Université catholique de Louvain
École polytechnique de Louvain
Unité de Génie civil et environnemental

Mesh adaptation with large deformations: theory, algorithms and implementation

Doctoral Dissertation presented by

Gaëtan Compère

in fulfillment of the requirements
for the degree of
Doctor in Applied Sciences

Thesis committee:

Prof. Jean-François Remacle, Université catholique de Louvain (Advisor)
Prof. Emilie Marchandise, Université catholique de Louvain
Prof. Vincent Legat, Université catholique de Louvain
Dr. Philippe Geuzaine, Center for Excellence in Aeronautics (Gosselies, Belgium)
Prof. Thierry Coupez, Ecoles des Mines ParisTech (Sophia Antipolis, France)
Prof. Grégoire Winckelmans, Université catholique de Louvain (President)

Louvain-la-Neuve, November 2009

A mes parents,

Je tiens à remercier très chaleureusement mon promoteur, le professeur Jean-François Remacle, qui m'a permis de réaliser cette thèse de doctorat, et m'a apporté une aide préciseuse tout au long de ces quatre années. Ses larges connaissances, les nombreux conseils et suggestions qu'il a apportés et son enthousiasme quotidien ont été des atouts majeurs dans la réalisation de ce travail.

Je remercie vivement les membres de mon comité d'accompagnement, les professeurs Émilie Marchandise, Vincent Legat et Philippe Geuzaine, pour leur très grande disponibilité et les conseils précieux qu'ils m'ont apportés. Mes remerciements vont aussi aux professeurs Thierry Coupez et Grégoire Winckelmans, qui ont accepté de faire partie du jury.

Je tiens à exprimer ma gratitude aux membres de l'Unité de Génie civil et environnemental, et de l'Unité de mécanique appliquée de l'UCL, pour les nombreux échanges d'idées, mais aussi pour leur convivialité et leur bonne humeur de tous les jours.

Je voudrais exprimer mes remerciements les plus profonds à ma famille et mes amis pour le soutien indéflectible qu'ils m'ont apporté tout au long du parcours. Enfin, je tiens à remercier très chaleureusement Catherine, pour son écoute et ses précieux encouragements.

Finalement, je souhaiterais remercier le FNRS, qui a soutenu mon projet de thèse en me finançant durant ces quatre années.

Contents

Introduction

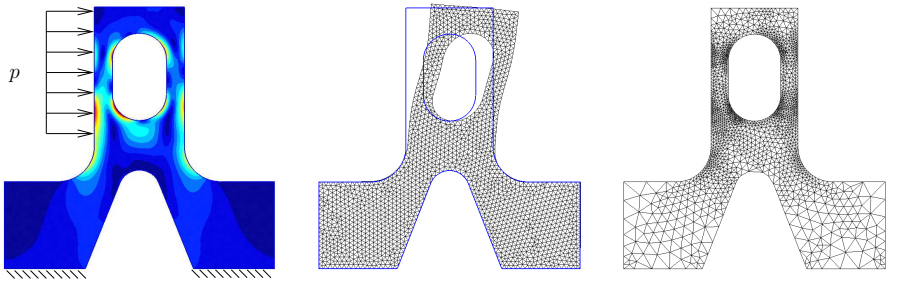
In[1] a recent presentation [92], Professor Thomas J.R. Hughes estimated that there were on the order of one million finite element analyses performed a day in engineering companies throughout the world. This has been made possible in part by computer aided design (CAD) systems, which allow engineers to represent design geometry up to manufacturing tolerances: a CAD model is usually considered as an exact representation of the geometry. However, finite element analysis also requires a computational mesh, and it is commonly admitted that about 80% of the human time spent in finite element analysis is spent in the construction of a suitable mesh [92].

In an attempt to decrease the time required for a finite element analysis, a significant effort has been devoted to the development of numerical methods that either do not require a mesh [21,31] or for which the mesh generation process is dramatically simplified [125,190]. Unfortunately, none of these methods have become a credible replacement to standard finite element analysis. On the other hand, although over the past twenty years there has been many important theoretical and practical results [176,76,55,136,112], relatively little effort is being devoted to the improvement of current mesh generation techniques: there are currently fewer than twenty research teams in the world that actively work on mesh generation and mesh adaptation research.

Although mesh generation techniques started to develop with the early developments of the finite element methods, mesh adaptation has only been considered since the 70s, when the idea that a mesh could be modified according to particular requirements of a computation first appeared. Since then, the usual role played by mesh adaptation has been to optimize the distribution of the computational resources over the domain: the mesh is refined where the solution should be more accurately computed, and coarsened where the error on the solution has a smaller impact on the quantities of interest.

The common way to decide where to refine or coarsen a mesh is to build an error estimate of the solution [1]. Goal-oriented mesh adaptation procedures [66,18] aim at reducing the global error, or distributing it uniformly over the domain. As an example, Figure 1 (a) depicts a problem in which the equation of plane strain elasticity is solved using linear finite elements. Figure 1 (b) and (c) show respectively a mesh with a uniform element size, and the same mesh on which we applied an adaptation procedure which objective is to minimize the global error with an approximately constant number of elements.

[1] Parts of the introduction have been taken from our paper [149].



(a) Von-Mises stresses, loads and fixations.

(b) Initial mesh, 3,301 triangles, deformed structure.

(c) Optimal mesh, 3,230 triangles.

Figure 1: An example of mesh adaptation for reducing the global error with an approximately constant number of elements (see [149] for details).

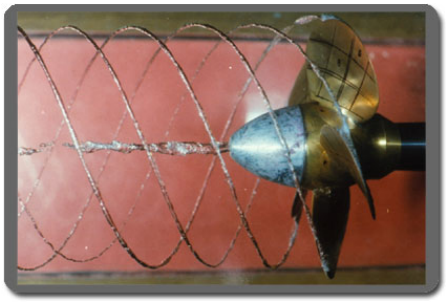
More details can be found in [149]. A simple error procedure [193] has been used for estimating the error. In this example, for an equivalent number of elements, the energy norm of the error is reduced by a factor 2.5.

Because it was a critical issue in this field, mesh adaptivity has mainly been developed in the framework of CFD (Computational Fluid Dynamics) with the goal of improving the efficiency of the existing methods. Refining, coarsening and optimizing locally the mesh was found to be an interesting mean to reduce the computational cost of the very expensive CFD computations while increasing the range of scales that can be captured in a simulation: the size of the elements can be reduced where the small scales appear, and enlarged elsewhere. Mesh adaptivity has also proved to be an interesting technique for capturing discontinuities like shocks [152, 71], or interfaces between fluids [44].

In parallel to the error control, a particular field of applications for mesh adaptivity has arisen: the computations with deforming domains. With applications in forging, machining and other industrial processes [39, 82, 137, 29], fluid-structure interaction problems in aeronautics like blade fluttering [174], crash simulations [33,101] or insect flights [146], and other fields like multiphase flows with interface tracking [54], flows in moving machines like pumps or pistons [85,5] and crack propagation [138,160] among others, the mesh adaptation methods have greatly extended their potential. Recently, the biomedical community has shown an increasing interest in mesh adaptation for deforming domains, in particular for modeling the human cardiovascular system [165,40,163], or medical devices like blood pumps [8].

The issues related to deforming domains can be partly covered by the node repositioning techniques or *r-adaptation*, as for instance the very popular elastic analogy proposed by Tezduyar et al. [186]. However, the r-adaptation methods suffer from intrinsic limitations: if the deformation of the mesh is too large, the quality of the elements is degraded, and at some point the r-adaptation techniques even fail in returning a valid mesh.

(a) Swimming man



(b) Flow behind a helix

(c) Hummingbird flight

(d) Multiple wind turbines

Figure 2: Example applications for which a computation may require deforming domains.

A classical solution to overcome that issue is to re-mesh the entire domain with for instance an advancing front technique [136], a Delaunay kernel [27], or a volume minimization principle [52] when the quality or size distributions of the mesh become insufficient. An advantage of this approach is that the geometry of the domain is implicitly taken into account in the meshing process. On the other hand these methods imply a mesh-to-mesh projection of the solution over the entire domain every time the domain is re-meshed. Another drawback of the this approach is that the mesh generation in parallel is very challenging. Today, parallel mesh generation is still an open problem. In order to circumvent those issues, methods that adapt the mesh locally have been proposed. The investigations achieved in the present manuscript deal with this class of methods. The re-meshing and local mesh adaptation methods are usually termed *h-adaptive* methods.

There exist a small number of other approaches to handle large mesh deformations, among which the sliding mesh techniques [123]. Compared to local mesh modifications, they have the disadvantage of requiring a flux computation at the interface and working with non coinciding meshes. Furthermore, this approach is not applicable if a sliding surface cannot be a priori determined.

We also mention the overset grid methods which yielded results in [107, 144] but they require grid assemblies and donor cell search processes. The local mesh modification technique has a unique mesh and a single prescribed boundary motion, which leads to a conceptually simple method and robust mesh movements.

Despite the need for efficient and reliable mesh adaptation methods, few computational packages include 3D mesh adaptation tools, and only a small number of groups are working on adaptive methods in the scientific community in comparison to the efforts made in the field of finite elements and finite volumes methods. We think that one of the reasons is the lack of freely available software sources, although the meshing field is very technical and the efficiency of the solutions usually depends on the proposed algorithms and implementations. In the 3D meshing field, several open source packages have been developed, brought to maturation, and recognized as references in the domain, like Netgen [166], Tetgen [172] or Gmsh [78]. However, no free solution has really broken into 3D mesh adaptation up to now.

Historical review and state of the art

The mesh adaptation by local mesh modifications has been under development since the 1970s with the first developments of Babuska [11] and those of Berger, Oliger and Colella [23, 22] who developed a block-structured grid method to solve compressible [22,135,122] and incompressible flows problems [158]. Block-structured grid methods allow to increase the resolution with multiple levels of refinements on blocks located near regions where the scales cannot be resolved by the coarse grid. This technique is quite appropriate to solve problems on uniform grids, as well as the quadtree based methods.

The quadtree grids are more general and more efficient than the block-structured grids in the sense that additional points are only added where it is required. Nevertheless, quadtrees require complex data structures which results in complicated algorithms and impacts on the computational time. Furthermore, the approach is not suitable for anisotropic meshes and leads to very complex algorithms when turning to parallel computations. Examples of applications of the quadtree methods can be found for the Euler incompressible equations [141], two-phase flows computations [115], and volume-of-fluid methods [80].

More appropriate methods for unstructured grids were first designed to perform local refinements with a control on elements quality by ordered edge splits, like the longest-edges refinement for triangulations [155, 156, 157], the successive bisection procedure controlled in terms of affine transformation [108, 110, 111], the reg-green algorithms [97, 26, 25], and the newest vertex bisection algorithm [124].

These methods were incremented by coarsening algorithms which consist in undoing the refinement modifications [155, 147, 113, 97, 176]. The disadvantage of these methods is that they tend to over-refine the mesh and provide only a limited control on elements quality. Furthermore, the coarsening is limited to the initial mesh size.

The next generation of methods included general coarsening operators, the edge and face swaps and the node relocation in the set of local modifications in order to improve the control on mesh quality [94, 32, 55, 95]. A particular issue with the adaptation methods based on local mesh modifications is the apparition of *slivers*, i.e. elements with a poor quality, but edges with reasonable lengths. Such undesirable elements can be generated by refinement or coarsening operations, or result from vertex relocations. Techniques to eliminate the sliver elements and optimize element shapes were progressively added to the global procedures [70, 15, 106, 48].

The combination of the different modifications in an adaptation method governed by criterions on element size and quality distributions were set up for triangular meshes in [36, 34, 62] and later for tetrahedral meshes in [133, 13].

In [106], the classical modifications were combined with a refinement procedure cutting all long edges at once and applying refinement templates for a tetrahedron with one to six split edges. This approach was preferred to individual edge splits and Delaunay insertions since, according to the authors, it minimizes over-refinement, does not create sliver elements and is relatively efficient. A drawback of template refinements is that they introduce non-tetrahedrizable polyedra (Schöenhart polyedra) so that the introduction of unwanted extra points (Steiner points) is mandatory.

Isotropic and anisotropic mesh adaptation procedures based on the previous local mesh modifications and governed by element sizes and shape criterions have been widely applied in fluid mechanics. A non-exhaustive list of applications includes shallow water problems [154], compressible flows [185, 152, 71, 3], blood flows [127, 163] and multi-phase flows [54, 44].

Extensions to the local mesh modifications methods to comply to a possibly non-manifold CAD model have been proposed recently by Shephard, Li et al. [105, 170]. In particular, algorithms to snap vertices on the exact location returned by the CAD model have been introduced. However, the proposed methods are based on complex procedures and the robustness of the methods can still be challenged.

The first achievements made to supply to r-adaptive methods with the local mesh modifications consisted in applying refinement/coarsening procedures according to both shape and deformation measures of the elements [14, 13]. The robustness of the method was improved recently by adding edge and face swaps to eliminate sliver elements [38, 54].

In parallel with these developments, another approach using a local remeshing based on a minimum volume principle was proposed by Coupez et al. [52] and mostly applied in the field of forging and manufacturing. This approach has the advantage of starting from a mesh which can be non conform to the geometry of the domain. The method was parallelized in [51] and applied to multi-domain problems in [81].

We also mention another class of methods based on the minimization of a spring-like energy function associated with each edge and dependent of its length and the local scales of the flow [53, 191, 6]. The method is combined with local node repositionings and edge swaps in order to improve element shapes.

Recently, a fixed point procedure for steady and unsteady computations was proposed by Alauzet, Frey et al. [3] to yield a convergence for both the adaptive mesh and the CFD solution together.

From the implementation point-of-view, only a few packages have been proposed as open source libraries for mesh adaptation in 3D. We mention the mesh adaptation module of the Gerris flow solver [142] and the libMesh [100,99] library. However, these packages are based on quadtrees and only provide refinement/de-refinement procedures which is far behind the current state of the art of mesh adaptation.

Objectives of the present work

Today, many problems related to mesh adaptivity remain open. Finite element and finite volume methods could largely benefit by proposing solutions to the remaining issues. Among the challenges that are still to be taken up, we mention the adaptation for curvilinear meshes, the anisotropic mesh adaptation with high aspect ratios, the gradation control in highly anisotropic meshes, the automatic generation and adaptation of boundary layer meshes, the handling of complex CAD models, issues in mesh optimization like fully robust sliver elimination, and adaptivity for very large meshes.

In the present work, we address some of these issues, in particular those related to the mesh adaptation with large domain deformations. Some key points for the applicability of the method to industrial or biomedical problems are reached, like CAD models handling, automatic boundary layer meshing, element shapes optimization or computational efficiency.

Mesh adaptivity applied to two-phase flows Although the first contribution presented in this work [44] is not strictly speaking intended to be applied to large deforming domains, it evaluates the applicability of the mesh adaptation method on which the present work is based to a class of highly transient problems: two-phase flow computations.

An interface capturing method is considered here for modelling the interface between the fluids. With interface capturing methods, the position of the interface is not directly related to the position of some mesh entities. In our case, the iso-zero surface of a level set function is used to locate the interface.

The presented mesh adaptation procedure is used to maintain a refined zone around the interface in order to capture the small scales, evaluate more accurately the surface tension forces, and limit the mass losses inherent to the computational method. The benefit in terms of efficiency coming from anisotropic elements aligned with the interface is also evaluated.

Arbitrarily large deformations It is well recognized in the literature [13, 38, 54] that the robustness is a key element of the mesh adaptation procedures for large deformations since a single ill-shaped element can be responsible for a global failure of the method.

Recent results allow to perform large deformations on relatively complex geometries in particular fields: multiphase flows with interface tracking [54],

and separation of objects in an external flow [38]. However, when arbitrarily large deformations occur, existing methods can still fail in providing a valid mesh.

In this work, we consider both large deformations [48, 47] and complex geometries [45], and we address the issues related to the robustness of the method, as well as the control on element quality. A particular attention is paid to the sliver elimination. The objective is not to target a particular application but rather to provide the general techniques that will be able to face any motion of the domain.

Automatic boundary layer meshing Among the issues that are raised when generating a mesh, one of the most time-consuming task is certainly the generation of a suitable mesh in the regions where a boundary layer (BL) has to be captured. For fixed domains, techniques exist to generate structured BL meshes in a semi-automatic way. Those techniques usually extrude the surface mesh several times, which yields successive layers of structured elements. Several problems may arise with this approach. First, particular techniques have to be designed for the different geometrical cases in which a simple extrusion of the surface mesh yields an invalid or low quality mesh. The resulting elements can have poor qualities in those particular regions, and sometimes the intervention of an experimented user is required. Secondly, such a mesh is not suitable when dealing with large deforming meshes: when the boundaries undergo large deformations, preserving the quality and above all the validity of a semi-structured mesh is a challenging task.

In a previous work [161] it has been shown that unstructured BL meshes yield acceptable results for viscous flows, although the solution exhibits spurious oscillations that do not appear with structured meshes. In the present work, an additional test is performed to evaluate the potential of unstructured anisotropic meshes for capturing BL at low Reynolds numbers.

From the mesh adaptation method proposed for the large deforming domains [48, 47], an anisotropic unstructured BL mesh adaptation method can be derived by building the appropriate anisotropic size field representing the desired sizes in the BL and intersecting it with the other prescribed size field(s). The advantage of such an approach is that the adaptivity of the BL mesh and the interior mesh are gathered under a single adaptation procedure via size field intersections. The BL meshing is then automatic, provided that the construction of the size field in the BL region is also automatic. Such an approach is well suited for generating a BL mesh in an existing mesh, and adapting it through the computation whether the domain is deformed or not.

The automatic construction of the size field in the BL can be obtained from the determination of the tangent and normal directions to the wall inside the BL region. In order to obtain a well-posed problem for the mesh adaptation procedure, the anisotropic size field of the BL has to be computed carefully. Around curved walls, the curvatures of the boundary restrict the possibilities of building elements with large anisotropic ratios, simply because highly anisotropic elements cannot be piled up and fill the space in a curved region. Furthermore, a curved boundary should be discretized by several elements in

order to capture the non-uniform flow. The present work addresses this issue and describes the unified method for generating and adapting the mesh in both the interior domain and the BL region.

Geometrical model handling When the mesh adaptation methods are used in the context of industrial computations, the domain discretized by the mesh is usually defined by a Computer Aided Design (CAD) model. Ensuring the compatibility of the mesh to the CAD model is therefore an important task when running an adaptation method since successive refinement, coarsening and optimization operations can dramatically alter the shape of the boundaries of the mesh if special care is not given to the compliance with the model. A simple way to preserve the shape of the domain is to deny mesh modifications on the boundaries, thus disabling any possibility to refine/coarsen or optimize the boundary elements. This solution is very restrictive in most cases since interesting phenomenons can occur at the vicinity of the boundaries, like travelling shock waves or boundary layer separation.

Adapting the mesh of a complex domain with an automatic compliance to a CAD model is a challenging task in which robustness and elements quality are key points. Most of the methods based on global re-meshing implicitly include the compliance to the model since the volume mesh is built on the surface mesh of the model boundaries. For the methods based on local mesh modifications, special considerations have to be made when designing the mesh modifications. In particular, the creation of new boundary nodes requires to snap the nodes to their corresponding location on the model boundary, which yields the issue of the quality, indeed even volume positivity, of the neighbor elements. Coarsening and swap modifications are also to be considered since they may yield a mesh which is no longer compatible with the CAD model. For instance, an invalid coarsening would put in contact two surfaces of the model which were initially separated.

New developments are made in the present work [45] to address these issues. The aim is to state in details the constraints brought by the handling of a CAD model in a mesh adaptation procedure by local modifications and to provide new techniques that improve the reliability of the procedure.

Mesh adaptation platform A contribution of the present work is the development of an open source mesh adaptation platform: MAdLib (**M**esh **Ad**aptation **Lib**rary) [46, 47] which implements the achievements proposed in this work.

The mesh adaptation field is very technical in the sense that the resulting meshes depend on the algorithms and implementations. Some algorithms are based on heuristics, like for instance the sliver elimination procedures. Furthermore, the efficiency of an algorithm can be dramatically altered if some considerations about its implementation are not taken into account, like in the research of the optimal edge swap configuration for instance.

We think that providing such an open code, contributes to the research in the field of numerical mechanics for the following reasons:

- We believe that adaptive procedures have not reached a sufficient impact in engineering design and that one of the main reasons of that relative

success is that there are too few freely available solutions. Despite the presence of a literature in the mesh adaptation field, many resources must be invested by a research or industrial group to obtain a robust and efficient 3D mesh adaptation tool. This investment is often discouraging when a group wishes to improve a mesh adaptation method, or use it in an industrial project or in a third-part research.

- The repeatability of the numerical experiments is hard, indeed even impossible to ensure when the original implementation is not available. It is therefore very hard to compare different methods, or to experience the effect of modifications in the existing methods.

- Mesh related codes need time and users to become usable. By distributing a source code, we hope to build a community around it, which may allow for improvements in the code thanks to the comments and contributions of the community, return on the method when applied in different fields of application, and new ideas or suggestions for development lines around the proposed package. Such a community has already been built around other platforms in the field of mesh generation like for Gmsh [77].

Outline

The outline of the present work is as follows. The first chapter describes the mesh adaptation procedure and its application to two-phase flow computations. To this end, the paper entitled *Transient adaptivity applied to two-phase incompressible flows*, Compère, Marchandise, Remacle, Journal of Computational Physics (2008) is presented. In Chapter 2, the techniques proposed to extend the method to arbitrarily large domain deformations are discussed and the paper *Transient mesh adaptivity with large rigid-body displacements*, Compère, Remacle, Marchandise, Proceedings of the 17^{th} International Meshing Roundtable (2008) is proposed. The investigations performed for the problems related to the boundary layer meshes are discussed in Chapter 3, and the handling of CAD geometries is detailed in Chapter 4, where the paper entitled *Mesh adaptivity complying to a geometrical model*, Compère, Remacle (not submitted yet) is presented. Finally, a discussion on source code distribution and the description of the mesh adaptation package MAdLib are given in Chapter 5, which reproduces the paper *A mesh adaptation framework for dealing with large deforming meshes*, Compère, Remacle, Jansson, Hoffman, accepted for publication in the International Journal for Numerical Methods in Engineering.

Chapter 1

Mesh adaptivity by local mesh modifications: an example application

This chapter describes the first achievement of the present work, which consists in applying an existing mesh adaptation method to two-phases flow simulations. This is the issue reached in Article I, *Transient adaptivity applied to two-phase incompressible flows*, Compère, Marchandise, Remacle, J. of Comp. Phys. (2008) presented here after. In this framework, the computational domain remains unchanged along the computation.

The main interest of this chapter is to introduce the mesh adaptation by local mesh modifications, which will be the basis of the rest of the work, and to evaluate its contribution to an example application: highly transient two-phase flows.

The mesh adaptation method is described in details in [104, 106]. It has already proved to be appropriate to adapt meshes on fixed domains [152, 154]. The main ingredients of the method are

- A mesh size field, which is the way to specify the desired (non-homogeneous) length of the edges over the mesh. It can be isotropic or anisotropic. The size field allows to classify an edge e as *long*, acceptable or *short* according to the ratio between the prescribed and the real length of e. A rigorous definition of long and short edges can be found in Article I.
- The local mesh modification operators, which goal is to replace a cavity $\mathcal{C}$ (a set of connected elements) of the mesh by another cavity $\mathcal{C}'$ with the same boundary as $\mathcal{C}$.
- A sliver elimination procedure. A sliver element is an element with a poor quality and no short or long edge.
- A global procedure linking the ingredients together in order to produce an optimal mesh, i.e. a mesh that satisfies the mesh size field and with element shapes that comply to some criterions.

The application proposed is the two-phase flow modeling with high density and viscosity ratios between the fluids. In this work, an interface capturing method based on level set functions [119] is used. In this method, the interface is given by the iso-zero of a level set function which is advected based on the fluid velocities, contrary to the interface tracking method, in which the interface is given by a set of mesh entities, thus constraining the mesh to follow the motion of the interface. Here, despite the mobile interface, the meshed domain is fixed.

A drawback of this method is the mass transfer between the fluids, which results from an inaccurate discretization of the fluid variables and level set function at the interface.

Here, we use the mesh adaptation procedure to produce a mesh which is fine at the vicinity of the interface, and relatively coarse elsewhere, thus concentrating the computational resources on our zone of interest: the interface. The objective is to capture smaller scales of the flow, which reduces the mass losses, and evaluates more accurately the surface tension forces. We also evaluate the benefit of using slightly anisotropic elements aligned with the interface.

Article I

Transient adaptivity applied to two-phase incompressible flows

Gaëtan Compère[1,3*], Emilie Marchandise[1,3], Jean-François Remacle[1,2]

[1] *Université catholique de Louvain, Department of Civil Engineering, Place du Levant 1, 1348 Louvain-la-Neuve, Belgium*

[2] *Center for Systems Engineering and Applied Mechanics (CESAME), Université catholique de Louvain, 1348 Louvain-la-Neuve, Belgium*

[3] *Fonds National de la Recherche Scientifique, rue d'Egmont 5, 1000 Bruxelles, Belgium*

Abstract

An anisotropic adaptation process is applied to a three-dimensional incompressible two-phase flow solver. The solver uses a level-set/finite-element method on unstructured tetrahedral meshes. We show how the level set function can be used to build an anisotropic mesh with good properties. Some computations with a strong transient character and large densities ratios (1/1000) are presented. We show that the efficiency of the computations can be deeply enhanced by mesh adaptations.

Key words: Adaptivity, Anisotropic, Two-phase, Finite elements, Level set

I.1 Introduction

Among all the problems that have to be addressed in the numerical simulations of two-phase flow problems, obtaining an accurate representation of the interface is certainly the most difficult issue. At least two arguments lead to this observation:

- most of the fluid vorticity is concentrated near the interface [180],
- the rapid variation of the fluid properties generates spurious oscillations in the velocities close to the interface [7].

Those issues are even more critical in the presence of large density and/or viscosity jumps across the interface. This is indeed the case in many practical applications, and among those are the ones that involve interfaces between air and water. There, the density ratio is about 1/1000 while the viscosity ratio is about 1/100.

In previous works [120, 119, 118], our team has developed a technique to accurately model three dimensional two-phase flows. A level set [132, 181, 119]

approach was chosen to model the interface[1]. The level set function $\phi(\mathbf{x}, t)$ is an implicit function that is defined on the whole mesh. Its iso-zero $\phi_0 \equiv \phi(\mathbf{x}, t) = 0$ is a surface that evolves in time and that represents the fluid interface. The property of the fluid at one point $\mathbf{x}$ is identified by the sign of the function. The level set function ϕ is transported and deformed by the fluid flow. An equation of transport is solved at each time step that allows to move the fluid interface [120]. The level set method is well suited to represent complex flows with dramatic changes in the interface topology. Applications of level set in two-phase flow calculations have been extensively described by Sussman, Smereka and Osher in [184, 182, 181] and used by [42, 41, 131, 129] among others.

This paper starts from the following observation: in two-phase flows problems, the denser is the mesh at the vicinity of the interface, the better is the solution. In other words, an adaptive strategy is required in order to put the effort in the region where the solution is more complex.

Adaptive mesh refinement (AMR) has now been used for two decades in flow problems. The first achievements are those of Berger, Oliger and Colella [23, 22] who developed a block-structured grid method to solve compressible [22, 135, 122] and incompressible flows problems [158]. Those methods have been more recently applied to two-phase flow problems by Sussman et al. [180, 179]. Block-structured grid methods allow to increase the resolution with multiple levels of refinements on blocks located near regions where the scales cannot be resolved by the coarse grid. This technique is quite appropriate to solve problems on uniform grids, as well as the quadtree based methods. The quadtree grids are more general and more efficient than the block-structured grids in the sense that additional points are only added where it is required. Nevertheless, quadtrees require complex data structures which results in complicated algorithms and impacts on the computational time. Results of the coupling of the Euler incompressible equations with the quadtree method have been presented by Popinet [141]. Two-phase flows computations were first performed by Losasso [115], and recently Greaves proposed a coupling with a volume-of-fluid method [80]. Finally, we mention the methods based on the minimization of a *mesh energy* used by Cristini et al. [53, 191, 6]. In those methods, a spring-like energy is associated with each edge and is dependent of its length and the local scales of the flow. A more detailed review of the AMR techniques coupled with level set methods and incompressible flows can be found in [114].

In this paper we describe an alternative method to the techniques presented above. Its simplicity allows robust anisotropic mesh adaptations [152, 154]. Because our aim is to adapt the mesh in time, we use an adaptive procedure that is based on local mesh modification operators. Starting from an existing mesh, we apply local mesh modification operators until each edge of the resulting mesh has a non-dimensional length that is close to 1. Non-dimensional edge lengths are computed using a non uniform anisotropic metric field that is based

[1] Level set and volume of fluid [87,139,178] can be classified as interface capturing methods. They differ from interface tracking methods [86, 91, 63, 58, 83] in the sense that interface tracking methods use an explicit representation of the interface.

on the position of the interface. An interest of using anisotropic elements is that the high variations of density and viscosity across the interface can be captured in a more efficient way by having more elements along the crossing direction. As a consequence, mesh adaptation can be seen as an alternative to the common solution consisting in smoothing the fluid properties on a thin layer around the interface [182, 129]. In the same way, the high pressure gradients generated at the interface in presence of surface tension forces can be evaluated more efficiently with anisotropic elements, which reduces spurious velocities.

The remainder of this paper is organized as follows. In section 2, the physical model is described as well as the numerical methods used in the flow solver. Section 3 presents the adaptation technique while section 4 gives numerical examples that show the accuracy and the efficiency of the method.

I.2 Two-phase flow computation

In this section, we give a rapid summary of the different ingredients that are used in the simulations: constitutive equations, numerical methods and coupling between the fluid and the interface solvers. For more details of the recipe, see [119].

Constitutive equations

Our solver computes three-dimensional laminar flows involving two incompressible non-miscible fluids. The fluids are identified by $(+)$ and $(-)$, and their density and viscosity are respectively (ρ_+, μ_+) and (ρ_-, μ_-).

The physics of the fluids are given by the two-phase incompressible Navier-Stokes equations:

$$\frac{D\mathbf{u}}{Dt} = -\frac{\nabla p}{\rho(\phi)} + \frac{1}{\rho(\phi)}\frac{1}{Re}\nabla\cdot(2\mu(\phi)S) + \frac{\mathbf{e_g}}{Fr^2} + \frac{\kappa\mathbf{n}}{We} \tag{I.1}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{I.2}$$

where $\mathbf{u}$ and p are the non dimensional velocity and static pressure of the fluids, $\rho(\phi)$ and $\mu(\phi)$ are the non dimensional density and dynamic viscosity, $S = \frac{1}{2}\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right)$ is the deformation rate tensor, $\mathbf{e_g}$ is the direction in which the gravity $(\mathbf{g})$ acts, κ is the curvature of the interface and Re, Fr and We are the numbers of Reynolds, Froude and Weber defined as

$$Re = \frac{\rho_R U_R L_R}{\mu_R}, \quad Fr = \frac{U_R}{\sqrt{g L_R}}, \quad We = \frac{\rho_R U_R^2 L_R}{\sigma_R}, \tag{I.3}$$

where the subscript $_R$ denotes the reference value and σ is the surface tension. The solutions in both phases are obtained simultaneously.

The interface is captured using a level set method. This method implies a function ϕ named the *level set* whose value is 0 on the interface (ϕ_0), positive in one fluid and negative in the other. Complex configurations like sloshing problems with bubbles, separations and merging can then be represented.

This function is advected using a tracer equation that can be written in a conservative form:

$$\partial_t\phi + \nabla\cdot(\mathbf{u}\phi) = 0\,. \tag{I.4}$$

Further details about the level set method in two-phase flow problems can be found in [184, 119].

Numerical method

Fluid flow: We use a finite element method for computing the fluid flow (Eqs. (I.1)-(I.2)). Both velocity and pressure are approximated using a piecewise linear continuous polynomials. As this representation violates the Babuska-Brezzi (BB) condition [10], spurious pressure modes have to be removed from the solution. These oscillations can be avoided using a pressure stabilized Petrov Galerkin (PSPG) method. The inviscid fluxes are discretized in a stable manner using an upwind finite volume stabilization [19, 168].

A second-order three-point backward difference scheme is employed for the time-integration. An inexact Newton method based on a finite difference Newton-Krylov algorithm [79] is used to solve at each time step the system of nonlinear equations. The iterative solution of the large sparse linear systems that arises at each Newton iteration is solved by the GMRES method preconditioned by the RAS [35] algorithm.

Level set advection: The interface equation (I.4) is discretized using discontinuous finite elements. The solution is represented using piecewise discontinuous polynomials (N^p) of order p. The discontinuity of this numerical approximation is not to be confused with the discontinuous nature of the solution of two-phase flows. In other methods like in the ghost fluid method [67] the physical discontinuity is exactly fitted by a discontinuity in the model and a specific treatment of the interface is designed to handle the jump conditions. This specific treatment leads to the advantage of very small spurious velocities. Here, the interface is represented by the iso-zero of a continuous level set function which is *discretized* in a discontinuous way that is not related to the physical discontinuities. The surface tension forces are computed according to the *continuum surface force* method of Brackbill et al. [30]. Further details about the computation of the surface tension forces can be found in [118]. The discontinuity of the physical properties of the fluid (density and viscosity) are handled by an exact discontinuous integration over the elements overlapping the interface [175, 187].

As Equation (I.4) involves $\nabla\phi$ and $\mathbf{u}$, we should use a discretization of the level set for which the gradient is at least in the space of the velocity, i.e. $p \geq 2$. Nevertheless, as it is shown for the two-dimensional dam break problem presented in the last section of this paper, the use of first order polynomials is often more efficient if we use an adaptation procedure. Indeed, for the same computational time, we can build a finer mesh near the interface with a first order discretization than with a second or a third. The difference in terms of accuracy due to the order of discretization will be exceeded by the level of refinement of the mesh.

A Runge-Kutta algorithm of order $p+1$ is employed for the time discretization. More details about the time and space discretization of the level set function can be found in [120].

Coupling of the two solvers

The coupling between the fluid and the interface solvers requires a special attention as the discretization differs. The level set function is discretized by using discontinuous p-order elements, while the flow solver uses continuous linear approximations (N^1) for the velocity $\mathbf{u}$

$$\mathbf{u} = \sum_{i=1}^{4} \mathbf{u}_i N_i^1 \quad \text{and} \quad \phi = \sum_{i=1}^{n_p} \phi_i N_i^p,$$

where n_p is the number of Lagrangian points in each tetrahedral element

$$n_p = (p+1)(p+2)(p+3)/6.$$

For efficiency reasons, the same mesh is used by both solvers.

The algorithm that couples the solvers is summarized as follows. At initial time t^0, initialize the level set ϕ. Then, for each time t^n, $n = 1, 2, \ldots$

1. Solve the Navier-Stokes equations in time $t \in [t, t+\Delta t]$ to find $\mathbf{u}(t+\Delta t)$ and $p(t+\Delta t)$.
2. Project the velocity field onto the degrees of freedom of the level set.
3. Solve the interface equation in time $t \in [t, t+\Delta t]$ using sub-time steps and linear interpolations of $\mathbf{u}$ between $\mathbf{u}(t)$ and $\mathbf{u}(t+\Delta t)$ to find $\phi(t+\Delta t)$.
4. Project the level set function onto the degrees of freedom of the velocity.
5. Increment in time $t = t + \Delta t$ and go back to step (1).

I.3 Mesh adaptation

The appropriate way to ensure that a mesh-based numerical analysis procedure produces the most effective solution results is to apply an adaptive solution strategy. Efforts on the development of these techniques have been underway for over twenty five years.

In this work, a local approach is used for adapting the mesh. The size field that is used for building the optimized mesh is build up using principally the fluid interface data.

Adapting the mesh using local mesh modifications

It is only recently that authors have developed adaptive methods for transient problems that are able to be applied to unstructured grids. Transient adaptive simulations require to modify the mesh in time. For that purpose, two approaches are possible. The first one requires to build a new mesh any time the mesh has to be adapted. [4,27,164]. In those global re-meshing techniques, the issues related to mesh to mesh interpolation are critical. In order to control the mesh interpolation errors, Alauzet et al. [3] were able to reduce the number of re-meshings using a fixed point algorithm. A metric field $\mathcal{M}$ is computed using the intersection of metric fields at successive times $t_1, t_2, \ldots, t_n$. The simulation

is rewound at time t_1 and the mesh is adapted against metric field $\mathcal{M}$. The adapted mesh is valid for a large time interval and it is therefore adapted less often.

The other way of doing the adaptation in time is to locally modify the mesh [104, 152, 154, 60]. A common belief is that doing local mesh modifications has to be faster than re-meshing. This is not usually the case. Global re-meshing procedures are in fact usually faster than local mesh modifications techniques. This is the case essentially because global re-meshing algorithms converge much faster: they allow to create vertices that are readily at their right locations when local mesh modification procedures iteratively add and remove vertices in the mesh. For transient adaptive computations, local mesh modifications have determinant advantages that are not linked with their computational efficiency:

- local solution projection procedures can be easily set up that ensure the exact conservation of conservative quantities [152, 154],
- the mesh remains unchanged in most of the domain, allowing to adapt the mesh frequently,
- local mesh modifications can be performed in parallel [61], enabling transient adaptive simulation to run on parallel computers.

Local mesh modification operators all consist in replacing a cavity of elements $\mathcal{C}$ by another one $\mathcal{C}'$. In our adaptation procedure, we have set up a moment when both cavities $\mathcal{C}$ and $\mathcal{C}'$ are simultaneously present. At this point, both fluid and interface solvers are called back so that a local solution projection procedure can be performed. The solution in the new cavity $\mathcal{C}'$ is computed using the information in $\mathcal{C}$. More details about this mesh adaptation procedure can be found in previous papers [152, 154, 24, 104].

As the level set is represented by discontinuous polynomials, projections are made at the elementary level. This is one of the major advantages of the DG method. The fluid solution, i.e. velocities and pressures, uses continuous piecewise linear approximations. Both fields are located at mesh vertices. In our projection algorithm, the only local mesh modification that requires some work is the edge splitting. When a new vertex is inserted in the mesh, the fluid solution is simply taken as the average of the solutions at the two nodes of the initial edge.

The mesh metric field

A mesh metric field is a smooth tensor valued $\mathcal{M}(\mathbf{x})$ defined over the domain. The metric at a point is a symmetric positive definite tensor. Let us consider a mesh edge e that defines a vector $\mathbf{e}$ that goes from its initial vertex to its final one. The non-dimensional length L_e^{tr} of e is computed as

$$L_e^{tr} = \int_e \sqrt{\mathbf{e}^t \, \mathcal{M}(\mathbf{x}) \, \mathbf{e}} \; dl. \tag{I.5}$$

The aim of the mesh adaptation procedure is to modify an existing mesh to make it a unit mesh, i.e. a mesh for which every edge is of size $L_e^{tr} = 1$.

The use of a tensor valued metric field allows the construction of anisotropic meshes. More details about metrics and metric-based operations are given in Appendix A.

Local mesh modification operators are essentially edge-based operators in the sense that they locally modify mesh cavities composed of all the tetrahedrons. The mesh adaptation algorithm works as follow.

(1) Loop over all edges of the mesh, consider the edge e of size L_e^{tr}

- Refinement : if $L_e^{tr} > \sqrt{2}$, e is a long edge. Edge e is therefore split in its middle, according to the metric.
- Coarsening : if $L_e^{tr} < 1/\sqrt{2}$, e is a short edge. The cavity surrounding e has therefore to be investigated and a coarsening procedure [106] in which e could be collapsed by merging its two nodes is applied.
- Shape optimization : the edge e is eliminated using an edge swapping algorithm if a better configuration is obtained after swapping.

(2) Go back to (1) if any short or long edge is still present in the mesh.

(3) Do one more step of shape optimization.

Typically, the algorithm stops when every edge of the domain has a dimensionless size in the interval $L_e^{tr} \in [1/\sqrt{2}, \sqrt{2}]$. Using this interval for short and long edges ensures that the two new edges created by a bisection will not be short edges. Oscillations between coarsening operations and refinements are therefore prevented.

The metric field is computed at every node of the present mesh using the results of a procedure that is described below.

For every vertex v of the mesh,

- Both the normal vector $\mathbf{n}$ and the normal curvature κ of the level set are computed using the classical formulas

$$\mathbf{n} = \frac{\nabla \phi}{\|\nabla \phi\|}, \quad \kappa = -\nabla \cdot \mathbf{n}.$$

- Two vectors $\mathbf{t_1}$ and $\mathbf{t_2}$ are found such that $(\mathbf{n}, \mathbf{t_1}, \mathbf{t_2})$ form an orthonormal basis of R^3.
- The distance d from the vertex v to the interface is computed. This step is achieved using a fast search tree method, namely the Approximate Nearest Neighbor algorithm (ANN) [9, 126].
- Mesh sizes S_n, S_{t1}, S_{t2} are computed in the three directions $\mathbf{n}, \mathbf{t_1}, \mathbf{t_2}$ as a function of d and κ. The kind of parametrization that is used in this work is presented in the remainder of this section (§I.3).
- The metric field $\mathcal{M}$ at vertex v is computed by

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

with

$$\mathcal{R} = \left(\begin{array}{ccc} \mathbf{n}, & \mathbf{t_1}, & \mathbf{t_2} \end{array} \right), \quad \text{and} \quad \mathcal{D} = \begin{pmatrix} S_n^{-2} & 0 & 0 \\ 0 & S_{t1}^{-2} & 0 \\ 0 & 0 & S_{t2}^{-2} \end{pmatrix}. \quad \text{(I.6)}$$

Size field parametrization

Because most of the complex physics involved in two-phase flows is located near the interface, the mesh is refined at the vicinity of the iso-zero of the levelset. To be more quantitative, we introduce a distance parameter $2\,\delta$ that represents the thickness of the refined region $[\phi_0 - \delta, \phi_0 + \delta]$ around the interface i.e. the thickness of the zone that has the maximal mesh refinement. We call this region the *proximity zone*. In our algorithm, we ensure that the interface never leaves the proximity zone. When the interface gets too close to the end of the proximity zone, the mesh is modified. Figure I.1 shows schematically how the interface is kept inside the refined region. The parameter δ has to be chosen carefully. If δ is big, the number of nodes of the mesh increases and so goes the time spent in the flow solver. If δ is small, the number of mesh adaptations increases because the interface leaves the proximity zone quicker. A typical choice for δ is 5 times the element size in the proximity zone.

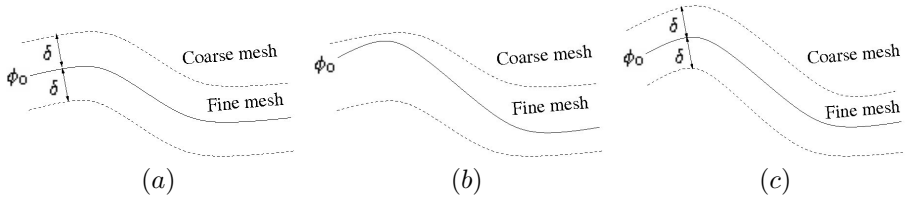


Figure I.1: Schematic view of the proximity zone during a simulation: (a) proximity zone just after a mesh adaptation procedure, (b) new position of the interface close to the boundaries after several iterations of the fluid solver, (c) new position of the proximity zone after the next mesh adaptation procedure.

Outside the proximity zone, we build an isotropic mesh of variable size $L_{iso}(d)$ (see figure I.2) where:

$$L_{iso}(d) = \begin{cases} h_0 + s_1(d-\delta) & \text{for} \quad \delta < d \leq d_1 \\ h_1 + s_2(d-d_1) & \text{for} \quad d_1 < d \leq d_2 \\ h_2 & \text{for} \quad d_2 < d \end{cases} \quad \text{(I.7)}$$

where s_1 and s_2 are defined by

$$s_1 = \tfrac{h_1 - h_0}{d_1 - \delta}, \quad s_2 = \tfrac{d_2 - d_1}{d_2 - d_1}. \quad \text{(I.8)}$$

In this setup, h_0, h_1, h_2 are user-defined mesh sizes: h_0 is the smallest size that defines the resolution of the mesh in the proximity zone, h_1 is an intermediary size between $\delta < d < d_1$ and h_2 is the mesh size in the far field. Following the

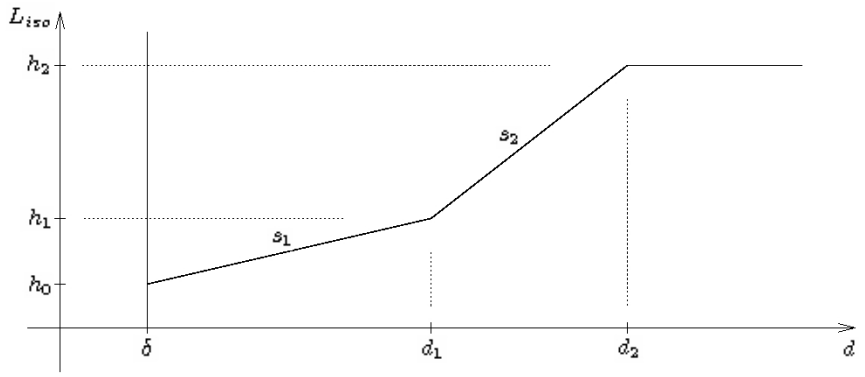


Figure I.2: Distribution of the edge sizes outside the proximity zone.

definition (I.6) of the metric, we have

$$S_n(d) = S_{t1}(d) = S_{t2}(d) = L_{iso}(d) \qquad \text{for} \quad d > \delta. \tag{I.9}$$

L_{iso} is a parameter that is similar to the length of the edges about a sphere defined in [6], except that two slopes are defined and that constant and anisotropic lengths are prescribed in the proximity zone. The length is maintained constant in this zone in order to allow the flow solver to advect the interface with a constant resolution without having to adapt the mesh at each time step.

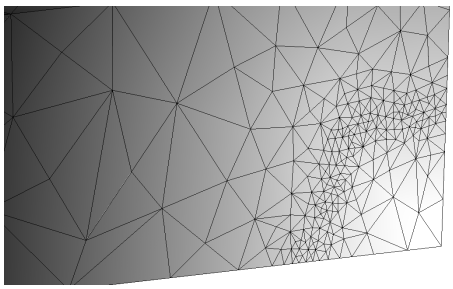
One of the advantages of the mesh refinement near the interface is the ability to reduce spurious velocity oscillations arising from a strong difference of properties between the fluids. We denote by $R_{\mathbf{n}}$ the mesh size reduction factor in the normal direction to the interface. The reference length S_n in the proximity zone is then

$$S_n = \tfrac{h_0}{R_{\mathbf{n}}} \qquad for \quad d \leq \delta. \tag{I.10}$$

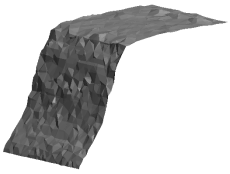
The ratio $R_{\mathbf{n}}$ is constant in time and space (in the proximity zone). Simulations were performed using isotropic and anisotropic mesh refinements. Figure I.3 shows the difference between results obtained with both isotropic and anisotropic refinements for a *dam break* simulation. More explanations about this simulation are given in the section related to the computational results. During the computation, both meshes keep almost the same number of nodes (about $2,850$ nodes in figure I.3). The mesh are adapted according to the following parameters:

- for the first mesh, $h_0 = 0.075$, $\delta = 0.1$,
- for the second mesh, $h_0 = 0.15$, $\delta = 0.05$, $R_{\mathbf{n}} = 4$,

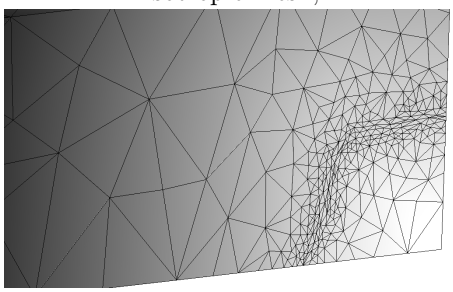
In both cases, the roughness of the interface is due to spurious velocity oscillations and the interface is clearly smoother using meshes that are adapted directionally.



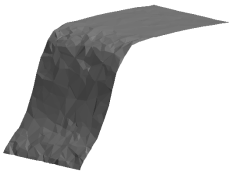
Isotropic mesh,



computed interface.



Anisotropic mesh,



computed interface

Figure I.3: Comparison of the meshes and the interfaces ϕ_0 in a *dam break* simulation after a 'small' time for isotropic and anisotropic meshes. The meshes have almost the same number of nodes (about $2,850$ nodes). For the first mesh, $h_0 = 0.075$, $\delta = 0.1$. For the second, $h_0 = 0.15$, $\delta = 0.05$, $R_{\mathbf{n}} = 4$. The other parameters are identical for the 2 simulations.

Another advantage of refining the mesh at the interface is the ability to capture smaller scales. This ability can still be improved by taking the curvature κ of the interface into account in the definition of mesh sizes [192]. To this end, we define a reducing ratio $R_{\mathbf{t}}(\kappa)$ as follows

$$R_{\mathbf{t}}(\kappa) = \min\left(R_{\mathbf{t},max}, 1 + \frac{\kappa}{\kappa_{max}}(R_{\mathbf{t},max} - 1)\right), \tag{I.11}$$

where $R_{\mathbf{t},max}$ and κ_{max} are arbitrary constants. This ratio is applied in the tangential directions $\mathbf{t_1}$, $\mathbf{t_2}$ to the interface:

$$S_{t1} = S_{t2} = \frac{h_0}{R_{\mathbf{t}}(\kappa)} \qquad \text{for} \quad d \leq \delta. \tag{I.12}$$

Figure I.4 summarizes the use of the ratios $R_{\mathbf{n}}$ and $R_{\mathbf{t}}(\kappa)$. Finally, the mesh refinement parameters are defined as:

- 'near' the interface: δ, h_0, $R_{\mathbf{n}}$, $R_{\mathbf{t},max}$ and κ_{max},
- 'far' from the interface: h_1, h_2, d_1 and d_2.

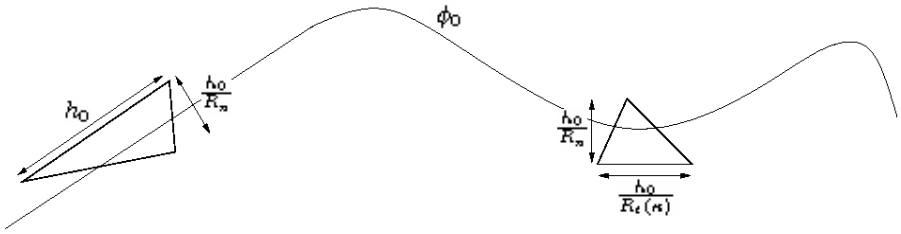


Figure I.4: Schematic view of the anisotropic aspect ratios $R_{\mathbf{n}}$ and $R_{\mathbf{t}}(\kappa)$ used near the interface.

I.4 Computational results

Our two-phase flow solver has already been validated using fixed meshes [119]. The two first tests check that the spurious velocities are reduced when a finer refinement is applied to the mesh. They also show that the anisotropic refinements exhibit smaller spurious currents than the isotropic ones. The next test checks that the mesh adaptation procedure at least does not deteriorate the solution of the fixed mesh solver. More complex simulations are then presented to show that the method can be applied to complex and highly transient flows.

In order to make comparisons between adaptive and non-adaptive computations in terms of computational time, we have to choose an edge length h_{na} for the non-adaptive mesh such that a similar accuracy is obtained with both approaches. As the non-adaptive meshes are isotropic, we define a criterion for the meshes to give similar results. We make the assumption that the accuracy at the interface ϕ_0 only depends of the size of the edges in the normal direction to the interface, and we obtain h_{na} as follows:

$$h_{na} \approx \frac{h_0}{R_{\mathbf{n}}}. \tag{I.13}$$

Two effects are not taken into account with this criterion.

- First, with the adaptive technique developed in this paper, the mesh is coarse far from the interface whereas it is homogeneous on the whole domain for the fixed mesh. The modeling of the flow can then be slightly different.
- Secondly, the positive effects of anisotropy on the parasitic velocities and the roughness of the interface discussed in I.3 only acts in adaptive computations.

Unless it is specified, the fluid characteristics are the following for each test (l and g subscribes stand for *liquid* and *gas*):

- density $\rho_l = 1000 kg.m^{-3}$, $\rho_g = 1 kg.m^{-3}$,
- dynamic viscosity $\mu_l = 10^{-3} Pa.s$, $\mu_g = 10^{-5} Pa.s$,

and the level set function is discretized using piecewise linear approximations ($p = 1$). The density and viscosity ratios are close to those of air and water.

The tolerance for the GMRES solver has been prescribed to 10^{-6}. All the computations have been performed on a Dual Core AMD Opteron Processor 265 (1800MHz).

Parasitic velocities: free surface at rest

As a consequence of the high density ratio of the two fluids, spurious velocity oscillations may occur at the interface. A way to highlight these velocities is to look on how our scheme is able to resolve the hydrostatic equilibrium. Spurious velocities will hopefully decrease with the size of the elements near the interface. We consider a plane rectangular free surface in a 3D box. The two fluids are at rest at initial time $t = 0$ and no perturbation is introduced in the shape of the flat interface. The pressure is initialized to the hydrostatic equilibrium value. The velocity $\mathbf{u}$ is supposed to remain zero and the equation of motion (I.1) becomes

$$\frac{\nabla p}{\rho(\phi)} = \frac{\mathbf{e_g}}{Fr^2}. \tag{I.14}$$

If elements cross the interface, the exact solution does not belong to the finite element space and spurious velocities are generated.

The setup of this test is the following. The domain has a width of 2×2 and a height of 1 and is equally divided into air and water. We characterize the parasitic velocities by the infinity norm $|\mathbf{U_{par}}|_\infty$ of the velocities on the domain after that it has reached a stable value:

$$|\mathbf{U_{par}}|_\infty = \max\left(\sqrt{\mathbf{u_{par}^2} + \mathbf{v_{par}^2} + \mathbf{w_{par}^2}}\right)$$

Table I.1 shows the maximum velocities obtained for different sizes of the elements. Results have been obtained for isotropic and anisotropic meshes. For the anisotropic refinements the ratio $R_\mathbf{n}$ is 4 and the edge length h_0 is multiplied by $2^{\frac{1}{3}}$ in order to have the same mean volume for the elements in the isotropic and anisotropic computations.

h_0^{iso}	isotropic	anisotropic ($R_\mathbf{n} = 4$)
0.1	$2.76\,10^{-4}$	$1.64\,10^{-4}$
0.01	$1.39\,10^{-5}$	$6.14\,10^{-6}$
0.001	$2.38\,10^{-7}$	$1.22\,10^{-7}$

Table I.1: Stationary case: Maximum parasitic velocities $|\mathbf{U_{par}}|_\infty$ with isotropic and anisotropic meshes. For the anisotropic computations, the ratio $R_\mathbf{n}$ is 4 and the edge length $h_0^{aniso} = 2^{\frac{1}{3}}\, h_0^{iso}$.

We note that anisotropic meshes need more iterations to reach the tolerance of the GMRES solver since the introduction of anisotropic elements which are not aligned with the flow are known to deteriorate the conditioning of the system. Table I.2 shows the number of iterations needed per time step for each computation.

h_0^{iso}	isotropic	anisotropic
0.1	12	14
0.01	19	22
0.001	44	56

Table I.2: Stationary test: number of GMRES iteration performed per time step for the isotropic and anisotropic ($R_{\mathbf{n}} = 4$) computations.

Parasitic velocities: 3D static droplet

In the case of a spherical droplet with no gravity effects, a stationary equilibrium occurs at the interface between the pressure forces and the surface tension forces and gives rise to a pressure jump at the vicinity of the interface. However, since the pressure discontinuity and the discontinuous surface tension forces are not approximated in exactly the same way, this equilibrium may be perturbed and spurious velocities may occur near the droplet interface. Several authors have highlighted this phenomena, like Popinet and Zaleski [143] and Francois et al. [69].

The setup is the same as in [69] and [189] for the 3D bubble. The domain is a cube of side size $C = 8$. The bubble has a diameter of $d = 4.0$ and is located in the center of the domain. There is no viscosity effect and the density ratio is set to 0.1 between the two fluids, the bubble having the highest density. A surface tension of $\gamma = 73$ is prescribed.

Here we show how the mesh refinement can be used to decrease the parasitic currents. Three levels of mesh refinements are presented, each one with an isotropic and an anisotropic refinement. The number of elements is kept approximately constant between corresponding isotropic and anisotropic computations by applying a factor of $3^{1/3}$ to h_0 ($h_0^{aniso} = 3^{1/3}\, h_0^{iso}$) since a ratio $R_{\mathbf{n}}$ of 3 is used. Table I.3 shows the maximum velocity U^{max} at $t = \Delta t$ and $t = 50\Delta t$, with $\Delta t = 0.001$. The results are compared with the best results of Francois et al. in [69] and Williams et al. in [189] obtained with $h = \frac{C}{40}$. We can observe that the velocities decrease with the size of the elements, which is a common result in the literature. An interesting result is that the anisotropic computations lead to smaller spurious velocities.

The number of iterations required by the GMRES solver at each time step increases slightly when an aspect ratio is applied, as shown in table I.4.

2D Dam break

We consider a column of water maintained by a wall (a dam) that is impulsively removed at $t = 0$. This test is the basis of many works which try to predict the effects of a dam break. This test case has been widely studied in the literature using experimental, theoretical and numerical approaches. As this test involves recombinations and strong deformations of the interface as well as a great unsteady character, it is also a good test case to validate our adaptive model.

h_0^{iso}	t	isotropic	anisotropic	Francois et al. [69]	Williams et al. [189]
$C/20$	$t = \Delta t$	$9.19\,10^{-2}$	$7.34\,10^{-2}$		
$C/20$	$t = 50\Delta t$	$9.26\,10^{-1}$	$4.36\,10^{-1}$		
$C/40$	$t = \Delta t$	$6.80\,10^{-2}$	$5.69\,10^{-2}$	$4.02\,10^{-3}$	$8.55\,10^{-2}$
$C/40$	$t = 50\Delta t$	$3.77\,10^{-1}$	$1.88\,10^{-1}$	$4.02\,10^{-2}$	$3.86\,10^{-1}$
$C/80$	$t = \Delta t$	$5.73\,10^{-2}$	$4.53\,10^{-2}$		
$C/80$	$t = 50\Delta t$	$2.59\,10^{-1}$	$1.31\,10^{-1}$		

Table I.3: Stationary spherical bubble case: maximum velocity U^{max} with several levels of refinement. The time step is $\Delta t = 10^{-3}$. The results are obtained for isotropic refinements and anisotropic refinements with a ratio of $R_{\mathbf{n}} = 3$ and the same numbers of nodes as in the corresponding isotropic meshes. The results are compared with the best results of Francois et al. in [69] and Williams et al. in [189] obtained with $h = \frac{C}{40}$.

h_0^{iso}	isotropic	anisotropic
$C/20$	12	14
$C/40$	18	21
$C/80$	22	25

Table I.4: Stationary spherical bubble case: number of GMRES iterations required for three levels of refinements in isotropic and anisotropic computations. The mean volume of the elements in the dense zone is the same between the corresponding isotropic and anisotropic computations.

The length of the domain is 6 and its height is 4. The water column is initially at the extreme right of the domain. The height and the width of the water column are 1. Slip conditions are applied on the bottom and the side walls, neglecting possible boundary layer effects. A zero-pressure condition is imposed on the upper boundary.

Two adaptive computations have been performed. Linear and quadratic elements have been used to discretize the level set function. Table I.5 shows the two sets of parameters. The effects of surface tension are neglected because this flow is essentially governed by the gravity. With these parameters, the

	p	h_0	h_1	h_2	δ	d_1	d_2	$R_{\mathbf{n}}$	$R_{\mathbf{t},max}$	κ_{max}
Comput. 1	1	0.1	0.2	0.4	0.1	0.3	0.6	3.0	2.0	10.0
Comput. 2	2	0.12	0.3	0.6	0.08					

Table I.5: 2D dam break test: parameters of the two first computations.

number of nodes ranges from $1,380$ to $3,151$ for the computation 1, from $1,013$

to $1,848$ for computation 2. Figure I.5 shows the interface and the mesh of the first computation at different times.

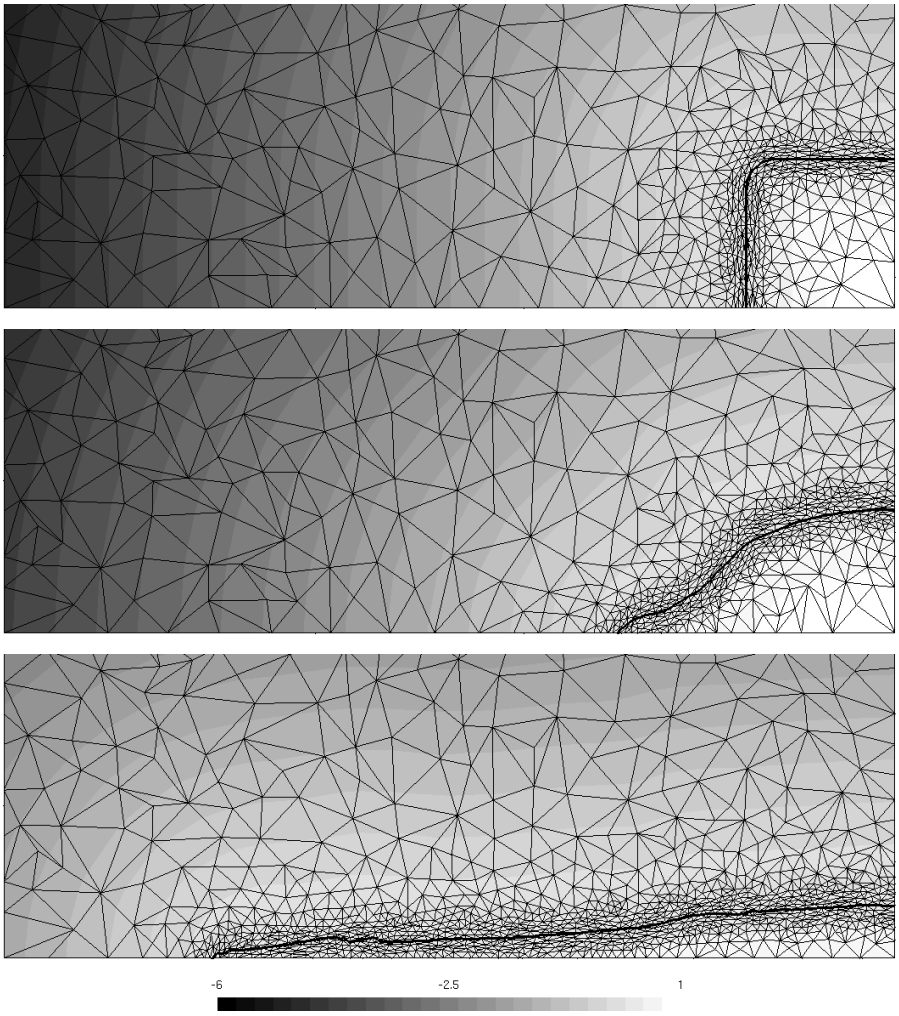


Figure I.5: 2D dam break problem: visualization of the meshes and the level set fields at dimensionless times 0, 1 and 3 for the first computation. The upper part of the domain is not represented.

The history of the dimensionless horizontal displacement of the water front is shown in fig.I.6. The time is non-dimensionalized by $t = \sqrt{h_l/g}$ where h_l is the height of the water column. The experimental results from Martin and Moyce [145] and the numerical results from Marchandise and Remacle [119] with a non-adaptive mesh are added to the diagram. The latter computation has been performed on a domain of height 1.5, with an unstructured mesh of $10,218$ nodes and a mean edge length of about 0.04.

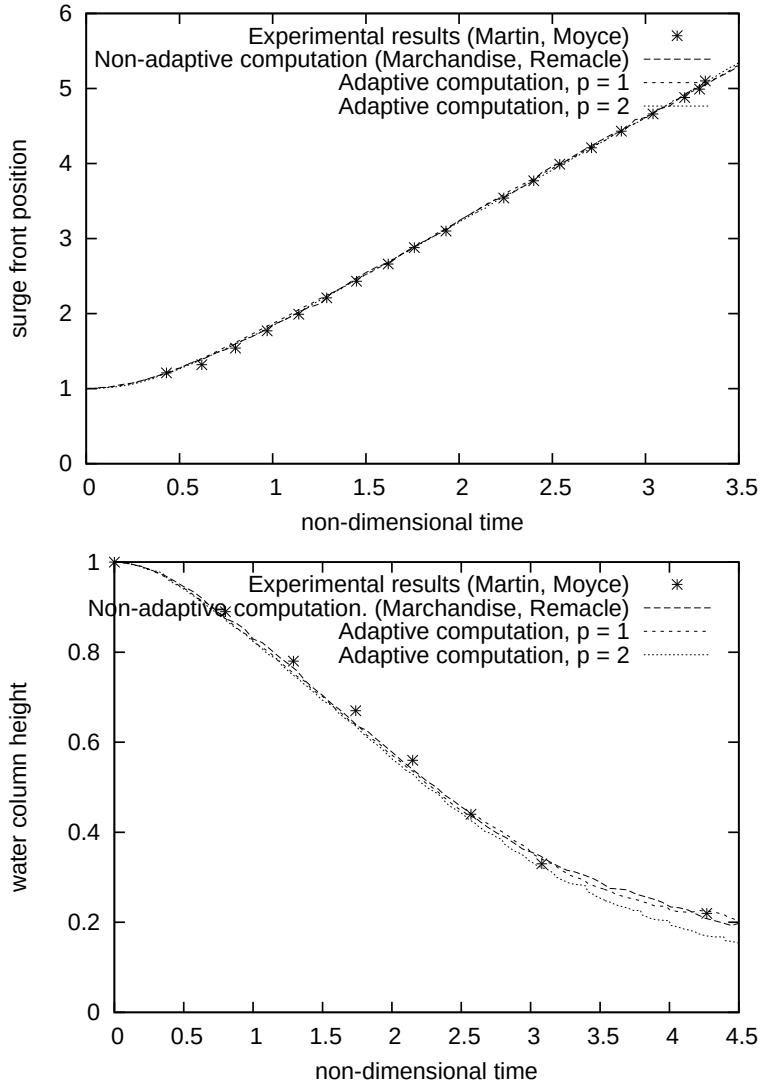


Figure I.6: 2D dam break problem: comparison between experimental results, non-adaptive method and present method.

One can see that these results are in good agreement with the non-adaptive computations and the experiment from Martin and Moyce, except for the height of water in the second simulation, for which the number of nodes is not sufficient to avoid significant mass losses.

For the first computation, the computational time to reach the dimensionless time 3 was 48 minutes, with about 24% spent in the adaptation process (involving projections). A non-adaptive computation with an edge length $h_{na} = 0.033$ has been performed with the same domain and fluids setup. The

mesh was composed of $46,204$ nodes ($230,418$ elements). The time 3 was reached after 15.4 hours.

The second computation was a bit slower than the first, with 85 minutes spent to reach time 3 (13.6% in adaptation process).

In order to highlight the efficiency of the anisotropic refinements compared to the isotropic refinements, the evolution of the front position computed with isotropic and anisotropic coarse refinements are presented in figure I.7. Two levels of accuracy are proposed, each one being tested with the values 1 and 4 for the ratio R_n. The size h_0 is chosen in such a manner that the number of nodes is roughly the same for the corresponding isotropic and anisotropic computations. The parameters of the meshes are mentioned in table I.6. In order to emphasize the differences between the results, the meshes are relatively coarse which leads to big errors.

	p	h_0	h_1	h_2	δ	d_1	d_2	$R_{\mathbf{n}}$	$R_{\mathbf{t},max}$	κ_{max}
Comput. 3	1	0.15	0.4	0.4	0.2	0.3	0.6	1.0	1.0	
Comput. 4		0.3						4.0	2.0	10.0
Comput. 5	1	0.1	0.4	0.4	0.2	0.3	0.6	1.0	1.0	
Comput. 6		0.2						4.0	2.0	10.0

Table I.6: 2D dam break test: parameters of the computations 3 to 6.

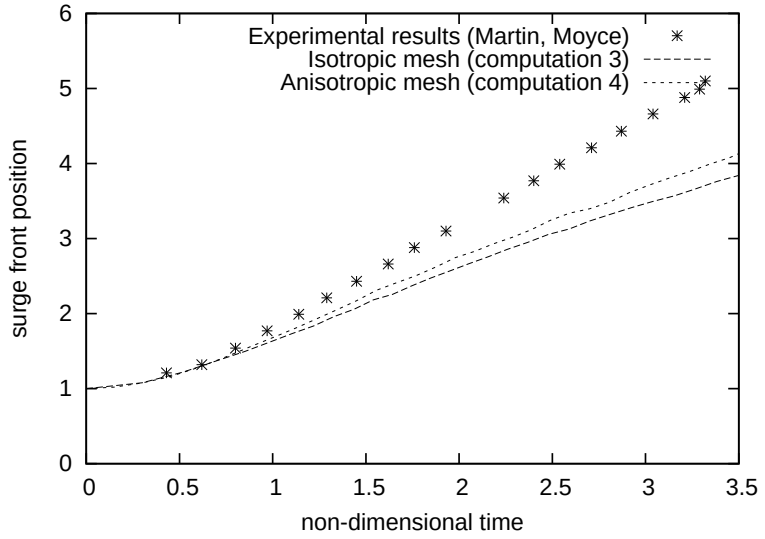
3D Dam break with a square pile

This test case is still a dam break, but the geometry of the problem is 3D. A column of water collapses under gravity and the path of the wave crosses a square pile, leading to the development of a complex three-dimensional highly transient flow. The complexity of the shape of the interface and its variability is space and time makes the adaptive approach highly competitive.

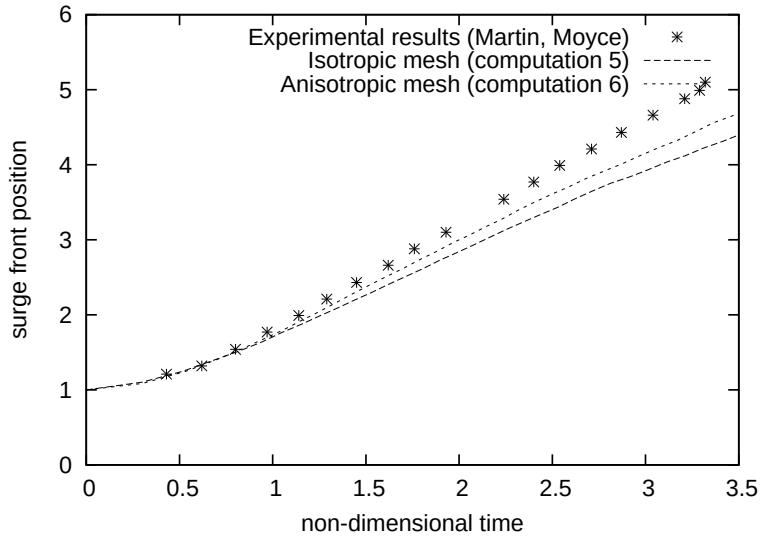
The geometrical setting is the following. The height and the length of the initial water mass are 1. Its width is 1.6. The distance to the pile is 1.35. The pile has a square base of 0.3 and is shifted of 0.1 to the left. The back wall is located 3 far from the water. Slipping conditions are applied on all the walls. A zero-pressure condition is applied on the upper face of the domain, which is located 5 above the bottom wall.

Two computations were performed, with the parameters of table I.7, a Reynolds number of $40,000$, a Froude number of 0.3193 and no surface tension. Note that the Reynolds number is purely indicative because boundary layers are not captured in our simulation.

With these settings, the number of nodes ranges from $5,202$ to $29,350$ for the first computation and from $3,361$ to $16,185$ for the second. The computational times were respectively 33.9 hours (21.4% in the adaptation process) and 51.0 hours (12.6% adapt.) hours to reach the dimensionless time 2.5. In order to build fixed meshes with the same accuracy according to criterion (I.13), the number of nodes would have raised to a number of about $600,000$ in the first simulation, and to about $250,000$ in the second simulation. Figure I.9 shows the



(a) Computations 3 and 4



(b) Computations 5 and 6

Figure I.7: 2D dam break problem: comparison between isotropic and anisotropic coarse refinements.

interface with the meshes of some boundaries at different times, while figure I.8 represents a cut in the tetrahedral mesh at time 1.59.

	p	h_0	h_1	h_2	δ	d_1	d_2	$R_{\mathbf{n}}$	$R_{\mathbf{t},max}$	κ_{max}
Comput. 1	1	0.15	0.4	1.0	0.045	0.4	0.6	4.0	3.0	10.0
Comput. 2	2	0.2			0.06					

Table I.7: 3D dam break case: parameters of the computations.

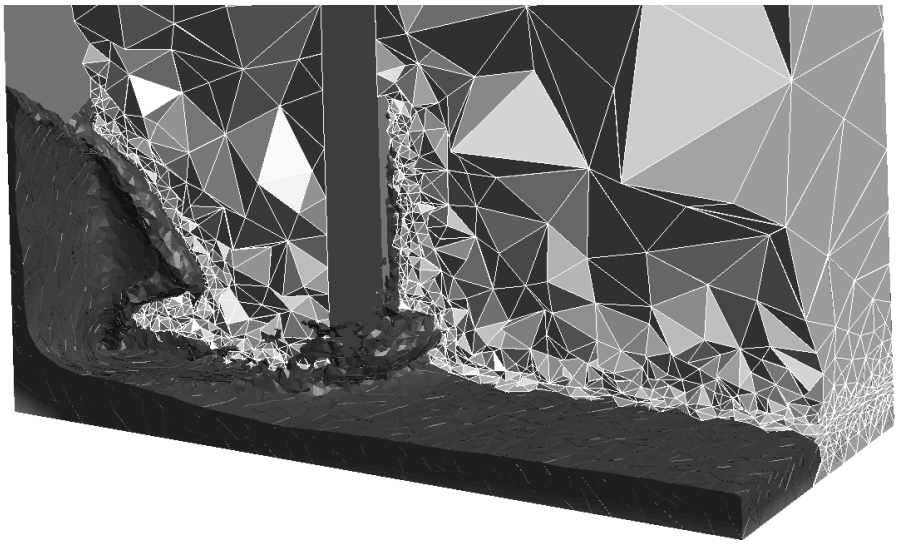


Figure I.8: 3D dam break case: volume of water and tetrahedral mesh at time 1.59.

Axisymmetric bubbles coalescence

In this test case, two air bubbles are immersed in a volume of water. The first one has a bigger radius and is initially located above the second. The bubbles rise under the effect of gravity, but the second rises faster because of the depression created by the first. Eventually, the bubbles merge into a single non-spherical bubble. The effects of surface tension are not taken into account.

The computational domain has a length of 1 in x and y directions, and 2.3 in z. The radius are respectively 0.15 and 0.10 for the first (above) and the second bubble. The bubbles are at a distance of 0.05. The gravity acts in the z direction. Slipping conditions are applied on all the boundaries. The Reynolds number and the Froude number are the following: $Re = 200$ and $Fr = 1$.

First order elements are used to represent the level set function. The meshes have the characteristics given by I.8. The meshes produced have a number of nodes ranging from $16,480$ to $24,922$.

Figure I.10 shows the interface and the meshes at different times. We were able to obtain very smooth interfaces thanks to mesh adaptation.

It took 17.7 hours to run this computation. About 19% of the time was spent in the adaptation process. A non-adaptive computation with a similar

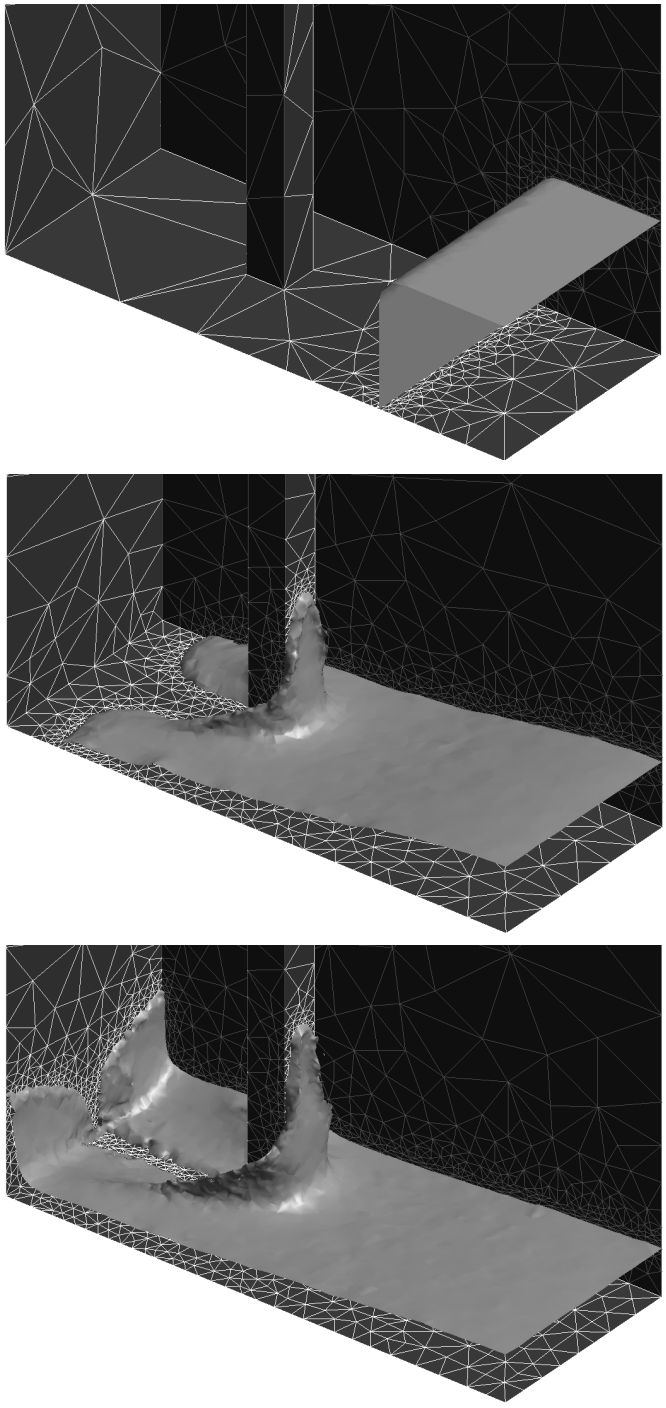


Figure I.9: 3D dam break case: interface and surface meshes at times 0, 0.88, 1.17 (continued on next page).

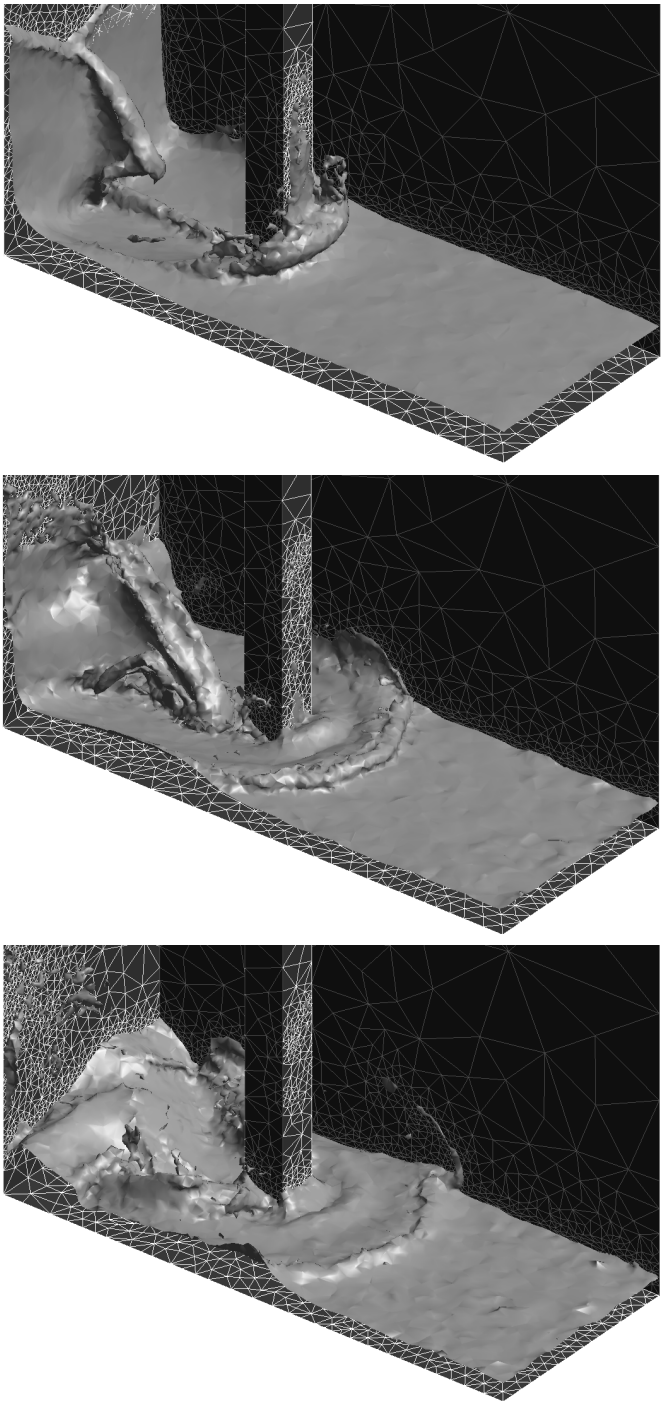


Figure I.9: 3D dam break case (continued): interface and surface meshes at times 1.59, 2.0 and 2.39.

h_0	h_1	h_2	δ	d_1	d_2	$R_{\mathbf{n}}$	$R_{\mathbf{t},max}$	κ_{max}
0.03	0.15	0.40	0.02	0.02	0.15	3.0	2.0	0.05

Table I.8: Axisymmetric bubbles coalescence: parameters of the meshes.

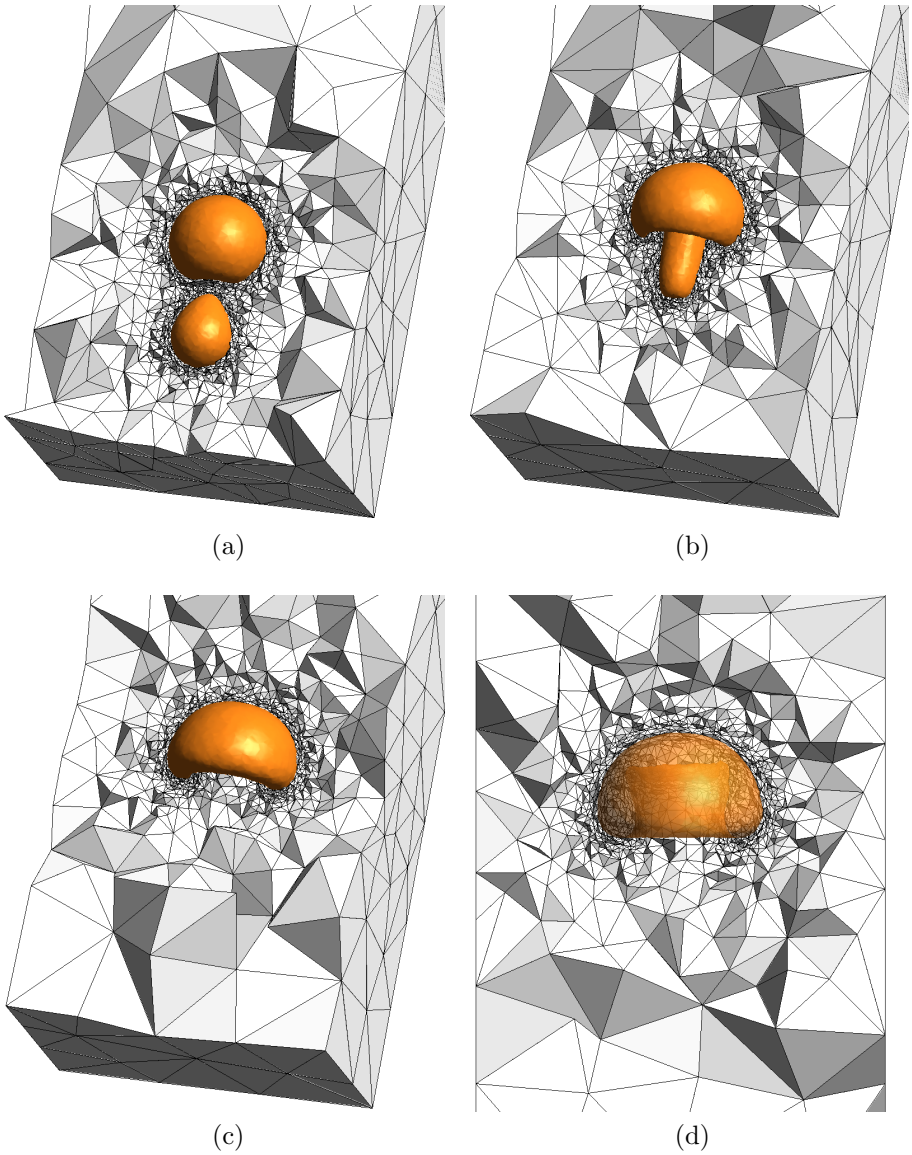


Figure I.10: Axisymmetric bubbles coalescence. $Re = 200$, $Fr = 1$, $\rho_1/\rho_2 = 1/1000$ and $\mu_1/\mu_2 = 1/100$. Interface and meshes at dimensionless times (a) 0.2, (b) 0.56 and (c,d) 1.

accuracy, according to (I.13), would have needed a mesh size $h_{na} = 0.01$. With this very small mesh size, the mesh would have contained more than $2,300,000$ nodes.

Oblique bubbles coalescence

For this test, two spherical bubbles of the same radius are immersed in the fluid with a shift between their horizontal positions. Both phases are initially at rest.

The domain is a parallelepiped of size $[0,4] \times [0,4] \times [0,8]$. The center of the bubbles are located at positions $(2,2,2.15)$ and $(2.5,2,1)$. Both bubbles have an initial radius $R = 0.5$. The following dimensionless values are set: $Re = 18.8$, $We = 50$, $Fr = 1$, $\rho_1/\rho_2 = 1/20$ and $\mu_1/\mu_2 = 1/26$. The computation has been made with quadratic shape functions for the level set.

Table I.9 resume the meshes parameters that have been chosen.

h_0	h_1	h_2	δ	d_1	d_2	$R_{\mathbf{n}}$	$R_{\mathbf{t},max}$	κ_{max}
0.4	0.4	2.0	0.15	0.3	2.0	4.0	3.0	0.2

Table I.9: Oblique bubbles coalescence: parameters of the meshes.

Figure I.11 shows the evolution of the bubbles. The results compare well with those obtained by Marchandise et al. [118] without mesh adaptation and a grid of $30 \times 30 \times 60$. They also compare well with those obtained by Sussman and al. [183] with a $64 \times 64 \times 128$ grid, those of Sousa et al. [57] and the experimental results obtained by Narayanan [130]. Whereas $54,000$ nodes were necessary in [118], all meshes produced with our computation comprised less than $6,000$ nodes, leading to a computation of 7 hours (about 12% spent for mesh adaptation) to reach $t = 3$.

We can see from figure I.11 that the anisotropic refined mesh combined with quadratic elements for the levelset were able to represent the smallest scales of the interface with a very good accuracy.

I.5 Conclusion

An adaptive procedure to improve the efficiency of the numerical method based on the model described by Marchandise and Remacle in [119] has been presented.

This model was already able to simulate highly transient flows with high ratios of densities and viscosities, with the advantages that it simply uses first order elements to represent pressure and velocity variables, and that it relies on a level set method for the interface, which is very flexible. Numerous applications are also in the scope of the model as it is able to represent the effects of surface tension in an accurate way.

The CPU time to perform complex simulations has been dramatically reduced by the adaptive improvement, without undermining the accuracy. This

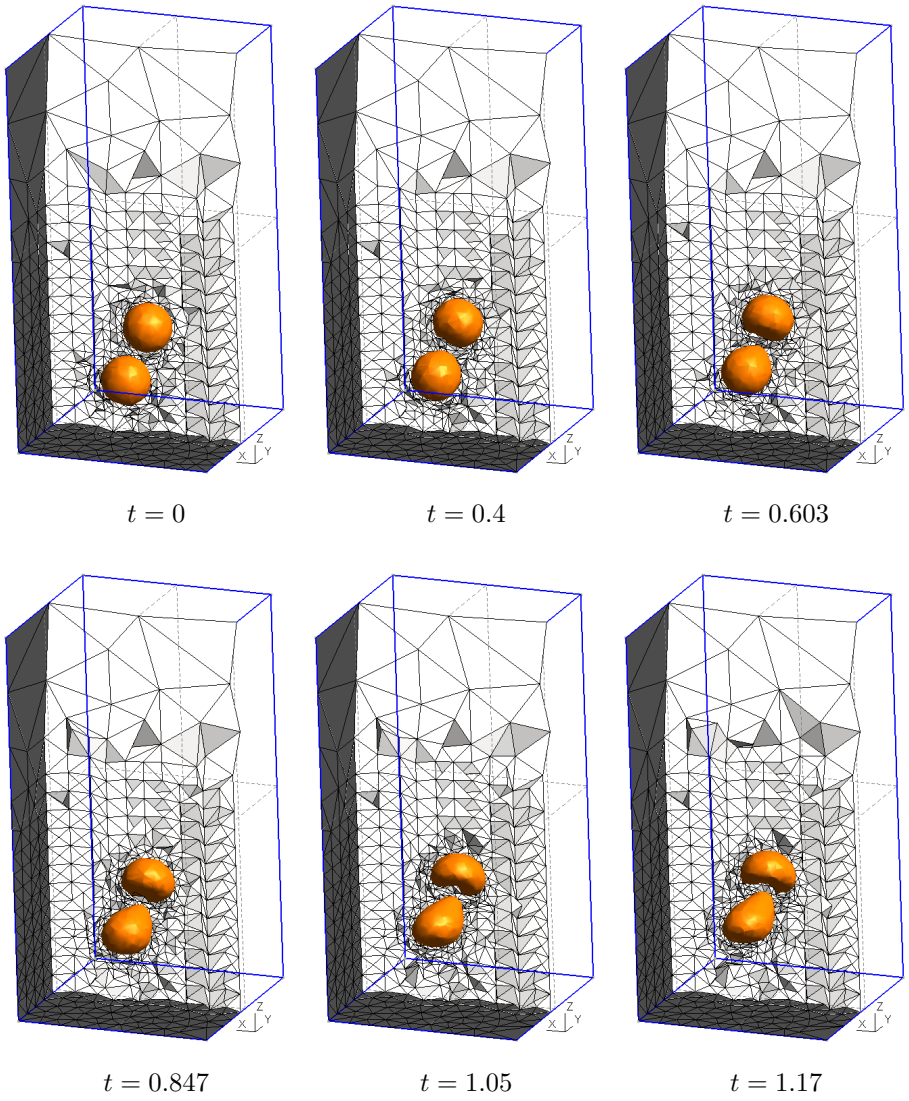
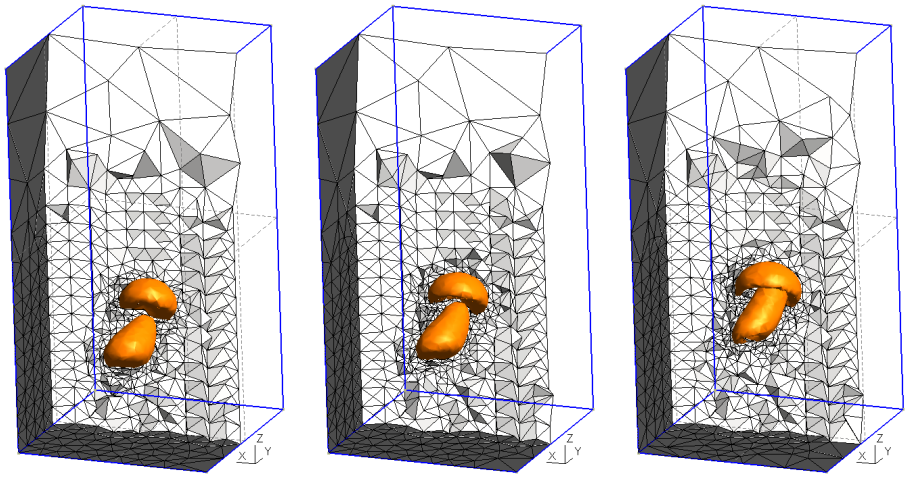


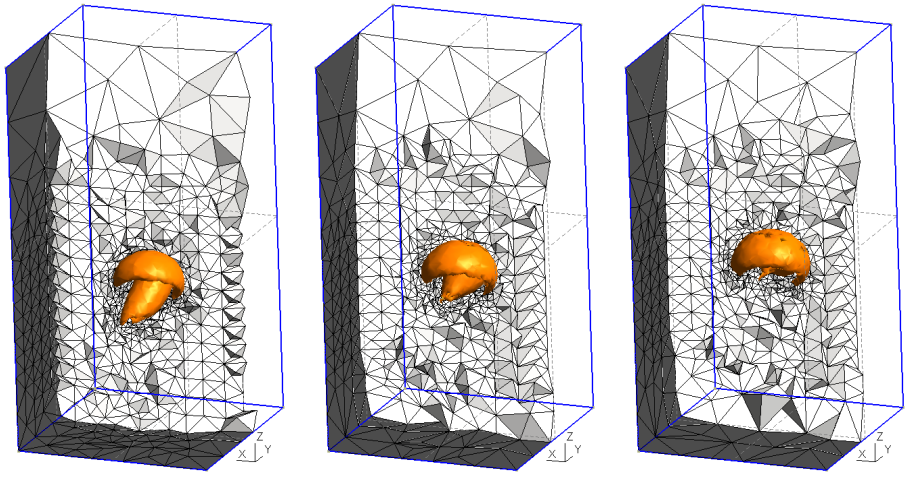
Figure I.11: Oblique bubbles coalescence. $Re = 18.8$, $We = 50$, $Fr = 1$, $\rho_1/\rho_2 = 1/20$ and $\mu_1/\mu_2 = 1/26$. Position of the interface at different dimensionless times ranging from 0 to 2.89. A view of the mesh is superimposed to the view of the interface (continued on next page).

enhancement allows us to handle more complex computations in which either small and large scales are solved.

In the future works, a parallel implementation will be carried out. The fluid solver is already able to perform parallel computations. We are now focusing on the parallelization of the adaptive process, resorting to the work of Dobrzynski [61].



$t = 1.33$ $t = 1.56$ $t = 1.95$



$t = 2.26$ $t = 2.55$ $t = 2.89$

Figure I.11: Oblique bubbles coalescence (continued). $Re = 18.8$, $We = 50$, $Fr = 1$, $\rho_1/\rho_2 = 1/20$ and $\mu_1/\mu_2 = 1/26$. Position of the interface at different dimensionless times ranging from 0 to 2.89. A view of the mesh is superimposed to the view of the interface.

Chapter 2

Mesh adaptation for large deformations

In this chapter, we focus on the improvements brought to the mesh adaptation methods based on local modifications in order to meet the requirements of the computations with deforming domains. The goal here is to be able to adapt a mesh when the meshed domain undergoes arbitrarily large deformations.

A common solution to the problems involving large displacements or deformations is to reposition the nodes globally, the displacements being the solution of an auxiliary problem like for instance the computation of an elasticity problem where an analogy is made between the domain and an elastic material [186], and the displacement of the boundaries is given as a Dirichlet boundary condition. However, when the deformations are too large or a shear layer appears in the underlying elastic problem, the node repositioning techniques fail in returning a valid mesh. In addition, such a method does not provide a control on the quality of the elements. As an example, Figure 2.1 shows the mesh of a rotating propeller. The nodal repositioning is computed by a method based on the elastic analogy. We can see that the elements become progressively ill-shaped at the tip of the blades and finally, tangled elements appear.

The solution proposed in the present work is to mix the nodes repositioning techniques with a mesh adaptation procedure in order to enable arbitrarily large domain deformations, and additionally to give a control over elements quality.

The starting point of the work was the mesh adaptation method presented in Chapter 1 since

- we had a relatively good experience of it,
- it proved to be efficient and robust in the work presented in Chapter 1,
- it allows for anisotropic adaptivity,
- the approach is parallelizable.

After the first tests, the following conclusions were made:

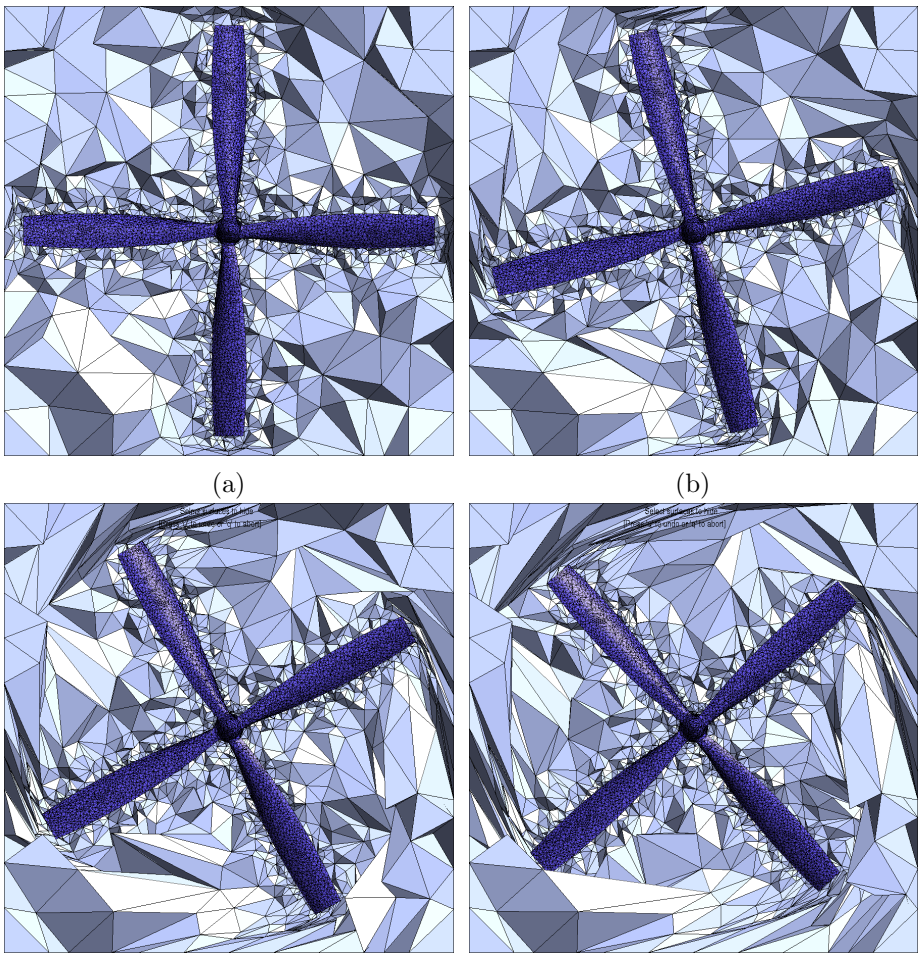


Figure 2.1: Example of a mesh undergoing large deformations if only node repositioning is applied: (a) initial mesh, (b) and (c) intermediate meshes, and (d) meshes when first tangled elements appear.

- The combination of the most common local mesh modifications (edge split, edge collapse and edge swap) and the sliver elimination procedure presented in [106] and used in Chapter 1 were able to provide valid meshes in some cases, like the straight displacement of a sphere in a box, but it failed for simple bodies with straight faces and sharp edges, like a moving cube.
- The combination described here above loses most of its interest of one of the mesh modifications is removed from the set.

- Combining the set of modifications and the sliver elimination procedure with a node repositioning technique improves the robustness and efficiency of the method and the quality of the resulting meshes. However, tangled elements can still appear in very simple motions, like a wall moving in its normal direction. The main reason for it is that the quality of the smallest elements located near the walls is strongly degraded by the motion of the body and the sliver elimination procedure can fail in eliminating them. It is therefore interesting to use node repositioning methods with a variable stiffness depending on the distance to the mobile walls or the size of the elements [177], as well as a better sliver elimination procedure.

The new techniques developed to reach the objectives of the present chapter are described in Article II, *Transient mesh adaptivity with large rigid-body displacements*, Compère, Remacle, Marchandise, Proc. of the 17$^{\text{th}}$ Int. Mesh. Roundtable (2008), proposed here below. The key points are (i) a new efficient sliver elimination procedure, and (ii) a global adaptation method that gathers the different ingredients (mesh sizes evaluation, local modifications, global repositioning with selective stiffness, slivers elimination) in an efficient way. In this paper, the resulting method is applied to simple fluid-structure interaction (FSI) problems.

In the paper entitled *A mesh adaptation framework for dealing with large deforming meshes* (Compère, Remacle, Jansson, Hoffman) (Article IV) presented in Chapter 5, an additional technique is presented to handle arbitrarily large domain deformations inside a single time step by applying topological mesh modifications during the global node repositioning. A more complex FSI problem in which the pure node repositioning technique is compared to the mesh adaptation method developed here is also presented, and the impact of the adaptation frequency on element shapes, number of modifications and computational time is analyzed.

Article II
Transient mesh adaptivity with large rigid-body displacements

Gaëtan Compère[1,2*], Jean-François Remacle[1], Emilie Marchandise[1]

[1] *Université catholique de Louvain, Department of Civil Engineering, Place du Levant 1, 1348 Louvain-la-Neuve, Belgium*

[2] *Fonds National de la Recherche Scientifique, rue d'Egmont 5, 1000 Bruxelles, Belgium*

Abstract

This paper presents a procedure for computing fluid-structure interaction problems when the boundaries of the domain undergo large displacements. The algorithm is based on both mesh motion and mesh adaptation techniques. More specifically, we use a node repositioning algorithm based on an elastic analogy together with a mesh adaptation procedure based on local mesh modifications [106]. This paper also includes a new technique for eliminating sliver tetrahedra. Some computational results are finally presented that include statistics on mesh quality measures.

II.1 Introduction

This paper deals with the issue of computing fluid-structure interaction (FSI) problems with large displacements of the structure. The most common way of dealing with FSI is to adopt an Arbitrary Lagrangian Eulerian (ALE) formulation of the fluid equations. ALE formulations allow to take into account small motions of the nodes in the fluid caused by the displacement of the structure. This approach suffers from obvious limitations as node repositioning cannot always provide a valid mesh when significant displacements or deformations of the structure are considered.

One way of addressing this problem is to re-mesh the entire domain when the displacements of the structure are too large to be handled [4, 27]. In this work, we rather use local mesh modifications [106, 152, 38, 13] both to optimize the quality of the tetrahedra and to comply a mesh size field. This approach is globally advantageous compared to global re-meshing:

- Local solution projection procedures can be built in a way that ensures local conservation [152],
- The mesh remains unchanged in large parts of the domain,

- Local mesh modifications can be performed in parallel, enabling transient adaptive simulation to run on parallel computers [38].

There exist a small number of other approaches to handle large mesh deformations, among which the sliding mesh techniques [123]. Compared to local mesh modifications, they have the disadvantage of requiring a flux computation at the interface and working with non coinciding meshes. The overset grid methods [107] are also developed in the literature but require grid assemblies and donor cell search processes. The local mesh modification technique has a unique mesh and a single prescribed boundary motion, which leads to a conceptually simple method and robust mesh movements [38].

Transient adaptive computations using local mesh modifications have already been applied to transient multiphase flow simulations in [44]. In this previous work, the mesh was adapted in order to capture the interface between two fluids, described by a level set function. In this paper, we extend the method to FSI problems.

In this context, two new issues have to be addressed. The first issue is the formulation of the mesh motion problem. In FSI, only the motion of the boundaries of the domain is prescribed. Typically, some kind of elastic analogy, possibly with a variable stiffness [177], is used to extend this motion inside the domain. Here, we do not make the assumption that the boundary motion is small or that the volume of the domain remains unchanged. An elastic approach does not give any guarantee on the validity of the mesh for large displacements. Even when the motion is limited, ill-shaped elements are produced in the process. This is the second issue. A new efficient procedure is presented that enables to eliminate all ill-shaped elements that are inevitably produced during the mesh motion.

The remainder of this paper is organized as follows: the first section introduces the different edge size fields. Section II.3 describes the set of local mesh modification operators while section II.4 presents the ill shaped elements elimination algorithm. In section II.5, we recall the principles of the elastic analogy for mesh motion and discuss the choice of local element stiffness. The global procedure is then described in section II.6. Section II.7 presents some numerical results.

II.2 Mesh size field

The aim of the mesh generation process is to build elements of controlled shape and size. Mesh generators are usually able to adapt to a so called mesh size field (see for instance [136,106]). An isotropic mesh size field is a scalar function $\delta(\mathbf{x}, t)$ that defines the optimal length of an edge at position $\mathbf{x}$ of the domain and at time t.

We typically define the dimensionless length L_e^{tr} of edge e as

$$L_e^{tr}(t) = \int_e \delta^{-1}(\mathbf{x}, t) dl. \tag{II.1}$$

The quantity L_e^{tr} represents the number of subdivisions of edge e that are necessary for having an edge that has exactly the right size with respect to the

size field. For the sake of simplicity, we do not consider here an anisotropic metric field and the corresponding dimensionless length defined by (I.5) and further described in Appendix A. More explicitly, (II.1) is a particular case of (I.5) in which the eigenvalues of $\mathcal{M}(\mathbf{x},t)$ are all equal to $\delta^{-2}(\mathbf{x},t)$.

There are many ways to define a size field: some are based on rigorous error estimation procedures [1] but there are many heuristics. For example, it is often considered that the mesh size should be smaller near boundaries: when dealing with viscous flows, a large part of the vorticity is created near the walls, when dealing with solid mechanics problems, stress concentration are usually located near the boundaries...

We define $d(\mathbf{x},t)$ as the distance to the closest boundary at time t. This distance can be computed *in place* using the Approximated Nearest Neighbor Algorithm [9]. A first mesh size field $\delta_1(\mathbf{x},t)$ is computed as follows:

$$\delta_1(\mathbf{x},t) = \delta_1^{\text{small}} + \alpha_1(\mathbf{x},t)(\delta_1^{\text{large}} - \delta_1^{\text{small}}),$$

where

$$\alpha_1(\mathbf{x},t) = \begin{cases} 0 & \text{if} \quad d(\mathbf{x},t) \leq d_1^{\min} \\ \frac{d(\mathbf{x},t)-d_1^{\min}}{d_1^{\max}-d_1^{\min}} & \text{if} \quad d_1^{\min} < d(\mathbf{x},t) < d_1^{\max} \\ \infty & \text{if} \quad d(\mathbf{x},t) \geq d_1^{\max} \end{cases}$$

with δ_1^{large} and δ_1^{small} a large and a small desired mesh sizes, $d_1^{\max}$ and $d_1^{\min}$ two field values that define the zone of refinement. An example of use of δ_1 is presented in figure II.1.

Other size fields $\delta_2(\mathbf{x},t), \delta_3(\mathbf{x},t), \ldots$ can be defined:

1. mesh sizes prescribed at model vertices and interpolated linearly on model edges;
2. prescribed mesh gradings on model edges (geometrical progressions, ...);
3. mesh sizes defined on another mesh (a background mesh) of the domain;
4. mesh sizes that adapt to the principal curvature of model entities.

The size field $\delta(\mathbf{x},t)$ is computed, at time t, as the minimum of all size fields. It is usually bounded by upper and lower values of mesh sizes.

In an ideal mesh, each edge has a dimensionless length $L_e^{tr} = 1$. This ideal situation cannot be attained in practice. In the adaptation procedure one has to decide whether an edge is acceptable, i.e. find a range $[L_{low}, L_{up}]$ for which an edge is considered to have a good size. Then a long edge has a size $L_e^{tr} > L_{up}$ and a short edge has a size $L_e^{tr} < L_{low}$.

In [106], the authors show that choosing a too narrow range for acceptable edge sizes may lead to infinite loops between splits and collapses. In § II.6 and § II.7, we show the influence of this interval on the mesh quality and on the number of infinite loops.

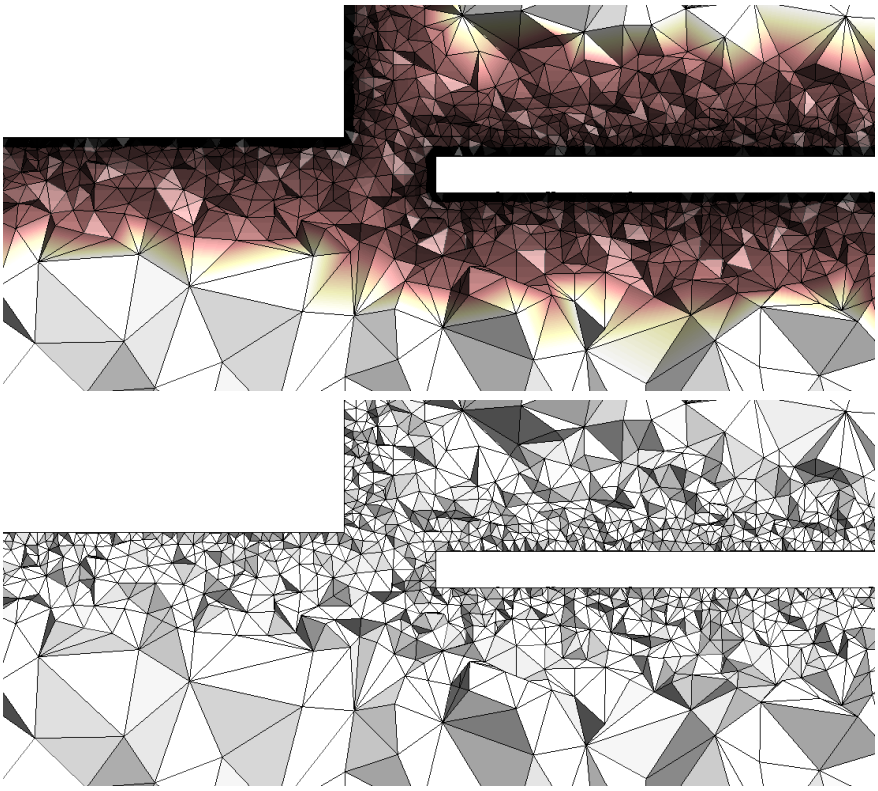


Figure II.1: Mesh adapted using the distance to boundaries: size field (top) and resulting mesh (bottom).

II.3 Local mesh modifications

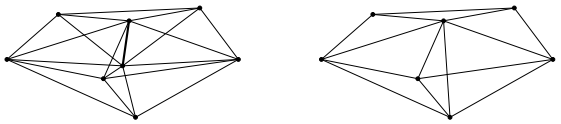
The three basic local mesh modifications are the edge split, the edge collapse and the edge swap. Other operators like face swap and some compound operators are also used in this work.

There are two ways of dealing with long edges. The first one consists in tagging every long edge of the mesh. Then, every tetrahedron of the mesh is subdivided using a template that is function of the number of edges that have been split. The second manner consists in splitting all tetrahedra surrounding a long edge, and then proceed to the next edge. The latter approach has been used because it is simpler (only one template has to be defined), more robust (no Steiner point has to be introduced) and (surprisingly) more efficient. Moreover, we have found out that both methods lead to meshes with elements of similar qualities. **The edge split** operator is depicted on figure II.2.

When a short edge is found, an **edge collapse** is applied. The edge collapse operator (see figure II.3) removes an edge and all its bounding elements from the mesh by merging its two extremities at one of their locations.

(a) (b)

Figure II.2: Edge split operation: (a) initial cavity, (b) cavity after the edge split.



(a) (b)

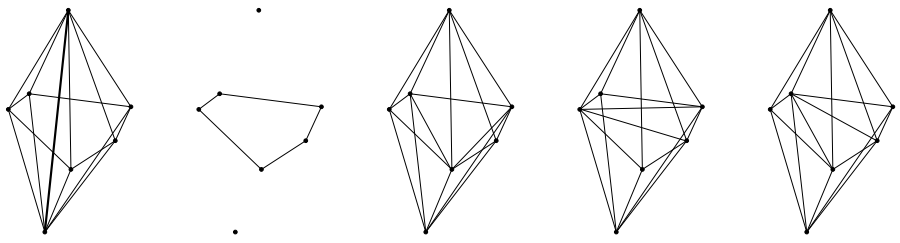
Figure II.3: Edge collapse operation: (a) initial cavity, (b) cavity after the edge collapse.

The edge swap consists in re-meshing the cavity surrounding the edge with the aim of improving the worst element shape in that cavity. The shape quality measure of a tetrahedron K is chosen to be the cubic mean ratio η_K^3 [108], defined by

$$\eta_K^3 = 15552 \frac{{V_K}^2}{\left(\sum_{i=1}^{6} (L_i)^2\right)^3}, \tag{II.2}$$

with V_K the volume of K and L_i the length of the i^{th} edge of K. The 15552 factor is set to scale η_K^3 such that it ranges from 0 to 1. Its value is 1 for an equilateral tetrahedron and 0 for a flat tetrahedron.

Figure II.4(a) shows an edge to be swapped that is surrounded by five tetrahedra. Figures II.4(c), (d) and (e) depict three of the five possible configurations after the swap. The retained configuration is the one that has the best minimal element quality in the cavity.



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

Figure II.4: Edge swap operation: (a) initial cavity with $n = 5$, (b) mean surface that is going to be triangulated, (c),(d) and (e) three possible configurations after the swap.

The face swap removes a face and replaces it by an edge, leading to the creation of three tetrahedra instead of two. This operation can be seen as the inverse edge swap with $n = 3$.

The face collapse operator can be seen as the compound of an edge split and an edge collapse. The operation is depicted in Figure II.5. One of the edges of the face is split and the resulting new edge is collapsed on the new vertex. This operation is particularly well suited to eliminate an ill-shaped

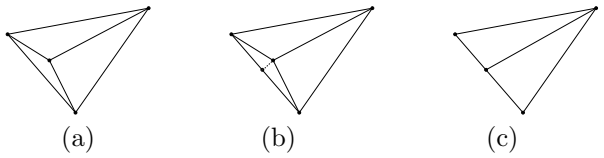


Figure II.5: Face collapse operation: (a) initial tetrahedron, (b) split of an edge of the concerned face, (c) collapse of the new edge on the new vertex.

element that has a face with a small ratio area/edge lengths.

The double edge split collapse compound operator consists in splitting two edges of a tetrahedron and then collapsing the edge joining the two new vertices. This operator is used when a tetrahedron with a small volume has two edges nearly intersecting. The operation is depicted on Figure II.6.

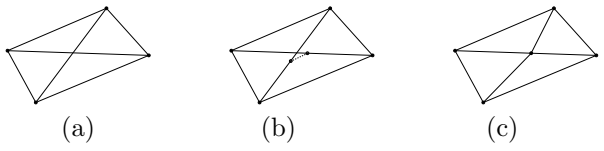


Figure II.6: Double edge split + edge collapse operation: (a) initial tetrahedron, (b) situation after the edge splits, (c) situation after the edge collapse.

II.4 Sliver tetrahedra handling

A tetrahedron is said to be a sliver when it has a small volume and no short edge. Such a tetrahedron can be classified in one of the two categories [32] depicted on figure II.7. Type I slivers are tetrahedra in which two edges almost intersect. In type II slivers, one vertex is very close to its opposite face. The determination of the sliver type is important because it gives a useful information about the best local mesh modification that can be applied in order to eliminate it. The projection algorithm presented in [106] is used in this work to determine the type of sliver.

In FSI, when the mesh motion solver is pushed to its limits, it inevitably generates slivers near the boundaries undergoing a motion. In [106], an algorithm selecting the local mesh modifications to be applied was proposed to eliminate most of them. In this work, we propose an extension to this algorithm that eliminates more slivers.

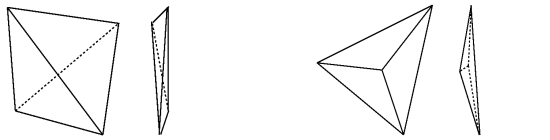


Figure II.7: Classification of the sliver tetrahedra: (a) type I, (b) type II.

A new sequence of mesh modification operations is proposed in Table II.1. The operations are sorted with respect to their average efficiency to eliminate slivers of type I and II. The efficiency measure that we have used takes into account both the rate of success of eliminating slivers and CPU time consumption of the given operator. The rates of success of the different operations have been deduced from various test cases. Some of them are presented in section II.7. We notice that the vertex motion is the very last solution as it is highly probable that the tetrahedron becomes a sliver again after the next step of the boundaries motion. The target location for the motion is computed with the method proposed in [56].

Type	Priority	Local mesh modification
I	1	Split one of the two key edges
	2	Collapse one of the edges of the tetrahedron
	3	Split both key edges and collapse the new interior edge
	4	Split one of the key edges and collapse the new vertex
	5	Swap one of the two key edges
	6	Relocate a vertex
II	1	Collapse one of the edges of the tetrahedron
	2	Collapse the key face
	3	Swap any of the edges bounding the key face
	4	Swap the key face
	5	Relocate a vertex

Table II.1: Sequence of local mesh modifications attempted to eliminate a sliver.

Note that those sequences are designed to be part of the sliver elimination algorithm inserted in the global adaptation procedure presented in section II.6. Other sequences could be more efficient if the global procedure was modified.

In this work, we consider that eliminating the slivers is more important than enforcing locally the length criterion, which means that an operation that can eliminate a sliver or at least improve the local quality is performed, even if it creates long or short edges. The long or short edges will be eliminated later on by the global procedure but only if they do not create a sliver tetrahedron (see section II.6).

The construction of the list of sliver tetrahedra is made by computing the quality of every element of the mesh by (II.2). An element is considered as a sliver (type I or II) if its quality is under a certain threshold Q_{sl}. In all simulations presented in this paper, Q_{sl} is set to 0.01.

II.5 Mesh motion solver

In FSI problems, the structure imposes its motion to some of the boundaries of the fluid domain. Moving only the boundary nodes leads to the generation of ill conditioned tetrahedra. A way to circumvent partially this issue is to relocate the nodes of the volume using a linear elasticity analogy [186, 13].

A linear elastic problem is solved every time the mesh is moved. Here, we use the approach of [177] in which a stiffness alteration based on the Jacobian of the elements is used to stiffen the smaller elements. The corresponding finite element formulation is written as follows. Consider the vector of displacements of the nodes $\mathbf{y}^h \in \mathcal{S}^h$, where $\mathcal{S}^h$ is the piecewise linear nodal finite element space. The modified energy of deformation of a tetrahedron e submitted to a displacement $\mathbf{y}^h$ (with the corresponding strain and stress tensors $\boldsymbol{\epsilon}$ and $\boldsymbol{\sigma}$) is written as:

$$\frac{1}{2}\int_e \left[\boldsymbol{\epsilon}(\mathbf{y}^h) : \boldsymbol{\sigma}(\mathbf{y}^h)\right]^e J^e \left(\frac{J^0}{J^e}\right)^{\chi} dv, \tag{II.3}$$

where χ is the stiffening parameter and J^0 a fictitious volume constant over the mesh.

Figure II.8 illustrates the influence of χ. With $\chi = 0$, we observe ill shaped elements near the walls. The elastic solver fails to achieve the mesh deformation with positive volumes of the elements if a motion of the cube of more than $0.1 \times h$ where h is the size of the cube is imposed. With $\chi = 1$ and $\chi = 2$, a motion of $1.0 \times h$ can be attained without any particular problem. In the $\chi = 2$ case, the elements are very well preserved around the cube but the quality falls outside of the refined zone. The choice of $\chi = 1$ seems to be a good compromise and has been used in all the computations presented in this paper.

II.6 Global mesh modification procedure

Our FSI procedure has two different stages: the node repositioning stage and the mesh adaptation stage. The node repositioning step does not imply any change in the mesh topology. As it does not need to reallocate the resources for the storage of the topology, the solution or any data, it can be performed often at a reasonable computational cost, that is, each time a displacement is prescribed at the interface between the fluid and the structure. Moreover, the node repositioning is explicitly taken into account in the ALE formulation of the governing equations and does not need any mesh to mesh interpolation. In the second step of the procedure, the mesh is adapted using the local mesh modifications presented in section II.3 and the global size field presented in section II.2.

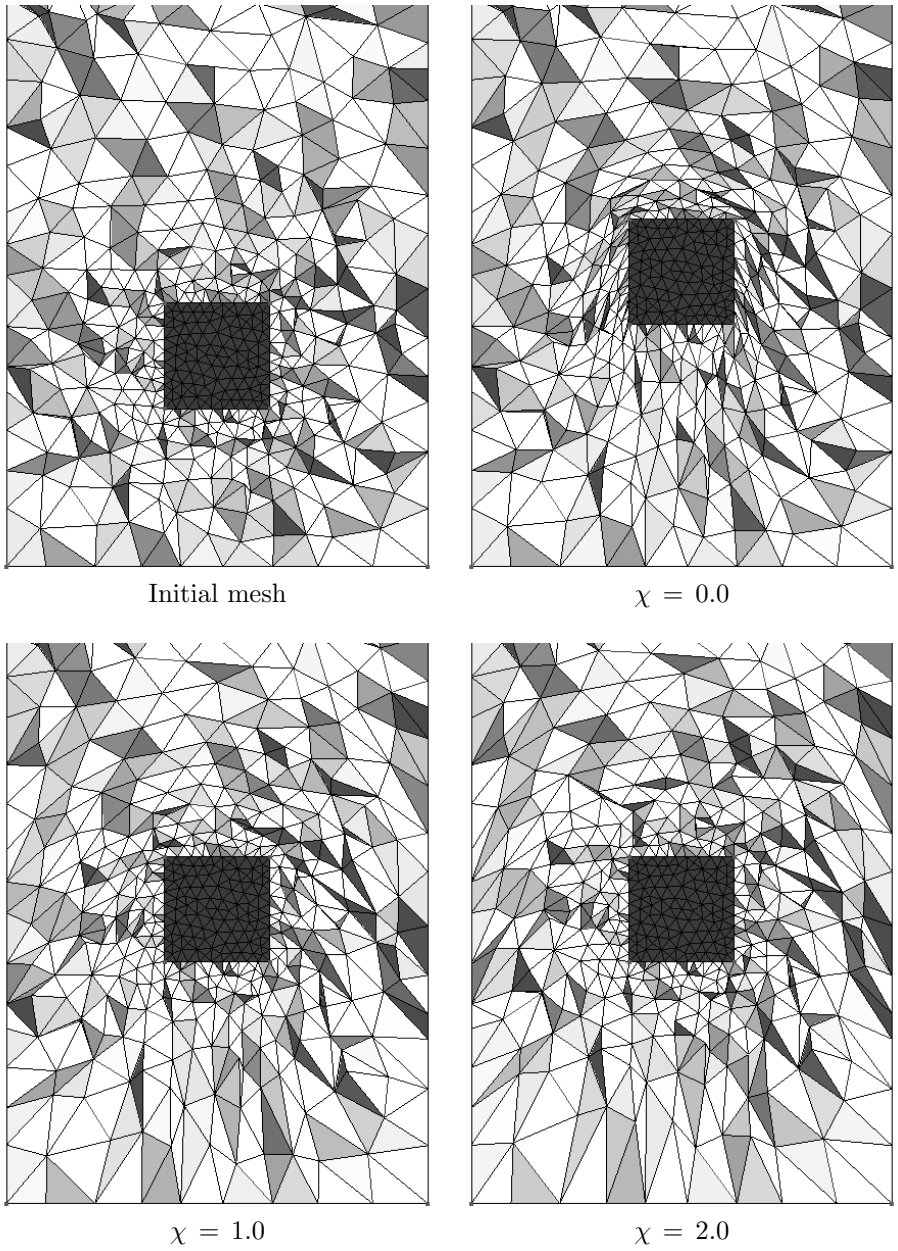


Figure II.8: Deformation of the mesh around a moving cube (straight motion) for various values of χ in (II.3).

The mesh adaptation algorithm

The aim of the adaptation procedure is twofold. First, the dimensionless size of all edges of the mesh have to lie in the interval $[L_{low}, L_{up}]$. Then, the quality of the resulting mesh has to be optimal. Minimal quality criterions can be expressed for instance in terms of a minimal quality for every element and/or a minimal mean quality of the elements.

The mesh adaptation procedure can be described as follows:
`Do {`

- Collapse short edges i.e. edges that have lengths $L_e^{tr} < L_{low}$.
- Do an edge swap loop: loop over all edges and compute the minimal quality of the elements surrounding the edge. The quality is given by the cubic mean ratio (equation II.2). If the quality is lower than a given threshold Q_{swap}, try to find a swap configuration. Apply the swap if it improves the minimal quality in the cavity and if it does not create a long or a short edge. The threshold Q_{swap} is fixed to 0.1 in all the simulations presented in this paper.
- Eliminate sliver tetrahedra with the algorithm described in section II.4.
- Split long edges i.e. edges that have lengths $L_e^{tr} > L_{up}$.

`} While the mesh is modified.`

The collapse loop is performed first in order to avoid memory peaks. Indeed, the number of nodes decreases during this loop. As the edge swap loop is costly, it is performed just after the coarsening loop, with the minimum number of nodes. The sliver region elimination can then be performed without having to compute any element shape, as all element qualities are computed at the previous step.

Infinite loops

There are a few ways in which the adaptation procedure can degenerate in an infinite loop between two or more configurations.

If the interval of tolerance for the length of the edges is not large enough, an infinite loop between split and collapse operations can appear. Due to the heuristic nature of the mesh adaptation, we cannot guarantee that such a loop will never appear, even for a large interval. For that reason, a maximum number of iterations is imposed in the global procedure. The edge length interval should then be carefully chosen in order to strongly limit the number of infinite loops without being too far from the unit mesh. More detailed results are presented in section II.7.

Another possibility of having an infinite loop is illustrated in Figure II.9, in which an edge split is followed by an edge collapse and an edge swap. This is possible because the goal of the edge split and collapse operators is to respect a criterion on edges length while the edge swaps tend to improve the quality of the tetrahedra. A way to eliminated such a scenario is to forbid an edge swap if it creates a long or short edge.

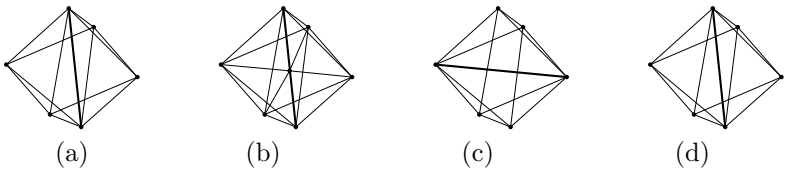


Figure II.9: Infinite loop between an edge swap, an edge split and an edge collapse: (a) initial cavity, (b) cavity after the split, (c) cavity after the collapse, (d) cavity after the swap, identical to (a).

Finally, the operators used in the sliver regions handler can also create a set of complex infinite loops. This possibility can be avoided if any operation that creates a sliver tetrahedron is forbidden. Of course, as an exception, we allow the replacement of a sliver by a better sliver.

II.7 Computational results

Our adaptation procedure has been evaluated on three test cases. The quality of the meshes, the elimination of the sliver tetrahedra and the evolution of the number of nodes are presented in the first and the second cases. The third test case is a fluid-structure interaction computation with a very large motion of one or two spheres in a fluid at rest.

Cylinder and tube

This test case consists in the penetration of a cylinder in a tube, the diameters of the objects being quite close. The initial geometry and mesh are shown in figure II.10. The cylinder has a radius of 0.9 and a length of 3.0. The tube has the same length with an internal radius of 1.0 and an external radius of 1.2. The initial distance between the two objects is 1.0. The two objects move with a velocity of 1.0 for the cylinder and -1.0 for the tube. The end of the test is fixed at time 4.0, when the objects are completely separated and the distance between them is 1.0.

A global edge length of 0.6 is prescribed on the whole domain and both objects are equipped with a local size field. The parameters of the local size fields for the cylinder and the tube are

$$\left\{ \begin{array}{l} d_{cyl}^{\min} = 0 \\ d_{cyl}^{\max} = 1.0 \end{array} \right. \quad \left\{ \begin{array}{l} \delta_{cyl}^{\text{small}} = 0.1 \\ \delta_{cyl}^{\text{large}} = 0.6 \end{array} \right. \quad \left\{ \begin{array}{l} d_{tube}^{\min} = 0 \\ d_{tube}^{\max} = 1.0 \end{array} \right. \quad \left\{ \begin{array}{l} \delta_{tube}^{\text{small}} = 0.4 \\ \delta_{tube}^{\text{large}} = 0.6 \end{array} \right.$$

For an interval $[L_{low}, L_{up}]$ set at $[0.5, 1.4]$, the number of nodes ranges from about 23.500 to 33.000 during the adaptation. This is sufficient to get representative data and statistics about the mesh quality and the sliver elements handling.

A time step of 0.01 has been chosen for this test case. It corresponds to a relative motion of the objects of $0.2 d_{cyl}^{\min}$ at each time step.

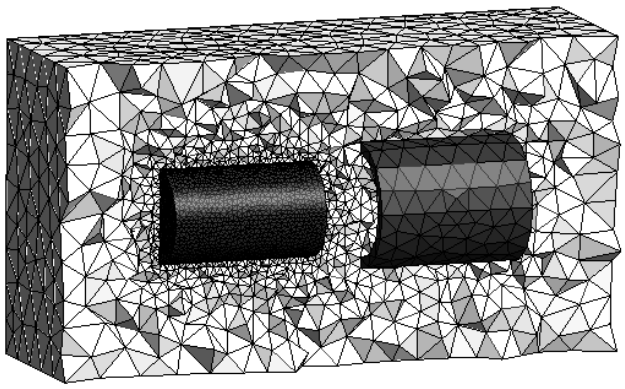


Figure II.10: Cylinder and tube test case: cut in the initial mesh.

The mesh aspect obtained at different times with a time step of 0.01 and an interval $[L_{low}, L_{up}]$ set at $[0.5, 1.4]$ is shown in figure II.11.

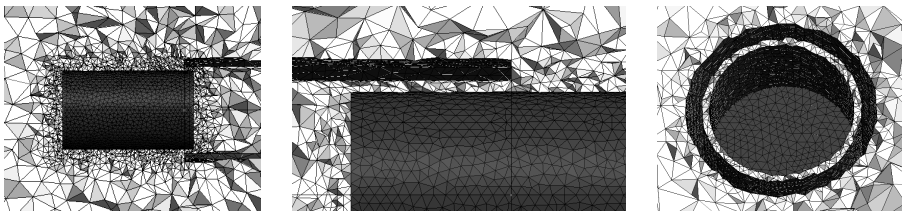


Figure II.11: Mesh aspect at different times with $\Delta t = 0.01$, $L_{low} = 0.5$ and $L_{up} = 1.4$.

Sliver tetrahedra handling: In order to get more information about the efficiency of each mesh modification applied in the sliver region elimination, we have run the adaptation procedure in which the slivers handling algorithm is enhanced with a routine that tests all operations individually on each sliver before going in the normal algorithm of elimination. This routine has no influence on the results, but provides the statistics shown in table II.2. The first columns indicates the number of sliver tetrahedra that the operators can eliminate without creating a new sliver. The second column shows the number of situations in which the operator can only eliminate the sliver by creating another sliver with a better quality, while the third column shows the number of slivers that could not be eliminated or for which a worse sliver would appear. The rate of success of the operator (not including the cases in which a new sliver is created) is indicated in the last column.

We can see that combining all operators does not lead to a total elimination of the slivers but provides a very good rate of elimination. The vertex

	Successes	New sliver	Failures	Success rate (%)
Type I (547)				
All operators	541	1	5	98.9
Edge collapse	244	3	300	44.6
Edge split	346	20	181	63.3
Double split+collapse	210	11	326	38.4
Face collapse	279	13	255	51.0
Edge swap	306	17	224	55.9
Vertex relocation	504	10	33	92.1
Type II (308)				
All operators	306	1	1	99.4
Edge collapse	119	2	187	38.6
Face collapse	155	1	152	50.3
Edge swap	202	5	101	65.6
Face swap	112	6	190	36.4
Vertex motion	292	1	15	94.8

Table II.2: Tube and cylinder test case: statistics about sliver tetrahedra handling.

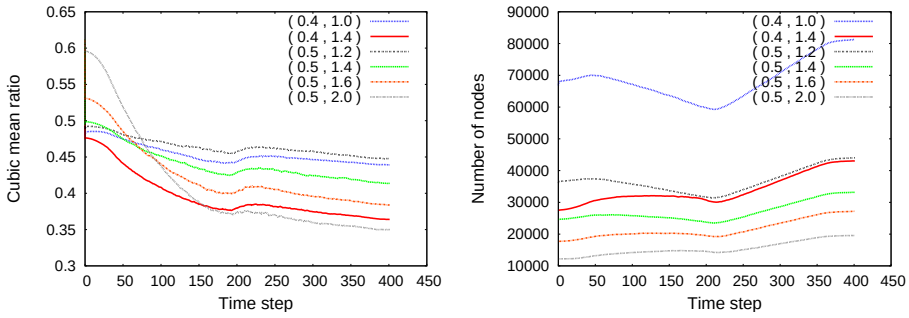
repositioning looks attractive but we observed that many slivers eliminated by a repositioning become slivers again once a global node motion is applied. For type I slivers, the edge split is very efficient if we notice that it is a very fast operator compared to the edge swaps. Note that those results are influenced by the fact that the sliver elimination is done after the edge collapse and edge swap loops and before the edge split loop.

If a sliver cannot be eliminated by the present method, an alternative is to use a local re-meshing technique [52]. In that case, a cavity including the sliver has to be deleted and re-meshed. The size of this cavity will be iteratively increased until it can be meshed without any sliver element.

Mesh quality and infinite loops: A particular result that we observe in our simulations is that the mean quality of the mesh depends on the boundaries of the interval $[L_{low}, L_{up}]$. Figure II.12 (a) shows the evolution of the quality for various intervals, figure II.12 (b) shows the evolution of the number of nodes in each case, while table II.3 shows the number of time steps in which an infinite loop was created.

Interval	$[0.4, 1.0]$	$[0.4, 1.4]$	$[0.5, 1.2]$	$[0.5, 1.4]$	$[0.5, 1.6]$	$[0.5, 2.0]$
Inf. loops	400	0	400	81	0	0

Table II.3: Tube and cylinder test case: comparison between intervals $[L_{low}, L_{up}]$: number of infinite loops.



(a) (b)

Figure II.12: Cylinder and tube test case: comparison between intervals $[L_{low}, L_{up}]$: (a) evolution of the mean quality of the mesh and (b) the number of nodes.

We can see that the condition $L_{low} < 0.5\, L_{up}$ is not sufficient to avoid infinite loops. Indeed, there is an infinite loop at each time step with the intervals $[0.4, 1.0]$ and $[0.5, 1.2]$ while some infinite loops appear with $[0.5, 1.4]$. We would then recommend to use an interval with a similar range than $[0.4, 1.4]$ or $[0.5, 1.6]$.

We also observe that the quality is dependent on the boundaries of the interval. If we compare the intervals $[0.4, 1.4]$ and $[0.5, 1.4]$, the quality seems to be better with the largest L_{low}. If we examine the results for $[0.5, L_{up}]$, we can see that the quality decreases to a smaller value when L_{up} is bigger. These two observations lead to the conclusion that the quality tends to a bigger value if the interval is smaller.

In order to get the best mean quality without running into infinite loops, a compromise has to be found on the size of the interval. The smaller interval which do not lead to infinite loop should be used. From the set of intervals tested here above, our choice would turn towards $[0.4, 1.4]$ or $[0.5, 1.6]$.

Worm screw

The second test case is a worm screw whirling inside a cylinder. It has been chosen to demonstrate the capabilities of the adaptation procedure on relatively complex geometries.

Figure II.13 shows the geometry and the initial mesh of the domain at time 0. The domain is bounded by a worm screw of diameter 4.94 and length 15.6 turning with an angular speed of 1.0, and a fixed cylinder of diameter 5.2 and length 16.2. The computation ends when a complete turn has been operated, i.e. at $t = 2\pi$. A time step of 0.02 is chosen.

The maximum size on the whole domain is set to 1.0, while a size field with the following parameters is prescribed at every wall:

$$d_1^{\min} = 0 \quad d_1^{\max} = 0.4 \quad \delta_1^{\text{small}} = 0.2 \quad \delta_1^{\text{large}} = 0.4$$

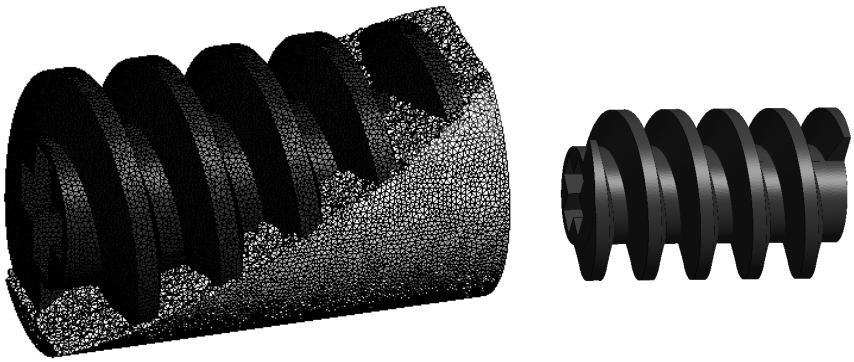
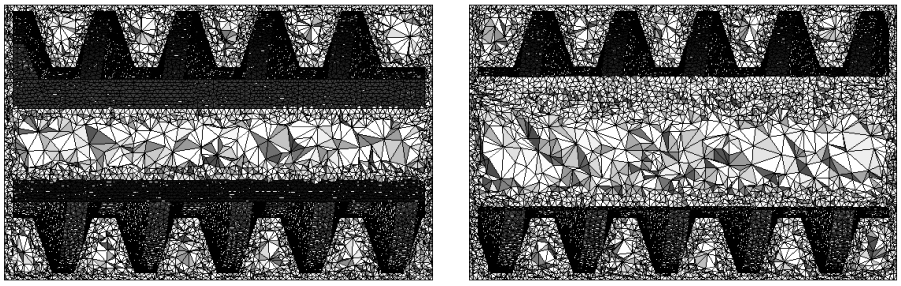


Figure II.13: Worm screw: initial geometry and mesh.

The interval $[L_{low}, L_{up}]$ is set to $[0.5, 1.6]$. With these parameters, the number of nodes ranges from 125.000 to 155.000 during the computation.

Figure II.14 shows the aspect of the mesh at times 4.0 and 6.0.

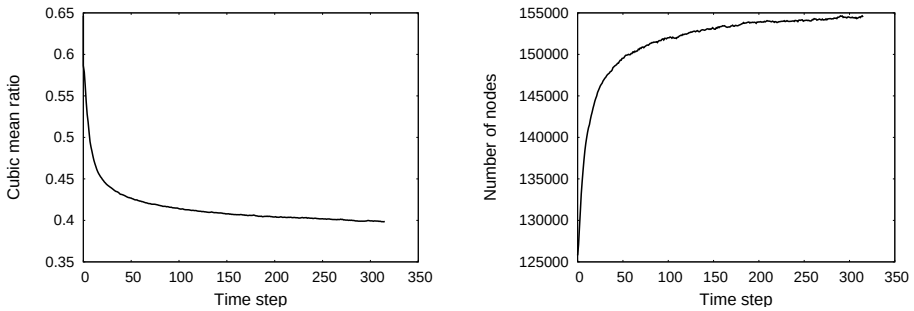


(a) (b)

Figure II.14: Worm screw: cuts in the mesh at times 4.0 and 6.0.

The evolution of the mean quality is shown in figure II.15 (a). The cubic mean ratio tends to a value close to 0.40, which is quite similar to what was observed in the previous test case with the same interval $[L_{low}, L_{up}]$. The evolution of the number of nodes is drawn on figure II.15 (b). It stabilizes around 155.000. The quality distributions in the initial and final meshes are shown in figure II.16. Note that Q_{swap} is set to 0.1 here, which means that edge swaps have only be attempted to improve elements for which quality was lower than 0.1.

The statistics about the elimination of the sliver tetrahedra are shown in table II.4. Every sliver is eliminated and most of them are eliminated at the first attempts.



(a) (b)

Figure II.15: Worm screw: (a) evolution of the mean cubic mean ratio, and (b) evolution of the number of nodes.

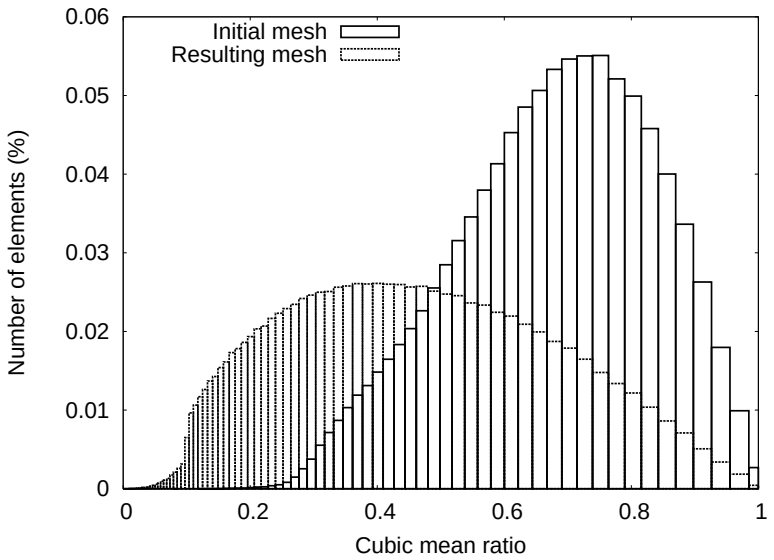


Figure II.16: Worm screw: quality distributions in the initial and final meshes.

	Applied (no sliver)	Applied (new sliver)
Type I (1532)		
Edge split	1182	0
Edge collapse	239	0
Double split+collapse	15	1
Face collapse	18	0
Edge swap	4	1
Vertex relocation	72	0
Remaining	0	
Type II (1521)		
Edge collapse	1430	0
Face collapse	33	1
Edge swap	17	0
Face swap	1	0
Vertex motion	38	1
Remaining	0	

Table II.4: Worm screw: statistics about sliver tetrahedra handling.

Spheres falling in a fluid

In this test case, the mesh adaptation procedure is applied to a fluid-structure interaction solver. The problem solved implies the motion of one or two spheres falling in a fluid.

The fluid is governed by the incompressible Navier-Stokes equations. The equations are discretized in a dual finite elements / finite volumes formulation.

The motion of the sphere is computed with the Newmark method and the coupling is achieved with a CSS procedure [140] equipped with sub-iterations cycles.

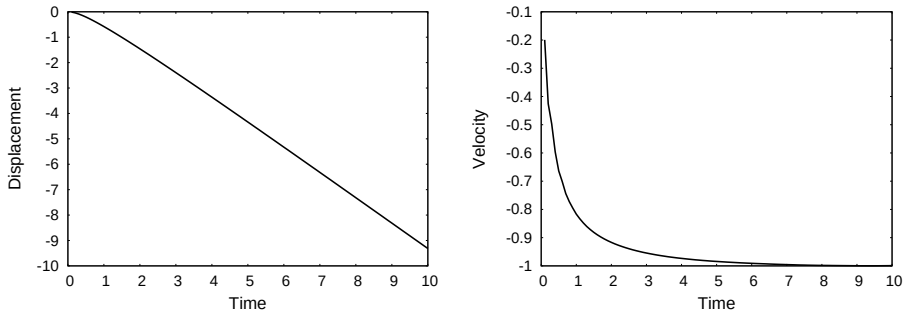
Single sphere

For this computation, a single sphere is immersed in the fluid. We fix the parameters of the fluid so that the Reynolds number Re is equal to 1 at equilibrium. Due to the gravity, the sphere accelerates until the gravity is exactly balanced by the drag force and the Archimede's force. We fix the mass of the sphere such that the theoretical velocity at equilibrium is 1.

Figure II.17 shows the evolution of the displacement and velocity of the sphere. We notice that the velocity tends to 1 as expected. The evolution of the mean quality and the final distribution of elements quality are shown on Figure II.18. We observe that the quality decreases slowly so that the loss along the computation is not significant.

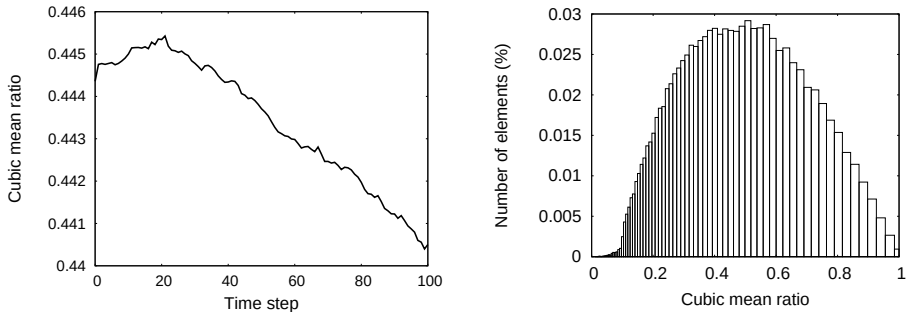
Two spheres

In the next computation, the motion of two vertically aligned spheres are investigated.



(a) (b)

Figure II.17: Single sphere test case: (a) evolution of the displacement, and (b) velocity.



(a) (b)

Figure II.18: Single sphere test case: (a) evolution of the mean quality of the mesh, and (b) elements quality distribution at time 10.

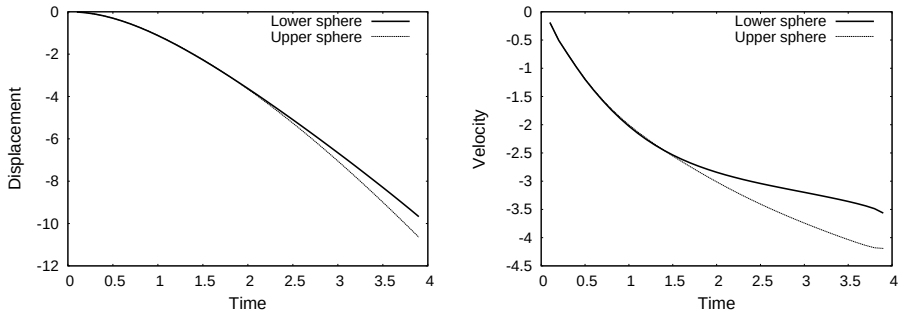
The evolution of the displacements and the velocities of the spheres are shown in figures II.19 (a) and (b). We observe that from the time 1.5, the upper sphere is aspirated by the flow around the first one. Eventually, the spheres touch and the computation stops.

The cut of the computational mesh is represented on figure II.20 for different times as well as the pressure field.

II.8 Conclusion

A procedure to handle large deformations of a mesh in FSI problems by nodes movement and local mesh adaptation was presented. The procedure is robust and can handle complex geometries with very large deformations of the domain.

For that purpose, a node repositioning technique with a selective treatment for the elements previously applied for two-dimensional meshes was applied for three-dimensional meshes.

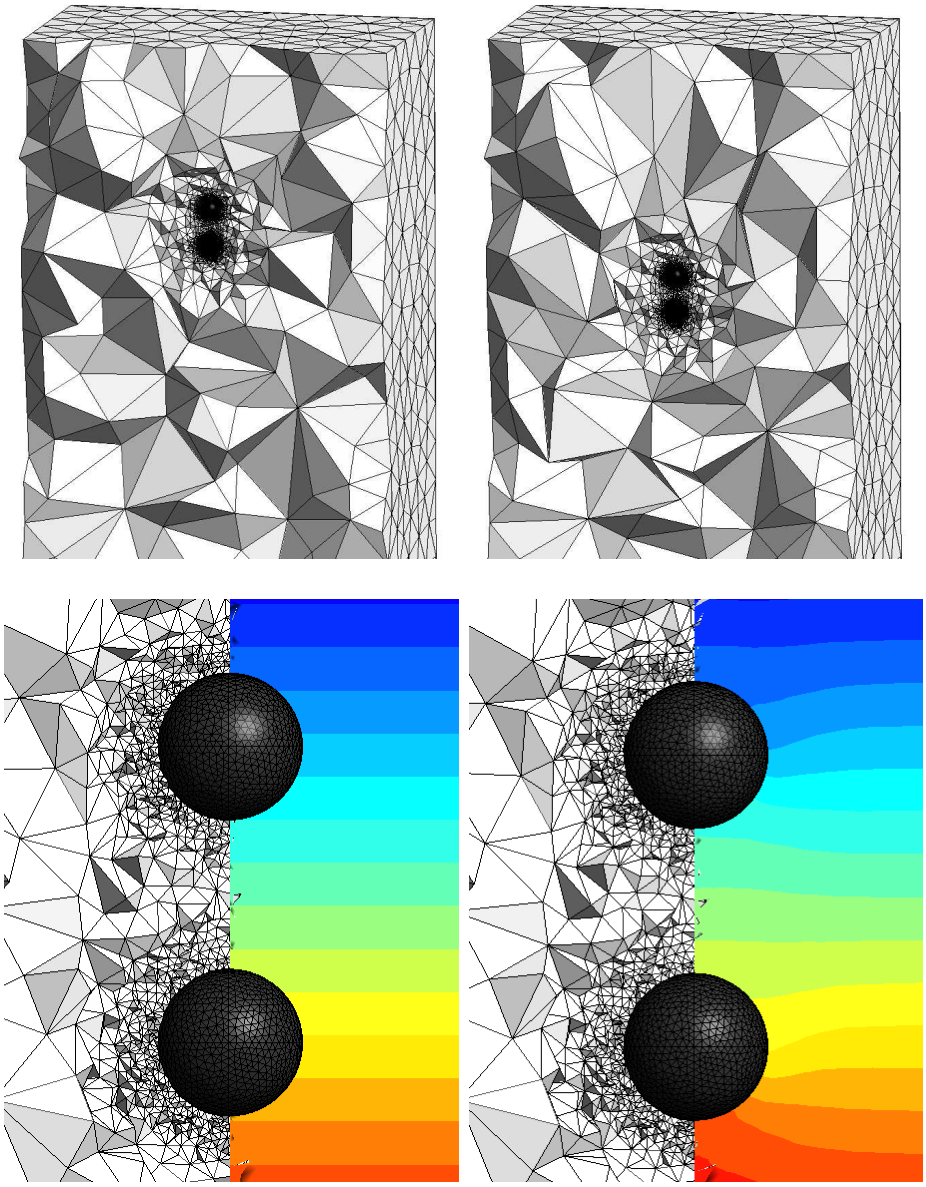


(a) (b)

Figure II.19: Two spheres test case: (a) evolution of the displacements and (b) velocities.

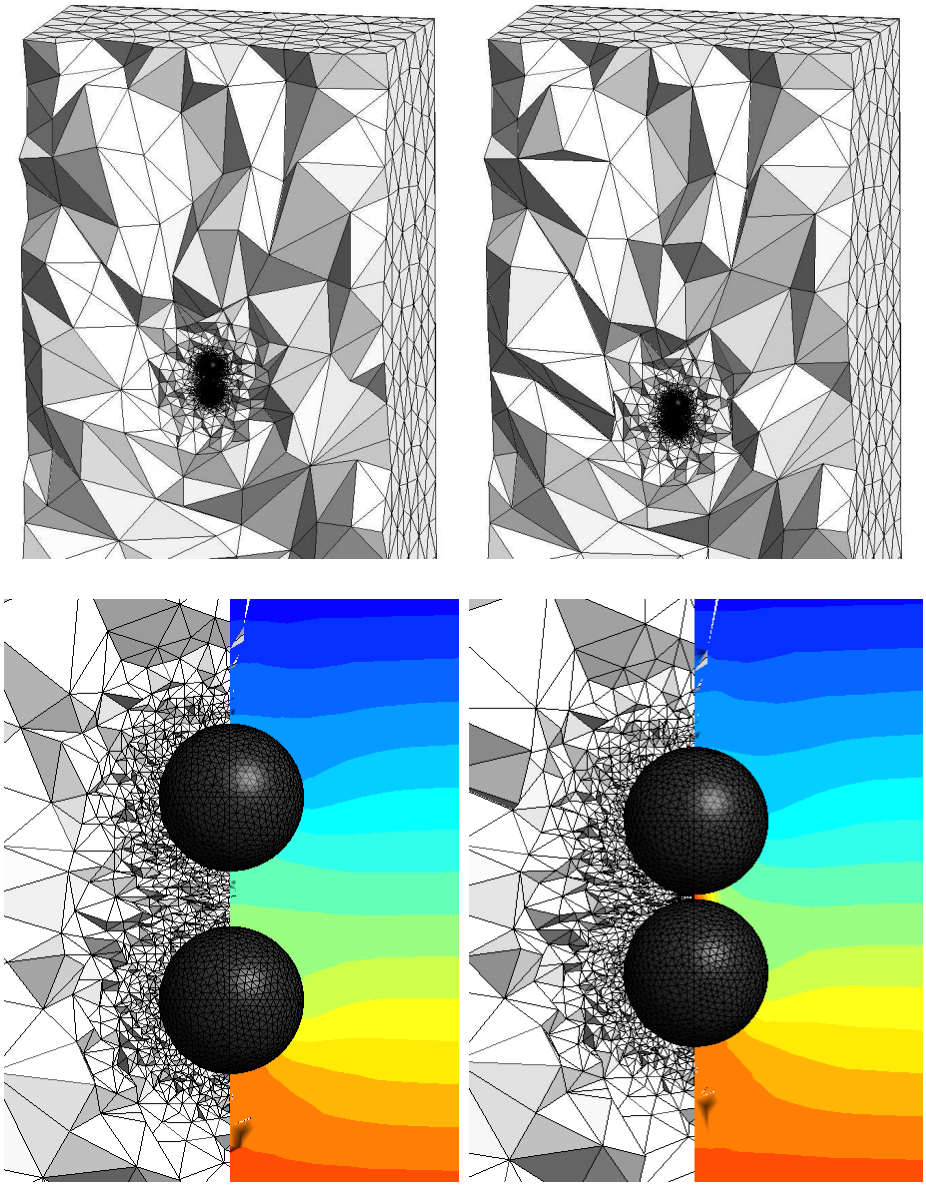
A new adaptation procedure has been proposed in which an efficient handling of the sliver tetrahedra is included. The procedure achieves a good quality for the mesh and complies a mesh size field. Important parameters of the procedure like the boundaries of the interval $[L_{low}, L_{up}]$ were studied with a set of computations. In particular, some clues are given to choose them by looking at the quality, number of nodes and production of infinite loops.

Finally, mesh deformation tests and fluid-structure computations have been performed in order to show the potential of the presented approach.



Time: 0.0 2.0

Figure II.20: Two spheres test case: mesh and pressure field at different time steps (continued on next page).



Time: 3.4 3.9

Figure II.20: Two spheres test case: mesh and pressure field at different time steps (continued).

Chapter 3

Adaptive boundary layer meshes

Capturing the boundary layer is an important point in most of the CFD computations. The behaviour of the flow around a wall deserves a particular attention for several reasons: (i) at high Reynolds numbers, the boundary layer can have a significant impact on the global solution, in the case of a boundary layer separation for instance, (ii) for low Reynolds flows, although the global flow does not significantly depend on the boundary layer, local quantities of interest like the wall shear stress can be dramatically affected by the error on the boundary layer computation. Furthermore, high variations of the velocity occur in the boundary layer, which requires an appropriate design method for the mesh.

In the literature as well as in the commercial softwares, solutions are usually proposed to build structured or semi-structured simplicial or non-simplicial meshes [121, 84, 98, 74, 93, 169] for the boundary layer mesh. Those methods rely on an extruded mesh around the no-slip walls and the issues raised by geometrical configurations like corners, sharp ridges, concave cavities, ... (see Figure 3.1) are solved by ad-hoc algorithms. The complexity of the global method increases with the apparition of new geometrical features. In some cases, manual interventions are even required. In the industry, it results in a significant amount of human time spent in preparing a CAD model and meshing it, regarding the computational time of the CFD simulation.

Furthermore, the common meshers can yield meshes with a non-optimal distribution of the elements, as in the example presented in Figure 3.2, and non-optimal element quality. Finally, the directions induced by the layers can affect the solution of the CFD solver. All those issues are raised by the structured nature of the mesh.

When dealing with mesh adaptation, structured meshes raise other issues, like the preservation of the structure of the adapted mesh, or the elements validity and quality control with large deformations. Its only recently that the issues related to mesh adaptivity for 3D semi-structured meshes started to receive some attention in the literature with the works of Sahni et al., who de-

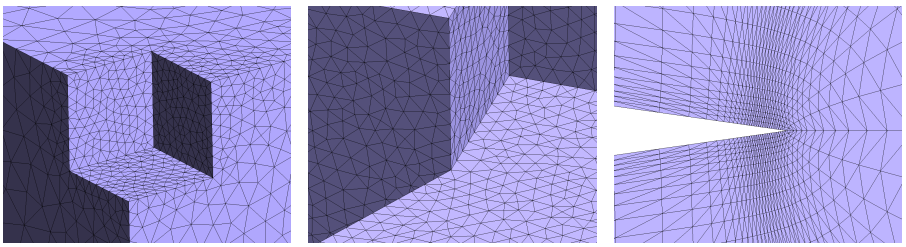


Figure 3.1: Examples of geometrical configurations in which specific techniques are needed in order to build structured meshes.

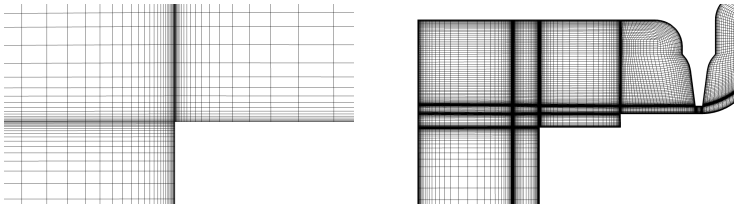


Figure 3.2: Structured mesh with non-optimal element distribution.

signed a method to adapt semi-structured meshes for viscous flow computations when using linear [161] and curved [162] meshes.

To sum up, working on structured or semi-structured meshes is a strong constraint. As it will be discussed in the first section of this chapter, constraining the mesh to be structured in the boundary layer regions is not always justified. For inviscid and low Reynolds number flows, unstructured simplicial meshes lead to reasonable results, but with a numerical noise [163, 161]. For the high Reynolds number flows however, the reliability of the unstructured meshes currently proposed in the literature has still to be improved.

An attempt to generate anisotropic unstructured boundary layer meshes with a reasonable quality is presented in [37] for 2D flows, where it is proposed to design the mesh metric field in such a way that the eigenvectors of the metric are aligned with the normal and tangent directions of the wall. It is also proposed to relocate nodes in order to force their alignment with the tangent directions. The approach has not been extended to 3D yet, and the issue of building suitable mesh metric fields around curved boundaries is not handled in the paper.

In this chapter, we investigate a method to automatically build and adapt an unstructured mesh including viscous boundary layers with the mesh adaptation procedure described in Chapter 2 by modifying the size field near the no-slip walls. Such a method has several advantages:

- the mesh can be build automatically based on general specifications of the boundary layer mesh like layer thickness and number of elements across the thickness,

- the method can be easily used to build and adapt meshes based on an a posteriori error estimate (in the boundary layer as elsewhere), like the Hessian of a solution [163],
- the method is independent of the complexity of the geometry, as it uses the techniques described in Chapter 4 for adapting the mesh on the boundaries,
- the domain can be arbitrarily deformed,
- the mesh size field can be smoothed (see for instance [28] for anisotropic size field smoothing) in order to control the transition between the boundary layer mesh and the rest of the domain.

Following the idea of [37], the size field is built in such a way that its eigenvectors are aligned with the normal and tangent directions of the wall. A particular attention is given to the computation of an appropriate prescribed edge length near curved boundaries. Indeed, since it is impossible to fill a curved region with highly anisotropic elements, the specified sizes around curved surfaces should take its curvature into account.

3.1 Testing the non-structured BL meshes: the flat plate test

In order to compare structured and unstructured meshes for boundary layer modeling in viscous flows, we show here a very simple example for which an analytical solution is known: the plate test.

Flow problem The configuration of the test is depicted in Figure 3.3. The plate has a length $L = 2.07\,m$, and a region of length $0.67\,m$ is left upstream of the leading edge in order to capture the leading edge flow. The far stream is located at $0.5\,m$, which is far enough to consider it as fairly uniform. Since a 3D fluid solver is used to compute the flow, the domain is extruded in the spanwise direction to the plane and symmetry boundary conditions are applied on the two resulting planes.

The fluid enters in the domain with constant velocity $U = 1\,m/s$ and pressure $p = 1\,Pa$ in space and time. The same boundary conditions are applied to the top boundary. No-slip and slip wall boundary conditions are applied respectively to the plate and the bottom boundary in the upstream region. At the outlet, the boundary conditions are a pressure $p = 1\,Pa$ and a zero strain tensor $\mathbf{S}$:

$$\mathbf{S} = \mu\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right) = \mathbf{0}, \tag{3.1}$$

where μ is the dynamic viscosity of the fluid. The flow is incompressible and the equations solved are the incompressible Navier-Stokes equations. The following density ρ and dynamic viscosity μ of the fluid are chosen

$$\rho = 1\,kg/m^3, \quad \mu = 10^{-4}\,m/s^2. \tag{3.2}$$

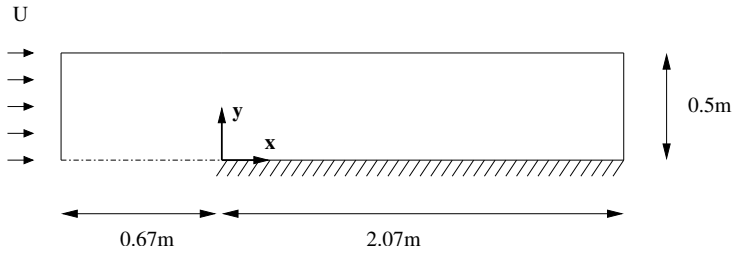


Figure 3.3: Geometrical settings for the plate test (reproduced from [117]).

With the values given here above, the Reynolds number of the flow corresponding to the length of the plate is computed by

$$Re = \frac{\rho L U}{\mu} = 2.07\,10^4. \tag{3.3}$$

The transition from laminar to turbulent flows occurs around $Re = 5\,10^5$.

The quantity of interest we choose to compare the different solutions is the skin friction coefficient C_f, which is given by

$$C_f = \frac{\tau_w}{\frac{1}{2}\rho u_\infty}, \tag{3.4}$$

with the wall shear stress τ_w

$$\tau_w = (\mathbf{S}.\mathbf{n}) \cdot \mathbf{t}. \tag{3.5}$$

Theory A reference solution for this problem has been obtained by Blasius (see for instance [188]), for which the theoretical skin friction coefficient is given by

$$C_f = \frac{0.664}{\sqrt{Re_x}}, \tag{3.6}$$

where Re_x is the Reynolds number at abscissa x given by

$$Re_x = \frac{\rho x u_\infty}{\mu}, \tag{3.7}$$

with u_∞ the velocity at an infinite distance from the plate. We assume that the domain is large enough to write $u_\infty = U$.

According to the solution of Blasius, the thickness $\delta^*_{99\%}$ of the boundary layer, defined as the distance at which $u = 0.99\,u_\infty$, is given by

$$\delta^*_{99\%}(x) \approx 3.5\sqrt{\frac{2\nu x}{u_\infty}} \approx 4.95\frac{x}{\sqrt{Re_x}}, \tag{3.8}$$

with $\nu = \mu/\rho$ the kinematic viscosity of the fluid.

Computational settings The fluid solver used to solve the incompressible Navier-Stokes equations is *Argo*, which is developed by Cenaero[1]. Argo uses a mixed finite elements / finite volumes (convective fluxes) method. The non-linear problem is solved by a Newton method. In this work, we choose to solve the linear system with a GMRES solver together with an ILU preconditionner and a fill level of 2.

The stabilization of the incompressible flow is done by a PSPG method where the stabilization parameter is chosen as constant and set to 0.1.

The computation is performed with six different meshes. Structured and non-structured meshes will be compared both on relatively coarse, medium and fine meshes.

The meshes are depicted on Figure 3.4. The corresponding structured and non-structured meshes have approximately the same element size distributions in the boundary layer. The structured and unstructured meshes have respectively 3810 and 3051 elements (5 elements in the boundary layer), 4634 and 3835 elements (6 elements in the BL), and $124,800$ and $101,372$ elements (over-refined meshes). The difference between the structured and unstructured mesh sizes comes from the refined zone out of the region of interest in the structured case. An additional mesh with only 2583 elements but a refinement located in the region of the boundary layer defined by (3.8) is also evaluated. Such a mesh could typically be obtained by an adaptive computation although here the solution of Blasius was used.

Computational results Figure 3.5 depicts the skin friction coefficient C_f along the plate obtained with the structured and unstructured meshes. We observe that the unstructured meshes yield results which are as close to the theoretical solution as the structured ones, but with a numerical noise. The same observations are made from the results presented by Sahni et al. [163,161] in the case of cylindrical pipes. We also notice that the mesh refined in the boundary layer region defined by (3.8) yields better results than the simple unstructured coarse mesh although it has less elements. When the mesh is over-refined, Figure 3.5(c) shows that both types of meshes converge to the same solution.

The velocity profiles at different abscissa x and for $y \in [0, 0.05]$ obtained with the unstructured mesh adapted according to the Blasius solution are depicted on Figure 3.6.

In the unstructured meshes, the element shapes are variables and therefore could be locally worse than in the corresponding structured meshes. However, the convergence of the computation is not affected, as shown in Figure 3.7, where only a structured mesh yields a degradation of the convergence.

3.2 General method

The solution we propose to generate or adapt an anisotropic boundary layer mesh relies on two components: the mesh adaptation method described in

[1]Center for Excellence in Aeronautics, Gosselies, Belgium

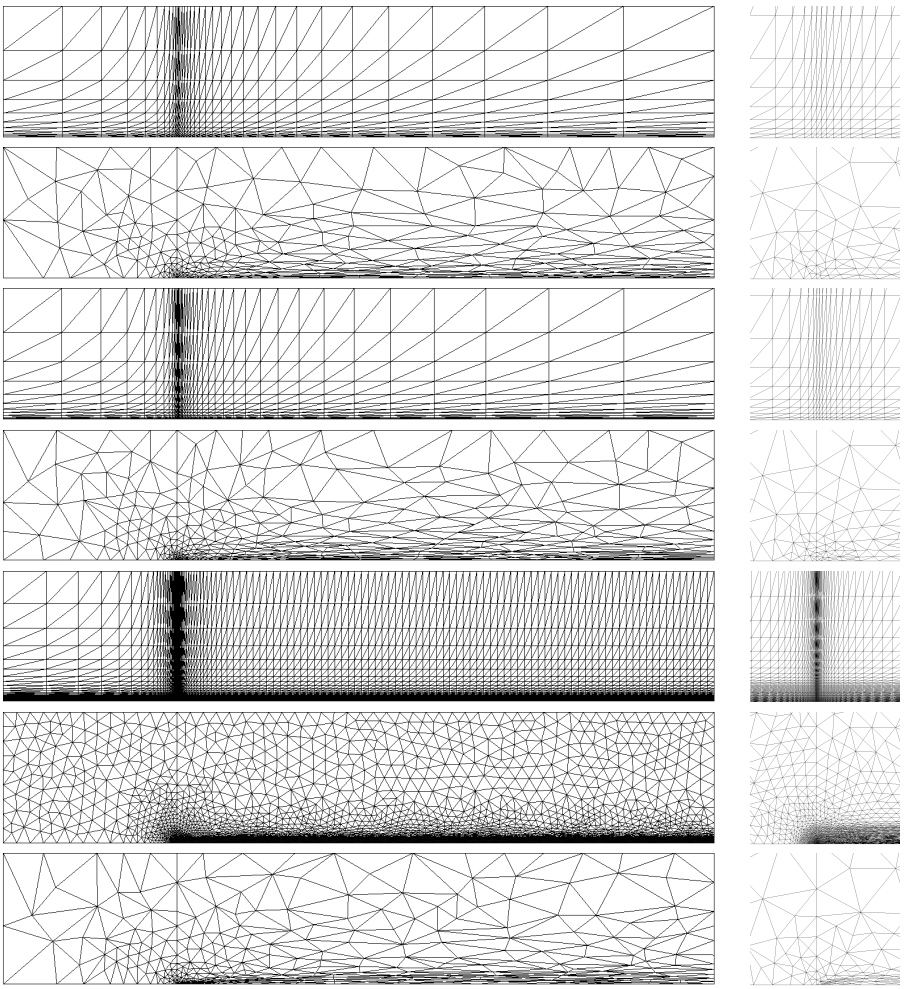


Figure 3.4: Structured and unstructured meshes evaluated for capturing the boundary layer on the plate test. The last mesh is refined in the region defined by (3.8).

Chapter 2, and an anisotropic mesh size field over the boundary layers which will be intersected with the other size fields prescribed over the domain.

The boundary mesh size field must have the following characteristics:

- a small and progressive size in the normal direction to the wall,
- large sizes in the tangent directions,
- the tangent size should be realistic for the geometry of the wall. It should be locally limited to a size which is proportional to the radius of curvature of the wall.

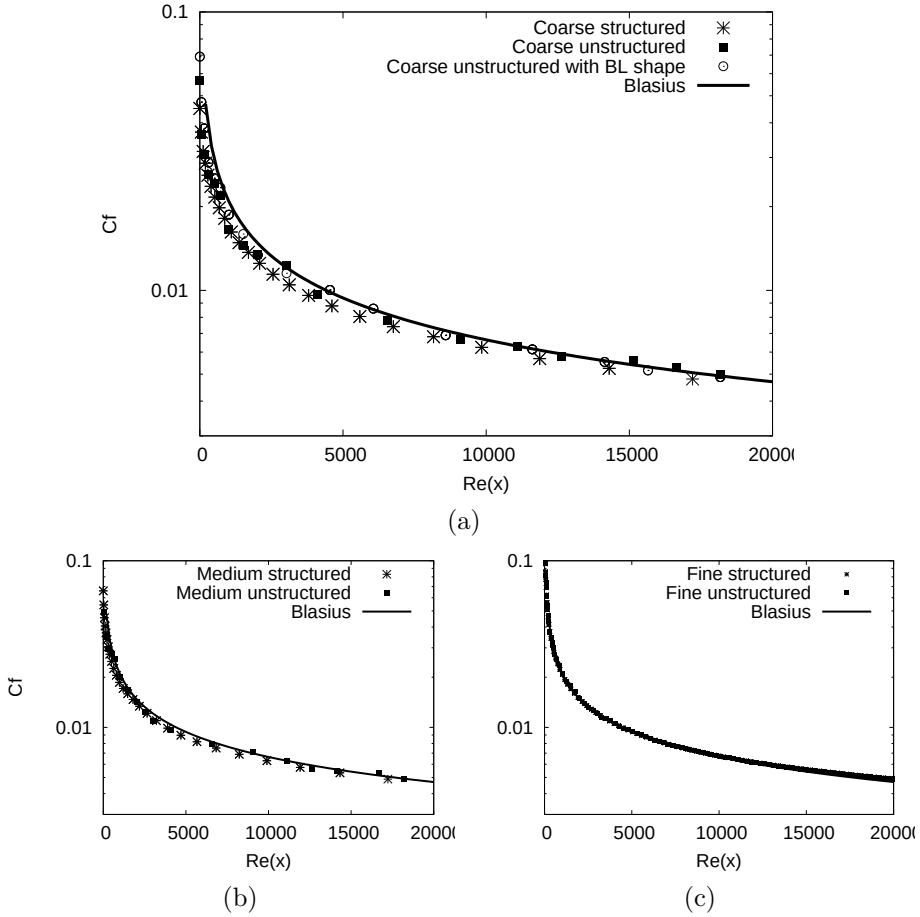


Figure 3.5: Skin friction coefficient C_f along the plate obtained with structured and unstructured meshes for (a) 5 elements in the BL, (b) 6 elements in the BL, and (c) over-refined meshes.

The last condition is needed to avoid ill-defined problems: if the local radius of curvature of the wall is small compared to the prescribed size, the size field is geometrically impossible to satisfy, even with a large interval $[L_{low}, L_{up}]$. In such a case, the behaviour of the adaptation procedure will be affected: in the optimal mesh, short edges will remain, as well as elements with a poor quality in the transformed space.

The technique presented here allows to build this size field almost automatically. The desired normal and tangent sizes in the boundary layer have to be prescribed, but the features of the geometry are automatically treated by the method.

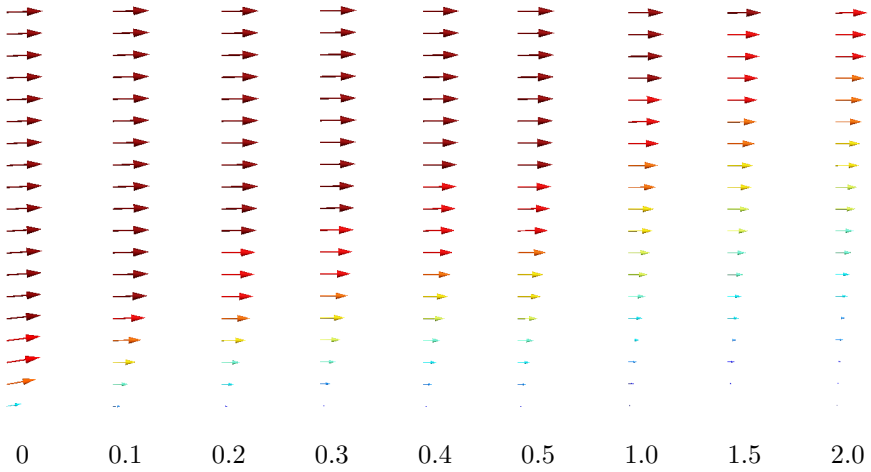


Figure 3.6: Velocity profiles at different x obtained with the mesh refined in the boundary layer region defined by the solution of Blasius.

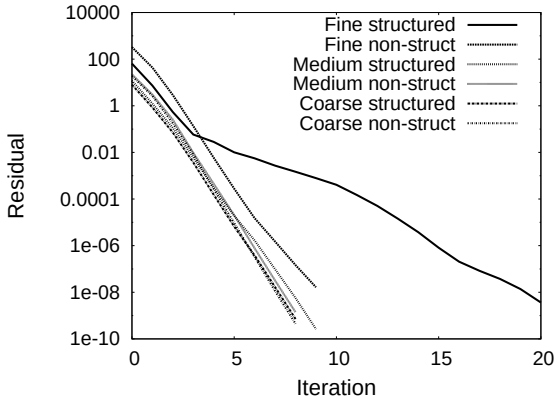


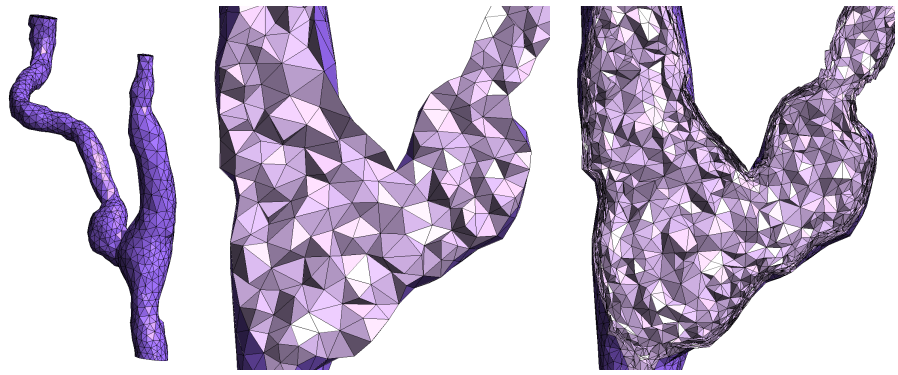
Figure 3.7: Plate test: convergence of the different computations.

3.2.1 Applications

Using the **mesh adaptation** procedure presented in Chapter 2 for adapting boundary layer meshes allows to merge the adaptation of the whole mesh, possibly with large deformations, and the particular adaptation of the boundary layer mesh by simply intersecting the size field of the boundary layer with the other size fields before running the adaptation procedure. For that reason, the techniques presented here can be seen as an extension of the proposed adaptation method to the CFD computations that capture boundary layers.

Note that since the initial mesh can be coarse and isotropic, a simple **mesh generation** method with boundary layer meshes can therefore be obtained

from the present work. It simply consists in producing a first mesh without any care of the mesh size, and then adapting it by the present method to get the desired mesh, as illustrated in Figure 3.8.



(a) (b) (c)

Figure 3.8: Mesh of the junction between an artery and a bypass (anastomosis): (a) mesh boundaries, (b) cut in initial mesh, (c) cut in mesh with boundary layer generated by the present method.

Finally, we mention the possibility to use the present method to control the discretization of the geometric entities by providing a mean to relate the curvature of the **geometry** to the surface mesh size. This application of the method is not further investigated here. The reader can refer to [72, 73] for related works.

3.2.2 Size field construction

As explained in Chapter 1, an anisotropic size field is given by a metric, which is a symmetric definite positive tensor. We denote by $\mathcal{L}(\mathbf{x})$ the metric related to the boundary layer mesh size field. As $\mathcal{L}(\mathbf{x})$ is symmetric and definite positive, it can be written as

$$\mathcal{L}(\mathbf{x}) = \mathcal{T}^t(\mathbf{x}) \begin{pmatrix} 1/h_1^2(\mathbf{x}) & 0 & 0 \\ 0 & 1/h_2^2(\mathbf{x}) & 0 \\ 0 & 0 & 1/h_3^2(\mathbf{x}) \end{pmatrix} \mathcal{T}(\mathbf{x}), \tag{3.9}$$

where $\mathcal{T}(\mathbf{x})$ is a tensor for which the matrix columns are the principal directions of the size field in $\mathbf{x}$, and $h_i(\mathbf{x})$ are the prescribed sizes in the corresponding directions.

Metric specification Since we are interested in distinguishing the sizes in the normal and tangent directions to the wall, we choose $\mathcal{T}(\mathbf{x})$ as

$$\mathcal{T}(\mathbf{x}) = \begin{pmatrix} \mathbf{n}(\mathbf{x}) & \mathbf{t_1}(\mathbf{x}) & \mathbf{t_2}(\mathbf{x}) \end{pmatrix}, \tag{3.10}$$

where $\mathbf{n}(\mathbf{x})$ is the normal direction to the wall and $\mathbf{t_1}(\mathbf{x})$, $\mathbf{t_2}(\mathbf{x})$ are the tangent directions. Note that these directions have to be defined in the boundary layer, which is a region of the mesh, not a surface. More exact definitions of $\mathbf{n}$, $\mathbf{t_1}$ and $\mathbf{t_2}$ are given in the remainder of this chapter.

Consecutively to the definition of $\mathcal{T}$, we set $h_1(\mathbf{x}) = h_n(\mathbf{x})$ with $h_n(\mathbf{x})$ the desired size in the normal direction, and $h_2(\mathbf{x}) = h_3(\mathbf{x}) = h_t(\mathbf{x})$, where $h_t(\mathbf{x})$ is the prescribed size in the tangent direction. Note that in this work, we restrict the developments to a single size for both tangent directions. A possible extension of the method with two different sizes is discussed in §3.5.

To sum up, the following directions and sizes have to be defined in order to compute the metric $\mathcal{L}(\mathbf{x})$:

- the normal direction $\mathbf{n}(\mathbf{x})$,
- the tangent directions $\mathbf{t_1}(\mathbf{x})$, $\mathbf{t_2}(\mathbf{x})$,
- the sizes $h_n(\mathbf{x})$ and $h_t(\mathbf{x})$.

Normal and tangent directions In order to obtain the main directions of the metric, the normal direction to the wall has to be defined in the region of interest. At least two methods can be used to get this normal in a point with coordinates $\mathbf{x}$. The first one is based on the distance $d(\mathbf{x})$ to the wall. Indeed, the normal can be computed as the gradient of the distance to the wall:

$$\mathbf{n}(\mathbf{x}) = \nabla d(\mathbf{x}).$$

Another way is to choose the unit vector with the direction of the straight line passing through $\mathbf{x}$ and the closest point to $\mathbf{x}$ on the wall. Both solutions require the computation of the distance to the wall, but the second is faster and yields more accurate results in the computation of the curvature, as explained here after.

The tangent vectors can be any vectors $\mathbf{t_1}(\mathbf{x})$ and $\mathbf{t_2}(\mathbf{x})$ such that the basis $(\mathbf{t_1}(\mathbf{x}), \mathbf{t_2}(\mathbf{x}), \mathbf{n}(\mathbf{x}))$ is orthogonal, since the sizes prescribed in the tangent directions are the same.

Normal size The size $h_n(\mathbf{x})$ has to be fixed according to the Reynolds number of the flow and the desired number of elements in the thickness of the boundary layer. A common way to build a structured boundary layer mesh is to start from a given size h^*_{n0} for the first layer of elements, and to build n successive layers using a progression factor p, which yields a size $h^*_{ni} = p^i h_{n0}$ for the i^{th} layer. The symbol * is used to mention the discrete nature of a value. The thickness t_{BL} of the boundary layer mesh is given by

$$t_{BL} = h^*_0 \frac{p^n - 1}{p - 1}. \tag{3.11}$$

The same progression is used here, but we specify it in a continuous rather than a discrete formulation, which simply corresponds to a linear dependence between the size h_n and the distance $d(\mathbf{x})$ to the wall:

$$h_n(\mathbf{x}) = h_0 + \gamma d(\mathbf{x}), \tag{3.12}$$

where h_0 is the desired size at the wall and γ is a constant, with $\gamma \geq 1$. The correspondence with the discrete formulation is obtained by the following relations:

$$h_0 = \frac{h_0^*}{p}, \quad \gamma = \frac{p-1}{p}. \tag{3.13}$$

Tangent size In the tangent directions, the prescribed size could be very large. In viscous flow computations for instance, the solution does usually not vary a lot along the tangent directions. That is the main reason for using anisotropic meshes for modeling boundary layers. However, as explained here above, the prescribed size has to be limited by the curvature of the neighboring wall, in order the end with a geometrically well defined problem.

The solution proposed is to relate directly the limitation of the prescribed size to a *region curvature* $\kappa(\mathbf{x})$, i.e. a curvature defined in the region of the boundary layer which is a smooth function and has the value of the surface curvature on the walls. The details about the computation of the curvature are given in §3.3.2. In our approach, we choose to limit the tangent size to a value which is proportional to the radius of curvature $r(\mathbf{x}) = \kappa^{-1}(\mathbf{x})$. If a prescribed size h_t^{pr} is given, we choose $h_t(\mathbf{x})$ by

$$h_t(\mathbf{x}) = min\left(h_t^{pr}, \alpha r(\mathbf{x})\right), \tag{3.14}$$

where α is a constant coefficient determining the required resolution level of the geometry to consider that the problem is geometrically well posed. From the numerical experiments a reasonable choice is $\alpha = 0.3$.

3.3 Distance and curvature computations

In the previous section, we described a mesh size field which depends on the computation of the normal to the walls and a region curvature. Both are related to the distance to the wall. We explain here how to define and compute this distance and the region curvature. The computation of the normal was already discussed in §3.2.2.

3.3.1 Distance

The distance of a point p with coordinates $\mathbf{x}$ to a wall can be computed in different ways depending on the available information:

- If a geometrical model is available[2], the shape of the wall is exactly known, and the exact distance to $\mathbf{x}$ can be given by the model.
- If the wall is only known by its discretization, a simple definition of the distance of p to the wall is the distance between p and the closest element of the discretization.

[2]More details are given about the geometrical models and their relation to the mesh in Chapter 4

- An equivalent function to the distance can be obtained by solving a partial differential equation (PDE) on the domain (see for instance [102]).

In this chapter, we assume that no geometrical model is given, which excludes the first possibility. An advantage of the second method is that it returns an accurate value of the physical distance to the walls, and the normal is trivially obtained. On the other hand, the gradient of the physical distance is not continuous, and the determination of the closest entity is mainly based on heuristics, as explained here below. The third possibility produces a smooth distance function with a continuous gradient but is less physical and requires to determine the value of some artificial parameters.

In the present work, we mainly investigate the use of the physical distance, although we plan to explore the potential of the third possibility in a near future.

Closest mesh entity In 3D, the simplicial entities discretizing a wall are triangles. The problem here is to find which triangle of the wall is the closest to p. The transposition to the 2D case is not described here since it does not raise any particular issue.

The algorithm proposed starts from an initial guess and advances in the neighboring elements until an element with a minimal distance is found:

```
Define the variable t with type 'triangle'
1. Find w, the closest point to p on the wall
2. t becomes the closest triangle around w (initial guess)
3. Mark all triangles around w as 'checked'
4. While a neighbor of t is not marked as 'checked' {
   4.1 t becomes the closest triangle between t and its neighbors
   4.2 Mark all triangles around t as 'checked'
}
```

Finding the closest vertex to p on the wall can be performed efficiently using kd-tree data structures [9]. The open source Approximate Nearest Neighbor (ANN) library [126] implements searching algorithms based on kd-tree data structures that are able to perform a search in $\mathcal{O}(log(n))$, where n is the number of vertices of the wall.

Note that the proposed algorithm is not perfectly robust as it is only able to find a local minimum and relies on the initial guess to converge to the right solution, although it performs well in most of the cases. Further works should investigate this issue and propose algorithms to solve the problem of the global optimum location.

Distance to a triangle In the previous algorithm, the distance d between the point p and a triangle t has to be computed. According to the relative position of p and t, the distance d can be either:

- the distance from p to a summit s_i of t,
- the distance from p to its projection on an edge e_i of t,

- the distance from p to its projection on t.

In order to determine in which of these cases we fall, p is projected on the plane T containing t. We denote p' the projection point of p in T. According to the location of p' relatively to t, the edges e_i and the summits s_i, we can determine the way to compute d. The regions of T corresponding to the different cases are depicted in Figure 3.9. The plane T is divided in 7 zones in which p' can fall.

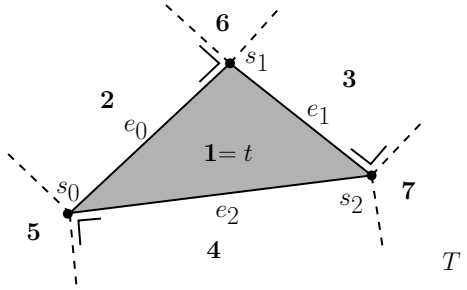


Figure 3.9: The different zones of the plane of a triangle on which the projection of a point can fall.

According to Figure 3.9, the distance between p and t is equal to the distance between p and

- p' if p' lies in zone 1,
- e_0, e_1 or e_2 if p' lies respectively in zone 2, 3 or 4,
- s_0, s_1 or s_2 if p' lies respectively in zone 5, 6 or 7.

Note that if p' is in zone 2, 3 or 4, the projection of p on the line containing the corresponding edge falls inside the edge. Once the zone has been determined, the distance d is then simply obtained by a computation of the distance between two points or a point and a line. In addition to the distance, the corresponding normal is directly obtained, since the closest point to p in the triangle is known.

3.3.2 Curvatures

In order to determine the tangent size inside the boundary layer, a region curvature $\kappa(\mathbf{x})$ has to be defined in the boundary layer zones. This curvature has to be smooth enough since the size field depends directly on it. Furthermore, its value on the walls should correspond to some definition of the surface curvature. A very common way to obtain the curvature of a surface is to choose the divergence of the normal:

$$\kappa(\mathbf{x}) = \nabla \cdot \mathbf{n}. \tag{3.15}$$

From a voluming point-of-view, since the normal to a wall is the gradient of the distance, we choose to define the curvature $\kappa(\mathbf{x})$ by

$$\kappa(\mathbf{x}) = \Delta d(\mathbf{x}) = \nabla \cdot (\nabla d(\mathbf{x}))\,. \tag{3.16}$$

In order to simplify the notations, the dependence in $\mathbf{x}$ is omitted in the remainder of the section. Starting from the distance computed as explained here above, we can compute its Laplacian by a finite element approach. By multiplying (3.16) by test functions $\hat{\phi}_j$, integrating over a mesh element Ω, and integrating by parts, we obtain

$$\int_\Omega \kappa\, \hat{\phi}_j \, d\Omega = -\int_\Omega \nabla\hat{\phi}_j \cdot \nabla d \, d\Omega + \int_{\partial\Omega} \hat{\phi}_j \, . \, (\mathbf{n} \cdot \nabla d) \, d\partial\Omega. \tag{3.17}$$

Discretizing κ by $\kappa = \sum_i \kappa_i \phi_i$ yields

$$\sum_i \int_\Omega \kappa_i \phi_i \hat{\phi}_j \, d\Omega = -\int_\Omega \nabla\hat{\phi}_j \cdot \nabla d \, d\Omega + \int_{\partial\Omega} \hat{\phi}_j \, . \, (\mathbf{n} \cdot \nabla d) \, d\partial\Omega. \tag{3.18}$$

We choose the shapes functions ϕ_i as the classical linear shape functions.

In order to save computational time, the mass matrix $\sum_i \int_\Omega \kappa_i \phi_i \hat{\phi}_j \, d\Omega$ on the left-hand side is lumped in the computation.

The evaluation of the right-hand side of (3.18) requires the computation of the gradient of d. The evaluation of ∇d can be done in at least two ways. A first solution consists in setting $\nabla d = \sum_i D_i \nabla\phi_i$, which yields a piecewise constant approximation of ∇d. This choice is the most natural if the gradient at nodes is not known a priori, and a piecewise linear approximation of the distance is available. If the normals to the walls are available at nodes, a piecewise linear approximation of the distance gradient can be used: $\nabla d = \sum_i (\nabla d)_i \phi_i$. This is the case if the second option is chosen for the distance computation (see §3.2.2), i.e. d is the physical distance to the closest point of the wall.

Figure 3.10 shows the curvatures obtained with both approaches in a region bounded by two concentric spheres. From the difference between (d) and the other figures, we observe that the smoothness and accuracy of the curvature is highly dependent of the computation of the gradient and that the piecewise linear approximation of the gradient (c) yields better results than the constant one (a), even when the gradient is computed from the analytical distance (b). Note that in the piecewise linear gradient approach, the computation of the gradient is directly related to the computation of the distance. The computation is also efficient since the distance and its gradient are computed at the same time by the technique described in §3.3.1.

We notice that the resulting curvature is quite noisy. A smoothing is therefore applied before using it in the limitation of the tangent size.

3.4 Examples

In this section some examples of the application of the proposed method are presented.

Planar surface The first test consists in meshing a cube with a desired boundary layer mesh at the vicinity of a face. The anisotropic ratio at the wall is about 1200. Figure 3.11 shows the resulting mesh. It has been checked that the curvature obtained is 0 everywhere.

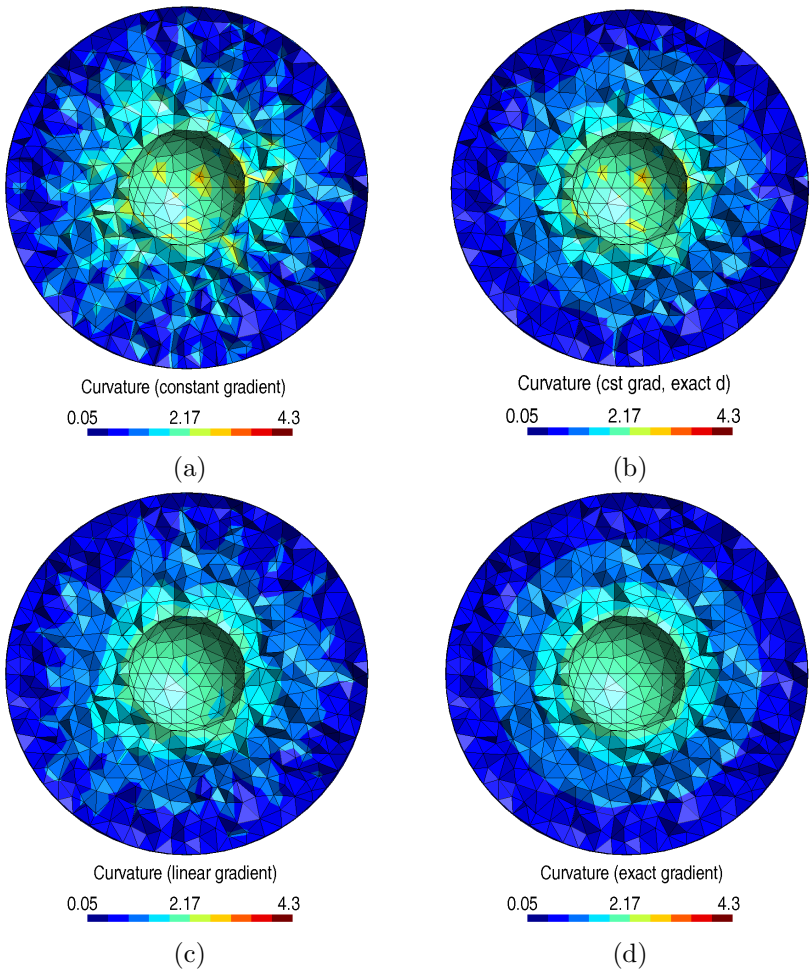


Figure 3.10: Evaluation of the region curvature κ between two concentric spheres of radii 1 and 3 with (a) the piecewise constant gradient approach, (b) the same approach with the analytical distance, (c) the precomputed piecewise linear gradient, and (d) the analytical gradient.

Corners and ridges We consider here geometries in which particular difficulties would arise if semi-structured meshes had to be built. For those cases, no physics computation is made and the geometries are just intended to test the mesh adaptation techniques developed in this chapter. The first test is a parallelepiped from which a cube of side length 1 has been removed, leading to corners and ridges at 90°. In order to generate a simple mesh adaptation test (no physics), the surfaces of this cube are the only surfaces along which a boundary layer mesh is generated. Figure 3.12 shows the meshes obtained with an anisotropic ratio of 40 at the walls for $\alpha = \infty$ (no limitation based on the curvature), $\alpha = 0.3$, and $\alpha = 0.3$ with a smoothing of the curvature field. The

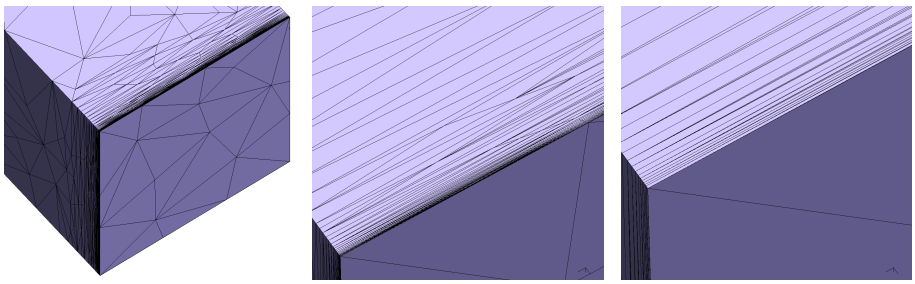


Figure 3.11: Cube test: resulting mesh with an anisotropic ratio 1200 at the wall.

edge length is limited to 0.03 in order to limit the refinement on the leading edges, which corresponds to a limitation of the curvature to 10. Figure 3.13 depicts the computed curvature before and after the smoothing. We observe that the mesh obtained with a control of the size field by the curvature has the desired edge length at the corners and the ridges, while the elements have both inappropriate shapes and sizes without taking the curvature into account.

We also point out the high difference in size between the elements located on the ridges and the other elements if no smoothing is applied to the curvature field.

In the second test, another region is enclosed in surfaces that form corners and ridges and a particular point is located at the intersection between a corner and a ridge. Figure 3.14 depicts the resulting mesh. The coarsely meshed surfaces are those along which a boundary layer size field was prescribed. The anisotropic ratio at the wall is also 40 for this test, α is set to 0.3 and the curvature field is not smoothed. Figure 3.15 shows successive cuts in the mesh perpendicularly to the sloping surface. The gray scale indicates the computed curvature.

Anastomosis In the field of biomedical engineering, an important domain of application of the fluid mechanics is the study of blood flows. In particular, an objective of the numerical computations is to allow to predict accurately the effects of a surgical procedure like a bypass operation. The current example deals with an anastomosis, which is the junction between a sane artery and an artificial vessel implanted to bypass an obstructed artery.

In order to enable realistic simulations of the blood flow of a particular patient, imaging techniques have been designed to scan the arteries and produce realistic representations of it. The resulting three-dimensional images can then be converted in surface triangulations, and the internal region can be meshed by classical 3D mesh generation techniques. As an example, a mesh of an anostomosis located below the knee, at the intersection between the femoral and popliteal arteries is shown in Figure 3.8 (a) and (b).

Starting from this mesh, a mesh with an anisotropic ratio of 10 at the walls is produced by the present technique and depicted in Figure 3.16 (a). The

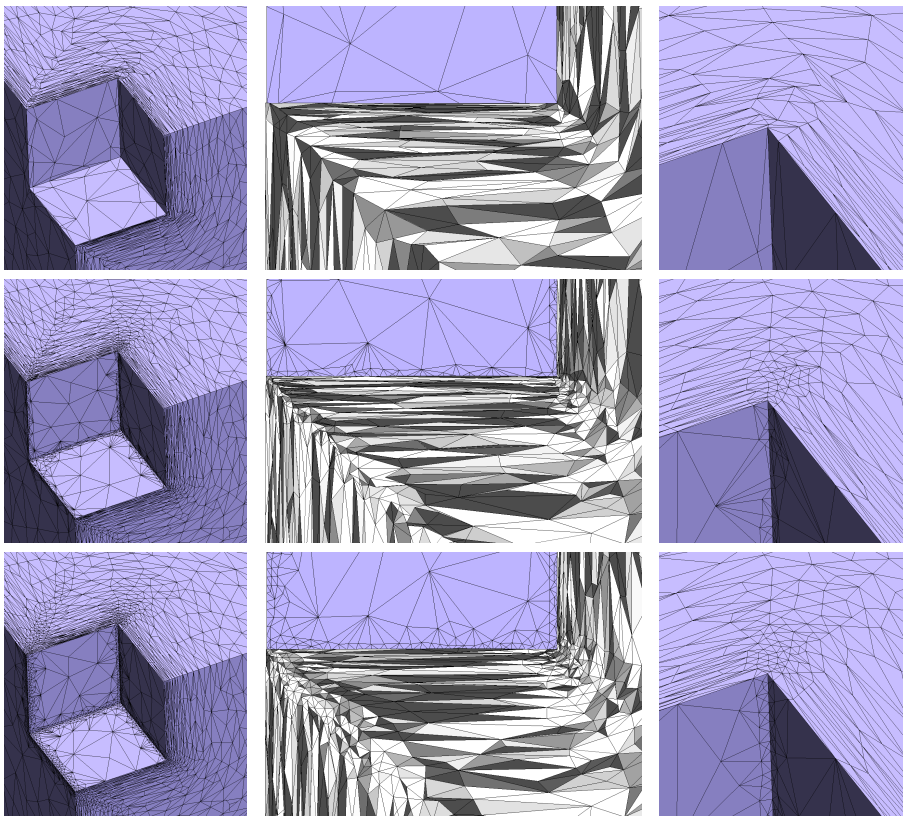
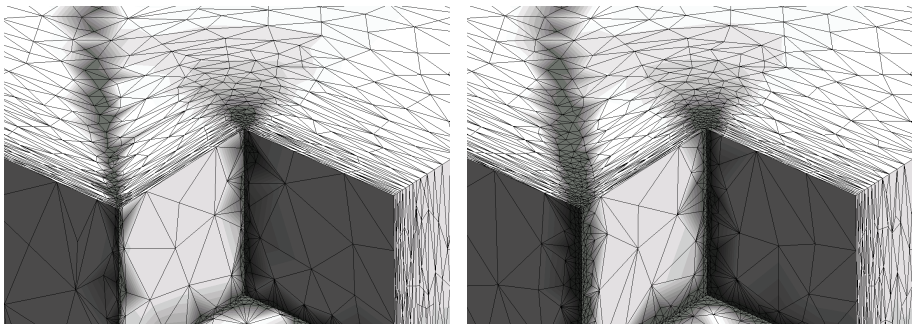


Figure 3.12: Corner test: no limitation based on curvature (top), limitation with $\alpha = 0.3$ (middle), and limitation with $\alpha = 0.3$ and a smoothed curvature (bottom). The second view is a cut in the mesh.

value $\alpha = 0.5$ is chosen and no smoothing of the curvature field is applied. In Figure 3.16 (b) and (c), a close view of the mesh near a slightly curved wall is depicted if the tangent size was bounded (c) or not (b) by $\alpha r(\mathbf{x})$. We observe that the mesh alignment is much better with a curvature limitation. The mesh with a constant tangent size prescribed in the boundary region is visibly more disorganized since an ill-defined size field is prescribed.

Figure 3.17 shows the skin friction (a), pressure (b) and velocity (c) of a steady flow with fluid parameters relatively close to blood: density $\rho = 1.06\,10^3\,kg/m^3$ and dynamic viscosity $\mu = 4.5\,kg/ms$. No-slip boundary conditions are applied at the walls and a parabolic velocity is imposed at the inlet with a global inflow of $8\,10^{-3}l/s$. The diameter of the artery is approximately $10\,mm$ at the smallest sections. With those values, the Reynolds number of the flow is around 200. A constant pressure is imposed at the outlet.

When the initial mesh is a coarse triangulation, the mesh adaptation procedure tends to refine it. Unfortunately, since the initial triangulation is the only



(a) (b)

Figure 3.13: Corner test: curvature computed with (3.16) (a) before and (b) after smoothing.

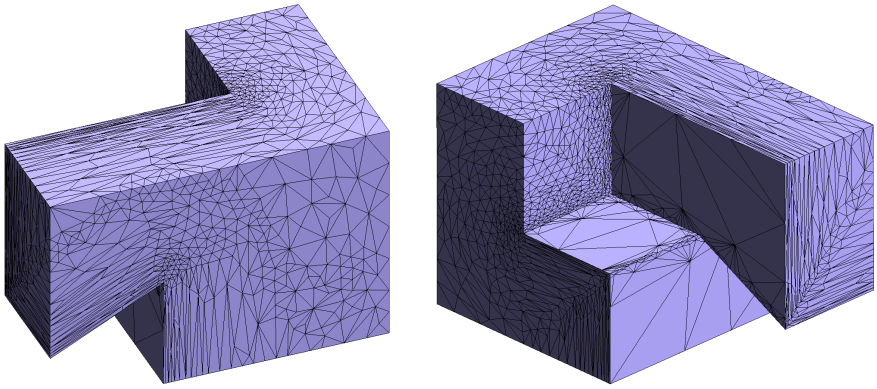


Figure 3.14: Corners and ridges: resulting mesh. The coarsely meshed surfaces are those along which a boundary layer size field was prescribed. The anisotropic ratio at the walls is 40.

geometrical data available, the new points are simply added along the split edge and no vertex snapping is performed. This impacts on the computation of the curvature. Figure 3.18 (a) shows the curvature field obtained on the adapted mesh if no smoothing is applied. Since only the local features are taken into account, the curvature is high between two original triangles, and equal to zero inside it. Figure 3.18 (b) shows the curvature field that can be obtained if a simple smoothing is applied. This last solution is much more interesting to use. Note that we proposed an alternative solution to that particular issue in [150]. It consists in building a cubic representation of the linear triangles and snap the new vertices on it.

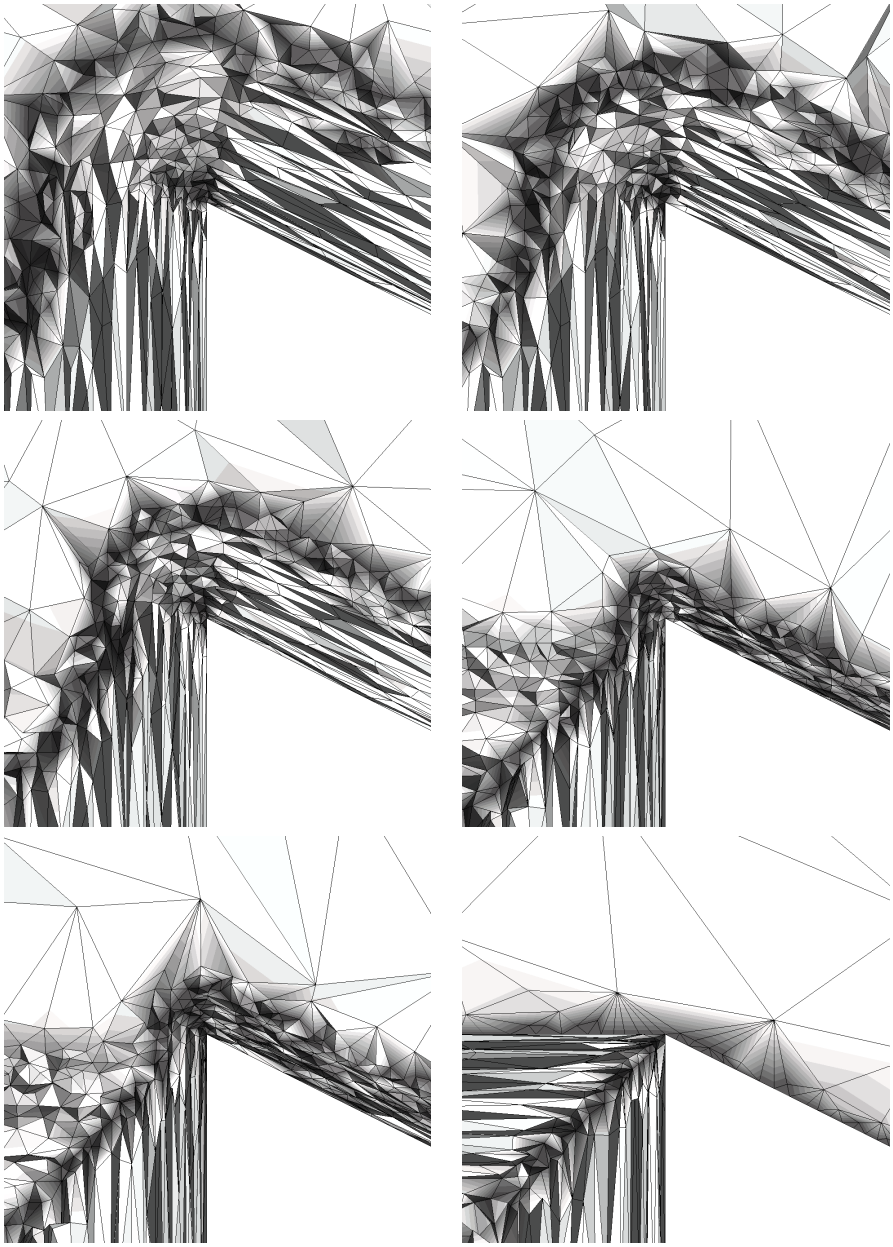
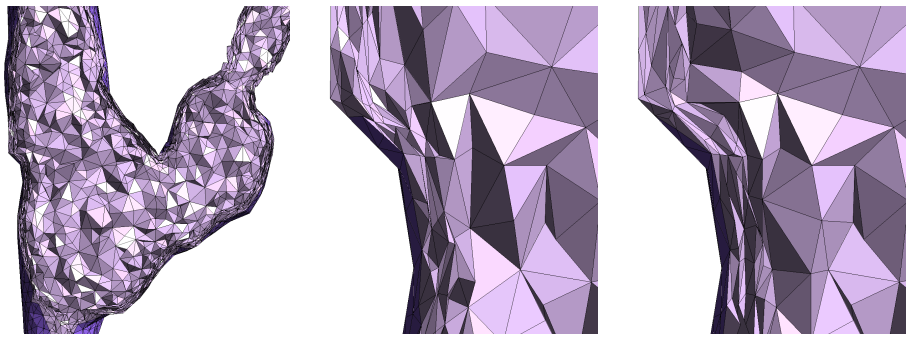


Figure 3.15: Corners and ridges: successive cuts in the mesh. The gray scale indicates the computed curvature.

3.5 Next steps

In a near future, several trails will be explored to improve the present method.



(a) (b) (c)

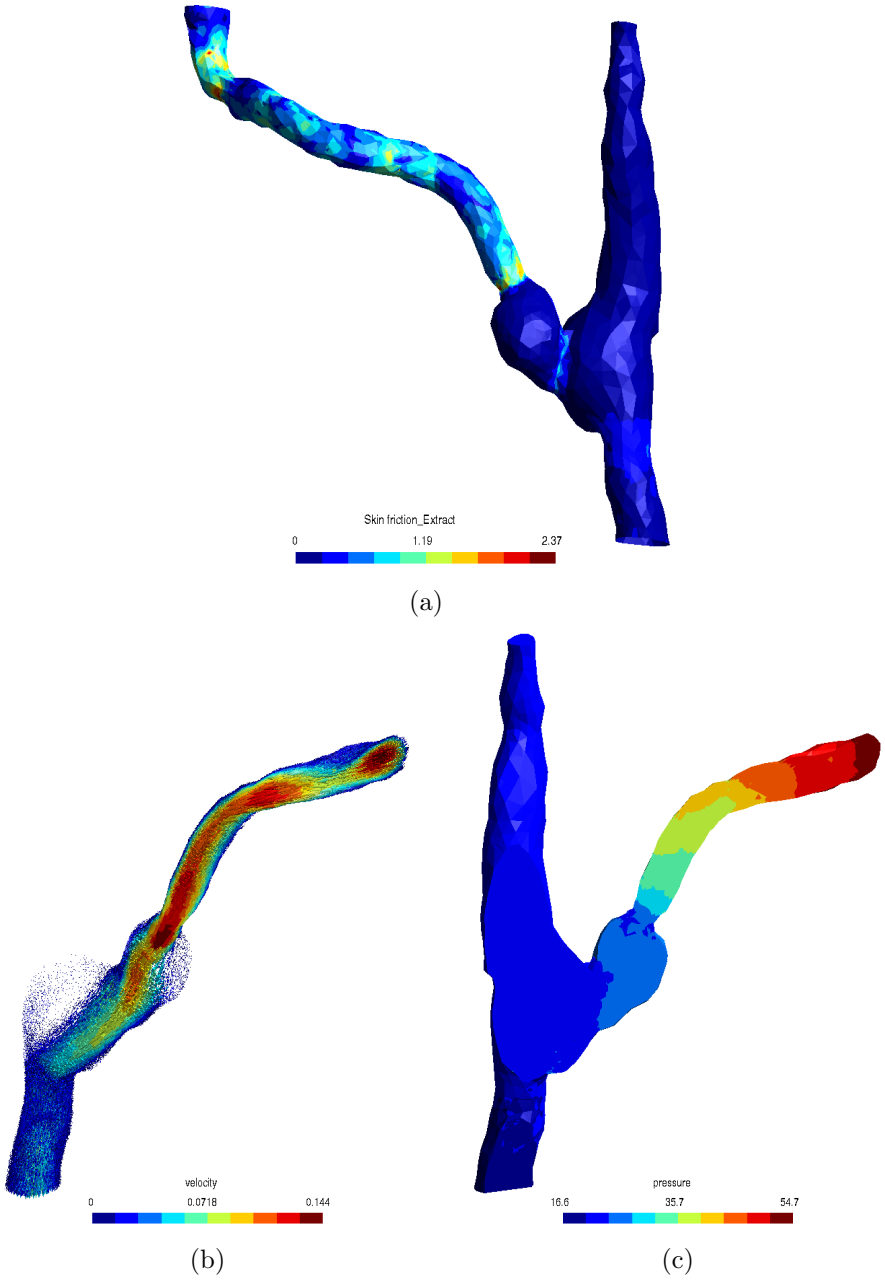
Figure 3.16: Anastomosis: comparison between resulting meshes with (b) a simple anisotropic size field, and (c) an anisotropic size field with a tangent size bounded by a function of the curvature.

First, we aim at investigating the generation of an artificial distance function by the resolution of an appropriate EDP, like proposed in the work of S. Legrand et al. [102]. Such a function could have interesting properties, like continuous derivatives. If the iso-lines of the distance function do not intersect each other, a method to generate structured meshes could been built from this function by adding vertices along these iso-lines, with a spacing dependent on the curvature. This method would combine the advantage of being independent from the complexity of the geometry with the advantages of the other structured meshes (suitable for higher Reynolds number flows, no noise in the solution).

Furthermore, such a function could act as a filter for the high frequencies in the geometrical input, with a cutoff frequency depending on the parameters of the EDP. This is particularly interesting in the case of STL triangulations as it was shown in the results presented above.

About the computation of the curvature, we have seen that using a piecewise constant approximation of the distance gradient yields very poor results. If the distance function is given by an EDP, a piecewise linear gradient could be obtained if quadratic shape functions were used to discretize the distance, but the computational cost would become prohibitive. Another possibility suggested by E. Marchandise [117] is to use a linear or quadratic reconstruction of the gradient by a least-square method. The results shown in [117] are promising.

Another important improvement to bring to the method is the computation of the principal curvatures and the corresponding directions. In this work, a unique curvature equal to the divergence of the normal is computed, which is not optimal when the curvatures in the principal directions are very different. In the case of a cylinder for instance, we wish to have an edge length proportional to the radius in the direction θ, and long edges in the direction z.



(a)

(b)

(c)

Figure 3.17: Anastomosis: steady flow computation at $Re = 200$ with the mesh presented in Figure 3.16 (a): (a) skin friction, (b) cut in the velocity field, and (c) cut in the pressure field.

The metric $\mathcal{L}(\mathbf{x})$ will therefore be written as

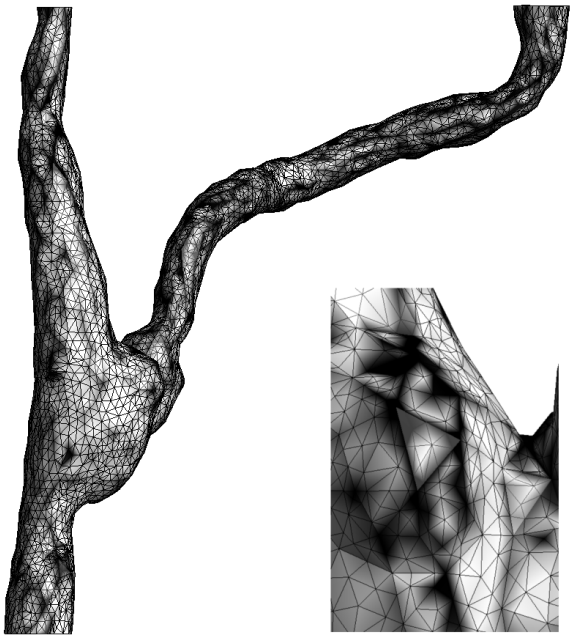
$$\mathcal{L}(\mathbf{x}) = \mathcal{T}^t(\mathbf{x}) \begin{pmatrix} 1/{h_n}^2(\mathbf{x}) & 0 & 0 \\ 0 & {\kappa_1}^2(\mathbf{x})/\alpha^2 & 0 \\ 0 & 0 & {\kappa_2}^2(\mathbf{x})/\alpha^2 \end{pmatrix} \mathcal{T}(\mathbf{x}), \tag{3.19}$$

with

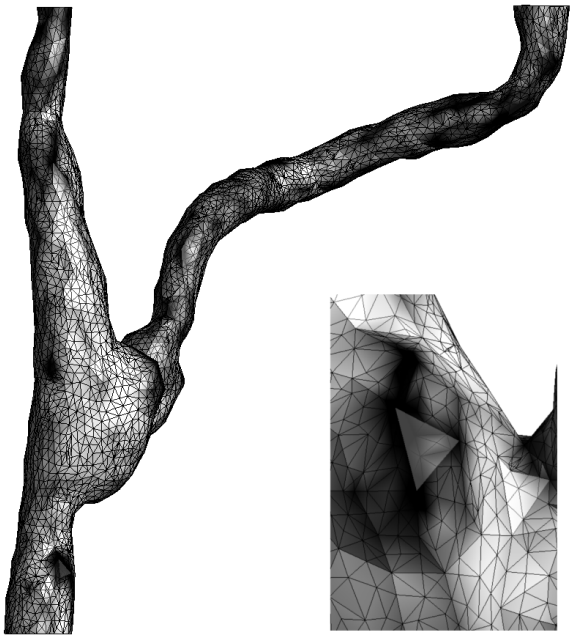
$$\mathcal{T}(\mathbf{x}) = \begin{pmatrix} \mathbf{n}(\mathbf{x}) & \mathbf{t_1}(\mathbf{x}) & \mathbf{t_2}(\mathbf{x}) \end{pmatrix}, \tag{3.20}$$

where $\mathbf{t_1}(\mathbf{x})$ and $\mathbf{t_2}(\mathbf{x})$ are the directions of the main curvatures $\kappa_1(\mathbf{x})$ and $\kappa_2(\mathbf{x})$. Those curvatures and their directions will be given by the eigenvalues and eigenvectors of the tensor $\mathcal{K}(\mathbf{x})$ defined by

$$\mathcal{K}(\mathbf{x}) = \nabla\nabla d(\mathbf{x}).$$



(a)



(b)

Figure 3.18: Anastomosis: computed curvature (a) without and (b) with a smoothing.

Chapter 4

Handling of geometrical models

When dealing with industrial applications in solid or fluid mechanics, a Computer Aided Design (CAD) model is often provided to describe the exact geometry of the mechanical components to be analyzed. For instance, one could wish to analyze the stresses in a manufactured piece under some load, or to study the properties of the flow around an external component of an aircraft like a landing gear or a wing tip, ...

In order to take advantage of mesh adaptation methods in such computations, a particular issue is raised: how do we control the compliance of an adaptive mesh to the CAD model provided with the initial mesh ? A simple solution is to fix the mesh near the model boundaries. However, this solution is quite restrictive:

- The mesh size cannot be controlled on the boundaries, which results in a lower efficiency of the physics computation in case of a goal-oriented mesh adaptation like with error minimization methods. Furthermore, particular phenomenons happening at the vicinity of the boundaries may require to be captured, like a boundary layer separation or a traveling shock wave.

- The accuracy of the finite element method is known to be sensitive to the gradation of the mesh size. In mesh adaptation, the common way to control this gradation is to limit the gradient of the size field. However, if the elements located near the model boundaries are fixed, the edge length at the vicinity of the boundary is not linked anymore to the edge length given by the size field.

The other solution is to allow the mesh adaptation procedure to modify the boundary elements. In that case, several requirements are to be satisfied by the local mesh modification operators: the *classification* of the new mesh entities, i.e. their link to the entities of the CAD model, has to be set properly, and the topology of the CAD model has to be preserved.

Secondly, when new boundary nodes are created, they have to be relocated (*snapped*) at their right location on the corresponding geometrical entity. Although this relocation is simple in most of the cases, it can sometimes generate ill-shaped, or even tangled elements. Up to now, this issue has not been given a lot of interest in the literature, which results in a lack of robust methods.

The issues brought up here are addressed in the Article III, *Mesh adaptivity complying to a geometrical model*, Compère, Remacle (not submitted yet) laid out here below. The paper describes the key components of a robust mesh adaptation method complying to a CAD model:

Relation between mesh and geometrical model A CAD model representation is described, and the relation of the mesh to the model is studied. In particular, the notion of *classification* is recalled, and a definition of *compatibility* of the mesh to the geometrical model is provided, setting the framework of the other developments.

Vertex snapping By combining two techniques described in Chapter 2, the node relocation technique and the optimization by edge and face swaps, an algorithm allowing to snap the new boundary nodes in a robust way is proposed. This method is similar to the one used for handling arbitrarily large mesh deformations in the paper entitled *A mesh adaptation framework for dealing with large deforming meshes* (Compère, Remacle, Jansson, Hoffman) (Article IV) presented in Chapter 5.

Mesh modification operators The mesh modification operators described in Chapter 2 are analyzed in the general case of a mesh linked to a CAD model. Detailed conditions on the applicability of the modifications in the different configurations are given.

Article III
Mesh adaptivity complying to a geometrical model

Gaëtan Compère[1,3*], Jean-François Remacle[1,2]

[1] *Université catholique de Louvain, Department of Civil Engineering, Place du Levant 1, 1348 Louvain-la-Neuve, Belgium*

[2] *Center for Systems Engineering and Applied Mechanics (CESAME), Université catholique de Louvain, 1348 Louvain-la-Neuve, Belgium*

[3] *Fonds National de la Recherche Scientifique, rue d'Egmont 5, 1000 Bruxelles, Belgium*

Abstract

In this paper, a method is proposed to enhance the mesh adaptation techniques based on local modifications. To aim of the method is to make the modifications conform to the CAD model (possibly non-manifold) associated to the initial mesh in a robust way. In particular, a new vertex snapping technique is proposed. It uses an elastic analogy to relocate the closest vertices together with edge and face swaps to control the minimal quality of the modified elements. Also, the compatibility between the mesh and the geometrical model is analyzed with more details than in the previous literature. The subsequent constraints on the mesh modifications are given. The resulting operators and the snapping method are tested on meshes of complex geometries in order to show their robustness.

Key words: Mesh, Adaptation, snapping, geometry, CAD, model

III.1 Introduction

In the last decades, the increasing complexity of the computations and the limited growth of the computational resources lead the researchers to produce adaptive methods in order to concentrate the efforts on particular regions of the domain. Among the adaptive methods that arisen, the *h-adaptive* methods consist in modifying the mesh during the simulation. The first class of these methods are based on complete re-meshings: a new mesh is built from scratch when it is considered as necessary [136, 75, 4]. Another type of h-adaptivity consists in modifying the mesh locally [94, 32, 36, 55, 50, 106, 48]. The latter methods have the advantages of requiring only local mesh-to-mesh projections and being easily parallelizable.

When dealing with domains defined by CAD (Computer Aided Design) models, the simulation performed on the resulting meshes can be highly dependent on the compliance of the mesh to the geometry. Efforts have been

made in the past for meshing complex 2D geometries (see for instance [43, 73]) and building tetrahedral meshes on it [65, 173, 77]. A set of open source tools is already available [172, 166, 78] to do this job. However, only a few works have been performed for handling a CAD model in the 3D mesh adaptation procedures based on local mesh modifications. The necessity to snap the new boundary vertices on their exact location on the geometry has been brought out in [171] and [96], and a procedure to snap those vertices in a relatively robust way is presented in [104, 105]. This procedure tries to move the vertices one by one and applies local mesh modifications if the motion yields an invalid mesh (step 1). In some cases, the procedure fails and a local re-meshing is performed (step 2).

In this paper, we present a relatively simple technique intended to replace the first step of the previous method in order to improve its robustness. The proposed technique combines two existing tools: a node repositioning procedure based on an elastic analogy [186, 13] for repositioning the nodes around the snapped vertices, and local mesh modifications like edge and face swaps to improve locally the mesh if necessary. The new method allows to snap the new vertices in a robust way with a limited number of mesh modifications. Complex 3D test cases are presented to show its robustness.

In previous works [171, 96, 170], other issues related to the topological compatibility between the mesh and the geometrical model were raised. However, few details are given about the particular checks that have to be performed on the local mesh modifications. In this paper, we go further in the exact definition of local mesh modifications preserving the compatibility of the mesh to the geometrical model.

In order to allow other authors to build on the scientific repeatability of the current research, we publish the implementation of the method in the open source library MAdLib [46].

The remaining of this paper is organized as follows. Section III.2 recalls the class of adaptation procedures on which the current method is built. The geometrical model and its relation to the mesh are described in section III.3. In section III.4, we reach the issues arising from the local mesh modifications on boundaries while the snapping procedure is presented in section III.5. Finally, numerical examples are shown in section III.6.

III.2 Mesh adaptation procedure

The mesh adaptation procedure on which we build the proposed method is the adaptation by local modifications [94, 32, 36, 55, 50, 106, 48]. Starting from an initial mesh, the procedure modifies it in order to reach two goals: (i) satisfying a mesh size field, i.e. a prescribed edge length in every part of the mesh, and (ii) maintaining or improving the quality of the elements.

There are many reasons why adapting the mesh can be required. We mention the followings among others:

- handling the mesh-related issues with moving boundaries [59, 48],
- complying to a maximal edge length based on an error estimator [12, 4, 103, 152, 149],

- controlling the size of the discrete problem [149],
- capturing shocks or other discontinuities in a medium [103, 3, 180, 44].

In this section, we recall the basis of the mesh adaptation procedure by local mesh modifications, and we detail the different mesh modifications involved in the method. These modifications will then be explored from the point-of-view of the geometrical model in §III.4.

Mesh size field. The usual way to control the mesh size is to use a mesh size field (see for instance [136, 106]). An isotropic mesh size field is a scalar function $\delta(\mathbf{x}, t)$ that defines the optimal length of an edge at a position $\mathbf{x}$ of the domain and at a time t. We typically define the non-dimensional length L_e^{tr} of edge e as

$$L_e^{tr}(t) = \int_e \delta^{-1}(\mathbf{x}, t) dl. \tag{III.1}$$

One of the goals of the mesh adaptation is to obtain a mesh which is as close as possible to the unit mesh, i.e. a mesh in which all edges have an non-dimensional length $L_e^{tr} = 1$. As the unit mesh cannot be exactly reached in practice, an interval $[L_{low}, L_{up}]$ has to be defined. An edge with a length $L_e^{tr} < L_{low}$ is considered as short, while a long edge is an edge for which $L_e^{tr} > L_{up}$.

Local mesh modifications. A local mesh modification is an operator that removes a cavity $\mathcal{C}$ of a mesh $\mathcal{M}$, i.e. a connected set of elements of $\mathcal{M}$, and replaces it by another cavity $\mathcal{C}'$ with the same boundary. Formally, we write

$$\mathcal{M}' = \mathcal{M} - \mathcal{C} + \mathcal{C}'.$$

In this paper, we adopt a very common approach which consists in selecting a finite number of local mesh modifications [70, 55, 106, 3, 44, 59], and applying them successively in a well defined order [106, 44].

In particular, we mention the edge split (see Figure III.1(a)) and edge collapse (see Figure III.1(b)) modifications which are designed to control the local mesh size: the edge split is used to split long edges, and the edge collapse aims at eliminating short edges.

Other modifications are defined to improve the quality of the elements:

- The edge swap [70] (see Figure III.1(c)) re-meshes the cavity surrounding an edge.
- The face swap [70] (see Figure III.1(d)) re-meshes the cavity surrounding a face.

Additional mesh modifications can be created by combining the previous ones. These modifications usually aim at removing the sliver elements, i.e. elements with a very poor quality but no long or short edge. The two following modifications are usually defined:

- The face collapse, which combines an edge split and an edge collapse [104] (see Figure III.1(e)): an edge of the face is split and the resulting edge lying inside the face is collapsed.
- The region collapse, which combines two edge splits and an edge collapse [104] (see Figure III.1(f)): two opposite edges of the region are split, and the new edge inside the region is collapsed.

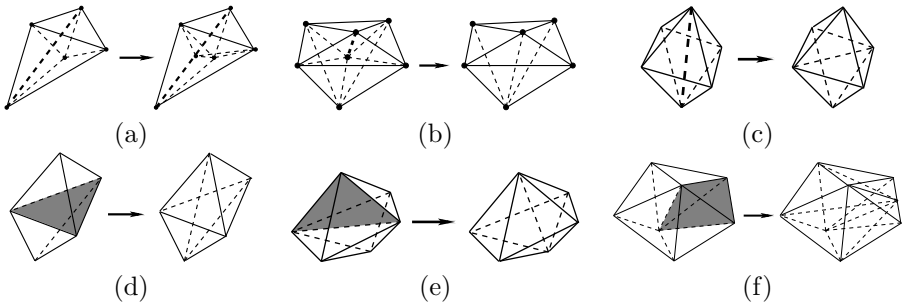


Figure III.1: The set of mesh modifications: (a) edge split, (b) edge collapse, (c) edge swap, (d) face swap, (e) face collapse, (f) region collapse.

Finally, isolated node relocations can be joined to the set of available modifications.

Mesh adaptation procedure. As an example, the following procedure [48] uses the local mesh modifications described here to satisfy the two goals mentioned here above.

```
Do {
1. Eliminate short edges ( L_e^tr  <  L_low ) with edge collapses
2. For every edge e:
  2.1 Evaluate the minimal element quality Q_e,min around e
  2.2 If Q_e,min is bigger than a tolerance value Q_swap,
      go to next edge
  2.3 Swap the edge if it increases Q_e,min
3. Apply the sliver elimination procedure
4. Eliminate long edges ( L_e^tr  >  L_up ) with edge splits
} While ( the mesh is modified )
```

In this procedure, $Q_{e,min}$ is an element quality above which the edge swap operation is not attempted. See [134, 109] for examples of element shape measures. Note that the interval $[L_{low}, L_{up}]$ has to be chosen sufficiently large so that no infinite loop occurs between split and collapse modifications. In this work, we choose the interval $[1/\sqrt{3}, \sqrt{3}]$.

The elimination of the sliver elements is performed by trying to apply different mesh modifications until one of them is successful. Refer to [48] or [106] for complete descriptions of sliver elimination procedures.

III.3 Geometrical model

A geometrical model is a collection of topological entities connected together. The connections can be for instance a surface bounding a region, or a line being bounded by two points. The topological entities found in any geometrical model can be classified according to their dimension. In this work, we consider the following groups of geometrical entities:

- the model vertices G_i^0, which have the dimension 0,
- the model edges G_i^1, which have the dimension 1,
- the model faces G_i^2, which have the dimension 2,
- the model regions G_i^3, which have the dimension 3.

The connections between the topological entities can then be stored in different ways. In this work, the Boundary Representation (BRep) of the geometrical model is used: a volume is bounded by a set of surfaces, a surface is bounded by a set of lines, and a line is bounded by points. This representation is implemented in the geometrical model of Gmsh [77, 78]. In Gmsh, a model entity contains the list of the entities G_j^{d+1} that it bounds and the entities G_k^{d-1} that it *uses*, i.e. the entities bounded by it. The connectivity of the geometrical model can be summarized as

$$G^0 \rightleftharpoons G^1 \rightleftharpoons G^2 \rightleftharpoons G^3, \tag{III.2}$$

Note that we do not restrict the present work to manifold geometries. For instance, the techniques described here are able to deal with model faces used by two model regions, or model edges used by several model faces. Rigorous definitions of manifold and non-manifold geometries can be found in [116].

In addition to its connectivity, every model entity has a representation in the physical space, a geometry. The geometry of a point is its position $\mathbf{x}$. The geometry of a line L_i is defined by its parametrization $\mathbf{p}(t) \in \text{Ł}_i$, $t \in [t_1, t_2]$ while the geometry of a surface S_i is defined by its parametrization $\mathbf{p}(u, v) \in S_i$. The geometry of a region is simply $\mathcal{R}^3$.

Relation mesh - geometry In order to be valid, the mesh has to be compatible with the geometrical model. By compatible with the geometrical model, we mean that the mesh must be a discretization of the geometrical model, thus complying perfectly to its geometry and topology.

In the remainder of this paper, we denote M^d the set of mesh entities M_i^d with dimension d. From [167], we recall the definition of the classification of a mesh entity on a geometrical entity:

DEFINITION. *Classification* [167, 171]. The unique association of a topological mesh entity of dimension d_i, $M_i^{d_i}$, to a topological model entity of dimension d_j, $G_j^{d_j}$, where $d_i \leq d_j$, is termed classification and is denoted

$$M_i^{d_i} \sqsubset G_j^{d_j}, \tag{III.3}$$

where the classification symbol, $\sqsubset$, indicates that the left-hand entity or set is classified on the right-hand entity.

In the remainder of the paper, we say that a mesh entity $M_i^{d_i}$ is *using* a mesh entity $M_j^{d_i-1}$ if $M_j^{d_i-1}$ is in the closure of $M_i^{d_i}$.

In a boundary representation of the geometrical topology, the following assumption is made on the classification of the mesh entities. For every mesh entity $M_i^{d_i}$ classified on a geometric entity $G_j^{d_j}$ $(d_i \leq d_j)$, the following set of conditions holds:

- If $d_i = 2$,
  - if $d_j = 3$, $M_i^{d_i}$ is used by exactly two mesh entities M_k^3, and $M_k^3 \sqsubset G_j^3,\ k = 1, 2$.
- If $d_i = 1$,
  - if $d_j = 3$, for every mesh entity $M_k^{d_k}$ using it, $M_k^{d_k} \sqsubset G_j^3$,
  - if $d_j = 2$, there are exactly two triangles M_k^2 using it and classified on a model face, and $M_k^2 \sqsubset G_j^2,\ k = 1, 2$,
  - if $d_j = 1$, there are at least two triangles using it and classified on a model face. Note that the classification of the triangles can be the same, as depicted in Figure III.2(a).
- If $d_i = 0$,
  - if $d_j = 3$, for every mesh entity $M_k^{d_k}$ using it, $M_k^{d_k} \sqsubset G_j^3$,
  - if $d_j = 2$, there are at least three edges M_k^1 using it and classified on a model face, and $M_k^1 \sqsubset G_j^2,\ k = 1, 2, 3$. The other edges using it are classified on a model region.
  - if $d_j = 1$, there are exactly two edges M_k^1 using it and classified on a model edge, and $M_k^1 \sqsubset G_j^1,\ k = 1, 2$,
  - if $d_j = 0$, there is at least 1 edge M_k^1 using it and classified on a model edge. Note that the case in which only one edge using it is classified on a model edge is rare and corresponds to the summit of a cone (see Figure III.2(b)).

In this work, we define the topological compatibility of the mesh to the geometrical model by this set of conditions. The conditions have to be satisfied at all times. The initial mesh is supposed to be topologically compatible and the modifications performed on the mesh have to maintain the compatibility.

III.4 Mesh modifications on boundaries

In this section, we emphasize two requirements imposed to the mesh modifications of the adaptation procedure:

1. The shape of the mesh has to be compatible with the shape of the geometry. This means that a mesh entity $M_i^{d_i} \sqsubset G_j^{d_j}$ has to lie on the zone of the space defined by the geometrical representation of $G_j^{d_j}$. For instance, a vertex classified on a line should not be relocated outside of the line.

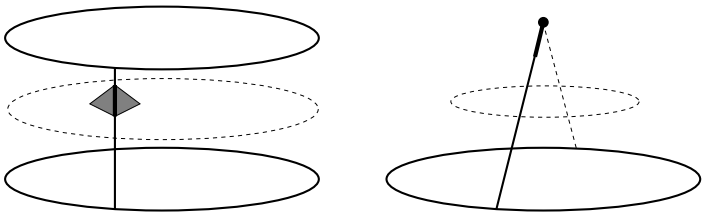


Figure III.2: Geometrical models for which the mesh involves (a) mesh edges classified on a model edge and used by two triangles classified on the same model face, and (b) a mesh vertex classified on a model vertex and used by only one edge classified on a model edge.

2. The mesh is supposed to be *topologically compatible* with the geometry at every time in the mesh adaptation procedure. We assume that the initial mesh is topologically compatible. In order to satisfy this condition, the modifications operated at the vicinity of a geometric boundary require a special attention. Two issues have to be considered:

   (a) If new mesh entities are created on a boundary, they have to be classified on the appropriate geometrical entity.

   (b) The modification is to be denied if it leads to a dimension reduction, i.e. if two mesh entities classified on boundaries are collapsed on each other. Also, new contacts between geometrical entities cannot be introduced. As an example, Figure 2b shows a mesh on which invalid mesh modifications lead to incompatibilities with the geometrical model.

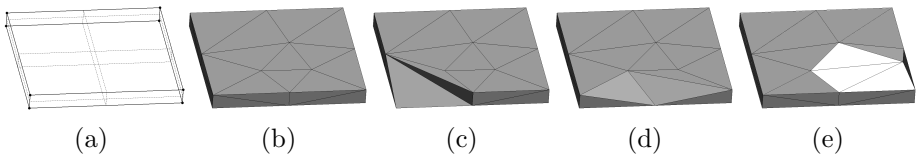


Figure III.3: Mesh modifications leading to an invalid mesh: (a) representation of the geometrical model, (b) initial mesh, (c), (d) and (e) mesh after different invalid edge collapses.

We introduce the following notation: the edge with summits v_i and v_j is denoted $\{v_i, v_j\}_e$, the triangle with summits v_i, v_j and v_k is noted $\{v_i, v_j, v_k\}_f$, and the tetrahedron with summits v_i, v_j, v_k and v_l is noted $\{v_i, v_j, v_k, v_l\}_t$.

In the remainder of this section, we analyze each mesh modification and describe the checks that have to be performed in order to comply to the topology and shape of the geometrical model.

Edge split: when a boundary edge is split, the new vertex has to be classified on the same geometric entity as the edge to be split, as well as the two resulting edges. Also, the faces resulting from the splitting of the classified faces have to be classified on the geometrical entities on which the deleted faces are classified.

Once every mesh entity is classified on the right geometric entity, we still have to snap the new vertex at a position located on its geometrical entity. For edges classified on a surface, the parametrization $\mathcal{P}$ of the geodesic curve of the surface passing through the extremities of the edge (see Figure III.4) is computed and the coordinate $\mathbf{u}_{\mathcal{P}}$ of the new vertex on $\mathcal{P}$ is obtained by:

$$\mathbf{u}_{\mathcal{P}} = (1-r)\,\mathbf{u}_{\mathcal{P}1} + r\,\mathbf{u}_{\mathcal{P}2}, \tag{III.4}$$

where $\mathbf{u}_{\mathcal{P}1}$ and $\mathbf{u}_{\mathcal{P}2}$ are the parametric coordinates of the summits of the split edge in the parametrization $\mathcal{P}$, and r is a value in $[0,1]$ giving the center of the straight edge in the metric space, the summits of the edge corresponding to $r = 0$ and 1. In order words, we choose to snap the new node on its closest point on the surface.

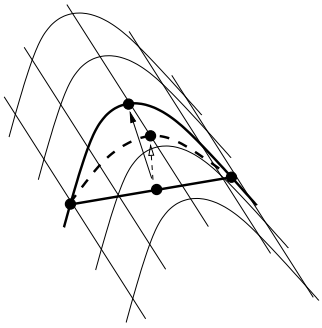


Figure III.4: Illustration of the snapping on a geodesic of the surface ((III.4)): the dashed line is the geodesic curve passing through the summits of the edge. The continuous curve is the curve on which the new vertex would lie if (III.5) was used.

A simpler way to compute the target location of the new vertex on a surface with a parametrization $\mathcal{S}$ is to give it the following coordinates in $\mathcal{S}$:

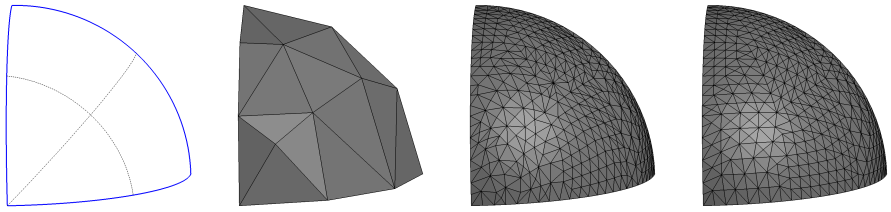
$$\mathbf{u}_{\mathcal{S}} = (1-r)\,\mathbf{u}_{\mathcal{S}1} + r\,\mathbf{u}_{\mathcal{S}2}, \tag{III.5}$$

where $\mathbf{u}_{\mathcal{S}1}$ and $\mathbf{u}_{\mathcal{S}2}$ are the parametric coordinates of the summits of the split edge in the parametrization $\mathcal{S}$. However, choosing a location from (III.4) is found to be more interesting for the following reasons:

- The alteration of the lengths of the new edges is smaller with (III.4) since the amplitude of the motion is minimized. A large relocation could indeed yield edges with a length $L_e^{tr} > L_{up}$ which would lead to the collapse of a new edge and therefore cancel the initial split operation.

- The resulting position does not depend on the parametrization of the surface. Indeed, the same surface can be defined by different parametrizations and we have no guarantee that (III.5) will yield a reasonably small relocation.

Fig III.4 shows two meshes of a sphere obtained by refining the same initial mesh. Fig III.4 (c) and (d) are the meshes obtained respectively using (III.4) and (III.5).



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

Figure III.5: Comparison between (III.4) and (III.5) for the computation of the location of the new vertex: (a) geometry, (b) initial mesh, (c) and (d) meshes after refinement with (III.4) and (III.5) respectively.

If the split edge is classified on a line, the coordinate $\mathbf{u}_{\mathcal{L}}$ of the new vertex in the parametrization $\mathcal{L}$ of the line is computed as:

$$\mathbf{u}_{\mathcal{L}} = (1-r)\,\mathbf{u}_{\mathcal{L}1} + r\,\mathbf{u}_{\mathcal{L}2}, \tag{III.6}$$

where $\mathbf{u}_{\mathcal{L}1}$ and $\mathbf{u}_{\mathcal{L}2}$ are the parametric coordinates of the summits of the split edge in the parametrization $\mathcal{L}$.

The split operation does not lead to topological incompatibilities with the geometry.

Edge collapse: The edge collapse modification is depicted on Figure III.6. We note $e \sqsubset G_e$ the edge being collapsed and its classification, $v_{del} \sqsubset G_{v_{del}}^{d_{v_{del}}}$ the vertex being deleted, $v_{tgt} \sqsubset G_{v_{tgt}}^{d_{v_{tgt}}}$ the vertex on which the edge is collapsed, and v_i^C the other vertices such that the edge $e_i^{mid} = \{v_{del}, v_i^C\}_e \sqsubset G_{e_i^{mid}}^{d_{e_i^{mid}}}$ or the edge $e_i^{top} = \{v_{tgt}, v_i^C\}_e \sqsubset G_{e_i^{top}}^{d_{e_i^{top}}}$ exists.

In order to comply to the geometry (shape and topology), several checks have to be made before allowing the edge collapse to be performed. We consider three sub-cavities of interest in the cavity modified by the edge collapse: the edge e itself, any face $f_i^e = \{v_{tgt}, v_{del}, v_i^C\}_f$ (see Figure III.6(d)), and any tetrahedron $t_{ij}^e = \{v_{tgt}, v_{del}, v_i^C, v_j^C\}_t$ with v_i^C, v_j^C being connected by an edge (see Figure III.6(c)). Note that the checks involve the cases in which f_i^e or t_i^e does not exist. The checks are the followings:

- On the edge e, ensure that $G_{v_{del}} = G_e$.

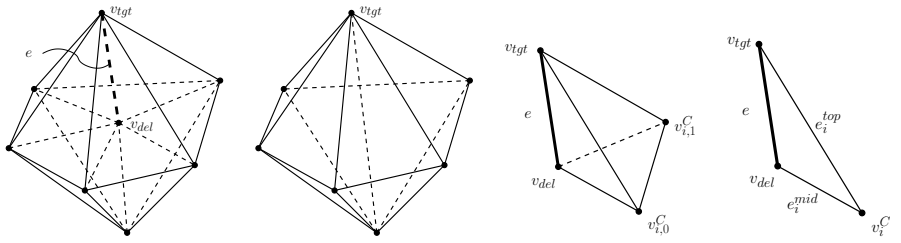


Figure III.6: Edge collapse: (a) edge e to be collapsed and its cavity, (b) cavity after the edge collapse, (c) a tetrahedron t_i^e to be collapsed, and (d) a face f_i^e to be collapsed.

- On the faces $f_i^e \sqsubset G_{f_i^e}^{d_{f_i^e}}$ (see Figure III.6(d)),
  - if e_i^{mid} exists, ensure that f_i^e exists,
  - if $d^{e_i^{mid}} = 1$, ensure that $d^{e_i^{top}} > 1$,
  - if $d^{e_i^{mid}} = 1$ and $d^{e_i^{top}} = 2$, ensure that $G_{f_i^e}^{d_{f_i^e}} = G_{e_i^{top}}^{d_{e_i^{top}}}$,
  - if $d^{e_i^{mid}} = 2$ and $d^{e_i^{top}} = 1$, ensure that $G_{f_i^e}^{d_{f_i^e}} = G_{e_i^{mid}}^{d_{e_i^{mid}}}$,
  - if $d^{e_i^{mid}} = 2$ and $d^{e_i^{top}} = 2$, ensure that $d_{f_i^e} = 2$.
- On the tetrahedrons t_{ij}^e (see Figure III.6(c)), whether t_{ij}^e exists or not, ensure that $\{v_{tgt}, v_i, v_j\}_f \sqsubset G^3$ or $\{v_{del}, v_i, v_j\}_f \sqsubset G^3$.

Face collapse: The face collapse can be seen as a compound edge split-edge collapse operation. It is depicted on Figure III.7. We note $f \sqsubset G_f^{d_f}$ the face being collapsed and its classification, $e \sqsubset G_e^{d_e}$ the edge being split and $v \sqsubset G_v^{d_v}$ the vertex opposite to e in f.

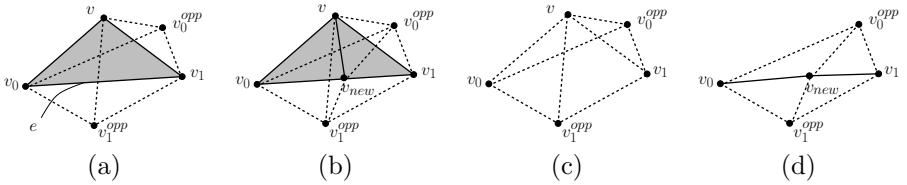


Figure III.7: Face collapse (edge split followed by edge collapse): (a) face f to be collapsed and its cavity, (b) cavity after the split of e, (c) cavity after the edge collapse in the case it is oriented from v^{new} to v, and (d) cavity if the other direction is chosen.

In the case the edge collapse is oriented from the new vertex v_{new} to v (Figure III.7(c)), the face collapse does not require to snap any vertex as the

only new vertex is deleted. On the other hand, the compliance to the geometric topology implies the verification of conditions prior to the application of the modification:

- ensure $G_e^{d_e} = G_f^{d_f}$,
- if we have $\{v, v_i, v_j^{opp}\}_f \sqsubset G_{f_{ij}}^{d_{f_{ij}}}$ and $\{v_0, v_1, v_j^{opp}\}_f \sqsubset G_{f_j}^{d_{f_j}}$, ensure that $\nexists\, i \in [0,1],\, j \in [0,1] \mid \{d_{f_{ij}} = 2 \;\&\; d_{f_j} = 2\}$,
- if we have $\{v_0, v_1, v_k^{opp}\}_f \sqsubset G_{f_k}^{d_{f_k}}$ and $\{v, v_k^{opp}\}_e \sqsubset G_{e_k}^{d_{e_k}}$, ensure that $\nexists\, k \in [0,1] \mid \{d_{e_k} \leq 2 \;\&\; d_{f_k} = 2\}$,
- if $\{v, v_0, v_1, v_i^{opp}\}_t$ does not exists, ensure that $\{v, v_i^{opp}\}_e$ does not exist. Note that a consequence is that the faces $\{v, v_0, v_i^{opp}\}_f$ and $\{v, v_1, v_i^{opp}\}_f$ do not exist as well.

If the edge collapse is applied in the other direction (Figure III.7(d)), v_{new} has to be repositioned on the appropriate geometrical entity in a similar way as in the edge split case. Furthermore, the verifications presented here above are also to be done in this case, except for the first one which is replaced by checking that $G_v^{d_v} = G_f^{d_f}$.

Region collapse: In this modification, two edges e_0 and e_1 belonging to the same tetrahedron t are split, leading respectively to the new vertices v_0 and v_1, and the new edge $e_{01} = \{v_0, v_1\}_e$. Then, e_{01} is collapsed (See Figure III.1(f)). With no loss of generality, we consider here the case in which e_{01} is collapsed from v_0 to v_1. . In order to maintain the mesh compatibility to the geometrical topology, only one condition has to be checked for this modification: if we have the classifications $t \sqsubset G_t^3$ and $e_0 \sqsubset G_{e_0}^{d_{e_0}}$, the condition $G_t^3 = G_{e_0}^{d_{e_0}}$ has to be satisfied.

Edge swap: We denote $e \sqsubset G_e^{d_e}$ the edge to be swapped and its classification. In order to comply to the geometrical topology the swap cannot be performed if $d_e \leq 1$. Furthermore, if $d_e = 2$ another verification has to be made. As a consequence of the compatibility of the mesh to the geometrical topology (see §III.3), if $d_e = 2$, two and only two faces f_i using e are classified on a model face. If we call v_i the vertex of f_i which is opposite to e in f_i, it should be checked that $\{v_0, v_1\}_e$ does not exist. Otherwise, the edge swap cannot be performed.

Face swap: The only condition for a face swap to maintain the compatibility of the mesh to the geometrical model is that the face is classified on a model region.

III.5 Vertex snapping

Given a new vertex v resulting from the split operation applied on a boundary edge e, the location of the vertex is obtained by its parameters $\mathbf{u}_{\mathcal{P}}$ in the parametrization $\mathcal{P}$ of its geometrical entity G.

For that purpose, $\mathbf{u}_{\mathcal{P}}$ is computed by eq. III.4, which assumes the parameters $\mathbf{u}_{\mathcal{P}i}$ of the summits of e in $\mathcal{P}$ are known. Actually, if the summits of e are classified on a different geometrical entity than G (we name this geometrical entity H, with a parametrization $\mathcal{H}$), we have to find $\mathbf{u}_{\mathcal{P}i}$ from $\mathbf{u}_{\mathcal{H}i}$, which requires the reparametrization of $\mathcal{H}$ in $\mathcal{P}$. Such a reparametrization is usually available in the interface of the CAD models, and Gmsh provides this functionality.

Given $\mathbf{u}_{\mathcal{P}}$ and $\mathcal{P}$, obtaining the coordinates $\mathbf{x}$ of v is straightforward. After the edge split, the location of v is on e, while the target location is $\mathbf{x}$ (see Figure III.5).

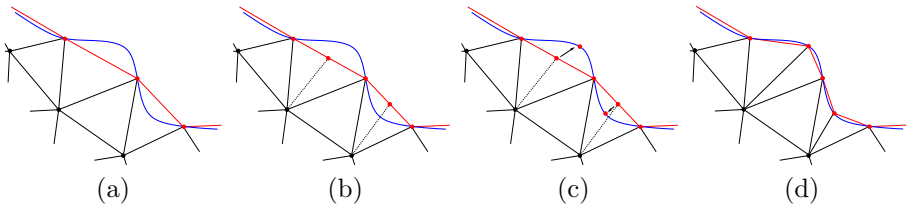


Figure III.8: Edge split operation on a boundary. The boundary mesh entities are represented in red, the geometrical representation of the boundary is in blue. In (b), two boundary edges are split. In (c), the target location of the new vertices on the geometrical boundary has been computed, and in (d) the vertices have been snapped on the boundary.

In most of the cases, one can simply move v to its target location without encountering any problem with the neighboring elements. However, in a few cases, snapping v to $\mathbf{x}$ leads to tangled elements, as illustrated on Figure III.5.

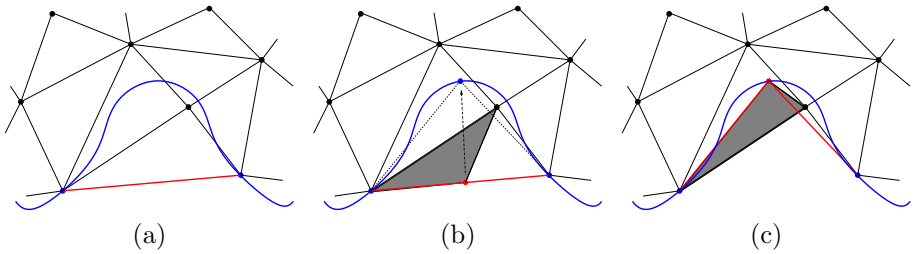


Figure III.9: Edge split and vertex snapping leading to tangled elements: (a) initial mesh, (b) mesh after the split, (c) mesh after the vertex snapping. The shadowed triangle is returned when the new vertex is snapped.

In order to circumvent that issue, we propose a method to adapt the mesh at the vicinity of the boundary. We first select a set S of elements surrounding the vertices to be snapped. We then compute the displacements dX_i of the vertices contained in S by making an analogy between S and an elastic material with a technique similar to the one presented in [177]. The target displacements of

the vertices to be snapped is imposed as a Dirichlet boundary condition in the elastic model.

This inner vertex repositioning allows to perform a larger displacement when a vertex has to be snapped. However, even if S contains the whole mesh, there are still situations in which the snapping with inner vertex repositioning leads to tangled elements. In order to avoid such a situation, a stepping procedure is proposed in which only a part α of dX_i is performed, with $0 \leq \alpha \leq 1$. The value of α is taken as the largest value in $[0, 1]$ that doesn't lead to an element flipping. The worst elements of S are then eliminated by edge or face swaps, edge or face collapses, or individual vertex relocations. Thereafter, a higher value of α can be chosen. This procedure is iterated until $\alpha = 1$.

The snapping algorithm is summarized as follows:

```
1. Select the set S of elements surrounding the vertices
   to be snapped
2. Compute the relocations dX_i of the inner nodes of S by an
   elastic analogy
3. Set α = 0
4. While ( α < 1 ), do
  4.1 Find the maximum α in [0,1] such that applying the
  relocations α dX_i does not tangle any element, and apply it
  4.2 If α = 1, exit the while loop
  4.3 Eliminate the sliver elements in S by
      local mesh modifications
  4.4 If no sliver was eliminated in 4.3, the snapping fails
 End of while
```

Note that the motion of the snapped vertices is performed all at once: all nodes are repositioned together with the same ratio α. Moving nodes independently from their neighbours would generate new situations in which tangled elements appear because of the local differences in the repositioning advancement.

This method is able to handle a very large number of snappings and is far more robust than the simple boundary vertex relocation, or the relocation with an elastic analogy but no sliver handling. However, the success of the method cannot be guaranteed as it depends on the ability to eliminate the elements turned into slivers during the stepping. If the snapping procedure fails (step 4.4 in the procedure), the snapping is not complete. In this case, the vertices are not moved to the desired location but they keep their classification on the boundary as well as their parametric coordinates so they can be relocated later, when the blocking cavities are modified. A local re-meshing could also be applied in that case, as in the second step of the procedure proposed in [105]. Such a case is very rare and does not happen in the computations presented in the present paper.

III.6 Results

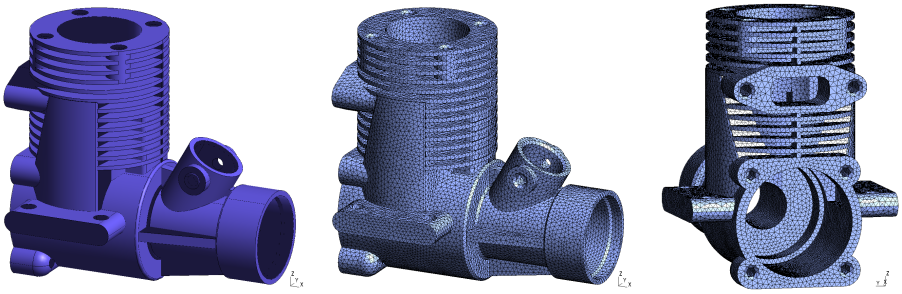
In order to show the robustness of the method, we present two examples of meshes of relatively complex geometries. In the first example, a time-dependent

size field is prescribed, leading to refinements and coarsenings on the boundaries. The second example is a linear elasticity problem in which the error is minimized using the mesh adaptation procedure.

The open source software Gmsh [78] was used for the generation of the initial meshes and the representation of the CAD models and meshes. Also, the CADs presented here are provided in the open source MAdLib [46] package so that the proposed examples can be easily reproduced.

Example 1: engine block

In this example, we modify the mesh of an engine block. The geometry is given by a STEP file. The geometry and the initial mesh are depicted on Figure III.10. Most of the classical geometrical features are present in the



(a) (b) (c)

Figure III.10: Engine block test: (a) representation of the CAD model, (b) and (c) views of the initial mesh.

CAD (straight planes, cylinders, cones, spheres, B-splines, ...), and some parts of the mesh are enclosed in very thin regions.

A global coarsening is applied to change the global (uniform) mesh size from $\delta = 1$ to $\delta = 5$, and a sinusoidal refinement front is propagated through the whole block by prescribing the following edge length:

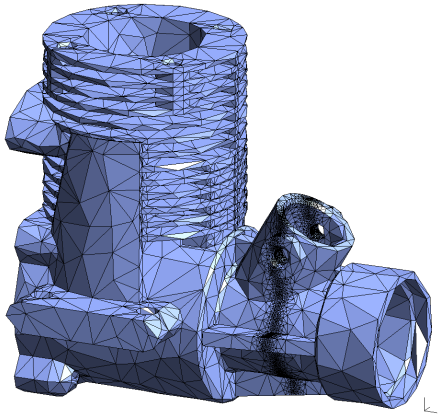
$$\delta = 5 \quad \forall |x - v\,t| > L/2, \tag{III.7}$$

$$\delta = 5 - 4.8\cos\left((x - v\,t)\,\frac{\pi}{L}\right) \quad \forall |x - v\,t| \leq L/2. \tag{III.8}$$

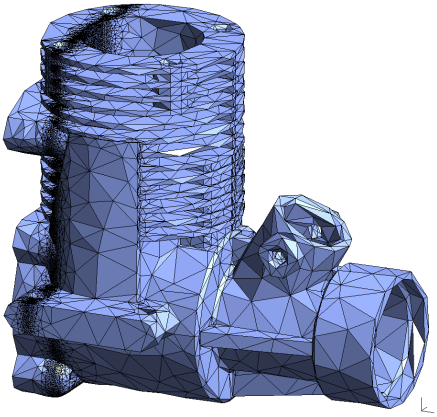
where L is the width of the refined zone, corresponding to a half-period of the cosine and v is the propagation velocity.

We choose $L = 10\,cm$ and $v = 1\,cm/s$. The length of the block in the x direction is $53\,cm$ and a time step of $1s$ is chosen, thus leading to a computation of 73 time steps.

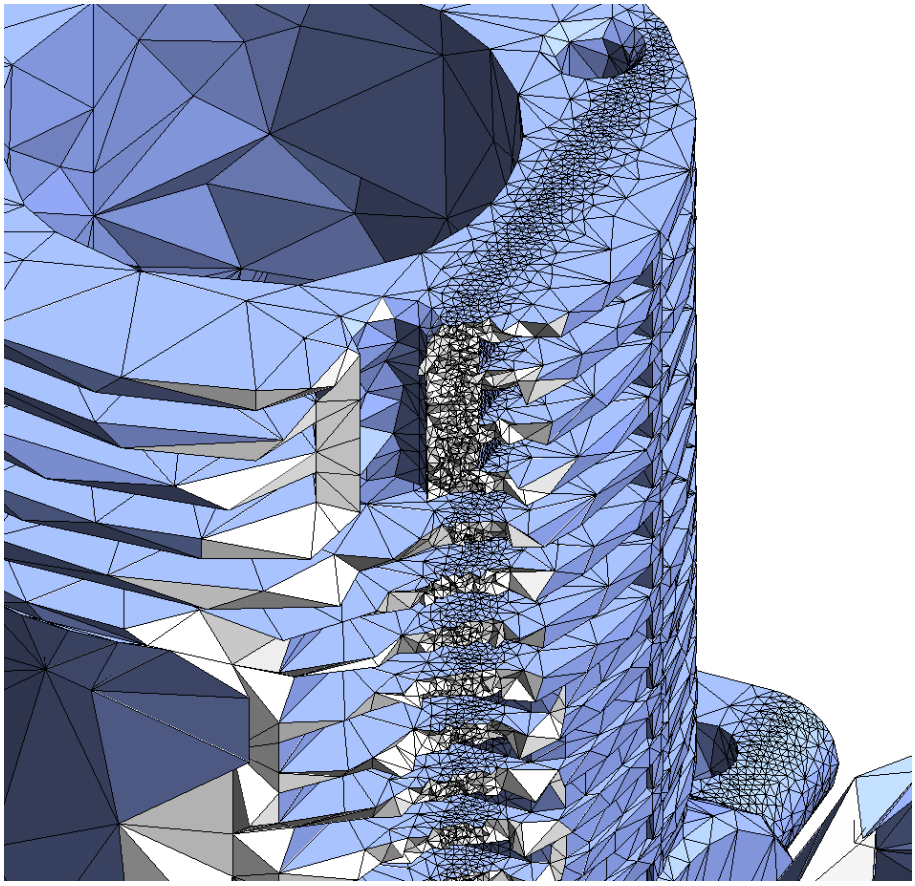
Figure III.11 shows the mesh at different time steps. We can see that despite the very complex parts of the geometrical model and the high level of coarsening reached out of the refinement zone, the proposed method is able to achieve the adaptation. We remark that the snapping algorithm never fails.



(a) time step 23



(b) time step 54

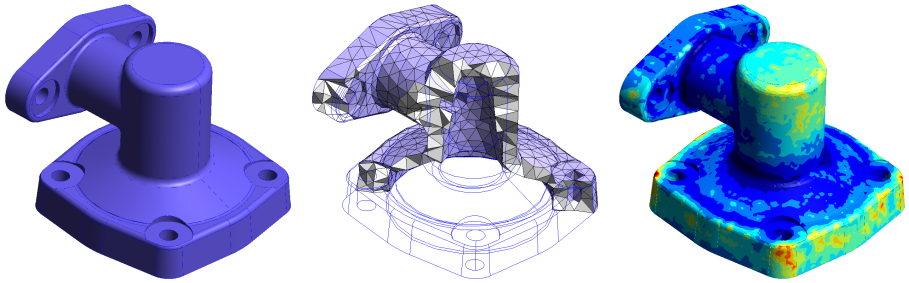


(c) time step 36

Figure III.11: Engine block test: mesh at different time steps.

Example 2: error minimization

In this test, we adapt the mesh presented in Figure III.12(b) in a way that minimizes the discretization error of the finite element computation of an elasticity problem. The problem consists in solving the linear elasticity equation



(a) (b) (c)

Figure III.12: Error-oriented adaptation: (a) geometry, (b) initial mesh, and (c) size field obtained with equation III.13.

on the piece presented in Figure III.12(a), when the bottom surface is fixed, and a couple M is applied to the upper plate.

More precisely, we define the error e_i^2 on an element Ω_i as the energy norm $\|.\|$ of the difference between the finite element solution u_h and the exact solution u of the elasticity problem:

$$e_i^2 = \int_{\Omega_i} \|u - u_h\|^2 \, dv. \tag{III.9}$$

The local discretization error converges to zero at a convergence rate $k = 1$ for this problem

$$e_i = Ch_i^k \tag{III.10}$$

with h_i the mesh size, and C a constant that does not depend on h_i. Our goal is to minimize the total error e^2 on the mesh, defined as

$$e^2 = \sum_{i=0}^{N} e_i^2, \tag{III.11}$$

with a target number of elements $\bar{N}$.

To this end, we apply the method proposed in [149], which defines the reduction factor r_i to be applied on edge lengths in order to obtain the minimal error e^2 with a mesh containing $\bar{N}$ elements. It can be proved [149] that, posing $\alpha = 2k/d$, with d the dimension of the problem, we have

$$r_i = e_i^{\frac{2}{d(1+\alpha)}} \alpha^{\frac{1}{d(1+\alpha)}} \left[\frac{(1+\alpha)\bar{N}}{\left(\sum_{i=1}^{N} e_i^{\frac{2}{1+\alpha}}\right)\left(\alpha^{\frac{2+\alpha}{1+\alpha}} + \alpha^{\frac{1}{1+\alpha}}\right)} \right]^{\frac{1}{d}}. \tag{III.12}$$

Equation III.12 allows us to define a size field based on the initial mesh size h_i:

$$\delta_i = \frac{h_i}{r_i}, \tag{III.13}$$

where h_i is chosen as the circumradius of the element Ω_i belonging to the initial mesh. Figure III.12(c) shows the size field obtained for $\bar{N} = 270000$, with a uniform mesh containing $\bar{N}$ elements.

The mesh adaptation method is applied on the initial mesh (Figure III.12(b)) with this size field. The results are shown in Figure III.13. The method presented here was able to move from a relatively coarse mesh to a mesh with much finer elements. The mean edge length in the initial mesh is about 9.0, while the minimal size prescribed is 0.56.

With the adapted mesh, we obtain $e^2 = 5.61\,10^8$ while an error of $9.65\,10^8$ was obtained with a mesh containing the same number of uniformly distributed elements.

III.7 Conclusion

A new technique to snap the vertices on their geometrical boundary is presented. It consists in relocating the neighboring vertices with a technique based on an elastic analogy. The repositioning is reinforced by a set of mesh modifications which are applied if invalid elements appear. Different levels of refinement can therefore be obtained on various geometries.

The different mesh modifications performed in the classical mesh adaptation methods are also analyzed regarding the conformity to the geometrical model. It is shown how these operations can be performed while keeping the mesh compatible to the geometrical model at all times.

In order to facilitate future developments and repeatability of the numerical experiments, the techniques proposed here are implemented in the open source library MAdLib.

In the future, we aim at using the current method to automatically build anisotropic meshes around the geometrical boundaries in order to capture boundary layers in viscous flow computations.

Figure III.13: Mesh adaptation based on error minimization: adapted mesh, with the initial mesh and the size field shown resp. in Figure III.12(b) and (c).

Chapter 5

Implementation of an open source library

When dealing with numerical mechanics, the availability of well designed software solutions implementing recent numerical methods is crucial. This is a reality for both industrial community that needs state-of-the-art softwares to solve new numerical problems without going through the time-consuming process of software design, and researchers who may want both to improve the existing methods or to combine it with other projects. This is especially true for mesh adaptation since it is more a tool for the numerical methods than a goal in itself. The time spent by a researcher in writing source code can be very significant, sometimes for implementing already published and tested techniques, when no open source solution is available.

The meshing field (generation and adaptation) is especially technical in the sense that the efficiency and robustness of a method is generally highly dependent on its implementation. This is the case for the mesh adaptation by local modifications. As an example, several choices are to be made when designing a mesh database. Non-optimal choices would significantly affect the performances of the method. The repeatability of the numerical experiments is also very dependent on the implementation.

However, although some open source softwares have recently emerged in the field of mesh generation (Netgen, Tetgen, Gmsh, ...), no open solution has really broken up yet in the mesh adaptation domain. By publishing the implementation of the techniques described in the previous chapters in a library, MAdLib (**M**esh **Ad**aptation **Lib**rary), we open the possibility for a community involving developers and users to be built around it, like it was the case for the packages cited here above in the field of mesh generation.

Together with the publication of the sources in MAdLib, this chapter reaches several aspects of the technical achievement of an adaptive computation. The most generic aspects of the work are presented in Article IV, *A mesh adaptation framework for dealing with large deforming meshes*, Compère, Remacle, Jansson, Hoffman (accepted for publication in the International Journal for

Numerical Methods in Engineering) presented here after. The issues addressed in the paper are the following:

Mesh database How to build a fast and relatively compact mesh database which fulfills the requirements of a mesh adaptation package: fast iteration over mesh elements, on-the-fly creation and deletion of mesh entities, ... ? What is the memory load resulting from the proposed solution ?

Solution projection An advantage of the adaptation by local mesh modifications is that the solution has only to be projected locally. It implies to perform a particular action when a cavity of the mesh is modified which requires a specific implementation inside the mesh adaptation package. However, the exact definition of the projection differs from a numerical method to another. How to cope with this technical issue ?

Insertion in a physics solver and interface How to use an external mesh adaptation package like MAdLib in a physics solver ? How to manage the solution and data allocation and projection ?

The performances of the proposed implementation are also analyzed in terms of computational efficiency, which opens the way to comparisons with other packages or methods.

The programming interface of MAdLib is described in the last section of this chapter.

Article IV

A mesh adaptation framework for dealing with large deforming meshes

Gaëtan Compère[1*], Jean-François Remacle[1,2], Johan Jansson[3], Johan Hoffman[3]

[1] *Université catholique de Louvain, Department of Civil Engineering, Place du Levant 1, 1348 Louvain-la-Neuve, Belgium*

[2] *Center for Systems Engineering and Applied Mechanics (CESAME), Université catholique de Louvain, 1348 Louvain-la-Neuve, Belgium*

[3] *Royal Institute of Technology, Computational Technology Laboratory, Stockholm, Sweden*

Abstract

In this paper, we identify and propose solutions for several issues encountered when designing a mesh adaptation package, like mesh-to-mesh projections and mesh database design, and we describe an algorithm to integrate a mesh adaptation procedure in a physics solver. The open source MAdLib package is presented as an example of such a mesh adaptation library. A new technique combining global node repositioning and mesh optimization in order to perform arbitrarily large deformations is also proposed. We then present several test cases to evaluate the performances of the proposed techniques and to show their applicability to fluid-structure interaction problems with arbitrarily large deformations.

Key words: Mesh adaptation, local modifications, large deformations, open source, fluid-structure interaction

IV.1 Introduction

Mesh motion due to a moving interface or boundary is an essential component in many modern finite element procedures [64, 90, 177], with applications in many domains, and in particular the numerical computation of fluid-structure interactions (FSI).

The mesh motion algorithm, or *r-adaptivity*, is a crucial ingredient of FSI computations. The standard mesh motion algorithm moves mesh points classified on moving/deforming interfaces. Then some kind of mesh smoothing is applied to the rest of the mesh in order to maintain reasonable mesh quality. The topology of the mesh is not modified during this process which means that the underlying graph of the mesh remains unchanged. However, this approach is not general and fails for even simple motion such as large rigid body translation and rotation. Furthermore, even though the mesh topology may be

preserved for simple mesh motion, such a procedure gives little control of the mesh size field $\delta(\mathbf{x}, t)$: some cells may be compressed or stretched undesirably due to the mesh motion and smoothing, likely leading to large error in the solution.

For what concerns large deforming domains, only a few works can be found in the literature about the mesh adaptation methods. The first achievements made to supply r-adaptive methods with the local mesh modifications consisted in applying refinement/coarsening procedures according to both shape and deformation measures of the elements [14, 13]. The robustness of the method was improved recently by adding edge and face swaps to eliminate sliver elements [38, 54]. In [48], a procedure based on local mesh modifications with a more robust sliver elimination algorithm is extended to the case of large domain deformations. Other authors [59] use Delaunay point insertions to provide anisotropic deforming meshes. However, all these procedures are not infallible since a very large domain deformation can still cause the node repositioning algorithm to fail.

The aim of our mesh adaptation process is, as usual, twofold: (i) to satisfy a prescribed mesh size field $\delta(\mathbf{x}, t)$ and (ii) to maximize mesh quality. In this paper, we detail the design choices that have been made to build a general mesh adaptation procedure applied to large mesh deformations from the mesh adaptation method and a node repositioning procedure based on an elastic analogy. In particular, we present a new technique to allow arbitrarily large domain deformations by applying mesh modifications during the node repositioning step itself instead of only adapting the mesh between two repositionings. Applying mesh modifications at this stage avoids the apparition of poorly shaped or tangled elements, which highly increases the robustness of the technique. A global procedure that allows general mesh motion is then presented. The procedure is based both on standard local mesh modification operators (edge splits, edge collapses and edge swaps) and the proposed node repositioning technique.

Finite element formulations in the time domain allow the mesh to vary in time. For vertex motions, formulations are usually written in the arbitrary Lagrangian-Eulerian framework. When topological modifications are performed, mesh to mesh interpolations are usually used [152, 148]. Disappointingly, most of the state-of-the-art finite element implementations only allow a limited set of operations. For example, implementations in [17, 68, 49] only allow local mesh refinement (no coarsening). However, several authors have proposed more general methods for local mesh adaptation. These methods either use local re-meshing [50], or rely on a larger set of mesh modifications [94, 32, 36, 55], leading to a well-proved [127, 152, 154, 163, 3, 44] class of mesh adaptation methods for fixed domain boundaries. But no open source implementation of such a method is available. Finally, some packages allow global re-meshing using closed and ad-hoc mesh generation softwares, as suggested in various works [136, 75, 4].

This paper can be seen as the technical companion of the MAdLib (**M**esh **Ad**aptation **Lib**rary) library. We have decided to distribute MAdLib as free software under the LGPL license. We hope to build a community around MAdLib, in the same manner as we have already done for Gmsh [77]. In our

opinion, there are only good reasons for distributing such a code as an open source. First, mesh adaptation procedures are very technical, in the sense that their robustness is extremely sensitive to their implementation. Mesh related codes need time and users to become usable. Therefore, distributing MAdLib will allow researchers to use an already stable version of mesh adaptation routines. On the other hand, the bigger the community is, the faster the remaining problems will be resolved. Another good reason for going open source is that we are convinced that adaptive procedures have not reached a sufficient impact in engineering design. We believe that one of the reasons of that relative success is that there are too few freely available solutions. Finally, mesh generation/adaptation in a scientific project is generally not an aim in itself: it is a tool that lies in the same category as linear system solvers or linear algebra packages. For that reason, research on meshing is done by relatively few research teams in comparison to the research that is done in finite element analysis. Therefore, mesh generation researchers should make their research as widely available as possible to accelerate scientific development.

The present paper has several goals: to explain the key points in the design of a mesh adaptation procedure that enables to deal with large mesh deformations, to discuss the most generic aspects of the implementation of a mesh adaptation package, and to show the efficiency and robustness of the proposed method and implementation by solving some non-trivial test cases. To this end, some test problems are presented:

1. A rotating propeller. This test shows the robustness of the mesh adaptation algorithm and analyzes the quality of the resulting meshes regarding the prescribed element quality. The efficiency of the implementation is also analyzed in terms of CPU time and memory consumption.

2. Two rigid spheres that fall in a viscous fluid. This example demonstrates the coupling of mesh adaptation with a fluid solver. In particular, the relative cost of mesh adaptation is studied.

3. A 3D turbulent fluid-structure flag problem representing a problem we can expect in real-world applications, where we compare standard mesh smoothing with mesh smoothing plus mesh adaptation.

The first section will recall the definition of the size field and the mesh adaptation by local modifications, and presents the global procedure for controlling a mesh with moving boundaries. Some efficiency aspects of the method are also reached. The next section describes the adaptive mesh database. The general coupling scheme with a physics solver is then described, including the issues related to the handling of projection algorithms through the adaptation process. The last section provides results from the application of the method to the different test cases.

IV.2 Mesh adaptation

There are basically two kind of techniques that enable to adapt a mesh. Remeshing techniques consist in removing the existing mesh and replacing it by

an adapted one. In the context of transient computations, this approach has two important drawbacks: (i) complete re-meshing introduces a lot of numerical diffusion in the mesh-to-mesh interpolation procedure and (ii) re-meshing approaches are difficult to be applied when the computation is done in parallel. The alternative way to do mesh adaptation is to use local mesh modifications. The latter technique is (surprisingly) known to be slower than re-meshing. Yet, it can be applied in parallel and it usually introduces much less numerical dissipation [152].

In this section, we start by briefly recalling the general concepts governing the mesh adaptation methods based on local modifications. The interested reader can refer to [55, 106] for more complete descriptions of the adaptation methods for fixed domains, and to [59, 48] for previous applications of the method to large deformations. We then present a new technique to perform arbitrarily large domain deformations, and we finish with considerations about efficiency intended to improve the performances of the mesh adaptation procedure.

Mesh Size Field: The mesh size field is a standard way of prescribing mesh sizes. It consists in defining a function $\delta(\mathbf{x}, t)$ that describes optimal mesh sizes at any point $\mathbf{x}$ of the domain and at any time t of the possibly time dependant simulation, see for instance [136, 106].

Using the size field, it is possible to define the non-dimensional length L_e^{tr} of edge e as

$$L_e^{tr}(t) = \int_e \delta^{-1}(\mathbf{x}, t) dl.$$

An edge with a non-dimensional size of $L_e^{tr} = 1$ is an edge with an optimal size. It is usually impossible to build meshes for which edges have the optimal size everywhere. Therefore, a range $[L_{low}, L_{up}]$ of acceptable sizes has to be defined: an edge for which $L_e^{tr} < L_{low}$ is a short edge while an edge with $L_e^{tr} > L_{up}$ is a long edge. This range of acceptable edge lengths is a very sensitive parameter of the adaptation process.

Local mesh modifications: Consider a mesh $\mathcal{M} = \{M_1, \ldots, M_N\}$ composed of N elements M_j, $j = 1 \ldots, N$. A cavity $\mathcal{C} = \{M_{k_1}, \ldots, M_{k_n}\}$ is a subset of M that forms a simply connected domain. For example, all mesh elements connected to one mesh edge or to one mesh vertex form a cavity. A local mesh modifications consist in removing elements from a cavity $\mathcal{C}$ and replacing them by a new submesh $\mathcal{C}'$ with elements that conform to the boundary of $\mathcal{C}$. Formally, we write

$$\mathcal{M}' = \mathcal{M} - \mathcal{C} + \mathcal{C}'.$$

We use a finite set of local mesh modifications. Elementary local mesh modifications are

- The edge split operator (see Figure IV.1(a)) that consists in splitting one edge of the mesh.
- The vertex collapse [104] operator (see Figure IV.1(b)) that consists in removing one vertex from the mesh.

- The edge swap [70] (see Figure IV.1(c)) for which one edge is removed from the mesh.
- The face swap [70] (see Figure IV.1(d)) for which one triangular face is removed from the mesh.

Swap operators (edge swaps and face swaps) aim at improving locally the quality of the elements. Splits and collapses are there to control the mesh size: long edges are split while one of the two vertices of short edges are collapsed [104,48].

Compound operators are also defined:

- The face collapse operator [104] (see Figure IV.1(e)) for which an edge of the face is split and the new vertex is collapsed on the opposite vertex, or the opposite vertex is collapsed on the new one.
- The double edge split collapse [104] (see Figure IV.1(f)) for which two opposite edges of a tetrahedron are split and one of the new vertices is collapsed on the other one.

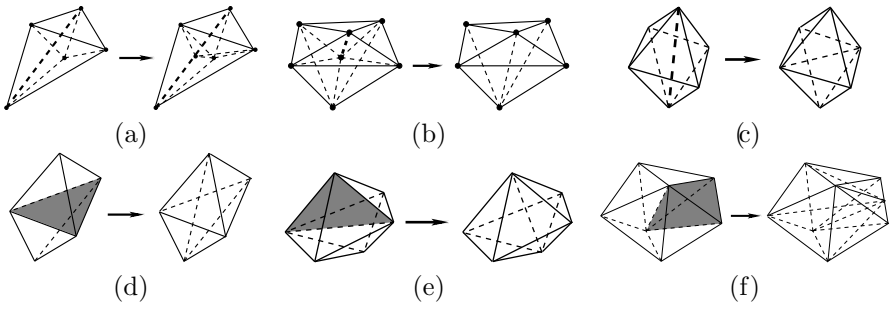


Figure IV.1: The mesh modification operators: (a) edge split, (b) edge collapse, (c) edge swap, (d) face swap, (e) face collapse, (f) region collapse.

Compound operators are usually designed to eliminate sliver elements [104, 48], i.e. elements with a very poor quality but no short or long edge.

Some authors also use template refinement [104]: long edges are all split at once and templates are defined that enable to divide one tetrahedron that has one to six split edges. In our work, we have found out that the use of templates was not necessarily a good idea, for three main reasons: (i) the use of the edge split operator only is more efficient in term of CPU time that the template refinement, (ii) the implementation of template refinement is extremely tedious and (iii) template refinement introduces non-tetrahedrizable polyedra (Schönhardt polyedra, [159]) so that the introduction of unwanted extra points (Steiner points) is mandatory.

Starting from an initial mesh, the adaptation procedure applies sequentially edge splits, vertex collapses and swaps. Compound operators are finally used for eliminating sliver tetrahedra. All the ingredients of the recipe are known. Yet, obtaining the good recipe is difficult. The following issues have to be addressed

- What is the optimal sequence of operators? Here, we apply collapses first, in order to reduce the size of the mesh before producing new nodes. This avoids memory peaks. Then, the most costly operations, swaps and slivers elimination, are applied while the size of the mesh is minimal, and finally edge splits are performed. The sequence is then reproduced until no modification occurs.

- How do we define the range $[L_{low}, L_{up}]$ of acceptable edge lengths? A sharp range will certainly introduce infinite loops between edge splits and collapses. A wider range will produce sub-optimal meshes [48]. Here, we use $[1/\sqrt{3}, \sqrt{3}]$.

- How do we deal with slivers? Looking at edge lengths is, in 3D, not sufficient to control tetrahedra's volumes. The elimination of slivers is a hard task. Slivers can be classified in different categories (see [104]). For each category, a specific sequence of operators [48] is applied that maximizes the probability of the removal of the sliver.

Mesh adaptation on a deforming domain: In the case of deforming domains, the mesh adaptation procedure generally combines two steps:

1. a global node repositioning step: nodes of the deforming boundaries are moved and the displacement of those nodes are propagated using an elliptic PDE. A common choice for the PDE is to use an elastic analogy [186, 13]. The node repositioning stage may be sufficient for small deformations.

2. When large deformations occur, a local mesh modification procedure like the one described in the previous paragraph enables to extend the applicability of the adaptation to arbitrarily large deformations. A mesh adaptation procedure is usually called between two repositionings in order to optimize the mesh and "prepare it" for the next motion.

The two steps are usually performed sequentially. The node repositioning does not require to reallocate the resources for the storage of the mesh, solution and other data. As a consequence, the node repositioning is usually called every time a boundary is moved, while the adaptation by local mesh modification is called when the quality of the mesh is not sufficient or the size criterion is not fulfilled. The local mesh modifications require local projections of the solution while the node repositioning can be taken into account in an arbitrary Lagrangian-Eulerian (ALE) formulation of the problem. Note that some authors are working on an extension of the ALE formulation to edge swaps (in 2D). It is therefore possible that, in the future, some of the local mesh modifications operations (essentially the ones that conserve the number of mesh vertices) could be taken into account inside the ALE formulation.

In the MAdLib package, from which the results of §IV.5 are obtained, the node repositioning is based on an elastic analogy with a variable stiffness [177]. The resulting linear system is solved by a conjugate gradient method. From an implementation point a view, the PETSc [16] library is chosen to solve the

linear system, as it is open source and it provides the best performances among the tested solvers.

When the mesh is deformed slowly, i.e. when the domain boundaries are not moved too much between two node repositionings, it is possible to adapt the mesh by simply combining the two steps: node repositioning and optional mesh adaptation. On the other hand, when the motion of the boundaries is large between time steps, the node repositioning can fail in returning a valid mesh.

Figure IV.2 shows a simple 2D test problem that illustrates this issue and the new technique proposed to solve it. A non convex object is rotated and translated through the domain. The colors given to the elements are related to their quality measure, η: for a triangle T and a tetrahedron K, we choose the mean ratio which is given respectively by

$$\eta_T = 4\sqrt{3}\frac{A_T}{\sum_{i=1}^{3}\left(l(e_i)\right)^2}, \quad \text{and} \quad \eta_K = 12\frac{\left(3V_K\right)^{2/3}}{\sum_{i=1}^{6}\left(l(e_i)\right)^2},$$

with A_T the area of T, V_K the volume of K, and $l(e_i)$ the length of the i^{th} edge of T or K respectively. This quality measurement lies in the interval $[0, 1]$, an element with $\eta = 0$ being a flat element.

The mesh around the object at its initial position is shown in (a). At the next time step, the position is given by (b). The displacement from (a) to (b) is quite large. A simple node repositioning yields the mesh shown in (c): the mesh comprises tangled elements. In order to circumvent that issue, a solution is presented in Figures IV.2 (d), (e) and (f). It consists in stopping the node repositioning just before a first element is returned (see (d) and (e)), and to apply edge swaps in order to eliminate the worst elements (see (f)). The node repositioning is then continued. This procedure is applied iteratively until the object has reached its final location. Note that the elastic computation is only made once.

A possible issue with this approach is the agglomeration of ill shaped elements in a region of the mesh (see Figure IV.2(d)). When several sliver elements are grouped together, swapping edges can be inefficient in eliminating them because applying an edge swap inside the group will likely generate other ill shaped elements. Consequently, the previous approach is modified in order to apply edge swaps before the step shown in Figure IV.2(d) is reached. We fix a threshold value Q^* for the elements quality under which the node repositioning is stopped, and the edge swap optimization is applied, as depicted in Figures IV.2 (g), (h) and (i). In (g), the node repositioning is stopped because an element with a quality $Q < Q^*$ exists (see (h)). Edge swaps are then applied to eliminate the worst elements (see (i)) and the motion is continued, yielding the mesh in (b). In our experience, the values $Q^* = \eta_T^* = 0.2$ and $Q^* = \eta_K^* = 0.1$ yield very robust node repositioning procedures for respectively 2D and 3D meshes.

Implementation. Some considerations about the computational complexity of the mesh adaptation methods can help to greatly improve the efficiency in terms of CPU time and memory consumption.

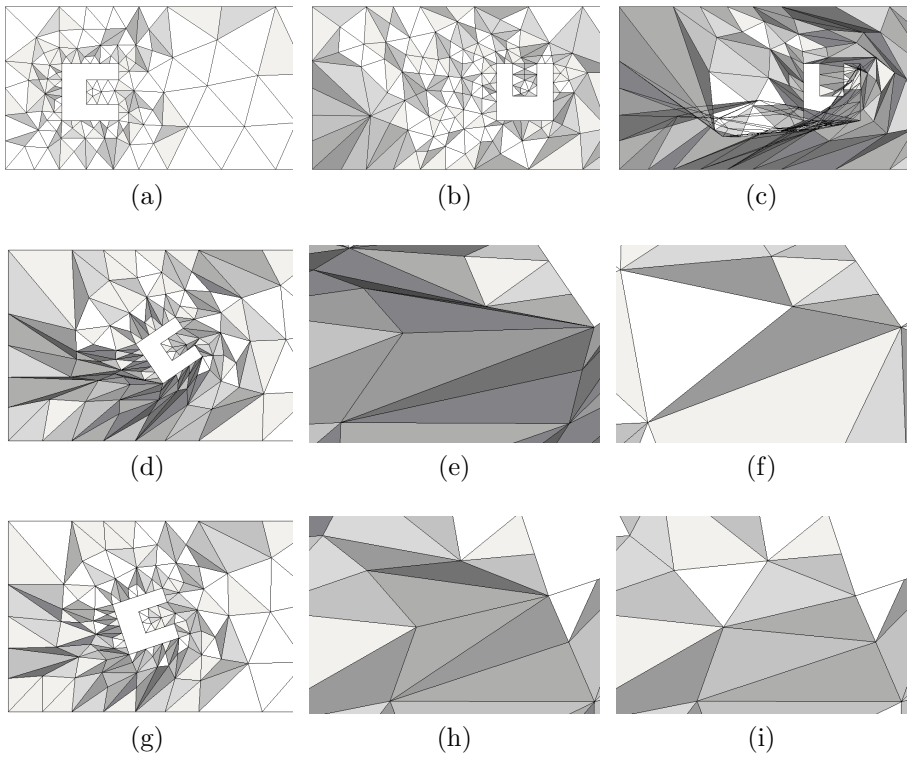


Figure IV.2: Simple 2D example of a deforming domain: an object is translated and rotated: (a) initial mesh, (b) next position of the object and mesh resulting from the proposed approach, (c) mesh if simple node repositioning is applied, (d) and (e) mesh just before a first element is returned, (f) same mesh after optimization by edge swaps, (g) and (h) mesh when a first element has a quality $Q < Q^*$, and (i) same mesh optimized.

A general consideration for the mesh adaptation procedures is that the operations reducing the number of nodes, typically the edge collapses, should be performed first, in order to avoid the memory peaks. The most costly operations, like those intended to improve elements quality, should be performed just after, so that the global time spent in the procedure is minimized.

Concerning the efficiency of the mesh modifications, the edge swap operations are usually responsible for more than half of the total computational time spent in the adaptation procedure (excluding the node repositioning step), when designed carefully. In [70], it is shown how the edge swap operator can be designed to minimize the number of element quality computations when evaluating a possible configuration.

Some authors choose the first configuration that enhances the minimal quality in the cavity. However, the mean number of element evaluations is not significantly lower than $n_t(p)$ and the resulting quality is not optimal. We also

recommend to set a threshold value for the minimal element quality around an edge above which the edge swap will not be evaluated.

As an example, consider the mesh around a tube and a cylinder undergoing an opposite displacement, as depicted in Figure IV.3. The full motion is divided

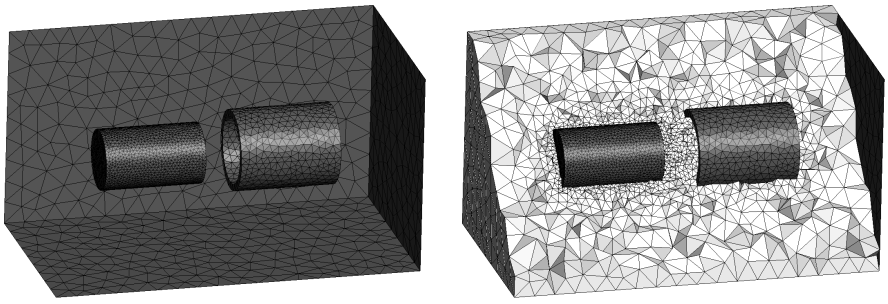


Figure IV.3: Cylinder and tube: initial geometry and mesh.

in 400 time steps in which the adaptation procedure is applied. The mesh after 100 time steps is shown in Figure IV.4. The mean time required per time step

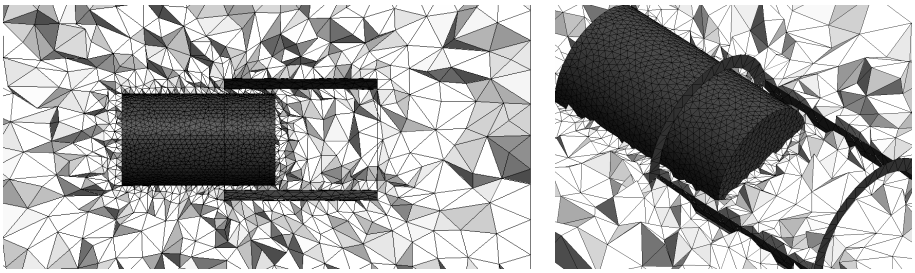
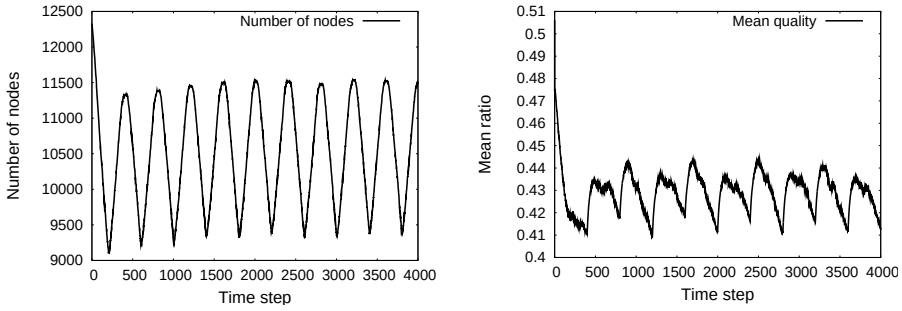


Figure IV.4: Cylinder and tube: mesh aspect after 100 time steps.

is 7.5 seconds for a number of nodes ranging in $[9.100, 12.300]$. In this window, 60% of the time is for the global node relocation (elastic analogy) and 40% is for the mesh adaptation by local modifications. The total time per cycle (forward and backward motion) is about 2900 seconds. The total numbers of split, collapse and swap operations performed are respectively 56682, 59537 and 94273, and the time spent in the different operators is 72.8, 137.1 and 742.5 seconds. Figure IV.5 shows the evolution of the number of nodes and mean element quality if 10 cycles are performed.

For measuring the quality of the shape of elements, various element shape measures are available in the literature [134, 109]. Here, we choose to use the mean ratio [108] for its relatively small computational cost.

Finally, we notice that the storage of elements quality requires only a small memory space but saves a significant computational time. The storage is made in the following way. When the quality of an element is required, the quality



(a) (b)

Figure IV.5: Cylinder and tube: evolution of (a) the number of nodes and (b) the mean element quality over a computation of 10 cycles.

previously computed is returned if available. If not, it is computed and stored. When a mesh modification is applied, the quality measures of the modified elements are deleted. As an illustration, we observe that the time required for the tube and cylinder test increases of 65% for a difference of $2Mb$ of memory if the element qualities are not stored.

IV.3 Mesh Database

When it comes to the implementation of a mesh adaptation procedure, issues associated to data structures have to be addressed. Describing an unstructured mesh on a computer can be done in various ways, depending on the mesh adjacencies that have to be accessed by algorithms.

We deal here essentially with tetrahedra. In a tetrahedral mesh with N_v vertices, there are about $6N_v$ tetrahedra, $12N_v$ triangular faces and $7N_v$ edges. Concerning upward adjacencies, there are about 14 edges adjacent to a vertex, 5 faces adjacent to an edge and 2 tetrahedra adjacent to a face. Storing all possible adjacencies cannot be envisaged for obvious reasons of memory. Different solution have been proposed in the past. Some were using a predefined static set of adjacencies [20], some others were using a dynamic set of adjacencies [153].

In this work, we have used the bi-directional data structure [153] for our mesh database. Every tetrahedra knows about his four triangular faces, every face knows both its adjacent tetrahedra and its three edges, every edge knows about its adjacent faces and its two ending vertices and every vertex knows all its adjacent edges and its coordinates.

In term of implementation, any adjacency requires to store the address of the adjacent entity. The bi-directional data structure that is used in MAdLib requires therefore

$$N_v(14) + 7N_v(2+5) + 12N_v(3+2) + 6N_v(4) = 147N_v$$

addresses (or pointers). In comparison, the usual element-to-node data structure requires

$$N_v + 6N_v(4) = 25N_v$$

addresses, which is way smaller. Yet, a simple element-to-node data structure is not rich enough to allow general local mesh modifications. Another common data structure that is used in mesh generation [77, 59] is an enriched element-to-node data structure where every tetrahedron knows about its four adjacent tetrahedra. In this case, the theoretical number of addresses required is

$$N_v + 6N_v(4 + 4) = 49N_v.$$

The enriched element-to-node data structure is one of the lightest that enables to do local mesh modifications. Yet, it does not allow to modify the discretization of both model surfaces and model edges in 3D.

In this work, we aim at providing algorithms that allows to perform mesh adaptation in transient computations. Therefore, allowing to adapt both volume and surface meshes at the same time is mandatory. In forthcoming papers, we will show that this approach also enables to do mesh adaptation that complies with the exact geometry of the model.

Our data structure requires to store all vertices, all edges, all faces and all tetrahedra of the mesh. Mesh adaptation procedures require to add and remove mesh entities form the mesh entity containers. In a previous approach [153], we used hash tables as mesh entity containers. In principle, removing or adding operations can be performed in constant time. We have found out that this approach was not optimal, both in terms of computational efficiency and of memory usage. We use now simple linked lists (`std::list` of the standard template library [128]) to store mesh entities because those containers provide some key advantages:

- they allow fast iteration (linear time) over the container,
- they allow fast insertion and deletion (constant time), as there is no sorting and no reallocation of the structure.

The main drawbacks of lists are that they don't allow random access, and, more important they do not allow fast search operations. In other words, removing an element of the list is only possible when the iterator is positioned on the element to remove. Consider the edge swap operation. We typically iterate on all edges of the mesh and check if the edge swap improves the quality of the elements surrounding the edge. Building the cavity around the edge is trivial with the data structure because edges know about their neighboring faces and faces know about their neighboring tetrahedra. Removing all those faces and tetrahedra cannot be done in constant time. In order to take full advantage of the lists (fast iteration, insertion and deletion), without needing random access, we have defined the following behaviors:

1. When an entity is created, it is added to the end of the list.
2. When an entity is removed from the mesh, it is not actually removed from the memory, as we do not necessarily have an iterator pointing to its location in the list. It is rather marked as a "dead" element.

3. Iterators on lists have been modified so that, when an iterator points to an entity that is marked as dead, the element that is pointed is actually removed from the memory. The iterator goes therefore to the next element of the list until it finds either the end of the list or a non dead element.

Those behaviors are illustrated on Figure IV.6.

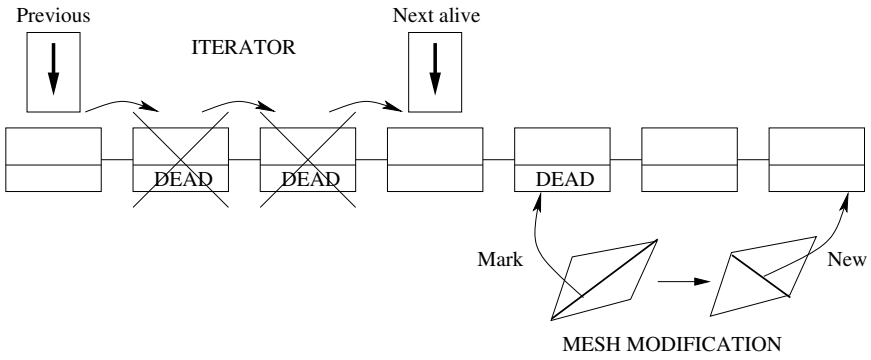


Figure IV.6: Behaviors of the list iterator and mesh modification operators about element creation, deletion and access.

The presented mesh database has already been integrated successfully in several codes. Its implementation in MAdLib gives the following results in terms of memory: $2.66\,Gb$ for one million nodes, which represents typically 6 millions tetrahedra, and $650\,Mb$ for one million nodes in 2D. Table IV.1 compares the memory usage of different mesh databases: MAdLib for both mesh loading and mesh adaptation (without nodes repositioning) and Gmsh for mesh loading and mesh generation.

Table IV.1: Memory consumption for different softwares: Gmsh and MAdLib, for tetrahedral meshes. The mesh database of Gmsh has been tested for two different configurations: simple mesh loading and mesh generation.

	Gmsh (read)	Gmsh (generation)	MAdLib
Memory per million nodes (Mb)	485	970	2660

IV.4 Adaptation within any physics solver

It is well known that mesh adaptivity is a specific field in the domain of computational mechanics. The issues encountered are often treated separately from solving the PDEs. The mesh adaptation functionalities are generally gathered in a separate module, or included from an external package, leading to an isolated framework.

As a consequence, integrating a mesh adaptation procedure in a PDE solver has become a technical challenge in addition to the theoretical issues of mesh adaptivity. In order for a mesh adaptation package to be usable, it is necessary to rely on an efficient interface to the physics solvers, and to provide generic means to project the solution from the initial mesh to the adapted one.

The integration of a mesh adaptation procedure in a PDE solver also raises the issue of the evolution of the performances in terms of CPU time and memory requirements.

In this section, we propose solutions to the different problems faced when integrating a mesh adaptation procedure in a computation. The callback function, which is the support for the projection of the solution is first described. The general scheme to achieve the coupling is then discussed.

Solution transfer

The projection issues are crucial when a solver uses a mesh adaptation technique. The projection algorithms have to be carefully designed in order to avoid the propagation of important errors. However, this design is highly dependent on the numerical method used by the solver and it would not make any sense to build a generic projection method in a mesh adaptation library. For that reason, we propose a flexible framework allowing to build any projection method and interfacing it to a mesh adaptation tool.

The physics solver has its own data structure for the solution or any other data, which is often static in the sense that it is not designed to allocate or deallocate parts of the solution. The method we propose is the following. Before starting the adaptation procedure, the static data has to be turned into a dynamic one. In order to do that, we extend the mesh database presented in §IV.3 by adding the possibility to store a set of informations within an object representing a mesh entity, which allows to *attach* a part of the solution to its corresponding mesh entity (vertex, edge, face or tetrahedron), as illustrated in Figure IV.7(a). The static data structures can thereafter be deleted. As a consequence, the data allocation or deallocation associated to a mesh modification can simply be performed at the same time as the modification itself, as depicted in Figure IV.7(b).

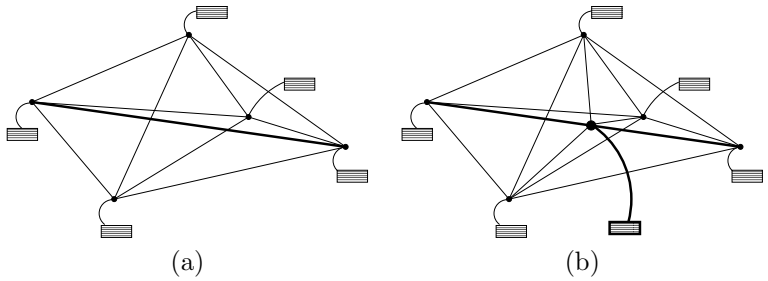


Figure IV.7: Dynamic data structure: (a) initial cavity with data attached to each node, (b) cavity after an edge split.

In order to set the variables to appropriate values when the mesh is modified, projections have to be performed. While the adaption is running, a set of functions registered by the physics solver are called every time a mesh modification occurs. These functions, named *callback functions*, are defined in the solver according to standard specifications that can be given by the interface of the mesh adaptation library.

We propose the following standard specification for the callback functions: they perform the projection on any cavity, including allocating/deallocating the data contained in the mesh entities of the cavity, and receive the following arguments:

- the cavity to be deleted,
- the new cavity,
- the type of operation,
- an identifier giving access to the attached data,
- a mesh entity that could be directly accessed depending on the type of operation, like the new node in the case of an edge split for instance.

The user of a mesh adaptation library implementing this framework is then totally free in the definition and registration of these functions, which allows him to design an appropriate projection scheme for every type of data.

Another option available with the callback functions is to use them to build a list of modified mesh entities without doing any projection, thus allowing the solver to perform all the projections at once on the modified elements at the end of the adaptation procedure. Note that in that case, the initial mesh and solution have to be kept along the mesh adaptation procedure.

Several examples of callback functions are available in the MAdLib [46] and FEniCS [68] open source packages.

Interfacing a mesh adaptation library

From our experiments, a mesh adaptation procedure relying on a dynamic mesh database as presented in §IV.3 can be integrated in the global stepping of any physics solver by applying the following sequence (see Figure IV.8):

1. deallocate the solver data built from the solution and/or the mesh,
2. build the dynamic mesh database from the solver mesh,
3. build the dynamic data structure from the solver solution (see §IV.4) and delete the solver solution,
4. delete the solver mesh,
5. create the size field,
6. run the adaptation procedure,
7. build a new mesh in the solver from the dynamic mesh database,

8. build the solver data structure for the solution,
9. delete the dynamic mesh and solution,
10. build the solver data from the solution and/or the mesh.

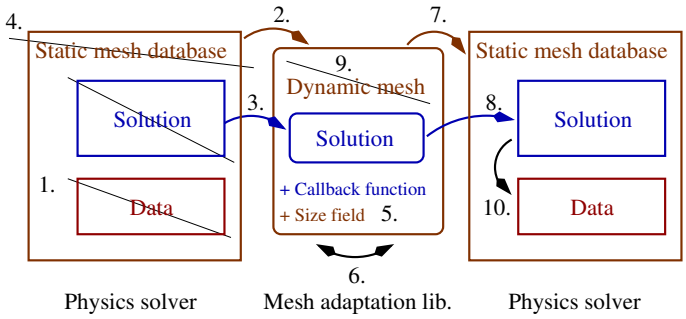


Figure IV.8: The general scheme for integrating a mesh adaptation procedure in any physics solver.

The proposed sequencing of the allocation and deletion of the data, solutions and mesh structures before and after the adaptation is designed to avoid memory peaks. The position of the deallocation and reallocation of the solver data in particular (resp. first and last steps) generally avoids any memory peak during the global scheme.

We notice that the number of operations is relatively small, which enables the writing of a compact interface to a mesh adaptation library. A minimal interface can be built with the following functionalities:

- Generic specification of the callback function.
- Construction/destruction of mesh entities.
- Allocation/deallocation of attached data, and functions to get or modify it.
- Construction/destruction of a size field, and function to specify a size at a vertex.
- Call to the adaptation procedure.

As an example, the MAdLib package [46] and the FEniCS project [68] are interfaced together with the method described here.

IV.5 Computational results

Several examples are shown to demonstrate the applicability of the proposed techniques when very large displacements or deformations occur. The quality

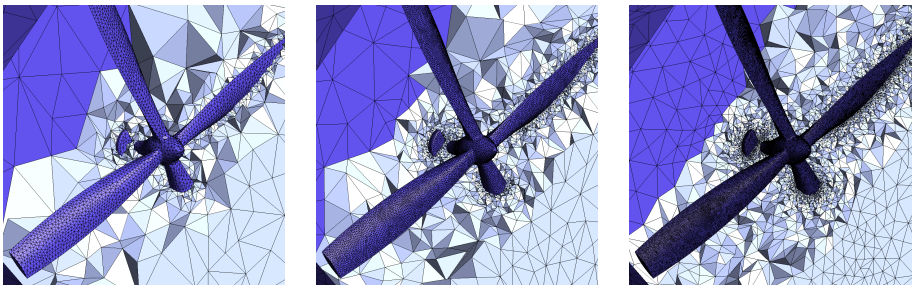
of the mesh and the requirements of the mesh adaptation procedure (computational time and memory) are illustrated in the first example. The second example is a simple fluid-structure test in which the structural parts undergo large displacements. The relative cost of the mesh adaptation in the fluid-structure problem is investigated. In the last example, the techniques developed in this paper are applied to a complex FSI problem with large deformations in order to illustrate the applicability of the technique and discuss the pertinence of the mesh adaptation and node repositioning techniques.

The computational times have been obtained with a single processor Intel(R) Core(TM)2 Duo CPU E6850 at a frequency of 3.00GHz. The mesh adaptation package is MAdLib 1.2.3 and the different codes were compiled with the GNU GCC 4.3 compiler.

Propeller

In this test, large displacements are imposed to a domain: a propeller rotates around its axis. For the current example, the propeller is enclosed in a cubic box. The rotation period T is set to 1 and the time step is chosen as $\Delta T = 0.001\,T$. The objective of this test is twofold: (i) to provide tangible informations about the CPU time and memory requirements for the MAdLib library, thus giving clues to evaluate the techniques presented in this paper in terms of computational performances, and (ii) analyzing the elements quality obtained when different quality threshold are given by the solver side, i.e. when the mesh has to be adapted more or less frequently.

Computational requirements In order to evaluate the performances of MAdLib with different mesh sizes, we first analyze some runs with various sets of size fields. The aspect of the meshes obtained for some size fields is depicted on Figure IV.9, while the meshes obtained at different time steps with a medium size ($\approx$ 55.000 odes) are shown on Figure IV.10.

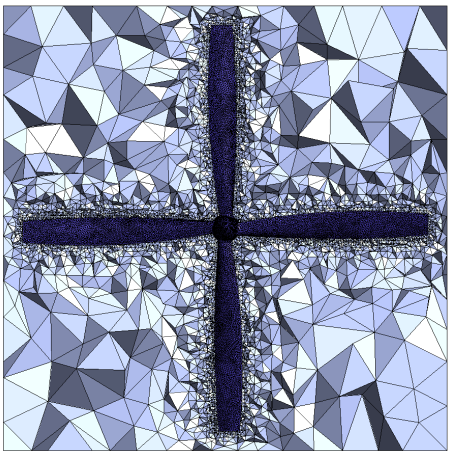


10.921 nodes 55.481 nodes 197.092 nodes

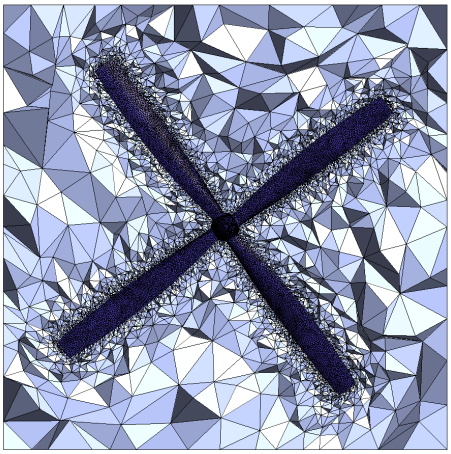
Figure IV.9: Propeller test: meshes obtained after 20 time steps for different sets of size fields.

Figure IV.11 shows the different CPU time and memory consumptions obtained. Note that for this example, the whole mesh is used in the node repositioning thus leading to a huge linear system. We observe that the memory

$t = 0.0$



$t = 1.6$



$t = 1.7$

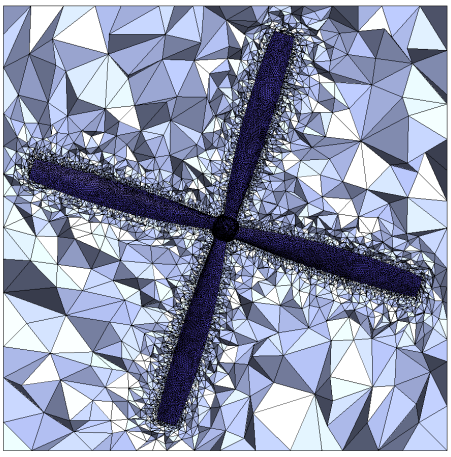


Figure IV.10: Propeller test: meshes obtained at different time steps. The number of nodes is approximately 55.000. At times 1.6 and 1.7, the propeller is in its second revolution.

is maximal during the node repositioning procedure. The amount of memory required including node repositioning tends to 5200 *bytes* per node, while the pure mesh adaptation costs 2900 *bytes* per node.

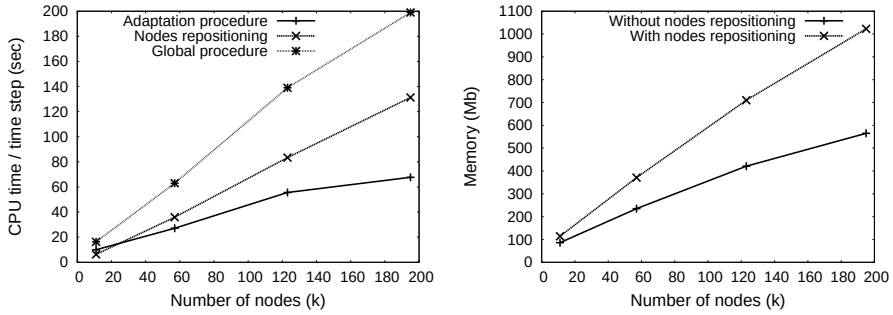


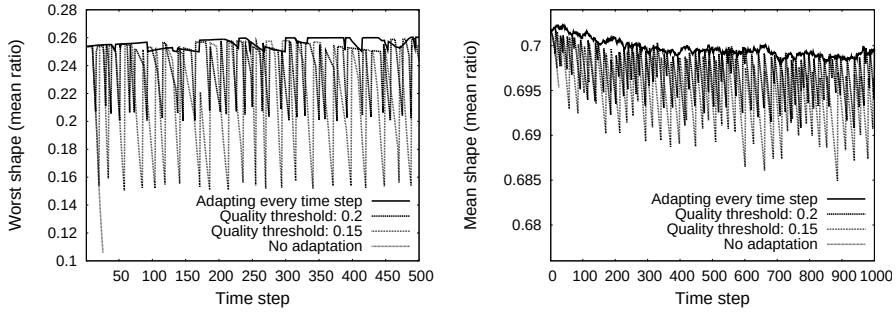
Figure IV.11: Propeller test: CPU time and memory required for the different mesh sizes.

Element quality We are now interested in the evolution of the element quality. One can decide to reposition the nodes and to adapt the mesh every time the boundaries of the domain are moved, or to fix a threshold value Q_{ad} for the minimum element quality over which the nodes are repositioned and no adaptation is performed. If an element has a quality below Q_{ad}, the node repositioning is followed by mesh adaptation.

In Figure IV.12, the worst and mean qualities are shown for several approaches: node repositioning only ($Q_{ad} = 0$), quality control by node repositioning and mesh adaptation with different values for Q_{ad}, and node repositioning with systematic mesh adaptation ($Q_{ad} = 1$). The initial mesh has around $15,000$ nodes. Note that the sliver quality threshold Q_{sl} is set to 0.25, and the quality threshold inside the node repositioning algorithm Q^* is set to 0.1, except in the case $Q_{ad} = 0$ in which no local mesh modification is performed.

We observe that adapting the mesh at each iteration yields results with a better quality, both for the mean and the worst qualities. On the other hand the mean CPU time per time step is around $23.4\,s$ in that case, while it is only around $15.5\,s$ and $14.4\,s$ for $Q_{ad} = 0.2$ and $Q_{ad} = 0.15$ respectively. Table IV.2 shows the number of mesh modifications performed during a complete rotation period with the different approaches. We observe that more mesh modifications are applied when the mesh is adapted more frequently. This is another drawback of the approach which consists in adapting the mesh at every time step.

When no adaptation is applied, the quality becomes very low in the first time steps and an element is returned at time step 116, corresponding to a rotation of about 40°.



(a) (b)

Figure IV.12: Propeller test: worst (a) and mean (b) element qualities for different approaches. The worst quality is depicted for the first 500 time steps (half period) for clarity, while the mean quality is shown for 1 complete period in order to observe the global evolution.

Table IV.2: Propeller test: number of mesh modifications performed during the first period.

Q_{ad}	Edge splits	Edge collapses	Edge swaps
0 (No adaptation)	0	0	0
0.15	18608	19654	68702
0.2	18888	20005	70521
1 (Systematic)	20645	21800	75152

Spheres falling in a fluid

In this test case, a simple FSI computation involving large displacements is shown. We consider two rigid spheres vertically aligned and immersed in a fluid initially at rest. The density of the spheres ρ_s and the fluid ρ_f are chosen such that $\rho_s > \rho_f$. At $t = 0$, the spheres accelerate downward under the effects of gravity. As the velocities of the spheres increase, a low pressure zone appears between them which results in an attraction between the spheres. Eventually, after some time, the two spheres touch.

The fluid is governed by the incompressible ALE Navier-Stokes equations while the displacements of the spheres are obtained by the second Newton's law. The densities ρ_s, ρ_f, the dynamic viscosity of the fluid μ_f and the diameter of the spheres D are chosen as:

$$\rho_s = 1.5, \quad \rho_f = 1.0, \quad \mu_f = 0.025, \quad D = 1.0.$$

The initial distance between the spheres centers is $4D$. With these values, the Reynolds number of the flow around the spheres ranges from 0 to ≈ 100 until the spheres touch.

The Navier-Stokes equations are solved with Argo (Cenaero, Belgium), a finite element solver with a P1/P1 interpolation of velocity and pressure, stabilized with a finite volume computation of the convective fluxes. A pressure stabilization of type PSPG for the incompressible flows is also used. The nonlinear problem is solved with a Newton method, and the linear system is solved by a GMRES solver with an ILU preconditioner (fill in: 2). The coupling is achieved with a conventional staggered scheme.

The callback function called by the mesh adaptation procedure is very simple:

- When an edge is split: interpolate linearly the velocity and pressure on the edge to fix the variables at the new vertex.
- When an edge is collapsed: remove the solution attached to the deleted node.
- When an edge is swapped: do nothing.
- When an face is swapped: do nothing.
- When a vertex is moved: do nothing (the motion is automatically included in the computation of the mesh velocity for the ALE fluxes).

The following sizes are prescribed for the edges:

- a maximum size of 4.0 is imposed over the whole domain,
- two size fields depending on the distance $d(\mathbf{x})$ to the walls are defined for every sphere with to the following parameters (identical for both spheres):

$$\delta_1(\mathbf{x}) = 0.05 + 0.15\, d(\mathbf{x}), \qquad \delta_2(\mathbf{x}) = 0.2 + 3.8\, d(\mathbf{x}).$$

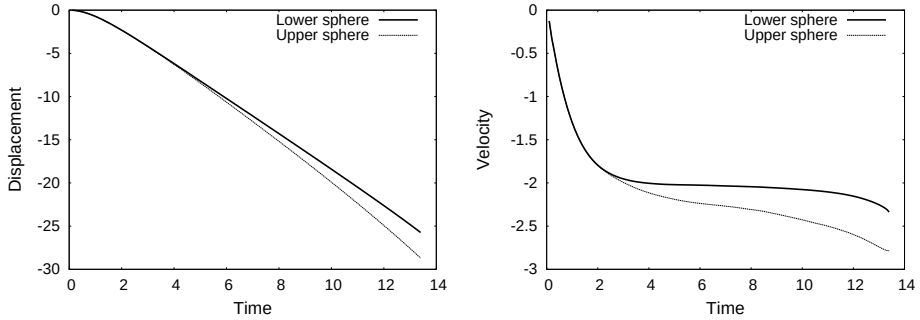
The resulting meshes have a number of nodes ranging from 26800 to 28500.

The displacements and velocities of the spheres are shown on Figure IV.13. We observe an increasing difference in velocity between the spheres from time 2.4. The computation is stopped at time 13.5, when the spheres touch.

Figure IV.14 shows snapshots of the mesh and the pressure in the fluid at different time steps. The mesh is adapted after every time step.

Concerning the performance of the adaptation procedure, we make the following observations.

- There is no memory peak in the adaptation procedure. The total amount of memory required is maximal (500-510 Mb) when the memory for the GMRES solver of Argo has been allocated. The memory required during the adaptation procedure reaches a maximum at 460-470 Mb. The mesh database and the adaptation data (size fields, elements quality, ...) consumes 80 to 85 Mb.
- The computational time for the full adaptation procedure described in §IV.4 represents 25.8% of the total time. We can subdivide the adaptation time in three parts:



(a) (b)

Figure IV.13: Spheres test: displacements (a) and velocities (b) of the spheres.

1. data transfer to the dynamic structures (steps 1 to 5): $\approx 5.1\%$,
2. mesh adaptation (step 6): $\approx 11.7\%$,
3. data transfer to the solver (steps 7 to 10): $\approx 9.0\%$.

The difference between the first and third parts comes from the solver data built from the solution and/or the mesh. These data has to be computed at step 10 of the procedure while it is deleted in step 1. We observe that the deallocation and reallocation of data (solution, meshes, other data) is responsible for less than 15% of the global time, which is reasonably small compared to the advantages of adapting on a specific mesh database.

Fluid-structure interaction: 3D turbulent flag

In [88] we describe a Unified Continuum (UC) model in Euler coordinates with a moving mesh for tracking a fluid-structure interface as part of the General Galerkin (G2) discretization. The UC model is implemented in the free software/open source Unicorn component, which is part of the FEniCS project [68]. We have extended the Unicorn implementation with an interface to the MAdLib library, and here show results for a test problem.

The test problem consists of a flexible flag mounted behind a fixed cube in turbulent flow and exhibits complex 3D behavior with torsion, large structure deformations and highly fluctuating flow. We choose an inflow velocity of $1 \cdot 10^2$ m/s, a cube of $1 \cdot 10^{-1}$ m side and a flag mounted at the top of the back face of the cube with a length of $3 \cdot 10^{-1}$ m and a thickness of $5 \cdot 10^{-2}$ m. The viscosity of the fluid is $1 \cdot 10^{-4}$ Pa s (density $\rho = 1$) which gives a representative Reynold's number $Re = 1 \cdot 10^5$. Note that the G2 method (used in Unicorn) is a LEStype method without full resolution of all physical scales in the flow, where the numerical stabilization of the method acts as a subgrid model, see [89] for details. For simplicity we here assume all boundary layers to be laminar, which is modeled by a no slip (zero velocity) boundary condition. Since the main purpose of this example is to demonstrate robustness we have not performed convergence studies or applied error control.

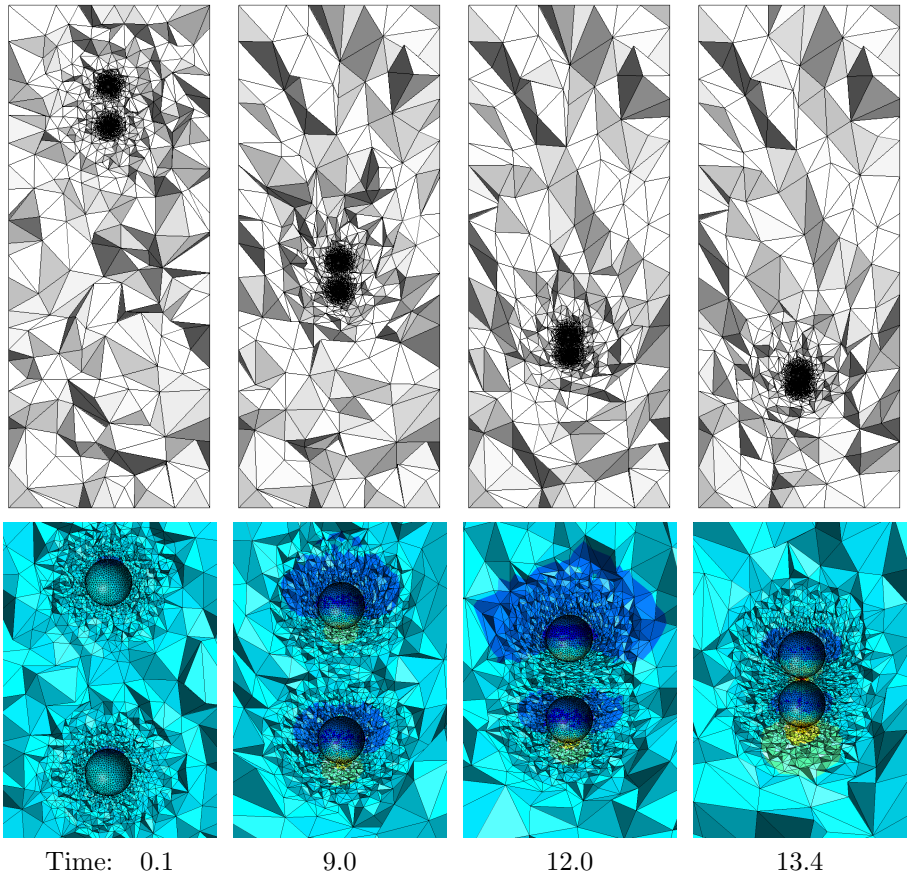
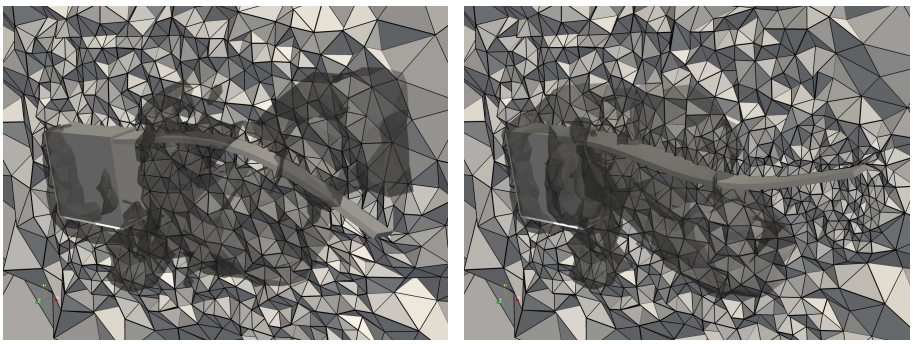


Figure IV.14: Spheres test: snapshots of the mesh at different time steps. The close views of the mesh show the pressure distribution.

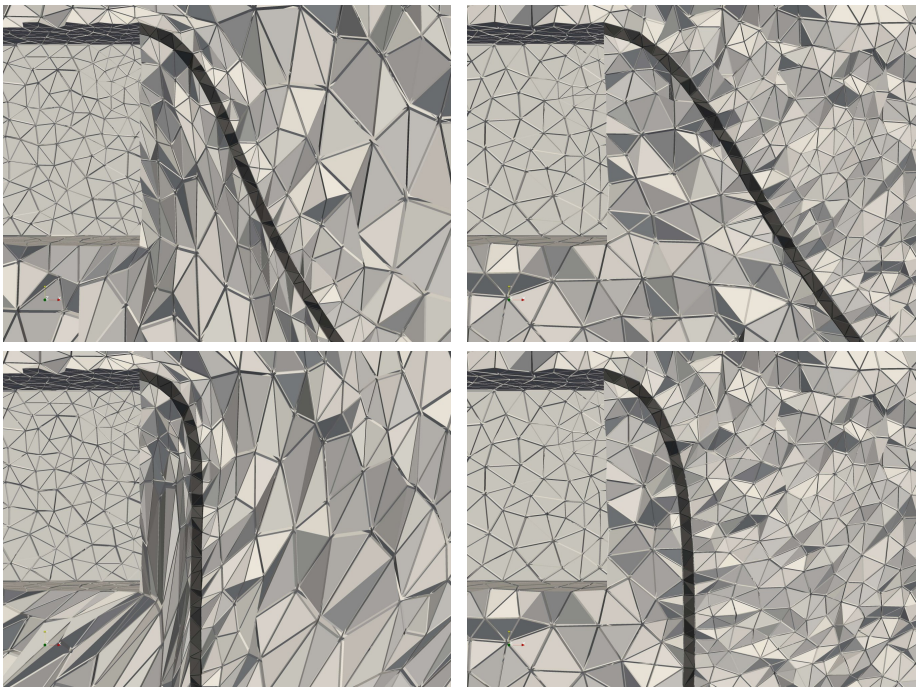
The deformation of the flag induced by the flow in the simulation is large. In [88], the resulting mesh deformation is handled by an elastic smoothing method which is sufficiently robust for this case, but in general cannot guarantee control of cell size and shape, and which can be costly. It would thus be desirable to be able to use mesh adaptivity to avoid being limited in the choice of problems by the mesh smoothing method.

We simulate the flag problem with (a) pure elastic smoothing and (b) mesh adaptation together with elastic smoothing acting as quality optimization on cells falling below a quality threshold. A representative snapshot of the flag motion is given in Figure IV.15, with a magnification of the flag tip in Figure IV.18 illustrating the mesh behavior with the different methods. We verify that the methods give roughly the same amplitude and frequency of oscillation by plotting the y-coordinate of the top-right-closest corner of the flag in time in Figure IV.17, where we can see that this is the case. We also perform a robustness test with zero inflow and a downward force on the flag causing the flag



(a) (b)

Figure IV.15: Flag simulation: snapshots with (a) elastic smoothing and (b) mesh adaptation. A cut of the mesh is shown together with an isosurface of the pressure to visualize the flow.



(a) (b)

Figure IV.16: Flag simulation: robustness test with (a) elastic smoothing and (b) mesh adaptation. Note the badly shaped cells squeezed between the cube and flag.

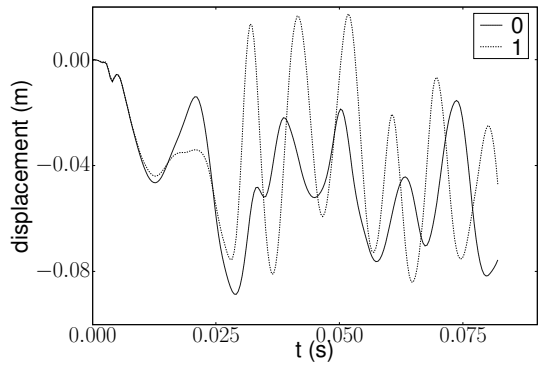
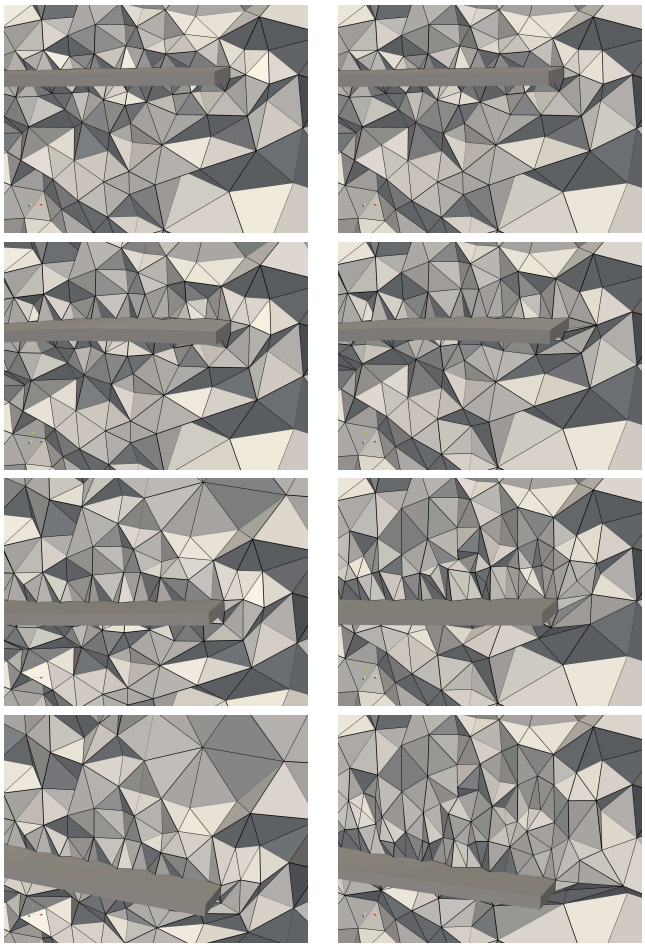


Figure IV.17: Flag simulation: plot of y-coordinate of displacement of top-right-closest corner of the flag for (0) elastic smoothing and (1) mesh adaptation.

to hang straight down from the cube. The mesh adaptation maintains good cell quality and satisfies the prescribed tolerance interval for $\delta(\mathbf{x}, t)$, while the elastic smoothing gives very distorted cells between the flag and cube. An even clearer robustness test would be a rotating cube where the elastic smoothing would fail completely.

We conclude that the mesh adaptation gives qualitatively similar solutions to mesh smoothing (pure motion of the mesh nodes), while giving a far superior control of cell size and quality, meaning that the presented mesh adaptation algorithm could replace or extend mesh smoothing implementations in fluid-structure solvers. We speculate that the optimal mesh motion algorithm should be a hybrid with a cheap mesh smoothing acting by default and with mesh adaptation being applied when the prescribed size field and cell quality is no longer satisfied. Since a cheap mesh smoothing algorithm should be less costly than pure mesh adaptation the hybrid should thus be far superior in generality and size and quality control to pure mesh smoothing and superior in cost and number of projections to pure mesh adaptation.

In future work we aim to apply the mesh adaptivity not only to mesh motion, but also in an adaptive error control algorithm. This would further show the advantages of having general mesh adaptation where we would then have full control of the size field in space and time to adapt the mesh to satisfy a tolerance on some output of the error, drag for example (extending [89]). We also aim to further study the behavior of hybrid algorithms combining elastic mesh smoothing (quality optimization) with mesh adaptation.



(a) (b)

Figure IV.18: Flag simulation: starting phase, zoomed in on tip, with (a) elastic smoothing and (b) mesh adaptation. Note that in (a) the surrounding cells are dragged along and deformed with the flag while in (b) the surrounding cells are fixed and only cells close to the flag are modified.

IV.6 Conclusion

In this paper, we have described a new node repositioning algorithm that combines a standard global node repositioning with local mesh modifications. The global procedure combines this technique with mesh modifications that are only applied when the quality of the mesh becomes insufficient. We have shown that the method and its implementation

- enables to deal with very large motions of the structure in FSI problems,
- enables to deal with domains in which the meshed volume varies,

- enables to control both the quality and the size of the elements.

In addition, we have shown that local mesh modifications do not introduce excessive amount of numerical dissipation, thanks to their locality.

We also presented a new open source library, MAdLib, that provides tools and algorithms to transiently adapt tetrahedral meshes. The use of MAdLib enables to perform computations involving large deformations or displacements of the boundaries.

A particular attention has been paid to the effectiveness of the implementation regarding its CPU time and memory consumptions. We have shown that the mesh adaptation procedure was not expensive when inserted in a state-of-the-art 3D incompressible fluid solver and that no memory peak occurred.

Both the simulation code [68] used for the flag test case and the adaptation library [46] are available as open-source codes. In our experience in the meshing community, we have seen that the effectiveness of mesh adaptation procedures were very sensitive to their implementation. We hope that making this source code available will enable to build a community around it, as we already did for mesh generation [77].

In future works, mesh adaptivity will be also applied to control the error of FSI computations. A hybrid strategy between the smoothing technique presented in [88] and the method presented here will also be investigated. Finally, we will use the technique to snap vertices to real geometries while doing mesh adaptation.

Acknowledgements

The authors would like to thank the Center for Excellence in Aeronautics (Cenaero, Belgium) for providing Argo, the Navier-Stokes solver used to solve the spheres test case.

The last author would like to acknowledge the financial support from The Swedish Foundation for Strategic Research and the European Research Council.

5.1 Programming interface of MAdLib

The source code of a library includes at least two important components: the definition of a public interface, and its implementation.

A good **interface** allows the user to easily take advantage of the full potential of the library. Its definition is therefore important. Important are also the means provided by the developers to the user to help him using it (manual, tutorial, examples, ...)

The **implementation** is a time-consuming process that needs to be done carefully in order to end with a library which is efficient, robust, portable and easy to maintain.

The key points of the implementation are reached in Article IV and are not further described here. In this section, we describe the specific aspects related to the interface of the library and the possible connection with a physics solver.

5.1.1 Interface of MAdLib

The MAdLib package is organized around four modules. These modules have a particular role to play and a related interface. These interfaces form the global interface of MAdLib. We describe here below every module:

- The **Tools** module contains the mathematical (vectors, matrices, metrics, ...) and general tools (error handling, CPU time and memory management, expression evaluation, ...) that are used by the other parts of the code. It is the only internal module (no public interface).
- The **Geometry** module contains a minimal geometrical model and the interface to the geometrical module of Gmsh. The geometrical entities are stored in this module. The interface contains the functions to manipulate the geometrical model (model I/O, iterators over model entities, ...) and the geometrical entities (entity closure, parametrization, curvature computation, ...).
- The **Mesh** module is intended to store the mesh: entities, connectivity, coordinates, classification on model entities, ... The interface includes all the functions related to the mesh (iterators over mesh entities, mesh I/O, ...), mesh entities (adjacencies questioning, geometrical computations, ...), attached data and validity checks.
- The **Adaptation** module contains all the components of the mesh adaptation: size fields, mesh modification operators, node repositioning, sliver elimination procedures, quality control tools, various outputs, ... The interface contains functions to modify a mesh in various ways (refinement, coarsening, element quality optimization, node repositioning, Laplace smoothing, ...), the global procedures proposed in the present work, and function to adjust the value of some parameters (tolerance interval for adimensional edge lengths, sliver quality threshold, ...).

5.1.2 Connection with a model solver

In §IV.4 (Article IV), an algorithm is proposed to embed the mesh adaptation method in a physics solver. We describe here an example implementation of the algorithm in order to provide a connection between the concepts presented in the paper and the interface of the code. In order to illustrate the present section, a tutorial module is provided in MAdLib and annotated C++ classes are reproduced from it in Appendix B. We also refer to the open source FEniCS project [68] for an example of a coupling between a model solver and MAdLib. The flag test case presented in §IV.5 was obtained by this coupling.

Consider a solver $\mathcal{S}$ which aim is to solve a partial differential equation by a finite element or finite volume method. The goal here is to embed the mesh adaptation procedure of MAdLib in the routines of $\mathcal{S}$. In order to fix the things, we assume that $\mathcal{S}$ can be abstracted by the classes shown in Appendix B.2. These classes contain basic functions as accesses to the mesh elements, nodal coordinates or solution, allocation and deallocation routines for the mesh, solution and other data, ...

In the solver implementation, we add the class laid out in Appendix B.1 which contains all routines dependent on both the solver implementation and MAdLib interface. The functions appearing in the class are:

- The global adaptation routine, which is the only public routine of the class. Its implementation is given by Appendix B.3 and corresponds to the procedure presented in § IV.4.
- The mesh and geometrical model conversions between the databases of MAdLib and $\mathcal{S}$.
- The construction of the mesh size field. The implementation will differ according to the objectives sought by using mesh adaptation: error and/or mesh size control, discontinuities capturing, boundary layer meshing, ...
- The solution transfer routines between the data structures of MAdLib and $\mathcal{S}$.

The mesh of $\mathcal{S}$ can then be adapted by simply calling the adaptation routine `adaptMesh()` when needed.

A typical implementation of a callback function is also given in Appendix B.4. It is specified as a callback function to the main adaptation class in the global adaptation routine presented in Appendix B.3.

Conclusions

Short summary

In the present work new techniques have been proposed to handle large deformations of the domain with local mesh modifications in a robust way. The basis of the work is a mesh adaptation method which relies on a set of local mesh modifications applied in a well defined order. Starting from an initial mesh, the aim of the method is twofold: produce elements with a desired size, and maintain elements quality to a desired level. In the present work, the method has been applied to two-phase flows with an interface capturing method based on a level set function. The method has been equipped with techniques to generate and adapt anisotropic and unstructured boundary layers meshes. The preservation of the compatibility of the mesh to a CAD model in the framework of the method has also been addressed. The different achievements presented in this work have been implemented in the open source library MAdLib.

Contributions, perspectives and future works

We give here more details about the different contributions of the present work and try to bring out new trails for future developments.

The first objective of this work was the handling of **arbitrarily large deformations**. In this framework, a new **sliver elimination** procedure has been presented. This procedure is a key stone of the method since the robustness of the node repositioning is highly dependent on the presence or not of low quality elements. Although more efficient and robust than the previous techniques, this sliver elimination procedure is not proved to (and do not) eliminate 100% of the slivers. Solutions involving local re-meshing could be investigated to further improve the robustness of the sliver elimination procedure.

A **repositioning procedure** that combines the classical node repositioning techniques with edge and face swaps has been proposed. This technique enables very large deformations when no intermediate mesh adaptation is possible or desired. Classical finite element formulations allow to solve an arbitrary Lagrangian Eulerian formulation of the equations, and therefore take implicitly into account the motion of the mesh coming from the node repositioning. The technique proposed here could be fully integrated in a finite element formulation if an equivalent **ALE formulation** was found for the edge swap. Note

that a formulation for the face swap would be trivially deduced by the latter since a face swap is the reverse modification of a 3-2 edge swap. This new formulation is currently investigated by G. Olivier and F. Alauzet (INRIA, Paris), and promising results have already been presented for the 2D edge swap.

For the elastic analogy itself, a choice that was made here was to relate the stiffness of an element to its volume. This **alteration of the stiffness** greatly improves the robustness of the node repositioning since we observed that the smallest elements were always the first to be returned. Another option would be to relate the stiffness to the distance to the closest moving boundary. This option could be particularly interesting in the case of rotating objects since the mesh enclosed in the domain of rotation could move in a relatively rigid way. The best choice could lie in a combination of these two stiffness alterations.

Concerning the **handling of CAD models** in the adaptation procedure, a robust method has been proposed and complex three-dimensional tests were performed. The method involves a node repositioning procedure based on an elastic analogy around the nodes to be snapped. In order to preserve the shape of the boundaries, a zero displacement is imposed as a Dirichlet boundary condition for all nodes classified on a surface, line or point. A possible improvement of the method would be to allow the surface nodes to move inside their model face. For that purpose an elastic computation could also be performed in the surfaces. A simple way to formulate the underlying elastic problem is to solve the 2D elasticity equation in the parametric plane of the surface. Such a relocation would be particularly useful when new nodes are added on a model line.

About the **boundary layer meshes**, a method to automatically build anisotropic unstructured boundary layer meshes in a geometrically compatible way has been presented. Several possible investigations to improve the method were presented in the last section of Chapter 3. The proposed method to compute the **distance to walls** is based on heuristics and raises issues like the determination of the closest point on the boundary. Furthermore, the gradient of the the resulting distance is not continuous. An alternative is to compute a fictitious derivable distance function by solving a partial differential equation (PDE). A PDE allowing to compute this function at a reasonable cost is proposed in a work of S. Legrand et al. [102].

Obtaining an accurate representation of the normal and tangent directions to a set of walls, as well as their curvatures, is an important deal in the proposed method. Additionally, it is a generic problem that can have much more outlets than the only boundary layer meshing. The normal direction and curvature are given by **the gradient and Laplacian** of the distance function. With the present approach, the error produced in the computation of the distance gradient yields significant oscillations in the curvature field. Indeed, obtaining an accurate distance gradient has been shown to be important for the computation of the curvature. In particular obtaining at least a linear representation of the gradient would be interesting, but costly since at least second order polynomials should be used to represent the distance. Another way to improve the gradient and curvature computations is suggested by E. Marchandise in [117], who investigated linear and quadratic reconstructions of the distance gradient

(actually, a level set gradient in [117]) by a least-square method and showed convincing results. We aim at applying such a technique to the present method in a near future.

Another important improvement to be brought to the method is the computation of the principal curvatures of the wall and the underlying directions. In most of the cases, those curvatures differ and different mesh sizes have to be prescribed in the main directions.

Eventually, a method to automatically build semi-structured meshes could be derived from the presented approach. A cloud of aligned vertices could be generated from the normal and tangent fields obtained in the boundary layer regions. Such a method would be very attractive since it would combine the advantages of the semi-structured meshes (no spurious oscillations in the solution, applicability to high Reynolds number flows) with the generic formulation and automatic behavior of the present approach.

The last part of the present work deals with the **implementation** of the techniques summarized here above. A *M*esh *Ad*aptation *L*ibrary, MAdLib, has been published as an open source software. The performances of the library, and the topics related to its implementation, like interfacing the library with a solver, designing an appropriate mesh database and the general algorithm for the mesh to mesh projection of a solution, have been described. By publishing the source code, we hope to build a community around it, which should accelerate its development and feed the project with new expectations and ideas. MAdLib is also intended to fill a gap in the field of the open source softwares in computational mechanics since, up to our knowledge, no open source code for general adaptivity of tetrahedral meshes exists, while several open source softwares can be found for finite element computations and mesh generation.

In the future, the new techniques that improve the current mesh adaptation method should be implemented in the library. Already published methods could also be integrated in the code. In particular, a smoothing technique for the metric field [2], and adaptivity for high order meshes [162] would be useful.

An important extension to bring to the present method is its **parallelization**. In a previous work, C. Dobrzynski has been able to establish a parallel mesh adaptation procedure for fixed domains. Besides, this technique is already implemented in MAdLib. Other authors also proposed similar techniques like H. de Cougny and M. Shephard [55], J-F. Remacle et al. [151] and P. Cavallo et al. [38]. It would be interesting to develop a parallel extension of the techniques presented in this manuscript including the global node repositioning.

The usual bottleneck of the **mesh generation** techniques for **large meshes** are the memory requirements in serial and the design of appropriate algorithms in parallel. Parallel mesh adaptivity for fixed domains opens a perspective for parallel mesh generation, since an initial coarse mesh can be generated in serial, partitioned by a classical partitioning procedure, and adapted by the present method. Therefore, the remaining question is about the required memory in parallel when dealing with very large meshes (> 10 millions nodes). This issue is common for both mesh generation and adaptation. In the present work, we described a mesh database that results from a compromise between memory requirements and time consumption for the most common operations. The

memory per million nodes is about 2.6 Gb for tetrahedral meshes. Compared to mesh generation softwares, this amount is relatively large. For instance, Gmsh uses 0.9 Gb per million nodes. The resulting gain in terms of CPU time is interesting when doing frequent mesh adaptation steps in a computation. However, for adapting or generating large meshes, different choices could be made for designing the mesh database. In particular, using simple linked list for storing the entities instead of double linked lists, and avoiding the storage of the faces and their adjacencies are possibilities to be explored.

Finally, a mesh adaptation library is a tool, a package intended to be used by other softwares. For that reason, the real usefulness of MAdLib comes from its **integration in other softwares**. As it is discussed here, mesh adaptivity can be used for both adapting meshes during a computation and providing an initial mesh with particular properties. MAdLib has already been interfaced in several softwares, like the finite element library FEniCS [68], the fluid solver Argo (Cenaero[1]), and the ice model of the SLIM[2] project, but lots of other projects could take advantage of mesh adaptivity through MAdLib. In the future, we also aim at interfacing the library with the meshing module of Gmsh, which will allow for generating meshes in parallel and using the size fields and local mesh modifications to optimize the generated meshes and make it comply to some specified element size distribution.

[1]Center for Excellence in Aeronautics (Gosselies, Belgium), `http://www.cenaero.be/`.

[2]Second-generation Louvain-la-Neuve Ice-ocean Model (UCL, Louvain-la-Neuve, Belgium), `http://sites.uclouvain.be/slim/`.

Bibliography

[1] M. Ainsworth and J.T. Oden. *A Posteriori Error Estimation in Finite Element Analysis*. Wiley, New York, 2000.

[2] F. Alauzet. Size gradation control in anisotropic meshes. *Finite Elements in Analysis and Design*, 2009. In press.

[3] 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. *Journal of Computational Physics*, 222:592–623, 2007.

[4] R. Almeida, P. Feijoo, A. Galeao, C. Padra, and R. Silva. Adaptive finite element computational fluid dynamics using an anisotropic error estimator. *Computer Methods in Applied Mechanics and Engineering*, 182(3-4):379–400, 2000.

[5] J.S. Anagnostopoulos. A fast numerical method for flow analysis and blade design in centrifugal pump impellers. *Computers and Fluids*, 38(2):284–289, 2009.

[6] A. Anderson, X. Zheng, and V. Cristini. Adaptive unstructured volume remeshing part i: The method. *Journal of Computational Physics*, 208:616–625, 2005.

[7] Y. Andrillon. *Simulation d'ecoulements a surface libre par une methode de capture d'interface en formulation totalement couplee*. PhD thesis, Ecole centrale de Nantes et Universite de Nantes, 2004.

[8] J.F. Antaki, O. Ghattas, G.W. Burgreen, and B. He. Computational flow optimization of rotary blood pump components. *Artificial Organs*, 19(7):608–615, 2008.

[9] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for Approximate Nearest Neighbor searching. *Journal of the ACM*, 45:891–923, 1998.

[10] I. Babuska. The finite element method with Lagrangian multipliers. *Numerische Mathematik*, 20:179–192, 1971.

[11] I. Babuska, J. Chandra, and J.E. Flaherty. *Adaptive computational methods for partial differential equations*. SIAM, 1983.

[12] T.J. Baker. Mesh adaptation strategies for problems in fluid dynamics. *Finite Elements in Analysis and Design*, 25:243–273, 1997.

[13] T.J. Baker. Mesh movement and metamorphosis. *Engineering With Computers*, 18(3):188–198, 2002.

[14] T.J. Baker and P.A. Cavallo. Dynamic adaptation for deforming tetrahedral meshes. *AIAA Journal*, pages 19–26, 1999. 99-3253.

[15] T.J. Baker and J.C. Vassberg. Tetrahedral mesh generation and optimization. In *Proc. of the 6th Int. Conf. on Numerical Grid Generation*, pages 337–349, 1998.

[16] S. Balay, K. Buschelman, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, and H. Zhang. PETSc Web page, 2009. http://www.mcs.anl.gov/petsc.

[17] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II — a general-purpose object-oriented finite element library. *ACM Trans. Math. Softw.*, 33(4), 2007.

[18] W. Bangerth and R. Rannacher. *Adaptive finite elements methods for differential equations*. ETH Zürich, 2003.

[19] T. Barth. *Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier-Stokes Equations*, chapter 4. Special Course on Unstructured Grid Methods for Advection Dominated Flows. AGARD R-787, May 1992.

[20] M.W. Beall and M.S. Shephard. A general topology-based mesh data structure. *International Journal for Numerical Methods in Engineering*, 40:1573–1596, 1997.

[21] T. Belytschko, Y. Krongauz, D. Organ, and M. Fleming. Meshless methods: an overview and recent developments. *Computer Methods in Applied Mechanics and Engineering*, 139(1–4):3–47, 1996.

[22] M. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, 1989.

[23] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.

[24] P.-E. Bernard, N. Chevaugeon, V. Legat, E. Deleersnijder, and J.-F. Remacle. High-order h-adaptative discontinuous galerkin methods for ocean modelling. *Ocean Dynamics*, 57:109–121, 2007.

[25] J. Bey. Tetrahedral grid refinement. *Computing*, 55:271–288, 1995.

[26] F. Bornemann, B. Erdmann, and R. Kornhuber. Adaptive multilevel methods in three-dimension spaces. *International Journal for Numerical Methods in Engineering*, 36:3187–3203, 1993.

[27] H. Borouchaki, P.-L. George, F. Hecht, P. Laug, and E. Saltel. Delaunay mesh generation governed by metric specifications - part i: Algorithms and part ii: Applications. *Finite Elements in Analysis and Design*, 25:61–83, 85–109, 1997.

[28] H. Borouchaki, F. Hecht, and P.J. Frey. Mesh gradation control. *International Journal for Numerical Methods in Engineering*, 43:1143–1165, 1998.

[29] R. Boussetta, T. Coupez, and L. Fourment. Adaptive remeshing based on a posteriori error estimation for forging simulation. *Computational Metal Forming*, 195(48-49):6626–6645, 2006.

[30] J.U. Brackbill, C. Kothe, and D.B. Zemach. A continuum method for modeling surface tension. *Journal of Computational Physics*, 2:335–354, 1992.

[31] C.A. Brebbia, L.C. Wrobel, and J.C.F. Telles. *Boundary element techniques: theory and applications in engineering.* Springer-Verlag, 1984.

[32] E. Briere de l'Isle and P.-L. George. *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations, I.*, volume 75, chapter Optimization of tetrahedral meshes, pages 97–128. I. Babuska et al. Eds., Springer, Berlin, 1993.

[33] K. Brown, S. Attaway, S. Plimpton, and B. Hendrickson. Parallel strategies for crash and impact simulations. *Computer Methods in Applied Mechanics and Engineering*, 184(2-4):375–390, 2000.

[34] G.C. Buscaglia and E.A. Dari. Anisotropic mesh optimization and its application in adaptivity. *International Journal for Numerical Methods in Engineering*, 40:4119–4136, 1997.

[35] X. Cai, C. Farhat, and M. Sarkis. A minimum overlap restricted additive Schwarz preconditioner and applications in 3D flow simulations. *Contemporary Mathematics*, 218:478–484, 1998.

[36] M. J. Castro-Diaz, F. Hecht, and B. Mohammadi. New progress in anisotropic grid adaptation for inviscid and viscous flows simulations. In *Proc. 4th Int. Meshing Roundtable*, pages 73–85. Sandia Nat. Lab., Albuquerque, NM, 1995.

[37] M.J. Castro-Diaz, F. Hecht, B. Mohammadi, and O. Pironneau. Anisotropic unstructured mesh adaptation for flow simulations. *International Journal for Numerical Methods in Fluids*, 25:475–491, 1997.

[38] P.A. Cavallo, N. Sinha, and G.M. Feldman. Parallel unstructured mesh adaptation method for moving body applications. *AIAA Journal*, 43:1937–1945, 2005.

[39] J.H. Cheng. Automatic adaptive remeshing for finite element simulation of forming processes. *International Journal for Numerical Methods in Engineering*, 26(1):1–18, 1988.

[40] E.M. Cherry, H.S. Greenside, and C.S. Henriquez. Efficient simulation of three-dimensional anisotropic cardiac tissue using an adaptive mesh refinement method. *Chaos*, 13(3):853, 2003.

[41] J. Chessa and T. Belytschko. An enriched finite element method and level sets for axisymmetric two-phase flow with surface tension. *International Journal for Numerical Methods in Engineering*, 58:2041–2064, 2003.

[42] J. Chessa and T. Belytschko. An extended finite element method for two-phase fluids. *Journal of Applied Mechanics (transactions of the asme)*, 70:10–17, 2003.

[43] L.P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the 9th Annual Symposium on Computational Geometry*, pages 274–280. ACM, New York, USA, 1993.

[44] G. Compère, E. Marchandise, and J.-F. Remacle. Transient adaptivity applied to two-phase incompressible flows. *Journal of Computational Physics*, 227:1923–1942, 2008.

[45] G. Compère and J.-F. Remacle. Mesh adaptivity complying to a geometrical model. , In preparation.

[46] G. Compère and J.-F. Remacle. MAdLib: Mesh Adaptation Library, 2008. `http://www.madlib.be`.

[47] G. Compère, J.-F. Remacle, J. Jansson, and J. Hoffman. A mesh adaptation framework for dealing with large deforming meshes. *International Journal for Numerical Methods in Engineering*. Published online, DOI: 10.1002/nme.2788.

[48] G. Compère, J.-F. Remacle, and E. Marchandise. Transient mesh adaptivity with large rigid-body displacements. In R. Garimella, editor, *Proc. 17th Int. Meshing Roundtable*, volume 3, pages 213–230. Springer, 2008.

[49] COMSOL. COMSOL multiphysics, 2009. http://www.comsol.com.

[50] T. Coupez and J.L. Chenot. Large deformations and automatic remeshing. In E. Onate (Eds.) E. Hinton, D.J.R. Owen, editor, *Computational Plasticity*, pages 1077–1088. Pineridge Press, Swansea, 1992.

[51] T. Coupez, H. Digonnet, and R. Ducloux. Parallel meshing and remeshing. *Applied Mathematical Modelling*, 25:153–175, 2000.

[52] T. Coupez, N. Soyris, and J.L. Chenot. 3-d finite element modeling of the forging process with automatic remeshing. *Journal of Materials Processing Technology*, 27:119–133, 1991.

[53] V. Cristini, J. Blawzdziewicz, and M. Loewenberg. An adaptive mesh algorithm for evolving surfaces: simulations of drop breakup and coalescence. *Journal of Computational Physics*, 168:445–463, 2001.

[54] M. Dai and D.P. Schmidt. Adaptive tetrahedral meshing in free-surface flow. *Journal of Computational Physics*, 208:228–252, 2005.

[55] H.L. de Cougny and M.S. Shephard. Parallel refinement and coarsening of tetrahedral meshes. *International Journal for Numerical Methods in Engineering*, 46:1101–1125, 1999.

[56] H.L. de Cougny, M.S. Shephard, and M.K. Georges. Explicit node point mesh smoothing within the octree mesh generator. Scorec report, Rensselaer Polytechnic Institute, 10 1990.

[57] F. de Sousa, N. Mangiavacchi, L. Nonato, A. Castel, M. Tome, V. Ferreira, J. Cuminato, and S. McKee. A front-tracking/front-capturing method for the simulation of 3d multi-fluid flows with free surfaces. *Journal of Computational Physics*, 198:469–499, 2004.

[58] W. Dettmer, P. Saksono, and D. Peric. On a finite element formulation for incompressible Newtonian fluid flows on moving domains in the presence of surface tension. *Computer Methods in Applied Mechanics and Engineering*, 19:659–668, 2003.

[59] C. Dobrzynski and P. Frey. Anisotropic delaunay mesh adaptation for unsteady simulations. In R. Garimella, editor, *Proc. 17th Int. Meshing Roundtable*, volume 3, pages 177–194. Springer, 2008.

[60] C. Dobrzynski, P.J. Frey, B. Mohammadi, and O. Pironneau. Fast and accurate simulations of air-cooled structures. *Computer Methods in Applied Mechanics and Engineering*, 195:3168–3180, 2006.

[61] C. Dobrzynski and J.-F. Remacle. Parallel mesh adaptation. *International Journal for Numerical Methods in Engineering.* , in preparation.

[62] J. Dompierre, M.-G. Vallet, Y. Bourgault, M. Fortin, and W.G. Habashi. Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver independent cfd. part iii: Unstructured meshes. *International Journal for Numerical Methods in Fluids*, 39:675–702, 2002.

[63] J. Donea. Arbitrary Lagrangian-Eulerian finite element methods. *Computational Methods for Transient Analysis*, 1:473–516, 1983.

[64] J. Donea, A. Huerta, J.-P. Ponthot, and A. Rodriguez-Ferran. *Encyclopedia of computational mechanics*, chapter Arbitrary Lagrangian-Eulerian Methods, pages 413–437. Wiley, 2004.

[65] H. Edelsbrunner and N.R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996.

[66] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson. *Computational Differential Equations*. Cambridge University Press New York, 1996.

[67] R.P. Fedkiw, T. Aslam, B. Merriman, and S. Osher. A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *Journal of Computational Physics*, 152:457–492, 1999.

[68] FEniCS. Fenics project, 2003. http://www.fenics.org.

[69] M. Francois, S. Cummins, E. Dendy, D. Kothe, J. Sicilian, and M. Williams. A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework. *Journal of Computational Physics*, 213:141–173, 2006.

[70] L.A. Freitag and C. Ollivier-Gooch. Tetrahedral mesh improvement using face swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1998.

[71] P.J. Frey and F. Alauzet. Anisotropic mesh adaptation for cfd computations. *Computer Methods in Applied Mechanics and Engineering*, 194:5068–5082, 2005.

[72] P.J. Frey and H. Borouchaki. Surface meshing using a geometric error estimate. *International Journal for Numerical Methods in Engineering*, 58:227–245, 2003.

[73] P.J. Frey and P.-L. George. *Mesh generation*. Wiley, London, 2008.

[74] R.V. Garimella and M.S. Shephard. Boundary layer mesh generation for viscous flow simulations. *International Journal for Numerical Methods in Engineering*, 49:193–218, 2000.

[75] P.-L. George and F. Hecht. *CRC Handbook of Grid Generation*, chapter Nonisotropic grids, pages 20.1–20.29. J. F. Thompson, B. K. Soni and N. P. Weatherill, Eds., CRC Press, Inc., Boca Raton, 1999.

[76] P.-L. George, F. Hecht, and E. Saltel. Automatic mesh generator with specified boundary. *Computer Methods in Applied Mechanics and Engineering*, 92(3):269–288, 1991.

[77] C. Geuzaine and J.-F. Remacle. Gmsh: a finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 2009. in press.

[78] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, 2009. `http://geuz.org/gmsh/`.

[79] P. Geuzaine. Newton-krylov strategy for compressible turbulent flows on unstructured meshes. *AIAA Journal*, 39:528–531, 2001.

[80] D. Greaves. A quadtree adaptive method for simulating fluid flows with moving interfaces. *Journal of Computational Physics*, 194:35–56, 2004.

[81] C. Gruau and T. Coupez. 3d tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric. *Computer Methods in Applied Mechanics and Engineering*, 194:4951–4976, 2005.

[82] K. Hans Raj, L. Fourment, T. Coupez, and J.L. Chenot. Simulation of industrial forging of axisymmetrical parts. *Engineering Computations*, 9(5):575–586, 1992.

[83] F. Harlow and J. Welch. Volume tracking methods for interfacial flow calculations. *Physics of fluids*, 8:21–82, 1965.

[84] O. Hassan, K. Morgan, E.J. Probert, and J. Peraire. Unstructured tetrahedral mesh generation for three-dimensional viscous flows. *International Journal for Numerical Methods in Engineering*, 39:549–567, 1996.

[85] W.D. Henshaw and D.W. Schwendeman. Moving overlapping grids with adaptive mesh refinement for high-speed reactive and non-reactive flow. *Journal of Computational Physics*, 216(2):744–779, 2006.

[86] C. Hirt, A. Amsden, and J. Cook. An Arbitrary Lagrangian-Eulerian computing method for all flow speeds. *Journal of Computational Physics*, 14:227–253, 1974.

[87] C. Hirt and B. Nichols. Volume of Fluid Method (VOF) for the dynamics of free boundaries. *Journal of Computational Physics*, 39:201–225, 1981.

[88] J. Hoffman, J. Jansson, and M. Stöckli. Unified continuum modeling of 3d fluid-structure interaction. *SIAM Journal on Scientific Computing (Submitted)*, 2009.

[89] J. Hoffman and C. Johnson. *Computational Turbulent Incompressible Flow: Applied Mathematics Body and Soul Vol 4*. Springer-Verlag Publishing, 2007.

[90] J. Hron and S. Turek. A monolithic FEM solver for ALE formulation of fluid structure interaction with configurations for numerical benchmarking. In M. Papadrakakis, E. Onate, and B. Schrefler, editors, *Computational Methods for Coupled Problems in Science and Engineering*, volume First Edition, page 148, 2005. Konferenzband 'First International Conference on Computational Methods for Coupled Problems in Science and Engineering' (Santorini, May 25th - 27th).

[91] T. Hughes, W. Liu, and T. Zimmermann. Lagrangian-Eulerian finite element formulation for incompressible viscous flows. *Computer Methods in Applied Mechanics and Engineering*, 29:239–349, 1981.

[92] T.J.R. Hughes. Consider a spherical cow – conservation of geometry in analysis: Implications for computational methods in engineering. In *IMA Hot Topics Workshop: Compatible Spatial Discretizations for Partial Differential Equations*, 2004.

[93] Y. Ito and K. Nakahashi. Unstructured mesh generation for viscous flow computations. In *Proc. 11th Int. Meshing Roundtable*, pages 367–378, 2002.

[94] B. Joe. Three-dimensional triangulations from local transformation. *SIAM J. Sci. Statist. Comput.*, 10:718–741, 1989.

[95] B. Joe. Construction of three dimensional improved-quality triangulations using local transformations. *SIAM Journal on Scientific Computing*, 16:1292–1307, 1995.

[96] M.T. Jones and P.E. Plassmann. Adaptive refinement of unstructured finite-element meshes. *Finite Elements in Analysis and Design*, 25:41–60, 1997.

[97] Y. Kallinderis and P. Vijayan. Adaptive refinement-coarsening scheme for three dimensional unstructured meshes. *AIAA Journal*, 31:1440–1447, 1993.

[98] A. Khawaja and Y. Kallinderis. Hybrid grid generation for turbomachinery and aerospace applications. *International Journal for Numerical Methods in Engineering*, 49:145–166, 2000.

[99] B. Kirk, S. Petersen, J.W. Peterson, R.H. Stogner, and D. Gaston. libMesh, 2002-2008. `http://libmesh.sourceforge.net/`.

[100] B. Kirk, J.W. Peterson, R.H. Stogner, and G.F. Carey. libmesh: a c++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering With Computers*, 22:237–254, 2006.

[101] S.H. Lee, C.S. Han, S.I. Oh, and P. Wriggers. Comparative crash simulations incorporating the results of sheet forming analyses. *Engineering computations*, 18(5-6):744–758, 2001.

[102] S. Legrand, E. Deleersnijder, E. Hanert, V. Legat, and E. Wolanski. High-resolution, unstructured meshes for hydrodynamic models of the great barrier reef, australia. *Estuarine, Coastal and Shelf Science*, 68:36–46, 2006.

[103] P. Leyland and R. Richter. Completely parallel compressible adaptive unstructured meshes. *Computer Methods in Applied Mechanics and Engineering*, 184:467–483, 2000.

[104] X. Li. *Mesh Modification Procedures for General 3-D Non-manifold Domains*. PhD thesis, Rensselear Polytechnic Institute, 2003.

[105] X. Li, M.S. Shephard, and M.W. Beall. Accounting for curved domains in mesh adaptation. *International Journal for Numerical Methods in Engineering*, 58:247–276, 2003.

[106] X. Li, M.S. Shephard, and M.W. Beall. 3d anisotropic mesh adaptation by mesh modification. *Computer Methods in Applied Mechanics and Engineering*, 194:4915–4950, 2005.

[107] L.E. Lijewski and N.E. Suhs. Time-accurate computational fluid dynamics approach to transonic store separation trajectory prediction. *Journal of Aircraft*, 31(4):886–891, 1994.

[108] A. Liu and B. Joe. On the shape of tetrahedra from bisection. *Mathematics of computation*, 63:141–154, 1994.

[109] A. Liu and B. Joe. Relationship between tetrahedron shape measures. *BIT Numerical Mathematics*, 34(2):268–287, 1994.

[110] A. Liu and B. Joe. Quality local refinements of tetrahedral meshed based on bisection. *SIAM Journal on Scientific Computing*, 16:1269–1291, 1995.

[111] A. Liu and B. Joe. Quality local refinements of tetrahedral meshed based on 8-subtetrahedron subdivision. *Mathematics of Computations*, 65:1183–2000, 1996.

[112] R. Löhner. Extensions and improvements of the advancing front grid generation technique. *Communications in Numerical Methods in Engineering*, 12:683–702, 1996.

[113] R. Löhner and J. Baum. Adaptive h-refinement on 3d unstructured grids for transient problems. *International Journal for Numerical Methods in Fluids*, 14:1407–1419, 1992.

[114] F. Losasso, R. Fedkiw, and S. Osher. Spatially adaptive techniques for level set methods and incompressible flows. *Computers and Fluids*, 35:995–1010, 2006.

[115] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans Graph (SIGGRAPH Proc)*, 23:457–462, 2004.

[116] M. Mäntylä. *An introduction to solid modeling*. Computer Science Press, Rockville, Maryland, 1988.

[117] E. Marchandise. *Simulation of three-dimensional two-phase flows: coupling of a stabilized finite element method with a discontinuous level set approach*. PhD thesis, Universite catholique de Louvain, november 2006.

[118] E. Marchandise, P. Geuzaine, N. Chevaugeon, and J.-F. Remacle. A stabilized finite element method using a discontinuous level set approach for the computation of bubble dynamics. *Journal of Computational Physics*, 225:949–974, 2007.

[119] E. Marchandise and J.-F. Remacle. A stabilized finite element method using a discontinuous level set approach for solving two phase incompressible flows. *Journal of Computational Physics*, 219:780–800, 2006.

[120] E. Marchandise, J.-F. Remacle, and N. Chevaugeon. A quadrature-free discontinuous galerkin method for the level set equation. *Journal of Computational Physics*, 212:338–357, 2006.

[121] D.L. Marcum. Generation of unstructured grids for viscous flow applications. *AIAA Journal*, 1995.

[122] D. Martin and P. Colella. A cell-centered adaptive projection method for the incompressible euler equations. *Journal of Computational Physics*, 163:271–312, 2000.

[123] S.R. Mathur. Unsteady flow simulations using unstructured sliding meshes. *AIAA Journal*, June 1994.

[124] J.M. Maubach. Local bisection refinement for n-simplicial grids generated by reflection. *SIAM Journal on Scientific Computing*, 16:210–227, 1995.

[125] N. Moes, J. Dolbow, and T. Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46(11):131–150, 1999.

[126] D.M. Mount and S. Arya. ANN library, 2006. `http://www.cs.umd.edu/~mount/ANN/`.

[127] J. Muller, O. Sahni, X. Li, K.E. Jansen, M.S. Shephard, and C.A. Taylor. Anisotropic adaptive finite element method for modelling blood flow. *Computer Methods in Biomechanics and Biomedical Engineering*, 8(5):295–305, 2005.

[128] D.R. Musser and A. Saini. *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library.* Addison-Wesley, 1996.

[129] S. Nagrath, K.E. Jansen, and R. Lahey. Computation of incompressible bubble dynamics with a stabilized finite element level set method. *Computer Methods in Applied Mechanics and Engineering*, 194(42-44):4565–4587, 2005.

[130] S. Narayanan, H. Goossens, and N. Kossen. Coalescence of two bubbles rising in line at low reynolds number. *Chemical Engineering Science*, 29:2071–2082, 1974.

[131] C. Norman and M.J. Miksis. Non-linear dynamics of thin films and fluid interfaces. *Physica D: Nonlinear Phenomena*, 209:191–204, 2005.

[132] S. Osher and J.A. Sethian. Fronts propagating with curvature dependent speed: algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[133] C.C. Pain, A.P. Umpleby, C.R.E. de Oliveria, and A.J.H. Goddard. Tetrahedral mesh optimization and adaptivity for steady-state and transient finite element calculations. *Computer Methods in Applied Mechanics and Engineering*, 190:3771–3796, 2001.

[134] P.P. Pebay and T.J. Baker. Analysis of triangle quality measures. *Mathematics of Computation*, 72(244):1817–1839, 2003.

[135] R. Pember, J. Bell, P. Colella, W. Crutchfield, and M. Welcome. An adaptive cartesian grid method for unsteady compressible flow in irregular regions. *Journal of Computational Physics*, 120:278–304, 1995.

[136] J. Peraire, J. Peiro, and K. Morgan. Adaptive remeshing for three dimensional compressible flow computation. *Journal of Computational Physics*, 103:269–285, 1992.

[137] D. Peric, M. Vaz, and D.R.J. Owen. On adaptive strategies for large deformations of elasto-plastic solids at finite strains: computational issues and industrial applications. *Computer Methods in Applied Mechanics and Engineering*, 176(1-4):279–312, 1999.

[138] S. Phongthanapanich and P. Dechaumphai. Adaptive delaunay triangulation with object-oriented programming for crack propagation analysis. *Finite Elements in Analysis and Design*, 40(13-14):1753–1771, 2004.

[139] J. Pilliot and E. Puckett. Second order accurate Volume-of-Fluid algorithms for tracking material interfaces. *Journal of Computational Physics*, 199:465–502, 2004.

[140] S. Piperno and C. Farhat. Partitioned procedures for the transient solution of coupled aeroelastic problems - part ii: energy transfer analysis and three-dimensional applications. *Computer Methods in Applied Mechanics and Engineering*, 190:3147–3170, 2001.

[141] S. Popinet. Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries. *Journal of Computational Physics*, 190:572–600, 2003.

[142] S. Popinet. Gerris Flow Solver, 2009. `http://gfs.sourceforge.net/wiki/index.php/Main_Page`.

[143] S. Popinet and S. Zaleski. A front-tracking algorithm for accurate representation of surface tension. *International Journal for Numerical Methods in Fluids*, 30:775–793, 1999.

[144] N.C. Prewitt, D.M. Belk, and W. Shyy. Parallel computing of overset grids for aerodynamic problems with moving objects. *Progress in Aerospace Science*, 36:117–172, 2000.

[145] M. Quecedo, M. Pastor, and M. Herreros. An experimental study of the collapse of liquid columns on a rigid horizontal plane. *Philos. Trans. A*, 244:312–324, 1952.

[146] R. Ramamurti and W.C. Sandberg. A three-dimensional computational study of the aerodynamic mechanisms of insect flight. *The Journal of Experimental Biology*, 205:1507–1518, 2002.

[147] R. Rausch, J. Batina, and H. Yang. Spatial adaptation procedures on tetrahedral meshes for unsteady aerodynamic flow calculations. *AIAA Journal*, 30:257–272, 1992.

[148] J.-F. Remacle, J.E. Flaherty, and M.S. Shephard. An Adaptive Discontinuous Galerkin Technique with an Orthogonal Basis Applied to Compressible Flow Problems. *SIAM Review*, 45:53–72, 2003.

[149] J.-F. Remacle, C. Geuzaine, G. Compère, and B. Helenbrook. *Encyclopedia of Aerospace Engineering*, chapter Adaptive mesh generation and visualisation. John Wiley and Sons Ltd. submitted.

[150] J.-F. Remacle, C. Geuzaine, G. Compère, and E. Marchandise. High quality surface meshing using harmonic maps. *International Journal for Numerical Methods in Engineering*, 2009. , Under review.

[151] J.-F. Remacle, O. Klaas, J.E. Flaherty, and M.S. Shephard. Parallel algorithm oriented mesh database. *Engineering With Computers*, 18:274–284, 2002.

[152] J.-F. Remacle, X. Li, M.S. Shephard, and J.E. Flaherty. Anisotropic adaptive simulation of transient flows using Discontinuous Galerkin methods. *International Journal for Numerical Methods in Engineering*, 62(7):899–923, 2005.

[153] J.-F. Remacle and M.S. Shephard. An algorithm oriented mesh database. *International Journal for Numerical Methods in Engineering*, 58:349–374, 2003.

[154] J.-F. Remacle, S. Soares Frazao, X. Li, and M.S. Shephard. An adaptive discretization of shallow-water equations based on discontinuous galerkin methods. *International Journal for Numerical Methods in Fluids*, 52:903–923, 2006.

[155] M.C. Rivara. Selective refinement/derefinement algorithms for sequences of nested triangulations. *International Journal for Numerical Methods in Engineering*, 28:2889–2906, 1989.

[156] M.C. Rivara. A 3-d refinement algorithm suitable for adaptive and multigrid techniques. *Communications in Applied Numerical Methods*, 8:281–290, 1992.

[157] M.C. Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *International Journal for Numerical Methods in Engineering*, 40:3313–3324, 1997.

[158] A. Roma, C. Peskin, and M. Berger. An adaptive version of the immersed boundary method. *Journal of Computational Physics*, 153:509–534, 1999.

[159] J. Ruppert and R. Seidel. On the difficulty of tetrahedralizing 3-dimensional non-convex polyhedra. *Discrete Computational Geometry*, 7:227–253, 1992.

[160] M. Rüter and E. Stein. Adaptive finite element analysis of crack propagation in elastic fracture mechanics based on averaging techniques. *Computational Materials Science*, 31(3-4):247–257, 2004.

[161] O. Sahni, K.E. Jansen, M.S. Shephard, C.A. Taylor, and M.W. Beall. Adaptive boundary layer meshing for viscous flow simulations. *Engineering With Computers*, 24:267–285, 2008.

[162] O. Sahni, X.J. Luo, K.E. Jansen, and M.S. Shephard. Curved boundary layer meshing for adaptive viscous flow simulations. *Finite Elements in Analysis and Design*, 2009. In press.

[163] O. Sahni, J. Muller, K.E. Jansen, M.S. Shephard, and C.A. Taylor. Efficient anisotropic adaptive discretization of the cardiovascular system. *Computer Methods in Applied Mechanics and Engineering*, 195:5634–5655, 2006.

[164] P. Saramito and N. Roquet. An adaptive finite element method for viscoplastic fluid flow in pipes. *International Journal for Numerical Methods in Engineering*, 190:5391–5412, 2001.

[165] J.A. Schmidt, J.C. Johnson, J.C. Eason, and R.S. MacLeod. Applications of automatic mesh generation and adaptive methods in computational medicine. *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, pages 367–, 1995.

[166] J. Schöberl, J. Gerstmayr, and R. Gaisbauer. NETGEN, 2003-2009. `http://www.hpfem.jku.at/netgen/`.

[167] W.J. Schroeder and M.S. Shephard. Geometry-based fully automatic mesh generation and the delaunay triangulation. *International Journal for Numerical Methods in Engineering*, 26:2503–2515, 1988.

[168] V. Selmin and L. Formaggia. Unified construction of finite element and finite volume discretizations for compressible flows. *International Journal for Numerical Methods in Fluids*, 39:1–32, 1996.

[169] D. Sharov, H. Luo, J.D. Baum, and R. Löhner. Unstructured navier-stokes grid generation at corners and ridges. *International Journal for Numerical Methods in Fluids*, 43:717–728, 2003.

[170] M.S. Shephard, J.E. Flaherty, K.E. Jansen, X. Li, X. Luo, N. Chevaugeon, J.-F. Remacle, M.W. Beall, and R.M. O'Bara. Adaptive mesh generation for curved domains. *Applied Numerical Mathematics*, 52:251–271, 2005.

[171] M.S. Shephard and M.K. Georges. Reliability of automatic 3d mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 101:443–462, 1992.

[172] H. Si. TetGen, 2007. `http://tetgen.berlios.de`.

[173] H. Si and K. Gaertner. Meshing piecewise linear complexes by constrained delaunay tetrahedralizations. In *Proc. 14th Int. Meshing Roundtable*, pages 147–163, 2005.

[174] T.E. Smith and J.R. Kadambi. The effect of steady aerodynamic loading on the flutter stability of turbomachinery blading. *ASME, Transactions, Journal of Turbomachinery*, 1993.

[175] A. Smoliansky. *Numerical Modeling of Two-Fluid Interfacial Flows*. PhD thesis, University of Jyvaskyla, Finland, 2001.

[176] W. Speares and M. Berzins. A 3d unstructured mesh adaptation algorithm for time dependent shock-dominated problems. *International Journal for Numerical Methods in Fluids*, 25:81–104, 1997.

[177] K. Stein, T. Tezduyar, and R. Benney. Mesh moving techniques for fluid-structure interactions with large displacements. *Journal of Applied Mechanics*, 70:58–63, 2003.

[178] M. Sussman. A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *Journal of Computational Physics*, 187:110–136, 2003.

[179] M. Sussman. A parallelized, adaptive algorithm for multiphase flows in general geometries. *Computers and Structures*, 83:435–444, 2005.

[180] M. Sussman, S. Almgren, B. Bell, P. Colella, L. Howell, and M. Welcome. An adaptive level set approach for incompressible two-phase flows. *Journal of Computational Physics*, 148:81–124, 1999.

[181] M. Sussman and E. Fatemi. An efficient interface preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow. *SIAM Journal on Scientific Computing*, 20:1165–1191, 1999.

[182] M. Sussman, E. Fatemi, P. Smereka, and S. Osher. An improved level set method for incompressible two-fluid flows. *Computers and Fluids*, 27:663–680, 1998.

[183] M. Sussman and E. Puckett. A coupled level set and volume-of-fluid method for computing 3d and axisymmetric incompressible two-phase flows. *Journal of Computational Physics*, 162:301–337, 2000.

[184] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114:146–159, 1994.

[185] A. Tam, D. Ait-Ali-Yahia, M. Robichaud, M. Moore, V. Kozel, and W. Habashi. Anisotropic mesh adaptation for 3d flows on structured and unstructured grids. *Computer Methods in Applied Mechanics and Engineering*, 189:1205–1230, 2000.

[186] T.E. Tezduyar, M. Behr, S. Mittal, and J. Liou. Computation of unsteady incompressible flows with the finite element methods–space-time formulations, iterative strategies and massively parallel implementations. *ASME Pressure Vessels Piping Div Publ PVP., ASME, NY*, 246:7–24, 1992.

[187] A.-K. Tornberg. *Interface Tracking Methods with Applications to Multiphase Flows*. PhD thesis, Royal Institute of Technology (KTH), Finland, 2000.

[188] F.M. White. *Viscous fluid flow.* McGraw-Hill, New York, 1991.

[189] M.W. Williams, D.B. Kothe, and E.G. Puckett. Convergence and accuracy of continuum surface tension models. In W. Shyy and R. Narayanan, editors, *Fluid Dynamics at Interface*, pages 294–305. Cambridge University Press, Cambridge, 1999.

[190] E. Wyart, D. Coulon, M. Duflot, T. Pardoen, J.-F. Remacle, and F. Lani. A substructured fe-shell/xfe-3d method for crack analysis in thin-walled structures. *International Journal for Numerical Methods in Engineering*, 72(7):757–779, 2007.

[191] X. Yang, A. James, J. Lowengrub, X. Zheng, and V. Cristini. An adaptive coupled level-set/volume-of-fluid interface capturing method for unstructured triangular grids. *Journal of Computational Physics*, 217:364–394, 2006.

[192] X. Zheng, J. Lowengrub, A. Anderson, and V. Cristini. Adaptive unstructured volume remeshing ii: Application to two- and three-dimensional level-set simulations of multiphase flow. *Journal of Computational Physics*, 208:626–650, 2005.

[193] O.C. Zienkiewicz and J.Z. Zhu. Superconvergent patch recovery and a posteriori error estimates, part 1: The recovery technique. *International Journal for Numerical Methods in Engineering*, 33(7):1331–1364, 1992.

Appendix A

About metrics

When doing anisotropic mesh adaptation, it is required to specify the desired mesh sizes in terms of edge lengths and directions in which those sizes operate. It is also needed to evaluate how a given edge satisfies or not the size prescriptions. For the sake of simplicity, we will consider here the question in two dimensions, but the developments can easily be extended to three dimensions.

Metric and unit length

The problem is summarized on Figure A.1, in which an edge e is given as well as the prescribed sizes: h_1 and h_2, respectively in directions $\mathbf{t_1}$ and $\mathbf{t_2}$. We currently make the assumption that the prescribed sizes and their directions are constant along e. We also assume that the principal directions are perpendicular to each other.

An interesting way to measure the compatibility of e with the prescribed sizes is to transform the usual space in a space in which the length of a perfect edge is 1 whatever its direction in the usual space. To this end, we define a transformation which is the combination of a rotation $\mathcal{R}$ and a scaling $\mathcal{H}$. The rotation $\mathcal{R}$ aligns $\mathbf{t_1}$ and $\mathbf{t_2}$ with the main axis of the transformed space (Figure A.1 (b)). The columns of the matrix corresponding to the rotation are the t_i. The scaling $\mathcal{H}$ applies $h_i\mathbf{t_i}$ on $\mathbf{t_i}$ for $i = 1, 2$ (Figure A.1 (c)). The corresponding matrix is therefore a diagonal matrix with coefficients $1/h_i$.

If $\mathbf{e}$ is the vector joining the summits of e, the vector $\mathbf{e}^{\mathcal{M}}$ corresponding to the transformed edge is

$$\mathbf{e}^{\mathcal{M}} = \mathcal{H}\,\mathcal{R}\,\mathbf{e}. \tag{A.1}$$

Once this transformation is defined, we can write the dimensionless length $l_{e^{\mathcal{M}}}$ of e as

$$l_{e^{\mathcal{M}}} = \sqrt{\mathbf{e}^{\mathcal{M}} \cdot \mathbf{e}^{\mathcal{M}}} = \sqrt{\mathbf{e}^t\mathcal{R}^t\mathcal{H}^t\mathcal{H}\mathcal{R}\mathbf{e}} = \sqrt{\mathbf{e}^t\mathcal{R}^t\mathcal{H}^2\mathcal{R}\mathbf{e}} = \sqrt{\mathbf{e}^t\mathcal{M}\mathbf{e}}, \tag{A.2}$$

with $\mathcal{M} = \mathcal{R}^t\mathcal{H}^2\mathcal{R}$. The metric $\mathcal{M}$ is symmetric positive definite. The directions $\mathbf{t_i}$ are its eigenvectors, and the corresponding eigenvalues are $\lambda_i^{\mathcal{M}} = 1/h_i^2$.

Note that the length of $e^{\mathcal{M}}$ is dimensionless. An edge with a dimensionless size $l_{e^{\mathcal{M}}}$ equal to 1 is called a *unit* edge length. It has exactly the size prescribed

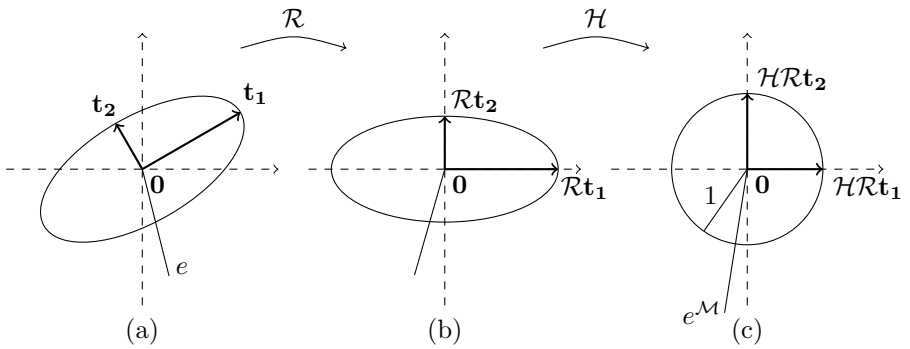


Figure A.1: The successive transformations from the usual space to the metric space.

for its orientation. In the transformed space, the unit edges are the edges that join a point of the unit circle to the origin. From Figure A.1, we remark that the set of points defining a unit edge, i.e. which lies on the unit circle in the transformed space form an ellipse (an ellipsoid in 3D) in the usual space, with principal directions $\mathbf{t_i}$ and radii and h_i.

If the metric $\mathcal{M}$ depends on the position $\mathbf{x}$, the dimensionless length of e is written

$$l_{e^{\mathcal{M}}} = \int_e \sqrt{\mathbf{e}^t \mathcal{M}(\mathbf{x})\mathbf{e}}\, dx. \tag{A.3}$$

Values in transformed space

Other values can be computed in the transformed space, like angles, areas or volumes.

Angles : In the usual space, the angle θ between two vectors $\mathbf{v_1}$ and $\mathbf{v_2}$ can be computed from

$$cos(\theta) = \frac{\mathbf{v_1} \cdot \mathbf{v_2}}{\| \mathbf{v_1} \| \| \mathbf{v_2} \|}. \tag{A.4}$$

Using the same formula to obtained the angle $\theta^{\mathcal{M}}$ between the transformed vectors $\mathbf{v_1}^{\mathcal{M}}$ and $\mathbf{v_2}^{\mathcal{M}}$, we have

$$cos(\theta^{\mathcal{M}}) = \frac{\mathbf{v_1}^{\mathcal{M}} \cdot \mathbf{v_2}^{\mathcal{M}}}{\| \mathbf{v_1}^{\mathcal{M}} \| \| \mathbf{v_2}^{\mathcal{M}} \|} = \frac{\mathbf{v_1} \mathcal{M} \mathbf{v_2}}{l_{\mathbf{v_1}^{\mathcal{M}}}\, l_{\mathbf{v_2}^{\mathcal{M}}}}. \tag{A.5}$$

Areas : Given a surface S in the usual two-dimensional space, like depicted in Figure A.2, we would like to obtain the area $A_S^{\mathcal{M}}$ of S in the transformed space. Considering the metric transformation as the combination of a rotation and a scaling, we observe that the scaling is the only transformation that affects the area of the surface. Given the area in the usual space A_S, it is therefore

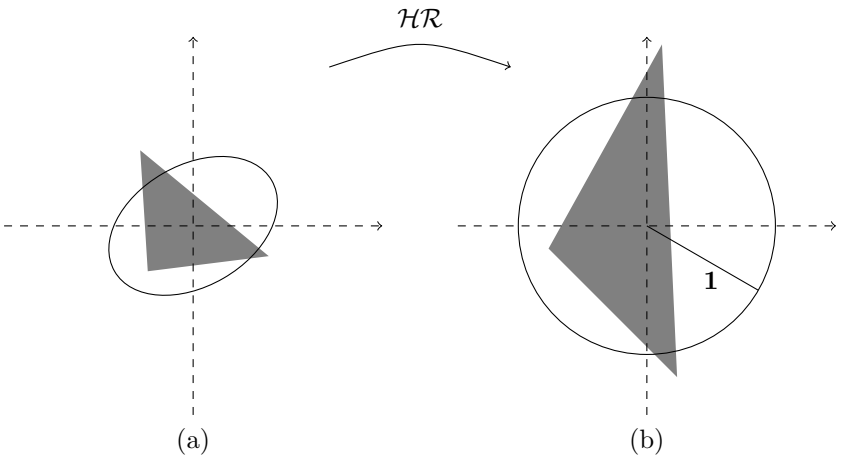


Figure A.2: Area in a space transformed by a metric.

straightforward to compute $A_S^{\mathcal{M}}$:

$$A_S^{\mathcal{M}} = \frac{A_S}{h_1 h_2}. \tag{A.6}$$

The 3D extension to the computation of the volume $V_R^{\mathcal{M}}$ of a region R in the transformed space is trivial and yields:

$$V_R^{\mathcal{M}} = \frac{V_R}{h_1 h_2 h_3}, \tag{A.7}$$

with V_R the volume of R in the usual space.

Metrics intersection

It is quite common that several metric fields are given, each one corresponding to different sizes and alignments. In such a case, the different metrics are intersected like depicted in Figure A.3. If two metrics $\mathcal{M}_1$ and $\mathcal{M}_2$ are given, a usual way to build the intersection metric $\mathcal{M}_{1\cap 2}$ is to find the largest metric such that the length of any vector $\mathbf{v}$ computed in the space transformed by $\mathcal{M}_{1\cap 2}$ is larger than the lengths computed in both transformed spaces of $\mathcal{M}_1$ and $\mathcal{M}_2$. Graphically, the ellipse corresponding to $l_{\mathbf{v}}^{\mathcal{M}_{1\cap 2}} = 1$ is exactly contained in the ellipses given by $l_{\mathbf{v}}^{\mathcal{M}_1} = 1$ and $l_{\mathbf{v}}^{\mathcal{M}_2} = 1$.

In the literature, several works can be found that describe this metrics intersection method. However, the descriptions are usually limited to the resulting formulae. We give here a more intuitive description of the method and show graphically how the classical results are obtained.

Figure A.4 (a) shows the ellipses corresponding to two metrics $\mathcal{M}_1$ and $\mathcal{M}_2$ and their intersection $\mathcal{M}_{1\cap 2}$ in the usual space. The lengths of a vector $\mathbf{v}$ in

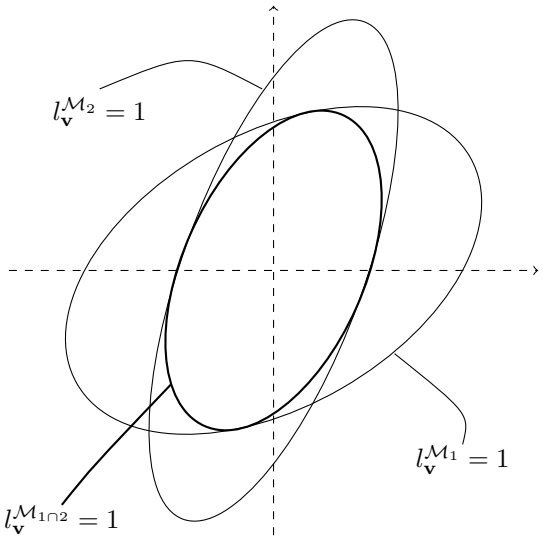


Figure A.3: Representation of the intersection of two metrics.

those metrics spaces are

$$l_{\mathbf{v}}^{\mathcal{M}_1} = \sqrt{\mathbf{v}^t \mathcal{M}_1 \mathbf{v}}, \tag{A.8}$$

$$l_{\mathbf{v}}^{\mathcal{M}_2} = \sqrt{\mathbf{v}^t \mathcal{M}_2 \mathbf{v}}, \tag{A.9}$$

$$l_{\mathbf{v}}^{\mathcal{M}_{1\cap 2}} = \sqrt{\mathbf{v}^t \mathcal{M}_{1\cap 2} \mathbf{v}}. \tag{A.10}$$

Figure A.4 (b) shows the same ellipses in the transformed space of the metric $\mathcal{M}_1$. In this space, $\mathbf{v}$ becomes $\mathbf{v}^{\mathcal{M}_1}$ which is obtained by

$$\mathbf{v}^{\mathcal{M}_1} = \mathcal{H}_1 \mathcal{R}_1 \mathbf{v}, \tag{A.11}$$

where $\mathcal{H}_1$ and $\mathcal{R}_1$ are the scaling and rotational transformations corresponding to $\mathcal{M}_1$. The ellipse of the vectors of unit length for $\mathcal{M}_1$ becomes a circle of radius 1, and the ellipse associated to $\mathcal{M}_2$ is given by

$$\mathbf{v}^t \mathcal{M}_2 \mathbf{v} = \mathbf{v}^{\mathcal{M}_1 t} \mathcal{R}_1^{-1 t} \mathcal{H}_1^{-1 t} \mathcal{M}_2 \mathcal{H}_1^{-1} \mathcal{R}_1^{-1} \mathbf{v}^{\mathcal{M}_1} = \mathbf{v}^{\mathcal{M}_1 t} \mathcal{M}_1^{-1} \mathcal{M}_2 \mathbf{v}^{\mathcal{M}_1} = 1. \tag{A.12}$$

We define the matrix $\mathcal{N}$ as

$$\mathcal{N} = \mathcal{M}_1^{-1} \mathcal{M}_2, \tag{A.13}$$

The matrix $\mathcal{N}$ can be diagonalized, which yields

$$\mathcal{N} = \mathcal{Q}^t \mathcal{L}^2 \mathcal{Q}, \tag{A.14}$$

with $\mathcal{Q}$ a matrix whose columns are the main directions of $\mathcal{M}_2$ expressed in the transformed space of $\mathcal{M}_1$, and $\mathcal{L}$ a diagonal matrix with coefficients $\lambda_i^{\mathcal{N}}$ equal to $\lambda_i^{\mathcal{M}_2} / \lambda_i^{\mathcal{M}_1}$.

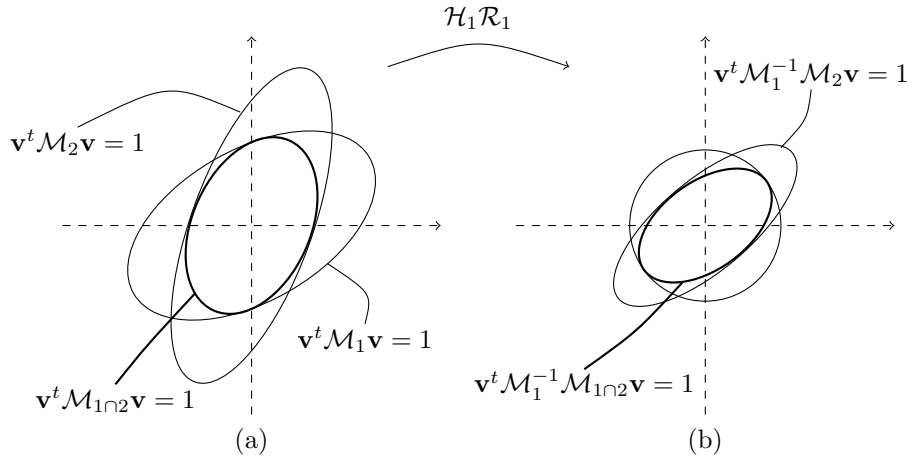


Figure A.4: Construction of the intersection of two metrics $\mathcal{M}_1$ and $\mathcal{M}_2$. The metrics and their intersection $\mathcal{M}_{1\cap2}$ are represented in (a) the usual space and (b) the transformed space corresponding to $\mathcal{M}_1$.

We observe that the ellipse corresponding to $\mathcal{M}_{1\cap2}$ in the transformed space of $\mathcal{M}_1$, i.e. $\mathcal{M}_1^{-1}\mathcal{M}_{1\cap2}$ is the intersection of the ellipse of $\mathcal{M}_2$ in the same space with the unit circle. In that space, the directions of $\mathcal{M}_{1\cap2}$ are therefore the directions of $\mathcal{M}_2$, and the intersection with $\mathcal{M}_1$ just results to a minimum value of 1 for its eigenvalues. The intersection metric in the space transformed by $\mathcal{M}_1$ is therefore

$$\mathcal{N}^* = \mathcal{Q}^t{\mathcal{L}^*}^2\mathcal{Q}, \tag{A.15}$$

with $\mathcal{L}^*$ a diagonal matrix with coefficients equal to $max\left(\lambda_i^{\mathcal{N}}, 1\right)$.

Coming back to the usual space, the intersection metric is computed as

$$\mathcal{M}_{1\cap2} = \mathcal{M}_1\mathcal{N}^* = {\mathcal{P}^{-1}}^t \begin{pmatrix} max\left(\lambda_1^{\mathcal{M}_1}, \lambda_1^{\mathcal{M}_2}\right) & 0 \\ 0 & max\left(\lambda_2^{\mathcal{M}_1}, \lambda_2^{\mathcal{M}_2}\right) \end{pmatrix} \mathcal{P}^{-1}, \tag{A.16}$$

where $\mathcal{P}$ is the matrix mapping the canonical basis to the basis associated with $\mathcal{N}$.

Metric interpolation

The problem here is to find a metric $\mathcal{M}(t)$ that is equal to a metric $\mathcal{M}_0$ for $t=0$, and a metric $\mathcal{M}_1$ for $t=1$, and which vary monotonously for $t \in [0,1]$. Typically, such an interpolation is useful for finding a metric in a point of a straight line when the only metric values that are known are the values at the extremities of the line.

A classical method to obtain $\mathcal{M}(t)$ is the following

$$\mathcal{M}(t) = \mathcal{P}^{-1t} \begin{pmatrix} \frac{1}{\left((1-t)h_1^{\mathcal{M}_0} + t\, h_1^{\mathcal{M}_1}\right)^2} & 0 \\ 0 & \frac{1}{\left((1-t)h_2^{\mathcal{M}_0} + t\, h_2^{\mathcal{M}_1}\right)^2} \end{pmatrix} \mathcal{P}^{-1}, \qquad \text{(A.17)}$$

where $\mathcal{P}$ is the matrix mapping the canonical basis to the basis associated with $\mathcal{N} = \mathcal{M}_1^{-1}\mathcal{M}_2$, and $h_i^{\mathcal{M}_j}$ is the length prescribed in the i^{th} direction in the metric $\mathcal{M}_j$, $j \in [0, 1]$. More details can be found in [73].

Appendix B

Annotated C++ classes for interfacing a PDE solver to MAdLib

B.1 Mesh adaptation interface

```
//------------------------------------------------------------------------------
// Class interfacing MAdLib with 'Solver'
//------------------------------------------------------------------------------
class MAdLibInterface {

public:

  MAdLibInterface();
  ~MAdLibInterface();

  void adaptMesh();

private:

  // Mesh to mesh conversion
  void importFromMAdMesh(const MAd::pMesh, Solver_mesh *);
  void exportToMAdMesh(const Solver_mesh *, MAd::pMesh);
  void importFromMAdModel(const MAd::pGModel, Solver_model *);
  void exportToMAdModel(const Solver_model *, MAd::pGModel);

  // Size field construction
  void buildSizeField(MAd::PWLSField *);

  // Solution to solution conversion
  void attachSolutionToMesh(MAd::pMesh);
  void getSolutionFromMesh(MAd::pMesh);

private:

  // The solver that needs mesh adaptivity
  Solver * solver;

  // Correspondence tables between nodal id's in the solver
  // and in the MAdLib mesh
  std::map<int,int> MAdToSolverIds;
  std::map<int,int> SolverToMAdIds;
};
```

B.2 Solver data interface

```
// Solver class containing the solver geometrical model if any.
class Solver_model
{
public:
  void addGeoEntity(int dim, int id);
  std::set<std::pair<int,int> > getAllGeoEntities() const;
  // return a set of pairs(dimension,id),
  // each pair representing a geometric entity.
};

//  Solver class containing the solver mesh
class Solver_mesh
{
public:
  void allocate (int nNodes, int nElements);
  void addNode (int id, double x, double y, double z);
  void addElement (int id, int * nodes);
  int getDim() const;
  int nVertices() const;
  int nElements() const;
  const double ** getCoordinates() const;
  const int ** getElements() const;
  const int * getElemGeoTags() const;
};

//  Solver solution. We assume a nodal solution but the current example can be
//  easily extended to other discretizations.
class Solver_solution
{
public:
  double * operator[](int i);
  const double operator[](int i) const;
};

//  Solver class containing pointers to solver data, solution and mesh
class Solver
{
public:
  Solver_model     * getModel()     {return model;}
  Solver_mesh      * getMesh()      {return mesh;}
  Solver_solution  * getSolution() {return solution;}
  void deleteMesh();
  void deallocateSolution();
  void allocateSolution();
  // optional functions:
  void deleteData();
  void allocateAndComputeData();
  double prescribedEdgeLength(int node);
private:
  Solver_model     * model;
  Solver_mesh      * mesh;
  Solver_solution  * solution;
};
```

B.3 Mesh adaptation in a model solver

```
void MAdLibInterface::adaptMesh()
{
  //-------------------------------------------------------
  // Step 1: Prepare for adaptation
  //-------------------------------------------------------

  // 1. Delete mesh/solution dependent data in the solver
  solver->deleteData();

  // 2.A. Build the MAdLib geometrical model.
  pGModel MAdModel = NULL;
  GM_create(&MAdModel,"theModel");
  exportToMAdModel(solver->getModel(), MAdModel);

  // 2.B. Build the MAdLib mesh.
  pMesh MAdMesh = M_new(MAdModel);
  exportToMAdMesh(solver->getMesh(), MAdMesh);

  // 3. Transfer solution to the MAdLib mesh as an attached data
  attachSolutionToMesh(MAdMesh);
  solver->deallocateSolution();

  // 4. Delete the solver mesh.
  solver->deleteMesh();

  // 5. Build the size field used in adaptation
  PWLSField * sizeField = new PWLSField(MAdMesh);
  buildSizeField(sizeField);

  //-------------------------------------------------------
  // Step 2: Run the adaptation
  //-------------------------------------------------------

  // 6.A. Build the adaptation tool
  MeshAdapter * adapter = new MeshAdapter(MAdMesh,sizeField);

  // 6.B. Register the callback function(s) of the solver
  adapter->addCallback(Solver_CBFunction,(void*)this);

  // 6.C. Edit the adaptation parameters if necessary
  adapter->setEdgeLenSqBounds( 1.0/3.0, 3.0 );
  adapter->... ; // see AdaptInterface.h

  // 6.D. Run the adaptation procedure
  adapter->run();

  // 6.E. Optional output
  adapter->printStatistics(std::cout);
  M_writeMsh(MAdMesh,"adapted_mesh.msh",2);

  // 6.F. Clean the adaptation objects
  delete adapter;
  delete sizeField;

  //-------------------------------------------------------
  // Step 3: Rebuild solver data and mesh
  //-------------------------------------------------------

  // 7. Rebuild the solver mesh
  importFromMAdModel(MAdModel, solver->getModel());
  importFromMAdMesh(MAdMesh, solver->getMesh());

  // 8. Get the solution from the MAdLib mesh
  solver->allocateSolution();
  getSolutionFromMesh(MAdMesh);

  // 9. Delete MAdLib mesh
  delete MAdMesh;
  delete MAdModel;

  // 10. Build mesh/solution dependent data in the solver
  solver->allocateAndComputeData();
}
```

B.4 Callback function: an example

```
void Solver_CBFunction (pPList before, pPList after, void *data,
                        operationType type, pEntity ppp) {

  // Data can point to the object of type 'MAdLibInterface' for instance
  // (not used here).
  MAdLibInterface * mi = static_cast<MAdLibInterface *>(data);

  // The data id used to identify the data attached to mesh entities
  pMeshDataId dataId = MD_lookupMeshDataId("SolutionTag");

  // Do the right manipulation on data according to the mesh modification
  switch (type) {
  case MAd_ESPLIT: // Edge split case
    //   - 'before' contains the split edge (not deleted yet)
    //   - 'after'  contains the two new edges
    //   - 'ppp'    contains the new vertex
    {
      // find the edge to be deleted
      void * temp = NULL;
      pEdge pE = (pEdge) PList_next(before,&temp);

      // get coordinates and data at old nodes
      double data0 = 0.;
      pVertex pV0 = E_vertex((pEdge)pE, 0);
      int gotit0 = EN_getDataDbl((pEntity)pV0, dataId, &data0);

      double data1 = 0.;
      pVertex pV1 = E_vertex((pEdge)pE, 1);
      int gotit1 = EN_getDataDbl((pEntity)pV1, dataId, &data1);

      if ( !gotit0 || !gotit1) { error(...); }

      // interpolate the data at the new vertex (here linear interpolation)
      double t = E_linearParams(pE,(pVertex)ppp);
      double newData = (1.-t) * data0 + t * data1;

      // attach this data to the new vertex
      EN_attachDataDbl(ppp, dataId, newData);
    } break;
  case MAd_ECOLLAPSE: // Edge collapse case
    //   - 'before' contains the regions (3D) or faces (2D) of the cavity
    //                       before the edge collapse (not deleted yet)
    //   - 'after'  contains the regions (3D) or faces (2D) of the cavity
    //                       after the edge collapse
    //   - 'ppp'    contains the vertex to be deleted (not deleted yet)
    {
      // remove the data on deleted vertex
      EN_deleteData(ppp, dataId);
    } break;
  case MAd_FSWAP: // Face swap case
    //   - 'before' contains the regions of the cavity before the face swap
    //              (not deleted yet)
    //   - 'after'  contains the regions of the cavity after the face swap
    //   - 'ppp'    contains the swapped face (not deleted yet)
    {
      // nothing to be done here
    } break;
  case MAd_ESWAP: // Edge swap case
    //   - 'before' contains the regions (3D) or faces (2D) of the cavity
    //                       before the edge swap (not deleted yet)
    //   - 'after'  contains the regions (3D) or faces (2D) of the cavity
    //                       after the edge swap
    //   - 'ppp'    contains the swapped edge (not deleted yet)
    {
      // nothing to be done here
    } break;
  default: error(...);
  }
};
```