterative process is used to converge the couple mesh/solution. The introduction of the continuous mesh model, that draws a formal correspondence between discrete entities such as elements and continuous entities such as metric fields, allows us to derive more precise error models, and to perform multiscale mesh adaption. In this chapter, the basic adaptation loop for steady cases was presented, in the following chapter, we are going to see how it is extended to unsteady simulations.

Chapter 5

Unsteady mesh adaptation

In the previous chapter, mesh adaptation has been considered for steady problems only, for which there is only one solution and one mesh. In this chapter, we are going to extend the concept to unsteady simulations, where the solution is time dependent: there are now an infinity of solutions.

Generating one adapted mesh for every solver time step is not conceivable for industrial applications, because the accuracy required results in too many very small time steps and thus too many meshes to generate, which is very time consuming. Adapting the mesh every few solver iterations (*i.e.* adapting it at time t^n then keeping the same mesh for the next few time steps) is not satisfactory either, because there is a time shift between the mesh and the solution: the mesh is adapted at time t^n, but not t^{n+1}, t^{n+2}... The evolution of the solution in the adaptation interval has to be anticipated to generate a good mesh.

This lead to the idea of splitting the simulation interval into several sub-intervals, and considering one mesh for each sub-interval, adapted to the solution at every time of the sub-interval. The error estimate described for the steady state has to be modified to take into account this whole time frame.

Furthermore, the non-linearity of the adaptation problem that was highlighted in the steady case is still present in the unsteady case, and needs to be addressed. The same idea of convergence of the couple mesh/solution is applied, except that now we have several meshes and solutions. In the first works on metric-based adaptation for unsteady flows [Alauzet 2007], each couple mesh/solution on one sub-interval could be converged independently from the others. In other words, the adapted mesh for a sub-interval only depended on the solution on this sub-interval, and one could apply a fixed-point procedure similar to the steady-case procedure for each sub-interval. Recently, a new error analysis was carried out, which is based on the continuous mesh model. In this analysis, an effort is made to control the error globally - in a way that will be detailed farther - which results in a global fixed-point procedure: a couple mesh/solution is not converged alone, but the whole set of meshes and the whole solution are converged together.

This chapter aims at describing the whole adaptation loop, from theory to practice. We detail our space-time error analysis based on the continuous mesh model, and present the resulting unsteady adaptation algorithm. Then we describe how the algorithm is implemented in practice, and we present a comparison of two methods to compute a quantity called average Hessian-metric involved in the algorithm. Finally, numerical examples are given, and the parallelization of the loop is analyzed.

In this chapter, the solver used in the examples is `Wolf`, whose ALE version was presented in Chapter 2. Here, a standard non ALE version is used, but the spatial and temporal schemes are the same.

The works presented in this Chapter were published in [Barral 2014c, Barral 2015].

5.1 Error estimate

Let us first detail the error analysis that leads to the adaptation algorithm for unsteady simulations. This analysis is performed in the context of the continuous mesh model: first, an error model is proposed, then this error is minimized to derive an optimal mesh.

In the context of time-dependent problems, the error analysis for the steady case detailed in Section 4.4 is not sufficient, since it controls only spatial errors, whereas temporal errors need to be addressed too. In this work, we do not account for time discretization errors but we focus on a space-time analysis of the spatial error. In other words, we seek for the optimal space-time mesh controlling the space-time spatial discretization error. For the type of simulations that are considered in this work, the assumption is made that *as an explicit time scheme is used for time advancing, then the error in time is controlled by the error in space under CFL condition.* This has been demonstrated under specific conditions in [Alauzet 2007]. As long as this hypothesis holds, the spatial interpolation error provides a fair measure of the total space-time error of the discretized unsteady system. Notably in the case of implicit time advancing solvers, the analysis would have to be completed to take into account the temporal discretization error, as is done in [Coupez 2013].

5.1.1 Error model

Our goal is to solve an unsteady PDE which is set in the computational space-time domain $\mathcal{Q} = \Omega \times [0,T]$ where T is the (positive) maximal time and $\Omega \subset \mathbb{R}^3$ is the spatial domain. Let Π_h be the usual $\mathbb{P}^1$ projector, we extend it to time-dependent functions:

$$\left(\Pi_h \varphi\right)(t) = \Pi_h\left(\varphi(t)\right), \ \forall\, t \in \ [0,T]\,. \tag{5.1}$$

The considered problem of mesh adaptation consists in finding the space-time mesh $\mathcal{H}$ of Ω that minimizes the space-time linear interpolation error $u - \Pi_h u$ in L^p norm. The problem is thus stated in an a priori way:

$$\text{Find } \mathcal{H}_{opt} \text{ having } N_{st} \text{ vertices such that } \ \mathbf{E}_{L^p}(\mathcal{H}_{opt}) = \min_{\mathcal{H}} \|u - \Pi_h u\|_{L^p(\Omega_h \times [0,T])}\,. \tag{5.2}$$

This problem is ill-posed and has far too many unknowns to be solved directly, as was explained previously. So it is rewritten in the continuous mesh framework under its continuous form:

$$\text{Find } \mathbf{M}_{L^p} = \left(\mathcal{M}_{L^p}(\mathbf{x},t)\right)_{(\mathbf{x},t)\in\mathcal{Q}} \ \text{ such that } \ \mathbf{E}_{L^p}(\mathbf{M}_{L^p}) = \min_{\mathbf{M}} \|u - \pi_{\mathcal{M}} u\|_{L^p(\Omega\times[0,T])}\,, \tag{5.3}$$

under the space-time constraint:

$$\mathcal{C}_{st}(\mathbf{M}) = \int_0^T \tau(t)^{-1} \left(\int_\Omega d_{\mathcal{M}}(\mathbf{x},t)\ \mathrm{d}\mathbf{x} \right) \mathrm{d}t = N_{st}\,. \tag{5.4}$$

where $\tau(t)$ is the time step used at time t of interval $[0,T]$. Introducing the continuous interpolation error, we recall that we can write the continuous error model as follows:

$$\mathbf{E}_{L^p}(\mathbf{M}) = \left(\int_0^T \int_\Omega \mathrm{trace}\left(\mathcal{M}^{-\frac{1}{2}}(\mathbf{x},t) |H_u(\mathbf{x},t)| \mathcal{M}^{-\frac{1}{2}}(\mathbf{x},t) \right)^p \ \mathrm{d}\mathbf{x}\ \mathrm{d}t \right)^{\frac{1}{p}}. \tag{5.5}$$

where H_u is the Hessian of sensor u. To find the optimal space-time continuous mesh, Problem (5.3-5.4) is solved in two steps:

(i) First, a spatial minimization is done for a fixed t.

(ii) Second, a temporal minimization is performed.

Note that both minimizations are performed formally.

5.1.2 Spatial minimization for a fixed t

Let us assume that at time t, we seek for the optimal continuous mesh $\mathbf{M}_{L^p}(t)$ which minimizes the instantaneous error, *i.e.,* the spatial error for a fixed time t:

$$\tilde{\mathbf{E}}_{L^p}(\mathbf{M}(t)) = \int_\Omega \text{trace}\left(\mathcal{M}^{-\frac{1}{2}}(\mathbf{x},t)\,|H_u(\mathbf{x},t)|\,\mathcal{M}^{-\frac{1}{2}}(\mathbf{x},t)\right)^p \,\mathrm{d}\mathbf{x}\,,$$

under the constraint that the number of vertices is prescribed to

$$\mathcal{C}(\mathbf{M}(t)) = \int_\Omega d_{\mathcal{M}(t)}(\mathbf{x},t)\,\mathrm{d}\mathbf{x} = N(t). \tag{5.6}$$

Similarly to 4.4, solving the optimality conditions provides the *optimal instantaneous continuous mesh in L^p-norm* $\mathbf{M}_{L^p}(t) = (\mathcal{M}_{L^p}(\mathbf{x},t))_{\mathbf{x}\in\Omega}$ at time t defined by:

$$\mathcal{M}_{L^p}(\mathbf{x},t) = N(t)^{\frac{2}{3}}\,\mathcal{M}_{L^p,1}(\mathbf{x},t)\,, \tag{5.7}$$

where $\mathcal{M}_{L^p,1}$ is the optimum for $\mathcal{C}(\mathbf{M}(t)) = 1$:

$$\mathcal{M}_{L^p,1}(\mathbf{x},t) = \left(\int_\Omega (\det|H_u(\bar{\mathbf{x}},t)|)^{\frac{p}{2p+3}}\mathrm{d}\bar{\mathbf{x}}\right)^{-\frac{2}{3}} (\det|H_u(\mathbf{x},t)|)^{-\frac{1}{2p+3}}\;|H_u(\mathbf{x},t)|. \tag{5.8}$$

The corresponding optimal instantaneous error at time t writes:

$$\tilde{\mathbf{E}}_{L^p}(\mathbf{M}_{L^p}(t)) = 3^p\,N(t)^{-\frac{2p}{3}}\left(\int_\Omega (\det|H_u(\mathbf{x},t)|)^{\frac{p}{2p+3}}\mathrm{d}\mathbf{x}\right)^{\frac{2p+3}{3}} = 3^p\,N(t)^{-\frac{2p}{3}}\,\mathcal{K}(t)\,. \tag{5.9}$$

For the sequel of this thesis we denote: $\mathcal{K}(t) = \left(\int_\Omega (\det|H_u(\mathbf{x},t)|)^{\frac{p}{2p+3}}\mathrm{d}\mathbf{x}\right)^{\frac{2p+3}{3}}$.

5.1.3 Temporal minimization

To complete the resolution of optimization Problem (5.3-5.4), a temporal minimization is performed in order to get the optimal space-time continuous mesh. In other words, we need to find the optimal time law $t \to N(t)$ for the instantaneous mesh size. In this work, we consider the case where the time step τ is specified by the user as a function of time $t \to \tau(t)$. The analysis can be extended to the case of an explicit time advancing solver subject to Courant time step condition [Alauzet 2012], but the resulting optimal mesh is too complex to be used in practice.

Let the time step τ be specified by a function of time $t \to \tau(t)$. After the spatial optimization, the space-time error writes:

$$\mathbf{E}_{L^p}(\mathbf{M}_{L^p}) = \left(\int_0^T \tilde{\mathbf{E}}_{L^p}(\mathbf{M}_{L^p}(t))\, \mathrm{d}t \right)^{\frac{1}{p}} = 3 \left(\int_0^T N(t)^{-\frac{2p}{3}}\, \mathcal{K}(t)\, \mathrm{d}t \right)^{\frac{1}{p}}, \tag{5.10}$$

and we aim at minimizing it under the following space-time complexity constraint:

$$\int_0^T \tau(t)^{-1} N(t)\, \mathrm{d}t = N_{st}. \tag{5.11}$$

We are trying to find *the optimal distribution of $N(t)$ when the space-time total number of nodes N_{st} is prescribed.* We can apply the following one-to-one change of variables:

$$\tilde{N}(t) = N(t)\tau(t)^{-1} \quad \text{and} \quad \tilde{\mathcal{K}}(t) = \tau(t)^{-\frac{2p}{3}}\, \mathcal{K}(t)\,.$$

Then, the temporal optimization problem becomes:

$$\min_{\mathbf{M}} \ \mathbf{E}_{L^p}(\mathbf{M})^p = 3^p \int_0^T \tilde{N}(t)^{-\frac{2p}{3}}\, \tilde{\mathcal{K}}(t)\, \mathrm{d}t \quad \text{under constraint} \quad \int_0^T \tilde{N}(t)\, \mathrm{d}t = N_{st}\,.$$

The solution of this problem is given by:

$$\tilde{N}_{opt}(t)^{-\frac{2p+3}{3}}\, \tilde{\mathcal{K}}(t) = const \quad \Rightarrow \quad N_{opt}(t) = C(N_{st})\ (\tau(t)\, \mathcal{K}(t))^{\frac{3}{2p+3}}\,.$$

Here, constant $C(N_{st})$ can be obtained by introducing the above expression in space-time complexity Constraint (5.11), leading to:

$$C(N_{st}) = \left(\int_0^T \tau(t)^{-\frac{2p}{2p+3}}\, \mathcal{K}(t)^{\frac{3}{2p+3}} \mathrm{d}t \right)^{-1} N_{st}\,,$$

which completes the description of the optimal space-time metric for a prescribed time step. Using Relations (5.7) and (5.8), the analytic expression of the optimal space-time metric in L^p-norm $\mathbf{M}_{L^p}$ writes:

$$\begin{aligned} \mathcal{M}_{L^p}(\mathbf{x}, t) = N_{st}^{\frac{2}{3}} \left(\int_0^T \tau(t)^{-\frac{2p}{2p+3}} \left(\int_\Omega (\det |H_u(\bar{\mathbf{x}}, t)|)^{\frac{p}{2p+3}} \mathrm{d}\bar{\mathbf{x}} \right) \mathrm{d}t \right)^{-\frac{2}{3}} \\ \tau(t)^{\frac{2}{2p+3}} (\det |H_u(\mathbf{x}, t)|)^{-\frac{1}{2p+3}} |H_u(\mathbf{x}, t)|\,. \end{aligned} \tag{5.12}$$

We get the following optimal error:

$$\mathbf{E}_{L^p}(\mathbf{M}_{L^p}) = 3\, N_{st}^{-\frac{2}{3}} \left(\int_0^T \tau(t)^{-\frac{2p}{2p+3}} \left(\int_\Omega (\det |H_u(\mathbf{x}, t)|)^{\frac{p}{2p+3}} \mathrm{d}\mathbf{x} \right) \mathrm{d}t \right)^{\frac{2p+3}{3p}}. \tag{5.13}$$

5.1.4 Error analysis for time sub-intervals

The previous analysis provides the optimal size of the adapted meshes for each time level. Hence, this analysis requires the mesh to be adapted at each flow solver time step. In practice this approach involves a very large number of remeshings which is very CPU consuming and spoils solution accuracy due to many solution transfers. In consequence, an adaptive strategy has been proposed in [Alauzet 2007, Alauzet 2011b] where the number of remeshings is controlled (thus drastically reduced) by considering a coarse adapted discretization of the time axis, and generating adapted meshes for several solver time steps.

The idea is to split the simulation time interval into n_{adap} *sub-intervals* $[t_{i-1}, t_i]$ for $i = 1, .., n_{adap}$. Each spatial mesh $\mathbf{M}^i$ is then kept constant during each sub-interval $[t_{i-1}, t_i]$. We could consider this partition as a *time discretization of the mesh adaptation problem*. In other words, the number of nodes N^i of the i^{th} adapted mesh $\mathbf{M}^i$ on sub-interval $[t_{i-1}, t_i]$ should for example be taken equal to:

$$N^i = \frac{\int_{t_{i-1}}^{t_i} N_{opt}(t)\tau(t)^{-1}\mathrm{d}t}{\int_{t_{i-1}}^{t_i} \tau(t)^{-1}\mathrm{d}t}.$$

Here, we propose a different option in which we get an optimal discrete answer.

Spatial minimization on a sub-interval

Given the continuous mesh complexity N^i for the single adapted mesh used during time sub-interval $[t_{i-1}, t_i]$, we seek for the optimal continuous mesh $\mathbf{M}^i_{L^p}$ solution of the following problem:

$$\min_{\mathbf{M}^i} \mathbf{E}^i_{L^p}(\mathbf{M}^i) = \int_\Omega \text{trace}\left((\mathcal{M}^i)^{-\frac{1}{2}}(\mathbf{x})\,\mathbf{H}^i_u(\mathbf{x})\,(\mathcal{M}^i)^{-\frac{1}{2}}(\mathbf{x})\right)^p \mathrm{d}\mathbf{x} \quad \text{such that} \quad \mathcal{C}(\mathbf{M}^i) = N^i\,, \tag{5.14}$$

where matrix $\mathbf{H}^i_u$ on the sub-interval can be defined by either using an L^1 or an L^∞ norm average:

$$\mathbf{H}^i_{L^1}(\mathbf{x}) = \int_{t_{i-1}}^{t_i} |H_u(\mathbf{x}, t)|\,\mathrm{d}t \quad \text{or} \quad \mathbf{H}^i_{L^\infty}(\mathbf{x}) = \Delta t_i \max_{t\in[t_{i-1},t_i]} |H_u(\mathbf{x}, t)|\,,$$

with $\Delta t_i = t_i - t_{i-1}$. Processing as previously, we get the spatial optimality condition:

$$\mathcal{M}^i_{L^p}(\mathbf{x}) = (N^i)^{\frac{2}{3}}\ \mathcal{M}^i_{L^p,1}(\mathbf{x})$$

with

$$\mathcal{M}^i_{L^p,1}(\mathbf{x}) = \left(\int_\Omega (\det \mathbf{H}^i_u(\bar{\mathbf{x}}))^{\frac{p}{2p+3}}\mathrm{d}\bar{\mathbf{x}}\right)^{-\frac{2}{3}} (\det \mathbf{H}^i_u(\mathbf{x}))^{-\frac{1}{2p+3}}\ \mathbf{H}^i_u(\mathbf{x}).$$

The corresponding optimal error $\mathbf{E}^i(\mathbf{M}^i_{L^p})$ writes:

$$\mathbf{E}^i_{L^p}(\mathbf{M}^i_{L^p}) = 3^p\,(N^i)^{-\frac{2p}{3}} \left(\int_\Omega (\det \mathbf{H}^i_u(\mathbf{x}))^{\frac{p}{2p+3}}\mathrm{d}\mathbf{x}\right)^{\frac{2p+3}{3}} = 3^p\,(N^i)^{-\frac{2p}{3}}\,\mathcal{K}^i\,. \tag{5.15}$$

where $\mathcal{K}^i = \left(\int_\Omega \left(\det \mathbf{H}^i_u(\mathbf{x})\right)^{\frac{p}{2p+3}}\,\mathrm{d}\mathbf{x}\right)^{\frac{2p+3}{3}}$.

To complete our analysis, we shall perform a temporal minimization. Again, we consider the case where the time step τ is specified by a function of time.

Temporal minimization for specified τ.

After the spatial minimization, the temporal optimization problem reads:

$$\min_{\mathbf{M}} \mathbf{E}_{L^p}(\mathbf{M})^p = \sum_{i=1}^{n_{adap}} \mathbf{E}^i_{L^p}(\mathbf{M}^i_{L^p}) = 3^p \sum_{i=1}^{n_{adap}} (N^i)^{-\frac{2p}{3}} \mathcal{K}^i$$

under the constraint:

$$\sum_{i=1}^{n_{adap}} N^i \left(\int_{t_{i-1}}^{t_i} \tau(t)^{-1} \mathrm{d}t \right) = N_{st} \, .$$

We set the one-to-one mapping:

$$\tilde{N}^i = N^i \left(\int_{t_{i-1}}^{t_i} \tau(t)^{-1} \mathrm{d}t \right) \quad \text{and} \quad \tilde{\mathcal{K}}^i = \mathcal{K}^i \left(\int_{t_{i-1}}^{t_i} \tau(t)^{-1} \mathrm{d}t \right)^{\frac{2p}{3}} ,$$

then the optimization problem reduces to:

$$\min_{\mathbf{M}} \sum_{i=1}^{n_{adap}} (\tilde{N}^i)^{-\frac{2p}{3}} \tilde{\mathcal{K}}^i \quad \text{such that} \quad \sum_{i=1}^{n_{adap}} \tilde{N}^i = N_{st} \, .$$

The solution is:

$$\tilde{N}^i_{opt} = \mathcal{C}(N_{st}) \, (\tilde{\mathcal{K}}^i)^{\frac{3}{2p+3}} \quad \text{with} \quad \mathcal{C}(N_{st}) = N_{st} \left(\sum_{i=1}^{n_{adap}} (\tilde{\mathcal{K}}^i)^{\frac{3}{2p+3}} \right)^{-1}$$

$$\Rightarrow \quad N^i = N_{st} \left(\sum_{i=1}^{n_{adap}} (\mathcal{K}^i)^{\frac{3}{2p+3}} \left(\int_{t_{i-1}}^{t_i} \tau(t)^{-1} \mathrm{d}t \right)^{\frac{2p}{2p+3}} \right)^{-1} (\mathcal{K}^i)^{\frac{3}{2p+3}} \left(\int_{t_{i-1}}^{t_i} \tau(t)^{-1} \mathrm{d}t \right)^{-\frac{3}{2p+3}} .$$

and we deduce the following optimal continuous mesh $\mathbf{M}_{L^p} = \{\mathbf{M}^i_{L^p}\}_{i=1,..,n_{adap}}$ and error:

$$\begin{aligned} \mathcal{M}^i_{L^p}(\mathbf{x}) = N_{st}^{\frac{2}{3}} \Bigg(\sum_{i=1}^{n_{adap}} (\mathcal{K}^i)^{\frac{3}{2p+3}} \Big(\int_{t_{i-1}}^{t_i} \tau(t)^{-1} \mathrm{d}t \Big)^{\frac{2p}{2p+3}} \Bigg)^{-\frac{2}{3}} \\ \Big(\int_{t_{i-1}}^{t_i} \tau(t)^{-1} \mathrm{d}t \Big)^{-\frac{2}{2p+3}} (\det \mathbf{H}^i_u(\mathbf{x}))^{-\frac{1}{2p+3}} \, \mathbf{H}^i_u(\mathbf{x}) \, , \end{aligned} \tag{5.16}$$

$$\mathbf{E}_{L^p}(\mathbf{M}_{L^p}) = 3 \, N_{st}^{-\frac{2}{3}} \left(\sum_{i=1}^{n_{adap}} (\mathcal{K}^i)^{\frac{3}{2p+3}} \left(\int_{t_{i-1}}^{t_i} \tau(t)^{-1} \mathrm{d}t \right)^{\frac{2p}{2p+3}} \right)^{\frac{2p+3}{3p}} , \tag{5.17}$$

with $\mathcal{K}^i = \left(\int_\Omega (\det \mathbf{H}^i_u(\mathbf{x}))^{\frac{p}{2p+3}} \mathrm{d}\mathbf{x} \right)^{\frac{2p+3}{3}}$.

5.1.5 Global fixed-point mesh adaptation algorithm

Finally, the unsteady adaptation algorithm can be derived from this error analysis. Three main ideas govern this algorithm:

- It is based on splitting the simulation into sub-intervals.
- It is an iterative *fixed point* algorithm.
- It is a global algorithm.

The basic idea consists in splitting the simulation time frame $[0,\, T]$ into n_{adap} adaptation sub-intervals:

$$[0,\, T] \;=\; [0 = t^0,\, t^1] \cup \ldots \cup [t^i,\, t^{i+1}] \cup \ldots \cup [t^{n_{adap}-1},\, t^{n_{adap}}]\,,$$

and to keep the same adapted mesh for each time sub-interval. On each sub-interval, the mesh is adapted to control the solution accuracy from t^i to t^{i+1}. Consequently, the time-dependent simulation is performed with n_{adap} different adapted meshes. This drastically reduces the number of remeshing during the simulation, hence the number of solution transfers. It can been seen as a coarse adapted discretization of the time axis, the spatial mesh being kept constant for each sub-interval when the global space-time mesh is visualized, thus providing a first answer to the adaptation of the whole space-time mesh.

We have seen in Chapter 4 that mesh adaptation is a non-linear problem, and that this is addressed with iterative algorithms to converge the mesh/solution couple. To that end we propose a fixed-point mesh adaptation algorithm. This is also a way to predict the solution evolution within a sub-interval and to adapt the mesh accordingly.

Previously, the optimal metric of a sub-interval could be computed directly once the simulation on the sub-interval had been run [Alauzet 2007]. However, the computation of the optimal continuous mesh for sub-intervals given by Relation (5.16) involves a global normalization term which requires the knowledge of quantities over the whole simulation time frame. In this case, the normalization term is:

$$N_{st}^{\frac{2}{3}} \left(\int_0^T \tau(t)^{-\frac{2p}{2p+3}} \left(\int_\Omega (\det |H_u(\bar{\mathbf{x}}, t)|)^{\frac{p}{2p+3}} \, \mathrm{d}\bar{\mathbf{x}} \right) \mathrm{d}t \right)^{-\frac{2}{3}},$$

which requires to know all the time steps $\tau(t)$ and Hessians $H_u(\mathbf{x}, t)$ over time frame $[0, T]$. Thus, the complete simulation must be performed before evaluating any continuous mesh.

To solve this issue, we suggest to consider a *global* fixed-point mesh adaptation algorithm covering the whole time frame $[0, T]$. All the solutions and Hessian-metrics are computed, and only then can the global normalization term and thus the metrics for each sub-interval be computed. This algorithm is schematized in Algorithm 4 where $\mathcal{H}$, $\mathcal{S}$ and $\mathcal{M}$ denote respectively meshes, solutions and metrics, and $\mathbf{H}$ is the Hessian-metric.

Algorithm 4 Mesh Adaptation Loop for Unsteady Flows

Initial mesh and solution$(\mathcal{H}_0, \mathcal{S}_0^0)$ and set targeted space-time complexity N_{st}

// Fixed-point loop to converge the global space-time mesh adaptation problem
For $j = 1, n_{ptfx}$

1. *// Adaptive loop to advance the solution in time on time frame* $[0, T]$
   For $i = 1, n_{adap}$
   (a) $\mathcal{S}_{0,i}^j$ = Interpolate conservatively next sub-interval initial sol. from $(\mathcal{H}_{i-1}^j, \mathcal{S}_{i-1}^j, \mathcal{H}_i^j)$;
   (b) $\mathcal{S}_i^j$ = Compute solution on sub-interval from pair $(\mathcal{S}_{0,i}^j, \mathcal{H}_i^j)$;
   (c) $|\mathbf{H}|_i^j$ = Compute sub-interval Hessian-metric from sol. sample $(\mathcal{H}_i^j, \{\mathcal{S}_i^j(k)\}_{k=1,nk})$;
   EndFor
2. $\mathcal{C}^j$ = Compute space-time complexity from all Hessian-metrics $(\{|\mathbf{H}|_i^j\}_{i=1,n_{adap}})$;
3. $\{\mathcal{M}_i^j\}_{i=1,n_{adap}}$ = Compute all sub-interval unsteady metrics $(\mathcal{C}^j, \{|H_{\max}|_i^j\}_{i=1,n_{adap}})$;
4. $\{\mathcal{H}_i^{j+1}\}_{i=1,n_{adap}}$ = Generate all sub-interval adapted meshes $(\{\mathcal{H}_i^j,\ \mathcal{M}_i^j\}_{i=1,n_{adap}})$;

EndFor

5.2 From theory to practice

5.2.1 Computation of the mean Hessian-metric

The optimal L^p metric involves an averaged Hessian-metric $\mathbf{H}_u^i$ on sub-interval i, but it still remains to know how to compute it practically, *i.e.*, how it is discretized. The strategy adopted [Alauzet 2007] is to sample the solution on the time sub-interval. More precisely, n_k solutions equally distributed on the sub-interval time frame are saved, including the initial solution at t^{i-1} and the final solution at t^i. Positive Hessian $|H_u(\mathbf{x}, t^k)|$ is evaluated for each sample. If samples are distributed regularly in time, the time elapsed between two samples is $\frac{\Delta t_i}{n_k - 1}$. In practice, 20 samples per sub-intervals are usually used.

The Hessian matrices are computed using a double L^1 projection as described in Section 4.2.4, or a double least-square procedure: first the gradients are computed using a linear least-squares reconstruction procedure, and a similar procedure is applied to recover the Hessians. Once these Hessians are computed, one needs to average them. In the previous algorithm [Alauzet 2007], the following discretization was used:

$$\mathbf{H}_{L^\infty}^i(\mathbf{x}) = \Delta t_i \bigcap_{k=1}^{n_k} |H_u(\mathbf{x}, t^k)| = \Delta t_i \, |H_{\max}^i(\mathbf{x})| \,,$$

where $\cap$ has to be understand as the metric intersection in time of all samples. This corresponds to an integration in time in L^∞ norm of the Hessians.

However, the new error analysis leads to write $\mathbf{H}^i$ as the integral over time of the Hessian

matrices, and thus the following discretization is preferred:

$$\mathbf{H}^i_{L^1}(\mathbf{x}) = \frac{1}{2}\frac{\Delta t_i}{n_k-1}|H_u(\mathbf{x},t_{i-1})| + \frac{\Delta t_i}{n_k-1}\sum_{k=2}^{n_k-1}|H_u(\mathbf{x},t_k)| + \frac{1}{2}\frac{\Delta t_i}{n_k-1}|H_u(\mathbf{x},t_i)| = \Delta t_i\,|H^i_{\text{avg}}(\mathbf{x})|\,,$$

where $\Delta t_i = t^i - t^{i-1}$ is the sub-interval time length and $t^k = t^{i-1} + \frac{k-1}{n_k-1}\Delta t^i$. This corresponds to an integration in time in L^1 norm of the Hessians.

A comparative study of both discretizations was performed and will be detailed in section 5.3.

5.2.2 Choice of the optimal continuous mesh

The optimal adapted mesh for each sub-interval is generated according to analysis of Section 5.1.4. For the numerical results presented below, we select the optimal mesh given by Relation (5.16) and the following particular choices have been made:

- the Hessian-metric for sub-interval i is discretized in time in L^1 norm.
- function $\tau : t \to \tau(t)$ is constant and equal to 1
- all sub-intervals have the same time length Δt.

Moreover, it is clear that integral $\int_{t_{i-1}}^{t_i}\tau(t)^{-1}\mathrm{d}t$ corresponds to the number of time steps (iterations) performed by the flow solver during the i^{th} sub-interval. In practice using the number of iterations of the flow solver for each sub-interval to define the continuous mesh is an issue because it highly depends on the discrete representation of the continuous mesh, *i.e.,* the generated discrete mesh. Thus, the number of iterations of a sub-interval may substantially vary between two fixed-point iterations. To avoid this issue, we prefer considering the flow solver time step constant on each sub-interval, and consider the time step $\tau(t)$ constant equal to Δt (because of the global normalization term). The integral $\int_{t_{i-1}}^{t_i}\tau(t)^{-1}\mathrm{d}t$ then reduces to 1.

With these choices, the optimal continuous mesh $\mathbf{M}_{L^p} = \{\mathbf{M}^i_{L^p}\}_{i=1,..,n_{adap}}$ simplifies to:

$$\mathcal{M}^i_{L^p}(\mathbf{x}) = N_{st}^{\frac{2}{3}}\left(\sum_{j=1}^{n_{adap}}\left(\int_\Omega(\det|\mathbf{H}^j_{L^1}(\mathbf{x})|)^{\frac{p}{2p+3}}\mathrm{d}\mathbf{x}\right)\right)^{-\frac{2}{3}}\left(\det|\mathbf{H}^i_{L^1}(\mathbf{x})|\right)^{-\frac{1}{2p+3}}|\mathbf{H}^i_{L^1}(\mathbf{x})|\,. \quad (5.18)$$

In that case, as we assume that theoretically, one time step is done by sub-interval, and N_{st}/n_{adap} represents the average spatial complexity by sub-interval and thus N_{st} is the total spatial complexity by summing the sub-intervals average complexity. We do not prescribe the temporal complexity, *i.e.,* we do not control the number of time steps done at each sub-interval.

Specification of the space-time complexity

The optimal continuous mesh from Equation (5.16) is obtained for a given space-time complexity that is the number of vertices times the number of solver time steps. Such a complexity should thus be prescribed for each adaptative simulation. However, in practice, it depends on the time discretization of the sub-intervals. Moreover, as explained above, the number of solver time steps is highly dependent on the sizes of the elements of the mesh, which is not perfectly controlled throughout the algorithm. For these reasons, we prefer to prescribe an average number of

vertices per sub-interval, which we sometimes refer to as "simplified space-time complexity" in what follows. The first steps towards a study of the space-time complexity is proposed in [Alauzet 2012].

5.2.3 Matrix-free P1-exact conservative solution transfer

For the unsteady adaptation algorithm, the interpolation step is crucial. Indeed, when moving from a sub-interval to the following, the solution has to be transferred from the mesh of the current sub-interval to the mesh of the following sub-interval, and this is done as many times as there are sub-intervals. It is obvious that if information is lost during the transfer process, then the accuracy of the global solution is greatly affected. As stated in Section 4.2.4, the following properties seem desirable for the interpolation method: (i) mass conservation, (ii) $\mathbb{P}_1$ exactness preserving the second order of the adaptive strategy and (iii) verify the maximum principle. The interpolation scheme used is detailed in Section 4.2.4.

5.2.4 The remeshing step

The remeshing step is also crucial in the adaptation process, as a poorly adapted mesh or a mesh with bad quality elements will impact the accuracy of the solution and spoil the efforts on all other parts of the process. In this work, the remesher `Feflo.a` [Loseille 2013] was used. `Feflo.a` belongs to the class of 3D anisotropic local remeshers. It aims at generating a unit mesh with respect to a prescribed metric field. Its main particularity is to adapt the volume and the surface mesh in a coupled way so that a valid 3D mesh is always guaranteed on output. It uses a unique cavity-based operator (that generalizes standard operators: point insertion, edge removal, edges swapping, point smoothing). This operator is governed by dedicated metric-based quality functions. This allows to reach a good balance between high level of anisotropy, necessary to capture the physical features of the solution, and mesh quality, necessary to ensure the stability and accuracy of the numerical scheme. `Feflo.a` has several enhancements designed for CFD computations, including explicit control and optimization of tetrahedra sizes to ensure a maximal time step for unsteady simulations, and surface mesh re-projection based either on CAD or on background discrete meshes.

5.2.5 Software used

Our implementation of the mesh adaptation algorithm described above requires successively four different software components:

- `Wolf` the second order Finite-Volume flow solver, whose ALE version was described in Chapter 2 - when fixed-mesh simulations are run, a standard version is used, but the numerical schemes are the same as the ALE ones, without the extra ALE velocity terms -,
- `Metrix` to compute the continuous space-time mesh and perform the metric fields gradation,
- `Feflo.a` the local adaptive remesher based on the cavity operator, described in Section 5.2.4,
- `Interpol` for the $\mathbb{P}^1$-conservative solution transfer, described in Section 5.2.3.

5.3 Choice of the mean Hessian-metric

All the unsteady adaptation algorithm lies on the computation of a mean metric for each sub-intervals. Instead of averaging real metrics, we average the Hessians on the go during the solver iterations.

There is a choice to make concerning the way this average is done, between the historical intersection or the sum dictated by theory. The intersection of metrics reads:

$$\mathbf{H}^i_{L^\infty}(\mathbf{x}) = \Delta t_i \bigcap_{k=1}^{n_k} |H_u(\mathbf{x}, t^k)| = \Delta t_i \, |H^i_{\max}(\mathbf{x})| \,,$$

It comes to making an average in L^∞ norm, and has the advantage of having a clear geometric interpretation: the intersection of two metrics is the largest metric included in the two metrics. In the context of unsteady adaptation, this means that the mean Hessian-metric has the smallest sizes met over the sub-interval. The sum of metrics comes to making an average in L^1 norm, and reads:

$$\mathbf{H}^i_{L^1}(\mathbf{x}) = \frac{1}{2}\frac{\Delta t_i}{n_k - 1}|H_u(\mathbf{x}, t_{i-1})| + \frac{\Delta t_i}{n_k - 1}\sum_{k=2}^{n_k-1}|H_u(\mathbf{x}, t_k)| + \frac{1}{2}\frac{\Delta t_i}{n_k - 1}|H_u(\mathbf{x}, t_i)| = \Delta t_i \, |H^i_{\text{avg}}(\mathbf{x})| \,,$$

It seems to be more coherent with theory. Indeed, when writing the minimization problem (5.14) with sub-intervals, we can formally derive the error from the unsteady error without sub-intervals (5.5) by moving the integral over time into the trace. However, it is only a formal manipulation, and the geometrical interpretation of the sum is much less clear: a term-to-term sum of metrics is clearly still a metric (a symmetric positive definite form), but the sizes of the axes of the resulting metric are not the sum of the sizes of the two input metrics. This justifies numerical experimentation to select the best approach in practice.

The computation of the Hessians is performed as detailed in Section 4.2.4. The absolute value of a hessian is a metric, however, numerical Hessians need to be corrected. The absolute value of the eigenvalues is taken, and too small eigenvalues are brought to a value ensuring that a maximal anisotropic ratio is not exceeded. These modified Hessians are called *Hessian-metrics*.

Even so, degenerated cases can appear, *i.e.* cases where the linear form associated to the matrix is not positive and definite, for instance if one eigenvalue is null. This is not a problem as far as sums of Hessian-metrics are performed, but in the case of metric intersection, the operator defined in 4.2.3 is not defined. Numerically, even almost null eigenvalues (if both eigenvalues are very small for instance) are enough to put out of action the standard definition. So a procedure was devised to handle all degenerate cases. This procedure is based on appropriate changes of bases. It is too long to enumerate all the different cases here, but the procedure is detailed in Appendix A.

Both intersection and sum of metrics were tested on numerous test cases. We present here the most significant. The study was carried out in 2D for time efficiency purposes, and because the observation of the meshes is easier than in 3D. In all the following examples, the adaptation is performed in L^2 norm (*i.e.* $p = 2$).

Advection

The two metric averages were first compared on pure advection cases, where it is easy to control the solution. To that end, a specific module was added to the flow solver to solve the linear advection equation:

$$\frac{\partial \mathbf{W}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{W} = 0\,, \tag{5.19}$$

where $\mathbf{v}$ is the velocity vector. A scalar quantity, defined by a function at time $t = 0$, is advected, which comes to considering $\mathbf{W} = \begin{pmatrix} \rho \\ \mathbf{0} \\ 0 \end{pmatrix}$. The numerical method is the same as described previously, but a specific upwind flux is used. Let v_{mean} be an average velocity on edge $\mathbf{e}_{ij}$: $v_{mean} = \frac{1}{2}(\mathbf{v}_i + \mathbf{v}_j) \cdot \mathbf{n}_{ij}$, where $\mathbf{n}_{ij}$ is the normal to the edge. Then the flux on this edge is:

$$\mathbf{\Phi}_{ij}(\mathbf{W}_i, \mathbf{W}_j, \mathbf{n}_{ij}) = \begin{cases} \mathbf{v}_{mean}\mathbf{W}_i & \text{if} \quad \mathbf{v}_{mean} > 0 \\ \mathbf{v}_{mean}\mathbf{W}_j & \text{if} \quad \mathbf{v}_{mean} \leq 0\,, \end{cases} \tag{5.20}$$

where the edges are defined so that vertex P_i is on the left and vertex P_j is on the right ($\mathbf{ij} \cdot \mathbf{n}_{ij} > 0$).

Several functions (shapes) and velocities were considered. The most meaningful cases were: accelerated translation of a square, rotation of a square and of Zalesak's disk and translation of sine curves. Each of these cases presents a specific feature of interest. The square has edges that are both parallel and orthogonal to the displacement, the first ones being a simplified model of shocks. Zalesak's disk allows to look at the rotation of both straight lines and curves. The sine curves are phenomena that vary continuously and potentially quickly.

The first case is the advection of an accelerated square in a tube. The tube dimensions are: $x \in [0, 10]$ and $y \in [0, 1]$, the scalar function is initially set to 0 everywhere except on a square, defined by $x \in [0.2, 0.8]$ and $y \in [0.2, 0.8]$, where the function is set to 1, and the velocity vector is $\mathbf{v} = (1 + 0.3t, 0)$. Regarding the adaptation, the number of fixed-point iterations is set to $n_{adap} = 5$, the number of sub-interval is varying. A complexity is set so that the meshes have an average size of $11,000$ vertices. This simple case shows that the error is smaller for the intersection of metrics for all the number of sub-intervals. Incidentally, one can notice that the more sub-intervals there are, the more accurate the solution is, but the bigger the total space-time complexity is. The aim of unsteady adaptation is to find a good trade off between both.

The observation of the meshes for that case (Fig. 5.1) explains why the L^∞ average is better: the resulting metric leads to putting much more vertices on the "shocks", which are the areas of interest here. The L^1 average is very sensitive to the orthogonal phenomenon, and leads to meshing more the edges orthogonal to the shocks. This can be simply explained: the Hessians are large in the directions orthogonal to the shocks, so their sum is large too, whereas their intersection is very small since they are all parallel and shifted. In this case, capturing the shocks is eventually more important than capturing the orthogonal phenomena.

Then two cases of rotating objects are considered. The domain is a square of dimensions $x \in [0, 1]$ and $y \in [0, 1]$. The scalar function is initially set to one on a square of dimensions

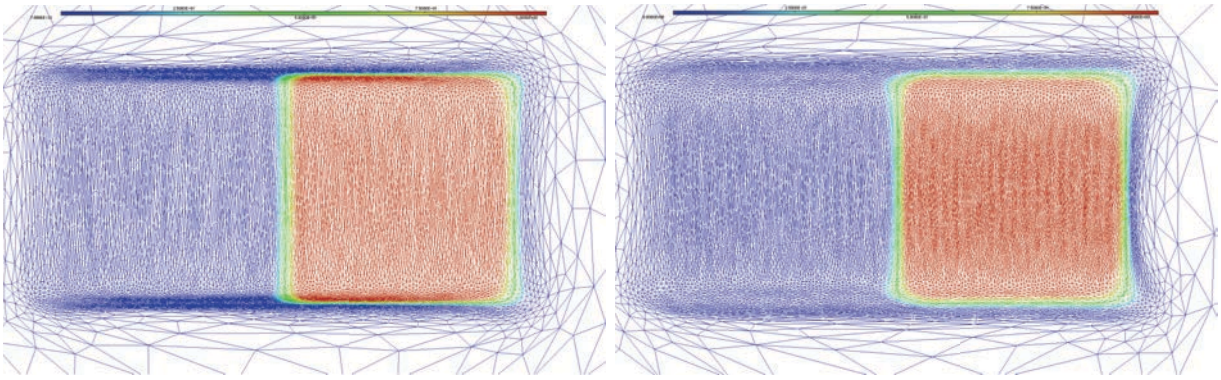

Figure 5.1: Adapted meshes on the last sub-interval of an adaption loop for the accelerated square. Left, using the sum of metrics and right, using the metric intersection.

$x \in [0.65, 0.85]$ and $y \in [0.4, 0.6]$ for the first case, and Zalesak's disk of radius $0,2$ centered in $(0.75, 0.5)$ for the second. The velocity vector is defined for any point of the domain by: $\mathbf{v} = (-(y - 0.5), x - 0.5)$. Regarding the adaptation, the number of fixed-point iterations is set to $n_{adap} = 6$, the number of sub-interval varying. A complexity is set so that the meshes have an average size of $12,500$ vertices. These cases are harder to analyze. It overall appears that the sum of Hessians gives better results (Fig. 5.2). Considering the previous analysis, and in the absence of shocks, it appears that the sum has lead to capturing better the rotation of the objects. We can see on 5.2 that the method with intersection of metrics is less good in capturing the angles of the crack.

Finally, we consider the advection of sine curves, parallel or orthogonal to the displacement. The definition of these cases is similar to the definition of the case of the accelerated square. The only difference is that the scalar function is not set to 1 on the square but to $\cos(\pi/0.6(x-0.2))$ for the parallel sine and $\cos(\pi/0.6(y-0.2))$ for the orthogonal one. Regarding the adaptation, the number of fixed-point iterations is set to $n_{adap} = 6$, the number of sub-interval varying. A complexity is set so that the meshes have an average size of $15,000$ vertices. These cases seem to confirm our previous analysis (see Fig. 5.3). In the case of the the sine parallel to the displacement, the intersection of Hessians leads to meshing the inside of the curve, while the sides are neglected. When the sine is orthogonal to the displacement, both methods are equivalent for capturing the oscillations, even if the intersection of metrics meshes more the top and bottom sides of the sine curves.

Sod shock tube

A classic Sod shock tube was also studied. This case is helpful since, not only do we known an almost analytic solution to the problem, but the solution presents two discontinuities (the shock and the contact discontinuity) and a smoother phenomenon (the rarefaction wave) to study the behavior of the adaptation algorithm.

The problem is the following: on a rectangular domain, with $x \in [0, 1]$ and $y \in [0, 0.2]$ a gas

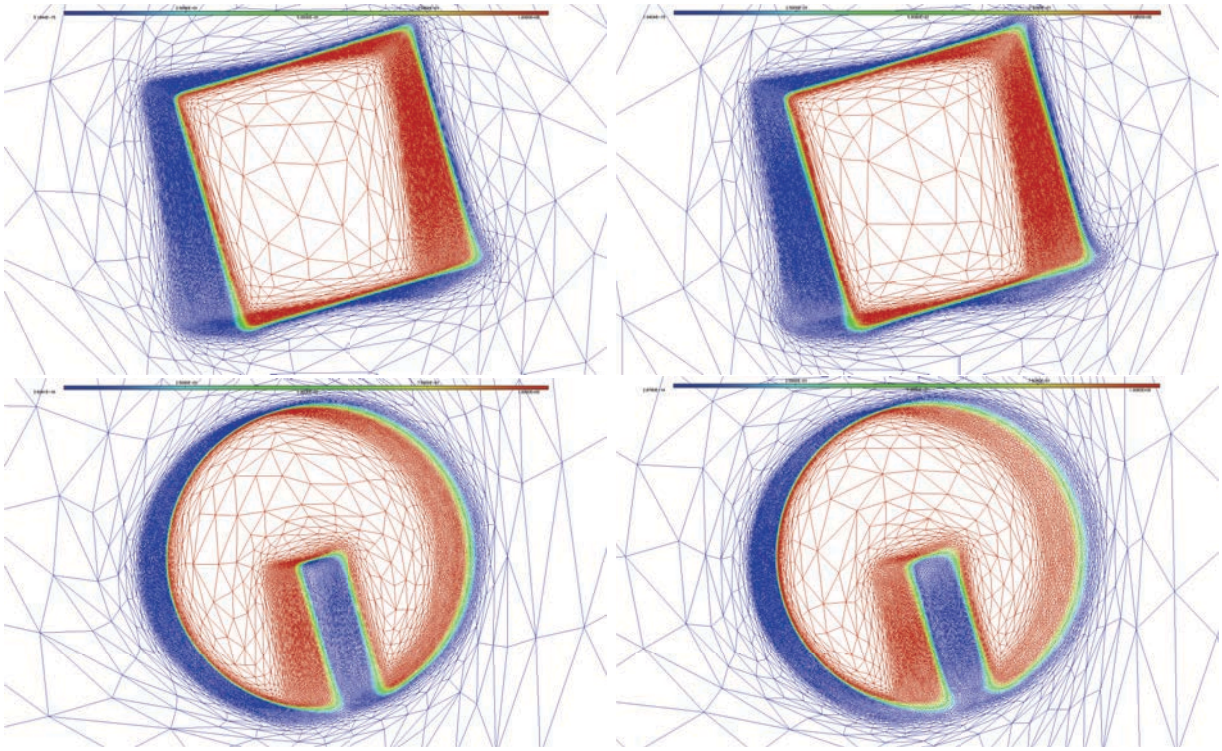

Figure 5.2: Two cases of advection with rotation: a square and Zalesak's disk. Adapted meshes on the last sub-interval of an adaption loop. Left, using the sum of metrics and right, using the metric intersection.

of heat capacity ratio $\gamma = 1.4$ is split into two states, a left state and a right state:

$$W_{left}(x < 0.5, t = 0) = \begin{pmatrix} \rho_{left} \\ \mathbf{u}_{left} \\ p_{left} \end{pmatrix} = \begin{pmatrix} 1.0 \\ \mathbf{0} \\ 1.0 \end{pmatrix} \quad \text{and} \quad W_{right}(x \geq 0.5, t = 0) = \begin{pmatrix} \rho_{right} \\ \mathbf{v}_{right} \\ p_{right} \end{pmatrix} = \begin{pmatrix} 0.125 \\ \mathbf{0} \\ 0.1 \end{pmatrix}.$$

The simulation is stopped at $t = 0.25$, before the waves reach the end of the tube. On the top and bottom sides of the tube, slipping boundary conditions are imposed, and Dirichlet conditions are imposed right and left of the tube.

The analytic solution of this problem is the classic solution of a one-dimension Riemann problem. There are three simple waves that delimit four areas. These waves are a shock, a contact discontinuity and an isentropic rarefaction. The analytic solution is described in details in the literature, for instance in [Liepmann 2002]. The main difficulty in the computation of the solution is that only an implicit form of the pressure behind the shock can be found.

This example also illustrates the importance of the choice of the sensor on which the adaptation is performed, since not all fields of the solution display all the physical phenomena. Indeed, the pressure does not see the contact discontinuity: if the pressure was chosen as adaptation sensor, the area around this discontinuity would not be refined, and the physical phenomenon would be lost. The density allows us to see all the phenomena so it is chosen as sensor.

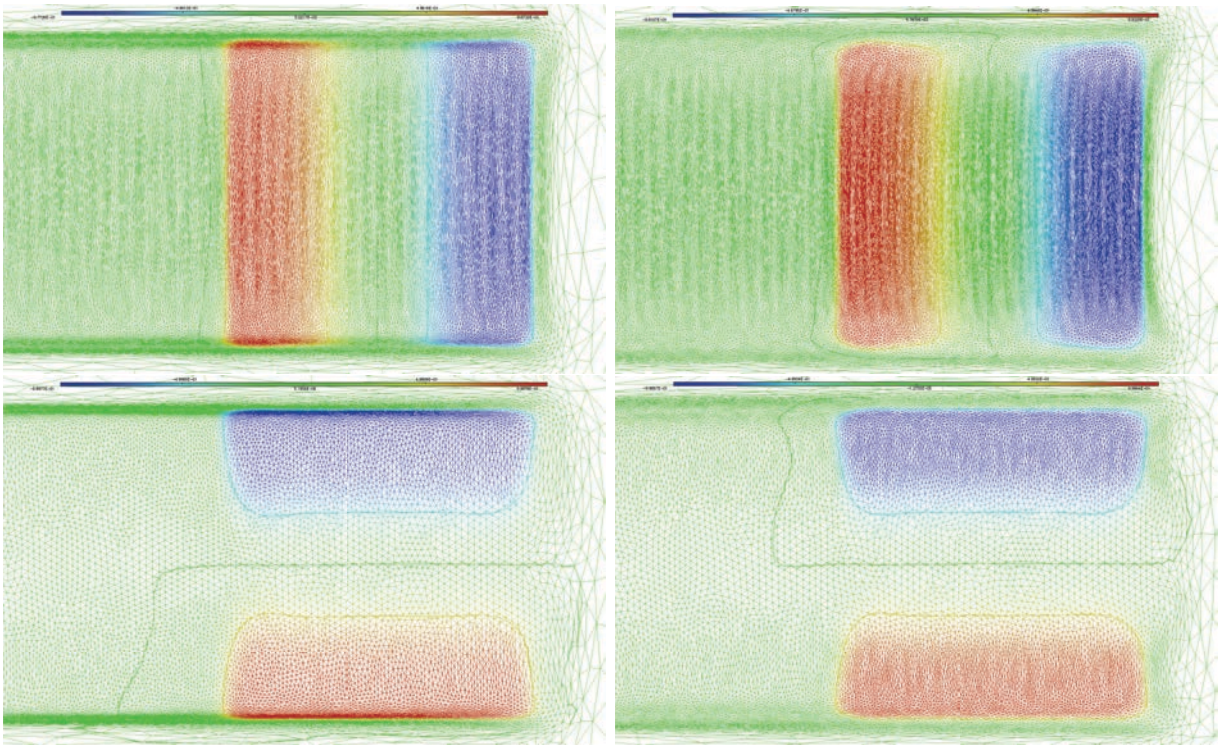

Figure 5.3: Two advection cases of sine curves, parallel (top) and orthogonal (bottom) to the velocity vector . Adapted meshes on the last sub-interval of an adaption loop. Left, using the sum of metrics and right, using the metric intersection.

The adaptation loop was used on this test case, with $n_{adap} = 15$ sub-intervals and 4 fixed-point iterations. A space-time complexity was prescribed so that the meshes have between $4,200$ and $4,900$ vertices, for an average mesh size of $4,500$ vertices. Snapshots of adapted meshes and of the solution are presented in Fig. 5.5.

Since an analytic solution is known for this problem, we can compute an error with respect to this analytic solution. To do so, we extract the solution along a line of the mesh, in this case the line $y = 0.1$. The line is discretized with 800 vertices regularly distributed on it. A linear interpolation is performed from the solution on the adapted mesh to this line, and the analytic solution is computed for each of the 800 vertices too. The error is then computed in L^1 norm. We find an error $\mathcal{E} = 0.12$ for the simulation with the sum of metrics, and $\mathcal{E} = 0.2$ for the simulation with the intersection of metrics. A close look at the solutions along this extraction line in Fig 5.6 confirms this result: the three curves, corresponding to the solution with sum of metrics, the solution with intersection of metrics, and the analytic solution are very close and almost indistinguishable. However, a closer look at the curves, when the solution gradients vary a lot, shows the difference between the two methods: at the beginning and the end of the rarefaction, the solution with sum of metrics is closer to the reference solution than the solution with metric intersection, and on a large part of the rarefaction the sum of metrics is better. Close to the shock, the solution with metric intersection is slightly better, but the difference is smaller than the difference in the rarefaction. These differences can be explained if we look at

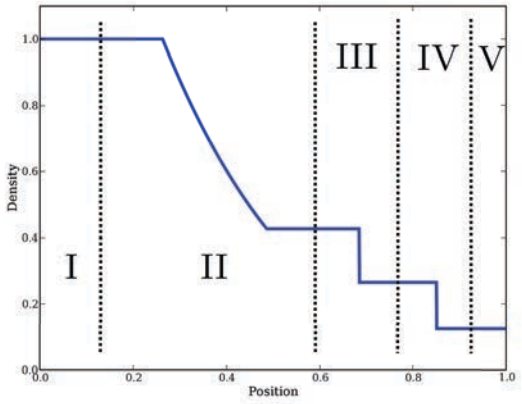

Figure 5.4: Sod shock tube solution. Region IV, the shock is moving to the right, region III the contact discontinuity moves to the right and region II the rarefaction moves to the left.[Image from Wikipedia]

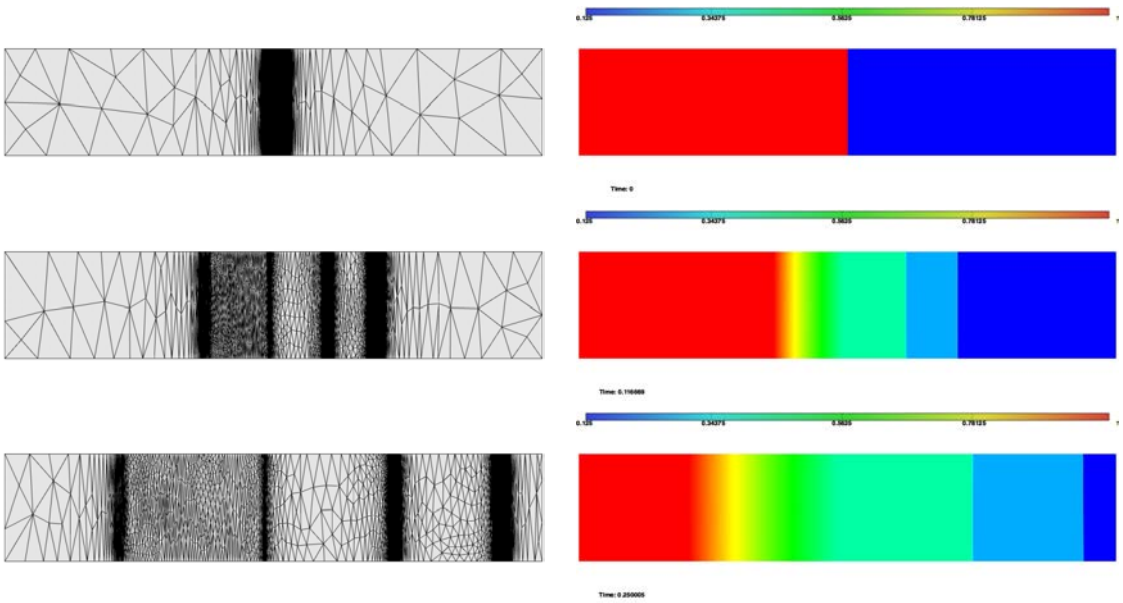

Figure 5.5: Sod shock tube: snapshots of adapted meshes and solution at times $t = 0$, $t = 0.117$ and $t = 0.25$.

the meshes in Fig. 5.7. The method with L^1 average tends to mesh the rarefaction with more vertices, whereas the method with L^∞ norm tends to gather most of the points in the band corresponding to the shock, which is very obvious if we zoom in this band.

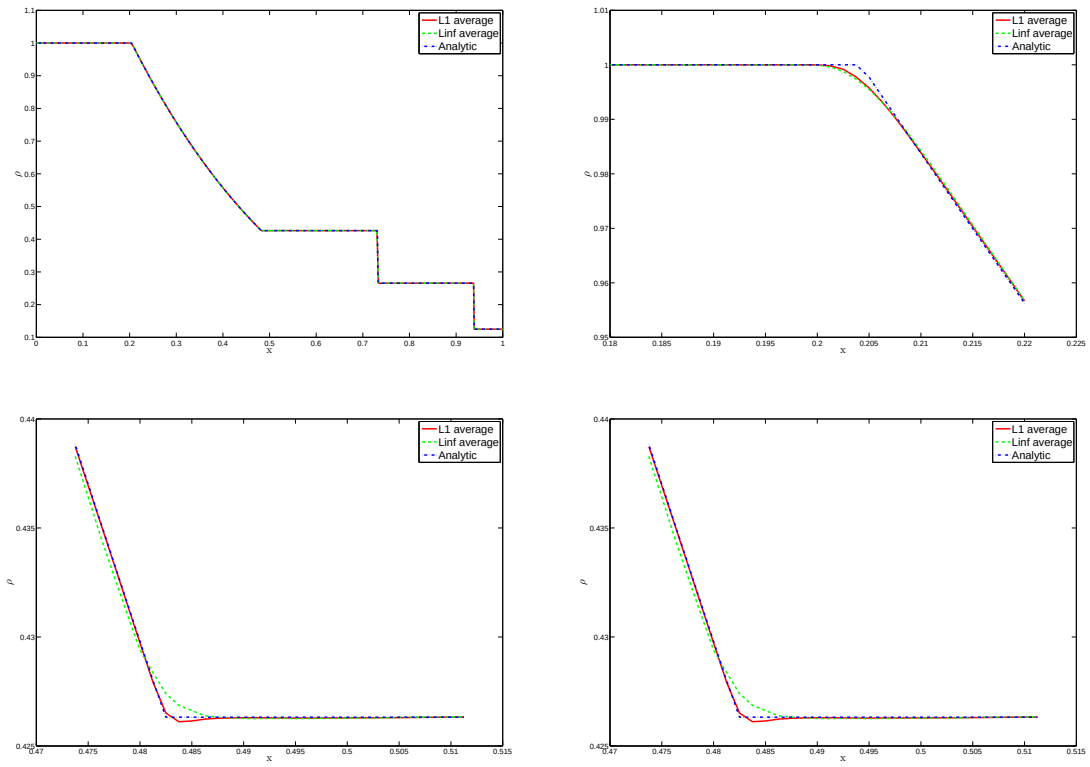

Figure 5.6: Sod shock tube: plots of 1D extraction of the solution for both L^1 average (sum) , L^∞ average (intersection) and analytic solution. Full solution (top left) and zooms on the discontinuities.

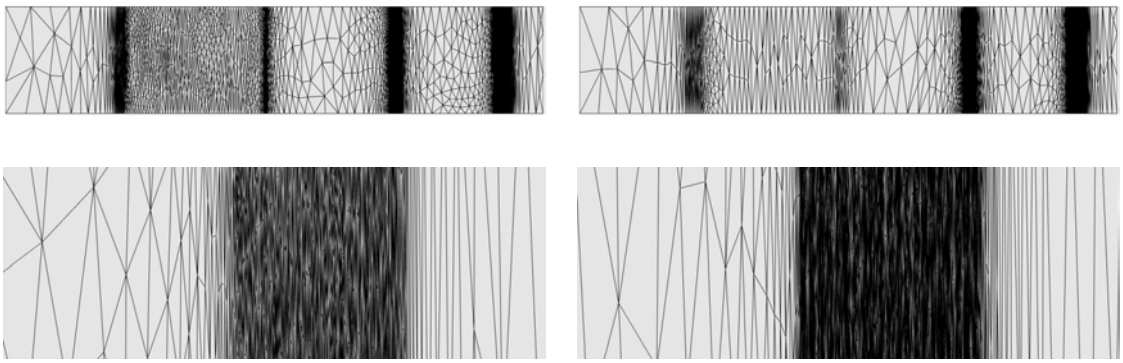

Figure 5.7: Sod shock tube: adapted meshes of the final sub-interval for the L^1 average (left) and the L^∞ average (right). Bottom, zoom on the band corresponding to the shock for these meshes.

Shock-bubble interaction

Finally, a more complex CFD case was studied: an interaction between a shock (pressure discontinuity) and a bubble (area of lower density), that is a 2D version of a problem proposed

in [Langseth 2000].

The domain is a 2D cylinder, in which three regions are defined that create a shock and a bubble, see Fig. 5.8. The initial conditions are:

$$\begin{pmatrix} \rho_{left} \\ \mathbf{u}_{left} \\ p_{left} \end{pmatrix} = \begin{pmatrix} 1 \\ \mathbf{0} \\ 10 \end{pmatrix} \quad , \quad \begin{pmatrix} \rho_{bubble} \\ \mathbf{u}_{bubble} \\ p_{bubble} \end{pmatrix} = \begin{pmatrix} 0.1 \\ \mathbf{0} \\ 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \rho_{right} \\ \mathbf{u}_{right} \\ p_{right} \end{pmatrix} = \begin{pmatrix} 1 \\ \mathbf{0} \\ 1 \end{pmatrix} ,$$

The simulation is run until time $t = 0.45$. Complex pattern are created behind the shock as it moves forward in and past the bubble. The aim of multiscale mesh adaption is to capture both the moving shock and these small features of the flow as precisely as possible.

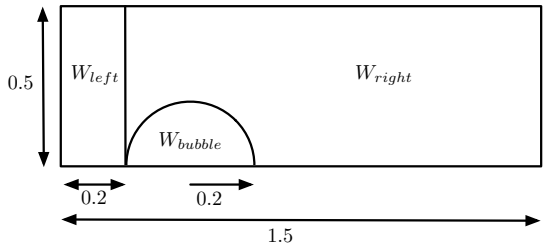

Figure 5.8: Shock bubble interaction test case.

The adaptation loop was used on this test case, with $n_{adap} = 25$ sub-intervals and 6 fixed-point iterations. A space-time complexity was prescribed so that the meshes have between $9,000$ and $18,000$ vertices, for an average mesh size of $14,600$ vertices. Snapshots of adapted meshes and of the solution are presented in Fig. 5.9.

Comparing the two methods for averaging the Hessians is more difficult than in the previous cases. Indeed, no analytic solution is known, and the phenomena behind the shock are instabilities and depend on the mesh on which they are computed, which means we cannot compute a reference solution on a very fine mesh, and computed errors with respect to that reference solution. Therefore, we are limited to a qualitative analysis of the results. The meshes and solutions at final time $t = 0.45$ are represented in Fig. 5.10 for both methods. The difference between both is easy to observe: the instabilities are much better refined - and thus accurately captured - with the sum of metrics than with the intersection. The intersection of metrics tends to refine more the shock, and less the smaller amplitude phenomena. This had already been observed for the advection cases, but in this case it is not desirable, since it leads to losing a part of the solution.

Conclusion

In conclusion of this study, it appears that both methods result in similar meshes in most cases. The purely advective cases have shown both have advantages and drawbacks. However for CFD simulations, the L^∞ average tends to result in meshes where the vertices are concentrated on the higher scale phenomena (typically shocks), whereas the L^1 average tends tor result in meshes where the smaller scale features are better captured. This is why we use preferably the L^1 average, or "sum of metrics".

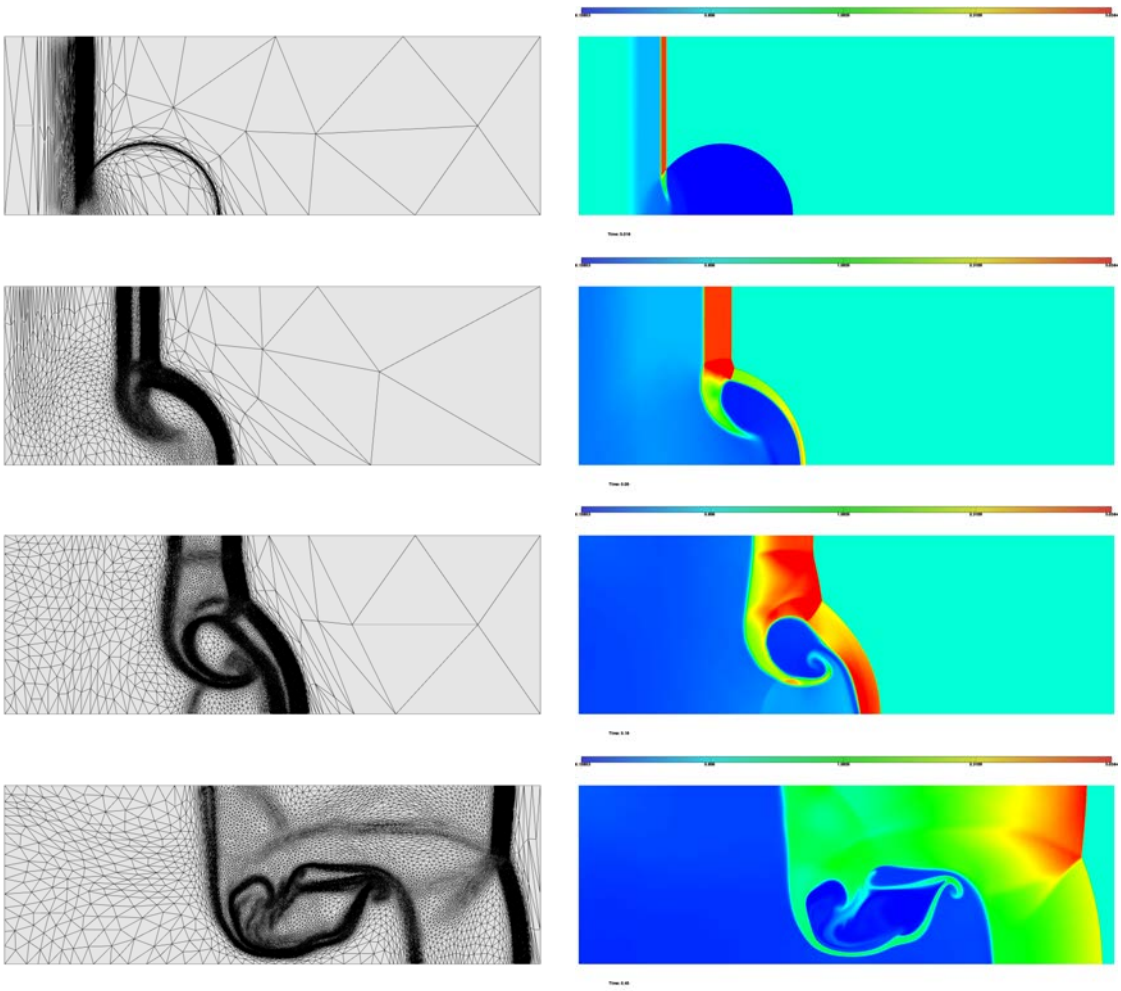

Figure 5.9: Leveque's shock-bubble interaction problem: snapshots of adapted meshes (left) and solution (right) at times $t = 0.018$, $t = 0.09$, $t = 0.18$, $t = 0.45$ from top to bottom.

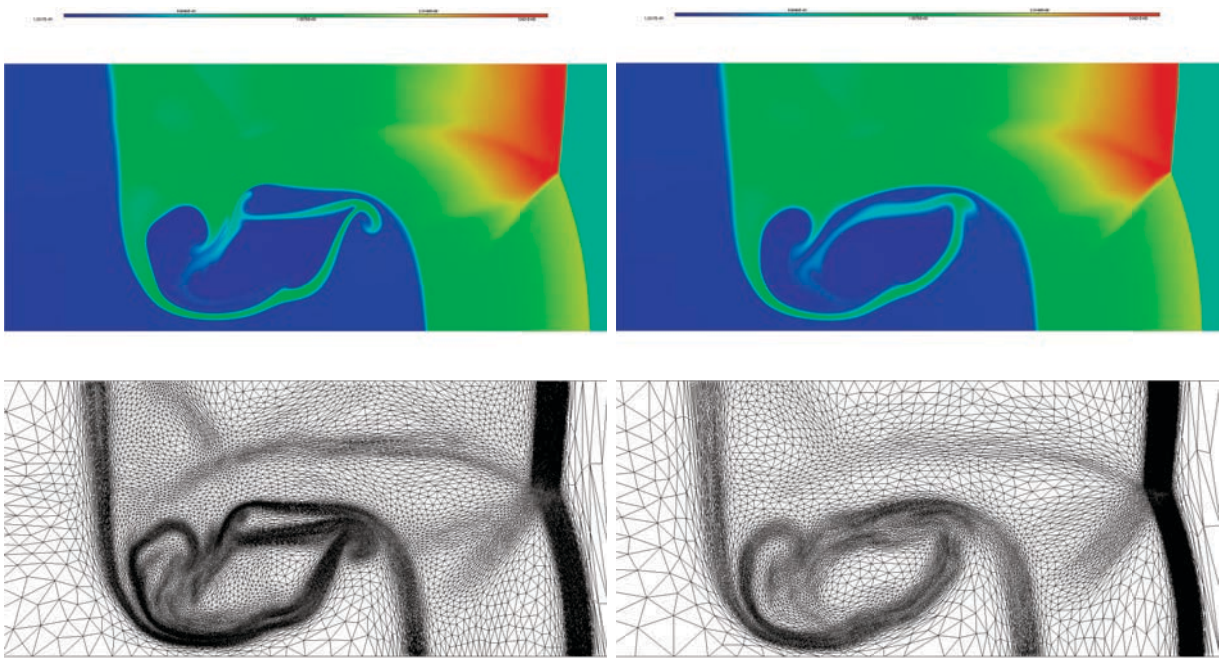

Figure 5.10: Leveque's shock-bubble interaction problem: solution (top) and mesh (bottom) at final time for the L^1 average (left) and the L^∞ average (right).

5.4 Numerical examples

The previous unsteady adaptation algorithm is applied on numerous cases, to demonstrate its ability to improve significantly the accuracy of the solution.

5.4.1 2D shock-bubble interaction

First, another version of a shock-bubble interaction case was studied, with an imposed speed for the shock. The setup is showed in Fig. 5.11. A circle of radius 0.2 is centered at $(0.3, 0.0)$ in the domain $[-0.1, 1.5] \times [-0.5, 0.5]$. The gas is initially at rest and has unit density and pressure. Inside the circle the density is $\rho_{bubble} = 0.1$. The incoming shock wave starts at $x = 0$ and propagates towards the bubble. The pressure behind the shock is $p_{left} = 10$. A Mach number of $M_s = 2.95$ is prescribed for the shock, which requires a small computation to derive the velocity u_{left} behind the shock. Rankine-Hugoniot relations read:

$$\begin{aligned}
&c_{right} M_s(\rho_{right} - \rho_{left}) = \rho_{right} u_{right} - \rho_{left} u_{left}\,,\\
&c_{right} M_s(\rho_{right} u_{right} - \rho_{left} u_{left}) = (\rho_{right} u_{right}^2 + p_{right} - \rho_{left} u_{left}^2 - p_{left})\,,\\
&c_{right} M_s(\rho_{right} E_{right} - \rho_{left} E_{left}) = ((\rho_{right} E_{right} + p_{right}) u_{right} - (\rho_{left} E_{left} + p_{left}) u_{left})\,,
\end{aligned} \tag{5.21}$$

where c_{right} is the sound celerity in the state in front of the shock, and the invariants:

$$\frac{\rho_{left}}{\rho_{right}} = \left(1 - \frac{2}{\gamma+1}(1 - M_s^2)\right)^{-1}, \tag{5.22}$$

$$\frac{p_{left}}{p_{right}} = 1 + \frac{2\gamma}{\gamma+1}(M_s^2 - 1)\,. \tag{5.23}$$

Finally,

$$\frac{u_{left} - u_{right}}{c_{right}} = \frac{\frac{p_{left}}{p_{right}} - 1}{\sqrt{1 + \frac{\gamma+1}{\gamma-1}\frac{p_{left}}{p_{right}}}} \frac{\sqrt{2}}{\sqrt{\gamma(\gamma-1)}}\,. \tag{5.24}$$

With the data of the case, and setting $\gamma = 1.4$, Relations (5.22) and (5.23) directly lead to:

$$\rho_{left} \approx 1 \quad \text{and} \quad p_{left} \approx 10.$$

From which we can deduce the sound celerities ($c = \sqrt{\frac{\gamma p}{\rho}}$):

$$c_{left} \approx 1.18 \quad \text{and} \quad c_{right} \approx 3.74\,,$$

and Relation (5.24) allows us to conclude:

$$u_{left} \approx 2.14\,.$$

Finally the three states are:

$$\begin{pmatrix} \rho_{left} \\ u_{x,left} \\ u_{y,left} \\ p_{left} \end{pmatrix} = \begin{pmatrix} 1 \\ 2.14 \\ 0 \\ 10 \end{pmatrix}, \quad \begin{pmatrix} \rho_{bubble} \\ u_{x,bubble} \\ u_{y,bubble} \\ p_{bubble} \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \rho_{right} \\ u_{x,right} \\ u_{y,right} \\ p_{right} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

The simulation is run until time $t = 0.4$. Again, complex vorticity and mixing pattern are created behind the shock after it hits the bubble. Multiscale mesh adaption aims at capturing both the moving shock and these small features of the flow as precisely as possible.

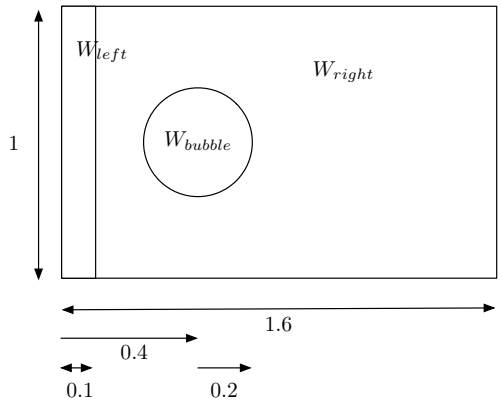

Figure 5.11: Shock bubble interaction test case.

The adaptation is performed on the density, in L^2 norm, with 60 sub-intervals and 5 fixed-point iterations. As already mentioned, it is very difficult to predict the space-time complexity of a simulation since it involves the number of vertices and the number of solver time steps. So we consider a simplified complexity, that is the sum of the numbers of vertices of the meshes of each sub-intervals, which does not depend on the time discretization of the sub-intervals. The desired accuracy was set to reach a simplified space-time complexity of $16,000,000$ vertices, *i.e.* a mean complexity of $275,000$ per mesh, with a maximal mesh size of $411,385$ vertices for the final sub-interval.

Snapshots of the meshes and solutions are shown in Fig 5.12. On the solution, one can see the level of accuracy of the instabilities behind the shock, as well as the multiple shock reflections and interactions. The accuracy of the solution is such that the fractal structure of the instabilities appears clearly. This accuracy is obtained thanks to the adaptation of the meshes, which are refined in the shock areas but also in the vicinity of the smaller scale features. On this example, one can clearly see anisotropic band-shaped areas, which are typical of our algorithm with sub-intervals: they correspond to the zones of evolution of the physical phenomenon during one sub-interval. The more sub-intervals we use, the thinner these bands are, and for a constant number of vertices per sub-interval, the more accurate the solution is. This is illustrated in Fig 5.13, where the final meshes and solutions for two adaptation loops with 10 and 60 sub-intervals are shown. The same average number of vertices per sub-interval was prescribed: $200,000$ vertices. It appears clearly that the bands are much thicker with 10 sub-intervals, while the accuracy of the solution is less good, notably on the shock, and in the areas where the flow is winding behind the shock, where less details are captured. This is due to the fact that a larger area has to be meshed with the same number of elements, resulting in larger elements, and thus a less precise solution.

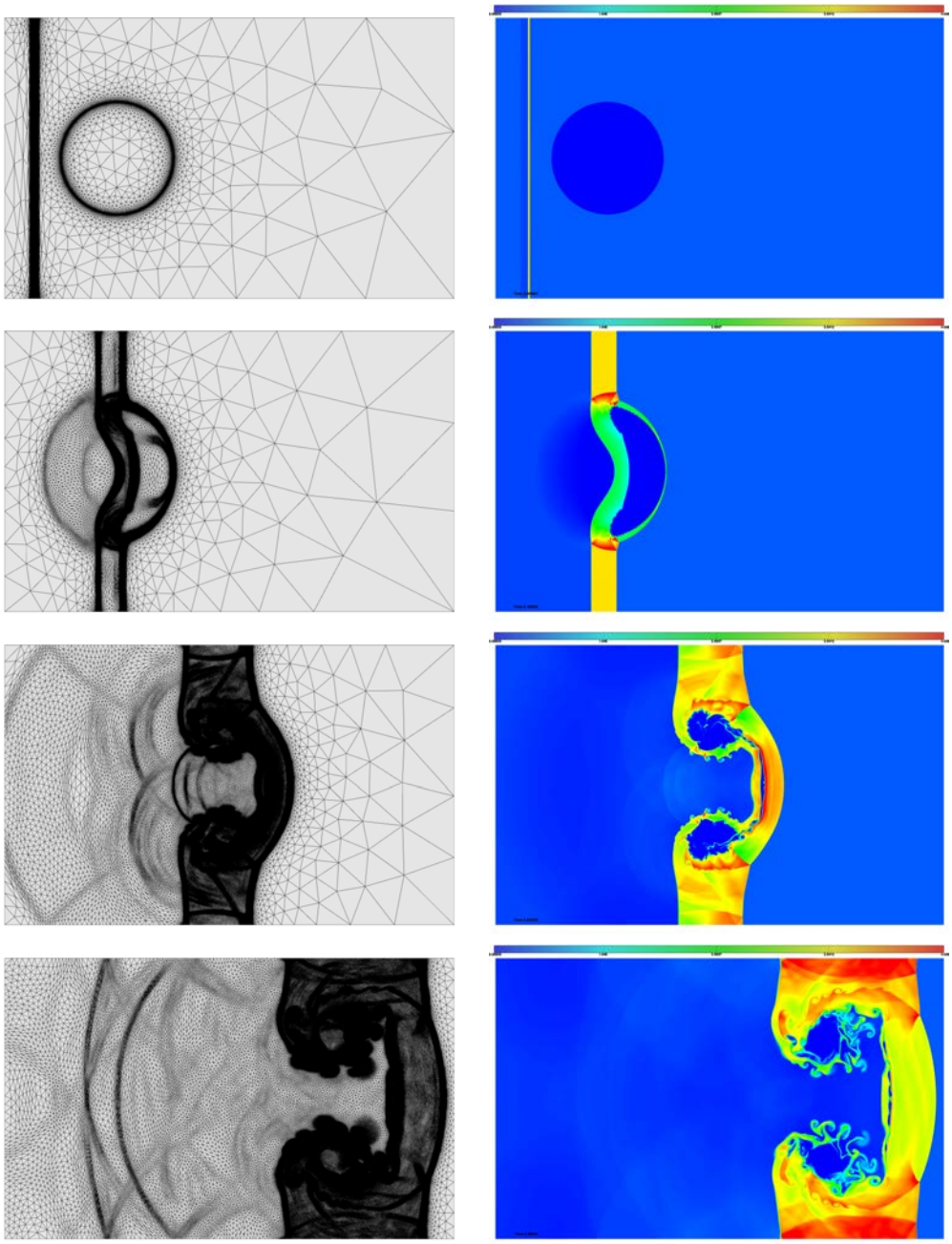

Figure 5.12: 2D shock-bubble case: snapshots of adapted meshes (left) and solution (right) at dimensionless times $t = 0.007$, $t = 0.1$, $t = 0.23$, $t = 0.4$ from top to bottom.

5.4.2 3D circular blast

We then consider a purely three-dimensional blast problem proposed in [Langseth 2000]. In this simulation, shock waves are reflected on a box and interact with each other.

The gas is initially at rest in a box of dimension $[-1.5, 1.5] \times [-1.5, 1.5] \times [0, 1]$. The density

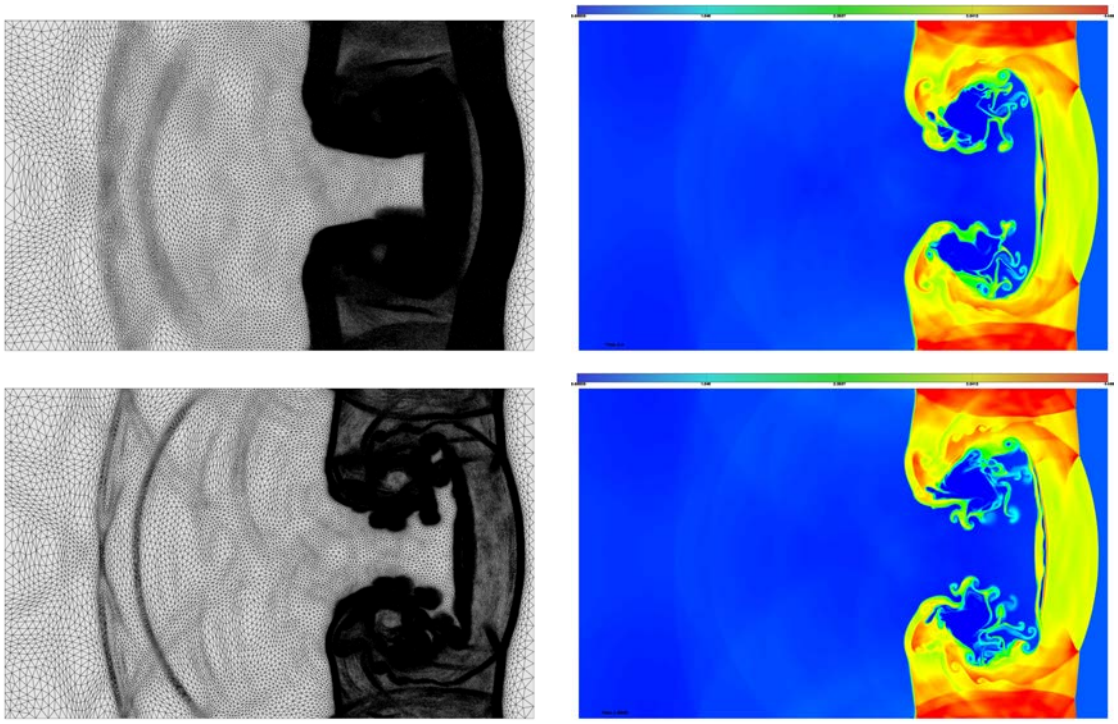

Figure 5.13: Comparison of final meshes and solutions for an adaptation loop with 10 (top) and 60 (bottom) sub-intervals..

and pressure equal one everywhere, except for a sphere of radius 0.2 centered in $(0, 0, 0.4)$, in which the pressure is 5. The gas is then left to evolve freely until time $t = 0.7$. Spherical shock waves emanate from the central region due to the overpressure. Wall conditions are imposed on the boundaries, so that the shock waves are reflected on them, and create complex pattern when they interact with each other.

The density of the flow is chosen as sensor variable for our mesh adaptation process. The space-time interpolation error on the solution is controlled in L^2 norm. The time frame was split into 40 adaptation sub-intervals and 5 fixed-point iterations were used to converge the non-linear mesh adaptation problem. The initial mesh is uniform and has $800,000$ vertices and $4,700,000$ tetrahedra. The desired accuracy was set to reach a simplified space-time complexity of $40,000,000$ vertices, *i.e.* a mean complexity of $1,000,000$ per mesh.

Figures 5.14 shows the solutions and the adapted meshes at different dimensionless times. The 3D mesh adaptation for the whole sub-interval is clearly illustrated. Indeed, the mesh refinement along band-shaped regions, is clearly visible, which correspond to the evolution zone of the physical phenomena during an adaptation sub-interval. The size of the depicted meshes varies between $600,000$ and 1.3 millions of vertices.

Thanks to multiscale mesh adaptation, the complex patterns of reflected weaker shocks interacting with each other are well captured even if strong shocks are moving in the flow field. It results in an accurate solution with very few vertices compared with a uniform mesh with the smallest element size. In the moving shock region, the mesh accuracy is around

$1.2\,e^{-4}$ at dimensionless time 0.07 and $9.8\,e^{-5}$ at dimensionless time 0.7 for a domain size of $[-1.5, 1.5] \times [-1.5, 1.5] \times [0, 1]$.

The CPU time to perform the whole adaptive computation (*i.e.,* the 5 loops with the 40 iterations of flow solver and solution transfer, the computation of the metric, the generation of the adapted mesh) is about 8 hours on a 20-core Intel Xeon processor at 2.5Ghz.

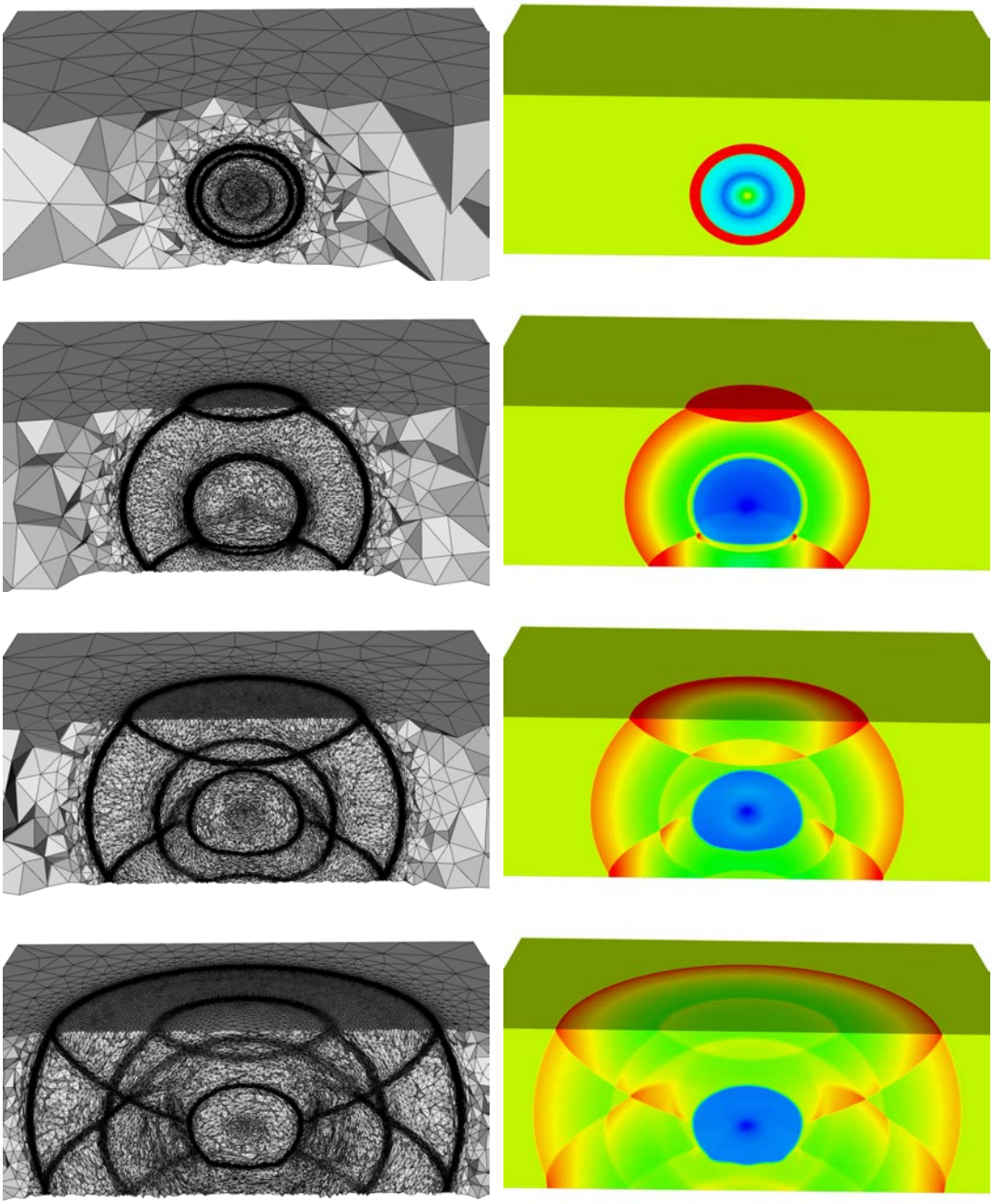

Figure 5.14: Adapted meshes and density solutions for the 3D blast test case, at dimensionless times 0.07, 0.315, 0.455 and 0.7. Cuts into the volume mesh are made along the plane $x = 0$.

5.4.3 3D shock-bubble interaction

We consider a 3D version of the case proposed in Section 5.3. The 3D setup is presented in Fig. 5.15: a shock wave emanates from a corner of a box and hits a bubble, a half-cylinder orthogonal to the shock, creating complex vorticity and mixing patterns. The three states are prescribed explicitly:

$$\begin{pmatrix} \rho_{left} \\ \mathbf{u}_{left} \\ p_{left} \end{pmatrix} = \begin{pmatrix} 1 \\ \mathbf{0} \\ 10 \end{pmatrix} \quad , \quad \begin{pmatrix} \rho_{bubble} \\ \mathbf{u}_{bubble} \\ p_{bubble} \end{pmatrix} = \begin{pmatrix} 0.1 \\ \mathbf{0} \\ 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \rho_{right} \\ \mathbf{u}_{right} \\ p_{right} \end{pmatrix} = \begin{pmatrix} 1 \\ \mathbf{0} \\ 1 \end{pmatrix} .$$

Slipping conditions are imposed on the boundaries, and the simulation is run until time $t = 0.5$

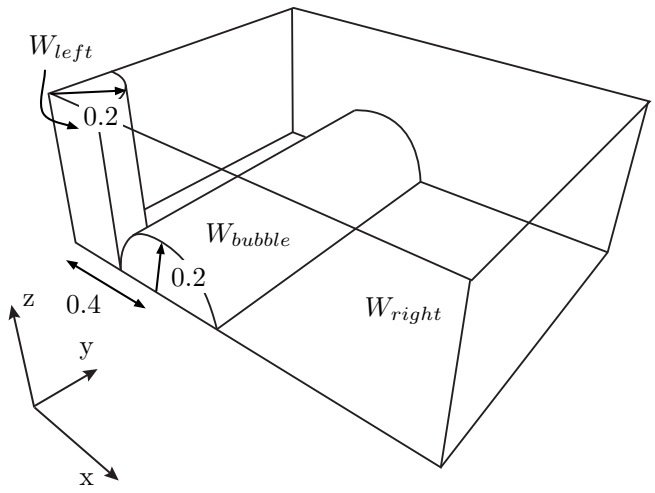

Figure 5.15: Initial configuration for the 3D shock-bubble problem.

Regarding the adaptation parameters, 60 sub-intervals are used, and 5 fixed-point iterations are performed. The initial mesh has $180,000$ vertices, and the space-time complexity is set so that the average number of vertices for the adapted meshes is $1,000,000$. More precisely, the number of vertices is in a range from $821,064$ to $1,504,609$ vertices. The adaptation sensor is the density and space-time interpolation error on the solution is controlled in L^2 norm.

Snapshots of the mesh and of the solution can be found in Fig. 5.16 and 5.17. It can be noted that this case is not a 3D extrusion of the 2D case, and the patterns created are really three-dimensional. The adaptation bands are clearly visible on the meshes, and the accuracy of the shock waves is greatly improved. Compared to the previous case, smaller scale mixing features are present in the flow, and multiscale adaptation aims at capturing them as well as the shock waves. A closer look at Fig. 5.16 and 5.17 confirms that these flow features are indeed captured, on the surface and in the volume. Quantitatively speaking, the mesh accuracy is

around $5.7\,e^{-6}$ at dimensionless time 0.080 and $2\,e^{-5}$ at dimensionless time 0.5 for a domain size of $[0, 1.5] \times [0, 1] \times [0, 0.5]$.

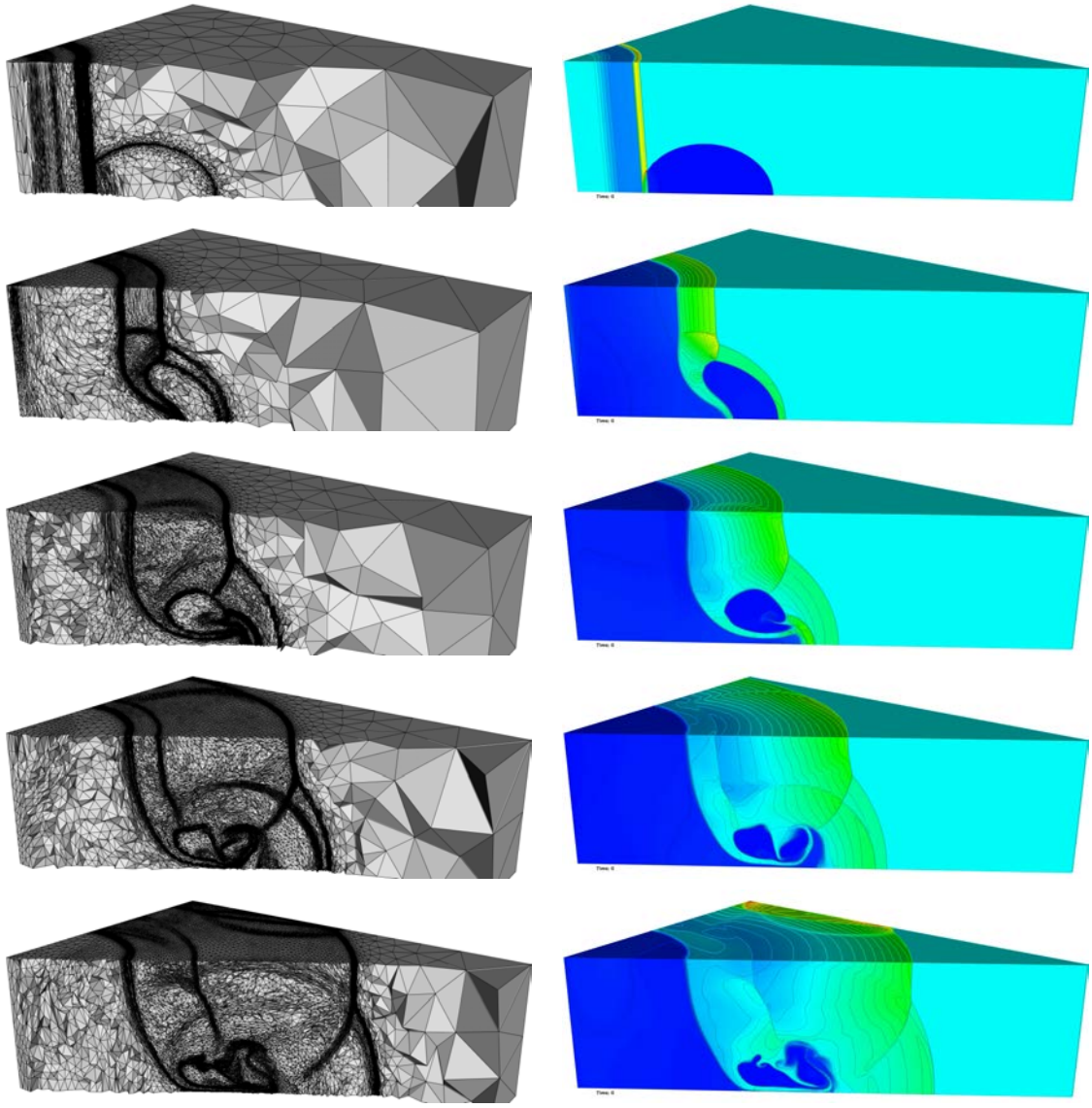

Figure 5.16: 3D shock-bubble interaction: snapshots of adapted meshes (left) and solution (right) at dimensionless times $t = 0.025$, $t = 0.14$, $t = 0.26$, $t = 0.38$ and $t = 0.5$ from top to bottom.

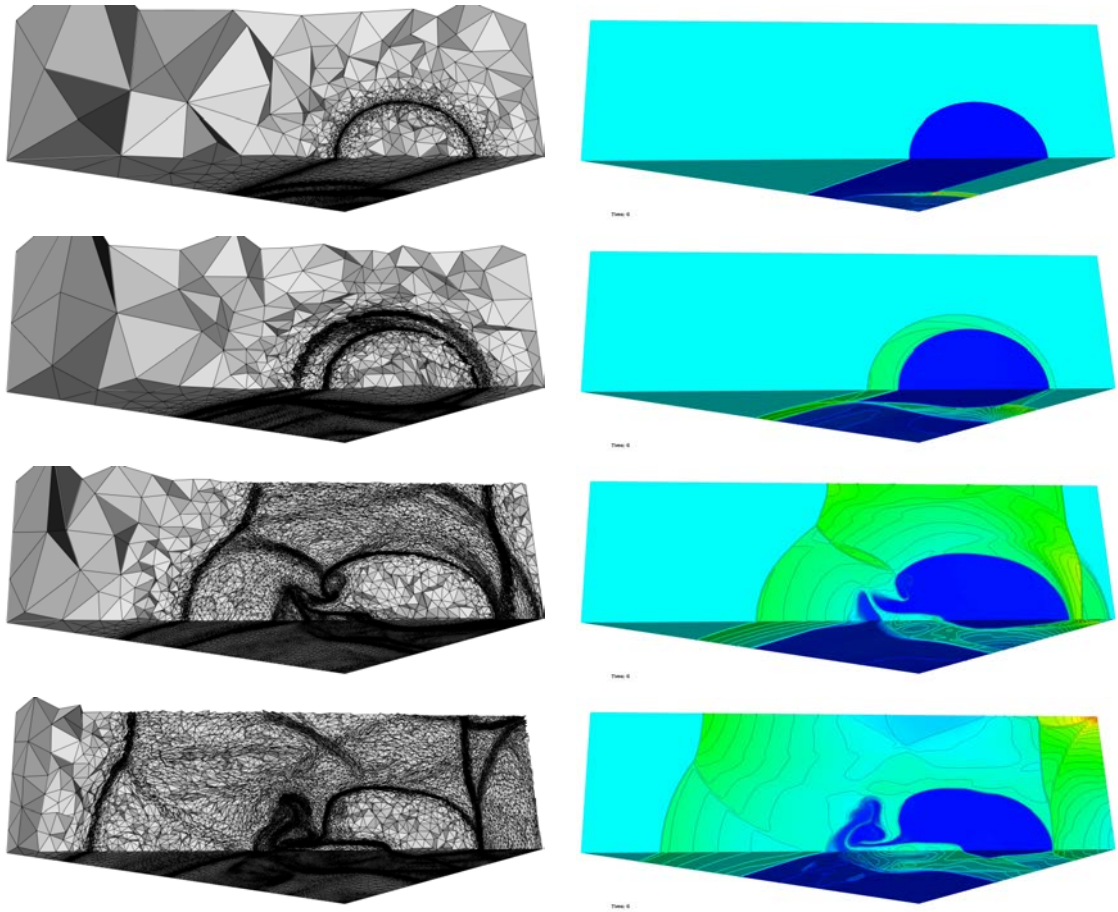

Figure 5.17: 3D shock-bubble interaction: snapshots of adapted meshes (left) and solution (right) at dimensionless times $t = 0.06$, $t = 0.21$, $t = 0.35$, $t = 0.5$ from top to bottom.

5.4.4 3D blast on the London Tower Bridge

Finally, the last example involves a very complex geometry. Indeed, we consider a blast on the London Tower Bridge, whose geometry was the object of the meshing contest of the 23rd International Meshing Roundtable meshing context. A CAD description of the bridge was provided, from which an initial surface and volume mesh were generated by us. The geometry and surface mesh are shown in Fig. 5.18, on which one can see all the tiny details of the surface of the building. These details are a challenge for mesh adaptation, because they result in complex pattern of the solution that the error estimate has to be able to capture, and because the adaptative (re)mesher has to be able to preserve them and preserve the anisotropy in their vicinity.

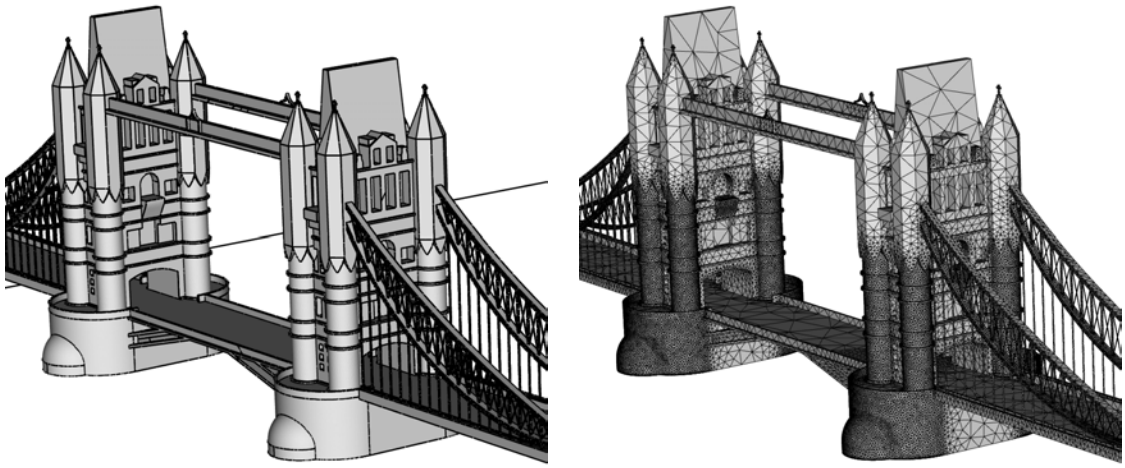

Figure 5.18: London tower bridge case: geometry and initial surface mesh.

The gas is initially at rest in a box of dimension $[-130, 130] \times [-72, 77] \times [0, 100]$. The gas is in a constant state U_0 everywhere, except for a sphere of radius 1 centered in $(3.5, 20, 10)$, where it is in state U_{blast}:

$$\begin{pmatrix} \rho_0 \\ \rho \mathbf{u}_0 \\ \rho E_0 \end{pmatrix} = \begin{pmatrix} 1 \\ \mathbf{0} \\ 2.5 \end{pmatrix} \text{ and } \begin{pmatrix} \rho_{blast} \\ \rho \mathbf{u}_{blast} \\ \rho E_{blast} \end{pmatrix} = \begin{pmatrix} 10 \\ \mathbf{0} \\ 250 \end{pmatrix}$$

The gas is then left to evolve freely until dimensionless time $t = 25$, and wall conditions are imposed on the boundaries. The spherical shock expands in the volume and hits the walls of the bridge, and insinuates itself in any hole of the geometry, while a "mushroom" develops itself in the center. We expect the adaptation algorithm to be able to capture both phenomena: the shocks and as many of their reflection as possible as well as the smaller scale instabilities of the mushroom.

To that aim, the simulation interval was split into 80 sub-intervals. The density field of the solution is chosen as sensor for the adaptation, and the error is controlled in L^2 norm.

To speed the computation up, the adaptation loop was run several times with growing space-time complexities: first one iteration with 40 sub-intervals, the solutions and hessians being used as input for another loop of 5 iterations with 80 sub-intervals but and a lower space-time complexity constraint, and finally one iteration with 80 sub-intervals and the targeted complexity constraint. This approach is faster than running 5 iterations of the loop with a high complexity from the start. For the last iteration, the complexity constraint was set such that the meshes have a size between 1 million and 3.8 million vertices, and the average mesh size is 2.97 million vertices. A total time of 144 hours was necessary to run the whole simulation (the iteration with 40 sub-intervals: 13 hours, the 5 iterations with 80 sub-interval and a smaller complexity: 110 hours, and the iteration with 80 sub-intervals and targeted complexity: 21 hours).

Snapshots of the meshes and solutions are shown in Fig. 5.21. The band-shaped regions distinctive of the adaptation sub-intervals are clearly visible. The accuracy of the solution is remarkable: the shocks are captured precisely, their reflections on the walls as well, even recirculations around the pillars of the bridge can be observed. The mushroom at the center of the blast is also captured with a great level of detail, as can be seen in Fig. 5.20. Views of the surface mesh can be seen in Fig. 5.19. One can notably see that the surface mesh is also adapted and anisotropic, regardless the complexity of the geometry.

At dimensionless time 3.125, the mesh accuracy is around $1.4\,e^{-3}$, and $1.1\,e^{-3}$ at dimensionless time 25 for a domain size of $[-130, 130] \times [-72, 77] \times [0, 100]$. If the domain had had to be meshed uniformly with this mesh size, several billions of elements would have been required, and the computational time would have exploded. This illustrates clearly the interest of unsteady mesh adaptation, if need still be.

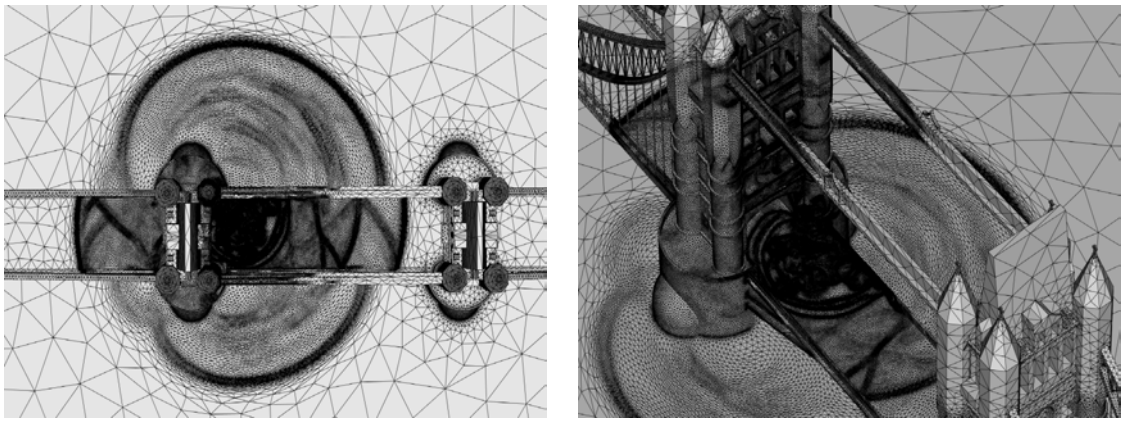

Figure 5.19: London tower bridge case: surface mesh at dimensionless time $t = 21$.

5.5 Parallelization of the mesh adaptation loop

5.5.1 Choices of implementation

All the steps of the adaptation loop have been parallelized to take advantage of modern multi-core computers. For now, we only consider shared memory computers. Our implementation of

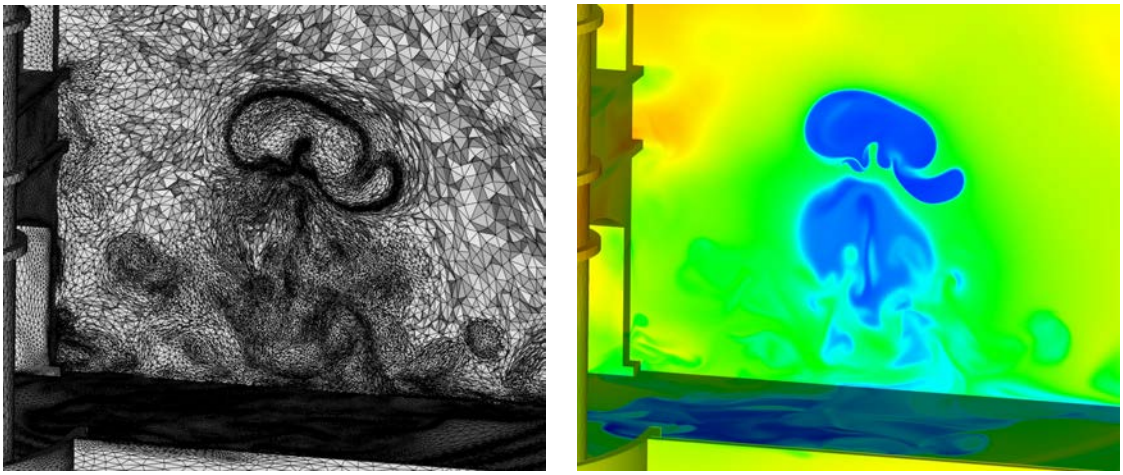

Figure 5.20: London tower bridge case: zoom on details of the mesh and the solution dimensionless time $t = 21$.

the mesh adaptation algorithm described above requires successively the four software components listed in Section 5.2.5.

Two different approaches are used for the two big parts of the loop: The solution computation, the solution transfer procedure and the estimation of the error are parallelized using a p-thread paradigm, whereas a pipelining strategy was selected for the metrics gradation and the generation of the meshes.

P-threads parallelization. The first strategy is an intrusive parallelization of the code using the p-threads paradigm for shared-memory cache-based parallel computers. It was already detailed in Part I Section 2.5.2. One of the main assets of this strategy resides in a slight impact on the source code implementation and on the numerical algorithms. Parallelization is at the loop level and requires few modifications of the serial code [Maréchal 2016]. However, to be efficient, this approach requires the use of a renumbering strategy, based of Hilbert space filling curves [Alauzet 2009].

Pipelining parallelization. After Step 3 of Algorithm 4, the metrics for all the sub-intervals have been computed, and there remains only to perform metric gradation and to generate the corresponding meshes. These tasks are totally independent for each sub-interval and can thus be run at the same time. Hence the pipelining approach adopted for these steps: each available processor executes the two tasks in serial for a given sub-interval, and then handles another sub-interval when it is done with the first one, until all the meshes have been generated. The main advantages of this method is that it requires no modification of the serial codes, and that it can be easily used either on a single machine or on an heterogeneous architecture. To improve the efficiency of the parallelization, it is necessary to organize the distribution of the sub-intervals on the processors to balance the load on each processor. In our implementation, the sub-intervals are sorted in descending order of the expected size of the corresponding mesh, supposing that the time to generate a mesh is approximately proportional to its size. Figure 5.22 explains why this is better than sorting in ascending order: if we have four meshes of different sizes and two

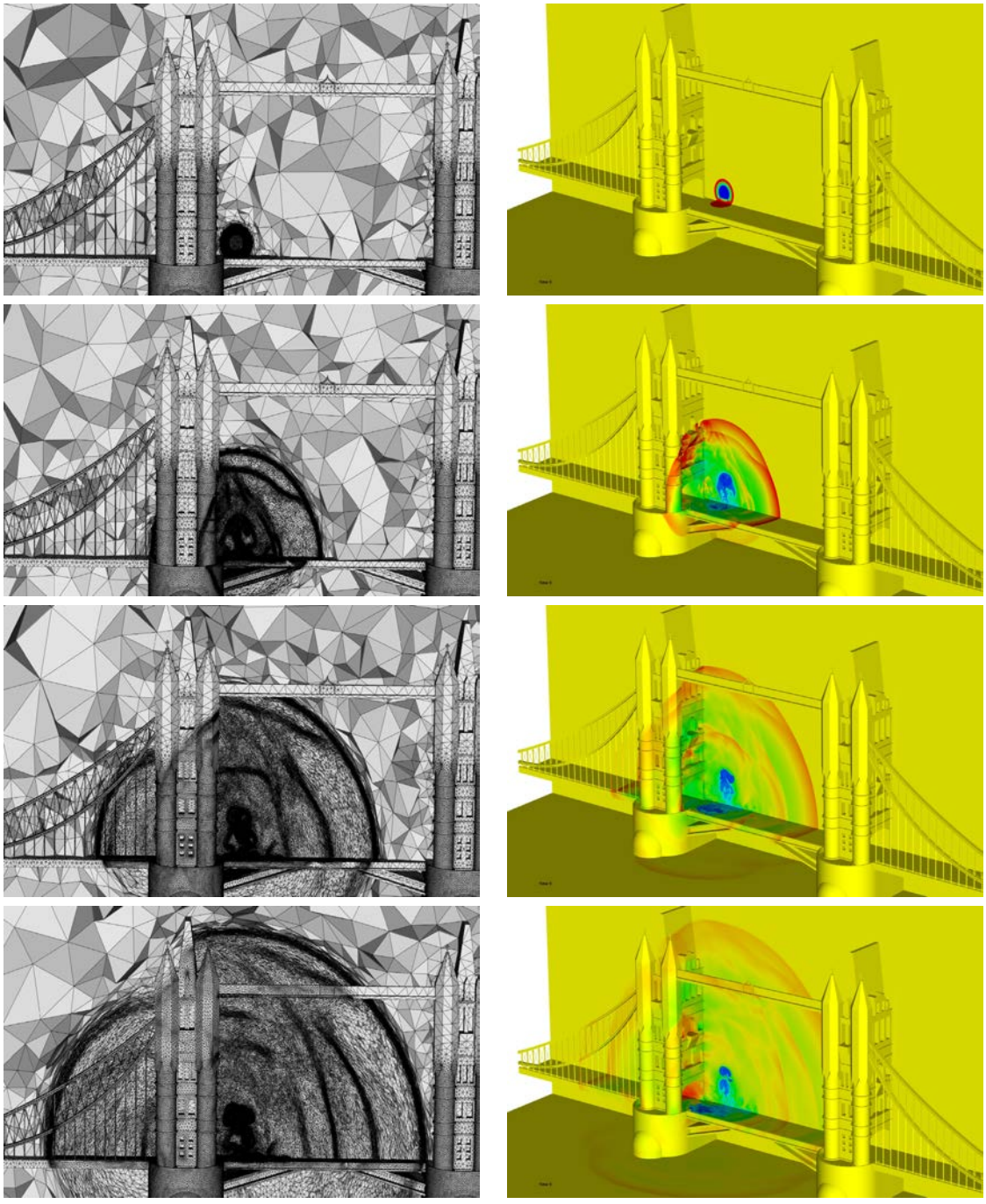

Figure 5.21: London tower bridge case: snapshots of adapted meshes and solution at dimensionless times $t = 0.63$, $t = 8.6$, $t = 17.2$, $t = 25$.

processors available, and that the two smaller ones are sent on the two processors first, then the two larger ones, the first processor will handle the second and the fourth meshes, whose accumulated size is larger than the accumulated sizes of the first and third meshes handled by

the first processor. If the meshes are sorted in descending order, the second processor has the time to handle two meshes while the first one handles the longest one.

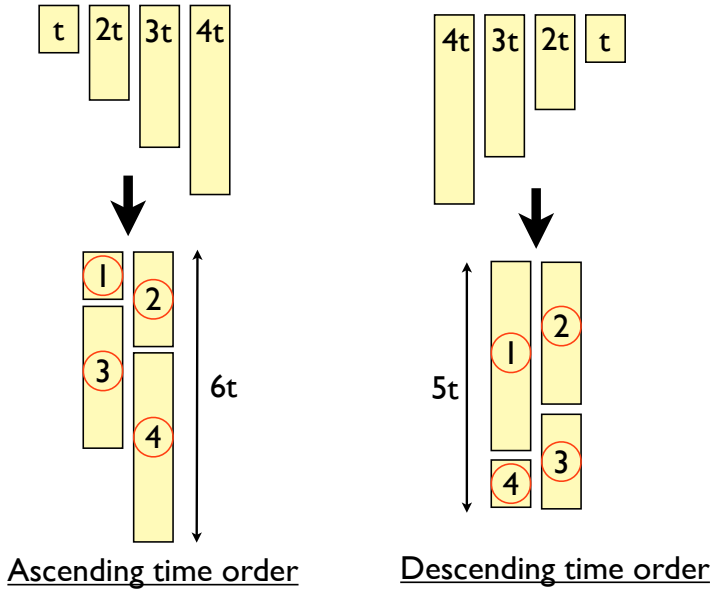

Figure 5.22: Sorting the sub-intervals in descending order of their expected runtime results in better balancing of the effort and a smaller overall runtime.

Data communication. In our implementation of the mesh adaptation algorithm, independent dedicated software are used for each step of the adaptation loop. The advantage of the proposed strategy is its flexibility. Each software block can be developed independently with its own programming language and its own optimal internal database. For instance, the flow solver can keep a static database, the mesh generator can use specific topological data structures such as the elements neighbors, etc. Consequently, we may expect a higher efficiency in memory and in CPU time for each software. The main drawback is that between two stages, the data is stored on the disk, and each software component has to write and read the data on the disk - typically a mesh and a solution field - and to rebuild its internal database. This results in a larger part devoted to I/O as compared to the all-in-one approach. However, it is expected that the CPU time for the I/O is generally negligible compared to the global CPU time. To reduce these communications, the amount of transfer out-of-core data is minimized and binary files are used.

A solution to improve the I/O overhead would be to let each software component read and build its database while the previous one is still computing. Of course, not all of the data could be loaded before the end of a computation, but in some cases a large part of it is already available. For instance, during the part of the loop where the solver and the interpolation code are alternating, the meshes are already known in advance, and could be pre-loaded, and the costly building of the topological database done in advance. Only the solution, which is lighter than the mesh, would have to be transferred between two code runs. This approach would however require some important changes in the organization of the initialization parts of the

concerned codes, and the codes would have to communicate in some way that is robust and cross-platform to know when they can start reading the data and when they have to wait for the end of the data to be written.

5.5.2 Analysis of parallel timings

In what follows, we present an analysis of parallel timings carried out on the blast test case presented previously.

Test case considered. The test case considered is the 3D spherical blast in a box presented in Section 5.4.2.

Machines used. Parallel performance has been analyzed on two different shared-memory multi-core computers with different processors and memory access speeds:

- Computer 1:
  - 2 chips: Xeon E5-2670 10 cores 2.5 GHz
  - both chips are connected by 2 QPI links with a speed of 16 GB/s
- Computer 2:
  - 4 chips: Xeon E7-4850 10 cores 2 GHz
  - all chips are connected to all by 1 QPI link with a speed of 16 GB/s

The characteristics of the machines and the memory mapping are represented in Figure 5.23.

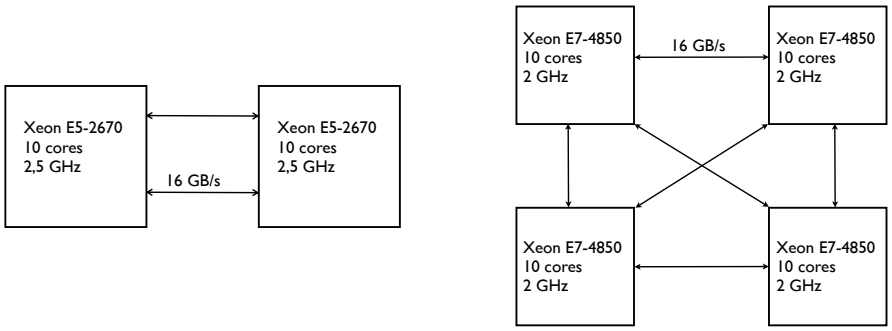

Figure 5.23: The two machines used to analyze the parallel performances of the adaptation loop. The squares represent the chips, and the arrows the memory links between the chips.

The machines considered have hyper-threading (HT) capabilities. For each physical core, the operating system sees two virtual cores, and distributes the workload between both when possible. Notably, it hides memory latency and cache-misses by running two processes alternatively, running one when the other is waiting for the memory. In this study, hyper-threading has been used by launching twice as many threads as the CPU has physical cores. For instance, the 4HT run means than 4 cores have been used but 8 threads have been launched. Hyper-threading clearly benefits to parallel performance, as is visible in the parallel timings (see tables

below) by comparing the serial run and the parallel run on one core with hyper-threading, *i.e.,* the 1HT columns. It is also mandatory to reach a good speed-up on multi-chips architectures by counterbalancing slower speed memory access between chips.

Timings for the solver. For the flow solver profile, the solution is computed from a-dimensioned time 0.6945 to 0.7 which corresponds to the last sub-interval when $n_{adap} = 128$. The mesh size is $2,173,612$ vertices, $13,037,975$ tetrahedra and $75,090$ boundary triangles. The solver performs 36 Runge-Kutta iterations with a 5-step scheme, leading to a total of 180 time steps. The initialization stage of the code and the I/O are not taken into account. Timings and speed-ups for Computer 1 and 2 are summarized in Table 5.1 and 5.2, respectively. We can see that the strong speed up is almost perfect up to 1 chip with 10 cores with hyper-threading. The speed-up decreases when 2 chips are used, which is due to memory access speed between chips, but it still remains good and very good on Computer 1 and 2, respectively. However, the speed-up drops drastically on 4 chips on Computer 2 because there is not enough memory links between the chips.

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 5.1: Flow solver timings and speed-up on computer 1 up to 20 cores with 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 5.2: Flow solver timings and speed-up on computer 2 up to 40 cores with hyper-threading.

Timings for the interpolation code. The five solution fields at a-dimensioned time 0.6945 of the spherical blast problem are interpolated. The background mesh size is $2,166,190$ vertices and $12,993,399$ tetrahedra and the new mesh size is $2,173,612$ vertices and $13,037,975$ tetrahedra.For that case, 226 million tetrahedron-tetrahedron intersections have been computed and 1.7 billion tetrahedra have been generated to mesh the intersections. The initialization stage of the code and the I/O are one again not taken into account. Timings and speed-ups for Computer 1 and 2 are summarized in Table 5.3 and 5.4, respectively. We observe that the speed-ups are excellent on both computers and even super-linear for a low number of cores thanks to hyper-threading which reduces memory latency. However, for Computer 2, when more chips are used, the speed-up degrades, which is again mainly due to slower memory access between the chips (only one link between each).

Timings for a full adaptation cycle. Finally, the parallel performance of a full adaptation cycle is analyzed: all the codes involved are timed, and everything is taken into account, including software initializations and I/O. The time frame has been split into 20 sub-intervals, *i.e.,* $n_{adap} = 20$. The space-time complexity prescription leads to meshes of sizes from $1,096,498$

Nbr. cores	Serial	1 HT	2 HT	4 HT	8 HT	10 HT	20 HT
Timings (sec.)	1,435	1,071	562	301	158	126	72
Speed-up	*1.0*	*1.3*	*2.6*	*4.8*	*9.1*	*11.3*	*19.93*

Table 5.3: Solution transfer timings and speed-up on computer 1 up to 20 cores with hyper-threading.

Nbr. cores	Serial	1 HT	2 HT	4 HT	10 HT	20 HT	40 HT
Timings (sec.)	2,087	1,625	1,011	413	193	115	63
Speed-up	*1.0*	*1.3*	*2.0*	*5.0*	*10.8*	*18.1*	*33.1*

Table 5.4: Solution transfer timings and speed-up on computer 2 up to 40 cores with hyper-threading.

to $2,067,655$, *i.e.* an average spatial mesh size for each sub-interval of $1,631,056$ vertices and $9,769,645$ tetrahedra. Timings and speed-ups for Computer 1 are given in Table 5.5. Several points can be noted:

- Global speed-ups are good, considered the complexity of the task, with a speed-up of 13 for 20 cores.
- Hyper-threading has a significant impact to accelerate the codes.
- One of the main reasons why the scaling is not perfect is the time dedicated to I/O and file management. This is particularly true for the solver code that often needs to save solutions in an unsteady context. A discussion on that topic was already proposed previously.
- The pipelining approach for the generation of the mesh is efficient up to a certain point: when the number of meshes is of the same order of magnitude than the number of available processors, the whole step can be slowed down by the largest mesh to generate.
- Most of the CPU time is spent on the computation of the solution and then interpolation step: respectively 60% and 30% of the total CPU time. In other words, the overhead due to the adaptation process is not so big, compared to the gain in terms of accuracy of the solution.

5.5.3 Conclusion

In conclusion of that study, let us give one more figure: the solver in the last fixed-point iteration, *i.e.* on the last series of adapted meshes, takes 16 hours in serial. In contrast, the whole adaptation process on an hyper-threaded 20-core machine takes only 4 hours. The impact of the parallelization of the process becomes obvious: even if the speed-ups are not perfect, the whole process is much faster when using the parallel capabilities of a modern computer than just computing the solution on the adapted meshes in serial.

Nbr. cores	1	2	4	8	16	20	20 HT
Timing Interp. (min)	921	493	257	145	82	72	58
Speed-up	*1.0*	*1.9*	*3.6*	*6.3*	*11.2*	*12.9*	*15.9*
Timing Solver (min)	1,865	977	511	310	172	151	145
Speed-up	*1.0*	*1.9*	*3.6*	*6.0*	*10.8*	*12.4*	*12.8*
Timing Metric comp. (min)	50	28	16	14	7	7	7
Speed-up	*1.0*	*1.8*	*3.2*	*3.6*	*6.9*	*7.5*	*7.6*
Timing Grad. + Remesher (min)	301	162	79	53	29	29	30
Speed-up	*1.0*	*1.9*	*3.8*	*5.7*	*10.3*	*10.4*	*9.9*
Timing total (min)	**3090**	**1636**	**850**	**513**	**287**	**254**	**237**
Speed-up	***1.0***	***1.9***	***3.6***	***6.0***	***10.8***	***12.2***	***13***

Table 5.5: Full adaptation cycle (4 iterations) timings and speed-up on computer 1 up to 20 cores with hyper-threading.

5.6 Conclusion

In this chapter, we have presented an analysis of the space-time interpolation error based on the continuous mesh framework, that results in a new optimal L^p unsteady metric and leads to improving the unsteady adaptation algorithm and turn it into a global fixed-point algorithm. The different steps of the practical implementation of the algorithm have been described. Two different methods to compute the average Hessian-metric have been compared, and we concluded that the L^1 average ("sum of metrics") was preferable to the L^∞ average ("intersection of metrics"). Thanks to its efficient parallelization, the algorithm allows us to run adaptative simulations on both complex CFD cases in a very reasonable time, with a huge gain in terms of solution accuracy. Several examples of such simulations, complex notably due to the size of the meshes, the complexity of the flow and of the geometry, have been run using the unsteady adaptation algorithm, demonstrating the efficiency of the approach.

Chapter 6

Extension of unsteady mesh adaptation to moving mesh simulations

The global fixed-point adaptation algorithm being efficient to deal with time-dependent simulations with fixed meshes, we would like to extend it to time-dependent simulations with moving boundaries. The strategy adopted to move the meshes is the one described in Part I: one body-fitted mesh is deformed to follow the moving boundaries, and an Arbitrary-Lagrangian-Eulerian (ALE) formulation of the equations is solved on this moving mesh. This moving mesh strategy preserves the number of vertices (no vertex insertion or deletion), which is a requirement in our error analysis framework. However, the error analysis presented in Chapter 5 does not take into account a possible motion of the vertices.

Several approaches can be considered to handle moving meshes within the adaptation algorithm. The main question is: how should metrics be handled ? The first approach is a Eulerian approach: the metric field is a "background" field, evolving in space and time but independently from the moving mesh. To know the value of the metric at a vertex, a projection and an interpolation has to be performed. This approach is simpler from a theoretical point of view, since the movement of the mesh is not taken into account in the metric field. However, it may be very costly in terms of CPU due to the numerous interpolations throughout a whole simulation. That is why the approach considered in this work is a Lagrangian approach: a metric is attached to a moving vertex and moves with it. If no costly interpolation step is necessary, the metric has to be corrected to take the motion into account. In pioneer works [Alauzet 2011b], no correction was performed, which gave satisfying results provided that the mesh displacement is not too large with respect to solution evolution. In this thesis, following [Olivier 2011a], we propose an ALE correction of the optimal unsteady metric, allowing us to plug it into the adaptation algorithm.

An ALE metric, which brings a first answer to this problem, was proposed in [Olivier 2011a]. In this chapter, after recalling this ALE metric, we analyze its performance on 2D and 3D analytic test cases. Then we propose an error analysis, which leads to integrating this ALE metric into the global fixed-point algorithm, and we detail how the algorithm is modified to handle the moving mesh. Finally, several examples of adaptative simulations in three dimensions and with moving boundaries are given and analyzed.

The works presented in this Chapter were published in [Barral 2015].

6.1 ALE metric

We consider a time evolving sensor $u(t)$, and a mesh moved with the strategy we just described. The problem is now: how to make sure that the mesh adapted to the beginning of a sub-interval will still be adapted all along the sub-interval, as it is moved with the displacement prescribed by the elasticity solution?

Let us first consider a simplified problem. t^n and t^{n+1} are two times, Ω^n and Ω^{n+1} the spatial domain at t^n and t^{n+1} respectively, and Φ is a mapping between those two domains (we assume that this mapping exists and is a diffeomorphism):

$$\begin{array}{rccl} \boldsymbol{\phi}: & \Omega^n & \longrightarrow & \Omega^{n+1} \\ & \mathbf{x}^n & \longmapsto & \mathbf{x}^{n+1} = \boldsymbol{\phi}\left(\mathbf{x}^n\right). \end{array} \tag{6.1}$$

and $\mathbf{d}$ is the corresponding mesh displacement field, such that:

$$\mathbf{x}^{n+1} = \boldsymbol{\phi}(\mathbf{x}^n) = \mathbf{x}^n + \mathbf{d}(\mathbf{x}^n). \tag{6.2}$$

We want to find the metric $\mathcal{M}_{L^p}^{n,\mathrm{ALE}}$ from which we will generate a mesh at time t^n that, once moved with displacement $\mathbf{d}$, will be adapted to the sensor u at time t^{n+1}.

In [Alauzet 2011b], analyzing an edge between t^n and t^{n+1}, and considering that the optimal metric at time t^{n+1} is the multiscale L^p metric found previously in Equation (5.7), the following ALE metric at time t^n was proposed [1] :

$$\mathcal{M}_{L^p}^{\mathrm{ALE}}(\mathbf{x}^n) = D_{L^p}^{\mathrm{ALE}} \left[\mathrm{Det}|\widehat{H_u}(\mathbf{x}^n, t^{n+1})|\right]^{-\frac{1}{2p+3}} \left(\nabla^n\boldsymbol{\phi}(\mathbf{x}^n) \cdot |\widehat{H_u}(\mathbf{x}^n, t^{n+1})| \cdot \nabla^n\boldsymbol{\phi}^T(\mathbf{x}^n)\right), \tag{6.3}$$

with

$$D_{L^p}^{\mathrm{ALE}} = \left(N^{n+1}\right)^{\frac{2}{3}} \left(\int_{\Omega^n} \left[\det\left(|\widehat{H_u}(\mathbf{x}^n, t^{n+1})|\right)\right]^{\frac{p}{2p+3}} |\det \nabla^n\Phi(\mathbf{x}^n)| \, \mathrm{d}\mathbf{x}^n\right)^{-\frac{2}{3}}. \tag{6.4}$$

where N^{n+1} stands for the complexity $\mathcal{C}\left(\mathcal{M}_{L^p}^{n+1}\left[u^{n+1}\right]\right)$, and the $\widehat{.}$ operator transports a quantity from Ω^{n+1} to Ω^n: $\widehat{H_u}(\mathbf{x}^n) = H_u(\Phi(\mathbf{x}^n), t^{n+1})$ and ∇^n is the gradient on domain Ω^n. Practically, the gradients are computed using the position of the vertices at time t^n.

This can be rewritten in the more compact form:

$$\mathcal{M}_{L^p}^{\mathrm{ALE}}(\mathbf{x}^n) = \left(\frac{N^{n+1}}{\displaystyle\int_{\Omega^n} \left[\det|H_u^*|\right]^{\frac{p}{2p+3}} \mathrm{d}\mathbf{x}^n}\right)^{\frac{2}{3}} \left\{\mathrm{Det}|H_u^*|\right\}^{-\frac{1}{2p+3}} |H_u^*|, \tag{6.5}$$

with

$$|H_u^*| = \left|\det \nabla^n\boldsymbol{\phi}(\mathbf{x}^n)\right|^{\frac{1}{p}} \left(\nabla^n\boldsymbol{\phi}(\mathbf{x}^n) \cdot |\widehat{H_u}(\mathbf{x}^n, t^{n+1})| \cdot \nabla^n\boldsymbol{\phi}^T(\mathbf{x}^n)\right). \tag{6.6}$$

[1] ∇^n denotes the gradient operator performed on domain Ω^n. Note that the gradient is not the Jacobian, *i.e.* for an arbitrary vector field $\mathbf{f} = (f_1, \ldots, f_n)$, its gradient matrix is $\nabla\mathbf{f} = \left(\frac{\partial f_j}{\partial x_i}\right)_{ij}$.

The latter formulation is similar to the classic unsteady L^p metric formulation of Equation (5.7), except that a Hessian matrix corrected with $\nabla^n \boldsymbol{\phi}$ is used, and it is thus the one used in practice. Relation (5.16) is then used to derive a metric for sub-intervals.

6.2 Analytic examples

The previous result gives us the optimal metric that will lead to an adapted mesh once the mesh it was used to generate is moved with a given displacement. Let us illustrate this result on analytic examples. We consider a variety of sensor functions and displacements, both in 2D and 3D.

6.2.1 Procedure

A uniform mesh $\mathcal{H}_0^n$, a displacement field $\mathbf{d}$ between two times t^n and t^{n+1}, and a sensor u^{n+1} at the final time are given. The goal is to generate a mesh $\mathcal{H}^n$ at the initial time that, once moved into $\mathcal{H}_{ALE}^{n+1} = \mathbf{d}(\mathcal{H}^n)$ will be adapted to the sensor. At the same time, a reference adapted mesh is computed, that is directly adapted to sensor u^{n+1}. To do so, the classic steady-state procedure described in Section 4.2.2.4 is considered. Since the sensor is analytic and to be fair with the ALE version, only one iteration of the algorithm is performed. The metric used is normalized to look like the regular optimal metric for unsteady simulations, obtained from the hessian of the sensor. In what follows, we note $\mathcal{H}_{ALE}^{n+1}$ the mesh obtained with the ALE procedure and $\mathcal{H}_{ref}^{n+1}$ the mesh directly adapted. The procedure is detailed in Algorithm 5.

Algorithm 5 Procedure used to generate the meshes with the ALE metric and the regular metric in order to compare the output.

- Generation of an adapted mesh with the ALE metric
  1. $\nabla^n \boldsymbol{\phi}$ = Compute transformation gradient $((\mathcal{H}_0^n, \mathbf{d})$
  2. $\mathcal{H}_0^{n+1}$ = Move mesh $(\mathcal{H}_0^n, \mathbf{d})$
  3. u^{n+1} = Compute target sensor $\left(\mathcal{H}_0^{n+1}\right)$
  4. $\mathcal{M}_{L^p}^{n,ALE}$ = Compute ALE metric $\left(\mathcal{H}_0^{n+1}, u^{n+1}, \nabla^n \boldsymbol{\phi}\right)$
  5. $\mathcal{H}^n$ = Adapt mesh $\left(\mathcal{H}^n, \mathcal{M}_{L^p}^{n,ALE}\right)$
  6. $\mathcal{H}_{ALE}^{n+1}$ = Move mesh $(\mathcal{H}^n, \mathbf{d})$
- Generation of an adapted with the standard method
  1. $\mathcal{M}_{L^p}^{n+1}$ = Compute metric $\left(\mathcal{H}_0^n, u^{n+1}, H_{u^{n+1}}\right)$
  2. $\mathcal{H}_{ref}^{n+1}$ = Adapt mesh $\left(\mathcal{H}_0^n, \mathcal{M}_{L^p}^{n+1}\right)$

According to the above developments, we expect the mesh $\mathcal{H}^{n+1}$, obtained by moving the vertices of $\mathcal{H}^n$ with $\mathbf{d}$, to be optimal for the control of the interpolation error of u^{n+1} in L^1

norm.

To study the adapted characteristic of the resulting meshes, two quantities are considered:

- the mesh quality, computed using the metric $\mathcal{M}_{L^p}^{n+1}$ of the reference adapted mesh (*i.e.* the one that was adapted directly). The quality measure is the one recalled in Section 4.2.2, Equation (4.16).
- the error committed when the sensor function is projected on the mesh remains the best way to evaluate if the mesh is really adapted to a sensor. Here we consider the interpolation error $\|\Pi_h u - u\|$ for sensor u, where $\Pi_h u$ is the piecewise $\mathbb{P}^1$ function with $\Pi_h u(P_i) = u(P_i)$ for any vertex P_i of the mesh.

The displacement considered are straight-line displacements of somewhat large amplitude, whereas infinitesimal displacements have been considered in the theoretical analysis. However, this is closer to real life simulations, with large displacements even within a sub-interval. Note that the ALE metric does not guarantee that the mesh moved at time t^{n+1} is still valid. In practice, mesh optimizations deal with this issue and ensure the validity of the mesh together with its quality. In this study, no mesh optimization is performed on the adapted mesh, or during the adaptation or moving process, to avoid altering the results of the quality analysis. For the same reason, unlike in the actual adaptation loop, no gradation is performed.

Computation of the error

We compute the error with respect to the analytic solution. To do so, we use quadrature formulas:

$$\begin{aligned} \|\Pi_h u - u\|_{L^1} &= \int_\Omega |\Pi_h u - u| \\ &= \sum_K \int_K |\Pi_h u - u| \\ &= \sum_K |K| \sum_{P_j^Q} w_j \, |\Pi_h u(P_j^Q) - u(P_j^Q)| \end{aligned} \tag{6.7}$$

where K are the elements (triangles or tetrahedra), P_j^Q are the quadrature points for each element and w_j are the weights of the quadrature scheme.

Let K be an element K in dimension n, and $(P_i)_{i=1\ldots n+1}$ its vertices. A quadrature point P^Q is identified with its barycentric coordinates $(\lambda_i)_{i=1\ldots n+1}$. Its Cartesian coordinates are a linear combination of the Cartesian coordinates of the vertices:

$$P^Q = \sum_{i=1\ldots n+1} \lambda_i \, P_i \, .$$

The analytic function can thus easily be computed on each quadrature point. The value of $\Pi_h u(P^Q)$ is a linear combination of the values at the vertices:

$$\Pi_h u(P^Q) = \sum_{i=1\ldots n+1} \lambda_i \, u(P_i)$$

.

We use the third order quadrature schemes described in [Dunavant 1985] for 2D and in [Jinyun 1984] for 3D. In 2D, there are 4 quadrature points, of multiplicity 1 and 3, *cf* Table 6.1, and in 3D there are 5 quadrature points of multiplicity 1 and 4, *cf* Table 6.2.

w_i	λ_1	λ_2	λ_3
-0.5625	1/3	1/3	1/3
0.520833	0.6	0.2	0.2
0.520833	0.2	0.6	0.2
0.520833	0.2	0.2	0.6

Table 6.1: 2D quadrature points for order 3 approximation.

w_i	λ_1	λ_2	λ_3	λ_4
-0.8	1/4	1/4	1/4	1/4
0.45	1/2	1/6	1/6	1/6
0.45	1/6	1/2	1/2	1/6
0.45	1/6	1/6	1/2	1/6
0.45	1/6	1/6	1/6	1/2

Table 6.2: 3D quadrature points for order 3 approximation.

6.2.2 Functions considered

The sensor functions here are taken from [Olivier 2011a], so we can confirm and complete the analysis proposed in this work. Each sensor presents specific features (smoothness, discontinuities, features of different scales...) that are as many difficulties for the adaptation process, so that handling them correctly shows the robustness of the approach. As for the displacements, they are set so that the displacement is null on the boundaries.

In 2D. We have a set of sensors represented in Fig. 6.1:

$$
\begin{aligned}
u_0^{n+1}(x,y) &= x^2 + y^2\,, \\
u_1^{n+1}(x,y) &= \begin{cases} 0.01\sin(50xy) & \text{if } |xy| \geq \dfrac{2\pi}{50} \\ \sin(50xy) & \text{if } |xy| < \dfrac{2\pi}{50} \end{cases}\,, \\
u_2^{n+1}(x,y) &= 0.1\sin(50x) + \arctan\left(\frac{0.1}{\sin(5y) - 2x}\right)\,, \\
u_3^{n+1}(x,y) &= \arctan\left(\frac{0.1}{\sin(5y) - 2x}\right) + \arctan\left(\frac{0.5}{\sin(3y) - 7x}\,,\right)
\end{aligned}
$$

and a set of displacements represented in Fig. 6.2:

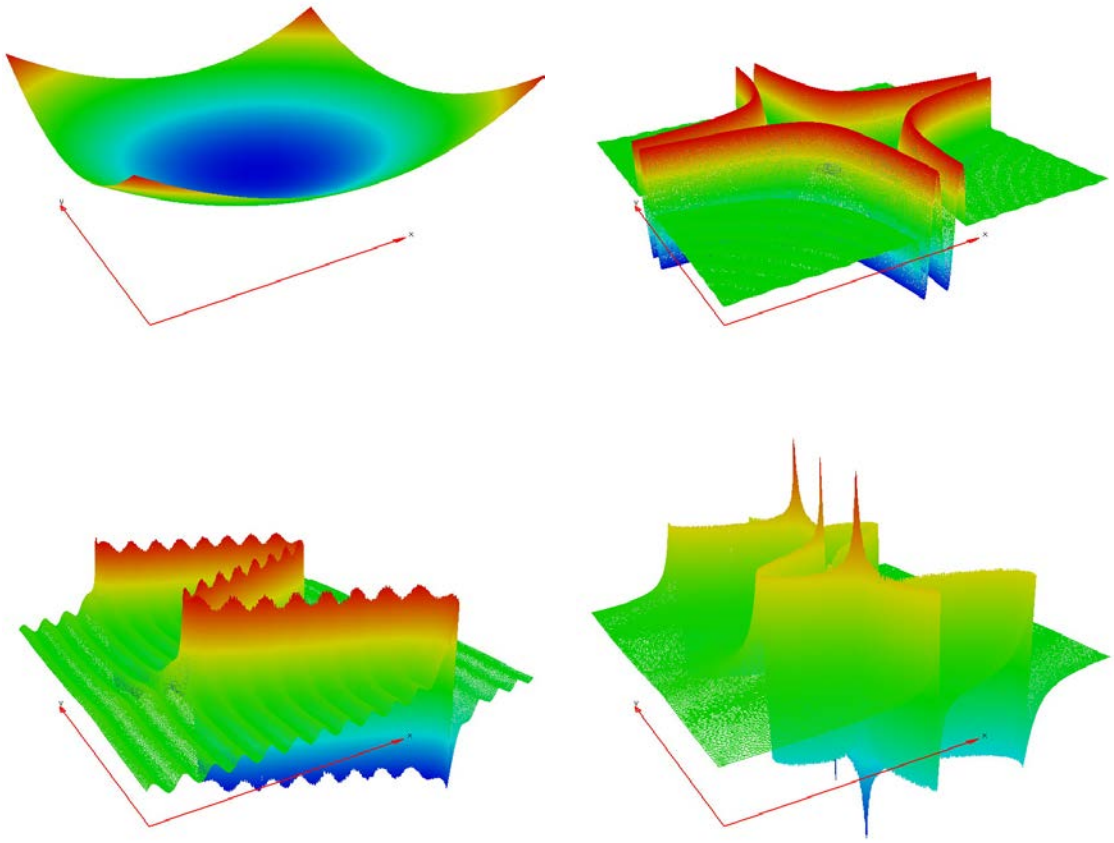

Figure 6.1: Adaptation sensors used: u_0^{n+1} (top left), u_1^{n+1} (top right), u_2^{n+1} (bottom left) and u_3^{n+1} (bottom right).

$$\mathbf{d}_1(x,y) = \begin{bmatrix} \begin{cases} -0.3\,(x+1)\left(y^2-1\right)\exp\left(-5x^2\right), & \text{if } x \geq 0 \\ 0.3\,(x-1)\left(y^2-1\right)\exp\left(-5x^2\right), & \text{if } x < 0 \end{cases} \\ \\ \begin{cases} -0.3\left(x^2-1\right)(y+1)\exp\left(-5y^2\right), & \text{if } y \geq 0 \\ 0.3\left(x^2-1\right)(y-1)\exp\left(-5y^2\right), & \text{if } y < 0 \end{cases} \end{bmatrix},$$

$$\mathbf{d}_2(x,y) = \begin{bmatrix} 0.5(x^2-1)(y^2-1) \\ 0 \end{bmatrix}.$$

In 3D. We have a set of sensors represented in Fig. 6.3:

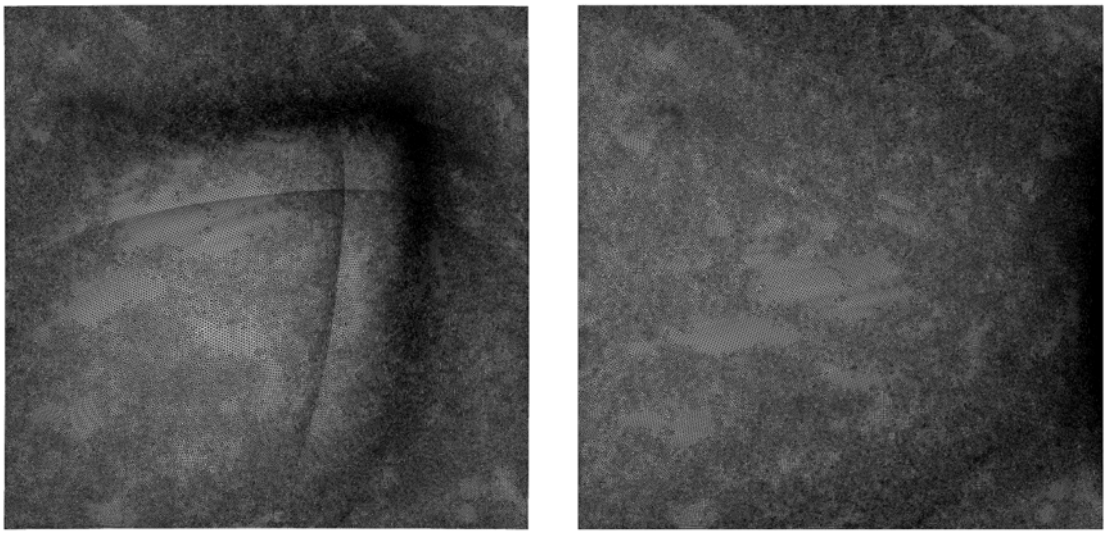

Figure 6.2: Displacements used: uniform mesh moved with displacements $\mathbf{d}_1$ (left) and $\mathbf{d}_2$ (right).

$$
\begin{aligned}
u_4^{n+1}(x,y,z) &= x^2+y^2+z^2\,,\\
u_5^{n+1}(x,y,z) &= \begin{cases} 0.01\sin(50xy) & \text{if } |xy| \geq \dfrac{2\pi}{50} \\ \sin(50xy) & \text{if } |xy| < \dfrac{2\pi}{50}\,, \end{cases}
\end{aligned}
$$

and a set of displacements (note that we have corrected the formulas from [Olivier 2011a]) represented in Fig. 6.4:

$$
\mathbf{d}_3(x,y,z) = \begin{bmatrix} 0 \\ \begin{cases} -0.3\,(y+1)\left(z^2-1\right)\exp\left(-5y^2\right), & \text{if } y \geq 0 \\ 0.3\,(y-1)\left(z^2-1\right)\exp\left(-5y^2\right), & \text{if } y < 0 \end{cases} \\ \begin{cases} -0.3\left(y^2-1\right)(z+1)\exp\left(-5z^2\right), & \text{if } z \geq 0 \\ 0.3\left(y^2-1\right)(z-1)\exp\left(-5z^2\right), & \text{if } z < 0 \end{cases} \end{bmatrix},
$$

$$
\mathbf{d}_4(x,y,z) = 0.5(x^2-1)(y^2-1)(z^2-1)\begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}.
$$

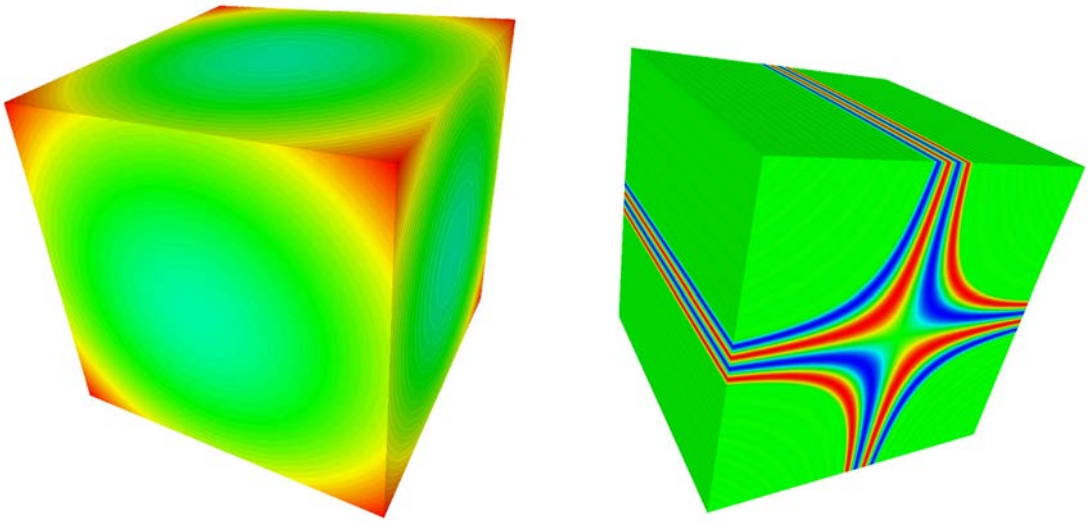

Figure 6.3: Adaptation sensors used: u_4^{n+1} (left) and u_5^{n+1} (right).

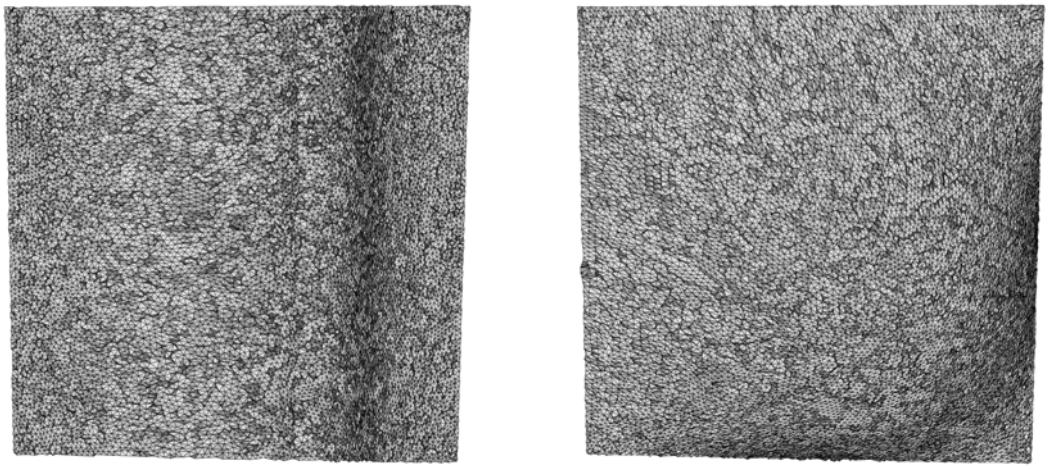

Figure 6.4: Displacements used: uniform mesh moved with displacements $\mathbf{d}_3$ (left) and $\mathbf{d}_4$ (right).

6.2.3 Results

Images of the meshes adapted with the ALE metric at time t^n (before being moved) and t^{n+1} (after being moved) are shown in Fig. 6.5 6.6 6.7 6.8 in 2D and Fig 6.9 and 6.10 in 3D. Visually, we can see that the meshes at time t^{n+1} are indeed adapted to the sensors as expected. Looking at the meshes at time t^n illustrates the mechanism of this adaptation: before the movement, the adapted mesh is deformed so that the areas with small elements are carried away by the displacement. Stretching and compression of the mesh are anticipated by creating smaller or

bigger elements.

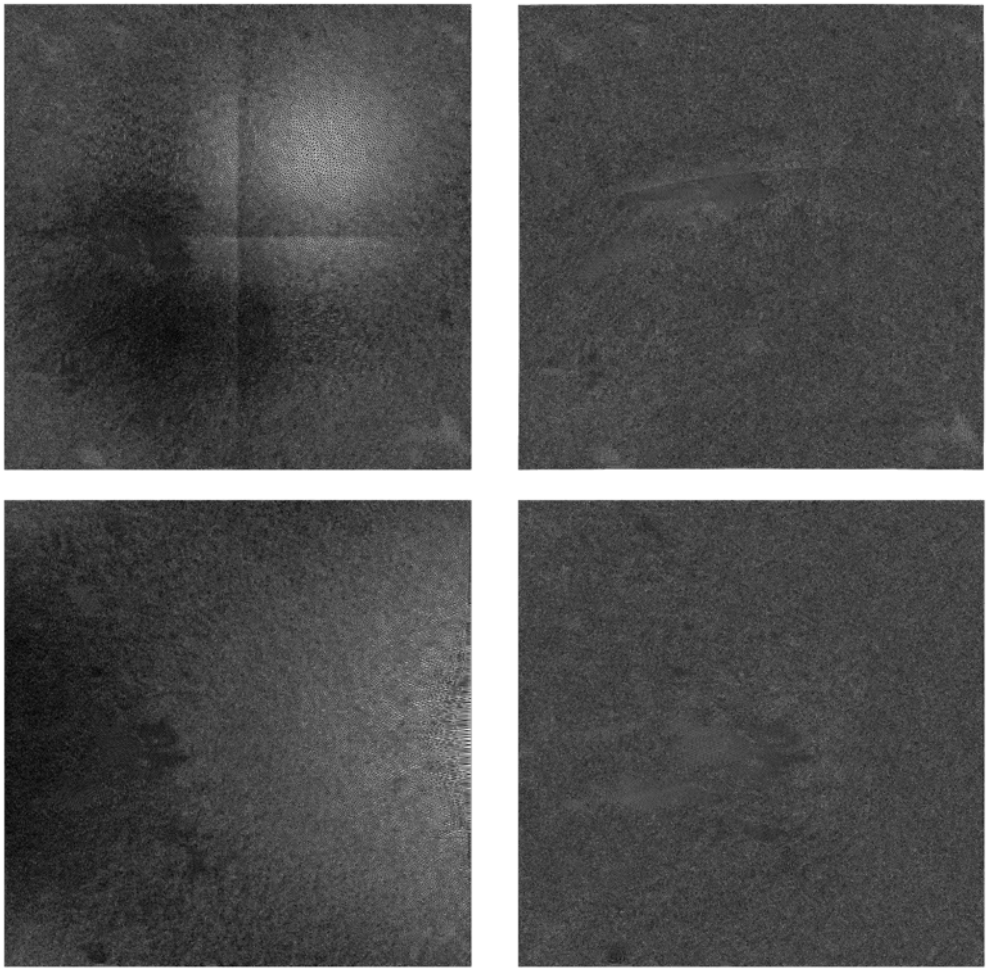

Figure 6.5: Mesh adapted with the ALE metric at time t^n (left) before the displacement and t^{n+1} (right) after the displacement, for sensor u_0^{n+1} and displacements $\mathbf{d}_1$ (top) and $\mathbf{d}_2$ (bottom).

Results for the error

In Tables 6.3 and 6.4, we show the error for the 2D cases, for a uniform mesh, for a mesh adapted directly and for a mesh adapted with the ALE metric and moved. Two mesh complexities are considered: a small one, with $25,000$ vertices, and a bigger one with $250,000$ vertices. First, we can note that for all the sensors, the error on the adapted mesh is generally significantly lower than on the uniform mesh, for the same number of vertices, even when the uniform mesh has small elements. Actually, we can see that the error on an adapted mesh with $25,000$ vertices is close to the error on a uniform mesh with $250,000$ vertices. There is one exception to this observation, for sensor u_0, which is a smooth quadratic function, for which the adapted mesh is slightly less good. Indeed in this case, the optimal mesh is the uniform mesh: the adaptation tends to be over-sensitive to discrete errors when generating the metric, which results in a non

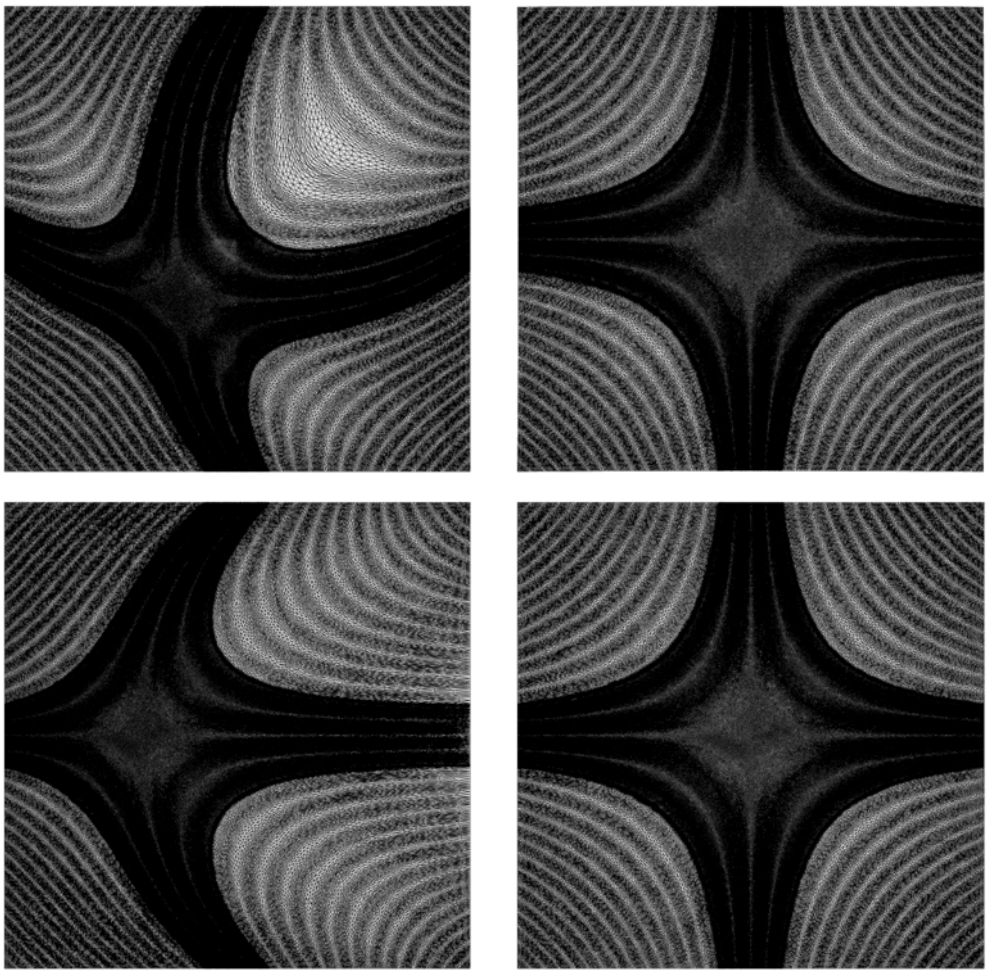

Figure 6.6: Mesh adapted with the ALE metric at time t^n (left) and t^{n+1} (right), for sensor u_1^{n+1} and displacements $\mathbf{d}_1$ (top) and $\mathbf{d}_2$ (bottom).

perfectly uniform mesh. However, the error for the adapted mesh is still very close to the error for the (optimal) uniform mesh, so the adapted mesh is good.

We can now compare the error for the meshes adapted directly and the meshes adapted with the ALE procedure. In all the cases, the error for the two meshes is very close (around 5% of relative difference), which clearly indicates that the ALE procedure results in meshes that are actually adapted as expected. The error for the meshes adapted with the ALE procedure is always slightly higher, which could have been predicted, due to the extra approximation made when deriving the ALE metric. Moreover, in these cases without mesh optimizations, a few elements may have a bad quality that influences the error.

Convergence curves are represented in Fig. 6.11, for sensor u_1 and displacement $\mathbf{d}_1$. The error is plotted for meshes ranging from $13,000$ vertices to $500,000$ vertices. To avoid bias, the initial mesh (on which the metric is computed) is the same in all cases, and has a large enough number of vertices. First, we can see both error curves are very close, the curve of the meshes

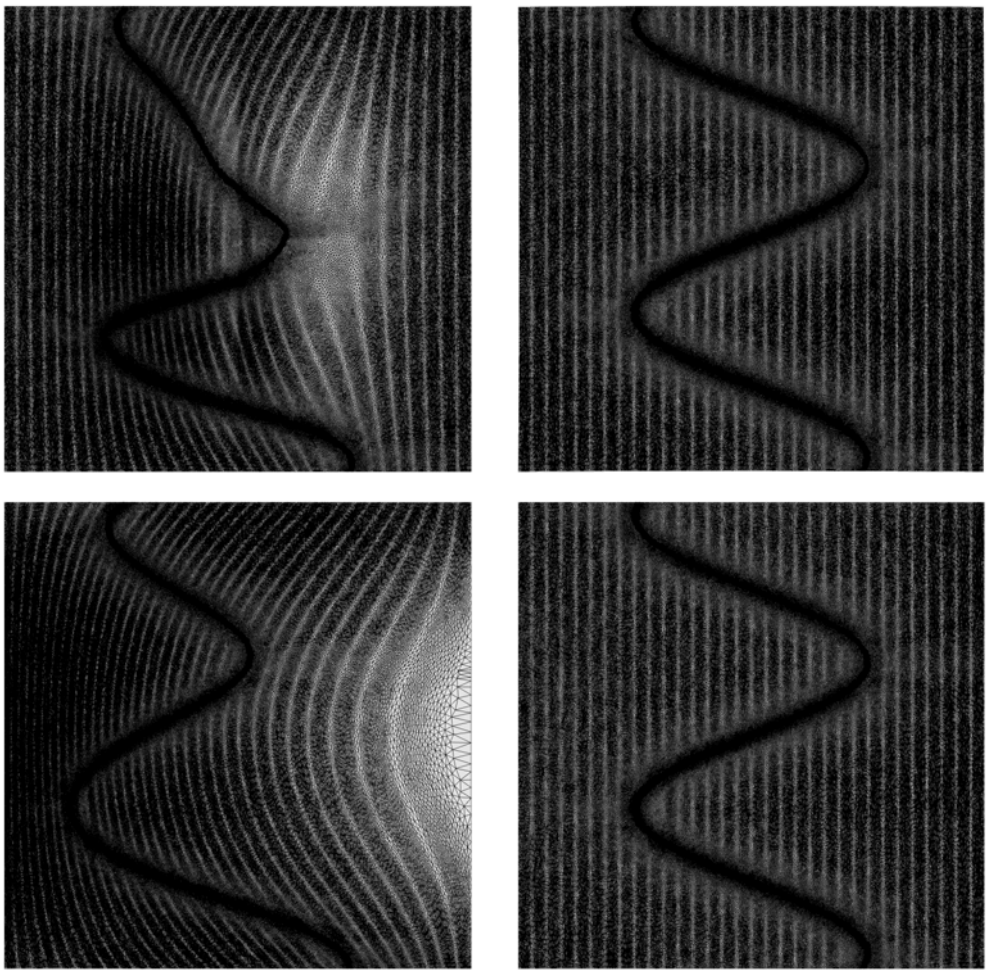

Figure 6.7: Mesh adapted with the ALE metric at time t^n (left) and t^{n+1} (right), for sensor u_2^{n+1} and displacements $\mathbf{d}_1$ (top) and $\mathbf{d}_2$ (bottom).

adapted with the ALE method being only slightly above the reference curve. Furthermore, both curves are straight lines, with the same slope, which is the order 2 slope (see Relation 4.34).

The study of the error for the 3D cases, given in Tables 6.5 and 6.6, for meshes of complexities $600,000$ and $6,000,000$, confirms the efficiency of both direct and with the ALE procedure, adaptation to reduce the error. Once again, the error on the adapted meshes is one order of magnitude smaller than the error on the uniform mesh for the non quadratic sensor, and is very close to the error on the uniform mesh for the quadratic sensor. For the 3D cases, the error for meshes adapted with the ALE metric is still slightly higher. This can be explained by a larger number of bad quality elements obtained after the mesh displacement, which is mainly due to the absence of mesh optimization.

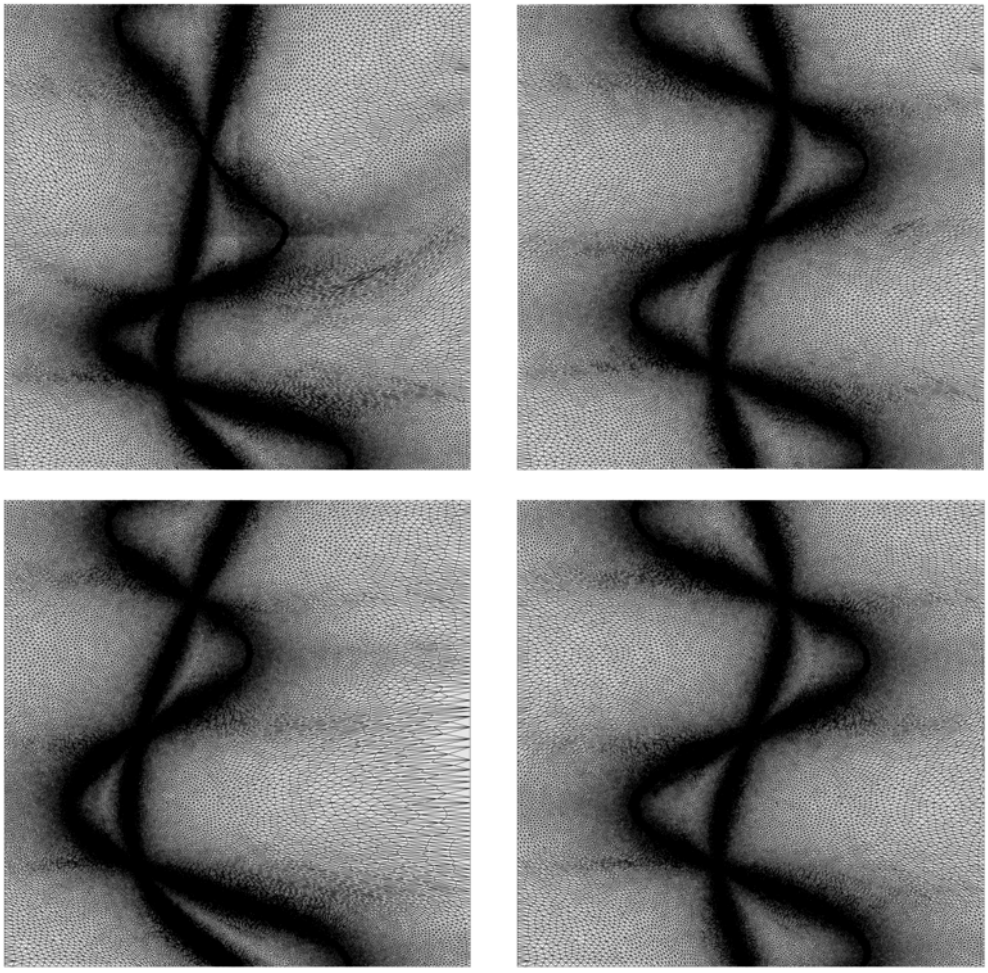

Figure 6.8: Mesh adapted with the ALE metric at time t^n (left) and t^{n+1} (right), for sensor u_3^{n+1} and displacements $\mathbf{d}_1$ (top) and $\mathbf{d}_2$ (bottom).

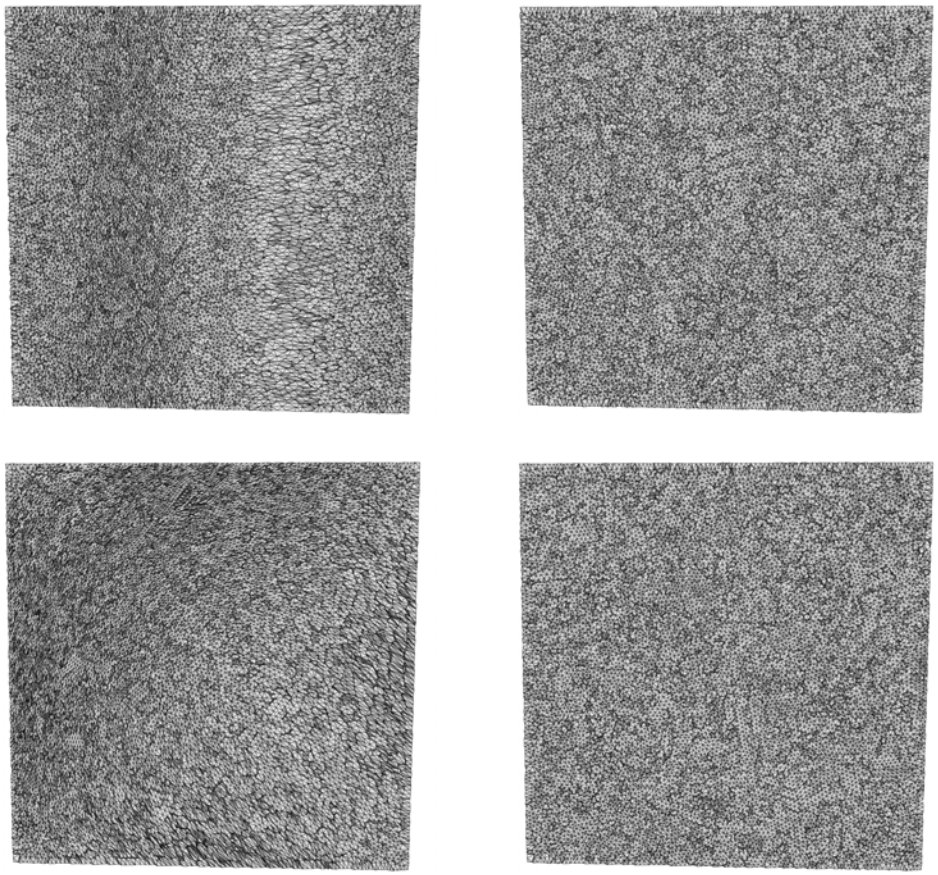

Figure 6.9: Mesh adapted with the ALE metric at time t^n (left) and t^{n+1} (right), for sensor u_4^{n+1} and displacements $\mathbf{d}_3$ (top) and $\mathbf{d}_4$ (bottom).

u	u_0^{n+1}		u_1^{n+1}		u_2^{n+1}		u_3^{n+1}	
d	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_1$	$\mathbf{d}_2$
Error for mesh $\mathcal{H}_{unif}$	4.36e-4	4.36e-4	2.69e-2	2.69e-2	9.05e-2	9.05e-2	1.18e-1	1.18e-1
Error for mesh $\mathcal{H}_{ref}^{n+1}$	4.63e-4	4.63e-4	2.93e-3	2.93e-3	3.09e-2	3.09e-2	3.64e-2	3.64e-2
Error for mesh $\mathcal{H}_{ALE}^{n+1}$	4.65e-4	4.55e-4	3.57e-3	3.56e-3	3.17e-2	3.31e-2	3.95e-2	4.07e-2

Table 6.3: L^1 error for 2D meshes of complexity $25,000$

u	u_0^{n+1}		u_1^{n+1}		u_2^{n+1}		u_3^{n+1}	
d	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_1$	$\mathbf{d}_2$
Error for mesh $\mathcal{H}_{unif}$	4.69e-5	4.69e-5	2.96e-3	2.96e-3	2.60e-2	2.60e-2	3.81e-2	3.81e-2
Error for mesh $\mathcal{H}_{ref}^{n+1}$	4.89e-5	4.89e-5	2.36e-4	2.36e-4	5.48e-3	5.48e-3	6.71e-3	6.71e-3
Error for mesh $\mathcal{H}_{ALE}^{n+1}$	4.69e-5	4.59e-5	2.70e-4	2.72e-4	5.65e-3	5.92e-3	7.26e-3	7.47e-3

Table 6.4: L^1 error for 2D meshes of complexity $250,000$

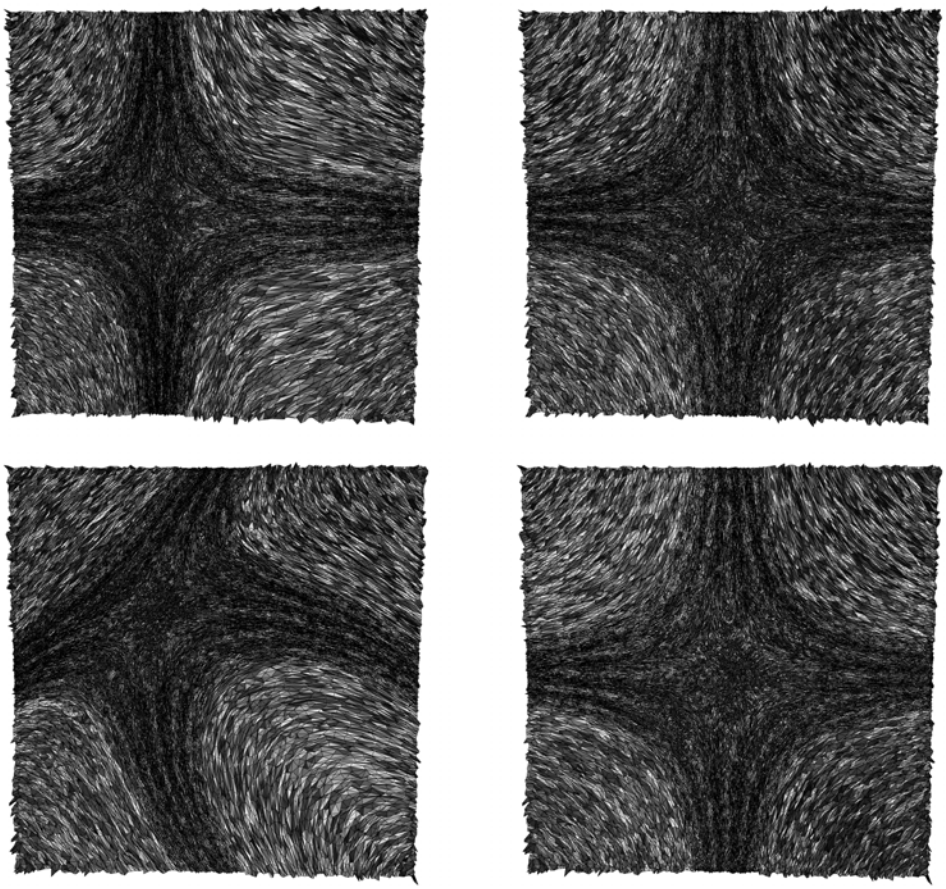

Figure 6.10: Mesh adapted with the ALE metric at time t^n (left) and t^{n+1} (right), for sensor u_5^{n+1} and displacements $\mathbf{d}_3$ (top) and $\mathbf{d}_4$ (bottom).

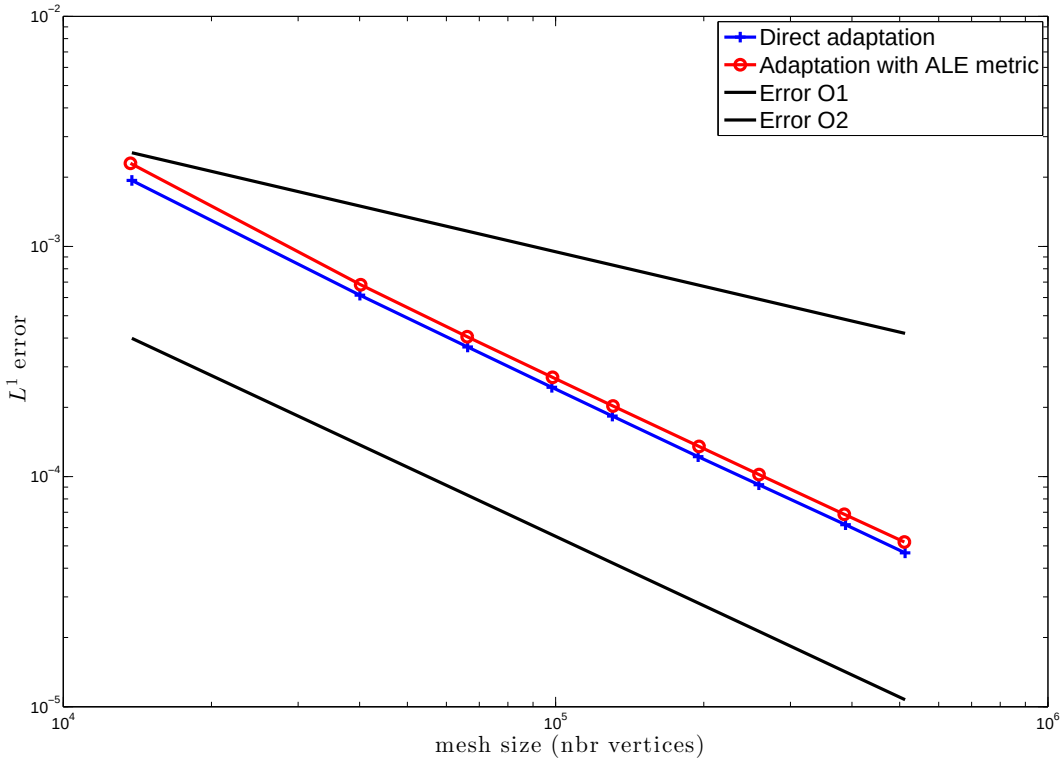

Figure 6.11: L1 error for several adapted mesh sizes for sensor u_1^{n+1}. The initial uniform mesh is the same and of size bigger than all the others.

u	u_4^{n+1}		u_5^{n+1}	
d	$\mathbf{d}_3$	$\mathbf{d}_4$	$\mathbf{d}_3$	$\mathbf{d}_4$
Error for mesh $\mathcal{H}_{unif}$	6.23e-3	6.23e-3	2.32e-1	2.32e-1
Error for mesh $\mathcal{H}_{ref}^{n+1}$	6.31e-3	6.31e-3	1.85-2	1.85e-2
Error for mesh $\mathcal{H}_{ALE}^{n+1}$	6.35e-3	6.31e-3	2.16e-2	2.17e-2

Table 6.5: L^1 error for 3D meshes of complexity $600,000$

u	u_4^{n+1}		u_5^{n+1}	
d	$\mathbf{d}_3$	$\mathbf{d}_4$	$\mathbf{d}_3$	$\mathbf{d}_4$
Error for mesh $\mathcal{H}_{unif}$	1.33e-3	1.33e-3	5.44e-2	5.44e-2
Error for mesh $\mathcal{H}_{ref}^{n+1}$	1.35e-3	1.35e-3	3.84e-3	3.84e-3
Error for mesh $\mathcal{H}_{ALE}^{n+1}$	1.36e-3	1.35e-3	4.50e-3	4.54e-3

Table 6.6: L^1 error for 3D meshes of complexity $6,000,000$

	$\mathcal{H}_{ref}^{n+1}$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_1$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_2$	
Average quality	1.0498		1.0498		1.0524	
Worst quality	2.1572		3.0465		8.5746	
$1.00 < Q < 2.00$	25,431	99.99 %	25,449	100.00 %	25,383	99.91 %
$2.00 < Q < 3.00$	3	0.01 %	0	0.00 %	16	0.06 %
$3.00 < Q < 4.00$	0	0.00 %	1	0.00 %	4	0.02 %
$4.00 < Q < 5.00$	0	0.00 %	0	0.00 %	1	0.00 %
$5.00 < Q < 10.00$	0	0.00 %	0	0.00 %	2	0.01 %

Table 6.7: Quality statistics for sensor u_0^{n+1}: average and worst quality and quality histograms with the number of elements and the percentage for the mesh adapted directly $\mathcal{H}_{ref}^{n+1}$ and the meshes adapted with the ALE procedure $\mathcal{H}_{ALE}^{n+1}$ for the two displacements considered.

Results for the quality

The analysis of the meshes quality is in agreement with the analysis of the error. We have gathered some quality statistics for the mesh adapted directly with the standard procedure ($\mathcal{H}_{ref}^{n+1}$), and the meshes adapted with the ALE procedure ($\mathcal{H}_{ALE}^{n+1}$) with the two displacements. Only the small complexity cases are shown here ($25,000$ vertices in 2D and $600,000$ vertices in 3D) because the results are the same for the larger complexity. We can see that the quality of the meshes adapted with the standard procedure is excellent for all sensors, with generally more than 99% of the elements having a quality between 1 and 2, which could be expected since the quality is computed using the metric that was used to generate these meshes: it proves the quality of the remesher `feflo.a`, both in 2D and 3D. If we have a look at the qualities for the meshes adapted with the ALE metric, we can see that the quality of these meshes is excellent. This clearly shows that meshes that have been generated at t^n with the ALE metric are almost perfectly adapted to the corresponding analytic sensors once moved at time t^{n+1}. More precisely, the meshes for the quadratic sensors u_0 (Tables 6.7) and u_4 (Table 6.11) have virtually no bad element, with the both displacements, since the worst quality is 13 in 2D and 36 in 3D. For the other sensors, u_1 (Tables 6.8), u_2 (Tables 6.9), u_3 (Tables 6.10) in 2D and u_5 (Table 6.12) in 3D, although the overall quality is excellent, a few bad elements may appear (less than 0.01% of the total number). These bad elements mostly appear close to the boundaries of the domain, and are very likely due to the fact that we explicitly do not move vertices of the boundaries. More particularly, bad elements appear for sensor u_5 with displacement $\mathbf{d}_4$. In this case the displacement is really complex, and maybe too far from the order 1 at which the analysis of Section 6.1 was carried on. Maybe higher order terms would help handling this case. However, it must be noted that the overall quality for this case is still very good.

	$\mathcal{H}_{ref}^{n+1}$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_1$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_2$	
Average quality	1.1070		1.1980		1.2420	
Worst quality	4.3239		12.362		171.16	
$1.00 < Q < 2.00$	26,818	99.52 %	25,976	96.97 %	25,943	96.44 %
$2.00 < Q < 3.00$	112	0.42 %	638	2.38 %	680	2.53 %
$3.00 < Q < 4.00$	12	0.04 %	117	0.44 %	147	0.55 %
$4.00 < Q < 5.00$	4	0.01 %	36	0.13 %	50	0.19 %
$5.00 < Q < 10.00$	0	0.00 %	18	0.07 %	52	0.19 %
$10.00 < Q < 50.00$	0	0.00 %	3	0.01 %	26	0.10 %
$50.00 < Q < 100.00$	0	0.00 %	0	0.00 %	1	0.00 %
$100.00 < Q < 10000.00$	0	0.00 %	0	0.00 %	3	0.01 %

Table 6.8: Quality for sensor u_1^{n+1}: average and worst quality and quality histograms with the number of elements and the percentage for the mesh adapted directly $\mathcal{H}_{ref}^{n+1}$ and the meshes adapted with the ALE procedure $\mathcal{H}_{ALE}^{n+1}$ for the two displacements considered.

	$\mathcal{H}_{ref}^{n+1}$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_1$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_2$	
Average quality	1.1716		1.1121		1.1855	
Worst quality	9.3683		12.929		10.709	
$1.00 < Q < 2.00$	25,690	99.40 %	25,115	97.48 %	25,307	97.15 %
$2.00 < Q < 3.00$	131	0.51 %	512	1.99 %	581	2.23 %
$3.00 < Q < 4.00$	16	0.06 %	89	0.35 %	121	0.46 %
$4.00 < Q < 5.00$	5	0.02 %	29	0.11 %	29	0.11 %
$5.00 < Q < 10.00$	2	0.01 %	19	0.07 %	11	0.04 %
$10.00 < Q < 50.00$	0	0.00 %	1	0.00 %	1	0.00 %

Table 6.9: Quality for sensor u_2^{n+1}: average and worst quality and quality histograms with the number of elements and the percentage for the mesh adapted directly $\mathcal{H}_{ref}^{n+1}$ and the meshes adapted with the ALE procedure $\mathcal{H}_{ALE}^{n+1}$ for the two displacements considered.

	$\mathcal{H}_{ref}^{n+1}$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_1$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_2$	
Average quality	1.1243		1.3090		1.3589	
Worst quality	26.431		42.545		348.81	
$1.00 < Q < 2.00$	24,823	98.81 %	23,482	92.88 %	23,188	92.17 %
$2.00 < Q < 3.00$	265	1.05 %	1243	4.92 %	1318	5.24 %
$3.00 < Q < 4.00$	22	0.09 %	305	1.21 %	347	1.38 %
$4.00 < Q < 5.00$	7	0.03 %	127	0.50 %	122	0.48 %
$5.00 < Q < 10.00$	3	0.01 %	107	0.42 %	138	0.55 %
$10.00 < Q < 50.00$	2	0.01 %	19	0.08 %	44	0.17 %
$50.00 < Q < 100.00$	0	0.00 %	0	0.00 %	0	0.00 %
$100.00 < Q < 1000.00$	0	0.00 %	0	0.00 %	1	0.00 %

Table 6.10: Quality for sensor u_3^{n+1}: average and worst quality and quality histograms with the number of elements and the percentage for the mesh adapted directly $\mathcal{H}_{ref}^{n+1}$ and the meshes adapted with the ALE procedure $\mathcal{H}_{ALE}^{n+1}$ for the two displacements considered.

	$\mathcal{H}_{ref}^{n+1}$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_3$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_4$	
Average quality	1.3258		1.3296		1.3349	
Worst quality	9.4196		18.949		36.292	
$1.00 < Q < 2.00$	694,392	98.43 %	690,374	98.28 %	690,492	98.00 %
$2.00 < Q < 3.00$	10,537	1.49 %	11,615	1.65 %	13,113	1.86 %
$3.00 < Q < 4.00$	448	0.06 %	393	0.06 %	754	0.11 %
$4.00 < Q < 5.00$	44	0.01 %	39	0.01 %	118	0.02 %
$5.00 < Q < 10.00$	20	0.00 %	18	0.00 %	81	0.01 %
$10.00 < Q < 50.00$	0	0.00 %	1	0.00 %	23	0.00 %

Table 6.11: Quality for sensor u_4^{n+1}: average and worst quality and quality histograms with the number of elements and the percentage for the mesh adapted directly $\mathcal{H}_{ref}^{n+1}$ and the meshes adapted with the ALE procedure $\mathcal{H}_{ALE}^{n+1}$ for the two displacements considered.

	$\mathcal{H}_{ref}^{n+1}$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_3$		$\mathcal{H}_{ALE}^{n+1}$ with $\mathbf{d}_4$	
Average quality	1.4351		1.6582		1.6829	
Worst quality	21.034		62.108		192.48	
$1.00 < Q < 2.00$	696,611	94.39 %	605,999	81.66 %	596,375	80.20 %
$2.00 < Q < 3.00$	39,303	5.33 %	113,377	15.28 %	121,931	16.40 %
$3.00 < Q < 4.00$	1,752	0.24 %	1,7077	2.30 %	19,160	2.58 %
$4.00 < Q < 5.00$	234	0.03 %	3,704	0.50 %	4,204	0.57 %
$5.00 < Q < 10.00$	139	0.02 %	1,815	0.24 %	1,842	0.25 %
$10.00 < Q < 50.00$	7	0.00 %	87	0.01 %	55	0.01 %
$50.00 < Q < 100.00$	0	0.00 %	1	0.00 %	1	0.00 %
$100.00 < Q < 1000.00$	0	0.00 %	0	0.00 %	1	0.00 %

Table 6.12: Quality for sensor u_5^{n+1}: average and worst quality and quality histograms with the number of elements and the percentage for the mesh adapted directly $\mathcal{H}_{ref}^{n+1}$ and the meshes adapted with the ALE procedure $\mathcal{H}_{ALE}^{n+1}$ for the two displacements considered.

6.3 Update of the adaptation algorithm

The ALE metric was developed to be used in the unsteady algorithm described in Chapter 5. The simulation time interval is split into n_{adap} sub-intervals $[t_{i-1}, t_i]$ for $i = 1, .., n_{adap}$. On each sub-interval, the mesh remains "constant" - meaning there is no remeshing, but it is moved to follow the geometry displacement. At each time step of the sub-interval, we want the mesh to be moved to its adapted configuration. To do so, we are going to use the ALE metric defined in Section 6.1. In order to have a mesh adapted for each time of the sub-interval, we are going to average these ALE metrics in some way or another, similarly to what is done in the fixed-mesh case, to come up with an averaged Hessian-metric. To take into account the displacement at each time step, we have no other choice than applying the ALE correction to the Hessian matrices throughout the sub-interval before averaging them. From these modified Hessians, an ALE metric is computed so that the mesh generated for the beginning of a sub-interval is still adapted once moved with the elasticity displacement.

6.3.1 Error analysis

To derive the metric that will be used in the algorithm we take inspiration from the analysis of the error in the unsteady case with fixed meshes. First, the error minimization problem from Eq. (5.14) is modified to take into account the mesh movement. Let us consider sub-interval i $t_{i-1}, t_i]$. In the fixed-mesh analysis, we simply move the integral over time into the Trace to make the mean Hessian metric appear, and this results in an error $\mathbf{E}^i_{L^p}$ constant on the whole sub-interval. Here, we still want to use a mean Hessian metric, but we cannot move the integral over time because $\mathcal{M}\rangle(t)$ is not constant anymore. However, we make the hypothesis that, at first order, the mesh is constant and equal to the mesh at the beginning of the sub-interval: $\mathcal{M}^i(t) \approx \mathcal{M}^i(t_{i-1}) = \mathcal{M}^i$. With this approximation, the error on the sub-interval is constant:

$$\mathbf{E}^{ALE,i}_{L^p}(\mathbf{M}^{ALE,i}) = \int_\Omega \text{trace}\left((\mathcal{M}^{ALE,i})^{-\frac{1}{2}}(\mathbf{x})\,\mathbf{H}^{ALE,i}_u(\mathbf{x})\,(\mathcal{M}^{ALE,i})^{-\frac{1}{2}}(\mathbf{x})\right)^p \mathrm{d}\mathbf{x} \quad \text{such that} \quad \mathcal{C}(\mathbf{M}^{ALE,i}) = N^i\,, \tag{6.8}$$

where matrix $\mathbf{H}^{ALE,i}_u$ is the mean Hessian metric. This Hessian is built by averaging the modified Hessians H^* from Eq. (6.6):

$$\mathbf{H}^{ALE,i}_{L^1}(\mathbf{x}) = \int_{t_{i-1}}^{t_i} |{H^*}_u(\mathbf{x}, t)|\,\mathrm{d}t \quad \text{or} \quad \mathbf{H}^{ALE,i}_{L^\infty}(\mathbf{x}) = \Delta t_i \max_{t\in[t_{i-1},t_i]} |{H^*}_u(\mathbf{x}, t)|\,. \tag{6.9}$$

This way, the mesh generated at time t_{i-1} is adapted to the solution at any time $t > t_{i-1}$ within the subinterval once moved with the mesh deformation displacement. The optimization problem then becomes finding the mesh $\mathbf{M}^{ALE,i}$ that minimizes the error from Eq. (6.8).

This problem, under the previous hypothethis that $\mathcal{M}^{ALE,i}$ is constant, is solved exactly like the fixed mesh problem in Section 5.1.4. First a spatial optimization leads to:

$$\mathcal{M}^{ALE,i}_{L^p}(\mathbf{x}) = (N^i)^{\frac{2}{3}} \left(\int_\Omega (\det \mathbf{H}^{ALE,i}_u(\bar{\mathbf{x}}))^{\frac{p}{2p+3}} \mathrm{d}\bar{\mathbf{x}}\right)^{-\frac{2}{3}} (\det \mathbf{H}^{ALE,i}_u(\mathbf{x}))^{-\frac{1}{2p+3}}\, \mathbf{H}^{ALE,i}_u(\mathbf{x}).$$

The corresponding error is:

$$\mathbf{E}_{L^p}^{ALE,i}(\mathbf{M}_{L^p}^{ALE,i}) = 3^p\,(N^i)^{-\frac{2p}{3}}\left(\int_\Omega (\det \mathbf{H}_u^{ALE,i}(\mathbf{x}))^{\frac{p}{2p+3}}\mathrm{d}\mathbf{x}\right)^{\frac{2p+3}{3}} = 3^p\,(N^i)^{-\frac{2p}{3}}\,\mathcal{K}^{ALE,i}\,, \quad (6.10)$$

where $\mathcal{K}^{ALE,i} = \left(\int_\Omega \left(\det \mathbf{H}_u^{ALE,i}(\mathbf{x})\right)^{\frac{p}{2p+3}}\mathrm{d}\mathbf{x}\right)^{\frac{2p+3}{3}}$. The error being constant on the sub-intervals by hypothesis, the temporal optimization problem for the case where the time step τ is specified by a function of time reads:

$$\min_{\mathbf{M}^{ALE}} \mathbf{E}_{L^p}^{ALE}(\mathbf{M}^{ALE})^p = \sum_{i=1}^{n_{adap}} \mathbf{E}_{L^p}^{ALE,i}(\mathbf{M}_{L^p}^{ALE,i}) = 3^p \sum_{i=1}^{n_{adap}} (N^i)^{-\frac{2p}{3}}\,(\mathcal{K}^{ALE,i})^{\frac{2p+3}{3}}\,,$$

under constraint:

$$\sum_{i=1}^{n_{adap}} N^i \left(\int_{t_{i-1}}^{t_i} \tau(t)^{-1}\mathrm{d}t\right) = N_{st}\,.$$

Finally, proceeding as in Section 5.1.4, we write the optimal ALE metric for sub-intervals:

$$\begin{aligned}\mathcal{M}_{L^p}^{ALE,i}(\mathbf{x}) = N_{st}^{\frac{2}{3}}\Bigg(\sum_{i=1}^{n_{adap}} (\mathcal{K}^{ALE,i})^{\frac{3}{2p+3}}\Big(\int_{t_{i-1}}^{t_i} \tau(t)^{-1}\mathrm{d}t\Big)^{\frac{2p}{2p+3}}\Bigg)^{-\frac{2}{3}} \\ \Big(\int_{t_{i-1}}^{t_i} \tau(t)^{-1}\mathrm{d}t\Big)^{-\frac{2}{2p+3}} (\det \mathbf{H}_u^{ALE,i}(\mathbf{x}))^{-\frac{1}{2p+3}}\,\mathbf{H}_u^{ALE,i}(\mathbf{x})\,.\end{aligned} \quad (6.11)$$

Once again, we stress that preserving the number of degrees of freedom when moving the mesh is essential in the process.

6.3.2 Algorithm

Few things need to be modified to extend the fixed-point algorithm to moving mesh ALE simulations. The overall algorithm remains unchanged: the simulation time frame is still divided into sub-intervals, and a global fixed-point strategy is used to converge the meshes and the solutions. On each sub-interval, the mesh remains the same, in that no remeshing is performed. However, and it is the first difference, within a sub-interval, the mesh is moved using the algorithm presented in Part I Section 1.1. One or several mesh deformation steps are performed during a sub-interval. To compute the ALE metrics at each sampling time, we use the displacement given by the mesh deformation step (elasticity or IDW solution), without taking into account the optimizations. To justify this, we consider that the displacement due to the smoothing is small compared to the mesh deformation. Moreover, the displacement due to the smoothing cannot be predicted. The second difference concerns the computation of the metrics in the solver. The metric is no more associated to a fixed location in space, but it is rather attached to the moving vertices. Moreover, the ALE metric from (Eq. 6.11) must now be used. As in the standard algorithm, the average Hessian-metric is computed from the modified Hessian given by Relation 6.6 and averaged in L^1 norm (performing a sum of metrics rather than an intersection), and the average is performed on a sample of Hessians of the solution.

6.3.3 Update of the metric for optimizations

As regards the mesh optimization procedure, which is applied throughout the sub-interval time frame during the displacement of the mesh, special care must be given to the choice of the metric on which the computation of the quality is based. Indeed, the use of an incorrect metric will drive the optimizations to move vertices to a wrong location and to break the anisotropy, thus spoiling the adaptation of the mesh. At a given fixed-point iteration j, the metric corresponding to the current solution is not known, since the global normalization term cannot be computed before the end of the iteration. Thus, we have to use the ALE metric of the previous sub-interval. However, since the mesh is deformed throughout a sub-interval, the metric $\mathcal{M}_{L^p}^{ALE,i}$ that is used to generate the mesh at the beginning of the sub-interval at time t_{i-1} does not correspond to the mesh at time $t > t_{i-1}$.

In the theory developed in the previous sections, the ALE correction is applied on the Hessian matrices H before the normalization and not on the metrics $\mathcal{M}$. Therefore, from the ALE metric at the beginning of a sub-interval $[t_{i-1}, t_i]$ $\mathcal{M}_{L^p}^{ALE,i}(\mathbf{x})$, we cannot find directly $\mathcal{M}_{L^p}^{ALE}(\mathbf{x}(t))$ for $t \in [t_{i-1}, t_i]$ without knowing the modified Hessian $H*(t)$, which means without storing them. Two possibilities arise from this remark. First, several $\mathcal{M}_{L^p}^{ALE}(\mathbf{x}(t_j))$ could be pre-calculated at fixed times t_j, and then the metric at time t would be interpolated from two of these pre-calculated metrics. This method has two drawbacks: it requires storing the modified Hessian matrices $H*$ throughout the sub-intervals to be able to compute the pre-calculated metrics, which is costly in terms of memory and I/O, and the numerous interpolations are costly in terms of CPU time. The second possibility is to derive an approximate metric at time t from the ALE metric at the beginning of the sub-interval and the displacement. We chose this last possibility, and deform the metric together with the mesh.

Here, we propose to adopt the same reasoning as the one that lead to the ALE metric in Section 6.1, but on a reverse time interval. Let our sub-interval be $[t_{i-1}, t_i]$, t a time in this sub-interval and $\mathbf{d}$ the displacement field of the mesh between t_{i-1} and t. Our goal is to "move" the metric that is given at the beginning of the sub-interval t_{i-1} into a metric at t. The problem is reversed, and we seek for the metric at time t that will result in a mesh adapted to the metric at time t_{i-1} once moved with displacement $-\mathbf{d}$. At time t_{i-1}, the metric is $\mathcal{M}_{L^p}^{ALE,i}$ (the mesh at time t_{i-1} is generated directly using this metric). For easier notation, we set $t = t^{k+1}$ and $t_{i-1} = t^k$. Displacement $\mathbf{d}$ is the displacement of vertices between t^k and t^{k+1}: $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{d}(\mathbf{x}^k)$ and we write $\mathbf{d}'(\mathbf{x}^{k+1}) = \mathbf{x}^k - \mathbf{x}^{k+1} = -\mathbf{d}(\mathbf{x}^k)$. We define

$$\begin{array}{rccl} \boldsymbol{\phi}' : & \Omega^{k+1} & \longrightarrow & \Omega^k \\ & \mathbf{x}^{k+1} & \longmapsto & \mathbf{x}^k = \boldsymbol{\phi}'\left(\mathbf{x}^{k+1}\right) = \mathbf{x}^{k+1} + \mathbf{d}'(\mathbf{x}^{k+1})\,. \end{array} \tag{6.12}$$

Following [Alauzet 2011b], we consider an edge $\mathbf{e}^{k+1}$ at time t^{k+1}, and write that we want it to be adapted to metric $\mathcal{M}_{L^p}^{ALE,i}$ at time t^k.

$$\begin{aligned} 1 &= \left[\mathbf{e}^k\left(\boldsymbol{\phi}'(\mathbf{x}^{k+1})\right)\right]^T \cdot \mathcal{M}_{L^p}^{ALE,i}\left(\boldsymbol{\phi}'(\mathbf{x}^{k+1})\right) \cdot \mathbf{e}^k\left(\boldsymbol{\phi}'(\mathbf{x}^{k+1})\right) \\ &= \left(\left[\nabla^{k+1}\boldsymbol{\phi}'(\mathbf{x}^{k+1})\right]^T \cdot \mathbf{e}^{k+1}(\mathbf{x}^{k+1})\right)^T \cdot \mathcal{M}_{L^p}^{ALE,i}(\mathbf{x}^k) \cdot \left(\left[\nabla^{k+1}\boldsymbol{\phi}'(\mathbf{x}^{k+1})\right]^T \cdot \mathbf{e}^{k+1}(\mathbf{x}^{k+1})\right) \\ &= \left(\mathbf{e}^{k+1}(\mathbf{x}^{k+1})\right)^T \cdot \left\{\nabla^{k+1}\boldsymbol{\phi}'(\mathbf{x}^{k+1}) \cdot \mathcal{M}_{L^p}^{ALE,i}(\mathbf{x}^k) \cdot \nabla^{k+1}\boldsymbol{\phi}'^T(\mathbf{x}^{k+1})\right\} \cdot \left(\mathbf{e}^{k+1}(\mathbf{x}^{k+1})\right). \end{aligned}$$

In other words, the metric we are looking for at time t is:

$$\mathcal{M}_{L^p}^{optim,i}(\mathbf{x}(t)) = \nabla\boldsymbol{\phi'}\left(\mathbf{x}(t)\right) \cdot \mathcal{M}_{L^p}^{ALE,i}\left(\mathbf{x}(t_{i-1})\right) \cdot \nabla\boldsymbol{\phi'}^T\left(\mathbf{x}(t)\right) , \tag{6.13}$$

and we expect that this metric is close to $\mathcal{M}^{ALE}(\mathbf{x}(t))$.

This metric is easy to compute on the fly, because the central term $\mathcal{M}_{L^p}^{ALE,i}\left(\mathbf{x}(t_{i-1})\right)$ is known and the gradients are easy to compute, since the displacement considered in $\boldsymbol{\phi'}$ is the opposite of the current displacement of a vertex.

6.3.4 Handling of the surface

In the present algorithm, we stick to the choice presented in Part I, Section 1.1.5, and mentioned in the previous section. The boundary vertices are not moved on the surfaces, except in the specific case of planes orthogonal to the axes of the frame. This may result in a small loss of accuracy close to the surfaces, where one might want the adaptation band-shaped areas to move consistently in the volume and on the surface, however, since the boundary displacement is consistent from one fixed-point iteration to the other, the ALE metric should be able to handle it.

6.4 3D numerical examples

A few examples in 3D are given to validate our approach. The codes used are the same as those listed in the previous chapter (see Section 5.2.5), with the exception that the solver is used in its ALE version detailed in Chapter 2.

6.4.1 Shock tube in expansion

The first example is a variation of the classic Sod shock-tube problem, with a tube being homogeneously expanded in one direction. This *a priori* simple test case shows that the refined regions follow the physical phenomena of interest, and that the anisotropy is well preserved despite the mesh being moved.

The tube initial dimensions are $[0,1] \times [-0.15, 015] \times [0,1]$, and the gas is split in two regions: for $x \leq 0.5$ the state is $(\rho_{left}, \mathbf{u}_{left}, p_{left}) = (1, \mathbf{0}, 1)$ whereas for $x > 0.5$ the state is $(\rho_{right}, \mathbf{u}_{right}, p_{right}) = (0.125, \mathbf{0}, 1)$. The gas is then left to evolve freely, and the classic rarefaction wave, contact discontinuity and shock wave appear and evolve in the tube. The tube is expanded to the right with the following movement imposed to the mesh vertices:

$$\begin{cases} x(t) &= x_0 + 1.5x_0 \cdot t \\ y(t) &= y_0 \\ z(t) &= z_0 \,. \end{cases}$$

The simulation is run until time $t = 0.66$ so that the size of the tube doubles. Special transmitting conditions are prescribed in $x = 0$, so that the rarefaction wave can go out of the tube. For that case, 20 sub-intervals were prescribed, and 4 fixed-point iterations were performed. The

density field is used as sensor for the adaptation, and the error is controlled in L^2 norm. A simplified space-time complexity of $400,000$ vertices was prescribed (*i.e.* the sum of the numbers of vertices per interval), and the meshes have an average size of $21,000$ vertices.

The meshes at different time steps and the corresponding solution are shown in Figure 6.12. Not only do the waves evolve in the refined bands, but those bands also move as the tube is expanded. The adapted regions are transported together with the mesh in accordance with our algorithm, that associates metrics to a moving vertex rather than to a fixed position in space. In Figure 6.14, we zoom on the solution at the beginning and the end of the eighth sub-interval to stress the evolution of the physical phenomena inside an adaptation band during a sub-interval. In Figure 6.13, we show the mesh at the beginning and at the end of the last sub-interval, to emphasize the movement of the adapted regions together with the movement of the tube boundaries, preserving the anisotropy.

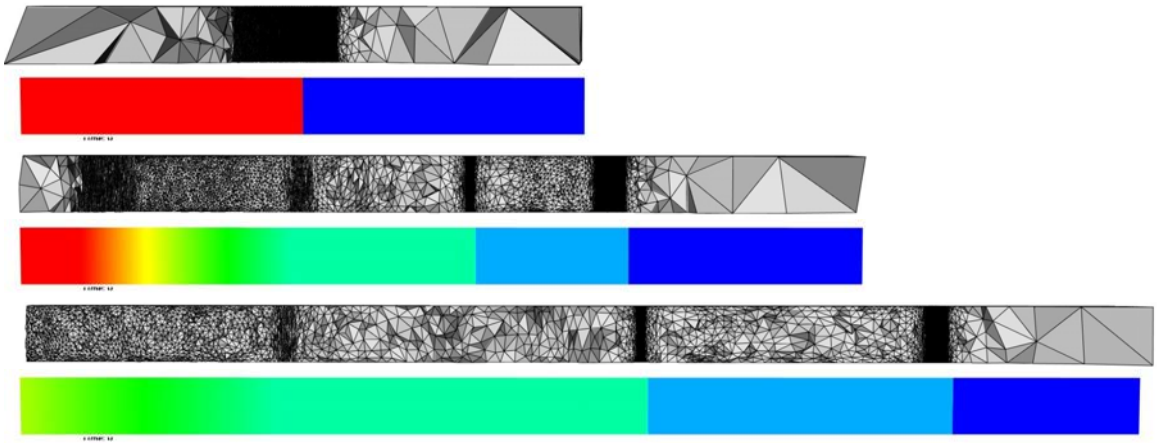

Figure 6.12: Adapted meshes and density solutions for the expanding shock tube case at time 0, 0.5 and 0.66. Cuts into the volume mesh are made along the plane $y = 0$.

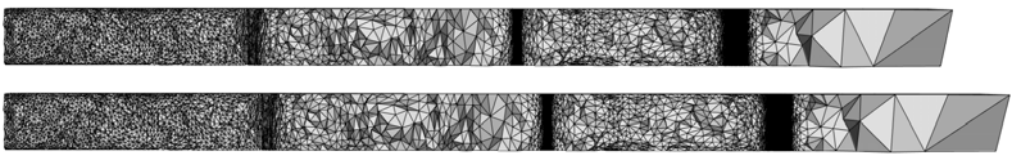

Figure 6.13: Mesh at the beginning (top) and end (bottom) of the eighth sub-interval. One can see the bands are move forward as the tube is being expanded.

6.4.2 Moving ball in a shock tube

In the second example, we take the same shock tube as previously, but we add a moving ball inside the tube instead of the expansion, that interacts with the shock and the contact discontinuity.

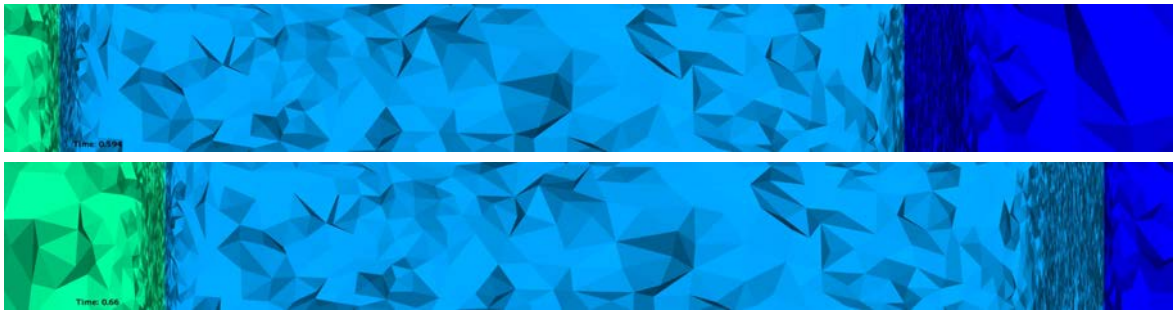

Figure 6.14: Zoom on the mesh and solution at the beginning (top) and at the end (bottom) of the last sub-interval. Once can see the two waves move forward in the adapted bands.

More precisely, the dimensions of the tube are: $[0,1] \times [-0.15, 0.15] \times [-0.1, 0.1]$. At initial time, the gas is split in two states: for $x \leq 0.5$ the state is $(\rho_{left}, \mathbf{u}_{left}, p_{left}) = (1, \mathbf{0}, 1)$ whereas for $x > 0.5$ the state is $(\rho_{right}, \mathbf{u}_{right}, p_{right}) = (0.125, \mathbf{0}, 1)$. A ball of radius $r = 0.02$ is immersed in the gas, its center being in position $(0.75, 0, 0)$. The tube is fixed, while the ball has a constant speed $\mathbf{v}_{ball} = -0.3\ \mathbf{e_x}$. The simulation is run until final time $t_f = 0.5$. After a while, the ball goes through the shock and the contact discontinuity, creating complex patterns both in front of it and in its wake. We expect all these physical phenomena to be captured correctly by the multiscale adaptation algorithm. For that case, 80 sub-intervals were prescribed, and 5 fixed-point iterations were run, and the density field is used as sensor for the adaptation. A simplified space-time complexity of $21,000,000$ vertices was prescribed, and the meshes have an average size of $260,000$ vertices with a size range going from $100,000$ to $450,000$ vertices.

The meshes at different time steps and the corresponding solution are shown in Figure 6.15. Since we are interested in the interaction of the ball with the shock and contact discontinuity, we only show the part $x > 0.5$ of the tube. The meshes shown are the ones at the end of the sub-intervals, *i.e.* the meshes that have been moved. The adaptation allows us to capture the two waves with a good accuracy all along the simulation, but also to recover some more complex features around the ball. It is interesting to note that the shock is perfectly meshed as soon as the ball has gone through it. A closer look at the surface of the ball also shows that it is adapted with highly anisotropic elements, and the anisotropy close to the surface is well preserved.

For this simulation, allowing mesh optimization (smoothing and edge/face swapping) is mandatory, otherwise, the shearing created by the advance of the ball in the volume mesh quickly creates invalid elements. Consequently, it is crucial to compute the quality criteria that triggers optimization with regards to the actual metric of the mesh. In this case, it is clear on the figures that the swaps did not break the anisotropy in the anisotropic adapted regions, although nearly 4.5 millions of swaps were performed during the last fixed-point iteration (around $55,000$ per sub-interval).

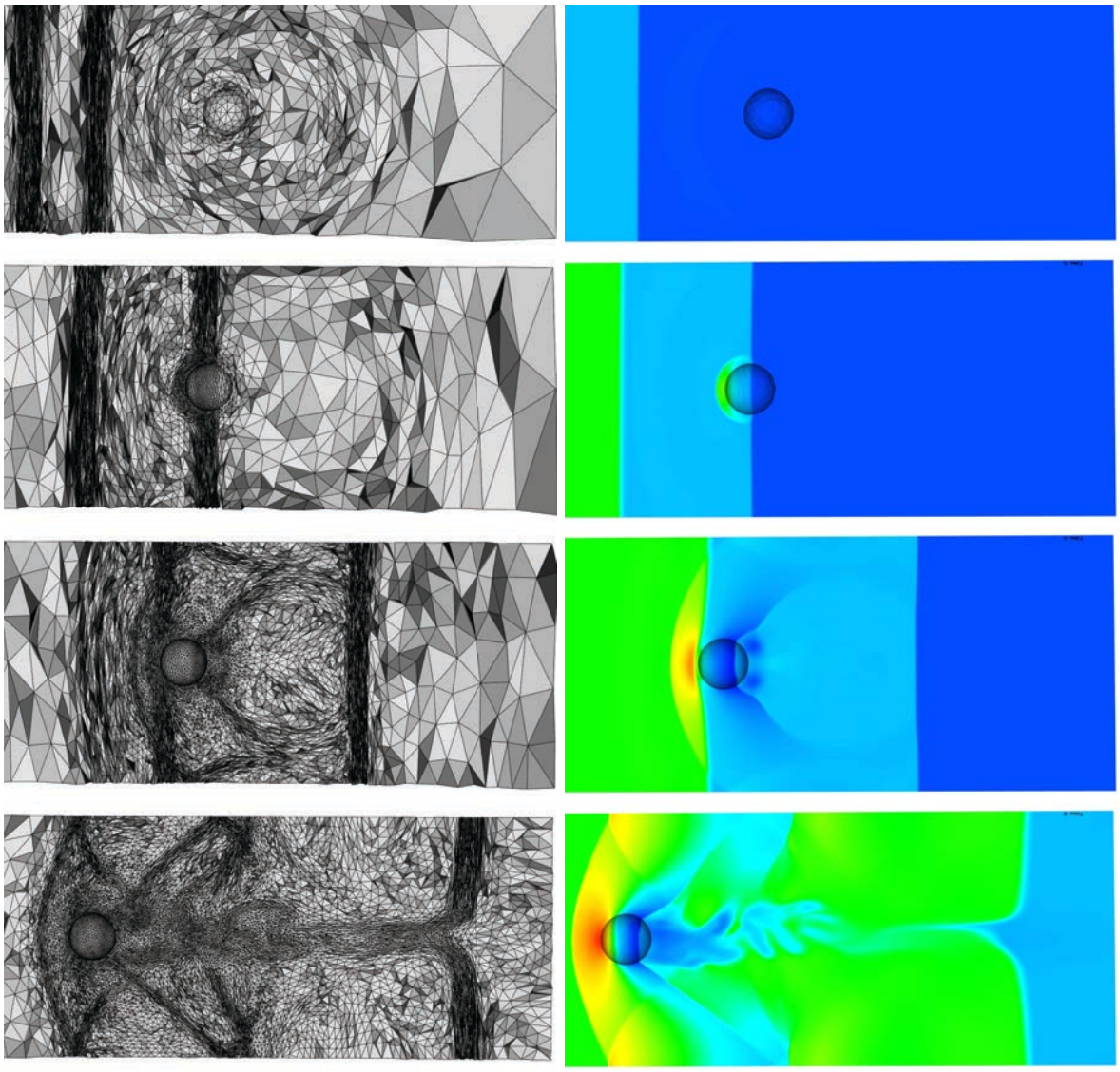

Figure 6.15: Adapted meshes and density solutions for the ball in a shock tube test case, at times 0.07, 0.12, 0.19 and 0.44. Cuts into the volume mesh are made along the plane $y = 0$.

6.4.3 Nosing-up f117

The following example has already been presented in Part I Section 3.2.2. We consider a F117 aircraft nosing up and down inside a susbsonic flow, creating vorticity in its wake.

The total simulation interval $t \in [0, 816]$ was divided into 96 adaptation sub-intervals, and 5 fixed-point iterations were used. The adaptation sensor is the local Mach number, and the error is controlled in L^2 norm. The number of vertices per mesh is $649,242$ on average, ranging from $100,000$ to $1,380,000$. The minimal tetrahedron height is $7e^{-4}$ at the beginning of the 66th sub-interval, where the number of vertices is maximal, while the aircraft is 230 units long.

Views of the adapted meshes and the corresponding solution are displayed in Fig. 6.16. In comparison with the simulation without adaptation, the vortical wake is propagated much farther from the plane, and the patterns of the vortices are more precisely captured. The observation of the meshes shows that the the meshes are actually refined only in the vicinity of the wake, and with such a precision that one can see the vortices being created, evolving and vanishing just by looking at the meshes. A view at several meshes within the same sub-interval shows that the mesh evolves continuously following the physical phenomena. In the wake far from the aircraft, the elements are highly anisotropic, whereas they have to be isotropic in the area of vorticity due to the characteristic of the physical phenomena (smaller size of the details).

6.4.4 Two F117 aircraft flight paths crossing

Finally, our last example was also presented in Part I Section 3.2.3. Two F117 aircraft have crossing flight paths, at a speed of Mach 0.4. This example was challenging in terms of moving mesh, due to the large displacement of the geometries and the shearing induced by the movement of the aircraft, and is all the more so with adapted meshes.

The adaptation parameters are the following: the adaptation is performed on the density field, the error is controlled in L^2 norm, the time interval is divided into 50 sub-intervals, 3 fixed-point iterations are performed, and a simplified space-time complexity of $38,000,000$ vertices is targeted. The meshes have between $360,000$ and $825,000$ vertices, for an average mesh size of $380,000$ vertices. The smallest height generated is $5.5\,e^{-4}$ at the beginning of the tenth sub-interval, for a domain of dimension $[-1800, 2400] \times [-800, 1000] \times [-1200, 1200]$.

The meshes and density solutions at different time steps are shown in Figure 6.18. The meshes are well adapted to the solution. Compared to the simulation without adaptation, the wakes of the F117s are captured far from the aircrafts, and their interactions with the waves are clearly visible. The anistropy of the meshes is clearly visible, in front of the aircrafts (corresponding to the acoustic waves), and in their wake, even if the anisotropic ratios are not so large, due to limitations in terms of computing power and time. On Figure 6.17, a close up on the two aircrafts is made, that shows the adapted surface mesh and solution.

6.5 Conclusion

In this section, we have recalled the formulation of an ALE metric, that is used to generate a mesh at a certain time t^n that will be adapted at a later time once moved with a certain displacement. We have verified this formulation on a variety of analytic cases, establishing that

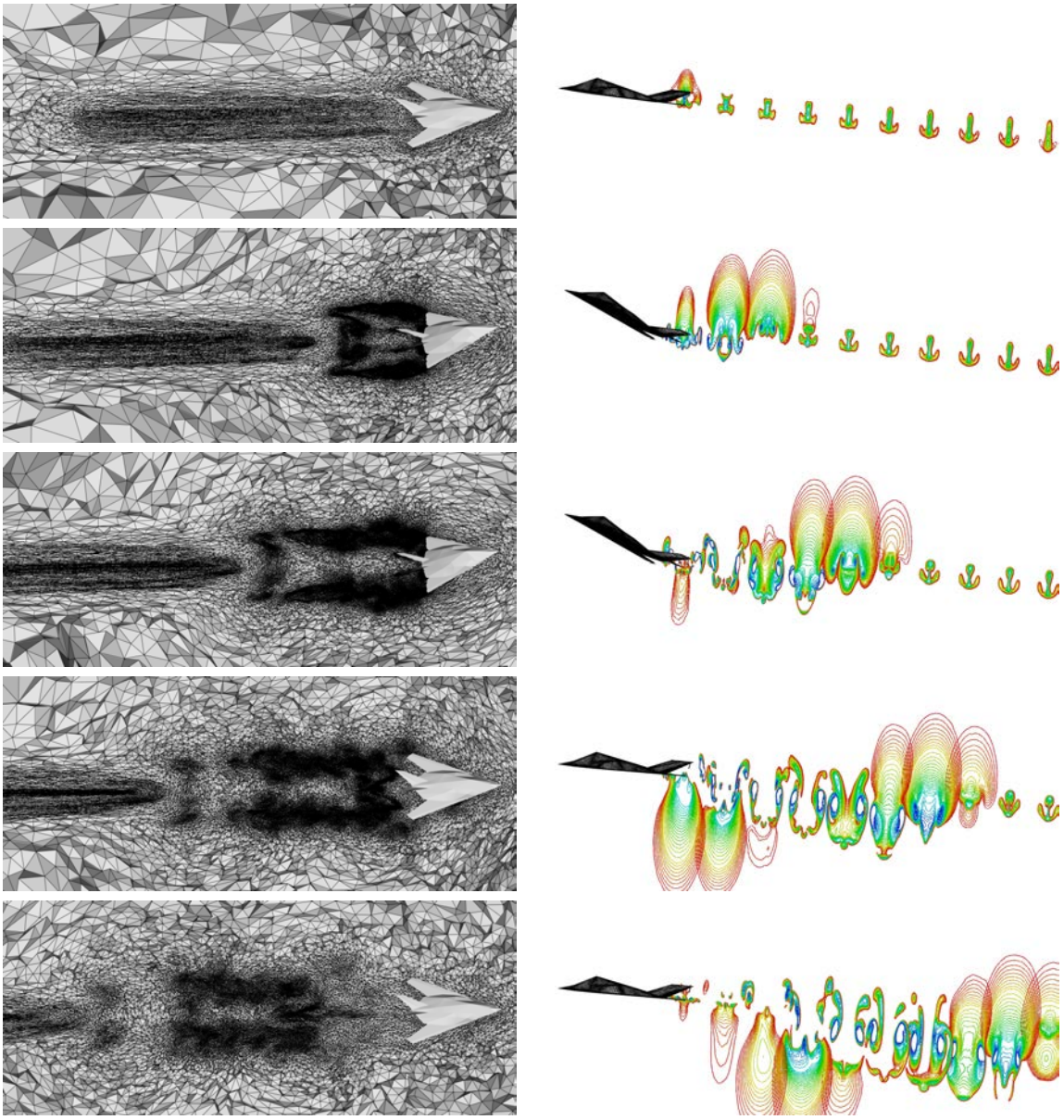

Figure 6.16: Nosing up F117 test case: adapted meshes (view from the top) and local Mach number isolines at different time steps.

the ALE metric results in meshes almost as good as the meshes adapted with the standard procedure, both in terms of error and quality. The error analysis from Chapter 5 has then been modified to take into account the mesh movement, resulting a slightly modified global fixed-point adaptation algorithm: a corrected Hessian-metric is computed instead of the regular Hessian-metric, and the mesh is moved with the connectivity-change moving mesh algorithm presented in Chapter 1. This algorithm was used to run adaptative simulations in 3D, for which the gain in terms of solution accuracy validates the chosen adaptation approach.

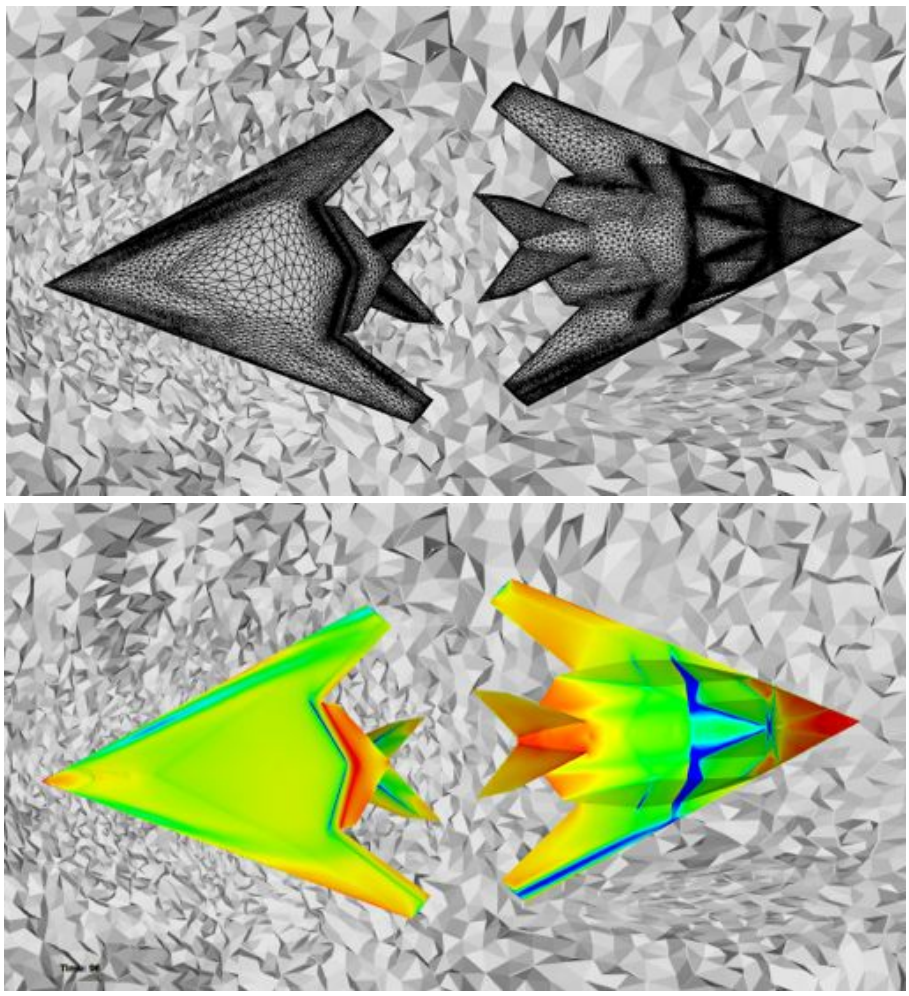

Figure 6.17: Two F117s test case : close-ups on the surface mesh of the aircraft and the corresponding solution.

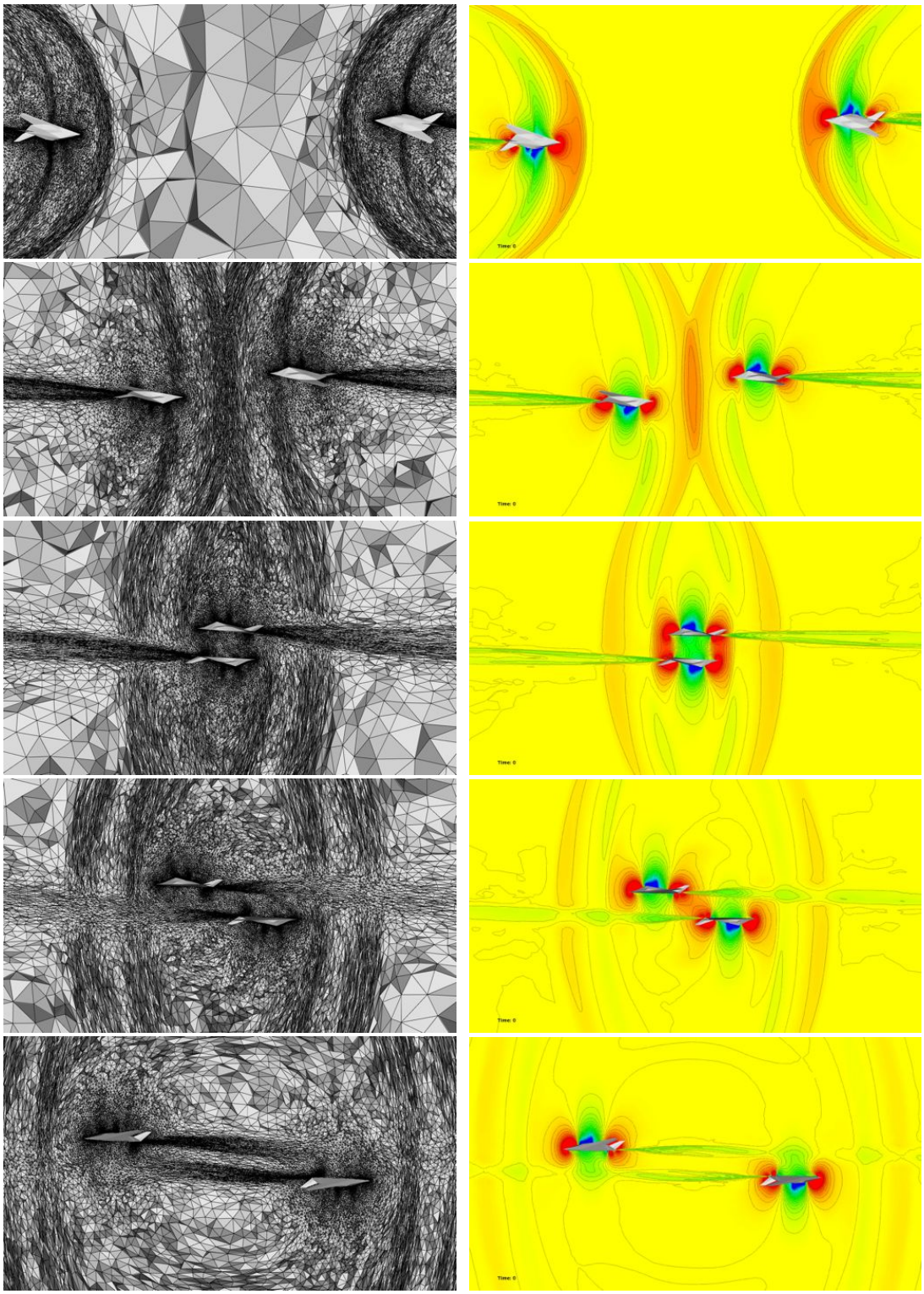

Figure 6.18: Two F117s test case: adapted meshes and density solutions at different time steps. Cuts into the volume mesh are made along the plane $y = 0$.

Conclusion

This part addressed metric-based mesh adaptation for time dependent simulations. In Chapter 4, a brief survey of mesh adaptation is presented, and all the concepts of metric-based adaptation on which this thesis is based are introduced, notably the concept of continuous mesh which enables new error analysis and optimization. This concept is used in Chapter 5 to derive an improved version of the unsteady fixed-point mesh adaptation algorithm. Notably, a global normalization term arises, which results in a global fixed-point algorithm. Two methods to compute a mean Hessian-metric on the adaptation sub-intervals are analyzed. An efficient parallelization of the algorithm was carried out, which makes it possible to run a number of unsteady 3D simulations with an accuracy unthinkable a few years ago, see [Alauzet 2015]. Finally, in Chapter 6, all the previous works converge to the design of a mesh adaptation algorithm for moving meshes. An ALE metric proposed in [Olivier 2011a] allows to generate meshes that will be adapted once moved with a certain displacement. Analytic examples validated this metric, and it was used within a new error analysis, to derive an ALE metric for adaptive sub-intervals. The regular time-dependent algorithm was modified to use this metric, and coupled with the moving mesh algorithm. Three dimensional adaptive simulations with moving boundaries are presented and analyzed, as proofs of concept of our approach.

The main contributions of this thesis with respect to pre-existing work in the GAMMA group are:

- A new formulation of the metric intersection was proposed.
- A new version of the global fixed-point unsteady mesh adaptation algorithm was used.
- Two methods to compute the average Hessian-metric were compared, in L^1 norm (sum of metrics) and L^∞ norm (intersection), leading to the use of the first instead of the latter that was used so far.
- An analysis of the parallel performance of the adaptation loop was carried out, resulting in improvements to the parallelization. Thanks to this parallelization, 3D unsteady adaptive simulations were run, with much larger mesh sizes than before.
- A study of the error on analytic cases for the ALE metric confirmed its adapted characteristic.
- A new error analysis was carried out for adapted moving meshes and the adaptation algorithm was deduced from it.
- An update strategy was proposed for the metric at mesh optimization times, to avoid spoiling the adaptation with optimizations.
- Several adaptive simulations with moving meshes were run in three dimensions.

Several perspective arise from this work, notably:

- To improve the adaptation algorithm, the next step is to control the temporal error. This would in particular lead to automatic adaptation of the size of the sub-intervals. This appears to be all the more important with moving meshes, where the size of the mesh is not even constant throughout a sub-interval.
- The convergence of this unsteady adaptative method needs to be studied, depending on the number of vertices and/or the size of the sub-intervals, as well as the influence of the initial solution on the final solution, and the ability of the algorithm to recover features of the solution lost due to coarse initial meshes.
- The metric used for optimizations during the adaptation loop should be analyzed in detail. Indeed, this step is crucial in the adaptation process, since it controls where the elements actually go.
- Vertices should be allowed to move on non plane surfaces. However, this is very complicated, since it requires reconstructing the surfaces with a good geometric accuracy to avoid introducing errors due to bad surface representation.
- At longer term, adaptative moving mesh methods mentioned in 4.1.3 appear as an interesting alternative to prescribe the displacement of the vertices, and possible interactions with metric based unsteady adaptation should be investigated.

Conclusion and perspectives

Conclusion

In this thesis, we have covered the whole chain of adaptive simulations with moving boundaries: moving mesh algorithms, ALE solvers and adaptation for unsteady flows have been studied, all finally converging to an adaptation algorithm for moving mesh simulations in three dimensions. The work presented here follows from [Olivier 2011a], and presented updates for most stages of the adaptation chain. It also presents a 3D implementation of our approach, demonstrating the efficiency of metric-based mesh adaptation for complex 3D adaptive time-dependent simulations with moving meshes involving large displacements.

The main improvement concerning moving meshes was the addition of a different mesh deformation method in the connectivity-change algorithm. The mesh deformation, *i.e.* the displacement of the vertices following the moving bodies, can be computed either via a PDE (with an elastic analogy) or via a direct interpolation method (the Inverse Distance Weighted method). Although a few differences appeared, both methods present generally excellent results, which proves the robustness of the overall moving mesh algorithm. Large displacement of the geometries can be handled efficiently without remeshing. So far, mostly rigid moving bodies had been considered. In this thesis, we addressed deformable geometries, on which the moving mesh algorithm works perfectly, and started to look into boundary layer meshes and large volume variations.

This moving mesh algorithm was devised to be used with an Arbitrary-Lagrangian-Eulerian solver, that was presented in detail. This solver is based on already existing schemes, but a choice was made among the numerous possibilities available in the literature to make it both robust and accurate. Spatial discretization mostly consisted in adding ALE velocity terms to the standard fixed mesh version of our in-house solver: second order spatial accuracy is reached thanks to an HLLC approximate Riemann solver and MUSCL-like gradient reconstruction. Regarding temporal discretization, an approach based on the Geometric Conservation Law allowed us to build high-order multi-step Runge-Kutta schemes. The computation of the geometric parameters required by these schemes was clarified in 3D. Swaps, that allow us to preserve a good mesh quality throughout the simulation, are handled with a linear interpolation between two solver time steps.

This solver was implemented in 3D and parallelized with a multi-thread paradigm. Several validation test cases were run, which confirmed the expected order of accuracy. Notably the influence of swaps, essential to the moving mesh algorithm, was studied and proved to be of little influence on the solution accuracy due to the limited number of such operations. Other 3D examples were run, involving complex geometries, displacements and flows, both with imposed motion or with Fluid-Structure Interaction. They prove the efficiency of the chosen approach, allowing us to run complex moving mesh simulations in a reasonable time, without ever remeshing.

Some progress had been made since the first unsteady fixed-point adaptation algorithm in [Alauzet 2003a]. In this thesis, a new version of the fixed-point algorithm is used, based on

a new analysis of the space-time error using the recent continuous mesh theory. Theory and experiment combined enabled to clarify certain points of the process, notably the choice of an averaged Hessian term essential in the algorithm. Thanks to an efficient parallelization, an analysis of which being provided, several 3D adaptative simulations were run with an accuracy unthinkable a few years ago.

Finally, all the bricks presented above are gathered and combined to result in an adaptation algorithm for unsteady simulations with moving meshes. An ALE metric had been presented in [Olivier 2011a], that makes it possible to move a mesh into an adapted mesh. This metric was plugged into an error analysis based on the continuous mesh model, and results in a correction to the optimal metric for the global fixed-point algorithm. Besides the use of this novel metric, a few updates were made to the fixed-mesh unsteady algorithm to handle moving meshes, notably the metric on which optimizations are performed had to be studied. Finally, several 3D simulations were run, that validate the chosen approach: adaptative 3D complex simulations with moving meshes can be run, bringing a noticeable gain in accuracy for a reasonable time.

Perspectives

This thesis is a small step in the field of mesh adaptation, but a wide variety of perspective stem from this work.

The unsteady fixed-point algorithm has proved to be remarkably efficient and flexible when it came to improve it with new theoretical results. Future work will certainly focus on improving the space-time error analysis with moving meshes, notably by introducing a better control of the temporal error. This would notably result in an automatic adaptation of the sizes of adaptation the sub-intervals of the fixed-point algorithm. More generally, the convergence of unsteady adaptative simulations must be addressed.

Concerning the solver, the handling of the swaps can also be improved, to make them totally transparent even on simulations where a large number of swaps would be required. Two trails are withing range: first, an ALE version of the operator was introduced in [Olivier 2011a] in 2D and could be extended to 3D, second, a conservative interpolation of swaps could be performed, using the interpolation procedure already used in the adaptation process. The first solution has the advantage of being fully ALE, but the second is much easier to design. Explicit solvers may not be optimal for some unsteady simulations, due to their limiting of the solver time step. Implicit solvers have been reported in the literature to improved the simulation cost and should be investigated.

Time consumption is a huge limitation in the size and the complexity of the problems considered. Mesh adaptation appears as an answer to this problem, by helping to reduce greatly the size of the meshes while preserving the solution accuracy. However, it is only really efficient if it is coupled to an efficient parallelization of the codes. In this thesis, only shared memory parallelization has been considered. In the medium term, it seems that considering other kinds of parallelization will be unavoidable: reflections on parallelization of meshing software on distributed memory were started in [Loseille 2013], and the parallelization of the fluid solver on GPU was started in this thesis (see Appendix B). Both approaches may have to be combined in order to take advantage of recent hybrid computation clusters.

The work presented in this thesis is an open gate to more complex problems. The ALE simulations that have been showed involve inviscid flows modeled by the Euler equations and rigid bodies. However, we have shown that moving deformable bodies with boundary layer meshes is within range, which opens the way to Navier-Stokes simulations. Already, a lot of ALE schemes are available for viscous simulations [Hay 2014], which facilitate the task, together with many works on adaptation for viscous flows [Menier 2015, Gammacurta 2010]. The FSI aspect can also be improved, by replacing the 6-DOF model by continuous elastic materials. The loose coupling could be replaced by more sophisticated strong coupling schemes. In this way, complex problems from aeroelasticity to biology could be run, in the interest of both industries and researchers.

Adaptative moving meshes are a very dynamic field of research, parallel to metric based adaptation. By coupling strongly the physical equations and the mesh movement, the vertices are moved at the same time as the solution to keep the meshes adapted at any time. These methods get rid of interpolation issues, with meshes of sensibly smaller size than those generated with the fixed point algorithm. However, most research on this subject has been so far limited to simple (convex) 2D geometries. Both approaches could be combined, for instance moving mesh equations providing the mesh deformation in the current fixed-point adaptation algorithm. During this thesis, preliminary works have been carried out to study Monge-Ampère equation [Chacón 2011] or a modified elasticity problem [Arpaia 2015] as moving mesh equations, and opened interesting perspectives.

Another promising research perspective is goal-oriented adaptative simulations, that reduce the size of the meshes by only meshing the features useful for the computation of a certain quantity. Goal-oriented metric based mesh adaptation has produced interesting results for steady problems [Loseille 2010a, Power 2006] and has been extended to unsteady ones [Belme 2011]. The latter work could be used as a starting point to derive a goal oriented adaptation strategy for moving meshes. Currently, one of the main bottleneck is the backward computation of the solution required by adjoint schemes, which requires to store the positions of the mesh at every time step. If an efficient strategy was found, the interest of industry would definitely be aroused, notably for Fluid-Structure Interaction problems.

Finally, the high-order elements have been used for a long while, mostly in Finite Element Methods (FEM), but only by increasing the order of the approximation polynomials. It has not been followed by meshing software, who have been generating flat pseudo high-order elements until very recently, *i.e.* standard $\mathbb{P}^1$ elements with more nodes on the edges. The generation of really high-order meshes is only in its earliest infancy. Lately, progress has been made on the generation of $\mathbb{P}^2$ curved meshes [Johnen 2013, Abgrall 2014], but theoretic results show the limitations of such approaches for too high orders for a whole mesh [George 2015b, George 2015a]. However, a smart use of curved meshes seems to be promising, notably by enabling a much more accurate representation of curved boundaries. The extension of metric-based mesh adaptation to such meshes has not been undertaken yet, let alone for moving boundary problems, but it seems to be an exciting field of research, full of new problems to solve.

Acknowledgments

This thesis was partially funded by a grant from the Airbus Group Foundation.

Appendix

Appendix A

Intersection of metrics

In Part II, Section 4.2.3, a new way to compute the intersection of two metrics is presented. It is based on simultaneous reduction and successive changes of basis. In the solver code, metrics are intersected on the fly. More precisely, Hessian-metrics are intersected, *i.e.* absolute values of Hessians, whose eigenvalues will be truncated later to make real metrics.

Hessian matrices can be degenerate quadratic forms, *i.e.* they can have null eigenvalues. When one eigenvalue is null or close to null (or more than one), the intersection procedure from Section 4.2.3 does not behave correctly. Specific treatments are thus required in those cases. In the following sections, we present what is done in each case.

Most of the times, the technique relies on the geometric interpretation of a metric: a metric can be represented by an ellipsoid whose axes are the eigenvectors and the sizes the axes the inverse of the eigenvalues. Degenerate (null) eigenvalues correspond to infinite sizes in the corresponding directions, leading to geometric representations such as cylinders, parallel lines or planes, or the whole space.

A.1 In two dimensions

Let us consider two real symmetric positive matrices:

$$\mathcal{M}_1 = \begin{pmatrix} \mathbf{u_1} & \mathbf{v_1} \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \mu_1 \end{pmatrix} \begin{pmatrix} \mathbf{u_1} & \cdots \\ \mathbf{v_1} & \cdots \end{pmatrix},$$

and

$$\mathcal{M}_2 = \begin{pmatrix} \mathbf{u_2} & \mathbf{v_2} \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \lambda_2 & 0 \\ 0 & \mu_2 \end{pmatrix} \begin{pmatrix} \mathbf{u_2} & \cdots \\ \mathbf{v_2} & \cdots \end{pmatrix},$$

where $\mathbf{u_1}$, $\mathbf{v_1}$, $\mathbf{u_2}$, $\mathbf{v_2}$ are the eigenvectors, and λ_1, μ_1, λ_2, μ_2 the eigenvalues. This decomposition is always possible because the matrices are real and symmetric, but the eigenvalues can be null. The eigenvectors of a matrix are considered normalized and orthogonal in twos.

The following **degenerate** cases can be distinguished:

1. If both matrices are null, the matrix of the intersection is null.
2. If one of the matrices is null, the intersection is the other matrix. (The non null metric is necessarily included in the null metric that has infinite sizes in both directions.)
3. If $\mathcal{M}_1$ has one null eigenvalue and not $\mathcal{M}_2$: one cannot take the inverse of the square root of $\mathcal{M}_1$, thus $\mathcal{M}_1$ and $\mathcal{M}_2$ are swapped, and the standard procedure is applied.

4. If $\mathcal{M}_2$ has one null eigenvalue and not $\mathcal{M}_1$: the standard procedure is applied without problem.

5. If $\mathcal{M}_1$ and $\mathcal{M}_2$ each have one and only one null eigenvalue (let us suppose $\mu_1 = \mu_2 = 0$), two cases are distinguished depending on whether the degenerate axis is the same or not.

   (a) If $\mathbf{v_1}$ and $\mathbf{v_2}$ are collinear, *i.e* if one metric is included in the other, the intersection is degenerated in this direction. The other direction is orthogonal and its size is given by the maximum of the non null eigenvalues.

   $$\mathcal{M}_{1\cap 2} = \mathcal{P} \begin{pmatrix} \max(\lambda_1, \lambda_2) & 0 \\ 0 & 0 \end{pmatrix} \mathcal{P}^T ,$$

   where $\mathcal{P}$ is the matrix made up with $\mathbf{u_1}$ et $\mathbf{v_1}$.

   (b) If the degenerate directions are different, the intersection is build considering the two degenerate axes: let us move to basis $\mathcal{B} = (\mathbf{v_1}, \mathbf{v_2})$, made up with the two degenerate directions. If $\mathcal{M}_1$ and $\mathcal{M}_2$ are written in this basis, writing $\mathcal{P}$ the matrix whose columns are $\mathbf{v_1}$ and $\mathbf{v_2}$:

   $$\mathcal{P}^T\mathcal{M}_1\mathcal{P} = \begin{pmatrix} 0 & 0 \\ 0 & \mathbf{v_2}^T\mathcal{M}_1\mathbf{v_2} \end{pmatrix} \quad \text{and} \quad \mathcal{P}^T\mathcal{M}_2\mathcal{P} = \begin{pmatrix} \mathbf{v_1}^T\mathcal{M}_2\mathbf{v_1} & 0 \\ 0 & 0 \end{pmatrix} ,$$

   Hence the intersection:

   $$\mathcal{M}_{1\cap 2} = \mathcal{P}^{-1^T} \begin{pmatrix} \mathbf{v_1}^T\mathcal{M}_2\mathbf{v_1} & 0 \\ 0 & \mathbf{v_2}^T\mathcal{M}_1\mathbf{v_2} \end{pmatrix} \mathcal{P}^{-1} .$$

   If the diagonal terms are expanded, we find:

   $$\mathbf{v_1}^T\mathcal{M}_2\mathbf{v_1} = \lambda_2 < \mathbf{v_1}, \mathbf{u_2} >^2 \quad \text{et} \quad \mathbf{v_2}^T\mathcal{M}_1\mathbf{v_2} = \lambda_1 < \mathbf{v_2}, \mathbf{u_1} >^2 ,$$

   where $(\cdot, \cdot)$ is the canonical scalar product.

A.2 In three dimensions

There are more degenerate cases to handle in 3D and some are more complex. Geometrically speaking, a degenerate metric with one null eigenvalue is represented by a cylinder with an elliptic basis, and a degenerate metric with two null eigenvalues is represented by two parallel planes. As previously, let us write:

$$\mathcal{M}_1 = \begin{pmatrix} \mathbf{u_1} & \mathbf{v_1} & \mathbf{w_1} \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \mu_1 & 0 \\ 0 & 0 & \nu_1 \end{pmatrix} \begin{pmatrix} \mathbf{u_1} & \cdots & \cdots \\ \mathbf{v_1} & \cdots & \cdots \\ \mathbf{w_1} & \cdots & \cdots \end{pmatrix} ,$$

and

$$\mathcal{M}_2 = \begin{pmatrix} \mathbf{u_2} & \mathbf{v_2} & \mathbf{w_2} \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} \lambda_2 & 0 & 0 \\ 0 & \mu_2 & 0 \\ 0 & 0 & \nu_2 \end{pmatrix} \begin{pmatrix} \mathbf{u_2} & \cdots & \cdots \\ \mathbf{v_2} & \cdots & \cdots \\ \mathbf{w_2} & \cdots & \cdots \end{pmatrix} .$$

1. If (at least) one of the matrices is null or if (at least) one the matrices is definite positive, the procedure defined in 2D is applied.

2. If each metric has two null eigenvalues (let us suppose $\mu_1 = \nu_1 = \mu_2 = \nu_2 = 0$), two cases are distinguished depending on whether the planes corresponding to the two metrics are parallel or not. $\mathbf{u_1}$ and $\mathbf{u_2}$ are the eigenvectors corresponding to the non null eigenvalue (and are thus vectors normal to the planes of the degenerate directions). Let $\mathbf{u} = \mathbf{u_1} \wedge \mathbf{u_2}$.

   (a) If $\mathbf{u} = 0$, *i.e.* if the planes are parallel, the intersection is made of the pair of planes with the smallest gap between them :

   $$\mathcal{M}_{1\cap 2} = \mathcal{P} \begin{pmatrix} \max(\lambda_1, \lambda_2) & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathcal{P}^T ,$$

   where $\mathcal{P}$ is made up with $\mathbf{u_1}$, $\mathbf{v_1}$ and $\mathbf{w_1}$.

   (b) If $\mathbf{u} \neq 0$, the planes are intersected, but direction $\mathbf{u}$ is still infinite (a straight line along $\mathbf{u} = \mathbf{u_1} \wedge \mathbf{u_2}$ cannot intersect either $\mathcal{M}_1$ or $\mathcal{M}_2$ by definition). The intersection is represented by a cylinder with an elliptic basis. To find this ellipse, a 2D basis is build with vectors included in the degenerate directions of $\mathcal{M}_1$ and $\mathcal{M}_2$: $\mathbf{e_1} = \mathbf{u_1^{\perp}}$ and $\mathbf{e_2} = \mathbf{u_2^{\perp}}$ with $\mathbf{e_1}, \mathbf{e_2} \in \text{vect}(\mathbf{u_1}, \mathbf{u_2})$. $\mathcal{M}_1$ and $\mathcal{M}_2$ are written in this basis:

   $$\overline{\mathcal{M}_1} = \begin{pmatrix} \mathbf{e_1}^T \mathcal{M}_1 \mathbf{e_1} & \mathbf{e_2}^T \mathcal{M}_1 \mathbf{e_1} \\ \mathbf{e_1}^T \mathcal{M}_1 \mathbf{e_2} & \mathbf{e_2}^T \mathcal{M}_1 \mathbf{e_2} \end{pmatrix} \quad \text{and} \quad \overline{\mathcal{M}_2} = \begin{pmatrix} \mathbf{e_1}^T \mathcal{M}_2 \mathbf{e_1} & \mathbf{e_2}^T \mathcal{M}_2 \mathbf{e_1} \\ \mathbf{e_1}^T \mathcal{M}_2 \mathbf{e_2} & \mathbf{e_2}^T \mathcal{M}_2 \mathbf{e_2} \end{pmatrix} .$$

   So, considering that $\mathbf{e_1}$ and $\mathbf{e_2}$ belong to the degenerate directions:

   $$\overline{\mathcal{M}_1} = \begin{pmatrix} 0 & 0 \\ 0 & \mathbf{e_2}^T \mathcal{M}_1 \mathbf{e_2} \end{pmatrix} \quad \text{and} \quad \overline{\mathcal{M}_2} = \begin{pmatrix} \mathbf{e_1}^T \mathcal{M}_2 \mathbf{e_1} & 0 \\ 0 & 0 \end{pmatrix} .$$

   So, in this basis:

   $$\overline{\mathcal{M}_{1\cap 2}} = \begin{pmatrix} \mathbf{e_1}^T \mathcal{M}_2 \mathbf{e_1} & 0 \\ 0 & \mathbf{e_2}^T \mathcal{M}_1 \mathbf{e_2} \end{pmatrix} .$$

   Then, to move back to 3D, a classic change of basis is performed:

   $$\mathcal{M}_{1\cap 2} = \mathcal{P}^{-1^T} \begin{pmatrix} \mathbf{e_1}^T \mathcal{M}_2 \mathbf{e_1} & 0 & 0 \\ 0 & \mathbf{e_2}^T \mathcal{M}_1 \mathbf{e_2} & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathcal{P}^{-1} ,$$

   where $\mathcal{P}$ is the matrix made up with vectors $\mathbf{e_1}$, $\mathbf{e_2}$ et $\mathbf{u}$.

3. If one of the two metrics has two null eigenvalues and the other has only one null eigenvalue, intersecting the metrics comes to intersecting two parallel planes with a cylinder. Let us suppose $\mu_1 = \nu_1 = \nu_2 = 0$. $\mathbf{u_1}$ is normal to the degenerate planes of $\mathcal{M}_1$, and $\mathbf{w_2}$ is the infinite direction of the cylinder of $\mathcal{M}_2$.

(a) If $\mathbf{u_1}$ and $\mathbf{w_2}$ are orthogonal, *i.e.* if the cylinder is parallel to the planes, the intersection is a cylinder of direction $\mathbf{w_2}$. To find its basis, we move to a cutting plane orthogonal to $\mathbf{w_2}$, and it comes up to the 2D case 4.
As previously, we write $\mathcal{M}_1$ et $\mathcal{M}_2$ in basis $(\mathbf{u_2}, \mathbf{v_2})$:

$$\overline{\mathcal{M}_1} = \begin{pmatrix} \mathbf{u_2}^T\mathcal{M}_1\mathbf{u_2} & \mathbf{v_2}^T\mathcal{M}_1\mathbf{u_2} \\ \mathbf{u_2}^T\mathcal{M}_1\mathbf{v_2} & \mathbf{v_2}^T\mathcal{M}_1\mathbf{v_2} \end{pmatrix} \quad \text{and} \quad \overline{\mathcal{M}_2} = \begin{pmatrix} \mathbf{u_2}^T\mathcal{M}_2\mathbf{u_2} & \mathbf{v_2}^T\mathcal{M}_2\mathbf{u_2} \\ \mathbf{u_2}^T\mathcal{M}_2\mathbf{v_2} & \mathbf{v_2}^T\mathcal{M}_2\mathbf{v_2} \end{pmatrix}.$$

In this 2D basis, $\overline{\mathcal{M}_2}$ is not degenerate, so the 2D standard procedure can be applied.

(b) If $\mathbf{u_1}$ and $\mathbf{w_2}$ are not orthogonal, the intersection is not degenerate, and it is built by taking the degenerate directions of the metrics.

$$\mathcal{M}_{1\cap 2} = \mathcal{P}^{-1^T} \begin{pmatrix} \mathbf{v_1}^T\mathcal{M}_2\mathbf{v_1} & 0 & 0 \\ 0 & \mathbf{w_1}^T\mathcal{M}_2\mathbf{w_1} & 0 \\ 0 & 0 & \mathbf{w_2}^T\mathcal{M}_2\mathbf{w_2} \end{pmatrix} \mathcal{P}^{-1},$$

where $\mathcal{P}$ is made up of eigenvectors $\mathbf{v_1}$, $\mathbf{w_1}$ and $\mathbf{w_2}$ (see 2D case 5b).
Alternatively, one can take the basis of $\mathcal{M}_2$, and truncate it in the degenerate direction:

$$\mathcal{M}_{1\cap 2} = \mathcal{P}^{-1^T} \begin{pmatrix} \lambda_2 & 0 & 0 \\ 0 & \mu_2 & 0 \\ 0 & 0 & \xi \end{pmatrix} \mathcal{P}^{-1},$$

where $\mathcal{P}$ is made up of eigenvectors $\mathbf{u_2}$, $\mathbf{v_2}$ and $\mathbf{w_2}$ and $\xi = \frac{\lambda_1}{<\mathbf{u_1},\mathbf{w_2}>^2}$.

4. If both metrics have one and only one null eigenvalue ($\nu_1 = \nu_2 = 0$), we intersect cylinders with elliptic bases.

(a) If the degenerate directions are the same, the intersection is represented by a cylinder. Finding its basis comes to finding the intersection of two 2D non degenerate metrics. The same procedure as in the 3D case 3a is applied.

(b) If the degenerate directions are not the same, the metric can be expressed simply in basis $(\mathbf{w_1}, \mathbf{w_2}, \mathbf{w} = \mathbf{w_1} \wedge \mathbf{w_2})$.

$$\mathcal{M}_{1\cap 2} = \mathcal{P}^{-1^T} \begin{pmatrix} \mathbf{w_1}^T\mathcal{M}_2\mathbf{w_1} & 0 & 0 \\ 0 & \mathbf{w_2}^T\mathcal{M}_1\mathbf{w_2} & 0 \\ 0 & 0 & \max(\mathbf{w}^T\mathcal{M}_1\mathbf{w}, \mathbf{w}^T\mathcal{M}_2\mathbf{w}) \end{pmatrix} \mathcal{P}^{-1},$$

where $\mathcal{P}$ is the matrix made up with vectors $\mathbf{w_1}$, $\mathbf{w_2}$ and $\mathbf{w}$.

Remark 6. *If these procedures are applied, one is sure to have a resulting metric included in the metrics to be intersected, and the degenerate cases are no more blocking points for the code. However, it remains to be proved that the resulting metrics are indeed of maximal volume, which is not obvious, especially in some 3D cases.*

Appendix B

GPU acceleration of a 3D Finite Volume flow solver

The use of graphic boards (Graphics Process Units, ou GPU) appears as a salutary perspective to match the ever increasing need for computational power that classic process units have difficulties matching, notably due to the mediocre energetic efficiency. The specific architecture of GPUs is supposed to allow them to reach an energetic efficiency far above the one of classic CPUs. During this thesis, as part of technological surveillance, we have tried to explore this trail. Many examples of the performances of GPU computations are available in the literature, however, the examples of implementation of complex software are very scarce. We have decided to progressively implement on GPU our in-house solver `Wolf` and to compare the performances of both GPU and CPU versions. In this study, we took particular care to give meaning to timings, comparing the GPU code to a CPU code that we believe to be efficient and on comparable machines.

Note. This study was carried out in 2012 at the beginning of this thesis. Since then, the performance of computation dedicated machines has evolved, but the main lessons are still valid.

B.1 Solver used

We have parallelized a simplified version of our in-house solver `Wolf`, which is described in great detail in its ALE version in Chapter 2.

Simplified version for GPU

The parallelization of such a code raises a lot of algorithmic difficulties, which is why we consider a reduced version of the solver in a first phase. We consider the pseudo-steady solver of the compressible Euler equations. We have only kept the HLLC solver of order 1, with an explicit advance in time using an order 2 Runge-Kutta with 2 steps (RK 2-2). The computation of the MUSCL reconstructed gradients presents a lot of difficulties that will be dealt with later. Local time-stepping is kept as well to avoid artificially slowing down convergence.

A module was added to the code in which this simplified version is coded. This module only uses the initialization routines of the standard code (I/Os and allocation and initialization of data structures) and is totally autonomous after that. This module contains side by side the CPU and GPU functions, ensuring that the code is the same in both cases so that the output is the same.

B.2 GPU acceleration

Architecture of a GPU

Modern GPUs all respect the CUDA architecture, on which is also based the OpenCL model. This architecture is still heavily influenced by the graphic usage of these boards, the prime objective being to be able to run massively parallel simple computations.

Modern GPU architecture is based on two points: massive multicore and a reduced latency. A GPU is made up of a large number of cores (from a few tenths to a thousand or more) that are relatively slow, and that are organized in 3D groups, each group having one control unit for all its cores. This way, each of the cores of a group processes exactly the same task in parallel, possibly on different data (Single Instruction Multiple Data programming model). These control units have a very small latency compared to the one of a classic CPU, which enables running a large number of threads on each group. There are several levels of shared memory, a registry for each core, a shared memory for each group, possibly shared memories for groups of groups, a global memory for the whole board and several levels of cache. The distribution of the threads on the groups of cores and the transfers between the different levels of memory are explicitly handled by the programmer. These main ideas are shared by CUDA and OpenCL, the differences between both lying in the vocabulary and in details.

Programming model

A good understanding of the architecture of GPUs is mandatory to write appropriate code. The principle is to run a simple task on big data. The simplest and most typical example is the addition of two big arrays: two arrays of size N have to be added term by term, which requires N operations. These N operations are divided into groups of N_t threads, N_t being set by the programmer. The task of each of these threads is to add one element of each array, and the groups of threads are executed successively by the groups of cores of the GPU (all at once if N_t is smaller than the number of cores by group, or some after the others else). Since a group of cores has only one control unit, the threads running in parallel on a group all execute exactly the same instruction at the same time.

The work of a programmer on GPU is to parallelize operations executed on big data, and to remove all that would make two threads different (conditions on the index of an element in an array for instance). Data structures that enable unconditional linear processing must be set up.

An important characteristic of GPUs is that access to their central memory of the CPU is very slow, so trying to avoid memory exchanges between the CPU and the GPU is one of the central axes of GPU programming. However, the memory on classic GPUs is relatively small, so algorithms reducing the use of memory on the GPU and the communications with the central memory have to be found. On top of this, cache misses and indirections also slow down considerably a GPU code, for the reasons explained above.

OpenCL

Two GPU development platforms coexist: CUDA and OpenCL. Our choice immediately fell on OpenCL due to its portability. Not only is it compatible with most GPU manufacturers, but it also addresses all kinds of generic parallel architecture, including usual multicore CPUs and coming hybrid processors.

OpenCL is more precisely a set of specifications: both for the language on the GPUs and for a library interfacing CPUs and GPUs. An OpenCL code is always included in a standard code on CPU, the "host" program. This host code is responsible for executing OpenCL functions that handle the interface with the GPU: compilation of GPU code, data exchange and trigger of the GPU code execution. The GPU code itself is defined in special functions known as kernels: a kernel is a standard C function, but it is compiled on the fly and is aimed at being executing on one or several groups of cores. A kernel is coded in C99, enhanced with a set of commands facilitating the use of many cores. An instance of a kernel is contained in a thread, the threads are gathered into workgroups, the workgroups are assigned to one or several work units (*i.e.* group of cores with a single control unit). In what follows, workgroup will both designate the pool of threads and the corresponding cores. For more details on the architecture and the API, see the official documentation [Munshi 2008].

B.3 State of the art

The use of GPUs in scientific computing is fairly recent, and the first serious attempts date back to around 2005. It is a novel field of research, with a lot of publications, but no reference has really imposed itself. Many research labs began porting very different codes on GPUs, with very different approaches and variable performances. Most article on this topic focus on the performance optimization of generic algorithms, without context. They are not about implementing simulation codes, but about trying to get the best out of available GPU devices, using generic approaches [Appleyard 2011].

A trend tends to stand out from this abundance of works: the implementation of solvers on structured grids. Structured grids perfectly correspond to the material architecture of GPUs, and the transcript of structured codes is both easy and efficient. Finite Difference methods in particular lead to remarkable performance. Other methods such as Lattice-Boltzmann methods can also be well parallelized. More generally, GPU acceleration seems to be adapted to methods with structured grids and/or of order 1.

There are very few available articles on the parallelization of bigger codes, such as the one with which we work. Many documents are available on the web, presenting on-going work, but not many present finished works. Two articles are representative of the literature, in which the authors adopt the same approach as ours: start from a complex CFD code and simplify it to parallelize it. In [Kuo 2011], a structured solver is considered, allowing the authors to reach high speed up factors, but the details of the GPU implementation are omitted, and the authors provide us with no way to compare their GPU timings to CPU performance. In [Elsen 2008], the authors give more details on the adopted strategy. The code solves the 3D Euler equations on structured grids with a Finite Difference method. The authors explain the problems they met (numerical precision, mapping from the computational grid onto the grid of the GPU,...),

and present their results on well-documented test cases, notably a 3D supersonic vehicle. The main lesson from this article is that on big 3D structured cases, speed-ups of 15 or 20 are satisfying.

Our study follows from these works: we aim at implementing on GPU an inviscid solver, in 3D and on unstructured meshes. The main novelty of the work is to consider unstructured meshes, whose topology does not map to the topology of the GPU.

B.4 Principles of implementation

Choices of implementation

Some choices had to be made prior to the implementation on GPU, more or less directly related to OpenCL, which we list here. The first determinant choice was to use a wrapper library written by Loïc Maréchal [Maréchal 2012]. This wrapper simplifies greatly the communications between the CPU and the GPU, notably the initialization of the connection to the GPU device, the compilation and instanciations of kernels and memory accesses. In standard OpenCL, each of theses tasks requires a consequent number of lines of code using several complex routines, that are factored in this wrapper. This way, the use of OpenCL is simplified in the host code, and one can focus on writing kernels. The consequence of this factoring is the use of default parameters in OpenCL routines. This could be a drawback if we were trying to optimize the code in detail, trying to use all subtleties of GPU architecture, but for a first attempt, we will only focus on algorithmic principles.

The other main choice, which may have more consequences, is to forget the distribution into workgroups, and act as if there was only one workgroup with all the cores and on which the whole pool of threads is executed. The distribution into workgroups is left to an OpenCL built-in routine, according to a heuristic based on the number of registries used by the kernels, and we do not use the manual handling of the different memory levels. This choice probably results in a sensible loss of performance, but we supposed it was sufficient to reach a good speed-up factor in a first phase.

Finally, the third choice is to use floating point numbers with simple precision on the GPU, in order to save some memory space, since it is limited on GPU devices. This may result in slowing down the convergence of the solver, and even preventing it in some specific cases.

Identification of 4 parts in the code

To simplify the code of the solver, we have identified four big parts, relatively autonomous, and their corresponding data structures. This segmentation allows us to work step by step in the parallelization of the code: one part can be ported to GPU while the other still run on CPU, so that each part can be debugged independently.

These parts come from the natural structure of the code, corresponding to the four big steps of the loop of a pseudo-steady solver:

- Computation of the time step: computation of the local CFL number, of the global minimal CFL number and of the local time step.

- Computation of the flux: application of the HLLC Riemann solver.
- Computation of the boundary conditions: application of different correction to the flux at the boundaries.
- Advance in time: computation of the new state and of the pseudo-steady residual.

Memory being a bottleneck of GPUs, early identification of all the structures that will be necessary is required to devise the code. These tables mainly contain:

- the current solution, of size 5 times the number of vertices,
- the initial solution (for each time step), of size 5 times the number of vertices,
- the flux, of size 5 times the number of vertices,
- the local time step, of size the number of vertices,
- geometrical parameters of the control volumes, of size twice the number of vertices,
- the normal to the edges, of size 4 times the number of edges,
- the vertices of the edges, of size twice the number of edges,
- a description of boundary tetrahedra, of size at least 7 times the number boundary tetrahedra,
- various details on the mesh topology...

Let us recall that the descritezation is done with a vertex-centered Finite Volume method, so the number of vertices is equal to the number of control volumes. In practice, for the solutions and the flux, we use arrays of width 6, because each line of these arrays is considered as small OpenCL vector, which are of sizes multiple of 2. In general, we use data structures as linear as may be (adjacent in memory, being read and written linearly...).

Implementation details

I cannot give all the details of the implementation here, so I only focus on the two main problems that had to be overcome.

Reduction. Reduction is a classic problem in parallel computing, which proves to particularly difficult on GPUs. There are two reductions in the solver code: the computation a minimal time step over the whole domain is necessary to set up a maximal bound for the local time step, and the computation of the residual is an integration over the domain, thus a sum over all vertices. We have tested two strategies: the classic one described in the literature, and a simpler one. Both eventually come to iteratively dividing the size of the array to reduce.

The classic method reduces the array step by step, by storing the result in the first elements of the array, which results in reducing the size of the data they handle every time. In other words, the array is split into 2 sub-arrays, and each pair of elements having the same index in

the sub-arrays is reduced and the result is stored in the first sub-array. This method gives way to a lot of optimizations, but our global strategy (with no explicit management of memory and workgroups) prevents us from making it really efficient.

Our alternative method consists in preparing in advance arrays of reduced sizes, and then reduce consecutive portions of the array to iteratively have arrays of size n/8 instead of arrays of size n, until we reach an array of size 256 that can be processed serially. In order to test both methods, the computation of the time step was implemented with the classic method and the computation of the residual with the other method.

Flux computation. The computation of the flux is the trickiest part of the code, because the flux at the vertices is updated by computing fluxes at the edges. In the sequential code, the edges are processed linearly, the formulas of the flux are applied on each edge, and the states of the two vertices at the ends of the edge are updated with the edge flux. On a GPU, this does not work anymore: if the computations of the fluxes of each edge are run in parallel all the threads corresponding to an edge sharing one vertex are going to try to update the state of this vertex at the same time, resulting in an undetermined state for the vertex (concurrency issue).

To overcome this issue, we adopted a simple solution. The computation is split in two steps: first, the edges are processed, storing the edge flux in an array indexed by the indexes of the edges, then the vertices are updated linearly using this edge flux array. To that end, it is required to compute the ball of edges neighboring a vertex at the initialization stage of the code. This way, concurrency issues are avoided, but the drawbacks are first the use of an extra array of size 5 times the number of edges to store the edge flux, and second the need to launch a second kernel to update the vertices (time is lost due to the overhead when launching a kernel, and due to the time necessary to go through all vertices).

B.5 Results and timings

To validate the code and test its performances, we have used a classic test case provided by the ONERA: a subsonic flow around a M6 wing geometry. The liberty to chose the parameters of the solver being reduced, this case offered a good balance, simple enough to run with the simplified solver but complex enough to be meaningful.

Two meshes were used (see Fig. B.1): one not too big to run tests on it with $58,619$ vertices and $311,310$ tetrahedra and a bigger one with $1,015,149$ vertices, et $5,448,373$ tetrahedra.

The validation of the code was performed step by step: after having implemented one part of the code on GPU, its output was tested against the output of the CPU version. The global output of the solver was also tested all the time, with the remaining parts being launched in their CPU version. On Fig. B.2, we show the density field of the solution after $1,000$ solver iterations on the bigger mesh, and we can see that the solutions are the same, although numerical artifacts such as a different handling of floating point numbers lead to tiny (negligible) differences.

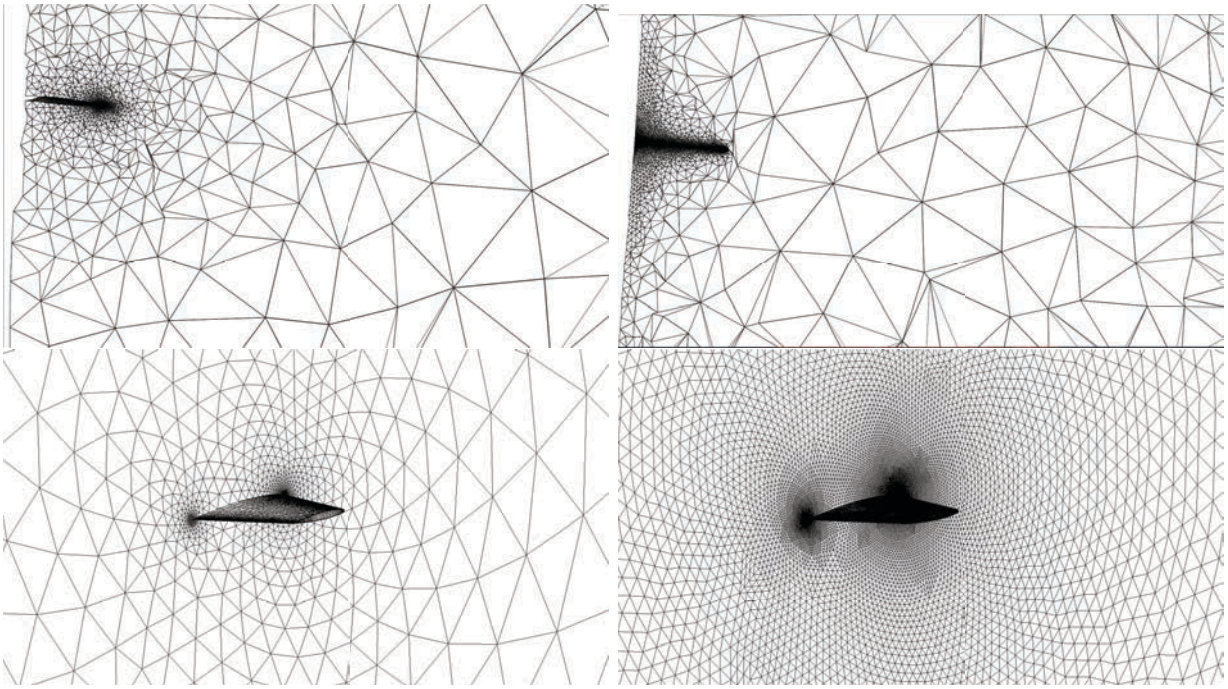

Figure B.1: Volumic and surfacic view of the meshes used, left with 58,619 vertices and right with 1,015,310 vertices.

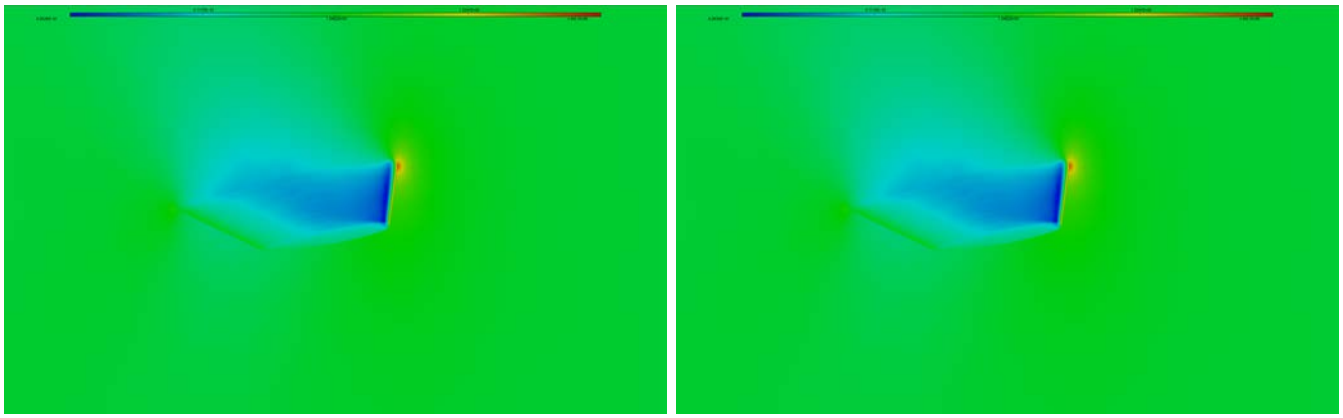

Figure B.2: Reference solution and solution fully computed on a GPU.

Timings

We have realized some interesting timings on these two test cases and on several machines. The three considered GPUs are:

- (GPU 1) an NVIDIA Tesla m2070 board (448 cores and 6 GB of memory), on the computation cluster of the Jacques Louis Lions Laboratory at University of Paris VI, which is a board dedicated to scientific computation,
- (GPU 2) an NVIDIA GeForce GT 330M board (48 cores and 512 MB of memory) on a

Macbook Pro laptop, typical of GPUs in everyday computers,

- (GPU 3) and an AMD Radeon HD 6750M board (480 cores and 512 MB of memory), which is a little bit old.

The CPU version was run on:

- (CPU 1) A server dedicated to scientific computation with a multicore processor: a 8 core Intel Xeon at 2.93 GHz and with hyperthreading,
- (CPU 2) A desktop computer with an hyperthreaded 4 core Intel i5 processor at 2.5 GHz, with GPU 3 integrated to the motherboard.

Timings (in sec)	GPU 1	GPU 2	GPU 3	CPU 2
Initializations	2.33	1.02	1.07	0.81
Computation of local time steps	0.39	1.57	0.68	
Computation of the min. time step	0.33	1.53	0.99	1.29
Computation of BCs	0.64	3.94	3.12	
Computation of the flux	11.11	29.08	12.12	29.17
Computation of the residual	0.32	1.45	0.39	
Advance in time	1.30	4.05	1.79	4.22
Total	**16.42**	**42.64**	**20.16**	**35.50**

Table B.1: Timings for the M6 wing mesh with $58,619$ vertices, with $1,000$ solver iterations, in seconds.

Timings (in sec)	GPU 1	CPU 1
Initializations	16.02	12.83
Computation of local time steps	7.56	
Computation of the min. time step	5.41	11.09
Computation of BCs	16.43	
Computation of the flux	192.47	181.73
Computation of the residual	2.97	
Advance in time	22.85	91.47
Total	**263.71**	**297.12**

Table B.2: Timings or the M6 wing mesh with $1,015,149$ vertices, with $1,000$ solver iterations, in seconds.

Two series of timings are presented in Tables B.1 and B.2. They both confirm the interest of the chosen approach, since the code is always faster on GPU, despite its many imperfections. However it is clear that reaching the same speed-up factors as on structured grids will be difficult.

On the smaller mesh, the computation-dedicated GPU is more than twice as fast as the CPU code on a 4-core hyperthreaded processor, which is reasonable for a first version. On the desktop computer, the GPU version is 1.75 times faster than the CPU version using all the cores: comparing the price and the energy consumption of both devices, the gain is obvious. On the bigger case, the GPU is still faster than a computation-dedicated server. The speed-up factor is lower (but still > 1), due to the limitations of our implementation choices, but the GPU code can still compete with a fast hyperthreaded 8-core CPU, with extreme memory conditions (for the bigger case, the memory of the GPU is full). In general, the laptop GPU is relatively slow, which was expected for laptop hardware, mostly designed for video display. To reach good performance, GPU devices with a large number of cores and efficient memory bandwith.

The timings can be analyzed more in depth, in relation with the 4 big parts described in section B.4.

The time required to initialize the data is clearly more important for the GPU, which is normal because all the data has to be converted in to the structures for the GPU, which leads to going through the whole mesh several extra times, and then the data has to be copied to the GPU memory, which is a slow process. However, the initialization stage is short relatively to the rest of the computation, all the more short as the size of the simulation is large.

The two reductions, implemented with two different methods, are interesting. The partly-optimized version (computation of the residual) is a little faster than the other (computation of the minimal time step). On the smaller mesh, the difference is very slight, but it is very perceptible on the bigger mesh, since the optimized version is twice as fast as the other. This observation should lead to go back on our choice to not handle explicitly memory levels. As expected, reduction is a bottleneck in GPU code: it is hardly faster on GPU than on CPU. The performance of these loops are very different depending on the GPU device, which is the occasion to observe the influence of the material architecture of GPUs on their performance: two big boards built with different designs can have very different behaviors.

The computation of boundary conditions is fast, which is due to the small number of boundary vertices. However, relatively to the number of vertices, the GPU time required is disappointing, and requires further investigation.

A lot of time is gained on the advance in time stage, which can be explained by the simplicity of the corresponding code, which is a textbook case for GPU programming: the sum of two arrays.

Finally, let us examine the longest part of the code, the computation of the volume fluxes. On GPU or CPU, it is the heart of the solver, and it is the part where a lot of time can be gained or lost. This stage is faster on GPU on the smaller mesh, but loses in efficiency on the bigger mesh. The longest part of this stage is the gather of the edge fluxes on the vertices: the code goes through the vertices, and for each vertices it goes through its ball of edges to access the corresponding edge fluxes. The ball of edges does not have the same size for every vertices, which is a problem since workgroups have only one control unit. Moreover, this loops suffers from multiple indirections and cache misses. Indirections cannot be circumvented, because we

have to go through the ball of edges one way or another. The use of an adapted sort (see Section 2.5.2) reduces caches misses, but cannot get totally rid of them.

A last point worth looking at is memory usage. We expected the limits of the central memories of GPUs to be quickly reached, but finally, we were able to run a relatively big case with a million of vertices.

Improvement of the flux computation

Considering the previous remarks, the longest loop (*i.e.* the computation of volume fluxes) was improved. In particular, we have used as much as possible small OpenCL vectors, and have re-organized the ball of edges structure. Instead of arrays of different sizes for every vertex, we have created a fixed size array of 16 edges. If a vertex does not have 16 neighbors, the array is filled with 0, and a small array containing the extra edges is created if necessary. This way, going through the ball of edge is the same thing for most of the vertices: going through an array of 16 indexes. This kind of data structure is much more adapted to GPU parallelization, and results in a much smaller time for this part.

With this improvement, we have run again the tests, on a new GPU device, a NVIDIA quadro 6000 (GPU 4). The device is supposedly the same as GPU 1, but with optimized and updated drivers. The timings reported in Table B.3 confirm that all this lead to better performance of the GPU code. In that table the two steps for the flux computation (computation of the edge flux, and assembly of the flux at the vertices) were separated for a better analysis.

Timings (in sec)	GPU 4	GPU 1	CPU 1	CPU 1 (serial)
Initialization	17.12	16.02	12.83	13
Computation of local time steps	0.81	2.31		
Computation of minimal time step	0.50	0.59	11.09	43
Computation of the BC	1.44	1.77		
Computation of the flux (edge flux)	12.17	43.98	181.73	1842
Computation of the flux (assembly)	37.55	72.14		
Computation of the residual	0.84	1.16		
Advance in time	3.05	8.80	91.47	94
Total	**73.48**	**146.77**	**297.12**	**1992**
Speed-up	**27.1**	**13.6**	**6.7**	**1**

Table B.3: Improved timings or the M6 wing mesh with $1,015,149$ vertices, with $1,000$ solver iterations, in seconds.

These timings (Table B.3) are really interesting, since the GPU code is 5 times as fast as a computation-dedicated 8-core hyperthreaded CPU, with a speed-up factor of 27. This relatively simple correction resulted in a great gain in GPU time, which shows the room for improvement ahead. The other parts have better timings due to the upgrade of the hardware drivers. The

part that requires the most CPU effort is going through the ball of edges, which represents a third of the total simulation time, and which we will be trying to speed up again.

B.6 Conclusion

These results are promising. With a reasonable investment in terms of man-time, and without dwelling into the subtle arcana of GPU architecture, we wrote a code that is five times as fast as the same code running on all the cores of a recent computation-dedicated server. These performances, in relation to the price of a GPU and to its electric consumption, make GPUs interesting for the conception of simulation codes, including complex codes.

At longer term, the success of such architectures will depend on their capability to evolve as fast as CPUs to remain competitive, to larger memories on the devices or to faster access to the central memory of the host machines, and probably as well on the existence of wrappers such as the `gmlib`, to facilitate semi-automatic conversion to GPU languages. In the mean time, hybrid architectures such as Intel Xeon Phi processors are developing, answering partially to the problems raised by GPUs, and may be the future of High Performance computing.

Bibliography

[Abgrall 2014] R. Abgrall, C. Dobrzynski and A. Froehly. *A method for computing curved meshes via the linear elasticity analogy, application to fluid dynamics problems.* International Journal for Numerical Methods in Fluids, vol. 76, no. 4, 2014.

[Aftosmis 2004] M.J. Aftosmis, M.J. Berger and S.M. Murman. *Applications of Space-Filling Curves to Cartesian Methods for CFD.* AIAA Paper, vol. 2004-1232, 2004.

[Alauzet 2003a] F. Alauzet. *Adaptation de maillage anisotrope en trois dimensions. Application aux simulations instationnaires en Mécanique des Fluides.* PhD thesis, Université Montpellier II, Montpellier, France, 2003. (in French).

[Alauzet 2003b] F. Alauzet and P.J. Frey. *Estimateur d'erreur géométrique et métrique anisotropes pour l'adaptation de maillage. Partie I : aspects théoriques.* RR-4759, INRIA, March 2003. (in French).

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

[Alauzet 2006b] F. Alauzet, A. Loseille, A. Dervieux and P.J. Frey. *Multi-dimensional continuous metric for mesh adaptation.* In Proceedings of the 15th International Meshing Roundtable, pages 191–214. Springer, 2006.

[Alauzet 2007] 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., vol. 222, pages 592–623, 2007.

[Alauzet 2009] F. Alauzet and A. Loseille. *On the use of space filling curves for parallel anisotropic mesh adaptation.* In Proceedings of the 18th International Meshing Roundtable, pages 337–357. Springer, 2009.

[Alauzet 2010a] F. Alauzet and A. Loseille. *High Order Sonic Boom Modeling by Adaptive Methods.* J. Comp. Phys., vol. 229, pages 561–593, 2010.

[Alauzet 2010b] F. Alauzet and M. Mehrenberger. *P1-conservative solution interpolation on unstructured triangular meshes.* International Journal for Numerical Methods in Engineering, vol. 84, no. 13, pages 1552–1588, 2010.

[Alauzet 2011a] F. Alauzet, A. Belme and A. Dervieux. *Anisotropic goal-oriented mesh adaptation for time dependent problems.* In Proceedings of the 20th International Meshing Roundtable, pages 99–121. Springer, 2011.

[Alauzet 2011b] F. Alauzet and G. Olivier. *Extension of Metric-Based Anisotropic Mesh Adaptation to Time-Dependent Problems Involving Moving Geometries.* In 49th AIAA Aerospace Sciences Meeting, AIAA Paper 2011-0896, Orlando, FL, USA, Jan 2011.

[Alauzet 2012] F. Alauzet. Contributions aux méthodes numériques pour l'adaptation de maillage et le maillage mobile. Habilitation à Diriger des Recherches, Université Pierre et Marie Curie, Paris VI, Paris, France, 2012.

[Alauzet 2014a] F. Alauzet. *A changing-topology moving mesh technique for large displacements*. Engineering with Computers, vol. 30, no. 2, pages 175–200, 2014.

[Alauzet 2014b] F. Alauzet. `Wolf` *documentation. A Navier-Stokes flow solver based on the MEV numerical scheme*. Internal report, INRIA, 2014.

[Alauzet 2015] F. Alauzet and A. Loseille. *A decade of progress on anisotropic mesh adaptation for Computational Fluid Dynamics*. Comput. Aided Des., 2015. Accepted.

[Allain 2009] O. Allain, D. Guégan and F. Alauzet. *Studying the impact of unstructured mesh adaptation on free surface flow simulations*. In Proceedings of the ASME 28th International Conference on Ocean, Offshore and Arctic Engineering, 2009.

[Appleyard 2011] J. Appleyard and D. Drikakis. *Higher-order CFD and interface tracking methods on highly-Parallel MPI and GPU systems*. Computers & Fluids, vol. 46, pages 101–105, 2011.

[Arpaia 2015] L. Arpaia and M. Ricchiuto. *Mesh adaptation by continuous deformation. Basics: accuracy, efficiency, well balancedness*. RR-8666, Inria Bordeaux Sud-Ouest, 2015.

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

[Astorino 2009] M. Astorino, F. Chouly and M.A. Fernández. *Robin-based Semi-Implicit Coupling in Fluid-Structure Interaction: Stability Analysis and Numerics*. SIAM J. Sci. Comput., vol. 31, no. 6, pages 4041–4065, 2009.

[Babuska 1981] I. Babuska, B. A. Szabo and I. N. Katz. *The p-Version of the Finite Element Method*. SIAM Journal on Numerical Analysis, vol. 18, no. 3, pages 515–545, 1981.

[Babuska 1994] I. Babuska and M. Suri. *The p and h-p Versions of the Finite Element Method, Basic Principles and Properties*. SIAM Review, vol. 36, no. 4, pages 578–632, 1994.

[Baines 1994] M.J. Baines. Moving finite elements. Oxford University Press, Inc., New York, NY, 1994.

[Baker 1997] T.J. Baker. *Mesh adaptation strategies for problems in fluid dynamics*. Finite Elem. Anal. Des., vol. 25, pages 243–273, 1997.

[Baker 1999] T.J. Baker and P. Cavallo. *Dynamic adaptation for deforming tetrahedral meshes*. AIAA Journal, vol. 19, pages 2699–3253, 1999.

[Bank 1993] R.E. Bank and R.K. Smith. *A posteriori error estimate based on hierarchical bases*. SIAM J. Numer. Anal., vol. 30, pages 921–935, 1993.

[Barral 2014a] N. Barral and F. Alauzet. *Large displacement body-fitted FSI simulations using a mesh-connectivity-change moving mesh strategy.* In 44th AIAA Fluid Dynamics Conference, AIAA Paper 2014-2773, Atlanta, GA, USA, June 2014.

[Barral 2014b] N. Barral and F. Alauzet. *Large displacement simulations with an efficient mesh-connectivity-change moving mesh strategy.* In ECCOMAS 2014, Barcelona, Spain, July 2014.

[Barral 2014c] N. Barral and F. Alauzet. *Parallel time-accurate anisotropic mesh adaptation for time-dependent problems.* In ECCOMAS 2014, Barcelona, Spain, July 2014.

[Barral 2014d] N. Barral, E. Luke and F. Alauzet. *Two Mesh Deformation Methods Coupled with a Changing-connectivity Moving Mesh Method for {CFD} Applications.* Procedia Engineering, vol. 82, pages 213 – 227, 2014. 23rd International Meshing Roundtable (IMR23), , London, UK.

[Barral 2015] N. Barral, F. Alauzet and A. Loseille. *Metric-Based Anisotropic Mesh Adaptation for Three-Dimensional Time-Dependent Problems Involving Moving Geometries.* In 53th AIAA Aerospace Sciences Meeting, AIAA Paper 2015-2039, Kissimmee, FL, USA, Jan 2015.

[Batina 1990] J. Batina. *Unsteady Euler Airfoil Solutions Using Unstructured Dynamic Meshes.* AIAA Journal, vol. 28, no. 8, pages 1381–1388, 1990.

[Batten 1997] P. Batten, N. Clarke, C. Lambert and D.M. Causon. *On the choice of wavespeeds for the HLLC Riemann solver.* SIAM J. Sci. Comput., vol. 18, no. 6, pages 1553–1570, 1997.

[Baum 1991] J.D. Baum and R. Löhner. *Numerical Simulation of a Shock Interaction with a Modern Battlefield Tank.* In AIAA-91-1666, 1991.

[Baum 1994] J.D. Baum, H. Luo and R. Löhner. *A New ALE Adaptive Unstructured Methodology for the Simulation of Moving Bodies.* In 32th AIAA Aerospace Sciences Meeting, AIAA Paper 1994-0414, Reno, NV, USA, Jan 1994.

[Baum 1996] J.D. Baum, H. Luo, R. Löhner, C. Yang, D. Pelessone and C. Charman. *A Coupled Fluid/Structure Modeling of Shock Interaction with a Truck.* In AIAA-96-0795, 1996.

[Bazilevs 2011] Y. Bazilevs, M.-C. Hsu, J. Kiendl, R. Wüchner and K.-U. Bletzinger. *3D simulation of wind turbine rotors at full scale. Part II: Fluid–structure interaction modeling with composite blades.* International Journal for Numerical Methods in Fluids, vol. 65, no. 1-3, pages 236–253, 2011.

[Belhamadia 2004] Y. Belhamadia, A. Fortin and E. Chamberland. *Three-dimensional anisotropic mesh adaptation for phase change problems.* J. Comp. Phys., vol. 201, pages 753–770, 2004.

[Belme 2011] A. Belme. *Unsteady aerodynamic and adjoint method.* PhD thesis, Université de Nice - Sophia Antipolis, Nice, France, 2011.

[Benek 1985] J.A. Benek, P.G. Buning and J.L. Steger. *A 3D Chimera Grid Embedding Technique*. In 7th AIAA Computational Fluid Dynamics Conference, AIAA Paper 1985-1523, Cincinnati, OH, USA, Jul 1985.

[Berger 1984] M. J. Berger and J. Oliger. *Adaptive mesh refinement for hyperbolic partial differential equations*. J. Comp. Phys., vol. 53, no. 3, pages 484 – 512, 1984.

[Borouchaki 1998] H. Borouchaki, F. Hecht and P.J. Frey. *Mesh gradation control*. Int. J. Numer. Meth. Engng, vol. 43, no. 6, pages 1143–1165, 1998.

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

[Bourgault 2009] Y. Bourgault, M. Picasso, F. Alauzet and A. Loseille. *On the use of anisotropic error estimators for the adaptative solution of 3-D inviscid compressible flows*. Int. J. Numer. Meth. Fluids, vol. 59, pages 47–74, 2009.

[Browne 2014] P.A. Browne, C.J. Budd, C. Piccolo and M. Cullen. *Fast three dimensional r-adaptive mesh redistribution*. Journal of Computational Physics, vol. 275, pages 174 – 196, 2014.

[Bruchon 2009] J. Bruchon, H. Digonnet and T. Coupez. *Using a signed distance function for the simulation of metal forming process: formulation of the contact condition and mesh adaptation. From Lagrangian approach to an Eulerian approach*. Int. J. Numer. Meth. Engng, vol. 78, no. 8, pages 980–1008, 2009.

[Buscaglia 1997] G.C. Buscaglia and E.A. Dari. *Anisotropic Mesh Optimization and its Application in Adaptivity*. Int. J. Numer. Meth. Engng, vol. 40, pages 4119–4136, 1997.

[Cao 2002] W. Cao, W. Huang and R.D. Russell. *A Moving Mesh Method based on the Geometric Conservation Law*. SIAM J. Sci. Comput., vol. 47, pages 118–142, 2002.

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

[Chacón 2011] L. Chacón, G.L. Delzanno and J.M. Finn. *Robust, multidimensional mesh-motion based on Monge–Kantorovich equidistribution*. Journal of Computational Physics, vol. 230, no. 1, pages 87–103, 2011.

[Ciarlet 1991] P.G. Ciarlet. Basic error estimates for elliptic problems, volume II of *Handbook of Numerical Analysis*. in: P.G. Ciarlet and J.L. Lions (Eds.), Handbook of Numerical Analysis, vol. II, Finite Element Methods (Part 1), North-Holland, Amsterdam, p.g. ciarlet and j.l. lions (eds.) édition, 1991. in: P.G. Ciarlet and J.L. Lions (Eds.), Handbook of Numerical Analysis, vol. II, Finite Element Methods (Part 1).

[Clément 1975] P. Clément. *Approximation by finite element functions using local regularization*. Revue Française d'Automatique, Informatique et Recherche Opérationnelle, vol. R-2, pages 77–84, 1975.

[Compère 2007] G. Compère, E. Marchandise and J.-F. Remacle. *Transient adaptivity applied to two-phase incompressible flows.* J. Comp. Phys., vol. 227, pages 1923–1942, 2007.

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

[Compère 2008] G. Compère, E. Marchandise and J.-F. Remacle. *Transient adaptivity applied to two-phase incompressible flows.* Journal of Computational Physics, vol. 227, no. 3, pages 1923 – 1942, 2008.

[Coupez 2000] T. Coupez. *Génération de maillages et adaptation de maillage par optimisation locale.* Revue Européenne des Éléments Finis, vol. 9, pages 403–423, 2000.

[Coupez 2013] T. Coupez, G. Jannoun, N. Nassif, H.C. Nguyen, H. Digonnet and E. Hachem. *Adaptive time-step with anisotropic meshing for incompressible flows.* Journal of Computational Physics, vol. 241, no. 0, pages 195 – 211, 2013.

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

[de Boer 2007] A. de Boer, M. van der Schoot and H. Bijl. *Mesh deformation based on radial basis function interpolation.* Comput. & Struct., vol. 85, pages 784–795, 2007.

[de Sampaio 1993] P.A. de Sampaio, P.R. Lyra, K. Morgan and N. Weatherill. *Petrov-Galerkin solutions of the incompressible Navier-Stokes equations in primitive variables with adaptive remeshing.* Comput. Methods Appl. Mech. Engrg., vol. 106, pages 143–178, 1993.

[Debiez 2000] C. Debiez and A. Dervieux. *Mixed-Element-Volume MUSCL methods with weak viscosity for steady and unsteady flow calculations.* Comput. & Fluids, vol. 29, pages 89–118, 2000.

[Degand 2002] Christoph Degand and Charbel Farhat. *A three-dimensional torsional spring analogy method for unstructured dynamic meshes.* Comput. & Struct., vol. 80, no. 3–4, pages 305 – 316, 2002.

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

[Dompierre 1997] J. Dompierre, M.G. Vallet, M. Fortin, Y. Bourgault and W.G. Habashi. *Anisotropic mesh adaptation: towards a solver and user independent CFD.* In AIAA 35th Aerospace Sciences Meeting and Exhibit, AIAA-1997-0861, Reno, NV, USA, Jan 1997.

[Donea 1982] J. Donea, S. Giuliani and J. P. Halleux. *An arbitrary Lagrangian-Eulerian finite element method for transient dynamic fluid-structure interactions.* Comput. Methods Appl. Mech. Engrg., vol. 33, no. 1, pages 689–723, 1982.

[Dunavant 1985] D.A. Dunavant. *High degree efficient symmetrical Gaussian quadrature rules for the triangle.* International journal for numerical methods in engineering, vol. 21, no. 6, pages 1129–1148, 1985.

[Dvinsky 1991] A.S. Dvinsky. *Adaptive Grid Generation from Harmonic Maps on Riemannian Manifolds.* J. Comp. Phys., vol. 95, pages 450–476, 1991.

[Elsen 2008] E. Elsen, P. LeGresley and E. Darve. *Large calculation of the flow over a hypersonic vehicle using a GPU.* Journal of Computational Physics, vol. 227, pages 10148–10161, 2008.

[Etienne 2009] S. Etienne, A. Garon and D. Pelletier. *Perspective on the Geometric Conservation Law and Finite Element Methods for ALE Simulations of Incompressible Flow.* J. Comp. Phys., vol. 228, no. 7, pages 2313–2333, 2009.

[Etienne 2010] S. Etienne, A. Garon, D. Pelletier and C. Cameron. *Philiadium gregarium Versus Aurelia aurita: On Propulsion Propulsion of Jellyfish.* In 48th AIAA Aerospace Sciences Meeting, AIAA Paper 2010-1444, Orlando, FL, USA, Jan 2010.

[Farhat 2001] C. Farhat, P. Geuzaine and C. Grandmont. *The Discrete Geometric Conservation Law and the Nonlinear Stability of ALE Schemes for the Solution of Flow Problems on Moving Grids.* J. Comp. Phys., vol. 174, no. 2, pages 669–694, 2001.

[Formaggia 2001] L. Formaggia and S. Perotto. *New anisotropic a priori error estimates.* Numer. Math., vol. 89, pages 641–667, 2001.

[Formaggia 2004] L. Formaggia and F. Nobile. *Stability analysis of second-order time accurate schemes for ALE–FEM.* Computer Methods in Applied Mechanics and Engineering, vol. 193, no. 39–41, pages 4097 – 4116, 2004.

[Formaggia 2009] L. Formaggia, A. Quarteroni and A. Veneziani. Cardiovascular Mathematics: Modeling and simulation of the circulatory system, volume 1. Springer - Modelling, Simulations and Applications, 2009.

[Fortin 1996] M. Fortin, M.-G. Vallet, J. Dompierre, Y. Bourgault and W.G. Habashi. *Anisotropic mesh adaptation : theory, validation and applications.* In Proceedings of ECCOMAS CFD, 1996.

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

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

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

[Froehle 2014] B. Froehle and P.-O. Persson. *High-order accurate fluid–structure simulation of a tuning fork.* Computers & Fluids, vol. 98, pages 230–238, 2014.

[Gammacurta 2010] E. Gammacurta, S. Etienne, D. Pelletier and A. Garon. *Adaptive Remeshing for Unsteady RANS Computations*. In 48th AIAA Aerospace Sciences Meeting, AIAA Paper 2010-1070, Orlando, FL, USA, 2010.

[George 1991a] P.L. George, F. Hecht and E. Saltel. *Automatic mesh generator with specified boundary*. Comput. Methods Appl. Mech. Engrg., vol. 92, pages 269–288, 1991.

[George 1991b] P.L. George, F. Hecht and M.G. Vallet. *Creation of internal points in Voronoi's type method. Control and adaptation*. Adv. Eng. Software, vol. 13, no. 5-6, pages 303–312, 1991.

[George 2003] P.L. George and H. Borouchaki. *"Ultimate" robustness in meshing an arbitrary polyhedron*. Int. J. Numer. Meth. Engng, vol. 58, no. 7, pages 1061–1089, 2003.

[George 2015a] P.L. George, H. Borouchaki and N. Barral. *Construction and geometric validity (positive Jacobian) of serendipity Lagrange finite elements, theory and practical guidance*. Mathematical Modelling and Numerical Analysis, 2015. Submitted.

[George 2015b] P.L. George, H. Borouchaki and N. Barral. *Geometric validity (positive jacobian) of high-order Lagrange finite elements, theory and practical guidance*. Engineering with computers, 2015. accepted.

[Gerbeau 2014] J.F. Gerbeau, D. Lombardi and E. Schenone. *Reduced order model in cardiac electrophysiology with approximated Lax pairs*. Advances in Computational Mathematics, pages 1–28, 2014.

[Gruau 2005] C. Gruau and T. Coupez. *3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric*. Comput. Methods Appl. Mech. Engrg., vol. 194, no. 48-49, pages 4951–4976, 2005.

[Guardone 2011] A. Guardone, D. Isola and G. Quaranta. *Arbitrary Lagrangian Eulerian formulation for two-dimensional flows using dynamic meshes with edge swapping*. Journal of Computational Physics, vol. 230, no. 20, pages 7706 – 7722, 2011.

[Guégan 2010] D. Guégan, O. Allain, A. Dervieux and F. Alauzet. *An L^∞-L^p mesh adaptive method for computing unsteady bi-fluid flows*. Int. J. Numer. Meth. Engng, vol. 84, no. 11, pages 1376–1406, 2010.

[Hassan 2000] O. Hassan, E. J. Probert, K. Morgan and N. P. Weatherill. *Unsteady flow simulation using unstructured meshes*. Comput. Methods Appl. Mech. Engrg., vol. 189, pages 1247–1275, 2000.

[Hassan 2007] O. Hassan, K.A. Sørensen, K. Morgan and N. P. Weatherill. *A method for time accurate turbulent compressible fluid flow simulation with moving boundary components employing local remeshing*. Int. J. Numer. Meth. Fluids, vol. 53, no. 8, pages 1243–1266, 2007.

[Hay 2007] A. Hay and M. Visonneau. *Adaptive finite-volume solution of complex turbulent flows*. Comput. & Fluids, vol. 36, no. 8, pages 1347 – 1363, 2007.

[Hay 2014] A. Hay, K.R. Yu, S. Etienne, A. Garon and D. Pelletier. *High-order temporal accuracy for 3D finite-element ALE flow simulations.* Comput. & Fluids, vol. 100, no. 0, pages 204–217, 2014.

[Hecht 1997] F. Hecht and B. Mohammadi. *Mesh adaptation by metric control for multi-scale phenomena and turbulence.* In 35th AIAA Aerospace Sciences Meeting and Exhibit, AIAA-1997-0859, Reno, NV, USA, Jan 1997.

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

[Hirt 1974] C. W. Hirt, A. A. Amsden and J. L. Cook. *An arbitrary Lagrangian-Eulerian computing method for all flow speeds.* J. Comp. Phys., vol. 14, no. 3, pages 227–253, 1974.

[Huang 2005] W. Huang. *Metric tensors for anisotropic mesh generation.* J. Comp. Phys., vol. 204, no. 2, pages 633–665, 2005.

[Huang 2010a] W. Huang, L. Kamenski and X. Li. *A new anisotropic mesh adaptation method based upon hierarchical a posteriori error estimates.* J. Comp. Phys., vol. 229, pages 2179–2198, 2010.

[Huang 2010b] W. Huang and R. D. Russell. Adaptive moving mesh methods, volume 174. Springer Science & Business Media, 2010.

[Hughes 1981] T. J. R. Hughes, W. K. Liu and T. K. Zimmermann. *Lagrangian-Eulerian finite element formulation for incompressible viscous flows.* Comput. Methods Appl. Mech. Engrg., vol. 29, no. 3, pages 329–349, 1981.

[Isola 2015] D. Isola, A. Guardone and G. Quaranta. *Finite-volume solution of two-dimensional compressible flows over dynamic adaptive grids.* J. Comp. Phys., vol. 285, pages 1 – 23, 2015.

[Jayaraman 2000] B. Jayaraman and D.L. Marcum. *A General Procedure for Dynamic Unstructured Grids.* In 7th International Conference on Numerical Grid Generation in Computational Field Simulations, Whistler, BC, Sep 2000.

[Jinyun 1984] Y. Jinyun. *Symmetric Gaussian quadrature formulae for tetrahedronal regions.* Computer Methods in Applied Mechanics and Engineering, vol. 43, no. 3, pages 349–353, 1984.

[Johnen 2013] A. Johnen, J-F. Remacle and C. Geuzaine. *Geometrical validity of curvilinear finite elements.* J. Comp. Phys., vol. 233, pages 359–372, 2013.

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

[Kitamura 2011] K. Kitamura, K. Fujimoto, E. Shima, K. Kuzuu and Z.J.Wang. *Validation of Arbitrary Unstructured CFD Code for Aerodynamic Analyses.* Transactions of the Japan Society for Aeronautical and Space Sciences, vol. 53, no. 182, pages 311–319, 2011.

[Koobus 1999] B. Koobus and C. Farhat. *Second-order time-accurate and geometrically conservative implicit schemes for flow computations on unstructured dynamic meshes.* Comput. Methods Appl. Mech. Engrg., vol. 170, no. 1-2, pages 103–129, 1999.

[Kucharik 2008] M. Kucharik and M. Shashkov. *Extension of Efficient, Swept-Integration-Based Conservative Method for Meshes with Changing Connectivity.* Int. J. Numer. Meth. Fluids, vol. 56, no. 8, pages 1359–1365, 2008.

[Kuo 2011] F.-A. Kuo, M. R. Smith, C.-W. Hsieh, C.-Y. Chou and J.-S. Wu. *GPU acceleration for general conservation equations and its application to several engineering problems.* Computers & Fluids, vol. 45, pages 147–154, 2011.

[Lacasse 2007] D. Lacasse, A. Garon and D. Pelletier. *Development of an adaptive Discontinuous-Galerkin finite element method for advection–reaction equations.* Computer Methods in Applied Mechanics and Engineering, vol. 196, no. 17–20, pages 2071 – 2083, 2007.

[Landon 1982] R.H. Landon. *Compendium of unsteady aerodynamic measurements.* AGARD Report, vol. 702, 1982.

[Langseth 2000] J.O. Langseth and R. J. LeVeque. *A wave propagation method for three-dimensional hyperbolic conservation laws.* J. Comp. Phys., vol. 165, pages 126–166, 2000.

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

[Lefrançois 2010] E. Lefrançois and J.P. Boufflet. *An introduction to fluid-structure interaction: application to the piston problem.* SIAM review, vol. 52, no. 4, pages 747–767, 2010.

[Lesoinne 1996] M. Lesoinne and C. Farhat. *Geometric Conservation Laws for Flow Problems with Moving Boundaries and Deformable Meshes, and their Impact on Aeroelastic Computations.* Comput. Methods Appl. Mech. Engrg., vol. 134, no. 1-2, pages 71–90, 1996.

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

[Liepmann 2002] H.W. Liepmann and A. Roshko. Elements of gas dynamics. Dover Publications, 2002.

[Löhner 1988] R. Löhner and P. Parikh. *Three-dimensional grid generation by the advancing front method.* Int. J. Numer. Meth. Fluids, vol. 9, pages 1135–1149, 1988.

[Löhner 1989a] R. Löhner. *Adaptive Remeshing for Transient Problems.* Int. J. Numer. Meth. Engng, vol. 75, no. 1–3, pages 195–214, 1989.

[Löhner 1989b] R. Löhner. *Adaptive Remeshing for Transient Problems.* Comput. Methods Appl. Mech. Engrg., vol. 75, no. 1-3, pages 195–214, 1989.

[Löhner 1990a] R. Löhner. *Three-Dimensional Fluid-Structure Interaction Using a Finite Element Solver and Adaptive Remeshing.* Computing Systems in Engineering, vol. 1, no. 2-4, pages 257–272, 1990.

[Löhner 1990b] R. Löhner. *Three-dimensional fluid-structure interaction using a finite element solver and adaptive remeshing.* Computing Systems in Engineering, vol. 1, no. 2-4, pages 257–272, 1990.

[Löhner 1990c] R. Löhner and J.D. Baum. *Numerical Simulation of a Shock Interaction with Complex Geometry Three-Dimensional Structures using a New Adaptive H-Refinement Scheme On Unstructured Grids.* In AIAA-90-0700, 1990.

[Löhner 1992] R. Löhner and J.D. Baum. *Adaptive h-refinement on 3D unstructured grids for transient problems.* Int. J. Numer. Meth. Fluids, vol. 14, no. 12, pages 1407–1419, 1992.

[Löhner 1998] R. Löhner and C. Yang. *Improved ALE mesh velocities for moving bodies.* Communications in numerical methods in engineering, no. 12, pages 599–608, 1998.

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

[Loseille 2007] A. Loseille, A. Dervieux, P.J. Frey and F. Alauzet. *Achievement of global second-order mesh convergence for discontinuous flows with adapted unstructured meshes.* In 37th AIAA Fluid Dynamics Conference and Exhibit, AIAA-2007-4186, Miami, FL, USA, Jun 2007.

[Loseille 2008] A. Loseille. *Adaptation de maillage anisotrope 3D multi-échelles et ciblée à une fonctionnelle pour la mécanique des fluides. Application à la prédiction haute-fidélité du bang sonique.* PhD thesis, Université Pierre et Marie Curie, Paris VI, Paris, France, 2008. (in French).

[Loseille 2009a] A. Loseille and F. Alauzet. *Optimal 3D highly anisotropic mesh adaptation based on the continuous mesh framework.* In Proceedings of the 18th International Meshing Roundtable, pages 575–594. Springer, 2009.

[Loseille 2009b] A. Loseille and R. Löhner. *On 3D anisotropic local remeshing for surface, volume, and boundary layers.* In Proceedings of the 18th International Meshing Roundtable, pages 611–630. Springer, 2009.

[Loseille 2010a] A. Loseille and F. Alauzet. *Fully Anisotropic Goal-Oriented Mesh Adaptation for 3D Steady Euler Equations.* J. Comp. Phys., vol. 229, no. 8, pages 2866–2897, 2010.

[Loseille 2010b] A. Loseille, A. Dervieux and F. Alauzet. *A 3D Goal-Oriented Anisotropic Mesh Adaptation Applied to Inviscid Flows in Aeronautics.* In 48th AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2010-1067, Orlando, FL, USA, Jan 2010.

[Loseille 2011a] A. Loseille and F. Alauzet. *Continuous mesh framework. Part I: well-posed continuous interpolation error.* SIAM J. Numer. Anal., vol. 49, no. 1, pages 38–60, 2011.

[Loseille 2011b] A. Loseille and F. Alauzet. *Continuous mesh framework. Part II: validations and applications.* SIAM J. Numer. Anal., vol. 49, no. 1, pages 61–86, 2011.

[Loseille 2013] A. Loseille and V. Menier. *Serial and Parallel Mesh Modification Through a Unique Cavity-Based Primitive.* In Proceedings of the 22th International Meshing Roundtable, pages 541–558. Springer, Orlando, Fl, USA, 2013.

[Loubère 2010] R. Loubère, P-H. Maire, M. Shashkov, J. Breil and S. Galera. *ReALE: a Reconnaction-Based Arbitrary-Lagrangian-Eulerian Method.* J. Comp. Phys., vol. 229, pages 4724–4761, 2010.

[Luke 2012] E. Luke, E. Collins and E. Blades. *A fast mesh deformation method using explicit interpolation.* J. Comp. Phys., vol. 231, pages 586–601, 2012.

[Marcum 2000] D.L. Marcum. *Unstructured grid generation using automatic point insertion and local reconnection.* Revue Européenne des Éléments Finis, vol. 9, pages 403–423, 2000.

[Marcum 2014] D. Marcum and F. Alauzet. *Aligned metric-based anisotropic solution adaptive mesh generation.* Proceedings of the 23th International Meshing Roundtable, Procedia Engineering, vol. 82, pages 428–444, 2014.

[Maréchal 2012] L. Maréchal. *Une aide à la programmation sur GPU pour le calcul scientifique. La librairie GM2.* Technical Note, INRIA, 2012.

[Maréchal 2016] L. Maréchal. *Handling unstructured meshes in multithreaded environments with the help of Hilbert renumbering and dynamic scheduling.* Parallel Computing, to be published 2016.

[Masters 2015] J.S. Masters and K.E. Tatum. *Mesh Manipulation for 3D Tetrahedral Meshes.* In 53th AIAA Aerospace Sciences Meeting, AIAA Paper 2015-2039, Kissimmee, FL, USA, Jan 2015.

[Mavriplis 1990] D.J. Mavriplis. *Adaptive mesh generation for viscous flows using Delaunay triangulation.* J. Comp. Phys., vol. 90, pages 271–291, 1990.

[Mavriplis 2006] D.J. Mavriplis and Z. Yang. *Construction of the Discrete Geometric Conservation Law for High-Order Time Accurate Simulations on Dynamic Meshes.* J. Comp. Phys., vol. 213, no. 2, pages 557–573, 2006.

[McCullen 2003] M. S. McCullen. *The Application of Non-Linear Frequency Domain Methods to the Euler and Navier-Stokes Equations.* PhD thesis, Stanford University, Stanford, CA, USA, 2003.

[McKenzie 2009] S. McKenzie, J. Dompierre, A. Turcotte and E. Meng. *On metric tensor representation, intersection, and union.* 11th ISGG Conference on Numerical Grid Generation, may 2009.

[Menier 2015] V. Menier. *Mesh Adaptation For the High Fidelity Prediction of Viscous Phenomena and Their Interactions. Application to Aeronautics.* PhD thesis, Université Pierre et Marie Curie, Paris VI, Paris, France, 2015.

[Michal 2012] T. Michal and J. Krakos. *Anisotropic mesh adaptation through edge primitive operations.* 50th AIAA Aerospace Sciences Meeting, Jan 2012.

[Miller 1981] K. Miller and R. N. Miller. *Moving Finite Elements. I.* SIAM Journal on Numerical Analysis, vol. 18, no. 6, pages 1019–1032, 1981.

[Munshi 2008] Khronos OpenCL Working Group – Aaftab Munshi. *The OpenCL Specification*, 2008.

[Murman 2003] S. Murman, M. Aftosmis and M. Berger. *Simulation of 6-DOF Motion with Cartesian Method.* In 41th AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2003-1246, Reno, NV, USA, Jan 2003.

[Nkonga 1994] B. Nkonga and H. Guillard. *Godunov type method on non-structured meshes for three-dimensional moving boundary problems.* Computer Methods in Applied Mechanics and Engineering, vol. 113, no. 1–2, pages 183 – 204, 1994.

[Nkonga 2000] B. Nkonga. *On the Conservative and Accurate CFD Approximations for Moving Meshes and Moving Boundaries.* Comput. Methods Appl. Mech. Engrg., vol. 190, no. 13, pages 1801–1825, 2000.

[Olivier 2011a] G. Olivier. *Anisotropic metric-based mesh adaptation for unsteady CFD simulations involving moving geometries.* PhD thesis, Université Pierre et Marie Curie, Paris VI, Paris, France, 2011.

[Olivier 2011b] G. Olivier and F. Alauzet. *A New Changing-Topology ALE Scheme for Moving Mesh Unsteady Simulations.* In 49th AIAA Aerospace Sciences Meeting, AIAA Paper 2011-0474, Orlando, FL, USA, Jan 2011.

[Pain 2001] C.C Pain, A.P. Humpleby, C.R.E. de Oliveira and A.J.H. Goddard. *Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations.* Comput. Methods Appl. Mech. Engrg., vol. 190, pages 3771–3796, 2001.

[Pendenza 2014] A. Pendenza, W. G. Habashi and M. Fossati. *A 3D mesh deformation technique for irregular in-flight ice accretion.* In 44th AIAA Fluid Dynamics Conference, AIAA Paper 2014-3072, Atlanta, GA, USA, June 2014.

[Peraire 1987] J. Peraire, M. Vahdati, K. Morgan and O.C. Zienkiewicz. *Adaptive Remeshing for Compressible Flow Computations.* J. Comput. Phys., vol. 72, pages 449–466, 1987.

[Peraire 1992] J. Peraire, J. Peiro and K. Morgan. *Adaptive Remeshing for Three-Dimensional Compressible Flow Computations.* J. Comput. Phys., vol. 103, pages 269–285, 1992.

[Picasso 2003] M. Picasso. *An anisotropic error indicator based on Zienkiewicz-Zhu error estimator: Application to elliptic and parabolic problems.* SIAM J. Sci. Comput., vol. 24, no. 4, pages 1328–1355, 2003.

[Picasso 2009] M. Picasso, V. Prachittham and M.A.M. Gijs. *Adaptive Finite Elements with Large Aspect Ratio for Mass Transport in Electro-osmosis and Pressure-Driven Microflows.* Int. J. Numer. Meth. Fluids, 2009.

[Piggott 2009] M.D. Piggott, P.E. Farrell, C.R. Wilson, G.J Gorman and C.C. Pain. *Anisotropic mesh adaptivity for multi-scale ocean modelling.* Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, vol. 367, no. 1907, pages 4591–4611, 2009.

[Power 2006] 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.* Comput. Math. Appl., vol. 52, pages 1213–1242, 2006.

[Puigt 2011] G. Puigt, M. Gazaix, M. Montagnac, M.-C. LePape, M. delaLlavePlata, C. Marmignon, J.-F. Boussuge and V. Couaillier. *Development of a new hybrid compressible solver inside the CFD elsA software.* In 41st AIAA Fluid Dynamics Conference and Exhibit, AIAA-2011-3048, Hawaii, HO, USA, Jun 2011.

[Rausch 1992] R.D. Rausch, J.T. Batina and H.T.Y. Yang. *Spatial adaptation procedures on tetrahedral meshes for unsteady aerodynamic flow calculations.* AIAA Journal, vol. 30, pages 1243–1251, 1992.

[Remacle 2005] J.-F. Remacle, X. Li, M.S. Shephard and J.E. Flaherty. *Anisotropic adaptive simulation of transient flows using discontinuous Galerkin methods.* Int. J. Numer. Meth. Engng, vol. 62, pages 899–923, 2005.

[Roe 1981] P. Roe. *Approximate Riemann solvers, parameter vectors, and difference schemes.* J. Comp. Phys., vol. 43, pages 357–372, 1981.

[Sagan 1994] H. Sagan. Space-filling curves. Springer, New York, NY, 1994.

[Saksono 2007] P.H. Saksono, W.G. Dettmer and D. Perić. *An Adaptive Remeshing Strategy for Flows with Moving Boundaries and Fluid-Structure Interaction.* Int. J. Numer. Meth. Engng, vol. 71, no. 9, pages 1009–1050, 2007.

[Selmin 1992] V. Selmin and L. Formaggia. *Simulation of hypersonic flows on unstructured grids.* Int. J. Numer. Meth. Engng, vol. 34, pages 569–606, 1992.

[Shaw 1992] J.A. Shaw and N.P. Weatherill. *Automatic topology generation for multiblock grids.* Applied Mathematics and Computation, vol. 52, pages 355–388, 1992.

[Shu 1988] C.W. Shu and S. Osher. *Efficient implementation of essentially non-oscillatory shock-capturing schemes.* J. Comput. Phys., vol. 77, pages 439–471, 1988.

[Southern 2010] J. Southern, G.J. Gorman, M.D. Piggott, P.E. Farrell, M.O. Bernabeu and J. Pitt-Francis. *Anisotropic mesh adaptivity for cardiac electrophysiology.* Procedia Computer Science, vol. 1, no. 1, pages 935 – 944, 2010. {ICCS} 2010.

[Speares 1997] W. Speares and M. Berzins. *A 3D unstructured mesh adaptation algorithm for time-dependent shock-dominated problems.* Int. J. Numer. Meth. Fluids, vol. 25, pages 81–104, 1997.

[Spiteri 2002] R.J. Spiteri and S.J. Ruuth. *A new class of optimal high-order strong-stability-preserving time discretization methods.* SIAM J. Numer. Anal., vol. 40, no. 2, pages 469–491, 2002.

[Steger 1981] J.L. Steger and R.F. Warming. *Flux vector splitting of the inviscid gasdynamic equations with application to finite-difference methods.* J. Comput. Phys., vol. 40, pages 263–293, 1981.

[Stein 2003] K. Stein, T. Tezduyar and R. Benney. *Mesh moving techniques for fluid-structure interactions with large displacements.* Jour. Appl. Mech., vol. 70, pages 58–63, 2003.

[Tam 2000] A. Tam, D. Ait-Ali-Yahia, M.P. Robichaud, M. Moore, V. Kozel and W.G. Habashi. *Anisotropic mesh adaptation for 3D flows on structured and unstructured grids.* Comput. Methods Appl. Mech. Engrg., vol. 189, pages 1205–1230, 2000.

[Thompson 1983] J.F Thompson and Z.U.A. Warsi. *Three-Dimensional Mesh Generation from Elliptic Systems.* In Proceedings of the AIAA CFD Conference, volume AIAA-83-1905, Danvers, USA, July 1983.

[Tong 2014] X. Tong, D. Thompson, Q. Arnoldus, E. Collins and E. Luke. *Robust surface evolution and mesh deformation for three dimensional aircraft icing applications on a swept GLC-305 airfoil.* In 6th AIAA Atmospheric and Space Environments Conference, AIAA Paper 2014-2201, Atlanta, GA, USA, June 2014.

[Venditti 2003] D.A. Venditti and D.L. Darmofal. *Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows.* J. Comp. Phys., vol. 187, no. 1, pages 22–46, 2003.

[Wackers 2012] J. Wackers, G. Deng, A. Leroyer, P. Queutey and M. Visonneau. *Adaptive grid refinement for hydrodynamic flows.* Comput. & Fluids, vol. 55, pages 85 – 100, 2012.

[Wackers 2014] J. Wackers, G. Deng, E. Guilmineau, A. Leroyer, P. Queutey and M. Visonneau. *Combined refinement criteria for anisotropic grid refinement in free-surface flow simulation.* Comput. & Fluids, vol. 92, pages 209 – 222, 2014.

[Weatherill 1994] N.P. Weatherill and O. Hassan. *Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints.* Int. J. Numer. Meth. Engng, vol. 37, pages 2005–2039, 1994.

[Winslow 1963] A. Winslow. *Equipotential Zoning of Two Dimensional Meshes.* Lawrence Livemore Laboratory, California, USA, 1963. Technical Report UCRL-7312.

[Wu 1990] J. Wu, J.Z. Zhu, J. Szmelter and O.C. Zienkiewicz. *Error estimation and adaptivity in Navier-Stokes incompressible flows.* Computational Mechanics, vol. 6, pages 259–270, 1990.

[Yang 2005] Z. Yang and D.J. Mavriplis. *Unstructured Dynamic Meshes with Higher-Order Time Integration Schemes for the Unsteady Navier-Stokes Equations.* In 41th AIAA Aerospace Sciences Meeting, AIAA Paper 2005-1222, Reno, NV, USA, Jan 2005.

[Yang 2007] Z. Yang and D.J. Mavriplis. *Higher-Order Time Integration Schemes for Aeroelastic Applications on Unstructured Meshes.* AIAA Journal, vol. 45, no. 1, pages 138–150, 2007.

[Yerry 1984] M.A. Yerry and M.S. Shephard. *Automatic three-dimensional mesh generation by the modified-octree technique.* Int. J. Numer. Meth. Engng, vol. 20, pages 1965–1990, 1984.

[Zienkiewicz 1992a] 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, vol. 33, no. 7, pages 1331–1364, 1992.

[Zienkiewicz 1992b] 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, vol. 33, no. 7, pages 1365–1380, 1992.

[Zienkiewicz 1994] O.C. Zienkiewicz and J. Wu. *Automatic directional refinement in adaptive analysis of compressible flows.* Int. J. Numer. Meth. Engng, vol. 37, pages 2189–2210, 1994.

Time-accurate anisotropic mesh adaptation for three-dimensional moving mesh problems

Abstract: Time dependent simulations are still a challenge for industry, notably due to problems raised by moving boundaries, both in terms of CPU cost and accuracy. This thesis presents contributions to several aspects of simulations with moving meshes. A moving-mesh algorithm based on a large deformation time step and connectivity changes (swaps) is studied. An elasticity method and an Inverse Distance Weighted interpolation method are compared on many 3D examples, demonstrating the efficiency of the algorithm in handling large geometry displacement without remeshing. This algorithm is coupled with an Arbitrary-Lagrangian-Eulerian (ALE) solver, whose schemes and implementation in 3D are described in details. A linear interpolation scheme is used to handle swaps. Validation test cases showed that the use of swaps does not impact notably the accuracy of the solution, while several other complex 3D examples demonstrate the capabilities of the approach both with imposed motion and Fluid-Structure Interaction problems. Metric-based mesh adaptation has proved its efficiency in improving the accuracy of steady simulation at a reasonable cost. We consider the extension of these methods to unsteady problems, updating the previous fixed-point algorithm thanks to a new space-time error analysis based on the continuous mesh model. An efficient p-thread parallelization enables running 3D unsteady adaptative simulations with a new level of accuracy. This algorithm is extended to moving mesh problems, notably by correcting the optimal unsteady metric. Finally several 3D examples of adaptative moving mesh simulations are exhibited, that prove our concept by improving notably the accuracy of the solution for a reasonable time cost.

Keywords: CFD, Unsteady simulations, moving mesh, swaps, ALE, metric-based mesh adaptation, continuous mesh framework