Parallel Mesh Adaptive Techniques for Complex Flow Simulation

THÈSE N^{O} 4163 (2008)

PRÉSENTÉE LE 19 SEPTEMBRE 2008

À LA FACULTE SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

LABORATOIRE D'INGÉNIERIE NUMÉRIQUE

PROGRAMME DOCTORAL EN MÉCANIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Angelo CASAGRANDE

M.Sc in maritime engineering science, University of Southampton, Royaume-Uni
et de nationalité italienne

acceptée sur proposition du jury:

Prof. J. R. Thome, président du jury
Dr P. Leyland, directrice de thèse
Prof. L. Formaggia, rapporteur
Dr S. Merazzi, rapporteur
Prof. A. Quarteroni, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2008

Abstract

Dynamic mesh adaptation on unstructured grids, by localised refinement and derefinement, is a very efficient tool for enhancing solution accuracy and optimise computational time. One of the major drawbacks however resides in the projection of the new nodes created, during the refinement process, onto the boundary surfaces. This can be addressed by the introduction of a library capable of handling geometric properties given by a CAD (Computer Aided Design) description. This is of particular interest also to enhance the adaptation module when the mesh is being smoothed, and hence moved, to then re-project it onto the surface of the exact geometry.

However, the above procedure is not always possibly due to either faulty or too complex designs, which require a higher level of complexity in the CAD library. It is therefore paramount to have a built-in algorithm able to place the new nodes, belonging to the boundary, closer to the geometric definition of it. Such a procedure is proposed in this work, based on the idea of interpolating subdivision.

In order to efficiently and effectively adapt a mesh to a solution field, the criteria used for the adaptation process needs to be as accurate as possible. Due to the nature of the solution, which is obtained by discretisation of a continuum model, numerical error is intrinsic in the calculation. A posteriori error estimation allows us to somewhat assess the accuracy by using the computed solution itself. In particular, an a posteriori error estimator based on the Zienkievicz Zhu model is introduced. This can be used in the adaptation procedure to refine the mesh in those areas where the local error exceeds a set tolerance, hence further increasing the accuracy of the solution in those regions during the next computational step. Variants of this error estimator have also been studied and implemented.

One of the important aspects of this project is the fact that the algorithmic concepts are developed *thinking parallel*, i.e. the algorithms take into account the possibility of multiprocessor implementation. Indeed these concepts require complex programming if one tries to *parallelise* them, once they have been devised serially. Another important and innovative aspect of this work is the consistency of the algorithms with parallel processor execution.

Keywords: mesh adaptation; computational geometry; computer aided design; interpolating subdivision; a posteriori error estimation; parallel computing.

Résumé

L'adaptation dynamique des maillages sur grilles non structurées, par moyen de raffinement et déraffinement localisé, est un moyen très efficace pour améliorer une solution numérique et pour optimiser le temps de calcul. Une des difficultés majeures de cette méthode est la projection des nouveaux nœuds, crés dans le processus de raffinement, sur les surfaces-frontière. Ceci peut être résolu avec l'introduction d'une librairie capable de gérer les propriétés géométriques données dans une description CAO (Conception Assistée par Ordinateur). Ceci est aussi intéressant pour améliorer le module d'adaptation quand le maillage est lissé, donc déplacé, pour pouvoir le reprojeter sur la surface de la géométrie exacte.

Cependant, la procédure ci-dessus n'est pas toujours possible à cause soit d'une géométrie avec des erreurs, soit d'un design trop complexe, ce qui demande un plus grand niveau de complexité de la librairie CAO. C'est donc primordial d'avoir un algorithme "built-in" capable de placer les nouveaux nœuds qui appartiennent à la frontière près de la définition géométrique de celle-ci. Une procédure de ce type, basée sur l'idée des subdivisions interpolantes, est proposée dans ce travail.

Pour pouvoir adapter efficacement un maillage au champ solution correspondant, les critères utilisés dans le processus d'adaptation doivent être les plus précis possibles. A cause de la nature de la solution, qui est obtenue par discrétisation d'un modèle continu, l'erreur numérique est intrinsèque au calcul. Une estimation a posteriori de l'erreur nous permet d'évaluer la précision en utilisant la solution elle-même. Plus en détail, un estimateur d'erreur a posteriori de type Zienkiewicz Zhu est introduit. Ce dernier peut être utilisé dans le processus d'adaptation pour raffiner le maillage dans les régions pour lesquelles l'erreur locale dépasse une certaine valeur, pour pouvoir augmenter la précision de la solution dans ces régions lors du pas de calcul suivant. Des variantes de cet estimateur d'erreur ont aussi été étudiées et mises en œuvre.

L'une des caractéristiques importantes de ce projet est le fait que l'algorithmique a été développée en prévoyant une utilisation en multiprocesseur du programme. En effet, ces concepts demandent un effort important de programmation pour pouvoir paralléliser un logiciel qui a été développé et conçu pour une exécution sérielle. Un autre aspect important de ce travail est la cohérence de l'algorithme avec l'exécution parallèle multiprocesseur.

Mots clés: maillage adaptatif; géométrie algorithmique; conception assistée par ordinateur; subdivision interpolée; estimation d'erreur a posteriori; calcul parallèle.

Acknowledgements

I would like to thank Dr. Pénélope Leyland for giving me this fantastic opportunity, and for encouraging me all along, as well as for the discussions on both work and places where to go skiing.

I would also like to thank Prof. Luca Formaggia for the brief but very fruitful discussions we had throughout the duration of this work.

Special thanks also to all the collegues I had the luck to meet throughout the years at the Laboratoire d'ingénierie numérique (LIN), for the coffee breaks, technical discussions and most of all their friendship.

Thanks also to all the guys at CMCS for all the mathematical discussions and their friendship throught these years in Lausanne.

Merci also to the Swiss National Science Foundation for the grant that made this research possible.

Of course I cannot forget the people whom I shared this adventure with more than others:
Matteo and Fabiano for the many, many sandwiches at Atlantide, and for the dinners, evenings, football matches (on television and... on the game console);
Riccardo, because sharing an office for ten hours a day is not so obvious if you don't get along.. luckily we do!
Michele, for the help with the SmartFish grids and for the many coffees and laughs, and Marco for the weekend lunches at EPFL.
All my friends here, in Milan, and everywhere, which includes anyone else I may have forgotten to mention!

Arianna for being there for me all the time, supporting and encouraging me, and Yoda for the early morning wakeups.

Last but not least my parents and all my family for the support throughout these years, even though I never really explained *exactly* what I was doing, now they can figure it out (hopefully)!

Contents

List of Tables

List of Figures

1 Introduction

The thesis aims at developing novel techniques for the solution of transient problems in physics and engineering by moving adaptive meshes. The methodologies developed here will be of a fundamental nature and extendible to different ranges of applications. Although the primary target will be transient aerodynamics or biological flow dynamics, the methodologies could be applied also to other classes of problems, such as fluid-structure interactions.

1.1 Why parallel adaptation?

Many concurrent compressible flow solvers for industrial problems are based on block structured grid methods, for their numerical robustness and simplifications of implementation for numerical schemes which require upwinding for instance. The grid generation is more tedious than unstructured methods, being less flexible. These constraints are being reduced more and more by more powerful, user-friendly tools, however, they remain at present complicated and time-consuming. For these reasons research teams and industry turn towards unstructured grid solvers which allow simpler automation of the initial grid generation process, and have the potential to achieve fast turnaround. Unfortunately, state of the art finite volume methods, which are typical for advection dominated situations, on unstructured grids can have disappointing accuracy and efficiency compared to the analogous methods on structured grids. Indeed, the accuracy of the standard methods may be hindered by irregularities in the mesh, which are often found in unstructured grids. Mesh adaption here plays an important role in ensuring that the mesh is correct, in a highly automated way.

Furthermore, present day research on solution techniques for transient phenomena is a very important subject, as the applications are widespread throughout the vast field of computational physics. In particular the present thesis concentrates on transport situa-

tions which are present everywhere within physics, engineering and biomedical applications. Firstly, in many engineering sectors such as the aeronautical, aerospace, aircraft components, motors, complex welding, as well as micromechanics and civil engineering which use simulation methods as part of the conception cycle. The integration of such computational methods allows also to consider accurate multidisciplinary design processes whereby coupling of heterogeneous phenomena can be taken into account, such as the coupling of the effect of flow on structural deformation, the effect of film cooling on near wall flow characteristics and the modified performance, unsteady effects in afterbody flow over aircraft and their effect on drag prediction for instance. Secondly, in biomedical flows, the comprehension of the flow phenomena and the effects of the moving boundaries are primordial to model and learn about the complex behaviour of the cardiovascular system. More and more multidisciplinary coupling procedures enter into the design cycle of economical, biomedical and engineering situations.

The present project will concentrate on methodologies for such applications in order to propose algorithms that are accurate and efficient.

1.2 State of the art

The development of mesh adaptation has seen many advancements since its first appearances in the early 1980s. Since then reviews have appeared on the subject to gather the most successful developments [1, 2, 3, 4]. These covered exhaustively the fundamental aspects of mesh adaptation, as well as other related aspects such as error estimation and parallelisation.
Here we give an overview of the different methods presented in literature in the last decade or so to place the contribution of the present work in context. In order to do so, the various aspects comprised in parallel mesh adaptation are treated separately, to follow the structure of the thesis which will be outlined at the end of this chapter.

1.2.1 Mesh adaptation

This aspect comprises many features, such as the techniques and criteria for adaptation, and the strategies employed for its achievement. The techniques for adapting the mesh are described in detail by Löhner [1] in one of the first reviews on the subject, and Mavriplis [2] in a report on mesh generation and adaptation during the same time period. These were picked up again in a later review by Baker [4]. These techniques, which fall into three main categories (grid movement, enrichment, solution enhancement), have been the basis for the adaptation process and remain practically unchanged since the first implementations. For

grid enrichment in particular, since it is the most widely used approach, various techniques have been proposed through the years [5, 6, 7, 8, 9, 10]. Most of these efforts concentrate on remeshing, either partial or complete, and interesting strategies for local remeshing. This last technique has many similarities with the subdivision enrichment technique when using optimisation techniques which locally remesh the grid, although this is usually done for internal elements and not boundary ones. However due to the possibility of remeshing the surface, the former type of remeshing needs the geometric definition of the surfaces, or alternatively the previous surface mesh.

The criteria to decide whether to adapt the mesh on the other hand have seen various methodologies being used. Once again the work by Löhner [1] illustrates the most popular indicators used at the time, which have not changed much in basic type but have evolved considerably along these lines since then. In particular a lot of work has been carried out on the subject of error estimation. Developments on error estimation are based on the work carried out by Babuvška and Rheinboldt [11], which introduced a residual based estimate of the error. Since then the procedures developed for error estimation may be divided into two mainstreams; the residual based just mentioned, which saw further development with important work from Ainsworth and Oden [12, 13] and others, and the recovery based approach. For further details on the background of the two methods we refer the reader to a recent paper by Zienkiewicz [14].
The work presented here is founded on recovery based a posteriori error estimation, which became increasingly popular in the mesh adaptation field after the introduction of a simple technique developed by Zienkiewicz and Zhu [15] for finite element approximations. This was further developed a few years later by the same authors [16, 17], which lead to one of the popular error estimators used up to date. Reviews by Verfürth [18] and Ainsworth & Oden [19] give a complete overview on the subject and are still the standard reference for any a posteriori error estimation publication. Further improvements were carried out by Zienkiewicz, Boroomand and Zhu [20, 21], and more recently by Möller and Kuzmin [22] and by Maisano et al. [23]. This last work introduces in particular the use of the recovered gradient to detect shock waves without over-refining the mesh as well as better localising elements.

Refinement and coarsening methods strongly depend on the adaptation technique that is employed. Whilst remeshing is associated to the problems encountered in grid generation, particular care is needed when refining and coarsening the existing mesh. Here the work by Kallinderis and Vajayan [24] introduced the concept of hierarchical mesh adaptation techniques, in which refinement leads to a nested procedure, whilst coarsening is only applied to previously refined meshes. Along the same timeline the initial work by

Richter [5] first, and that in collaboration with Leyland [25], introduces non-hierarchical mesh adaptation. This is the method used in this work, and although more complex in its algorithmic than background mesh techniques, it allows more freedom for mesh optimisation. Refinement techniques used here are common in literature and are based on the subdivision strategy developed in [26]. What is less common is the introduction of algorithms capable of preserving the boundary of the domain when adding nodes. A possibility to tackle this challenging problem was proposed by Weatherill [6] with the use of Hermite interpolation. Although this work was set originally for remeshing, and in a similar way the work by Löhner [27], it set the ground for the work that followed by Baker [28, 29], where the same technique is used for approximating the boundary and projecting the new mesh point to it. Coarsening on the other hand makes use principally of segment collapsing algorithms which have been widely developed through the years. The vast majority however uses a patch or a shell of elements around the edge to be collapsed, such as those proposed by Carey [30]. Some of these include local element reconstruction based on the Delaunay criteria [5] in the shell where the segment is collapsed, and an average positioning for the node resulting from the edge collapse based on a surrounding shell [31]. A similar procedure to this last one has been evaluated in a recent publication by Walter et al. [32], whilst Savoy [33] proposed the use of the shell curvature angle to decide what segment to collapse. This last concept is used here and will be described in greater detail in the chapter dedicated to mesh adaptation.

Mesh optimisation is generally linked with edge or face swapping, and smoothing. This topic has been thoroughly studied by Freitag and others [34, 35, 36] in recent years. Let us first begin by noting that the previously mentioned segment collapsing is also an optimisation technique. This can be carried out not only to coarsen the mesh, but related to an optimal number of neighbours for each node as suggested by Richter [5], and later by Carey [30].
Smoothing techniques rely on either Laplacian smoothing [34], or so called *spring analogy*, which was first introduced by Batina [37] and further developed in [38]. The approach followed here is the second, with amelioration to the original method carried out by Farhat et al. [39, 40] with the inclusion of torsional springs, and Savoy [33] with the inclusion of the influence of the number of node neighbours for each node as detailed in the chapter that follows. For details on the considerations of spring analogy we refer the reader to an investigation done by Blom [41].

Although the meshes used in this work are primarily isotropic, it is important to spend a few words on the topic of anisotropy in mesh refinement as it has been widely used and developed in the last decade. Here the idea is to capture strongly directional flows such

as boundary layers, and other directional features such as shocks by means of stretched elements. This allows to sensibly reduce the size of the mesh and consequently the number of degrees of freedom involved to obtain a solution. Some of the techniques to achieve this are resumed in the works of Formaggia and Selmin [42], Castro-Díaz et al. [43], Peraire and Morgan [44], and a series of papers by Habashi et al. [45, 46, 47, 48], where a more heuristic approach to the problem was used. Other techniques have been presented by Siebert[49], Apel [50], Formaggia and Perotto [51], where a more rigorous approach is used, based on error estimation. Along the same lines of this last technique are more recent publications by Formaggia et al. [52], Micheletti et al. [53, 54], Frey and Alauzet [55], and Picasso [56, 57], all of which have been very active in this field of research in the past years.

1.2.2 Geometry considerations

Since one of the aims of this work is to maintain the boundary description during the adaptation process both when there is a geometry description available and when there is only a computational mesh, we consider here some of the most relevant work that has been done in recent years on similar subjects.

Let us consider first the case where only a triangulated surface is available. Here once again the work of Löhner [27], Baker [28] and Weatherill [6] render interesting features, and in particular the approach described in Baker [29] seems very promising. Another promising work has been recently published by Persson et al. [58], whereby the use of Loop subdivision surfaces are used as a surrogate for the geometry description. This last idea shares the same principles to what is proposed in this work with the use of interpolating subdivision. This last has been profusely used in computer animation and much work has been presented on it by Zorin and Schröder [59, 60].
Yet another possibility that has been recently further developed by Krysl and Ortiz [61], and by Owen and White [62], is that of reconstructing a geometry from finite element meshes or surface triangulation.

When geometry description is available, the aim is to be able to use it during the refinement process to project the new nodes and therefore maintain the exact boundary. Due to the commercial value of such research topics it is problematic to find exhaustive publications on the matter (if any), and/or the tools used to implement it. A complete system for generating unstructured tetrahedral meshes, solving the equations of steady compressible inviscid flow on such meshes, including adaptation, and the necessary visualisation tools was developed by Peiró, Peraire and Morgan in the early 90s [63]. Whilst in the system just mentioned the adaptation is carried out by remeshing, the work carried

out in this thesis finds more similarities with the publications by Habashi et al. [48, 64] and by Merazzi et al. [65]. In order to achieve this, a Spline Library (SISL) [66], developed at the scandinavian research group SINTEF, was used. Some of the functionalities present in this library, that are used in the present work, are documented in the work by Dokken [67].

A different approach to maintaining an exact geometry is that proposed by Hughes [68, 69] in recent years. This method has many features in common with the finite element method and some features in common with meshless methods such as those presented by Löhner et al. [70, 71]. However, it is more geometrically based and takes inspiration from CAD. A primary goal is to be geometrically exact no matter how coarse the discretisation. Another goal is to simplify mesh refinement by eliminating the need for communication with the CAD geometry once the initial mesh is constructed. Yet another goal is to more tightly weave the mesh generation process within CAD.

1.2.3 Parallel aspects

Here the work by Karypis and Kumar [72, 73], Walshaw and Cross [74], and Bank and Smith [75] on mesh partitioning has been particularly important in the world of parallel computers. Much research has been carried out on parallel computing, and relative to the topic of mesh adaptation in particular. Some of the more noticeable developments on the subject have been done by Shephard et al. [3, 76], Leyland and Richter [77] which is the basis of this work, and Oliker et al. [78]. More recently the work on parallel adaptation has been extended also to unsteady flows, here it is worth noting the work by Waltz [79] and that by Park and Kwon [80].

1.2.4 Related research

Although not directly related to the work carried out here, some important research topics, that could offer interesting perspectives for either future implementations and/or technique advancements that could be applied here, are discussed. First we consider Hybrid techniques, where the aim is to use different type elements in the same mesh, to combine the advantages of both structured and unstructured grids approaches. For an introduction to the subject we refer the reader to Shaw [81], whilst for hybrid grids and adaptation Kallinderis et al. [82, 83, 84, 85, 86, 87, 88, 89, 90] and Mavriplis [91] have been particularly proficient in recent years.
Another interesting related subject is that of adjoint methods for partial differential equations (PDE). For a full review of the topic in detail the reader is referred to the work by Giles and Süli [92], as here only a very brief explanation is given. This subject is in fact strongly related to optimal control. The concept behind it is to consider the nonlinear

discrete equations written in residual form for the residual $R_h(U_h, g)$, where h denotes the discretisation, U_h the approximate solution, and g a design variable. Then by differentiating this with respect to the design variable g, this determines a change in the discrete solution U_h when the design variable is changed. Hence given a nonlinear objective function of these $J_h(U_h, g)$, and deriving with respect to the design variable, then

$$\left(\frac{\partial R_h}{\partial U_h}\right)^T V_h + \left(\frac{\partial J_h}{\partial U_h}\right)^T = 0 \ ,$$

where V_h is the solution of the adjoint equation.
This sets the basis for error analysis as developed by Becker and Rannacher [93, 94]. Following work by Giles [95, 96], in collaboration with Pierce [97, 98, 99], and others [100], set the steps for mesh adaptation using adjoint error analysis. Darmofal and Venditti [101, 102, 103, 104, 105] have been very prolific in recent years, and other developments have been made by Müller and Giles [106] and Park [107].
The importance of these methods relies on the fact that this approach corresponds to the functional analysis of the PDE system using duality, and hence tends to provide a mathematically "exact" estimation of the solution residual.

1.3 Thesis Outline

The present work is structured in the following chapters, including the present introduction:

Chapter 2 is dedicated to the introduction of the theory behind mesh adaptation. In particular the type of adaptation technique chosen is described, together with features implemented for refinement, coarsening. Geometrical and structural optimisation techniques are also outlined. Finally a few examples are shown to introduce the reader to the practical side of the problem.

Chapter 3 addresses the core aspect of this thesis, which is the conservation of the underlying geometry during the mesh adaptation procedure. A short theoretical background of the surface definitions used by the library introduced is given. This is followed by a detailed description of how the system is implemented and what this implies. Applications of the newly introduced feature are then shown by means of three test cases.

Chapter 4 deals with the approximation of the underlying geometry by means of interpolating subdivision. A short introduction on the basics of this technique begins the chapter, and the modified schemes for use with the adaptation process are presented. A guideline of the implementation of the schemes is then given, followed by practical examples with two test cases.

Chapter 5 contains the error estimators implemented in adaptation process. A detailed description of the approach is given for the two dimensional case, using triangles, and for the three dimensional one, with tetrahedra. The implementation of the error indicators is shown, with results for three practical examples shown.

Chapter 6 addresses the parallel aspects of the dynamic mesh adaptation in a message passing environment.

Chapter 7 provides a summary of the results obtained in this work. The general direction of the future work that could possibly follow on from what implemented here is also discussed, and conclusions are derived.

2

Principles of Adaptation

The accuracy of a numerical simulation is strongly dependant on the distribution of grid points in the computational domain. For this reason grid generation remains a topical task in CFD applications. Prior knowledge of the flow solution is usually required for a grid to be efficient, i.e. matching the features in the flow field with appropriate grid resolution. This however may not be available, requiring human intervention in analysing the results of an initial solution, going back to the pre-processing stage, and take an educated guess at how the mesh should be modified (fig. 2.1). Alternatively, a generally fine grid over most part of the domain is generated to obtain a relatively good solution. Both the above cases however require excessive time, effort and computational resources.

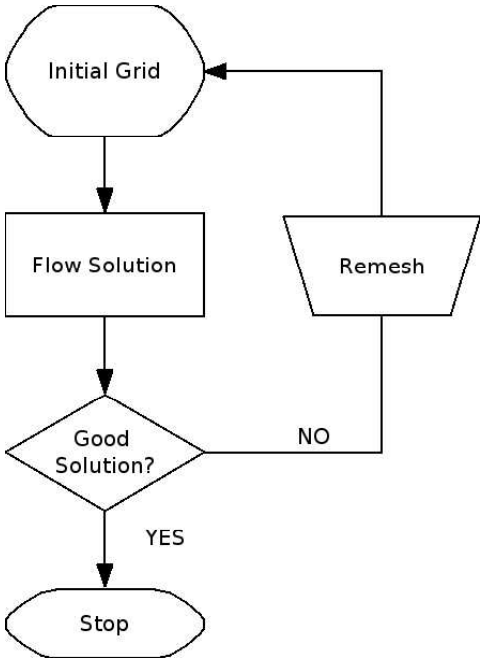


Figure 2.1: Manual adaptation process.

2.1 Adaptation techniques

Let us consider the case with manual intervention by the user. This step can be automated by adaptation, whereby the flow solution is analysed automatically, following some predefined criteria, and the grid resolution adjusted to the problem. The use of such techniques allows for computationally precise distribution of grid points (rather than eye-precision), and for extremely reduced user-intervention, thus addressing the time and effort issues. It also resolves problems related to computational time and costs, as the adapted grid can have fewer overall points, with similar resolution in areas of interest, than an unadapted fine mesh.

Adaptive methods can be grouped in three major types:

1. Grid movement, also known as r-refinement. In this case the mesh points and distribution are moved to follow the physics of the solution.

2. Grid enrichment (h-refinement). Here the density of grid points is increased in regions in order to minimise the space discretisation error.

3. Local solution enhancement (p-refinement). The precision of the solution is improved.

The first two techniques modify the grid by increasing the number of grid points in determined areas, and are the most widely used in CFD applications. The latter instead, increases the order of numerical approximation at locations of interest[7]. This powerful technique is related to solution adaptation and requires the underlying numerical scheme has a complete mathematical convergence theory. This is still an open question for many numerical schemes on finite element type meshes for non-linear parabolic and hyperbolic systems such as the compressible Euler and Navier-Stokes system of equations.

It is also possible and quite common to have combinations of the above techniques, which lead for example to hp-refinement and hr-refinement techniques.

2.1.1 Grid movement (r-refinement)

This technique consists in moving the grid points towards areas where a higher accuracy is required. As the name suggests, this is done without changing the topology of the mesh. Solution accuracy is also maintained, since no interpolation is needed, and can be of advantage when dealing with transient problems.

The main drawback with this approach is the fixed number of nodes in the mesh. Their

movement can hence cause severe distortions in the grid. This can be particularly problematic in the areas of the mesh close to the geometry, where this can be lost due to node movement, and geometry description can be no longer guaranteed.

There are however procedures to reduce these kind of drawbacks, which involve geometric criteria, and allow for better mesh regularity. This in turn guarantees no loss in geometry description, but usually strongly reduces node displacements, thus reducing the effectiveness of the method.

This technique has been mainly applied in structured meshes and 2D triangular meshes, in particular for moving and/or deforming bodies, and transient problems. In 2D cases, the conservation of geometry by explicit splines can be imposed.

2.1.2 Grid enrichment (h-refinement)

In this method the mesh topology is drastically changed, as nodes are added and removed in order to capture flow features and at the same time reduce the computational load in areas where the solution is sufficiently smooth. Therefore it is particularly suitable for unstructured grids, where the structure can undergo significant changes.

The criteria for refinement and de-refinement can be based on solution based criteria and/or error estimation criteria. Grid enrichment may be further divided into two main streams, grid remeshing and grid subdivision.

Grid remeshing

This consists in remeshing completely or partially the domain, based on the solution obtained. Grid generation time, and cost of solution interpolations, usually make this method expensive when the adaptation process requires several steps. This is particularly true with global remeshing, considering it is usually uncoupled from the flow solver, hence requiring user intervention. It has however the advantage of generating good quality grids, and generally being part of a CAD system, which allows for no loss of information on the bounding surfaces.

Grid subdivision

As the title suggests, here the grid is divided into smaller elements where necessary. New nodes are added to edges that are identified for refinement, and in turn the cells are divided. Therefore, it is easy to see how the use of unstructured grids can be particularly beneficial.

The advantage of this method with respect to the previous ones, is it's speed and efficiency. Drawbacks of this technique are the complex data structure, and most often, the lack of information of the underlying geometry on the bounding surfaces.

This technique can be approached in two different ways; with a hierarchical framework which saves parent-child relationship between cells at every step, and a non-hierarchical approach which discards the history of the original mesh during the filiation of successive grids. The method adopted here is completely non-hierarchical[5], since higher quality meshes can be achieved. This is due to the greater flexibility gained from the omission of the original macro mesh, which in turn allows the use of high performance structural optimisation algorithms. With the use of efficient (de-)refinement techniques, this approach is well suited for transient problems, or for producing coarse grids to be used in multigrid algorithms. In fact, the resulting grids are almost equivalent to those obtained by remeshing, with much less computational time required.

It is clear from the previous paragraphs how no single technique can satisfy all computational problems. A choice must be made depending on the type of problem and grid. Therefore the latter technique has been chosen in this work, as it suits well the type of problems studied herein. The drawbacks are taken into account and minimised by implementations that will be discussed later.

2.2 Adaptation criteria

Once the technique is chosen, another fundamental aspect is the criteria with which the mesh is adapted. The two main types of criteria considered here are mathematical and physical. The first can be further divided into two mainstreams, *a priori* and *a posteriori* error estimation. In all cases the criteria can be used for all kind of adaptation techniques and grids, with the appropriate modifications. There are also many ways of refining and coarsening meshes as stated in the introduction chapter.

2.2.1 Error estimation criteria

Although a more detailed description will be given later in the thesis about the method used, let us take a brief look at the main difference between the two criteria in order to understand the choice that has been done.

In the case of *a priori* error estimation the mathematics are applied directly to the system of equations. A first problem with this will clearly be that of the flow solver's source code

availability, as no commercial flow solver will be an option. The second drawback is that to be accurate this method is based on the discretised method, and supposes that both the continuous problem and the discretised one are well-posed. The third problem is that this estimation will be non-linear and difficult to constraint.

With *a posteriori* error estimation instead, no intervention at the level of the flow equations is needed, since it makes use of the solution field. This aspect together with the ease of implementation, and low computational cost, make it an ideal candidate for mesh adaptation. Since it makes use of only the solution, another important factor is that the method is rather independent of the problem and the governing equations[23], hence giving it much more flexibility than *a priori* error estimation.

Let us take as example the increase in accuracy of the solution process by locally enforcing the h-adaptivity using smaller discretisation elements. To fix the notation: u is the exact solution of the continuous problem,

$$\frac{\partial u}{\partial t} + Au = Lu = f \ , \tag{2.1}$$

where A and L are suitable differential operators and f the data of the problem. This is then approximated and solved iteratively on a computational mesh $\mathcal{T}_h = \bigcup_k T_k$:

$$\mathcal{D}_t^n\{u_h^n, u_h^{n-1}, \dots\} + \mathcal{A}_h(u_h^n) = \mathcal{L}_t^n\{u_h^n\} = f_h^n \ , \tag{2.2}$$

where $\mathcal{D}_t$ and $\mathcal{A}_h$ represent the numerical approximation operators in time and space defining the scheme. Time dependence will be dropped in the following notation as we describe h-adaptivity. This process tends to distribute uniformly the local error $\eta_h = u_h - u$ throughout the mesh, $\mathcal{T}_h$, where u_h is the computed solution.

In *a priori* finite element error estimation, we take a suitable norm of the discretisation error η_h and express it as a function of the mesh spacing and of the norm of the derivatives of the exact solution u [108]. The resulting error estimate would be of the form:

$$\|\eta_h\|_W \leq C_i h^m \|D^n u\|_Y \ . \tag{2.3}$$

Here C_i is an interpolation constant depending on the type of discretisation adopted but not on h, whilst $\|\cdot\|_W$ and $\|\cdot\|_Y$ are appropriate norms. D^n is a derivative of order n of function u, whilst m is an integer that is changed according to the chosen norms, the problem at hand and the type of discretisation used. When solving for the Poisson equation,

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{2.4}$$

we would have the following *a priori* error estimate:

$$\|u - u_h\|_{H^1(\Omega)} \leq C_i h |u|_{H^2(\Omega)} \leq C_i' \left[\sum_{T_k} h_k |u|^2_{H^2(T_k)} \right]^{1/2} , \tag{2.5}$$

where the element height h_k can then be adjusted as a function of $|u|^2_{H^2(T_k)}$, reducing it as the latter value increases. It is clear to see however how this type of error estimate is expressed in terms of u alone, and not u_h. Hence, although it can be used to predict the behaviour of the error norm as h gets smaller, it does not provide a useful error bound, since u is unknown, and is only (theoretically) the limit $h \to 0$ in some norm of u_h.

When making use of *a posteriori* error estimation, since u is clearly not available, the first step in a mesh adaptation algorithm of this type is to define the error estimates. These will be based on computable quantities, which well correspond to η_h. Therefore a typical *a posteriori* error estimate will is given by:

$$\|\eta_h\|_W \leq C_i h^l \|R_h(u_h)\|_Y , \tag{2.6}$$

where $R_h(u_h)$ represents the discrete residual $Au_h - f_h$, and $l > 0$ depends on th chosen norm, the problem at hand and the type of discretisation used. Hence for the Poisson equation we would get an estimate of the type:

$$\|u - u_h\|_0 \leq \alpha \|h^2 f_h\|_0 + \beta |D_h^2(u_h)|_0 \sim |h^2 R_h|_{L^2} , \tag{2.7}$$

where D_h^2 denotes the discrete second derivative.
Here the estimate provides an upper bound for the error η_h in terms of the discrete solution u_h, without any knowledge of the exact solution u needed. In contrast to the *a priori estimate*, here the error norm may be computed, however no prediction for the behaviour of the error is now possible, unless the behaviour of the discrete residual is known for h decreasing.

One of the possible ways of using a posteriori information on the calculated solution, to construct an adaptation criteria, is to mark out the zones to be modified in order to minimise global error and the number of mesh elements. Adaptation requires a local error estimate per mesh cell, $\eta(T_k)$, pondered by some tolerance levels:

$$\max_{T_k} \frac{\eta(T_k)}{\tilde{\eta}(T_k)} = \delta . \tag{2.8}$$

Here, $\tilde{\eta}(T_k)$ can be the average on the neighbours of T_k. If the ratio $\frac{\eta(T_k)}{\tilde{\eta}(T_k)} > \delta$, then the element T_k is to be refined[109].

2.2.2 Physical criteria

The physical adaptation criteria are based on flow quantities such as density, Mach number, pressure and entropy, as are also error estimators, but differ in the simplicity of their construction. In fact these use directly the physical quantities mentioned. A first method is to take the difference between the values at the nodes of a segment, and use its absolute value as an indicator for the adaptation process. Although this may seem as a very crude way of identifying flow features it is very effective applied to the grid enrichment method mentioned earlier. Another method employed is the undivided gradient along an edge [2]:

$$\varepsilon = \frac{\partial u}{\partial x}\, h \ , \tag{2.9}$$

which discretely can be written as:

$$\varepsilon = \Delta u \ . \tag{2.10}$$

From this it can be clearly seen that the value of ε, which should approximate the error, decreases as the mesh size h becomes smaller.
Various modifications to this method have been developed, such as inclusion of local mesh length scale:

$$\varepsilon = \Delta u \cdot \Delta x \ . \tag{2.11}$$

This leads to a more effective refinement criterion[110], as the simple form of the equation (2.10) remains approximately constant in the vicinity of shock waves, due to the steepened shock wave profile as the mesh is refined, and the jumps remaining relatively constant. The drawback is a heavier weight of larger cells than smaller ones because of the additional length scale, even in regions of smooth flow, leading to global refinement. Although these criteria have been successfully employed, they are not optimal. This is due to the tendency of excessively refining the mesh.

In fact the so called "physical" adaptation criteria correspond to exact mathematical error estimators. In [111], it is shown that for the linear advection diffusion problems, the evaluation of the jump of a characteristic variable across an edge is equivalent to the evaluation of the discrete H^1 norm of the solution. The relation between physical and mathematical criteria is therefore very close.

Two adaptation criteria used herein are based on physical criteria. Let us consider a solution field U that has been evaluated over the entire mesh for the selected criterion function[112]. A low- or high-pass filter is then applied on the solution field, which we will then denote as $\hat{U}$. For each segment the function $f = f(\hat{U})$ is computed, where $f(\hat{U})$ is one of the following:

- The difference of $\hat{U}$ between the vertices of the segment

$$|\hat{U}_a - \hat{U}_b|$$

- The gradient of $\hat{U}$ between the vertices of the segment

$$\frac{|\hat{U}_a - \hat{U}_b|}{l_{ab}}$$

where a and b are the end nodes of the segment and l_{ab} is the segments length. For the 2D case only, the choice also includes:

- The upwind flux of $\hat{U}$ through the segment
- The downwind flux of $\hat{U}$ through the segment

Further control on the field is obtained through the use of a filter F that removes part of the segments from the field. This can be applied in two ways, as an offset value, or as a cut-off value (fig. 2.2). In the first case each node of the mesh is examined and the following is applied:

$$\hat{U} = max(0, \hat{U} - F) \ .$$

In the second case the above changes to:

$$\hat{U} = min(F, \hat{U}) \ .$$

The refinement criterion is then built with $\bar{f}$, the mean value of f, over the grid. The segment i is then marked, for splitting in the case of refinement or for keeping in the case of the derefinement step, if:

$$f_i \geq \mathcal{F}\bar{f} \ ,$$

where $\mathcal{F}$ is a factor used to set the criterion as a function of the mean value $\bar{f}$ (fig. 2.3). In other words $\mathcal{F}$ is a coefficient that multiplies the mean value of the segment field, and all segments with higher values are marked.

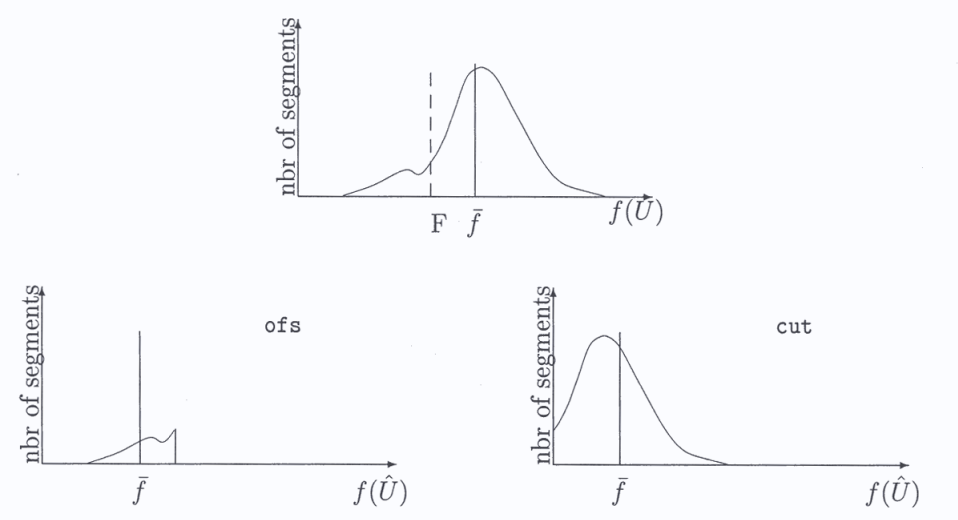


Figure 2.2: Graphical representation of the filter F and it's effect.

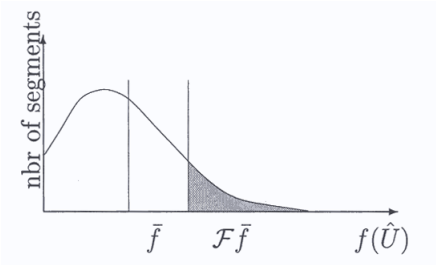


Figure 2.3: Graphical representation of factor $\mathcal{F}$ and $\bar{f}$ values used to limit segment marking.

2.3 Mesh refinement

Once the adaptation criteria is chosen and applied, and since the adaptation technique used here is grid enrichment with subdivision of mesh elements, the refinement step is performed. The finer grid obtained from this step will have elements nested in the parent (original grid) elements, and maintain the original vertices as a subset of its nodes. This allows for accurate and efficient transfer of variables from one grid to the next. This refinement procedure is not free from problems however, such as the generation of ill-conditioned elements. Therefore a certain logic and geometry rules must be followed in order to avoid rendering the mesh of little or no use.

2.3.1 Subdivision rules

Starting from the proposition that only triangular and tetrahedral grids will be used in this work, and that starting meshes will be as regular as possible, let us consider the symmetrical subdivision of a triangle first. This can be carried out either by adding a node in the middle of each side, or in the centre of the triangle. There are several reasons for picking the former over the latter, in particular the resulting triangles will tend to be as equilateral as possible. This avoids irregularities such as triangles with high aspect ratios. It also keeps a uniform distribution of the number of neighbours per node, which in turn helps increase the flow solver efficiency.

This leads us to the well known Red/Green subdivision approach, which addresses a first set of rules for the triangle subdivision. This kind of subdivision creates up to 3 new nodes n, and divides the original element in a maximum of $n + 1$ parts, as shown in figure 2.4. However, it is clear how the Green 2 type refinement could introduce irregular elements. Hence, in the case of an element with 2 edges marked for refinement, the third would also be marked to obtain a Red type element division. Since also the Green 1 subdivision type may create irregularities, some admissibility criteria must be applied. These are based on the number of nodes that may be added within the neighbouring elements, and a geometrical criterion based on the ratio of the smallest edge length to the others within an element.

This procedure is extended to tetrahedra in 3D, resulting in a maximum of 6 new nodes being created and the original element split in 8 parts at most (figure 2.5).

For further details regarding triangle subdivision, admissibility criteria and the refinement algorithm, see [5, 112].

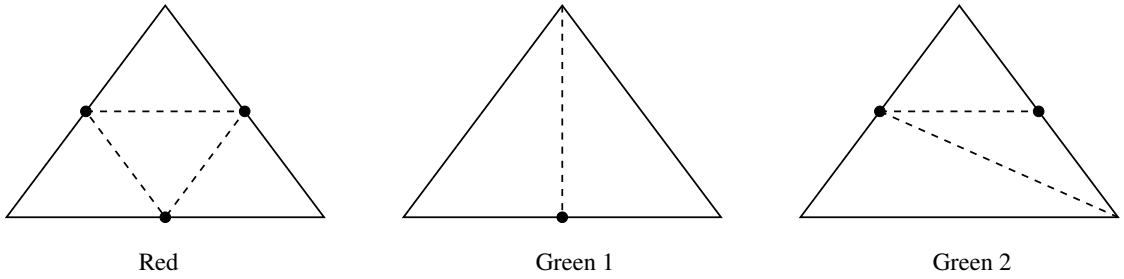


Figure 2.4: Red Green subdivision rule in two dimensions.

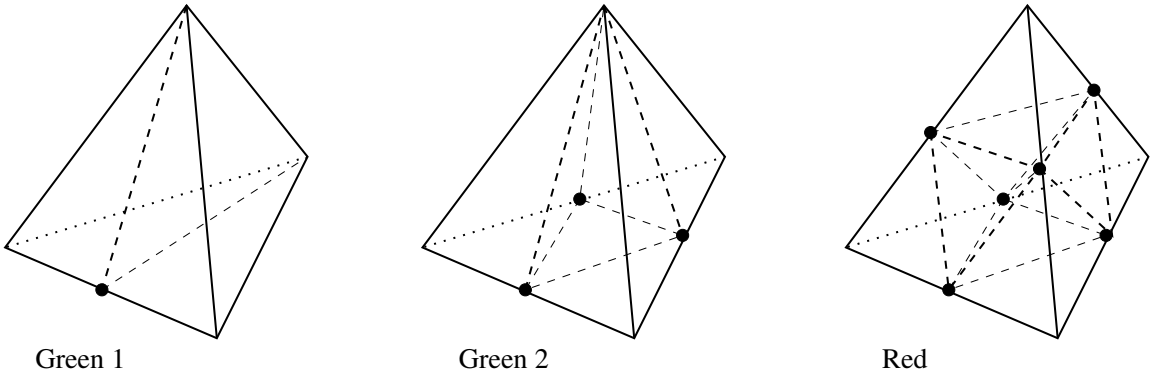


Figure 2.5: Three dimensional form of the element refinement.

2.4 Mesh coarsening

Although the technique is known as h-refinement or grid enrichment, it also involves the removal of vertices. This can be important for reducing the computational loads as mentioned in the introduction to the technique, and is essential in cases of transient flows. Here the use of non-hierarchical data structure mentioned earlier is particularly important as it allows deletion of nodes belonging to the initial mesh, and most importantly structural optimisation techniques that would be incompatible with conventional data structures.

Since the choice of points to be deleted is not straightforward, the inverse problem is solved by selecting the segments not to be removed, and fixing their nodes. This marking operation is carried out in the same way as for the refinement step but, being a distinct operation, doesn't necessarily need to have the same adaptation criteria, $f(\hat{U})$, $\mathcal{F}$, or F. There are sets of vertices however that must be fixed during this process, such as: symmetry points, nodes on edges marked for refinement, corners of the domain, nodes belonging to edges longer than a given reference. Once these nodes are blocked, an algorithm is applied to finalise the number of vertices that cannot be removed, in order to fix a maximal coarsening[5, 112]. In the 2D case this will result in at most four elements merging into one.

At this point of the coarsening procedure node deletion can be started. For the two-dimensional case a pre-processing step developed by Richter[5, 113] can be applied. This step is done principally because it reduces overall computing time. It consists in building a shell for each triangle with three marked vertices for deletion, consisting of the three neighbouring cells. Because of the maximal coarsening feature the three other nodes of this shell have to be kept (figure 2.6). All marked nodes belonging to one unique shell will remain during this pre-processing step. The shell is then considered as a new element, of type Green 1, 2 or Red, with respect to the number of marked nodes which have to be kept temporarily. The latter will be deleted during the next phase, which is done using a segment collapsing procedure which will now be described.

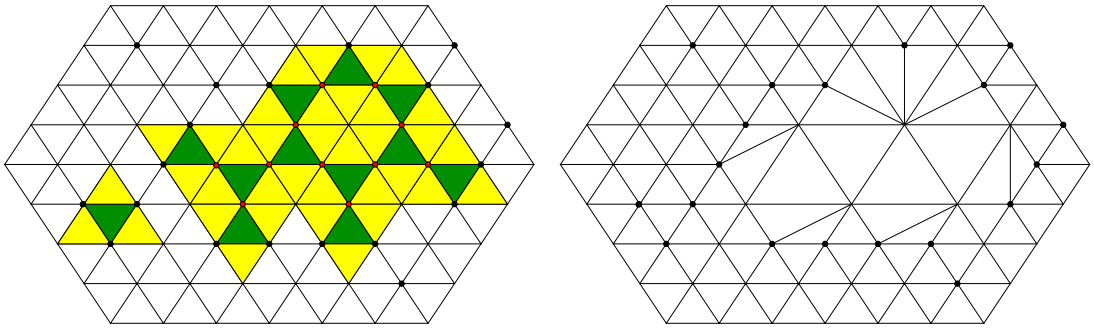


Figure 2.6: Two dimensional coarsening pre-processing step.

2.4.1 Segment collapsing

This method, developed by Savoy[112], consists in building a shell around the node marked for removal with it's surrounding elements. Let us consider the shell created around node 1 in figure 2.7(a). Vertex 2 will not be considered as it is also marked, whilst at all other vertices the inner curvature angle will be calculated. The next step consists in collapsing one of the inner segments in order to delete the centre node. The choice will fall onto the segment that connects the centre node to the neighbouring node with the greater angle associated to it, in this case α. Note that with this technique the risk of cell inversion (fig. 2.7(b)) and element distortion (fig. 2.7(c)) is minimised, resulting in improved mesh quality. However, shell volume conservation is also checked, in order to prevent accidental element inversion from happening.

The procedure works in both two and three dimensions, and is very efficient in the first case. Efficiency is somewhat reduced in the 3D case due to a large number of constraints

imposed during the marking, especially when avoiding element inversion, which in turn does not allow to remove a large amount of nodes.

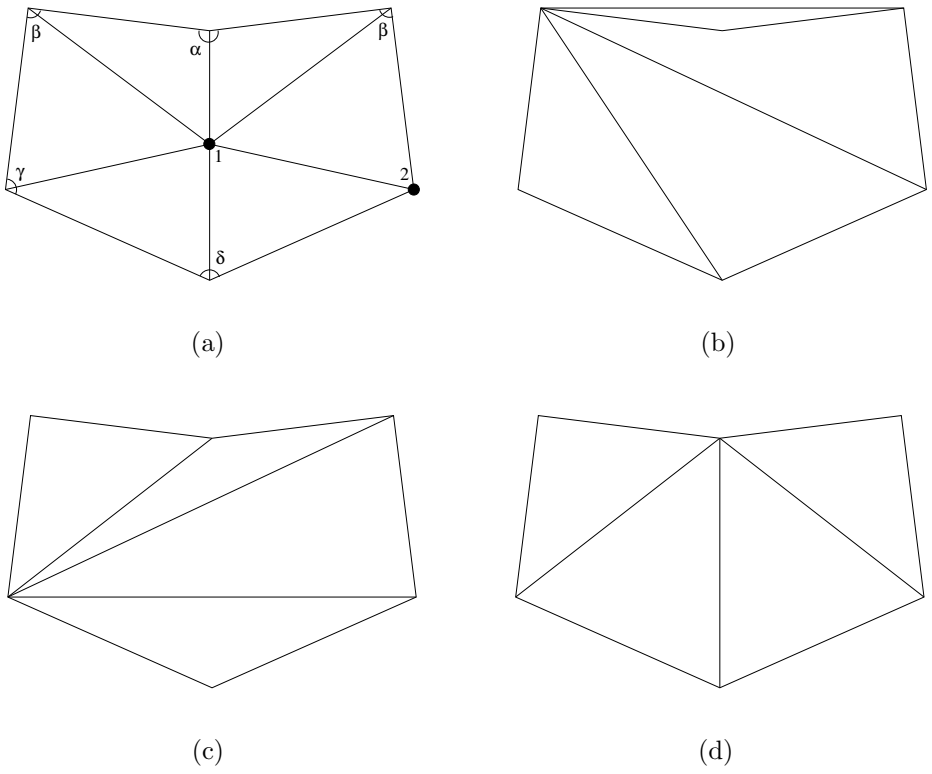


Figure 2.7: Segment collapsing with shell control: (a) initial shell, (b) shell inversion collapse, (c) collapse with element distortion, (d) best collapse available.

2.5 Optimisation techniques

Mesh quality and precision of the underlying discretisation is highly dependant on the shape of elements and shells just described. Therefore an equilibrium state would be desirable in the cells. This is achieved by equilateral triangles in 2D and equilateral type tetrahedra in 3D. However the mesh obtained after the refinement and coarsening steps will be far from this desired equilibrium state. This is due to the different local node density, and strong variations between element sizes and nodes angles. The number of node neighbours may also differ dramatically between vertices. In order to overcome these problems arising from the previous steps, the mesh must be optimised. This is done in several ways that may be grouped into two major strategies:

- Structural Optimisation
  - Diagonal swapping
  - Edge collapsing
- Geometrical Optimisation
  - Spring analogy
  - Boundary smoothing
  - Inverted elements

2.5.1 Structural optimisation

In this step the mesh is analysed and modified in function of the number of node neighbours $\mathcal{N}_i$. Following the Delaunay criterion [30], where the optimal element should be equilateral, $\mathcal{N}_{opt}$ is then related to the number of equilateral elements needed to fill the area around the node. In the two dimensional case $\mathcal{N}_{opt} = 6$ and can be easily calculated by considering $\pi/3$, in the Euclidean metric, as the optimal node angle. For the three dimensional case the spherical angle of the tetrahedron at each vertex is considered and the number of neighbouring elements calculated. The Euler-Descartes relation is then used to find the number of neighbouring nodes, leads to $13 < \mathcal{N}_{opt} < 14$ (for further details see [112, 114]).

Diagonal Swapping

This consists in swapping the internal edge of two neighbouring triangles, as shown in figure 2.8, for the two dimensional case.

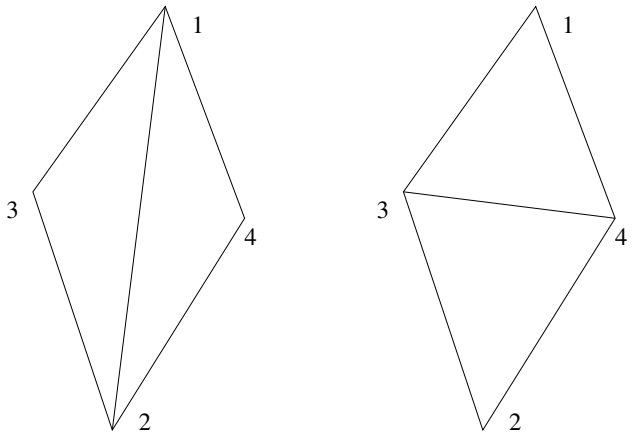


Figure 2.8: Two dimensional edge swapping.

The procedure is carried out to reduce the number of node neighbours $\mathcal{N}_i$ when this is greater than $\mathcal{N}_{opt}$. This is done by checking $\mathcal{N}_i$ on all vertices implicated in the operation. In particular the swapping is performed if the following conditions are satisfied:

$$\mathcal{N}_3 \ + \ \mathcal{N}_4 \ + \ 2 \ < \ \mathcal{N}_1 \ + \ \mathcal{N}_2 \ , \tag{2.12}$$

or

$$\begin{cases} \mathcal{N}_3 \ + \ \mathcal{N}_4 \ + \ 2 & = & \mathcal{N}_1 \ + \ \mathcal{N}_2 \\ \max\left(\mathcal{N}_3,\ \mathcal{N}_4\right) \ + \ 1 & < & \max\left(\mathcal{N}_1,\ \mathcal{N}_2\right) \ . \end{cases} \tag{2.13}$$

The three dimensional case requires more effort and attention, as the swap implies a face swapping, leading to complete remeshing of the shell built with the elements surrounding the deleted segment. The volume conservation must also be checked in order to avoid cell inversions during the shell remeshing. An example of a face swap is shown in figure 2.9.

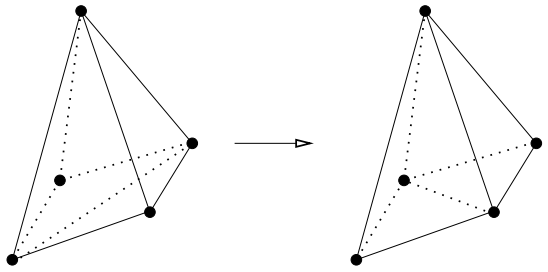


Figure 2.9: Three dimensional face swapping.

Edge Collapsing

This intervention is done when $\mathcal{N}_i < \mathcal{N}_{opt}$, and although the method is similar to the one shown in sec. 2.4.1, the scope is completely different. As for the swapping, the collapsing criteria are applied to the segments. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be the node neighbour numbers for the two vertices of the given segment, with $\mathcal{N}_1 \leq \mathcal{N}_2$. The collapsing is done by deleting the node which corresponds to $\mathcal{N}_1$. The collapsing is performed if:

$$\mathcal{N}_2' \ \leq \ \mathcal{N}_2 \qquad \text{or} \qquad \mathcal{N}_2' \ \leq \ \mathcal{N}_{opt} \ , \tag{2.14}$$

where $\mathcal{N}_2'$ is the node neighbours number resulting from the collapsing. It can be deduced from $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{M}$ the number of cells surrounding the segment:

$$\mathcal{N}_2' \ = \ \mathcal{N}_1 \ + \ \mathcal{N}_2 \ - \ \mathcal{M} \ - \ 2 \ . \tag{2.15}$$

These criteria are valid in both two and three dimensions. An example of edge collapsing in 2D is shown in figure 2.10.

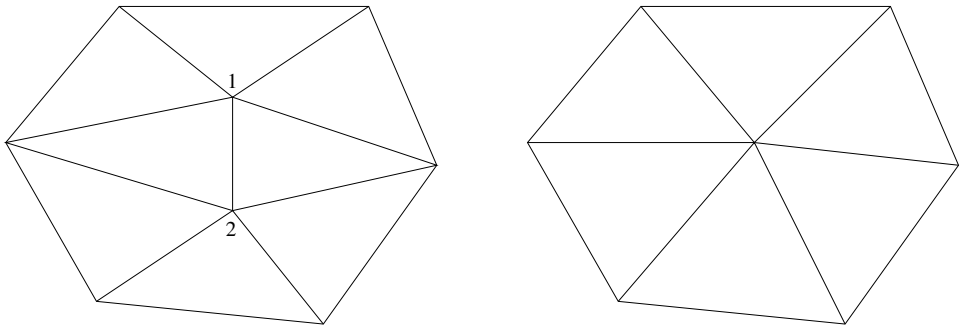


Figure 2.10: Edge collapsing in 2D.

2.5.2 Geometrical optimisation

The goal of this step is to modify the mesh without changes to the global data structure. This is achieved primarily by means of node displacement, based on spring analogy. However, other techniques must be applied to ensure a better handling of the node displacement. Node neighbours number for example will be employed again for adjusting the spring stiffness. Particular care will be given to nodes lying on the bounding geometry, and avoiding element inversion.

Spring Analogy

Here each segment in the mesh is replaced by an elastic spring (fig. 2.11). The objective is then to minimise the deforming energy of the overall elastic system. This will result in the force $\boldsymbol{F}$ at node i obtained using Hooke's law:

$$\boldsymbol{F}_i \;=\; \sum_{j \in k(i)} \alpha_{ij} (\boldsymbol{x}_j - \boldsymbol{x}_i) \;=\; \boldsymbol{0} \;. \tag{2.16}$$

Where $k(i)$ represents the set of node neighbours of vertex i, with size $\mathcal{N}_i$, and α_{ij} denotes the spring stiffness of the segment joining node i with neighbour j. Hence the equilibrium position at coordinates $\boldsymbol{x}_i$ can be expressed as:

$$\boldsymbol{x}_i \;=\; \frac{\displaystyle\sum_{j \in k(i)} \alpha_{ij}\, \boldsymbol{x}_j}{\displaystyle\sum_{j \in k(i)} \alpha_{ij}} \;, \tag{2.17}$$

which can be resolved using a Jacobi iterative scheme.

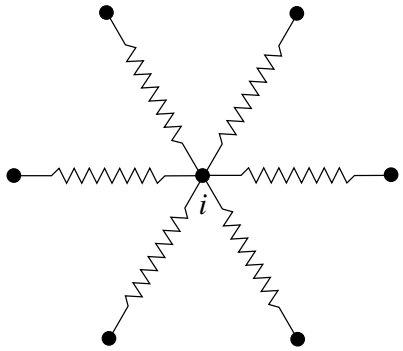


Figure 2.11: Spring analogy: springs replacing segments.

Spring Stiffness

Node neighbours number is once again very useful for mesh optimisation. In fact, if spring stiffness α_{ij} were to be set to one in order to produce equilateral elements, the following would occur:

- if $\mathcal{N}_i < \mathcal{N}_{opt}$, $k(i)$ move towards i (fig.2.12)
- if $\mathcal{N}_i > \mathcal{N}_{opt}$, $k(i)$ move away from i (fig.2.13)

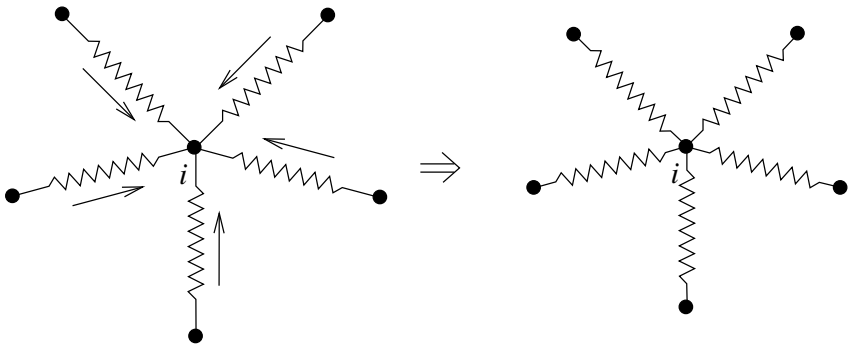


Figure 2.12: Springs movement based on node neighbours: springs contracting.

To partially avoid this problem the following weight function can be used to determine the spring stiffness:

$$\alpha_{ij} = \alpha_j = max\left[1, \mathcal{N}_{opt} + \mathcal{A}\left(\mathcal{N}_i - \mathcal{N}_{opt}\right)\right] \ . \tag{2.18}$$

This relates the spring stiffness to $\mathcal{N}_j$, the number of neighbours for the node $j \in k_i$. It also introduces the smoothing lineal factor $\mathcal{A}$, which is set manually.

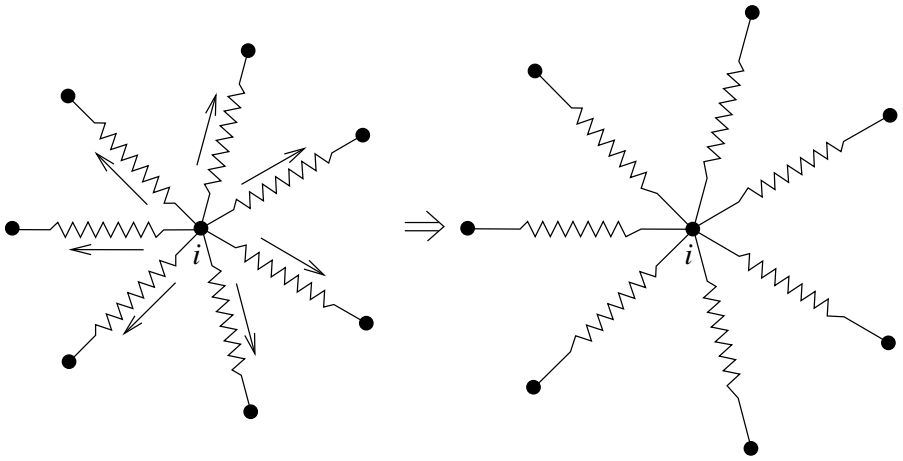


Figure 2.13: Springs movement based on node neighbours: springs expanding.

Boundary Nodes

Nodes lying on the geometric boundaries have to be moved with caution (if moved at all). Whatever the method used for positioning the node on the underlying geometry, a sufficient node density must be guaranteed within critical regions where the boundary curvature is large. This can be achieved by maintaining boundary nodes with a new spring joining the reference point $\tilde{\boldsymbol{x}}$ and the new position $\boldsymbol{x}^{n+1}$. The stiffness β_i of this new spring is then chosen as a function of the local maximum boundary curvature. The resulting force is then calculated as:

$$\boldsymbol{F}_i \;=\; \sum_{j \in k_\Gamma(i)} \alpha_j(\boldsymbol{x}_j - \boldsymbol{x}_i) \;+\; \beta_i(\tilde{\boldsymbol{x}}_i - \boldsymbol{x}_i) \;=\; \boldsymbol{0}\;, \tag{2.19}$$

where $k_\Gamma(i)$ represents the subset of $k(i)$ which contains all the node neighbours located on the boundary. The following formulation may then be obtained substituting $\tilde{\boldsymbol{x}}$ by $\boldsymbol{x}^n$:

$$\boldsymbol{x}_i^{n+1} \;=\; \frac{\displaystyle\sum_{j \in k_\Gamma(i)} \alpha_j\, \boldsymbol{x}_j^n + \beta_i\, \boldsymbol{x}_i^n}{\displaystyle\sum_{j \in k_\Gamma(i)} \alpha_j + \beta_i}\;. \tag{2.20}$$

The stiffness of the new spring is then defined as a function of the curvature angle ϕ. This angle is first filtered such that the node displacement is restricted, especially when it exceeds a given value Φ (fig. 2.14).

$$\beta_i \;=\; \mathcal{B}\left(\frac{1}{\cos^2\hat{\phi}} - 1\right) \qquad \text{with} \quad \mathcal{B} \geq 0\;, \tag{2.21}$$

where $\mathcal{B}$ is a user defined boundary stiffness factor.

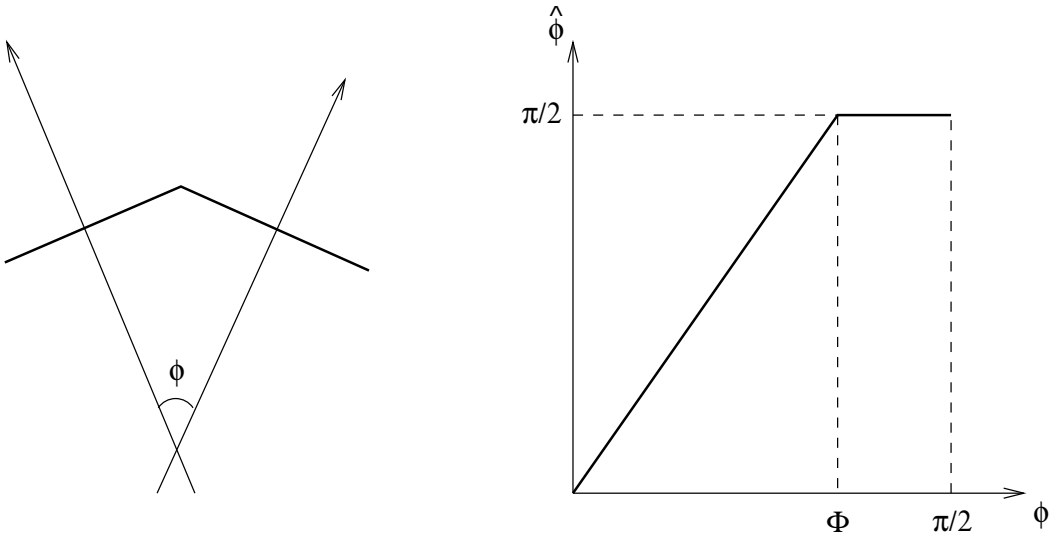


Figure 2.14: Curvature angle and filter.

Inverted Elements or torsional springs

This is a major issue which needs to be controlled thoroughly, as it causes loss in overall volume mesh conservation. It may occur when a vertex crosses over the opposite face of the element, which inverts the cell volume. This phenomenon, called *snap-through*, is shown in figure 2.15 and is prone to happening on the boundary when this moves. The configuration shown has a low energy as the springs a and b rotate. To remedy this the segment spring analogy is used together with initially rigid mesh boundaries, then semi-torsional springs are placed in the corner between adjacent edges, i.e. the stiffness of segment c is divided by the angle between segments a and b. As the sum of the angles is equal to π the stiffness is approximately unchanged if the triangle is equilateral. For deformed elements instead, the vertex angles that are closer to 0 or π become rigid.

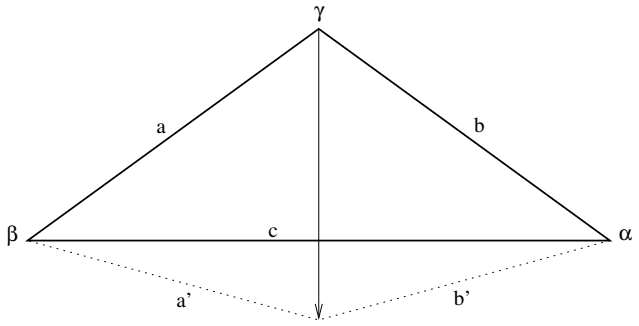


Figure 2.15: Inverted elements *snap-through*.

Cell inversion may also occur inside the heart of the mesh. A method to avoid this can be devised by setting critical cells rigid, with segment springs working in only one direction, rendering a relaxation of the elements. The vertex movement is then made free if it increases the element quality, which means that the introduced segment springs work like *stops* (fig. 2.16).

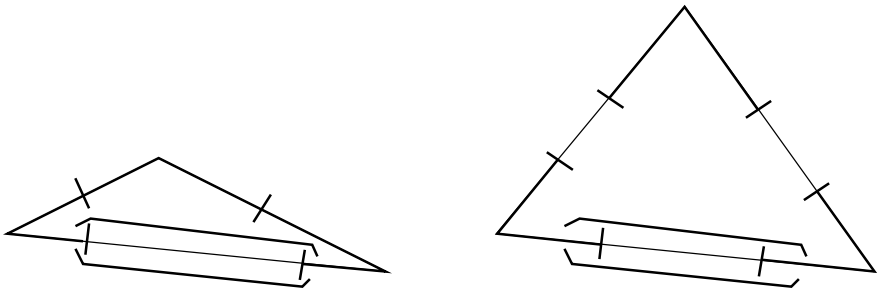


Figure 2.16: Cell inversion *stops*.

To determine the stiffness of the segment spring when it acts as a *stop*, the angular deformation energy of the cells is computed. A torsion spring is set at the opposite angle of each cell surrounding the segment (fig.2.17):

$$\mathcal{C} \;=\; \mathcal{C}(\frac{1}{\sin^2\theta} - 1) \qquad \text{if} \quad \sin^2\theta < \sin^2\Theta \;. \tag{2.22}$$

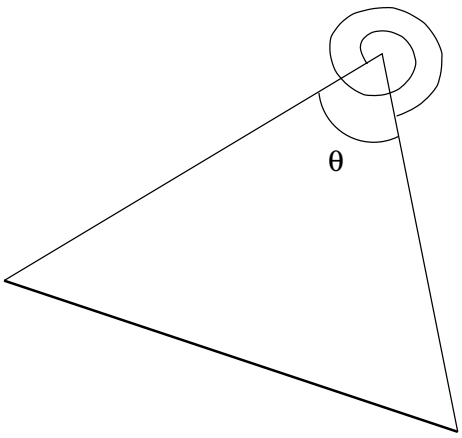


Figure 2.17: Torsional spring.

Where Θ is a filter value and $\mathcal{C}$ a user defined torsion stiffness factor. It allows to take into account only the most critical angles. The maximum of the torsion spring for a given segment is then converted to a segment spring using the following relation:

$$\gamma \;=\; \frac{1}{\delta}\mathcal{C} \;=\; \frac{\mathcal{C}}{\delta}\left(\frac{1}{\sin^2\theta} - 1\right) , \tag{2.23}$$

where δ is the distance between the segment and the opposite vertex in 2D and the opposite edge in 3D.

The non-isotropic behaviour of the *stops* causes the problem to be non-linear. A time advancing strategy must be implemented, with the *stops* relaxing during the evolution of the procedure. The force applied on the node i is then given by:

$$\boldsymbol{F}_i \;=\; \sum_{j\in k(i)} \alpha_j(\boldsymbol{x}_j - \boldsymbol{x}_i) \;+\; \sum_{j\in k(i)} \gamma_{ij}(\tilde{\boldsymbol{x}}_i - \boldsymbol{x}_i) \;=\; \boldsymbol{0} , \tag{2.24}$$

which leads to the following formulation:

$$\boldsymbol{x}_i^{n+1} \;=\; \frac{\displaystyle\sum_{j\in k(i)} \alpha_j\, \boldsymbol{x}_j^n + \sum_{j\in k(i)} \gamma_{ij}\, \boldsymbol{x}_i^n}{\displaystyle\sum_{j\in k(i)} \alpha_j + \sum_{j\in k(i)} \gamma_{ij}} . \tag{2.25}$$

Finally the effect of the torsion spring is shown in figure 2.18.

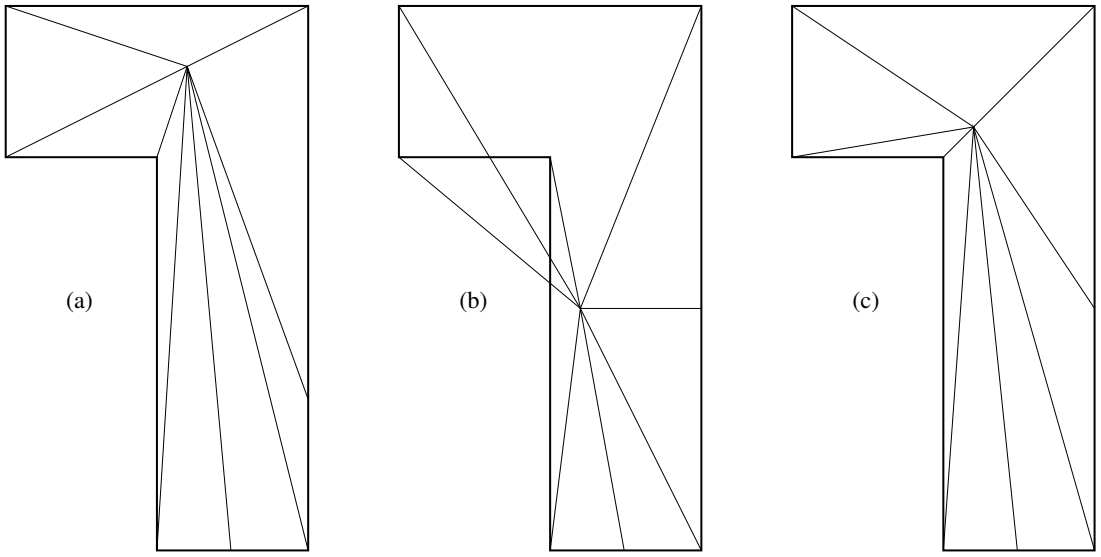


Figure 2.18: Torsion spring effect: (a) initial grid, (b) cell inversion, (c) torsion spring.

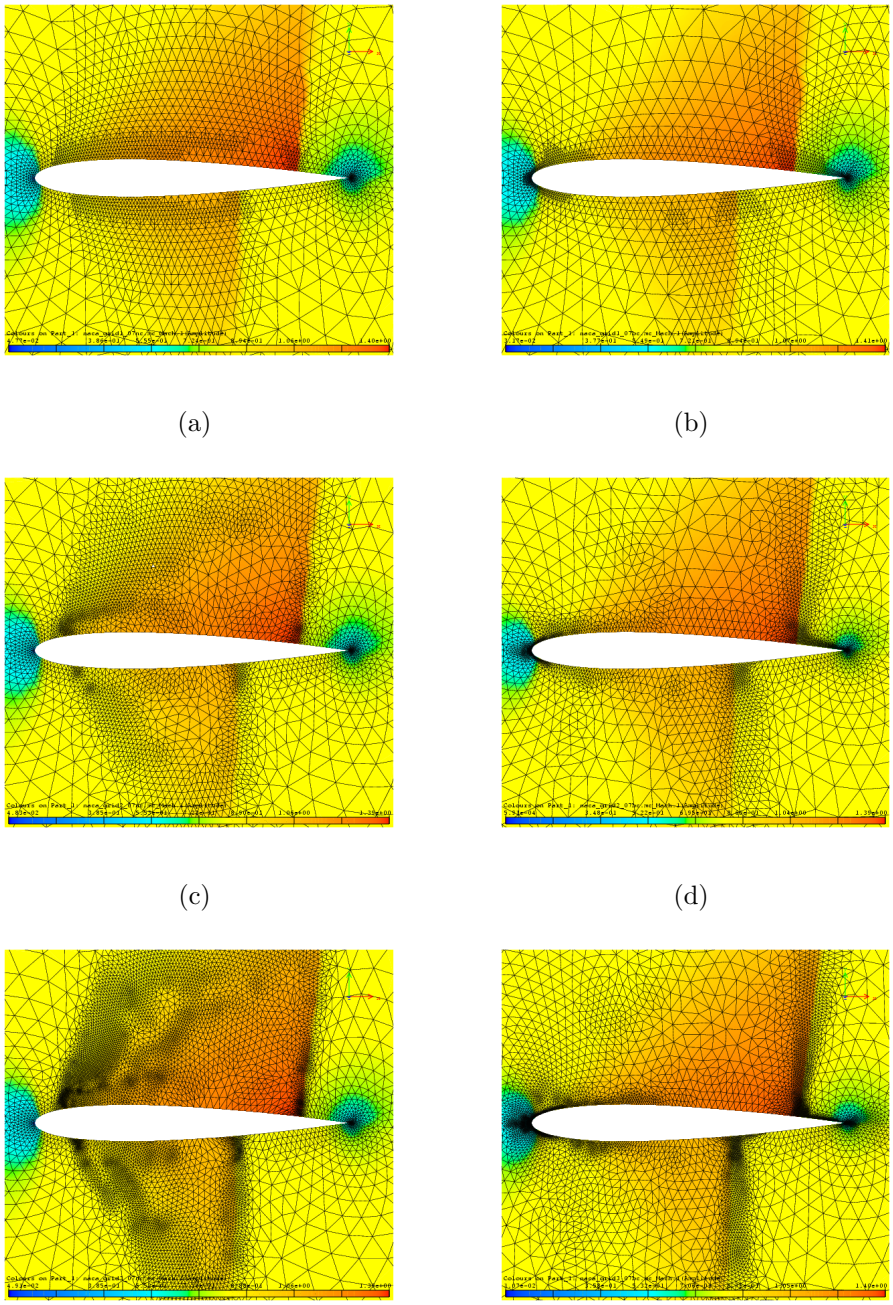
2.6 Applications

In order to assess the various functionalities of the techniques in place, a few test cases have been carried out in both two and three dimensions. For the two dimensional case a NACA 0012 airfoil is used. Instead for the three dimensional cases we consider a concept aircraft and a wedge. For all test cases a parallel, unstructured grid, Euler solver THOR[115] was used.

2.6.1 NACA 0012

This standard yet effective example, can give great insight on the capabilities of mesh adaptation. The grids and relative solutions presented here are obtained from an initial mesh of 2 355 nodes, 4 537 triangular elements, and 173 boundary faces. The solutions are refined with respect to the difference in Mach number along the edge, and by means of a more mathematical a posteriori error estimator that will be discussed later in the thesis. To compare the methods, the parameters were calibrated in such a way as to keep the number of elements and nodes close between the two, at each step of the process.

The cases were run at Mach number 0.85 and an angle of attack of 1°, so that shock waves would form. The solution is shown in figure 2.19 with: (a) and (b) a refined only mesh, with no smoothing, (c) and (d) refinement and derefinement and (e) and (f) final grids. For the final grid, refined with the difference scheme, we have 19 492 elements, 9 746 nodes, and 240 boundary faces, whilst for the error estimator 19 391 elements, 9 992 nodes, and 593 boundary faces. Further details of the computation are shown in figure 2.20.

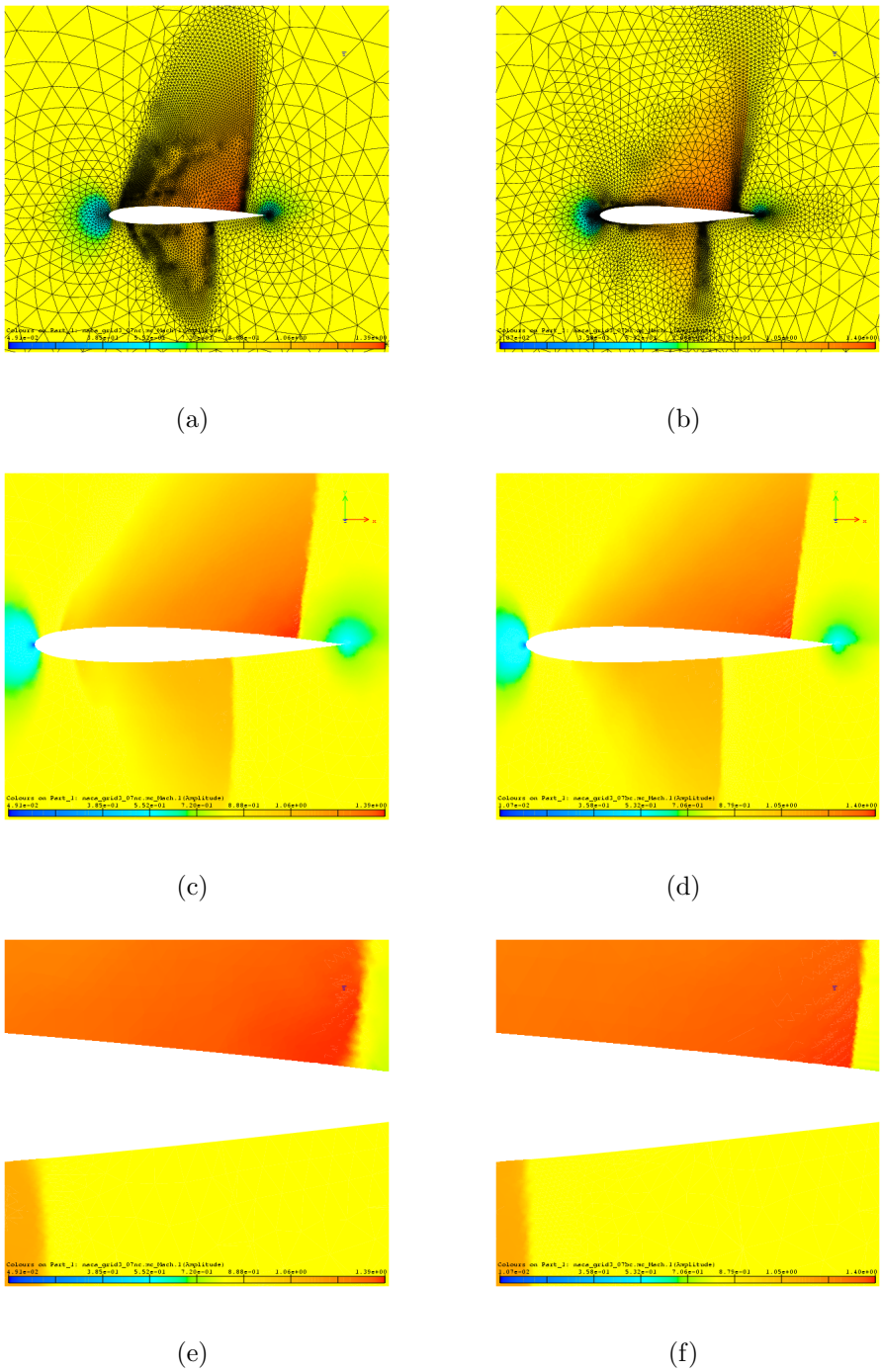


(a) (b)

(c) (d)

(e) (f)

Figure 2.19: NACA mesh for Mach number 0.85 and angle of attack 1°. Three adaptation cycles with respect to Mach: difference (left) and error estimation (right).



(a) (b)

(c) (d)

(e) (f)

Figure 2.20: Wider view of the final grids (a) and (b). Final solution fields (c) and (d). Closeup of the shocks at the boundary (e)and (f).

2.6.2 3D Wedge

The initial grid used for this test case is shown in figure 2.21, and is composed of 1 408 elements(NE), 325 nodes(NN), 448 boundary faces(BF). The solution is carried out for free-stream Mach number of 2.0, and adapted with respect to the density difference on the segment. The steps followed to obtain the final mesh and solution are shown in figures 2.22 to 2.26:

Figure 2.22 (a)Refined only mesh 7 136 NE, 1 387 NN, 1 328 BF and (b) and (c) relative solution, no smoothing;

Figure 2.23 (a)Refined only mesh 42 932 NE, 7 540 NN, 4 318 BF and (b) and (c) the solution obtained, smoothing turned on;

Figure 2.24 (a) Mesh after 1 step refinement, followed by 1 step derefinement 148 774 NE, 24 330 NN, 7 604 BF and (b) and (c) the solution obtained;

Figure 2.25 (a) Mesh after 1 step refinement, followed by 1 step derefinement 395 937 NE, 62 734 NN, 11 134 BF and (b) and (c) the solution obtained;

Figure 2.26 (a) Mesh after 1 step refinement, followed by 1 step derefinement 1 009 917 NE, 156 704 NN, 16 606 BF and (b) and (c) the solution obtained.

As expected, the solution is enhanced correctly by the grid refinements in the regions of the shocks.

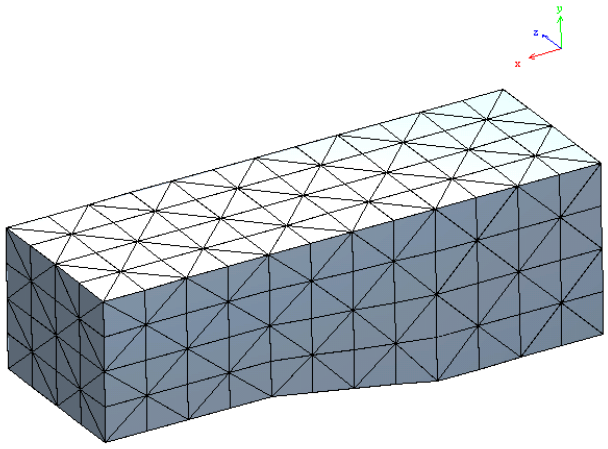
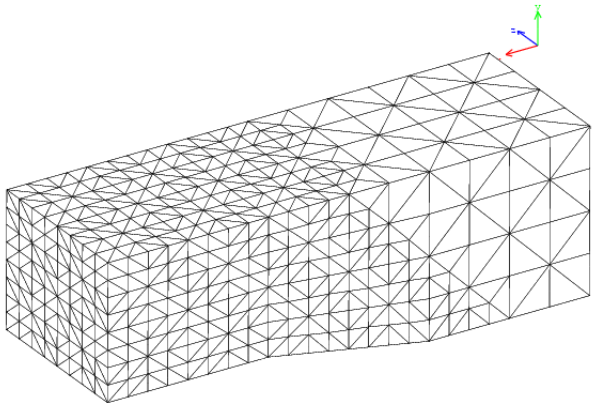
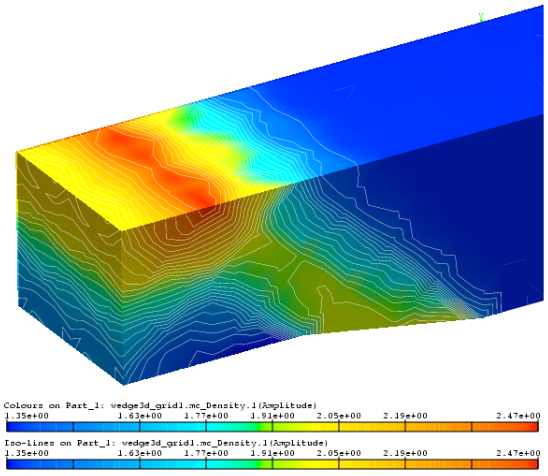


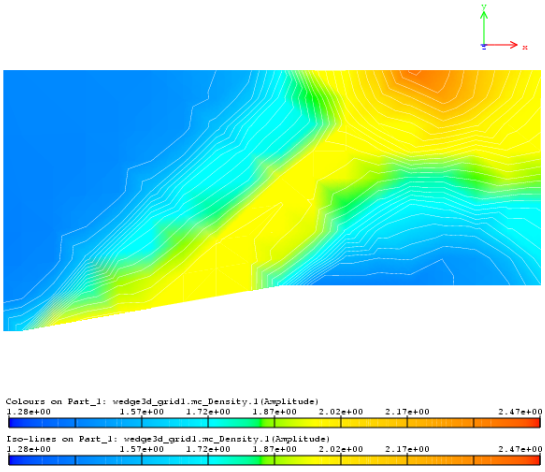
Figure 2.21: 3D Wedge initial mesh.



(a)

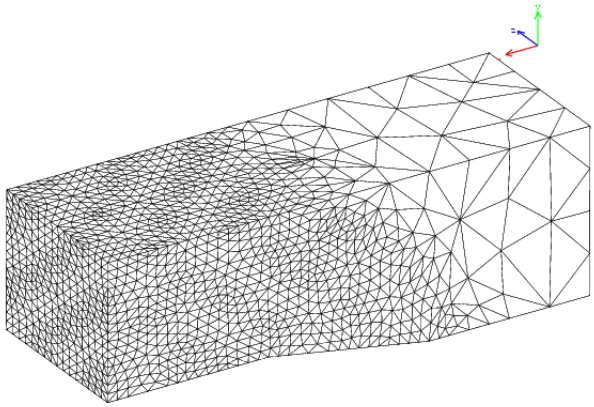


(b)

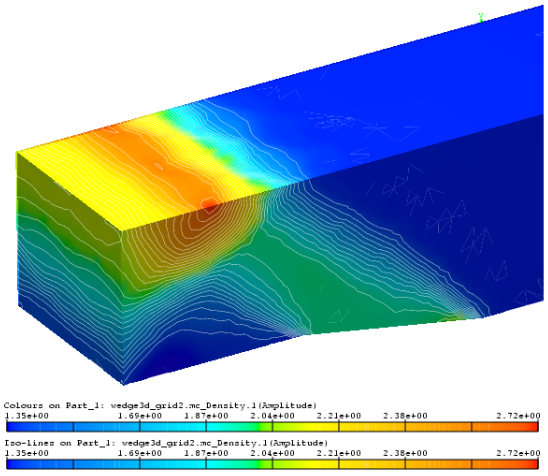


(c)

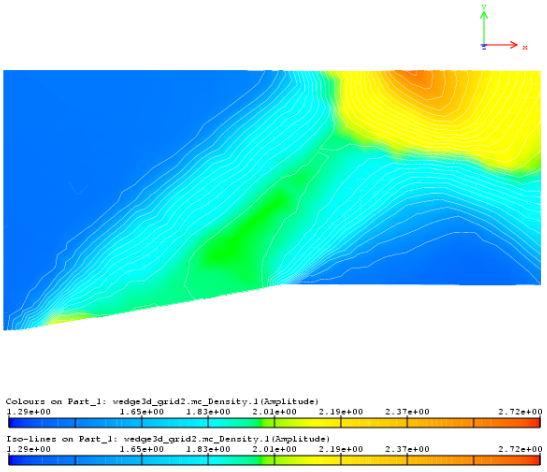
Figure 2.22: 3D wedge mesh and relative solution. 1 step adaptation (no smoothing).



(a)

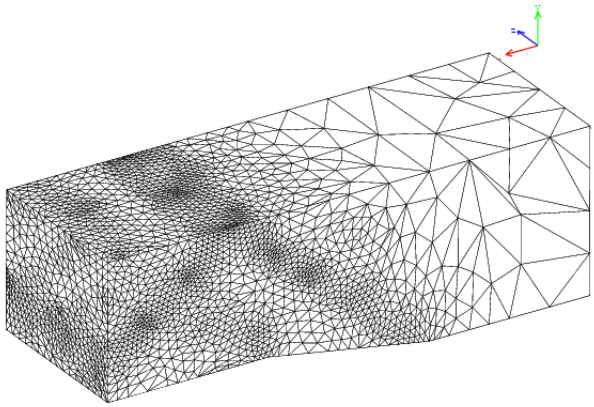


(b)

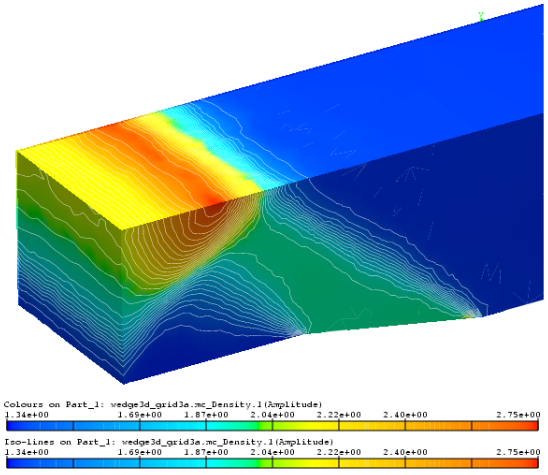


(c)

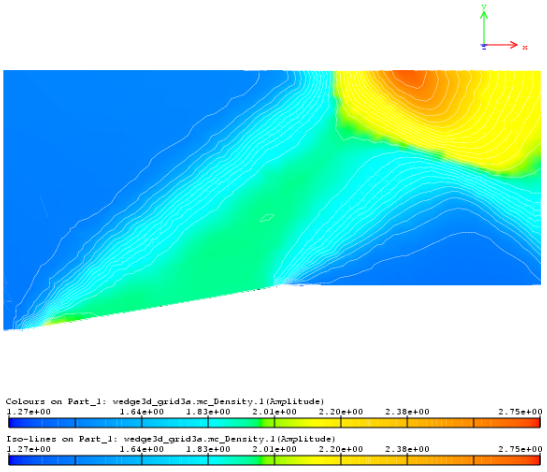
Figure 2.23: 3D wedge mesh and solution. 1 step adaptation (following from figure 2.22).



(a)

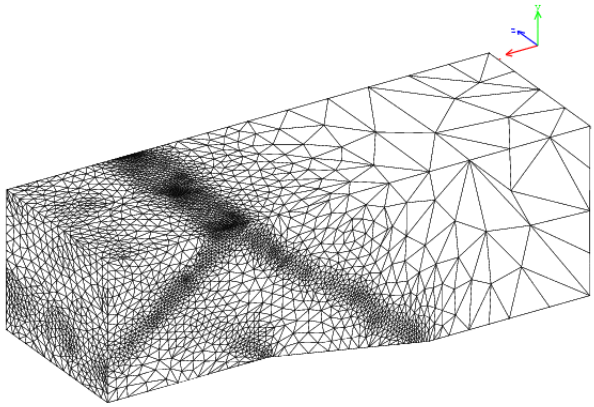


(b)

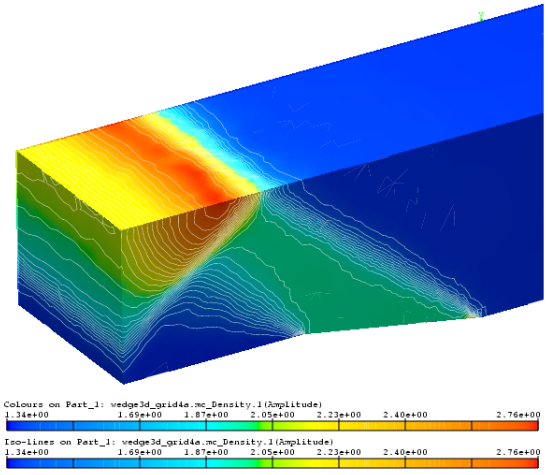


(c)

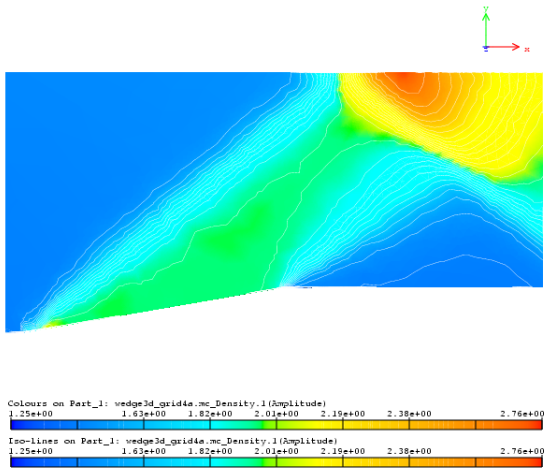
Figure 2.24: 3D wedge mesh and solution. 2 step adaptation (following from figure 2.23).



(a)

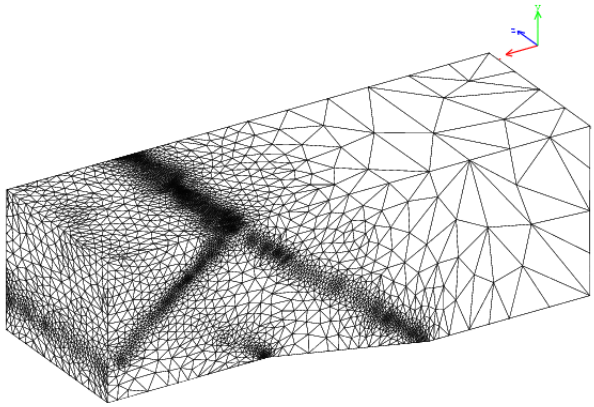


(b)

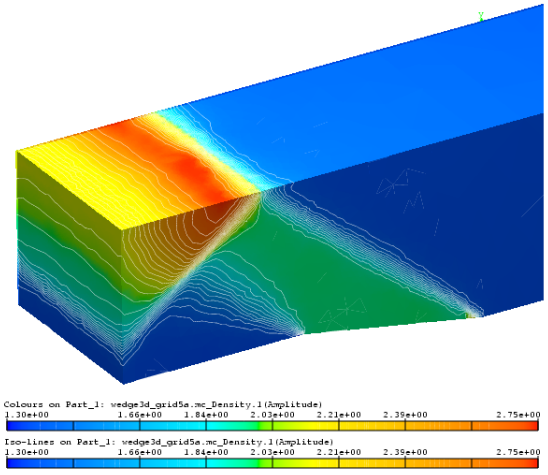


(c)

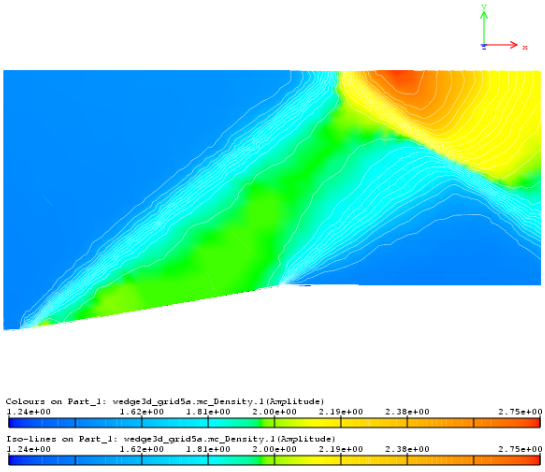
Figure 2.25: 3D wedge mesh and solution. 2 step adaptation (following from figure 2.24).



(a)



(b)



(c)

Figure 2.26: 3D wedge mesh and solution. 2 step adaptation (following from figure 2.25).

2.6.3 Concept aircraft

The second, three dimensional test case is represented by a concept aircraft, SmartFish[1] shown in figure 2.27. The interest of this geometry in this work is the extremely changing and complex form of the airplane, which poses a challenge for the grid generation and adaptation.

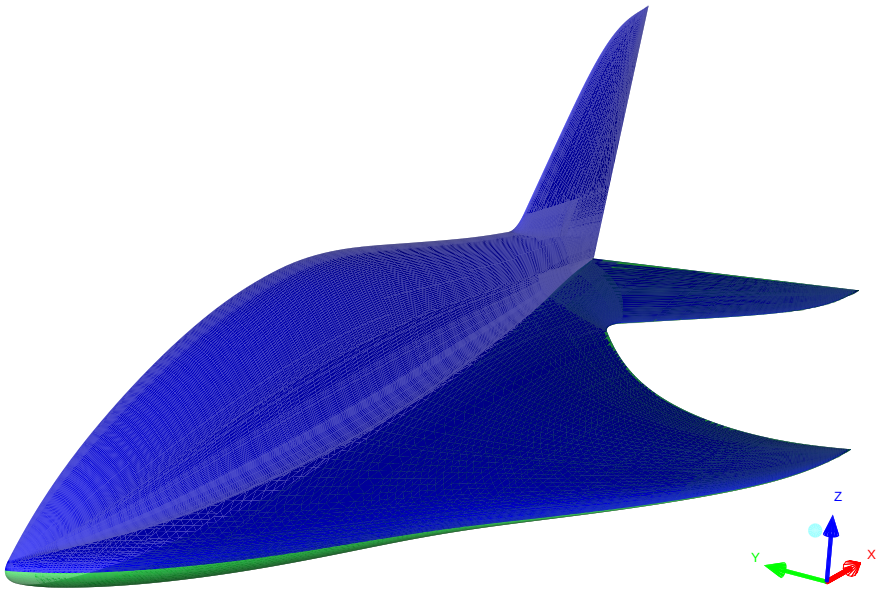


Figure 2.27: Concept airplane SmartFish.

Here we present the results of some adaptations with different initial grid sizes, and adapted with different physical criteria. The tests are carried out at transonic Mach numbers and with non zero angles of attack. In particular we first test a very coarse grid for this type of problem (fig. 2.28). The first adaptation is done with respect to the change in gradient of the Mach number, with two adaptation cycles. The mesh is heavily refined (fig. 2.29) but only along the leading edge and not much over the wing.

In the second case (fig. 2.30) the difference in Mach number along a segment is con-

[1] http://www.smartfish.ch/

sidered. With only one adaptation cycle, and derefinement with respect to Mach gradient, the grid is refined mostly over a large portion of the wing. This case however was run with a higher Mach number and angle of attack, and is known to have a shock on both upper and lower side of the wing. Relative solutions are shown in figure 2.31.

Moving onto a denser initial grid (fig. 2.32), the previous conditions are considered for the difference adaptation case. Here the adaptation gives a better result (fig. 2.33), mainly because of the better solution to which it was adapted, due to the finer starting mesh. The solution also confirms this as shown in figure (fig. 2.34).

Finally a relatively fine grid was used to start the process (fig. 2.35). The initial conditions are of Mach number 0.9 and angle of attack of 4^o. The grid is refined well in the area of the shock, above and below, as shown in figure 2.36.

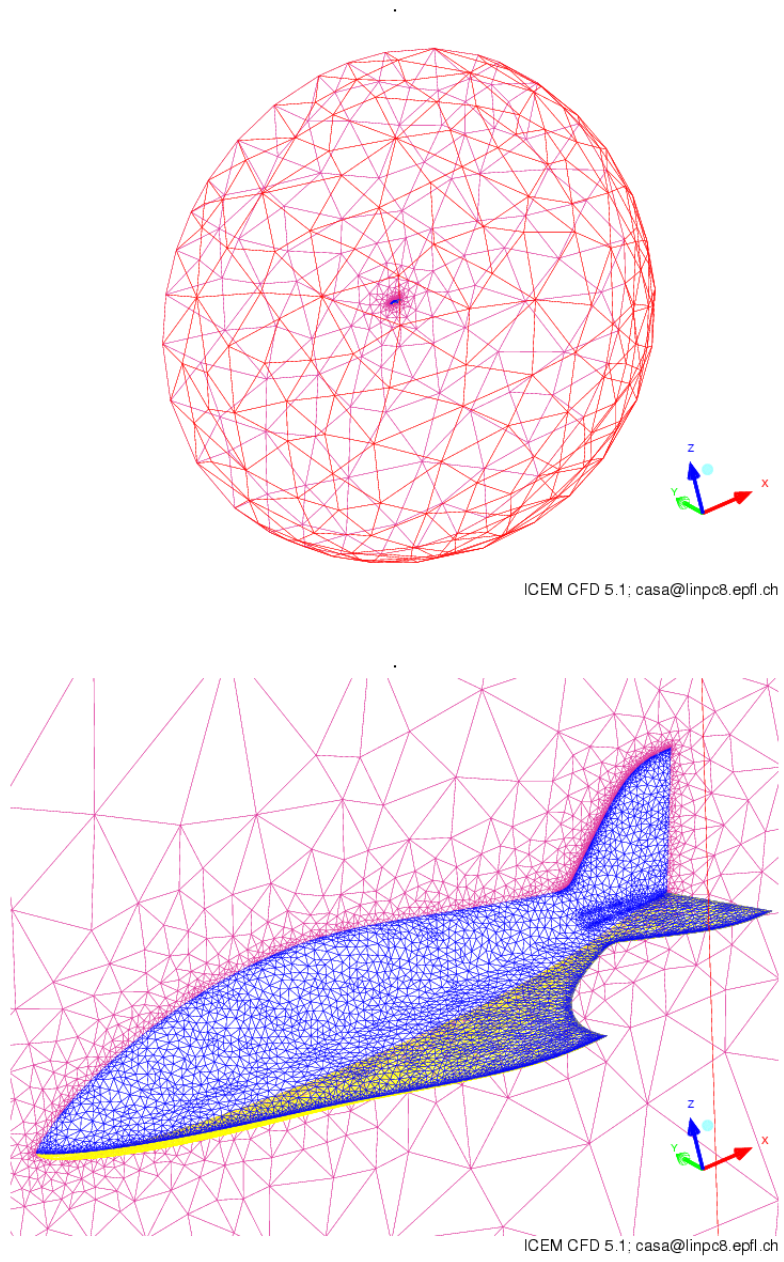


Figure 2.28: Far and closer views of coarser grid with 274 899 elements and 48 481 nodes.

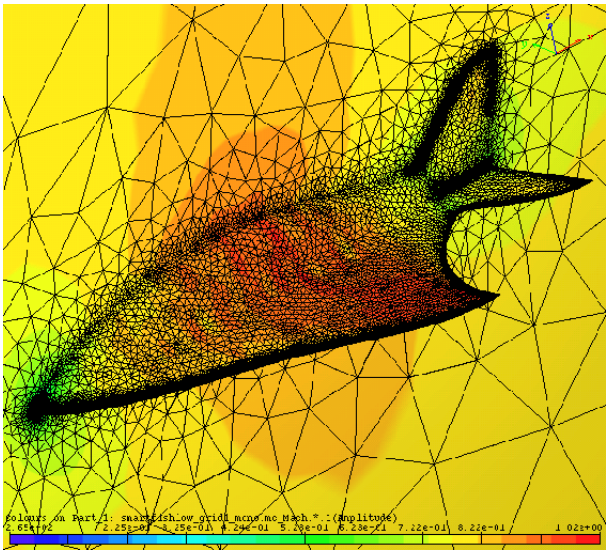


Figure 2.29: Adapted coarse grid with respect to Mach gradient. 2 adaptation cycles only with refinement at Mach number 0.8 and angle of attack 2^o, 6 569 277 elements and 1 098 081 nodes.

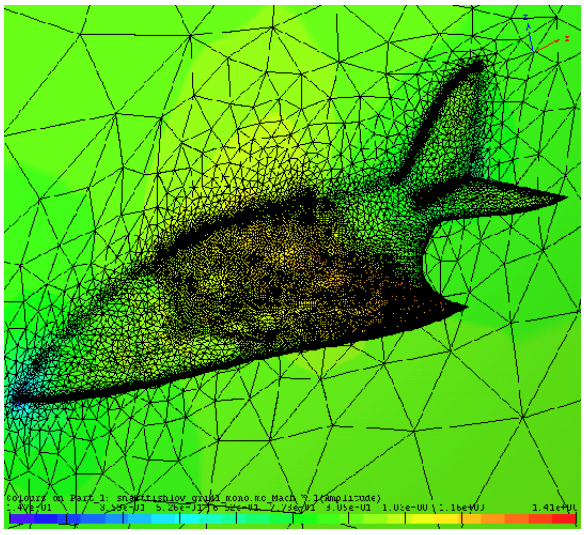


Figure 2.30: Adapted coarse grid with respect to Mach difference. 1 adaptation cycle with refinement and derefinement (with respect to Mach gradient) at Mach number 0.9 and angle of attack 4^o, 624 637 elements and 105 645 nodes.

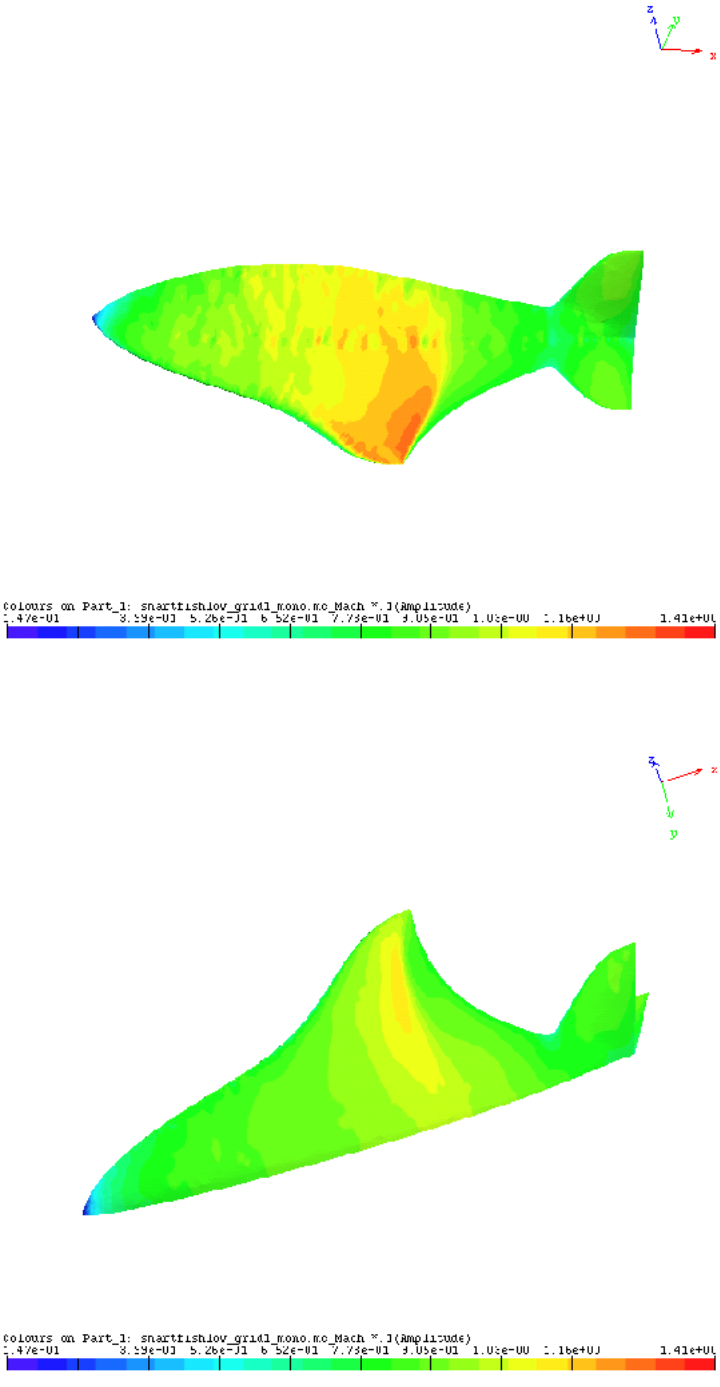


Figure 2.31: Mach number on upper and lower sides of Smartfish, from results obtained with the grid shown in figure 2.30.

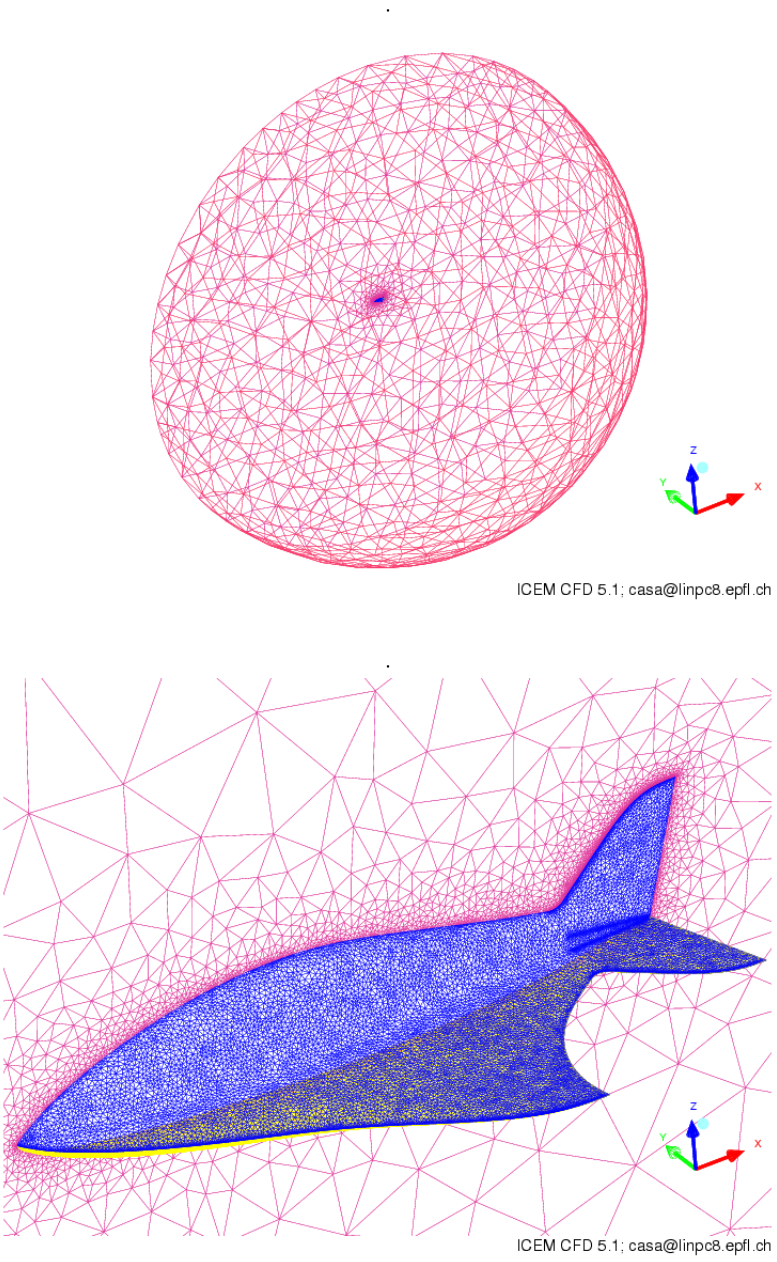


Figure 2.32: Far and closer views of medium grid with 742 294 elements and 129 865 nodes.

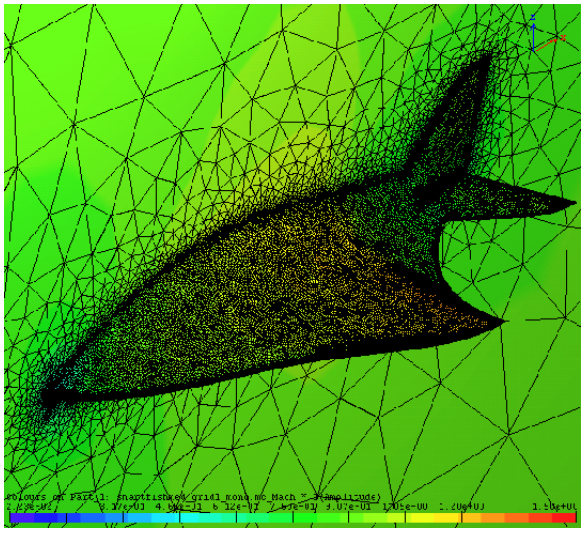


Figure 2.33: Adapted coarse grid with respect to Mach difference. 1 adaptation cycle with refinement and derefinement at Mach number 0.9 and angle of attack 4^o, 1 795 794 elements and 302 723 nodes.

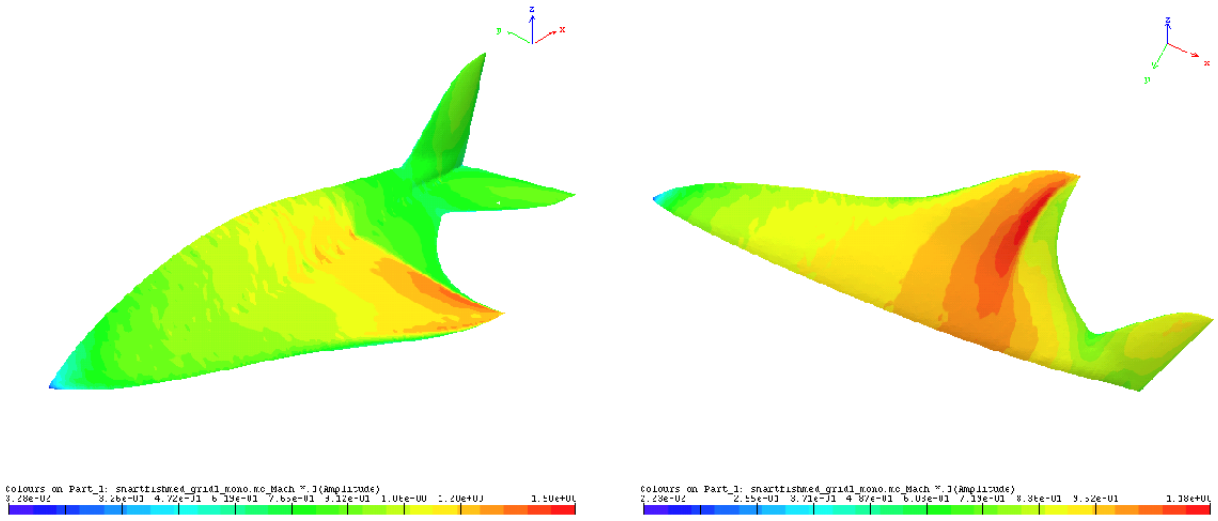


Figure 2.34: Mach number on upper and lower sides of Smartfish, from results obtained with the grid shown in figure 2.33.

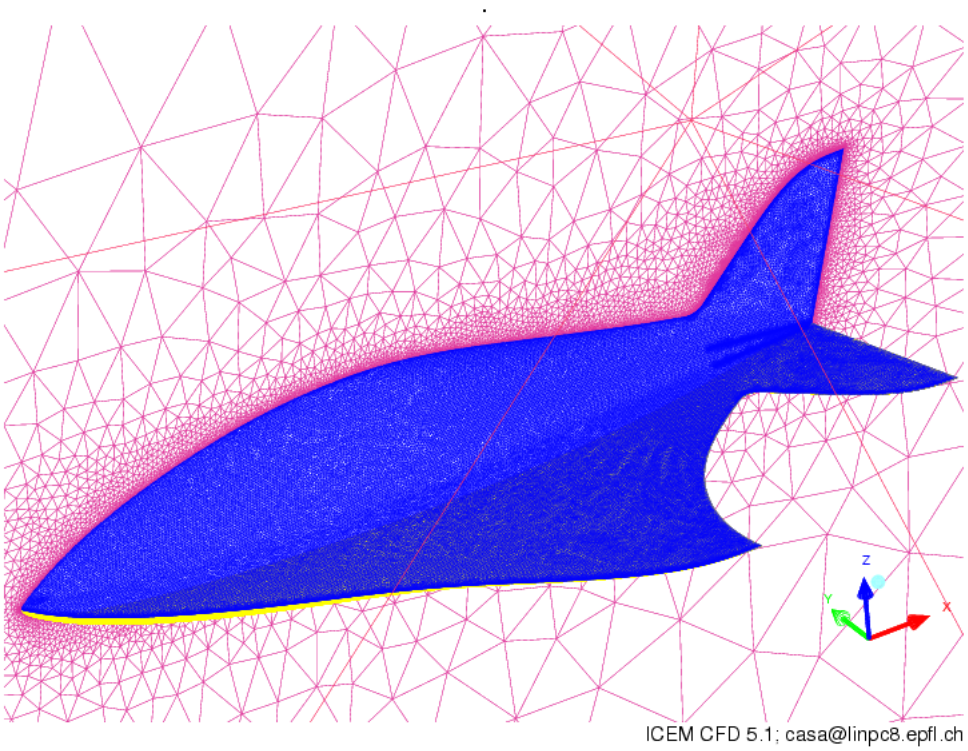


Figure 2.35: Close view of unadapted finer grid with 1 772 861 elements and 314 913 nodes.

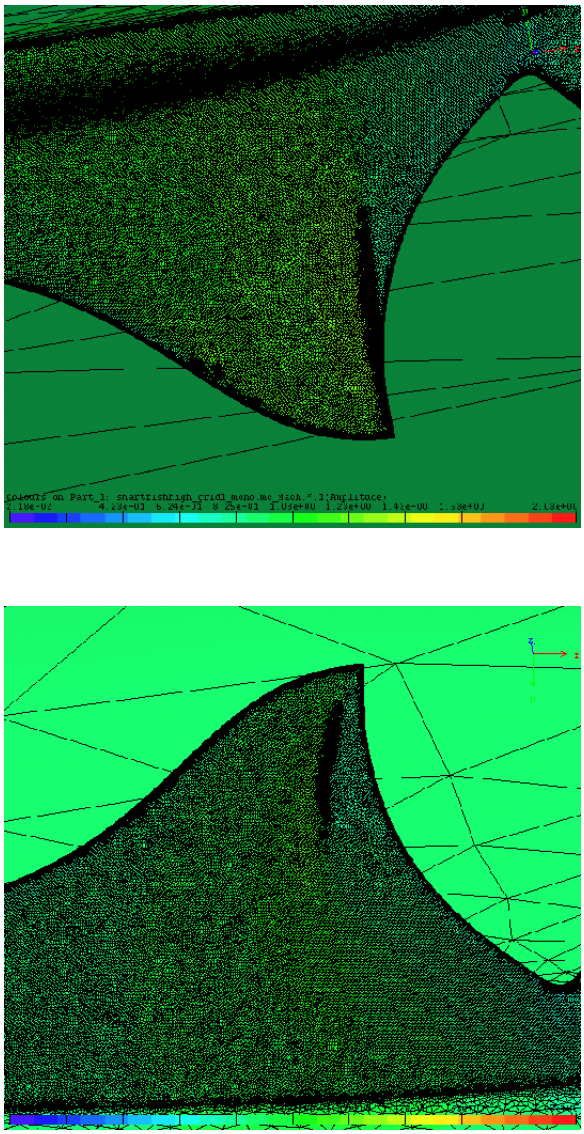


Figure 2.36: Adapted finer grid with respect to Mach difference. 4 adaptation cycles at Mach number 0.9 and angle of attack 4^o, 3 303 715 elements and 555 347 nodes. Upper side of the wing and lower side.

3

Geometry Conservation

One of the major drawbacks in most adaptation techniques (sec. 2.1) resides in the projection of the new nodes created, during the refinement process, onto the boundary curves and surfaces defining the original geometry. Without this step the adaptation process will be limited in its potential use. It will still be able to capture flow features such as shocks, and adapt to geometries with little or no gradient (sec. 2.6.2), but will need a 'final' mesh resolution where geometric features are most important. This last point clearly defeats an important aspect of mesh adaptation, as it requires prior knowledge of the importance of such features, and of what mesh size should be considered as 'final'. Ideally the best solution would be to integrate the Computer Aided Design (CAD) system used to generate the original geometry, which is unfeasible since this may change from one project to the next. This can be overcome by the introduction of a library capable of handling geometric properties given by a CAD description.

3.1 CAD integration

The insertion of a library capable of making use of the geometric data from a commercial CAD package is essential for the accuracy of calculations. However, this requires particular care and attention from the initial design stages. It will be seen later in the description of the methodology for implementation, how every step from the conceptual design down to the mesh generation is important for the success of the adaptive projection procedure.

In order to be as general as possible in the capability of gathering the data from CAD/CAM packages, it is important to have a library that is able to make full use of CAD definitions. Hence the use of B-splines and NURBS (Non-Uniform Rational B-splines) must be an integrating part of such a library.

The possibility of linking directly with a commercial design software has been discarded from the initial stages of the research as it would create a dependency to it. This would be disadvantageous both financially and in case the distribution would cease. It may also be more complex to implement than one would think, as source codes are generally unavailable, which implies an external call to carry out the point projection.

On the other hand, the development of such a library is not a trivial task. It was therefore chosen to address the problem with the use of a library of C functions to perform the necessary operations on NURBS geometries. This consists in the SINTEF Spline Library (SISL), developed at the scandinavian research group SINTEF, and available under the GNU General Public License. It is particularly suitable since it allows also the handling of interaction between implicit geometric representations such as planes, tori, etc., and NURBS, which is very popular in modern CAD/CAM systems. A general view of where the geometric toolkit interacts with the adaptation module is shown in figure 3.1.

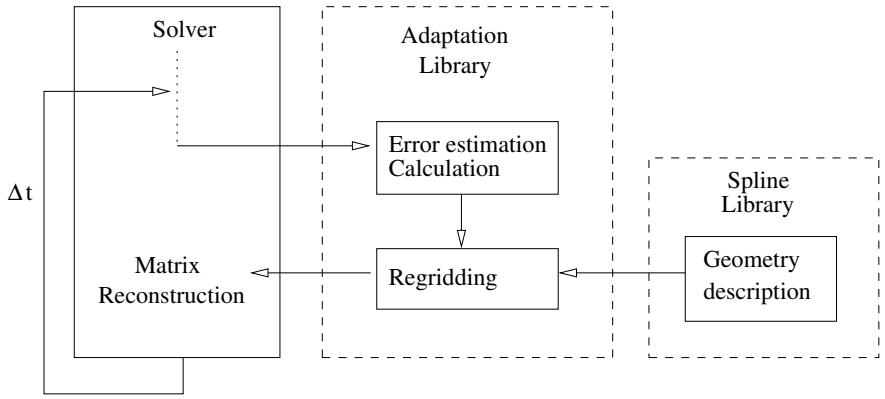


Figure 3.1: Brief schematic of the geometric library placement.

Although NURBS have increased precision, they also have disadvantages when compared to B-splines, it is therefore desirable to use the latter when the option is available. This will be clearer in the following paragraphs, where a brief description of the structures of the curves and surfaces is given.

Due to the vast of the subject, for an in-depth explanation and background of curves and surfaces in CAGD we refer the reader to [116, 117, 118]. However, in order to understand the problem at hand and how it is resolved, we must first illustrate the basics of the tools used herein.

3.2 B-splines

In order to describe NURBS we must first illustrate B-splines, since they are an extension of this form of splines, where the rational part comes in when projecting into affine space. First let us define the *knot vector* $\boldsymbol{t} = \{t_1, t_2, \ldots, t_{n+k}\}$, which is a one dimensional set of coordinates in the parametric space, where $t_i \in \mathbb{R}$ is the i^{th} knot and k is the order of the B-spline. The order is equivalent to the polynomial degree $p + 1$, and n is the number of basis functions that make up the B-spline, as well as the number of vertices of the curve. Knots play a fundamental part in the definition of B-splines. These can be *uniform* if evenly distributed in the parametric space, or *non-uniform* if unequally spaced [119]. Furthermore knots may be repeated, i.e. placed at the same location in the parametric space, and if a knot is repeated m times it is said to have *multiplicity* m [117].

3.2.1 B-splines Curves

The mathematical definition of a B-spline curve is given by

$$\boldsymbol{c}(t) = \sum_{i=1}^{n} \boldsymbol{g}_i B_{i,p}(t) \tag{3.1}$$

where $\boldsymbol{g}_i$ are the *control points* and $B_{i,k,t}$ a sequence of B-splines. This sequence is called a B-basis, or set of B-spline basis functions, which linearly combined in $\mathbb{R}^d$, gives the curve $\boldsymbol{c}$. The dimension of curve $\boldsymbol{c}$ will depend on the dimension of its control points, therefore in $\mathbb{R}^2$ it will be planar and in $\mathbb{R}^3$ spatial. The type of curve generated may be determined by the order k. This will be constant, liner, quadratic, cubic, etc., with $p = 0, 1, 2, 3, ..$ respectively. Hence if the order is four, the degree will be three, and the result will be a cubic B-spline [66]. The representation of a B-spline curve $\boldsymbol{c}$ with parametric range in the interval $[t_k, t_{n+1}]$, can be summarised as follows, with conditions:

- d : the dimension of the Euclidean space in which the control points lie.
- k : the order of the B-spline curve $(p + 1)$.
- p : the degree of the B-basis constructing the B-spline.
- n : the number of basis functions (also the number of vertices); which should be greater or equal to the order of the curve $n \geq k$.
- t : the knot vector of the B-splines; which must be non-decreasing $t_i \leq t_{i+1}$, and must have k multiplicity at the beginning and at the end of the knot vector.
- g : the control points of the B-spline curve in the Euclidean space.

The multiplicity condition at the ends of the knot vector makes this *open*, which is standard in CAD literature [119].

3.2.2 B-basis

We mentioned that B-spline curves are constructed from the linear combination of B-basis or basis functions. These are defined recursively, starting from piecewise constants of degree $p = 0$ for B-splines of order $k = 1$, as:

$$B_{i,p}(t) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1}, \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

and for higher orders

$$B_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} B_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(t) \ . \tag{3.3}$$

Therefore a B-spline of order k will be the result of the sum of the products, of two B-splines of order $k - 1$, with weights in the interval $[0, 1]$. In figure 3.2 the application of (3.2) and (3.3) to a uniform knot vector is shown for orders 1 and 2. In figure 3.3 we can see how a quadratic B-spline is constructed by the sum of the product between linear B-spline 1 and line a, and linear B-spline 2 and line b.
Some important properties satisfied by B-spline basis functions are:

- Each B-spline is non-negative $B_{i,p}(t) \geq 0 \forall t$ and,

$$\sum_{i=1}^{n} B_{i,p}(t) = 1 \ . \tag{3.4}$$

- Support of each $B_{i,p}$ is compact and contained in the interval $[t_1, t_{n+k}]$, which infers *local control*, i.e. moving one control point only alters the curve locally.

- *Endpoint property* which is due to the fact that the knot vector is open, and translates into $\boldsymbol{c}(t_k) = \boldsymbol{g}_1$ and $\boldsymbol{c}(t_{n+1}) = \boldsymbol{g}_n$, i.e. the first and last control points are also the beginning and end of the curve.

- *Convex hull property*, which means the curve lies in the convex hull of its control points.

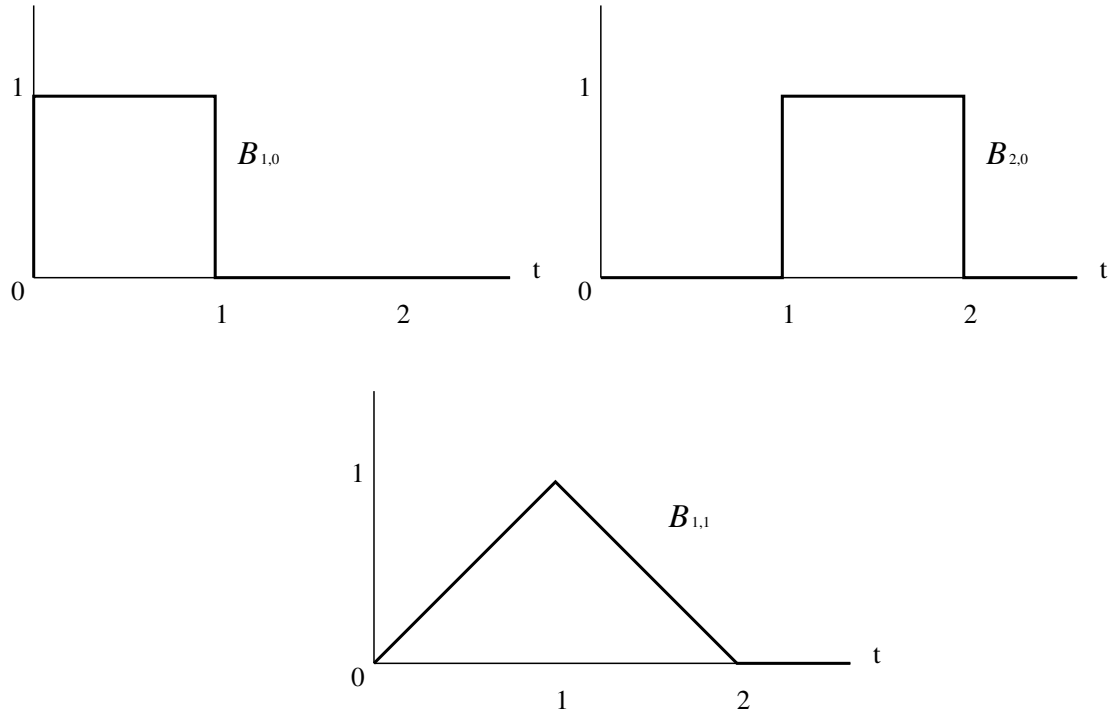


Figure 3.2: Basis functions of order 1 and 2 for uniform knot vector $\boldsymbol{t} = \{0, 1, 2, 3, 4, \ldots\}$.

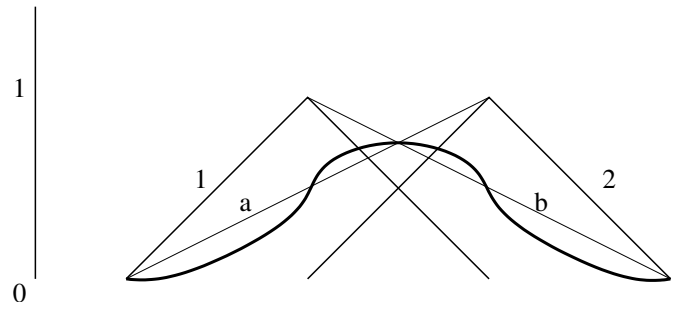


Figure 3.3: Quadratic B-spline constructed from two linear B-splines 1 and 2, weighted by lines a and b.

Control Polygon

We have seen the control points that define the vertices, we now look at the *control polygon*. This is the piecewise linear interpolation of the control points. In fact, the control polygon can be considered as a piecewise linear B-spline, whereby increasing the order increases the distance between the higher order B-spline and the control polygon, i.e. it smoothes the control polygon. An example of a control polygon is given in figure 3.4, where it can be clearly seen how the curve remains in the convex parts of the control polygon.

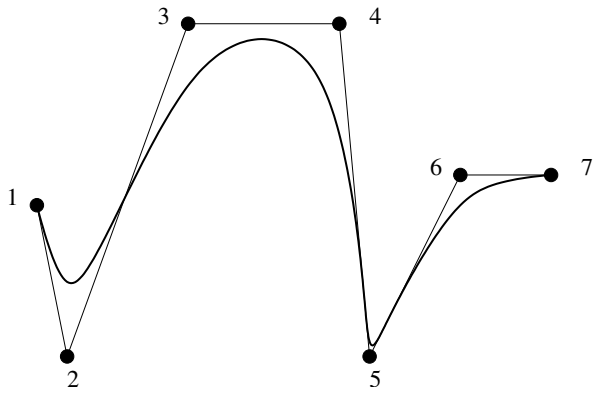


Figure 3.4: A quadratic B-spline (order 3) and it's control polygon.

Knot vector

As previously mentioned, knots play a major role in the characteristic features of B-splines. In particular, as well as describing the parametrisation of the curve, they define the continuity of the curve at the level of the control polygon. This particular feature is determined by the number of equal knots. In fact the curve will be:

- Interpolatory and discontinuous where k equal consecutive internal knots exist.
- Continuous but not in general differentiable where $k-1$ consecutive internal knots are equal.
- Continuously differentiable, with discontinuity in the second derivative where $k-2$ internal equal knots occur.

An example of this can be seen in figure 3.4, where two knots have been placed at the location of point 5. If we take the non-uniform open knot vector of a quadratic spline $\boldsymbol{t} = \{0, 0, 0, 1, 2, 3, 3, 4, 5, 5, 5\}$, we can notice how the multiplicity of knot $t = 3$ is equal to

the polynomial order. Hence it will be $C^{p-2} = C^0$ continuous at $t = 3$ and $C^{p-1} = C^1$ continuous elsewhere. More in general, for B-basis of order p, continuous derivatives of order $p - 1$ exist where no repeated knots or control points occur.

3.2.3 B-splines Surfaces

A B-spline surface is a tensor-product surface. This is due to the fact that the B-basis of a B-spline surface is the product of two basis functions of B-spline curves. If we consider knot vectors $\boldsymbol{u} = \{u_1, u_2, \ldots, u_{n+p+1}\}$ and $\boldsymbol{v} = \{v_1, v_2, \ldots, v_{l+q+1}\}$, with p and q the polynomial degrees of the B-splines in the first and second parameters respectively, and control points $\boldsymbol{g}_{i,j}$ forming the *control net*, the surface can be described as:

$$\boldsymbol{s}(u, v) = \sum_{i=1}^{n} \sum_{j=1}^{l} \boldsymbol{g}_{i,j} B_{i,p}(u) H_{j,q}(v) \ , \tag{3.5}$$

where B and H are the basis functions of B-spline curves, n and l the number of vertices with respect to the first and second parameter, $k_1 = p + 1$ and $k_2 = q + 1$ the order of the B-splines in the first and second parameter. As for B-spline curves the knot vectors must be non decreasing, and the number of vertices in both parameters be $n \geq k_1$ and $l \geq k_2$. Also, k_1 equal knots should be present at the beginning and end of knot vector $\boldsymbol{u}$, and k_2 equal knots at the beginning and end of knot vector $\boldsymbol{v}$. The properties of B-spline surfaces are similar to those of B-spline curves. As for the B-spline curve basis, the support for the B-spline surface basis function is compact and contained in an interval. This is the rectangle $[u_i, u_{i+k_1}] \times [v_j, v_{j+k_2}]$. The shape of the function will be pyramidal for degree one in both directions, bell shaped for higher degrees.

3.3 NURBS (Rational B-spline)

Rational B-splines are a generalisation of a B-spline curve, as the name suggests, with the addition of a weight vector $\boldsymbol{w} = \{w_1, w_2, \ldots, w_n\}$. One of the advantages of NURBS is the capability of representing conic sections exactly. This is obtained from the projective transformation of B-spline curves which lie in $\mathbb{R}^{d+1}$, in $\mathbb{R}^d$. For example a circle in a plane could be described from a piecewise quadratic curve in space. If we consider a set of control points $\boldsymbol{g}_i^w$ of a B-spline curve in $\mathbb{R}^{d+1}$, with knot vector $\boldsymbol{t}$, these are the projective control points of the rational B-spline curve in $\mathbb{R}^d$. The rational basis functions are described by:

$$R_i^p(t) = \frac{B_{i,p}(t) w_i}{\sum_{j=1}^{n} B_{j,p}(t) w_j} \ , \tag{3.6}$$

and the rational B-spline as:

$$\boldsymbol{c}(t) = \sum_{i=1}^{n} R_i^p(t)\boldsymbol{g}_i \ . \tag{3.7}$$

The trade-off in increased precision however, comes in the form of complexity and loss of efficiency when evaluating properties such as derivatives. This is due to the curve depending nonlinearly on the weights. Weights are strictly positive $w_i > 0$ for the convex hull property to hold.

3.3.1 NURBS Surfaces

Just like the NURBS curves, the surfaces are a generalisation of B-Spline surfaces, whereby:

$$R_{i,j}^{p,q}(u,v) = \frac{B_{i,p}(u)H_{j,q}(v)w_{i,j}}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{l} B_{\hat{i},p}(u)H_{\hat{j},q}(v)w_{\hat{i},\hat{j}}} \ , \tag{3.8}$$

and the resulting NURBS surface:

$$\boldsymbol{s}(u,v) = \sum_{i=1}^{n}\sum_{j=1}^{l} R_{i,j}^{p,q}(u,v)\boldsymbol{g}_{i,j} \ . \tag{3.9}$$

with the addition of the weights matrix $\boldsymbol{w}_{i,j}$. Like with NURBS curves, the surfaces can be used to represent exactly surfaces such as spheres, cones, etc., but just like the curves, there is a loss of efficiency when evaluating them. Also, just like for the curves $w_{i,j} > 0$.

3.4 Implementation

The addition of such a library is not a simple task, not only in terms of programming and data structures, but primarily in terms of different cases that can and will occur, such as the intersection between surfaces. A more detailed schematic view of where the projection algorithm is placed the source code is shown in figure 3.5. Here, we can see how and when the projection algorithm comes into play, and also some of the data it needs in order to work. In the next paragraphs we shall describe in more detail the algorithm developed for the projection decision-making, and all the necessary conditions for this to work.

First of all we consider the patch structure that is inherent in the adaptation module. The boundary description is defined at the lowest level with the help of faces (or segments in 2D), which are grouped in patches. Therefore each patch contains a specified number of faces, and corresponds in fact to a surface in the CAD definition of the geometry. This is a fundamental aspect of the projection algorithm, since if more than a surface belonged to a patch, it would be much more complex to tell which surface to project the new node

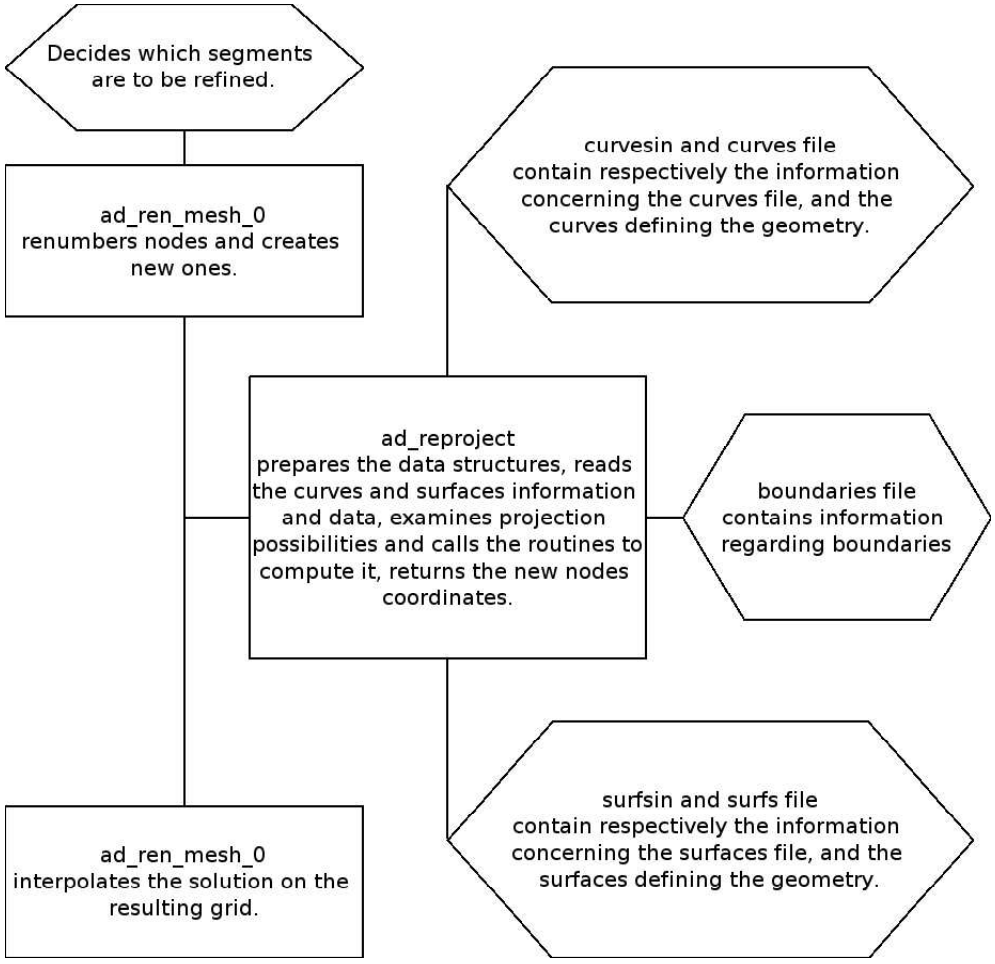


Figure 3.5: Brief schematic of the reprojection algorithm placement.

onto. An algorithm to evaluate the projection to each surface belonging to the patch, and verifying the closest surface to the point could resolve this issue, but with much higher costs in computational time. Patches work in both three dimensions and two dimensions. In the former they describe the finite discretisation of a surface, in the latter segments describe the boundary.

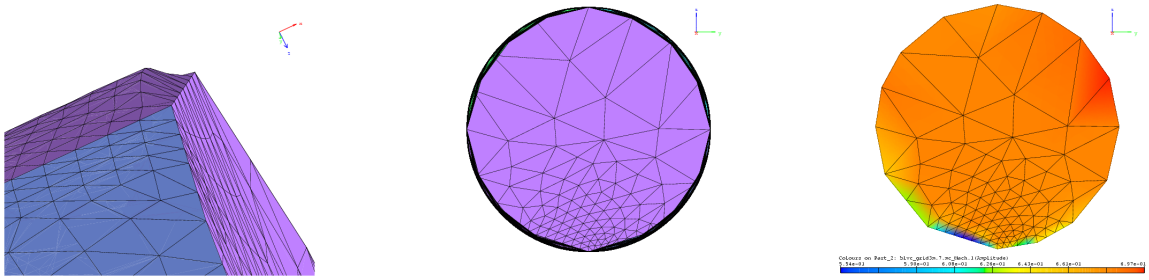
The CAD geometry descriptions are divided in a curves and a surfaces data files, which are linked to two other files with the number of entities in each. These last two files also contain the individual names of each geometric entity. The surface file contains also information regarding the curves that define each surface, in particular the number of curves and the identifying number of each. It should be noted that the sizes of these data files are extremely compact and have no significant impact in disk usage, unlike grid files.

At this point curves, in 2D, and surfaces, in 3D, are matched to patches with the use of boundaries file information. Segments that belong to the surface and that are marked for refinement are then identified. Note that this step is straightforward in 2D, where a segment may only belong to one boundary and is defined as a face itself. Hence it will be associated to a distinct single curve.

In 3D things become a little more complex since a segment is the interface of two surface faces. This limitation is assumed to be always true, since only a structure of zero thickness, at the point of intersection with other two surfaces, may produce an ambiguous segment belonging to more than two faces. Moreover it would be undesirable to model an unrealistic feature of this type. The complexity resides in the parenting of the segment, since the faces it belongs to could lie on two different surfaces, in which case a decision must be made onto which surface the new node should be projected.

The problem can be clearly seen in figure 3.6. Here the new nodes have been projected to the surface defining the exit, which is flat, hence they lie in the plane at the exact midpoint of the segment they refined. This would be correct, if it wasn't for the fact that the new node splits a segment that lies on the frontier of two patches, and the parent face to which it should have been referred for projection, should have been the one describing the wall of the channel. In fact, if this had been done to begin with, the problem would go unnoticed.

This is overcome with the information gathered earlier on the curves defining the surfaces. The curve that describes the intersection of the two patches is taken as the geometry definition to project onto. The result of this shown in figure 3.7. It can be noticed how the exit is also less refined. In the previous case the refinement was due to the flow altered by



(a) (b) (c)

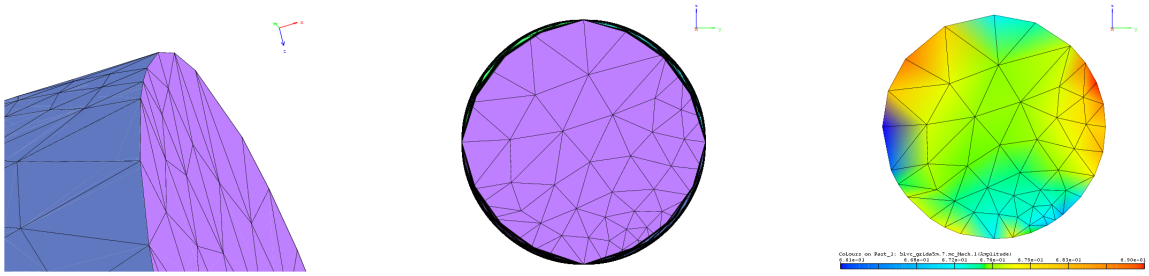
Figure 3.6: Projection to the wrong surface for segments on the intersection between patches. (a) Detail of the channel exit with a false step created. (b) Front view of the channel exit. (c) Relative solution with erroneous results due to wrong placement of new nodes.

the coarse mesh, which created an artificial step. If the refinement nodes had been placed correctly, the result would have been a smoother flow due to the exact geometry representation, which presents no such artificial features created by an incorrect (or too coarse) mesh.

Particular care has to be taken when preparing the curves and surfaces however. This method requires each surface to have only one curve in common with another surface. In this way the frontier curve between patches can be clearly found. If more than one curve was to define the intersection between two surfaces an algorithm should be added similar to the one considered for multiple surfaces describing a patch. The projection would have to be evaluated on all common curves and the closest one would be taken as good.

Let us take as example a blade in a cylindrical channel as shown in figure 3.8. In the case of inviscid flow this would be twice as much the needed geometry, in fact we could cut along the xz plane and add a symmetry plane. It is a good example though, of the type of problems that can be encountered above, and gives a good idea of the type of thought and pre-processing that must go into the design approach.

From a design point of view it would be natural to draw the blade as two surfaces with the leading edge and trailing edge in common. This however would create the condition described above, and hence is not viable. Therefore the blade is split in four surfaces as



(a) (b) (c)

Figure 3.7: Projection to the defining curve of the patches frontier. (a) Detail of the channel exit. (b) Front view of the channel exit. (c) Relative solution with correct results.

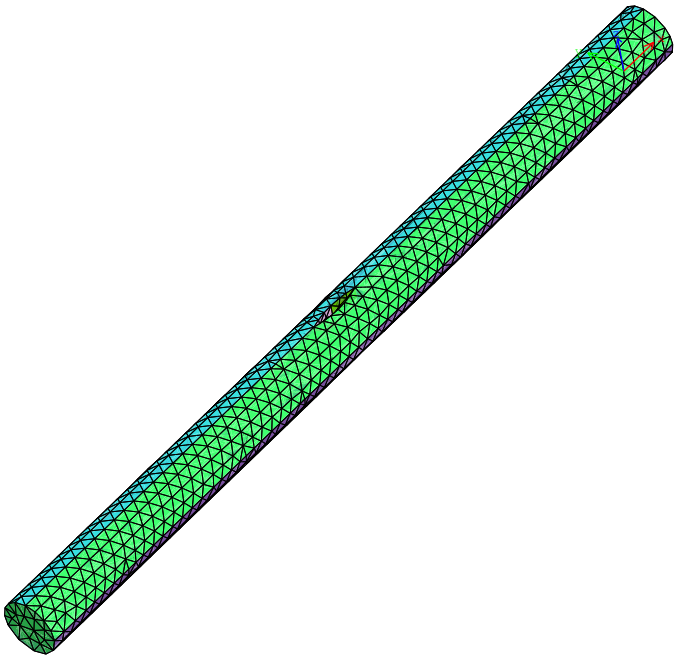


Figure 3.8: Blade in cylindrical channel.

shown in figure 3.9. In this way the problem is solved. There remains a problem with the cylinder walls. If we start from the idea that the walls are defined as two surfaces cut by the xz plane, then each surface would intersect twice with the two vertical blade surfaces on their side of the plane. A solution could be that of splitting again the blade in the xy plane. This would not be the good solution though, as the two wall surfaces would still touch at the top and at the bottom in two distinct places with two intersection curves that have no continuous solution as they are disconnected and parallel.

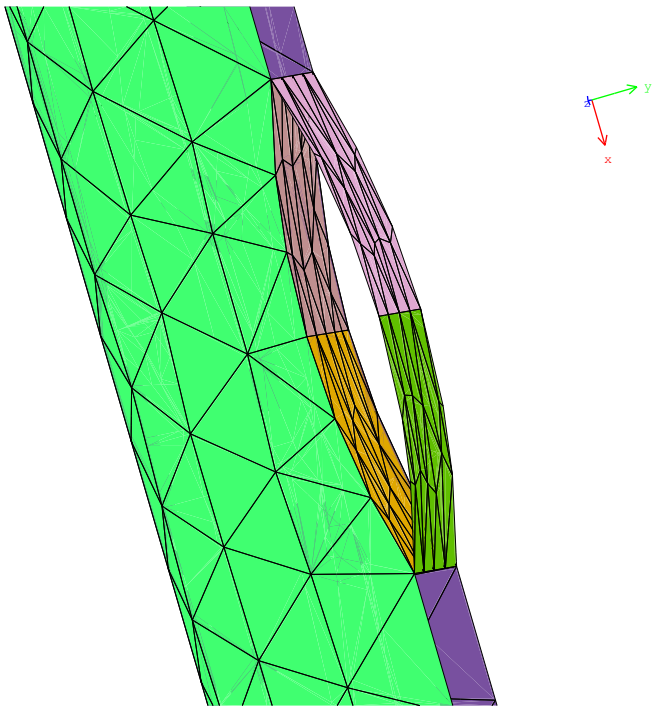


Figure 3.9: Detail of the blade from above with half wall channel blanked.

Hence the walls are split in four surfaces, at the xz and xy planes through the centre the circle defining the cylinder. An attentive eye will immediately notice that the two top wall surfaces will still have two intersections in common which are separated by the blade. This is a different case from the previous one. Here a single curve can be used to define the two intersections. If we examine figure 3.10, we can see that the curve that describes the two intersections is continuous. In this particular case it is allowed since the points are projected onto the same curve, and since the point to project is at the midpoint of the segment there is no risk of projecting it in onto the wrong intersection.

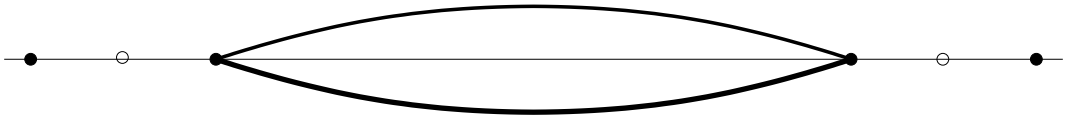


Figure 3.10: Curve connecting the top wall surfaces divided by the blade. The black dots are existing segment nodes, and the empty dots are the new nodes to be placed.

A further feature that is an important part of the algorithm, is that of being able to specify the surfaces to project onto. This has been implemented only for 3D applications since it has less (if any) appeal in two dimensions. The choice is made through the surfaces information file, where only the selected surfaces and their defining curves are taken into consideration. In the latter stages only on these entities the projection is carried out. It can be of particular interest in cases such as that shown in figure 3.11. Here it is obvious that the only surface that will need point projection, to insure the geometry is followed, is the bump, since all other surfaces are planar. The defining curves of the surface will also be needed, hence the information of all other surfaces must be retained, but not the surfaces definitions or those of curves that are not frontiers of the selected surfaces.

3.4.1 Re-projection

Finally, the projection algorithm is generalised for all elements that lie on the boundary. This is used in the case where the smoothing routine described earlier (sec. 2.5.2) is selected. In fact, the smoothing may still cause some problems when used, particularly when the starting mesh is very coarse and a few adaptation steps are necessary. In these cases the multiple iterations of the smoothing procedure may cause the boundary nodes to slowly drift away from the geometrical definition, despite all the checks in place to avoid this.

Hence the generalised projection routine is called at the end of each smoothing iteration to check that all nodes are correctly placed on the geometry, and reposition them when this is not the case. The effect of this can be clearly seen from figure 3.12, where the smoothing iterations pushed the nodes away from the surface *(a)*, and how the integration of the projection places them back onto the surface after each iteration *(b)*.

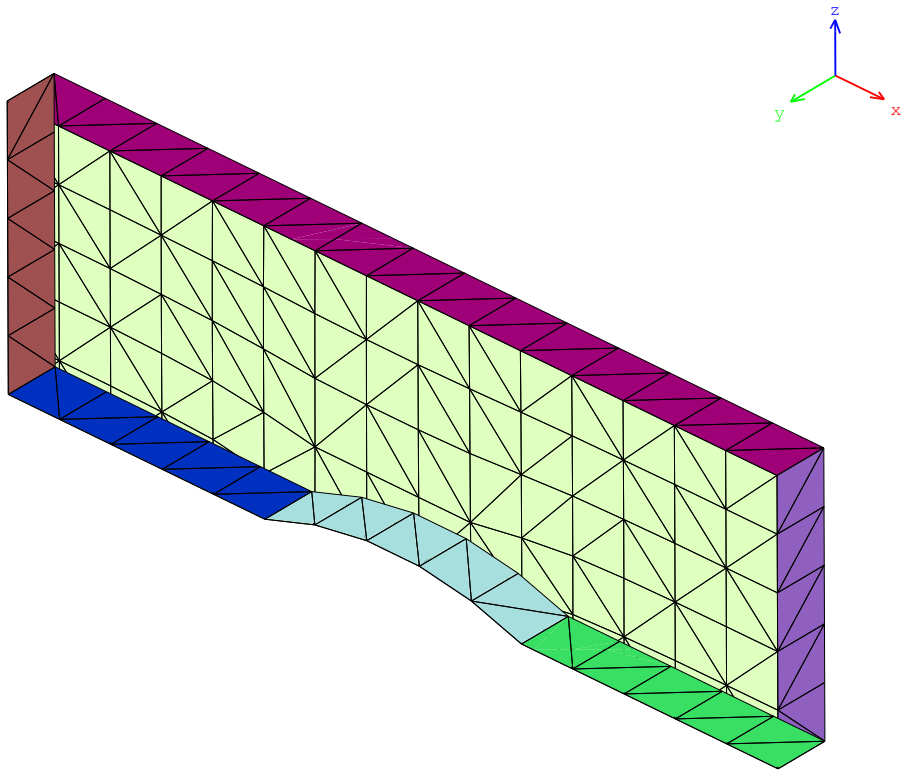
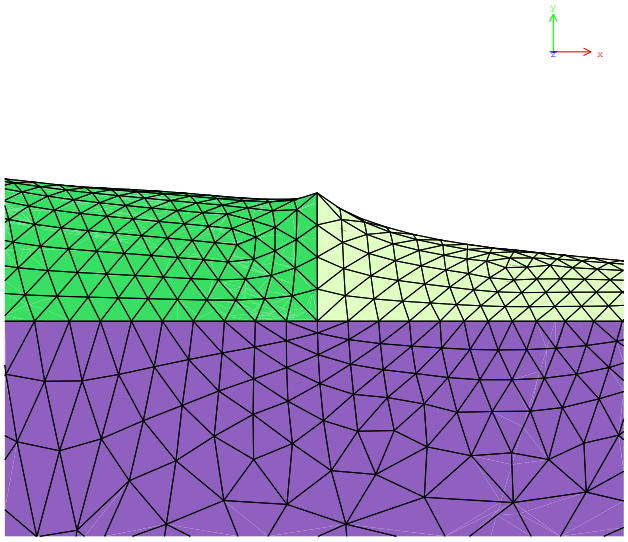
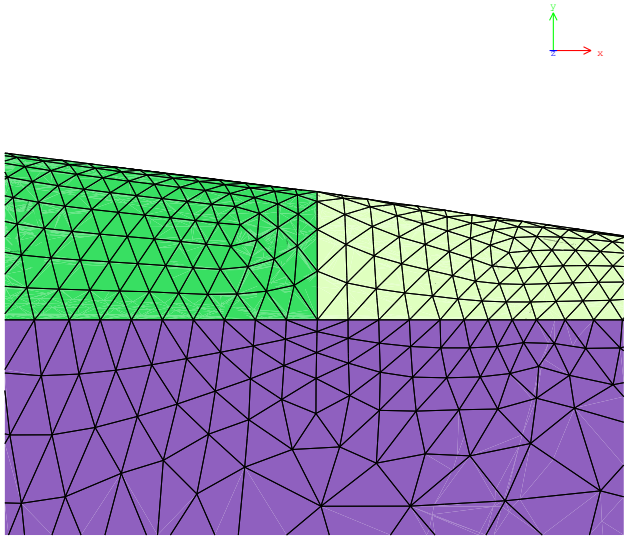


Figure 3.11: Bump in a channel: surface initial mesh and patches shown in different colours with one side blanked, viewed from the down side.



(a)



(b)

Figure 3.12: Projection curves and surfaces during the smoothing iterations: (a) without projection of all nodes, (b) with projection after each smoothing iteration.

3.4.2 Point projection

This can be summarised as the search of the minimal distance from the point $\boldsymbol{p}$ in space to the geometric entity onto which it should lie. The possibility of the point being at the same minimal distance between more than one entity is cancelled. This is done by identifying the node as belonging to a specific geometric entity, with the use of its parent nodes, as explained earlier. There remains therefore only the problem of finding the point on the geometric entity for which the distance to the point in space is minimised. This problem can be approached in various ways and is most commonly referred to in literature as the foot point problem [120]. A detailed description of a few commonly used methods, including the ones implemented in the NURBS library, can be found in [121, 122].

Three possible choices for carrying out such a task are given in the library. The first consists in a Newton iteration on the distance function between the point and the curve/surface, though if a bad initial guess is made then the iteration may give a local rather than a global closest point. However, since in our case the point is already close to the surface, the local minimum problem is not a concern. The second option is based on the first since it makes use of the same algorithm, but rather than having a user input initial guess, it takes the closest control point and estimates the corresponding curve parameter value or pair for the surface case. The value or values obtained are then used as the starting point for the iteration. The last option in the library is that of entering the equation of the curve/surface into the equation of a circle/sphere having $\boldsymbol{p}$ as its centre and radius zero. This results in a one-dimensional curve/surface for which the minima is found [67].

3.5 Applications

In order to verify the effectiveness of the procedures introduced in this section, three test cases were considered. The first is the earlier mentioned blade in a cylindrical channel (fig. 3.8). A second test case is represented by a 10% bump in a channel as shown in figure 3.11. In both these cases the geometry is created from scratch. The third and final configuration examined is that of the ONERA M6 wing, where the geometry is reconstructed from an available mesh. The details of each calculation are presented hereafter, together with the solutions which were obtained using THOR. The geometries are described with non-dimensional units in all cases.

3.5.1 Pipe blade

This geometry has been taken as example for describing the design approach, and proves a good test case for it's simplicity in the design process and yet complexity in the geometry

conservation during the adaptation process. The initial flow conditions are set to Mach number 2.0 at zero angle of attack in order to obtain some interesting flow features. Here we look principally at the adaptation projection process rather than the solution, hence only a first order scheme is used.

The blade is described by two arcs of 4 units chord length and has a maximum width of 0.6, whilst the distance from the tips of the blade to the ends of the pipe is 28 units. Finally the diameter of the pipe is of 4 units. The initial coarse grid is uniform and contains 2 486 nodes, 10 683 tetrahedra and 2 384 triangular surface elements (fig. 3.8).

Four distinct adaptation steps were then performed in series, each restarting with the previous steps adapted mesh. The adaptation is performed with respect to the difference in density on the segment. An identical procedure was also done without the projection algorithm, and used as comparison. This last computation serves also as illustration of initial mesh dependency when no projection is applied.

Adaptation step	With Projection			Without Projection		
	N_nodes	*N_elements*	*N_surf_el*	*N_nodes*	*N_elements*	*N_surf_el*
first	6 024	29 110	4 458	6 011	29 077	4 440
second	20 755	110 287	9 790	21 711	115 361	10 326
third	86 876	493 771	22 264	95 565	538 420	27 318
fourth	314 652	1 854 594	42 718	381 353	2 219 641	69 152

Table 3.1: Grid sizes for adapted pipe blade calculations: with and without projection algorithm.

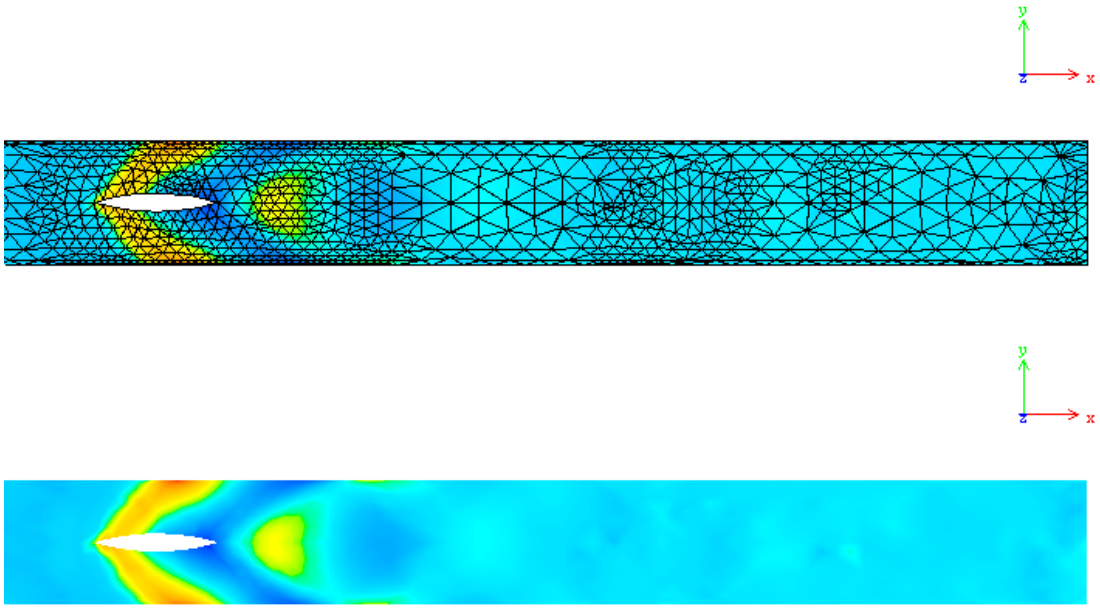
Considering the steps of the adaptation cycles reported in table 3.1, the surface grids show remarkable differences. As expected, the meshes obtained from the algorithm that makes use of the geometry definition are much smoother. In particular, grid induced flow features that may come from an excessively coarse initial mesh, are lost after one or two steps. Whereas in the case of no projection these features are further refined to no avail. Details of this are shown in figures 3.13 and 3.14. The effect is clearly shown with the smoother solution, and only some further refinement, in figures 3.13(a) and 3.14(a) where the projection algorithm is selected. In figures 3.13(b) and 3.14(b) it is noticeable how, despite the refinement, the original grid is still driving the solution, and because of this it is refined also in those areas where an artificial step is created by the mesh coarseness. This effect explains the values in table 3.1, which become increasingly higher for the

case without projection respect to that projecting, as more refinement steps are carried out.

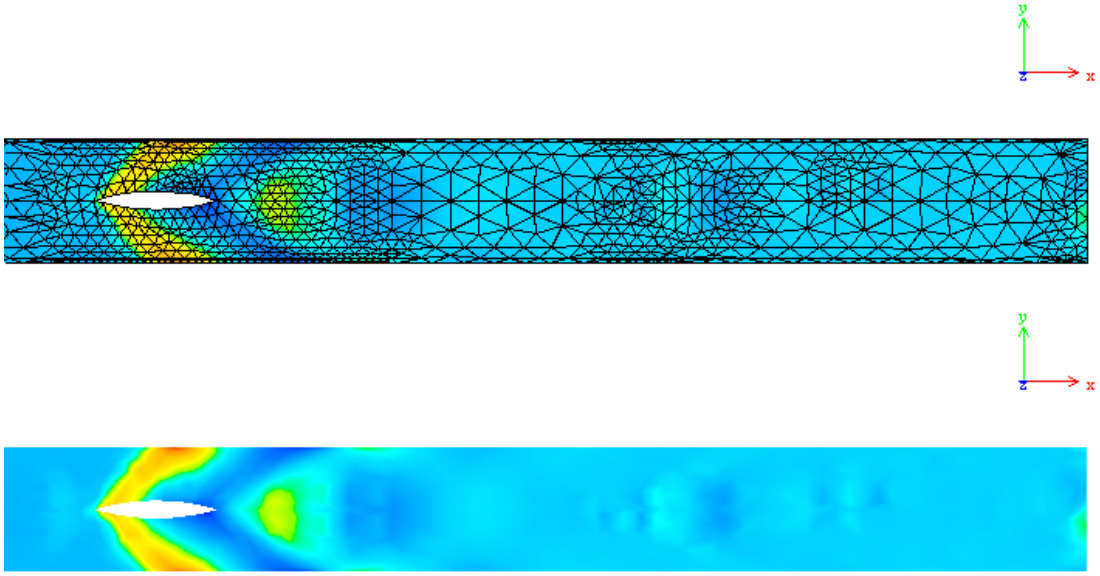
The percentage change was taken as $\%change = (N\_wp - N\_p)/N\_wp$, where wp and p stand for without and with projection respectively, and N is the number of the characteristic size of the mesh being compared i.e. nodes, elements, and surface elements. This is expressed graphically in figure 3.15, where it is clear to see the effect of the projection algorithm, particularly on the number of surface elements, which is where the procedure has the greatest influence.

Furthermore, by overlapping the original grid to the adapted grid after only two steps, there is a remarkable difference between the two. This is clearly shown in figure 3.16, with only two halves of one blade wall selected. From the internal side viewpoint of the blade only the initial grid is visible, with a few segments from the adapted grid, and vice versa. The reason for the visibility of some segments is due to the fact that some of the original segments may lie on the exact geometry. A detail of the gap between original and adapted grid at the top, is also shown in figure 3.16 (c).

Although the soultion itself might not be of particular interest, it is important to notice how and if this is improved by the projection process in order to verify it's effectiveness. A close-up of the zone at the location of the blade in the pipe is shown in figures 3.17 and 3.19, with a further detail of the area behind the blade in figure 3.18. In particular from figure 3.18 it is clear to see the projection on the channel wall and its effects. The solution examined is the fourth and final step of the case considered here, and shows the variation in density of the fluid flow. As expected, a much smoother and precise solution is obtained with the projection algorithm, confirming the method to effectively ameliorate the result.

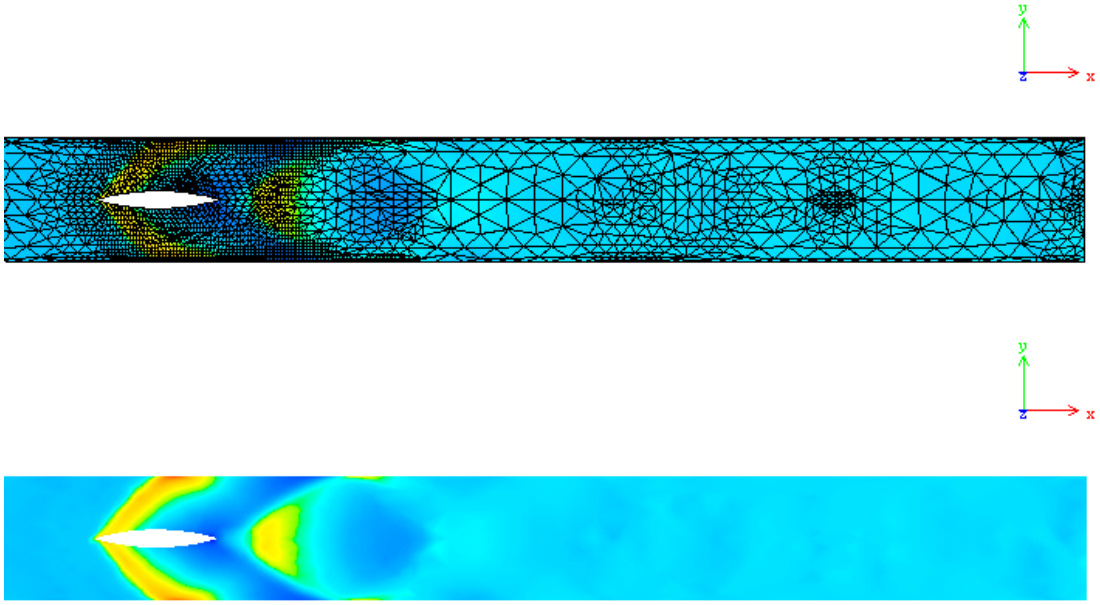


(a)

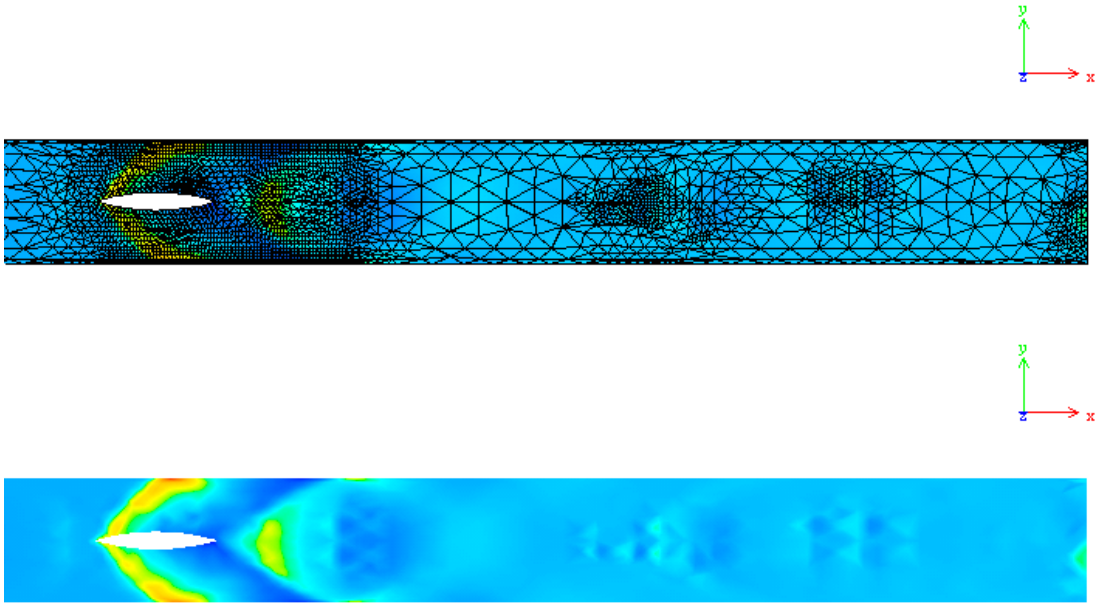


(b)

Figure 3.13: Grids and density solution downstream of the blade: (a) and (b), first step with and without projection respectively.



(a)



(b)

Figure 3.14: Grids and density solution downstream of the blade: (a) and (b), second step with and without projection respectively.

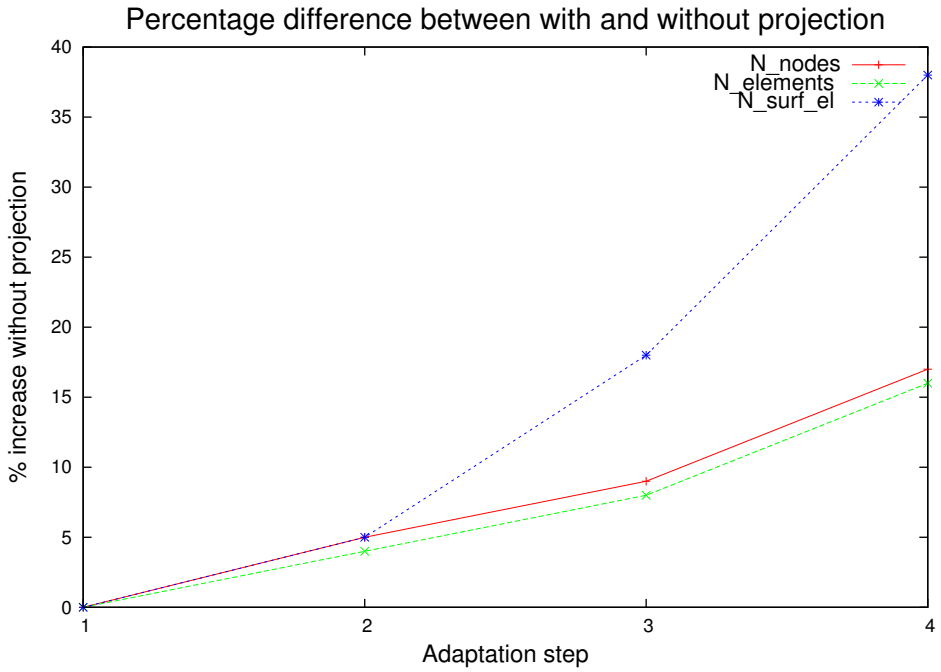
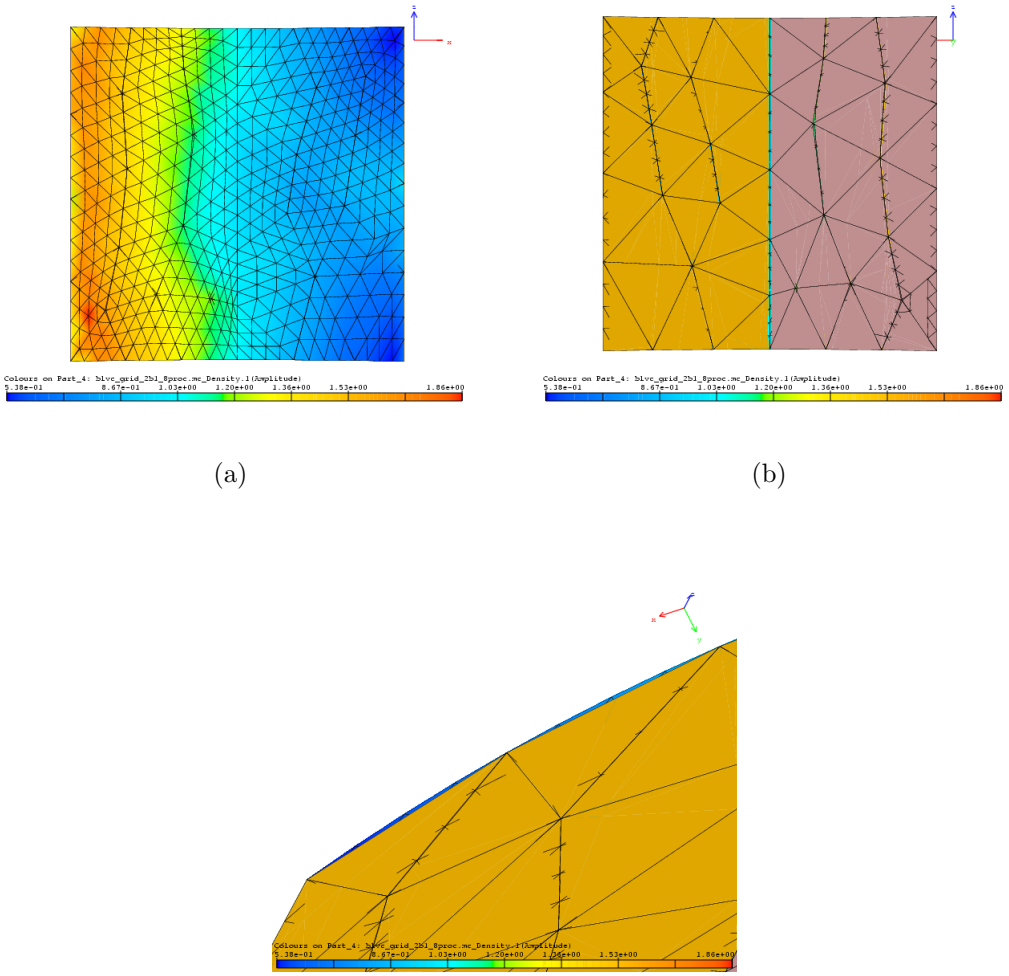


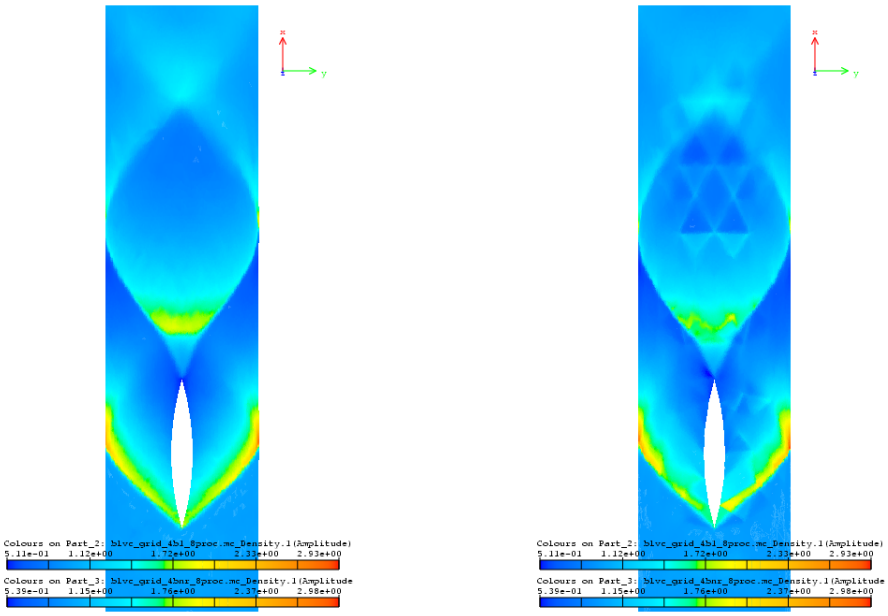
Figure 3.15: Percentage difference between mesh characteristic sizes of adaptations with and without projection.



(a) (b)

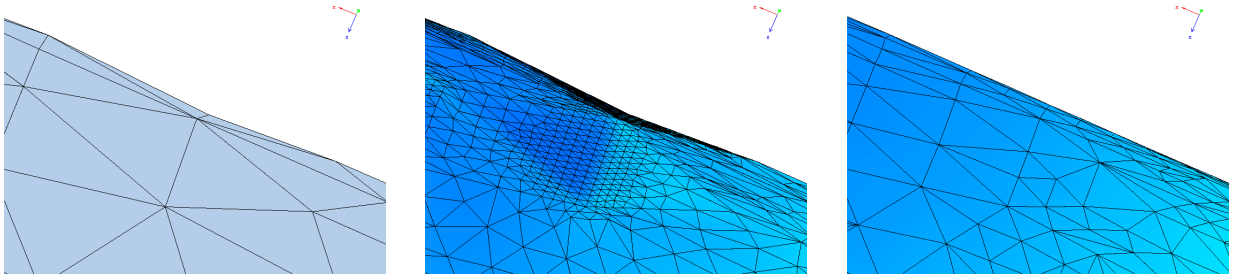
(c)

Figure 3.16: Detail of the difference between adapted projected mesh and the original grid of one side of the blade, view from:(a) outside, (b) inside and (c) top with gap detail.



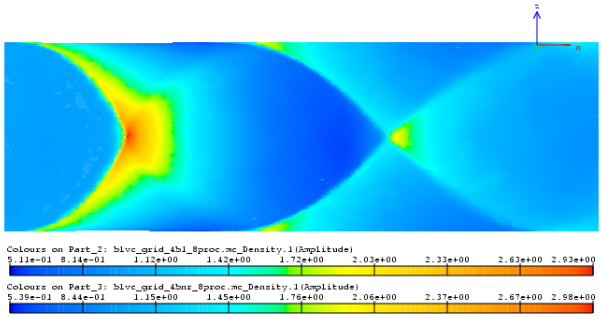
(a) (b)

Figure 3.17: Detail of the solution in proximity of the blade and downstream, viewed from underneath: on the left with projection, on the right without.

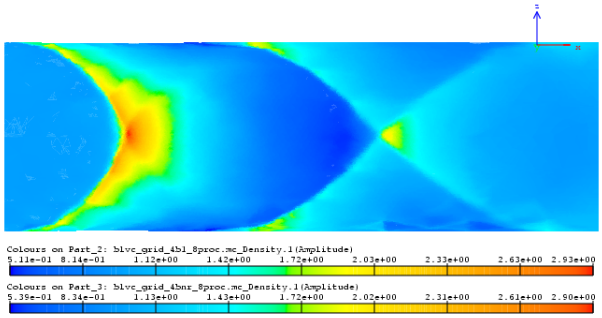


(a) (b) (c)

Figure 3.18: Detail of the area behind the blade of fig. 3.17, showing initial grid (a) dependency with artificial step created (b), and no dependency with projection (c).



(a)



(b)

Figure 3.19: Detail of the solution in proximity of the blade and downstream, viewed from the side: on the top with projection, on the bottom without.

3.5.2 Bump in a channel

This is a classical test case which offers the possibility to test all aspects of the adaptation process including the solution. The initial grid is that shown previously in figure 3.11, and it also allows us to test whether the feature of projecting only onto selected surfaces works correctly.

The geometry consists in a 10% circular bump in a channel, of unit chord length and 0.1 maximum thickness. Height, length and width of the channel are respectively 1, 3 and 0.2 units. The free-stream velocity is set to Mach number 0.675 at zero angle of attack, which causes the flow to be transonic. The initial grid consists of 184 nodes, 426 tetrahedra and 364 triangular surface elements.

Adaptation/ flow - step	**Projection**			**No Projection**		
	N_nodes	*N_elements*	*N_surf_el*	*N_nodes*	*N_elements*	*N_surf_el*
first-flow 1	600	2 058	908	600	2 058	908
second-flow 2	1 731	7 390	1 804	1 717	7 318	1 800
third-flow 3	5 104	25 204	3 506	4 952	24 373	3 430
fourth-flow 5	13 362	71 143	6 388	15 625	84 313	6 716
fifth-flow 6	31 933	179 573	10 236	48 158	273 635	12 830

Table 3.2: Grid sizes for adapted channel bump calculations and relative flow solver step: with and without projection algorithm.

At a first glance it is clear to see the differences with the prior test case from table 3.2. Here the main variance is between the number of nodes and elements, but only slightly and at the last stage of adaptation for the surface elements. This is normal since only a slight part of the surface mesh is projected onto the geometry. Also, unlike the pipe blade where only a first order scheme was used, here the approach was changed.

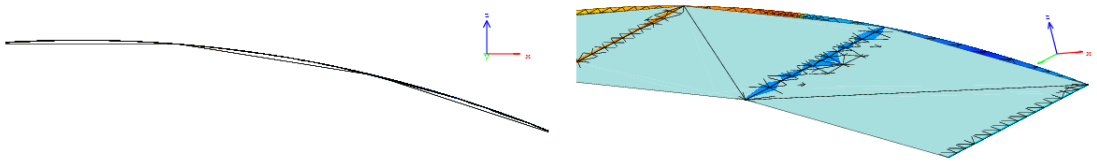
For compressible CFD, when performing a steady state or transient solution computation, the following steps are usually carried out:

1. On the initial grid, perform a first order spatial accurate solution technique to obtain a provisional approximate solution. This solution will have an initial position of shock waves and discontinuities, but will be very dissipative and not capture finer details.

2. This first solution can then be used for a first step of adaptation and/or serve as the initial condition for a further number of iterative cycles using a second order scheme, which will resolve more precisely the solution. At this point the mesh can be further adapted.

This procedure is standard and removes some stiffness from the problem resolution. It is also known as "Blending" first and second order solutions.

Initially a first order scheme is used to advance the solution and adapt the mesh, up until the third adaptation step (adaptation and flow steps 1-3). After this refinement the solution and grid obtained are used to start the second order solution, without adaptation (flow step 4). Once a solution is obtained, it is then used to start the following adaptation steps with the second order solutions (adaptation steps 4,5, and flow steps 5,6). In all steps the adaptation is performed with respect to the difference in density on the segment.



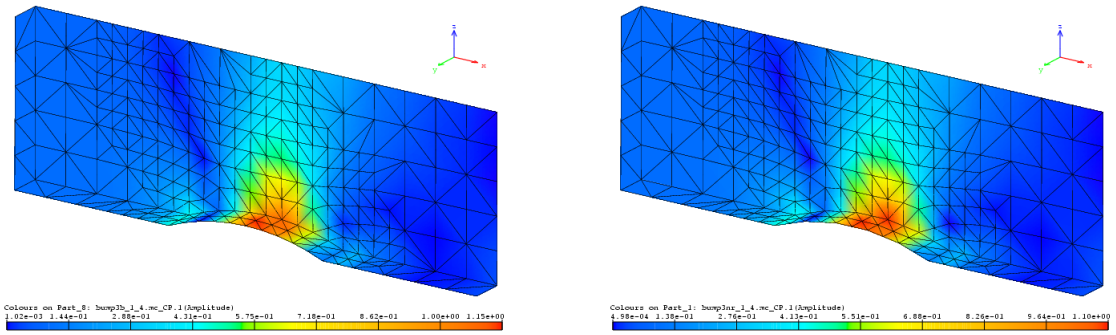
(a) (b)

Figure 3.20: Detail of the differences between original and adapted projected grid for the channel bump: (a) view from the side, (b) tilted view from underneath the bump.

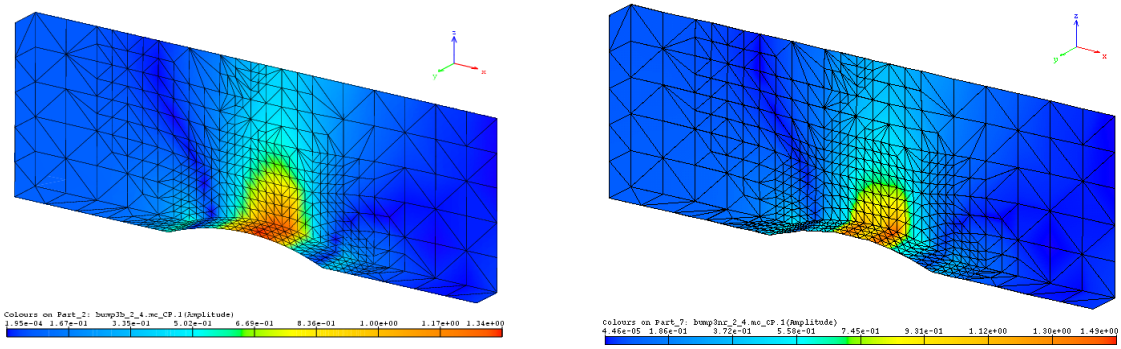
A detail of the difference in the projected part of the grid is shown in figure 3.20. In particular the gap between the original coarse mesh and the adapted projected grid can be clearly seen. As for the previous case, in the proximity of segments of the original grid that lie on the defining geometry, adapted segments are also visible.

The grids and solutions produced from the various steps are shown in figures 3.21-3.22, with solution field for the Mach number and C_p isolines of the final step shown in detail in figure 3.23. Here similarities in the grids can be seen until the second order solution

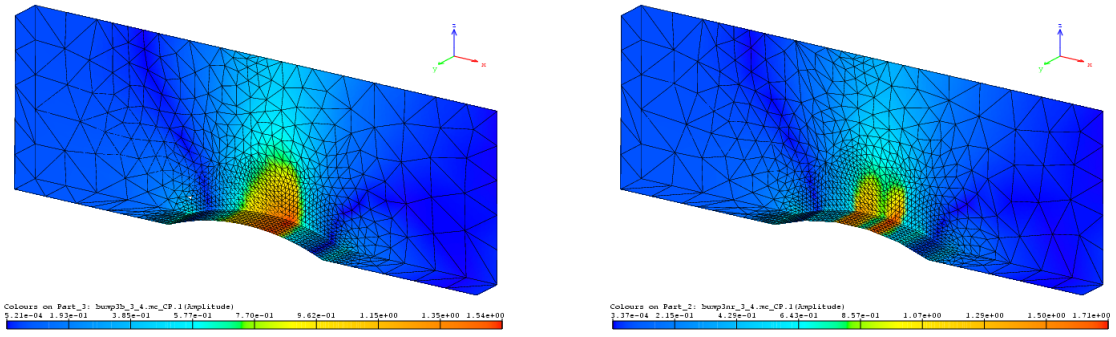
adaptation steps in (fig. 3.22(b)), however the solution shows discrepancies between the adapted projected and that without, already from the second step (fig. 3.21(b)). In particular, since without projection the original grid drives the solution, two regions of shock seem to appear in the second step, of which the first lies at the interface of the original coarse elements, which define a sharp feature rather than a smooth curvature of the bump. Although this is greatly reduced when switching to the second order solution (fig. 3.22(a)), large differences in both the adapted grids and the solution are clear in the final steps of the computation, with a correct shock placement for the one using the projection algorithm, and a corrupted one for the standard midpoint scheme.



(a)

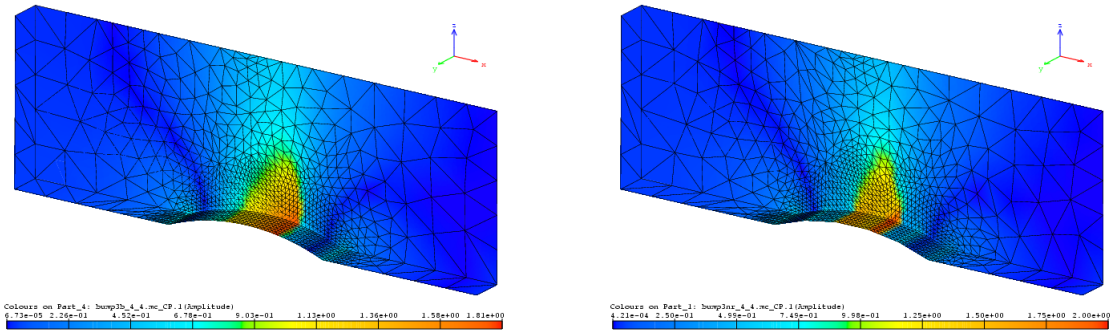


(b)

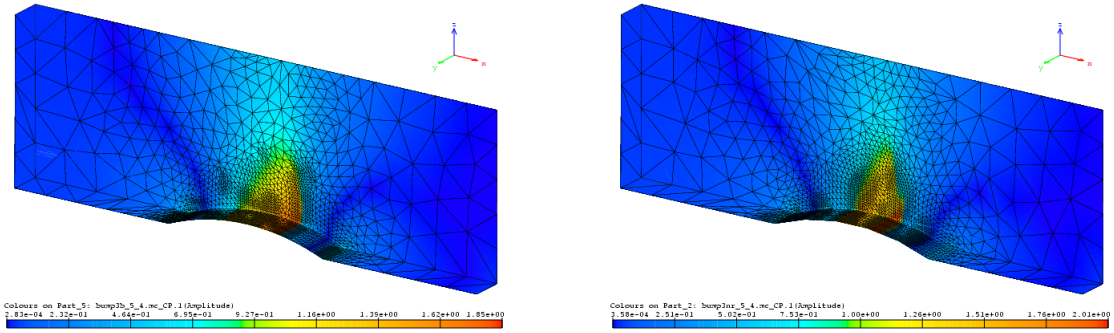


(c)

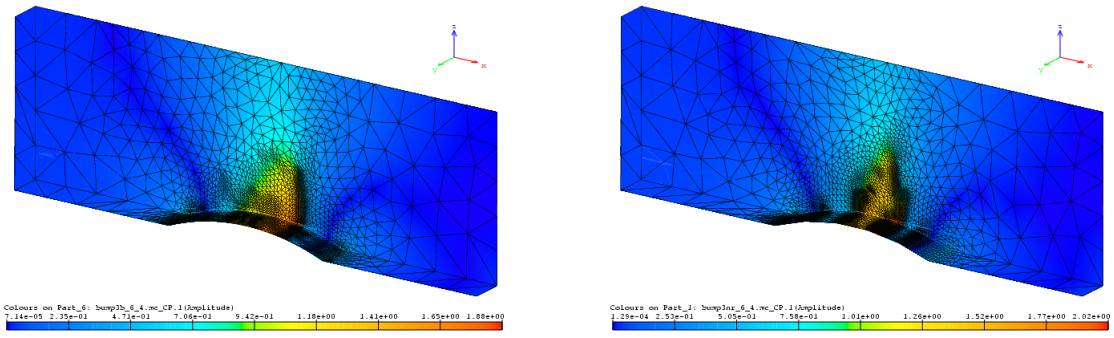
Figure 3.21: Adaptation/flow steps with (left) and without (right) projection: (a) first, (b) second, (c) third. Solution field is C_p.



(a)

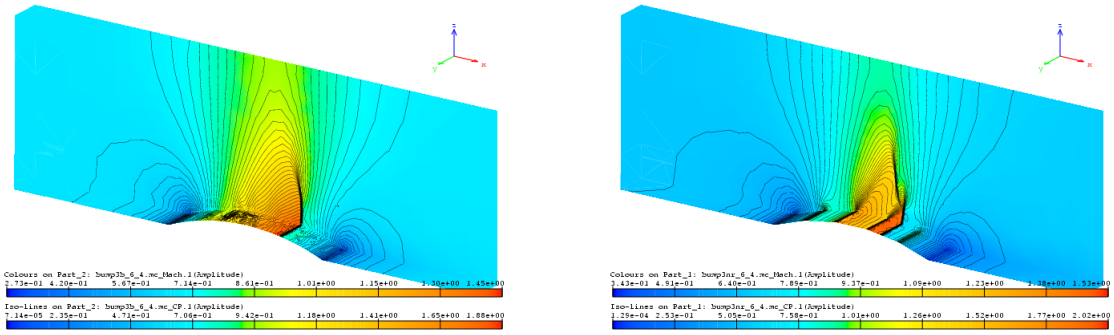


(b)

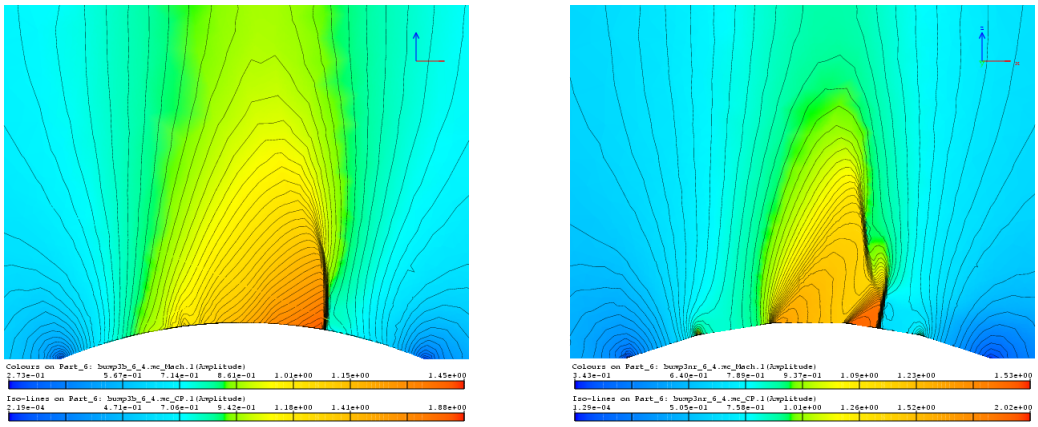


(c)

Figure 3.22: Adaptation/flow steps with (left) and without (right) projection: (a) only flow calculation step 4, (b) adapt fourth - flow 5, (c) adapt fifth - flow 6. Solution field is C_p.



(a)



(b)

Figure 3.23: Final solutions with (left) and without (right) projection. (a) Mach number solution field and C_p isolines and (b) detail from the side.

3.5.3 ONERA M6 wing

Unlike the previous test cases, an exploitable mesh is used to recover a usable set of curves to reconstruct the geometry. Although the geometrical information of the ONERA M6 wing can be easily found online, this test case is also used as basis to check the feasibility of recovering a usable geometry from an available mesh. The mesh used to recover the initial set of curves is shown in figure 3.24, in which the different patches of the mesh are also shown.

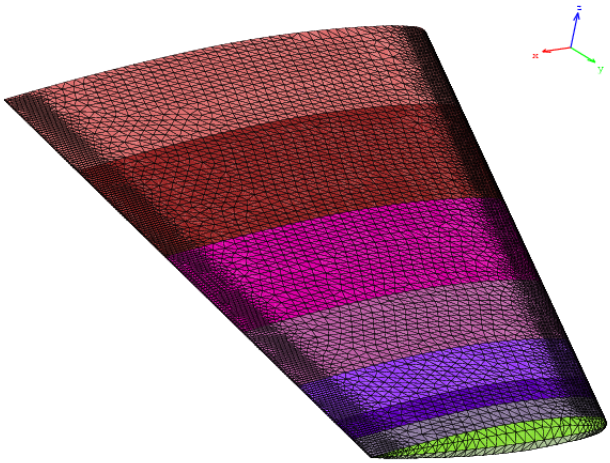
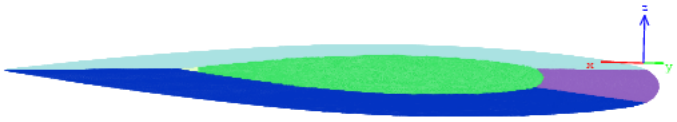


Figure 3.24: Original ONERA M6 wing grid and patches.

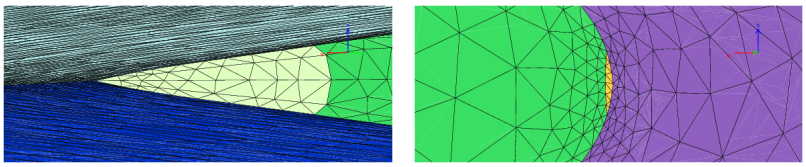
The procedure consisted first in recovering the nodes, at the sections described by the patches borders, which are then used as points to construct the curves. The curves are then split accordingly to the guidelines given earlier in section 3.4, in such a way that no two surfaces have more than one curve in common. The recovered geometry consists of 8 surfaces and 15 curves, the root chord length is 10 units long with the centre of the domain placed at the tip of the of the wing section on the symmetry plane.

Figure 3.25 shows the new patch structure on the wing, which coincides with that of the surfaces, with details of the tip and tail patches with mesh. Finally the far field is described by a half sphere of radius 12.5 times the root chord. This leads to the initial mesh shown in figure 3.26. The free-stream velocity is set to Mach 0.84 and the angle of attack at 3.06°, causing the flow to be transonic. The initial grid consists of 91 379 nodes,

477 262 tetrahedra and 38 926 triangular surface elements.



(a)



(b)

Figure 3.25: Patches of the reconstructed ONERA M6 wing mesh: (a) side view, (b) detail of the tail (left) and tip (right).

For all the following examples, an initial first order solution is carried out to start off the computation, and used in the second stage as a restart for a second order solution. This solution then becomes the base for all subsequent meshes in the first adaptation step. The first and second order solutions are shown in figure 3.27.

A first adaptation step is then performed, with respect to the difference in Mach number. The smoothing factor is kept to a minimum number of iterations and a relatively high boundary constraint in order to avoid as much as possible the unprojected solution to be influenced by excessive smoothing. The results of a computation without the projection algorithm and one with are shown in figure 3.28, where an initial difference in the solution due to the smoother surface, is already visible in the areas of greater surface gradient i.e. at the leading edge and at the tip. For completeness the grids are also shown, although the differences are minimal at this stage, and can only be noticed at the leading edge (fig. 3.29).

Adapting a second time gives us a clearer view of the differences on the solution, due to the effect of projecting the nodes onto the surface. The overall solution gives a rela-

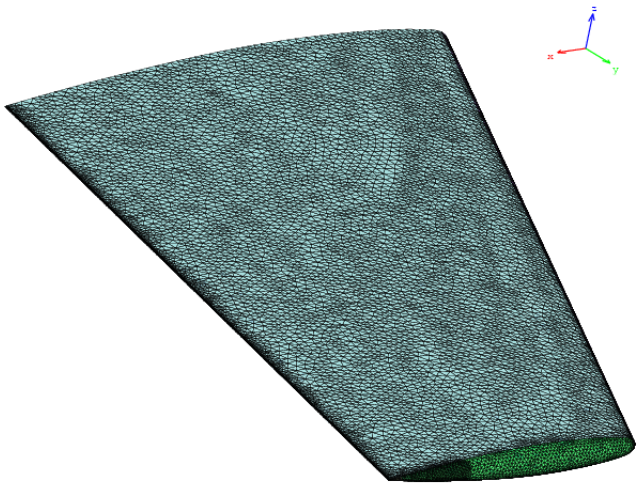
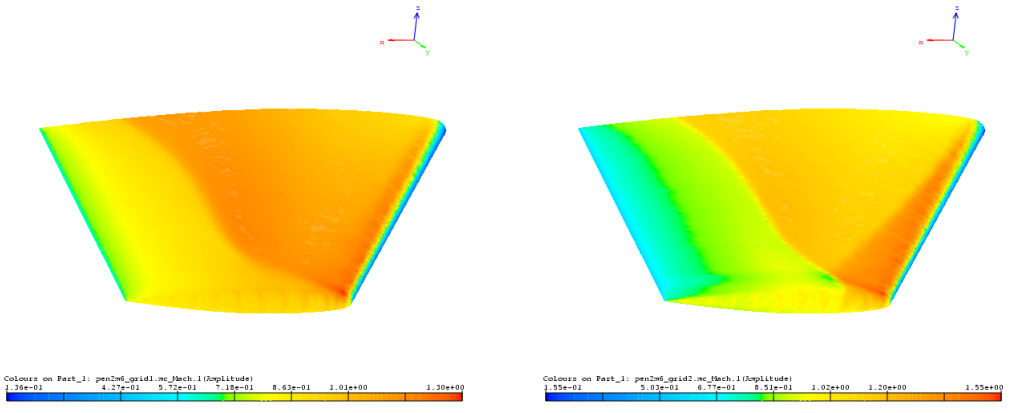


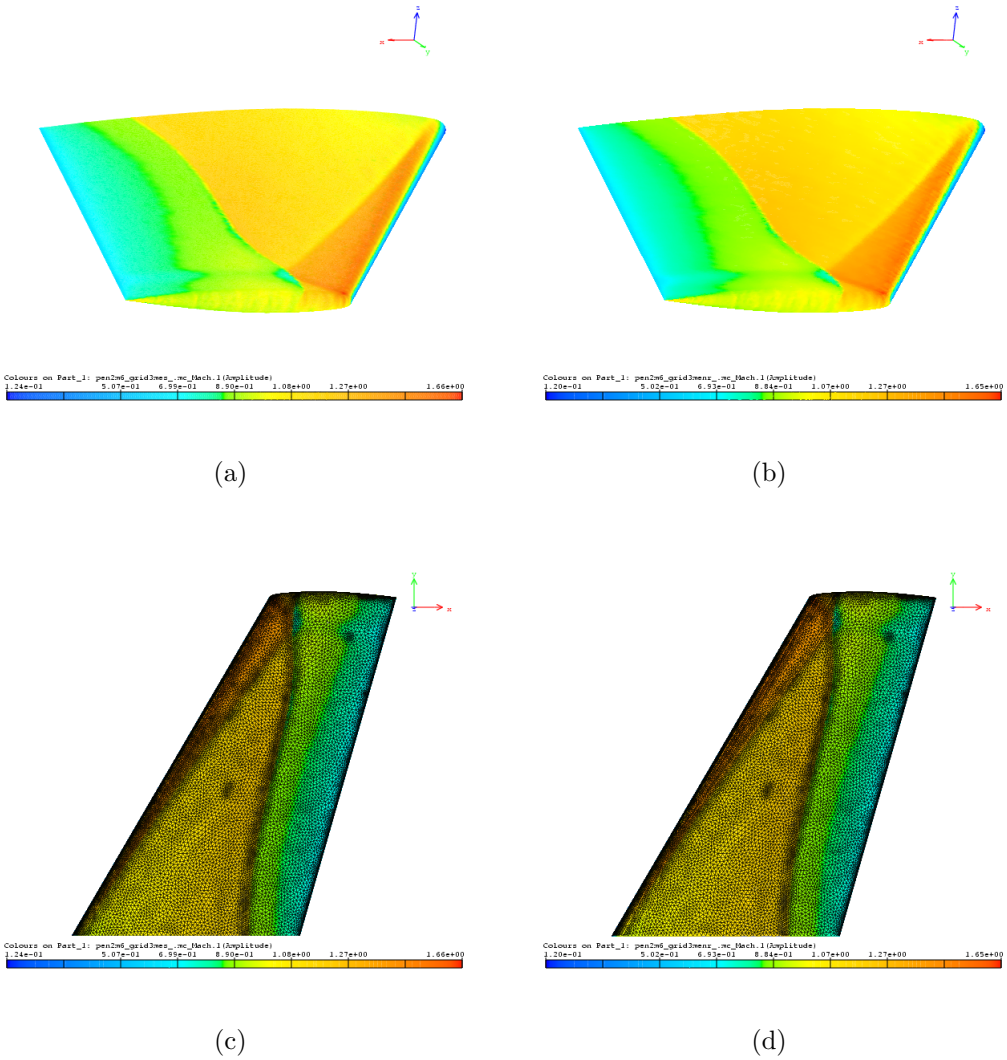
Figure 3.26: Initial ONERA M6 wing mesh.



(a) (b)

Figure 3.27: First and second order solutions(M_∞) of the ONERA M6 wing mesh: (a) initial solution first order, (b) restarted solution second orders.

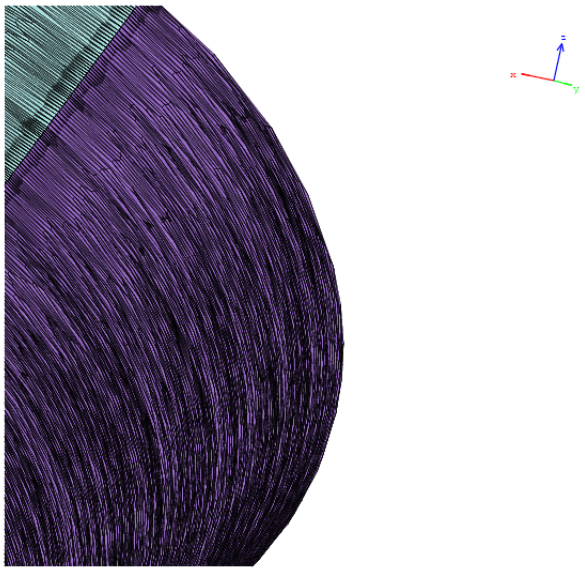
tively good result as we can see from figure 3.30, but even so, the noise produced by the original grid size is clear to see at the tip and at the leading edge in particular. Finally figure 3.31 illustrates the iso-Mach lines and a detail of the solution at the leading edge. In both cases, we can see a net improvement in the computation where the projection is included.



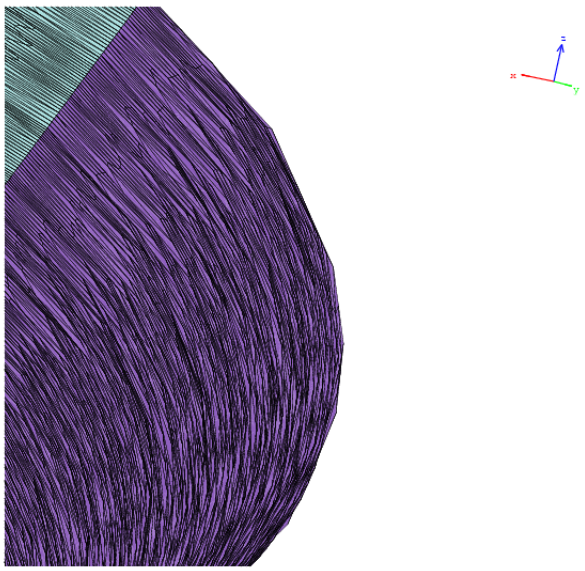
(a) (b)

(c) (d)

Figure 3.28: Solution(M_∞) and grids on the ONERA M6 wing with(left) and without(right) projection.

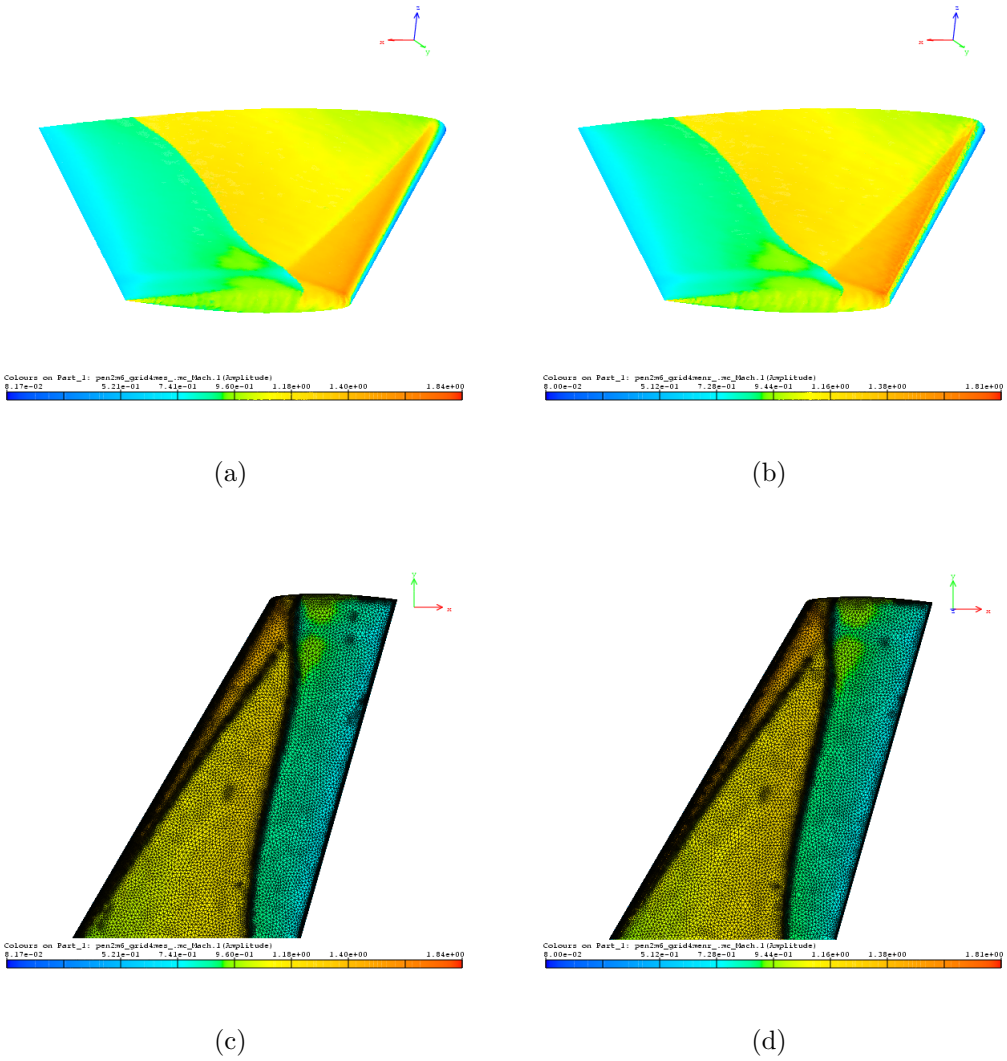


(a)



(b)

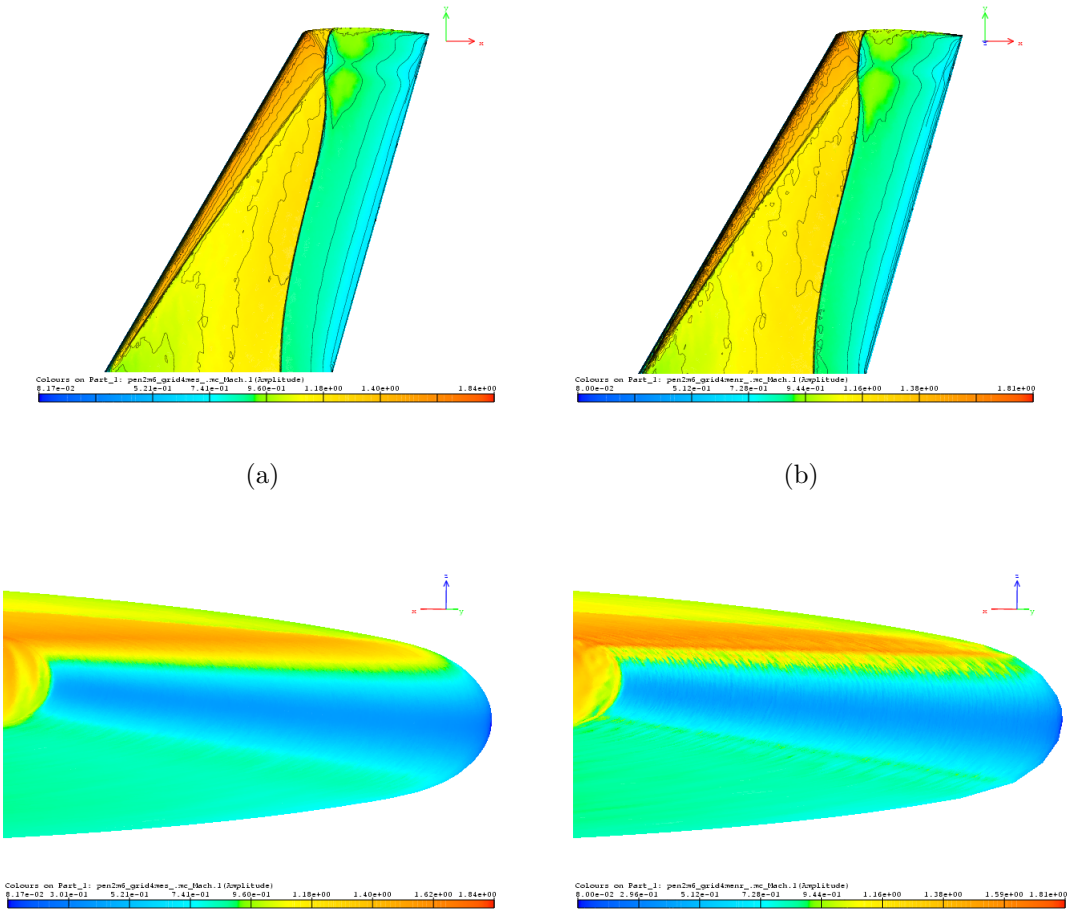
Figure 3.29: Grid on the ONERA M6 wing with(top) and without(bottom) projection.



(a) (b)

(c) (d)

Figure 3.30: Solution(M_∞) and grids after a second adaptation step on the ONERA M6 wing with(left) and without(right) projection.



(a) (b)

(c) (d)

Figure 3.31: Iso-Mach lines and solution (M_∞), with detail of the leading edge, after a second adaptation step on the ONERA M6 wing with(left) and without(right) projection.

4

Geometry Approximation

Geometry conservation as it has just been described, is not always possible due to either faulty, or intricate designs, which require a higher level of complexity in the CAD library. Alternatives to this have been used and proposed in the past, such as describing the geometry with a grid finer than the highest expected mesh resolution[2]. Although relatively crude, this method can be a valid alternative with increasing computer power. If we consider the ratio between surface and volume elements S/V_r, for the case of a tetrahedral 3D transonic wing mesh, this is approximately 0.1 which gives a reasonable number of nodes for the background mesh to be used as geometry. However even this method may not be usable in cases where only a mesh for carrying out the calculations is available, and no geometry is available for creating the background mesh.

It is therefore paramount to have a built-in algorithm able to place the new nodes, belonging to the boundary, closer to the geometric definition of it. Various methods may be used to achieve this, the work by Löhner [27] mentioned in the introductory chapter of this thesis, proposes to reconstruct a geometry from the available triangulation in order to obtain the necessary data for remeshing. Along a similar principle, the work by Baker [31, 28] and more recently in [29] makes use of Hermite interpolation to approximate the boundary surface. These methods however may still pose some problems in the generalised case and use considerable computational resources. Hence an alternative that can also be readily parallelised is searched. Such a procedure is proposed here, based on the idea of interpolating subdivision [123, 124, 60].

4.1 Interpolating subdivision

The underlying idea of this technique is to define a smooth curve or surface as the limit of a sequence of successive refinements. As we can see in figure 4.1, the older subdivision strategy that was in use, needs to have a very good piecewise linear definition of the ge-

ometry where the gradient of the curves and surfaces defining it are high, to begin with. As one can see, the older method consisted in dividing the edges marked for refinement at their midpoint.

What is being introduced here is a weighted average of the nearby old points, which remain undisturbed, as preferable considering they already lie on the surface.
When considering where the new points should be placed, a few properties we would like to have are:

- Simplicity: a restrained set of rules for the subdivision process;
- Efficiency: the computation of the space coordinates should be carried out with as little floating point operations as possible;
- Continuity: properties of the resulting curves and surfaces, such as differentiability;
- Local & Compact: rules based on points lying in the direct neighbourhood which should be small and finite;
- Affine invariance: if the original set of points is subject to a transformation, such as a translation, the resulting shape should also undergo the same transformation.

If we ignore the boundary points for the time being, the example shown in figure 4.1(bottom) uses interpolating subdivision taking into account two points to the right and two to the left, from the centre of the edge where the new point is being introduced. A simple weight formula of $1/16(-1, 9, 9, -1)$ is used, which is very efficient as it involves only 4 multiplies and 3 adds per coordinate (see equation (4.1)[123]).

$$\begin{aligned} x_5 &= \tfrac{1}{16}[(-1)\cdot x_1 + 9\cdot x_2 + 9\cdot x_3 + (-1)\cdot x_4] \ , \\ y_5 &= \tfrac{1}{16}[(-1)\cdot y_1 + 9\cdot y_2 + 9\cdot y_3 + (-1)\cdot y_4] \ . \end{aligned} \tag{4.1}$$

It is also very compact since only 2 neighbouring nodes on either side of the new point location are involved. It has a local definition due to the fact that the weights do not depend on the arrangement of the points and is an affinely invariant rule since the weights sum to one. Simplicity is also obtained since only one rule is used for the interior points, and only one other is needed for the boundaries. Finally continuity is also respected as the limit curves obtained by repetition of the process are C^1 continuous.

One should not forget the point of this addition to the adaptation library, is that of implementing an efficient, compact, fast, and simple algorithm to approximate the placement of the new nodes, coming from the refinement of the mesh, as close as possible to the exact

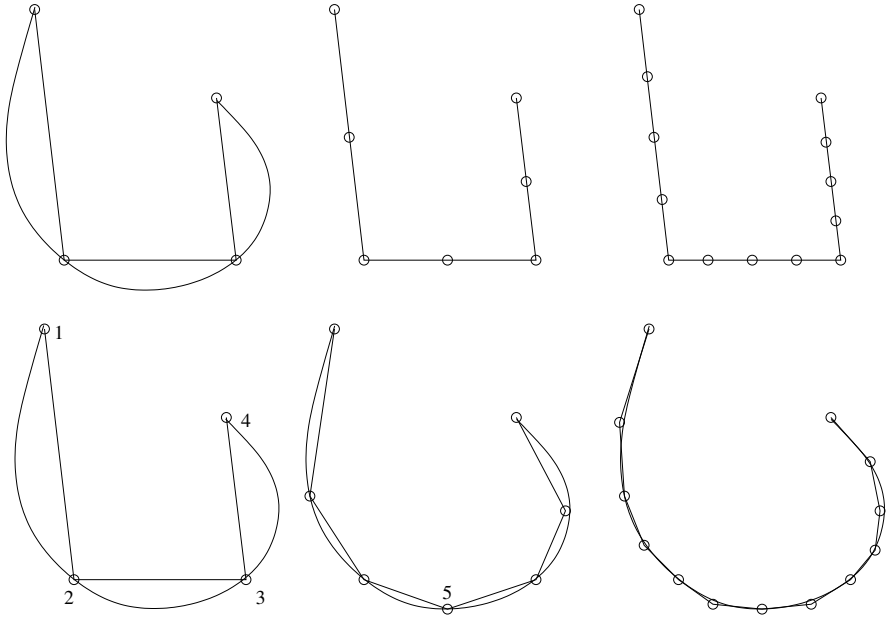


Figure 4.1: Example of interpolating subdivision for a curve in a plane. On the top: midpoint subdivision. On the bottom: Refinement with a piecewise linear curve connecting the points using a weighted average of nearby old points, as in eq (4.1).

geometry in case this is not accessible. This is particularly important for problems where the surface geometry moves and/or when the external field is unsteady. As we can see from the example above all these criteria can be satisfied with interpolating subdivision. If we compare this to other traditional approaches for modelling smooth surfaces we find that it performs better in all the following points:

1. **Efficiency.** Subdivision is easy to implement and computationally efficient as only a small number of neighbour nodes is necessary. Similar to this is knot insertion methods used in spline modelling, which is why many subdivision methods are a generalisation of this. However these are generally not interpolating and old points are moved in the subdivision steps. Implicit surfaces on the other hand are much more costly, and even more so variational surfaces, with a global optimisation problem to be solved at each surface adaptation.

2. **Arbitrary topology.** This consists of two properties, the arbitrary topological genus of the mesh and associated surface, and the arbitrary structure of the graph formed by the edges and vertices of the mesh (specifically the arbitrary degree of each vertex). Spline patches used in arbitrary control meshes, may encounter problems when enforcing higher order continuity at extraordinary vertices (extraordinary means where a number of patches other than four are connected to it in the case of

quadrilateral patches), considerably increasing the complexity of the representation. Implicit surfaces can be of arbitrary topology, but precise location and connectivity of the surface are typically difficult to control. Variational surfaces have high computational costs despite the ability to handle arbitrary topology extremely well. Subdivision can handle arbitrary topology quite well without loss in efficiency.

3. **Surface features.** It is desirable to control features such as creases and sharp edges. Implicit surfaces are very difficult to control since all modelling is performed indirectly and interactions between different parts of the surface can occur. Spline surfaces allow for precise control but introduction of features is computationally expensive and hard to do, particularly when the location of these features can be arbitrary. Variational surfaces provide good flexibility and exact control when creating features. Subdivision is more flexible than splines as the coefficients of subdivision can be manipulated in order to achieve effects such as sharp creases and control the behaviour of boundary curves, in addition to choosing locations of control points.

4.1.1 Basic subdivision scheme distinction

The first and foremost distinction between subdivision schemes is that of *approximating* and *interpolating* schemes [59]. Let us take a vector of points, that will be our control points of a given curve:

$$\mathbf{p} = \begin{bmatrix} \vdots \\ p_{-2} \\ p_{-1} \\ p_0 \\ p_1 \\ p_2 \\ \vdots \end{bmatrix} . \tag{4.2}$$

Let us now define the relationship between these control points at j different levels of subdivision

$$\mathbf{p}^{j+1} = S\mathbf{p}^j \ , \tag{4.3}$$

where S is the subdivision matrix. Using as example the case of the B-spline, the components of S will be

$$S_{2i+k,i} = s_k = \frac{l}{2^l}\binom{l+1}{k} \ , \tag{4.4}$$

where the non-zero entries in each row are the weights of the refinement equation, and successive rows are copies shifted by one column to the right every other row. For further details on the mathematical background we refer the reader to [59].
Looking more closely at one component, i, of our control points we see that

$$p_i^{j+1} = \sum_l S_{i,l}\, p_l^j \ . \tag{4.5}$$

In order to find which s_k is affecting which term, the above can be divided into odd and even entries.
Hence for the odd entries we have

$$p_{2i+1}^{j+1} = \sum_l S_{2i+1,l}\, p_l^j = \sum_l s_{2(i-l)+1}\, p_l^j \ , \tag{4.6}$$

and for the even entries

$$p_{2i}^{j+1} = \sum_l S_{2i,l}\, p_l^j = \sum_l s_{2(i-l)}\, p_l^j \ . \tag{4.7}$$

From which two different subdivision rules can be obtained, one for the new *even* control points of the curve, and one for the new *odd* control points. Another way to look at the distinction between *even* and *odd* subdivision, is to notice that odd points at level $j+1$ are newly inserted, while even points at level $j+1$ correspond to the old points from level j. Subdivision schemes that have these properties are called *interpolating*, since points, once they have been computed, will never move again.

In the case where even points, at level $j+1$, are local averages of points at level j so that $p_{2i}^{j+1} \neq p_l^j$, the scheme is called *approximating*. This last is the case of cubic splines, whilst the previous one is that of the 4 point scheme [123] used for the example in figure 4.1. Note that the subdivision matrix S encodes the refinement equation for the B-spline in the case where this is used in the definition of the curve.

Let us consider for example the cases shown in figure 4.2. Here we can see how in the approximating subdivision scheme the location of the old points is also going to be modified, by noticing how all the points in the new level are influenced by more than one point in the previous level. We can also see this by examining the relationship between control points at different levels of subdivision, from equation (4.3) using the subdivision matrix for the cubic B-spline:

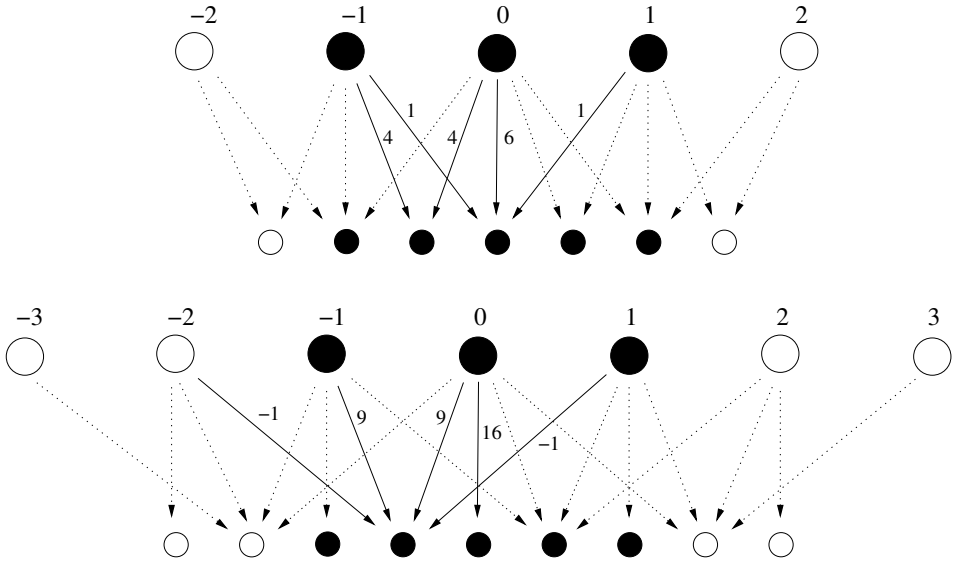


Figure 4.2: Approximating and interpolating schemes. On the top the case of a cubic B-spline approximating subdivision. The bottom case is the 4 point interpolating subdivision rule.

$$\begin{pmatrix} p_{-2}^{j+1} \\ p_{-1}^{j+1} \\ p_{0}^{j+1} \\ p_{1}^{j+1} \\ p_{2}^{j+1} \end{pmatrix} = \frac{1}{8} \begin{pmatrix} 1 & 6 & 1 & 0 & 0 \\ 0 & 4 & 4 & 0 & 0 \\ 0 & 1 & 6 & 1 & 0 \\ 0 & 0 & 4 & 4 & 0 \\ 0 & 0 & 1 & 6 & 1 \end{pmatrix} \begin{pmatrix} p_{-2}^{j} \\ p_{-1}^{j} \\ p_{0}^{j} \\ p_{1}^{j} \\ p_{2}^{j} \end{pmatrix} . \tag{4.8}$$

If instead we look at the interpolating 4 point subdivision scheme as shown in the example in figure 4.2, we can see how only the new points are affected by the points in the previous level. In this case the relationship between control points at different levels of subdivision in equation (4.3) is as follows

$$\begin{pmatrix} p_{-3}^{j+1} \\ p_{-2}^{j+1} \\ p_{-1}^{j+1} \\ p_{0}^{j+1} \\ p_{1}^{j+1} \\ p_{2}^{j+1} \\ p_{-3}^{j+1} \end{pmatrix} = \frac{1}{16} \begin{pmatrix} -1 & 9 & 9 & -1 & 0 & 0 & 0 \\ 0 & 0 & 16 & 0 & 0 & 0 & 0 \\ 0 & -1 & 9 & 9 & -1 & 0 & 0 \\ 0 & 0 & 0 & 16 & 0 & 0 & 0 \\ 0 & 0 & -1 & 9 & 9 & -1 & 0 \\ 0 & 0 & 0 & 0 & 16 & 0 & 0 \\ 0 & 0 & 0 & -1 & 9 & 9 & -1 \end{pmatrix} \begin{pmatrix} p_{-3}^{j} \\ p_{-2}^{j} \\ p_{-1}^{j} \\ p_{0}^{j} \\ p_{1}^{j} \\ p_{2}^{j} \\ p_{3}^{j} \end{pmatrix} . \tag{4.9}$$

Another interesting property of subdivision is that it has a geometric speed of convergence, in other words the difference between the actual curve and the piecewise one decreases by a constant factor on every subdivision step. However the most important feature is linked to the subdivision matrix, which can be used to create a variety of different curves by manipulating its coefficients. For example it could be changed within a subdivision level or between subdivision levels. This becomes particularly interesting where features such as sharp edges need to be treated differently.

4.1.2 Surface subdivision techniques

The variety of schemes available is widespread and it is not our purpose to present an exhaustive discussion of these. However we should specify the criteria used to classify most schemes, which are:

- the type of refinement rule (face split or vertex split);
- the type of generated surface mesh (triangular or quadrilateral);
- whether the scheme is approximating or interpolating;
- smoothness of the limit surfaces for regular meshes (C^1, C^2 etc.).

A summary of the most widely known and used type of schemes is given in table 4.1[59].

Face Split		
	Triangular meshes	*Quad meshes*
Approximating	Loop(C^2)	Catmull-Clark (C^2)
Interpolating	Mod. Butterfly (C^1)	Kobbelt (C^1)

Vertex split
Doo-Sabin, Midedge (C^1)
Biquadratic (C^2)

Table 4.1: Schemes classification.

The first choice that must be made is that of the type of mesh that is going to be used for the subdivision process. We are only interested in surface meshes that are triangular for

now, but it is important to know that such schemes are available for quadrilateral meshes for future research to be done with hybrid grids. Once the tiling of the plane is fixed, the decision that has to follow is how to refine it. The two main approaches to do this are the face or the vertex split, which can also be referred to as primal or dual schemes. In the first case the edges of a face are split, whilst in the second the vertex is split into the faces surrounding it (see figure 4.3).

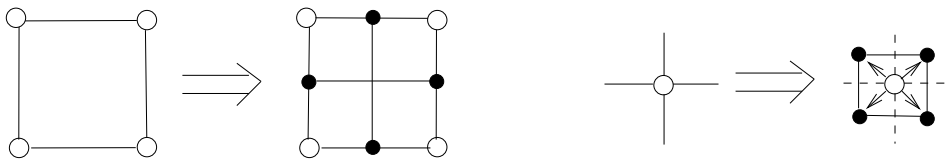


Figure 4.3: To the left the face split scheme. To the right the vertex split.

For demonstration purposes only the quadrilateral tilings have been shown in figure 4.3 as in the case of triangles vertex split schemes result in non-nesting hexagonal tilings, whilst quadrilateral meshes support both subdivision schemes easily. Once again we will concentrate on only one scheme, the face split, due to the fact that the adaptation module splits the edges when refining a mesh.

As in the case of curves previously examined, face-split schemes can be either interpolating or approximating. It is our choice therefore to concentrate onto the interpolating schemes since we would like to maintain the original control points in the same position, as they are on the exact surface.

Therefore only part of the modified butterfly scheme [124, 60] is going to be taken into account and used as a basis onto which construct our interpolatory subdivision for approaching the new node placements to the exact surface or curve. On boundary and creases, where usually curves will be defining the geometry in our CAD design, the 4 point subdivision scheme as shown in the previous sections should be used. This is made possible by the use of patches as in the previous chapter, whereby nodes belonging to more than one patch can be identified and the 4 point scheme used. On interior points the stencil shown in figure 4.4 can be used.

Some boundary rules must be set though for points to be added on vertices which do not lie on borders, but are connected to them. A reduced set of boundary and crease rules includes three main types: regular interior-crease, regular crease-crease 1, and regular crease-crease

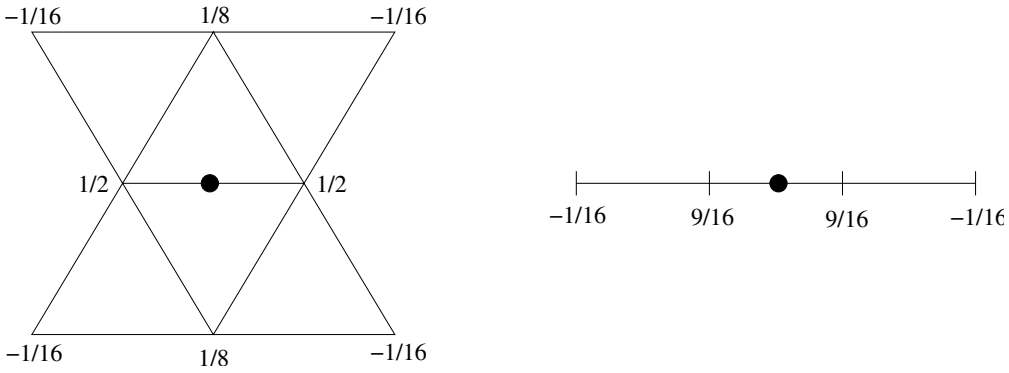


Figure 4.4: To the left the normal butterfly scheme for internal vertices. To the right the mask for crease and boundary nodes.

2. To put it all into a system, the main cases can be classified by types of head and tail vertices of the edge on which we add a new vertex. Some of these are illustrated in figure 4.5.

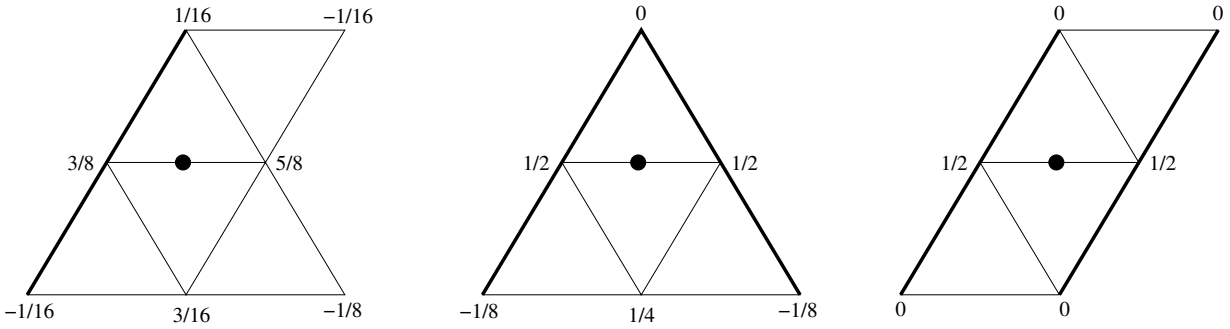


Figure 4.5: Regular modified butterfly boundary and crease rules. From left to right respectively: interior-crease rule, crease-crease rule 1, and crease-crease rule 2.

It is clear that some modifications and additions must be brought to the scheme in order to adapt it to our needs. A quick example of this is the possibility of having a border, from the newly added node in the crease-crease rule 1 in figure 4.5 to the top node, and then wanting to refine that edge. This is clearly an exceptional case, but it may happen as we are not dealing with classical subdivision, where all the edges are refined, but using the idea to apply it to our case in order to make node insertion closer to the underlying geometric boundary.

4.2 Modified subdivision techniques

Although the butterfly subdivision scheme adapts well to near-regular grids, and allows for very good approximations at creases and boundary nodes with the four point subdivision scheme, it may lack some precision when faced with interior sharp features and with more irregular grids. In fact it is not so obvious how to define a sharp feature in a mesh, which is then recognisable, without manipulating the mesh itself.

The objective in fact remains that of being able to use this interpolation when the geometry is too complex or intricate for a relatively simple use with the CAD library, and sometimes it may not be possible to place a patch intersection at every sharp feature present. This last method remains in fact the only safe and secure way, without a geometric description, to ensure the feature is maintained intact. Therefore some modifications to tune the schemes to our needs have been made. In particular we have concentrated on the 3D aspect of the scheme, since the 2D scheme implemented, suffers less the problems noted above.

4.2.1 Distance based butterfly scheme

Let us consider the butterfly scheme just described, and modify it as shown in figure 4.6.

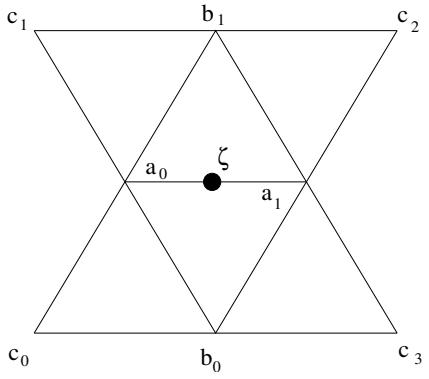


Figure 4.6: Adapted modified butterfly scheme.

If we now try to express the same scheme as before with these nodes, it would give us:

$$\boldsymbol{\zeta} = \frac{1}{16}\{8[\mathbf{a}_0 + \mathbf{a}_1] + 2[\mathbf{b}_0 + \mathbf{b}_1] - [\mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3]\} \ , \tag{4.10}$$

where $\boldsymbol{\zeta}$ is the new point inserted. A new system of weights is needed to take into account the possible grid irregularity. Taking the distance between each node and a virtual

mid-segment point $\boldsymbol{\xi}$, it may then be used as an inverse proportional weight to allow for closer nodes to have a stronger effect on the new position than the farther ones. However, since we would like to keep the key weights of the scheme above, we only apply this to the different areas of the scheme.

Let us first define the weights, which remain unchanged for the nodes at the end of the split segment, since $\boldsymbol{\xi}$ is at half way between them, and for the others are:

$$w_{b_i} = \frac{1}{\|\boldsymbol{\xi} - \mathbf{b}_i\|} \quad , \quad w_{c_i} = \frac{1}{\|\boldsymbol{\xi} - \mathbf{c}_i\|} \ , \tag{4.11}$$

where $\|\cdot\|$ stands for the Euclidean norm, and,

$$W_b = \sum_{i=1}^{2} w_{b_i} \quad , \quad W_c = \sum_{i=1}^{4} w_{c_i} \ . \tag{4.12}$$

Using these weights in equation (4.10) we obtain the following:

$$\boldsymbol{\zeta} = \frac{1}{16}\{8[\mathbf{a}_0 + \mathbf{a}_1] + 2[2\frac{1}{W_b}(\sum_{i=1}^{2} w_{b_i}\mathbf{b}_i)] - [4\frac{1}{W_c}(\sum_{i=1}^{4} w_{c_i}\mathbf{c}_i)]\} \ . \tag{4.13}$$

In this way the original weight distribution is still: 1 for the referring segment nodes, 1/4 for the two nodes opposite the segment, and $-1/4$ for the four nodes directly related to ones surrounding the segment. At the same time, an internal weight repartition based on the inverse of the distance is achieved. Note that for the boundary rules shown in figure 4.5 particular care must be taken since it is not so straightforward to apply. This is due to the different weights given because of the loss of symmetry in the scheme.

4.2.2 Multi-node scheme modification

A further modification of this scheme is represented in figure 4.7. Here, rather than limiting the scheme to four surrounding nodes, we extend it to all the nodes connected to the referring segment end nodes. The system used remains very similar to the one just outlined in the above paragraph.
The nodes which are not directly related to the cells having the segment in common, are split in two groups, each associated with one of the two **a** nodes. The definition for the weights of nodes **b** remains the same as in equation (4.11), the same is also true for **c**, with the addition of:

$$w_{d_i} = \frac{1}{\|\boldsymbol{\xi} - \mathbf{d}_i\|} \ , \tag{4.14}$$

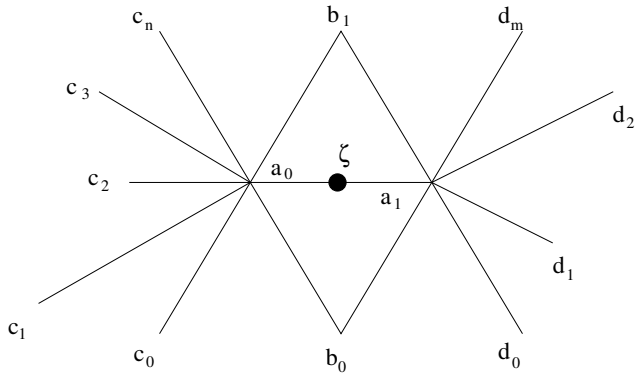


Figure 4.7: Adapted modified butterfly scheme with multiple neighbours.

and whilst **b** remains unchanged also in (4.12), the definition for **c** is changed and that of **d** is added:

$$W_c = \sum_{i=1}^{n} w_{c_i} \quad , \quad W_d = \sum_{i=1}^{m} w_{d_i} \ . \tag{4.15}$$

Adapting this new stencil to the system expressed in (4.13) we obtain:

$$\boldsymbol{\zeta} = \frac{1}{16}\{8[\mathbf{a}_0 + \mathbf{a}_1] + 2[2\frac{1}{W_b}(\sum_{i=1}^{2} w_{b_i}\mathbf{b}_i)] - [2\frac{1}{W_c}(\sum_{i=1}^{n} w_{c_i}\mathbf{c}_i)] - [2\frac{1}{W_d}(\sum_{i=1}^{m} w_{d_i}\mathbf{d}_i)]\} \ , \tag{4.16}$$

where n and m are the neighbour nodes of $\mathbf{a}_0$ and $\mathbf{a}_1$ respectively. As we shall see in the following sections, this scheme allows for a somewhat more precise idea of the surrounding nodes positions, without disrupting the simple and effective weight system on which it is based.

This is possible since the internal weighting system of each area of the scheme adds up to unity, which is the reason why it must then be multiplied by an extra weight on top of that from the scheme. It is clear to see this when comparing (4.10) and (4.13), where a factor 2 has been introduced for the **b** nodes, and a factor of 4 for the **c** nodes. Hence in (4.16) it does not matter whether n and m are different, or how many there are, as in the end the sum of the internal weights is multiplied by it's inverse, bringing it back to unity (weight wise). However, the drawback with regards to the previous stencil is a lesser variety of stencils for particular cases.

4.2.3 Diamond difference based scheme

A more basic method, founded on directional distances, has also been devised. If we consider the stencil shown in figure 4.8, only the surface elements that share the edge to be refined are used to interpolate the new point position. The particular to notice in the stencil is the dashed line representing the imaginary edge between the **b** nodes. We start by measuring first the directional changes along the two internal edges of the diamond:

$$dx_a = \|x_{a_1} - x_{a_0}\| \quad , \quad dx_b = \|x_{b_1} - x_{b_0}\| \ , \tag{4.17}$$

and in a similar manner we get dy_a, dy_b, dz_a and dz_b, and obtain:

$$Da = dx_a + dy_a + dz_a \quad , \quad Db = dx_b + dy_b + dz_b \ . \tag{4.18}$$

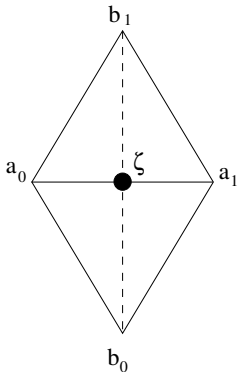


Figure 4.8: Diamond stencil with fictitious edge dashed.

In order to proceed, we introduce two variables ϱ and ς. The scope of these two variables is to act as checks on the directional variation along the two internal edges. This allows to estimate whether the use of one segment midpoint or the other is more appropriate for the new point placement. In particular, we may chose to use the midpoint of only one segment if the following conditions are satisfied:

$$\begin{cases} dx_b > \varrho dx_a \\ dy_a > \varsigma dy_b \\ dz_a > \varsigma dz_b \end{cases} \text{ or } \begin{cases} dy_b > \varrho dy_a \\ dx_a > \varsigma dx_b \\ dz_a > \varsigma dz_b \end{cases} \text{ or } \begin{cases} dz_b > \varrho dz_a \\ dx_a > \varsigma dx_b \\ dy_a > \varsigma dy_b \end{cases} \tag{4.19}$$

to use only $\overline{\mathbf{b}_0\mathbf{b}_1}$ segment, and:

$$\begin{cases} dx_a > \varrho dx_b \\ dy_b > \varsigma dy_a \\ dz_b > \varsigma dz_a \end{cases} \text{ or } \begin{cases} dy_a > \varrho dy_b \\ dx_b > \varsigma dx_a \\ dz_b > \varsigma dz_a \end{cases} \text{ or } \begin{cases} dz_a > \varrho dz_b \\ dx_b > \varsigma dx_a \\ dy_b > \varsigma dy_a \end{cases} \tag{4.20}$$

to use only $\overline{\mathbf{a}_0\mathbf{a}_1}$.

The idea behind this is simple. Consider one direction only, to begin with, along the two internal segments of the diamond. If the variation along that direction is clearly greater on one segment than the other, we continue to consider the other two directions. If the variation in both these directions is greater on the other segment respect to the previous one, then we consider only the segment with the greater change in only one direction. The values of ϱ and ς are therefore variable. Their values depend on where we want to set the limits of how much greater the difference in the directions must be to use only one set of nodes rather than the other. Here values of 1.5 for ϱ and 1.1 for ς were used. When these above conditions aren't satisfied, the position of $\boldsymbol{\zeta}$ is calculated using the midpoints $\boldsymbol{\xi}$ and $\boldsymbol{\chi}$ of the internal segments between the set of points $\mathbf{a}$ and $\mathbf{b}$ respectively. Then we use Da and Db from eq. (4.18) as weights in the following way:

$$\boldsymbol{\zeta} = \frac{1}{D_{ab}} \left(\frac{1}{D_a} \boldsymbol{\xi} + \frac{1}{D_b} \boldsymbol{\chi} \right) , \tag{4.21}$$

where

$$D_{ab} = \frac{1}{D_a} + \frac{1}{D_b} . \tag{4.22}$$

In this way all four points of the diamond stencil will influence the new point position. Also the influence is inversely proportional to the overall variation in the x, y and z directions, along the two internal edges. Since the points to which the weights will be applied are the midpoints of the edges, the new point will lie somewhere along the straight line connecting these two points.

4.3 Implementation

First we take into consideration the two dimensional cases, which are relatively simple to implement and have only one type of rule for border nodes. The normal interpolation rule used for the boundary, is the four point rule of equation (4.1) shown also in figure 4.4 on the right. The border rule introduced here is shown schematically in figure 4.9 and in equation:

$$\boldsymbol{\zeta} = \frac{1}{18}[(-1)\cdot \mathbf{a}_0 + 10\cdot \mathbf{a}_1 + 9\cdot \mathbf{a}_2] \ . \tag{4.23}$$

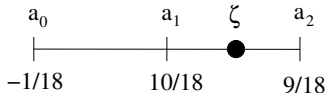


Figure 4.9: Border node rule for 2D four point rule.

A further control is introduced to avoid the accidental use of the normal rule when in presence sharp features (such as a cusp), when these haven't been modelled as separate patches in order to distinguish a border zone. This is done by checking that the angle between two segments is less than a given parameter set by the user beforehand. An example of these angles is given in figure 4.10. The result is that if one of the angles is too high, the border rule is used with the other segment satisfying the condition. If both segments fail to pass the control, then the midpoint of the segment is taken. The same is also done when examining a border rule case, with the angle greater than the set value.

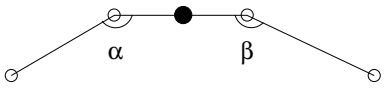


Figure 4.10: Angle control on four point scheme.

We can now move onto the three dimensional implementation. This is less straightforward than the previous one, since we are now dealing with surfaces rather than curves in a plane. In particular, all the different cases must be thoroughly examined before modifying the scheme for special rules to be added. Let us first consider the distance based butterfly scheme. Here the hardest part is finding the nodes which are associated to the scheme for each particular edge marked for refinement. Then the various surface elements contained in the stencil are associated to a patch. At this point all the elements of the stencil are considered, and the ones belonging to the same patch are picked to find the correct scheme to apply. This clearly means that if the two principal faces, adjacent to the segment that is being refined, belong to different patches, then the segment lies on the border and the crease four point rule is applied.

Moving onto the multi-node scheme, we encounter similar problems to those of the above scheme. The process is slightly more straightforward, in the sense that all the surrounding nodes, of vertices of the edge to be refined, are identified. The patch association is now done on the nodes rather than faces, and the borders are recognised by end nodes of a segment belonging to the same multiple patches. In this way a series of checks may be carried out to control the rule that should be applied. As in the previous case, if a segment lies on the border of two patches, then the crease four point rule is applied. If instead, only one of the two nodes of the edge to be refined lies on the boundary, then the neighbouring nodes of that vertex are ignored and the scheme (4.16) changed to:

$$\boldsymbol{\zeta} = \frac{1}{18}\{9[\mathbf{a}_0] + 7[\mathbf{a}_1] + 2[2\frac{1}{W_b}(\sum_{i=1}^{2} w_{b_i}\mathbf{b}_i)] - [2\frac{1}{W_c}(\sum_{i=1}^{n} w_{c_i}\mathbf{c}_i)]\} \ , \tag{4.24}$$

in the case where node $\mathbf{a}_1$ belongs to more than one patch. There exists the possibility that the other border nodes are not the $\mathbf{b}$ points and therefore *some* of the $\mathbf{d}$ vertices could still be taken into account. However, for simplicity, the rule in eq. (4.24) is preferable.

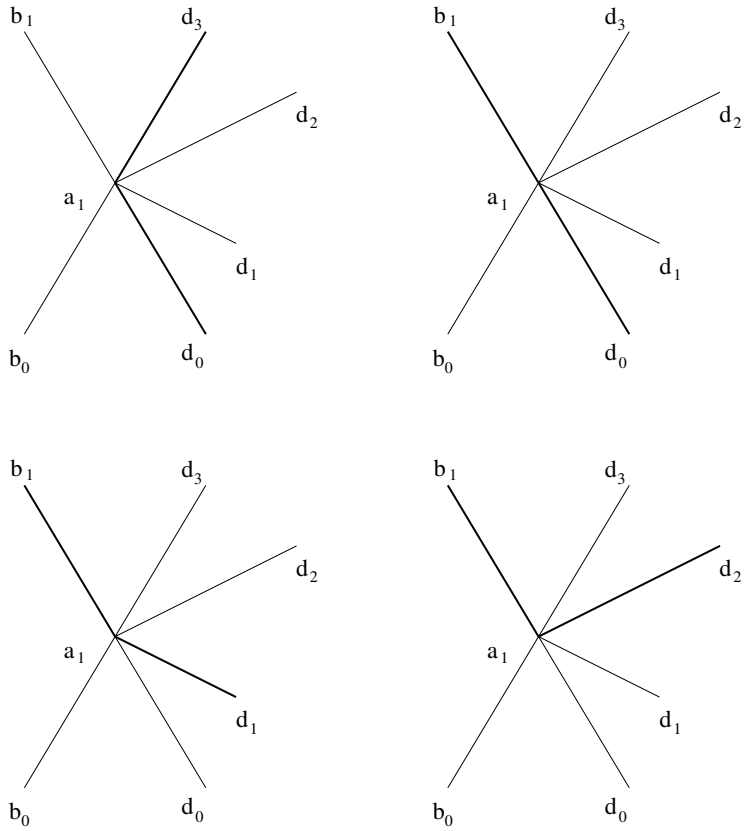


Figure 4.11: Border possibilities example for multi-node scheme.

If we consider the different possibilities shown in figure 4.11, we find that: the first on the top left shows a possible configuration which would justify the use of at least two nodes **d** in the stencil, the top right and bottom left configurations would induce an unfavourable point influence coming from only point $\mathbf{d}_0$ in the first case and also point $\mathbf{d}_1$ from the other. For the top right and lower left cases a new stencil should be used, but the difficulty lies in identifying when to use it, as it could be erroneously applied to the case in the bottom right of the figure, where the normal stencil would perform better, excluding only node $\mathbf{d}_3$.

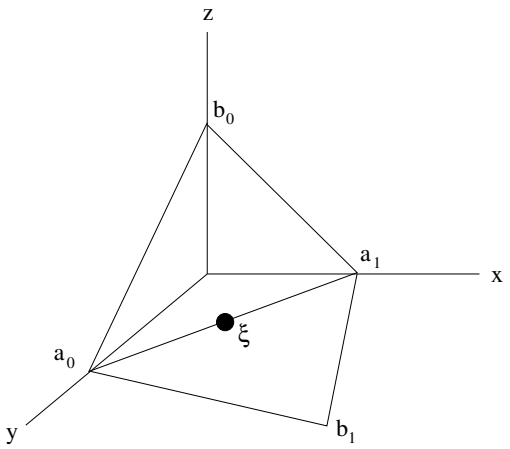


Figure 4.12: Directional distance control.

A further check is put in place to see which stencil to use and avoid inconsistencies during the subdivision process. The use of the directional variance in the mesh, such as that described in eq. (4.17) is particularly interesting. Let us consider our usual midpoint $\boldsymbol{\xi}$ (fig. 4.12) and find the directional distance from this to the vertices **a** and **b**,

$$dx_{a_0} = \|x_\xi - x_{a_0}\| \ , \ dx_{a_1} = \|x_\xi - x_{a_1}\| \ , \ dx_{b_0} = \|x_\xi - x_{b_0}\| \ , \ dx_{b_1} = \|x_\xi - x_{b_1}\| \ . \quad (4.25)$$

The average difference in the x direction is then computed as:

$$d\bar{x} = \frac{dx_{a_0} + dx_{a_1} + dx_{b_0} + dx_{b_1}}{4} \quad (4.26)$$

the same is also done for y and z directions. Note that when in a plane, such as the symmetry plane, one of the average values will be zero. Since it will be used for a division as we shall see in the following paragraph, it is set to a computational zero in order to avoid errors. This will guarantee the proper functioning of what follows without manipulating its significance. As we shall see, it is also true that if the average adds up to zero, then

the value than we shall divide will also be zero, leaving unchanged the idea behind the operation.

We now proceed to the second step of this check, which consists in comparing the change in direction of the **b** vertices, with respect to the average:

$$\kappa^x_{b_0} = \frac{dx_{b_0}}{d\bar{x}} \quad , \quad \kappa^x_{b_1} = \frac{dx_{b_1}}{d\bar{x}} \ , \tag{4.27}$$

with the same procedure applied to the other directions also. The limit for the $\boldsymbol{\kappa}$ variable introduced is user dependent, and should be set low enough to avoid anomalies, yet high enough to avoid restraining the choice to just the midpoint rule. In fact if any of the values of $\boldsymbol{\kappa}$ is greater than a certain tolerance, the midpoint rule is applied. The same procedure is then applied for each of the neighbouring nodes. In case a node does not satisfy the condition and is greater than a set tolerance (specific for neighbouring nodes), it is neglected.

Another control measure is that of evaluating the angles inside the two faces adjacent to the marked segment. In particular the two angles at the vertices of the segment as shown in figure 4.13. If one of the angles is greater than a set value then the surface element is considered too distorted for use with the scheme. The reference angle was chosen at 80°.

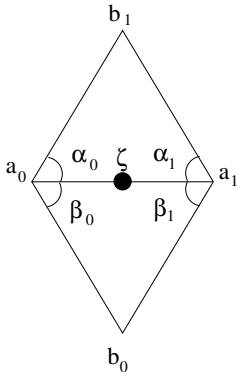


Figure 4.13: Angle control at refined segment vertices.

Finally we look at the diamond difference based scheme. As in the distance butterfly case, we find what patches the two faces adjacent to the edge belong to. In this way the borders are found and the four point crease rule may be used. Also, the same angle check in place for the multi-node scheme shown in figure 4.13, is used.

4.4 Applications

The effectiveness of the procedures introduced is then verified with two test cases. The first is a NACA 0012 airfoil, used for the 2D validation. The second test case is represented by the concept aircraft. SmartFish shown in figure 2.27. This is a particularly difficult test case to work with, due to the continuously varying geometry, sharp corners and cusps present. Also for these cases, solutions were obtained using the parallel, unstructured grid, Euler solver THOR. The geometries are described with non-dimensional units in all cases.

4.4.1 2D: NACA 0012 airfoil

The transonic flow conditions, with Mach number 0.85 have been used to test the case. Two angles of attack have been considered, with $\alpha = 0°$ and $\alpha = 1°$. The first in particular, to see the influence on the smoothness of the solution and adaptation process, for the symmetric case. The airfoil chord is 1 unit long with origin at the tip, and the far field is of radius 20 units. The original mesh is described with 2 355 nodes, 4 537 triangular elements, and 173 line elements, as shown in figure 4.14. In both angles of attack computations, the first two steps are done adapting the grid to a first order solution, and in the third step a second order computation without adaptation is carried out. Following that, the mesh is always adapted to a second order solution, with respect to an a posteriori error estimation of the Mach number on the node, that will be discussed later in this work.

Let us first examine the zero angle of attack case. Figures 4.15 to 4.17 show the various steps in the adaptation process, and how this correctly refines the mesh in the areas where flow features are more interesting. In particular the tip of the airfoil, and the area where the shocks occur are well adapted. As we may see from the first figure (4.15), the grids are very similar in the first steps of the process. However, in figure 4.15 (c) slight differences are perceptible, such are the greater refinement in the tip zone and on the lower side of the airfoil after the shock, in the mesh without the subdivision being used. This is less apparent when looking at the final grids and solution from slightly further away, as in figure 4.16 (a) and (b). It is clearer when looking at part (c), where the negative C_p is shown along the airfoil. Here the tip values are greatly disturbed in the part without subdivision, as well as having the upper and lower shocks not coinciding. A detailed view of the tip region after the final adaptation is shown in figure 4.17, where the influence of the original grid adapted with the midpoint rule is more evident.

In order to view the detail of the difference between the subdivided grid and the one using the midpoint division, an enhancement of the tip is shown in figure 4.18. The further enhancement in (c) shows how the subdivision continues to interpolate the new grid points

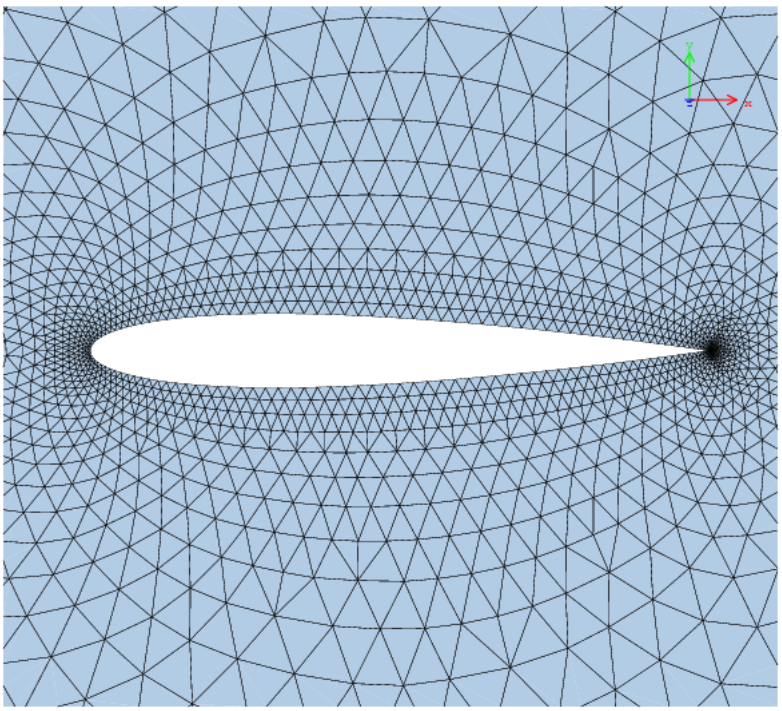
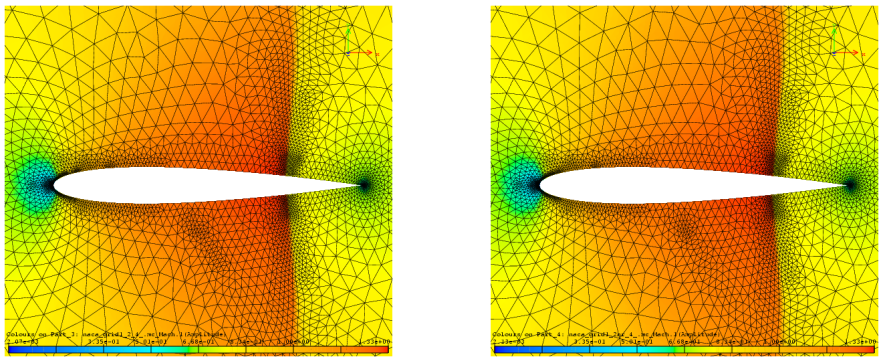


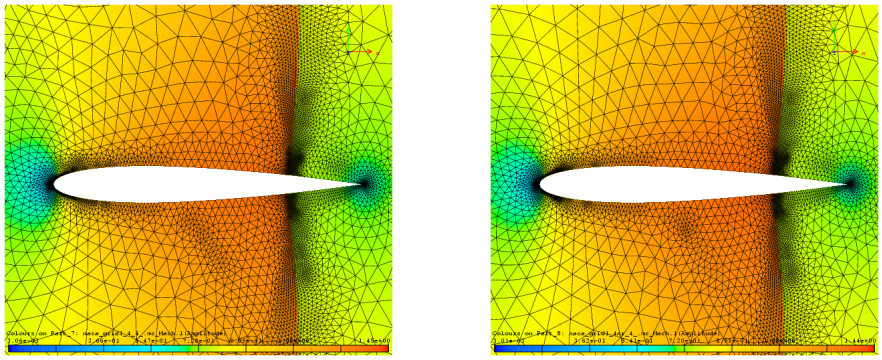
Figure 4.14: NACA 0012 initial mesh.

maintaining the ones from previous levels fixed.

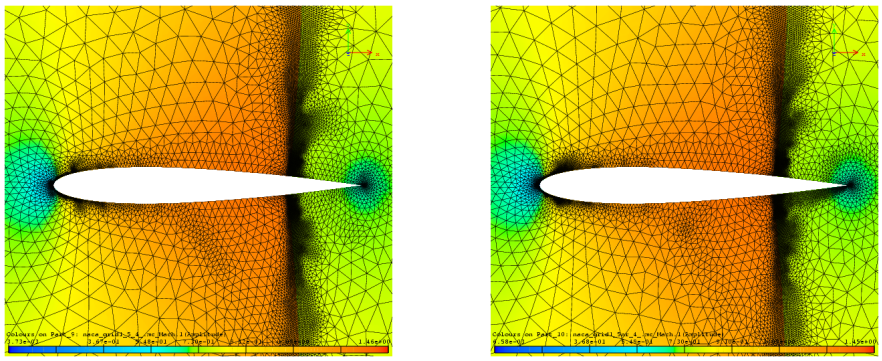
Passing onto the angle of attack 1° case, we can see once again how the solution is more disturbed at the tip in the computation where subdivision was not used (fig. 4.19). Also, in figure 4.20, a detail of the mesh downstream illustrates how the far field is also well described by the subdivision with respect to the normal midpoint division.



(a)

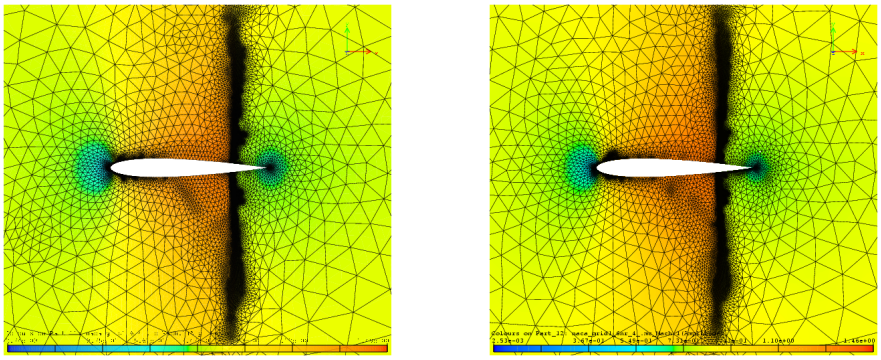


(b)

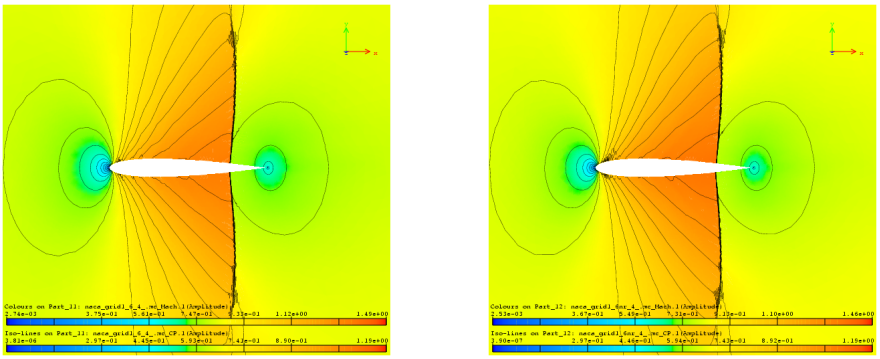


(c)

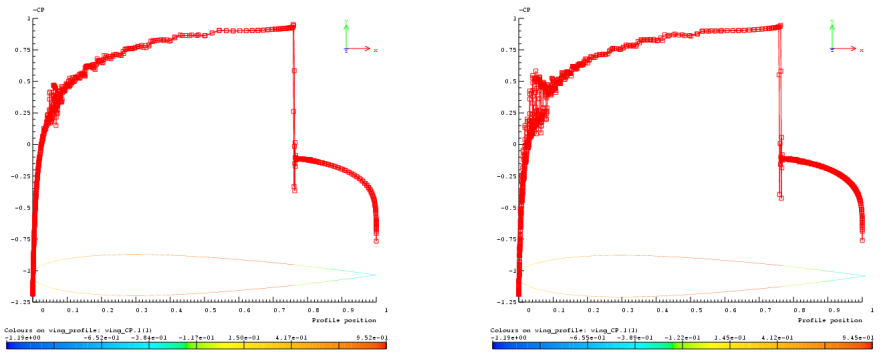
Figure 4.15: NACA 0012 adaptation process, on the left with subdivision, on the right without, grids and solution after: (a) 2 steps, (b) 3 steps, (c) 4 steps. Solution field is Mach number.



(a)

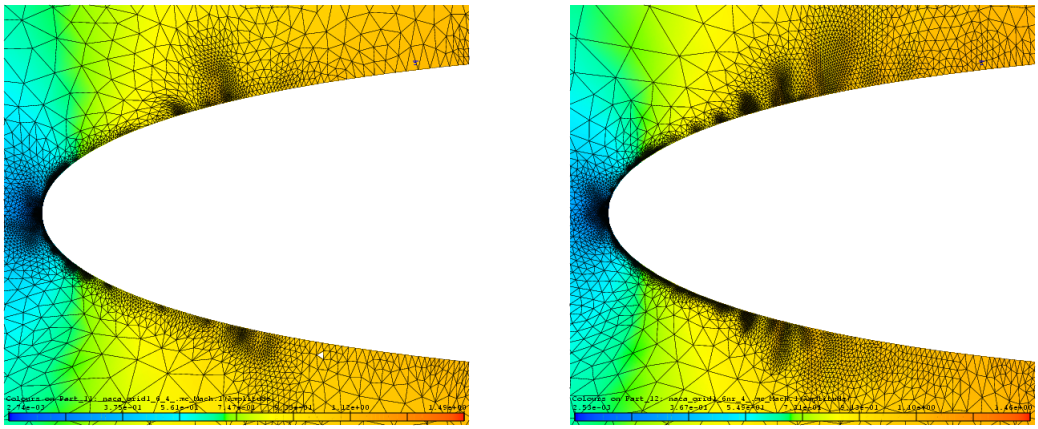


(b)

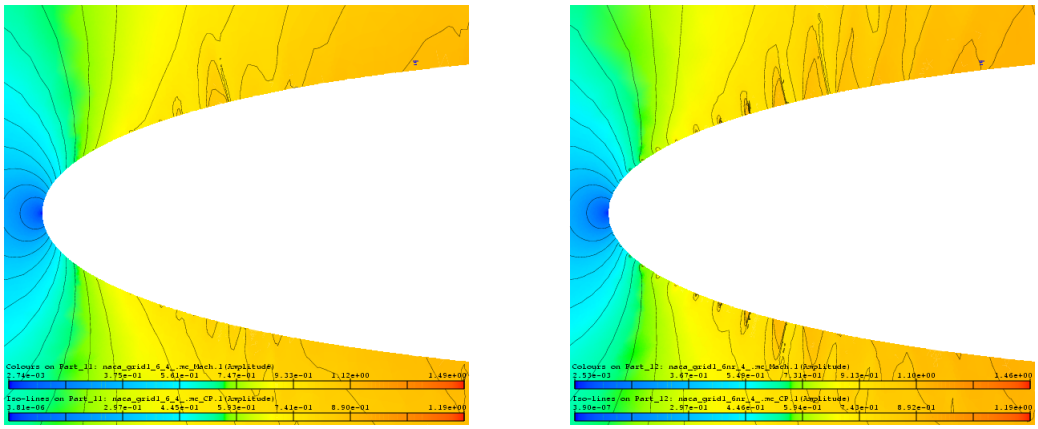


(c)

Figure 4.16: NACA 0012 adaptation process, on the left with subdivision, on the right without: (a) grids and solution after 5 steps, (b) solution, (c) $-C_p$ on the wing. Solution field is Mach number, isolines C_p.

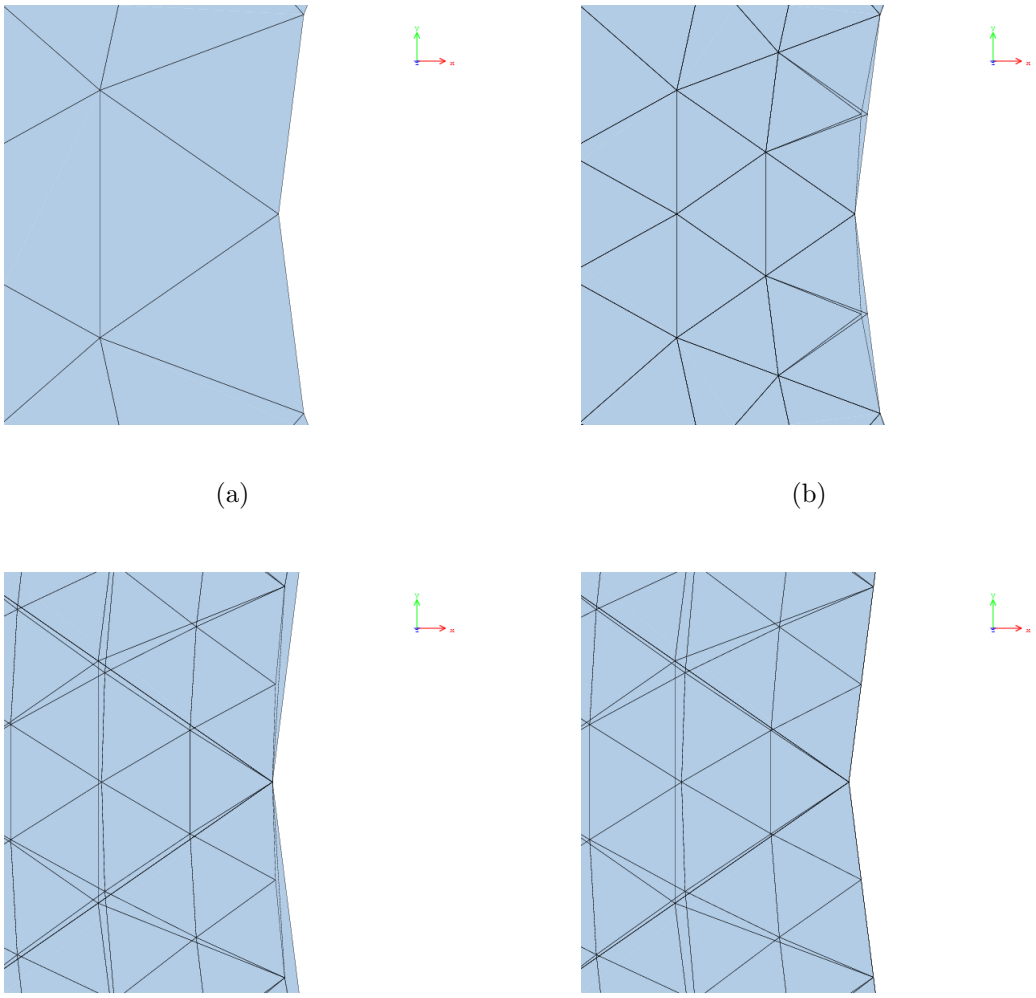


(a)



(b)

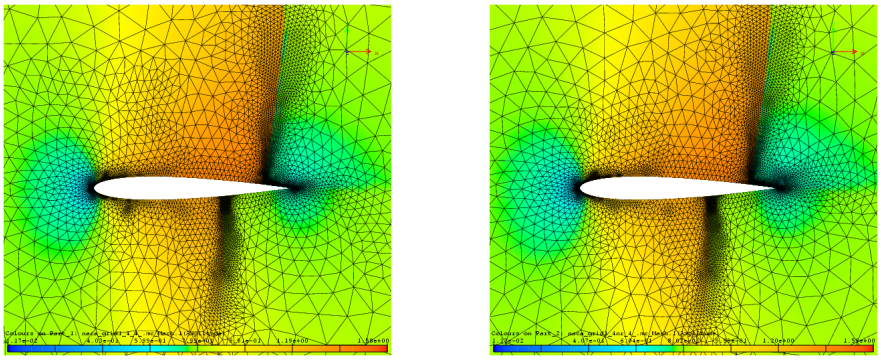
Figure 4.17: NACA 0012 adaptation process, on the left with subdivision, on the right without, tip closeup of: (a) grids and solution, (b) solution. Solution field is Mach number, isolines C_p.



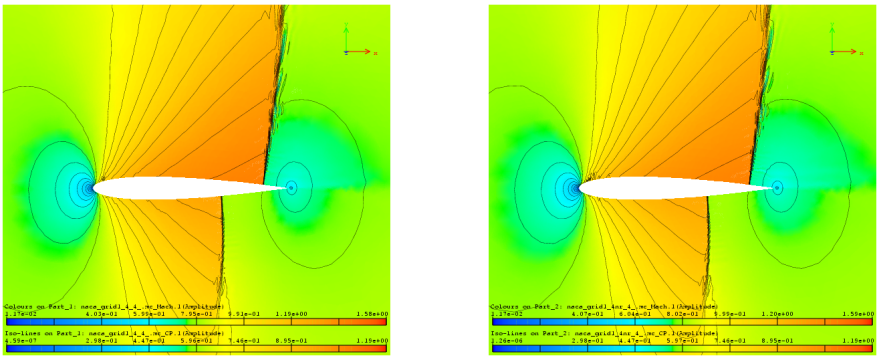
(a) (b)

(c) (d)

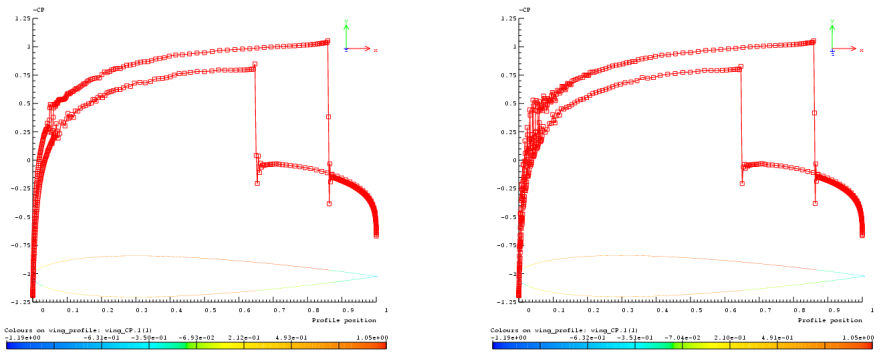
Figure 4.18: NACA 0012 tip detail with superimposed adapted grids: (a) original grid, (b) first step with and without subdivision, (c) further detail of subdivided second step, with first step and original mesh, (d) further detail of midpoint division grids step one and two.



(a)



(b)



(c)

Figure 4.19: NACA 0012 angle of attack 1°, on the left with subdivision, on the right without: (a) final grids and solution, (b) solution, (c) $-C_p$ on the wing. Solution field is Mach number, isolines C_p.

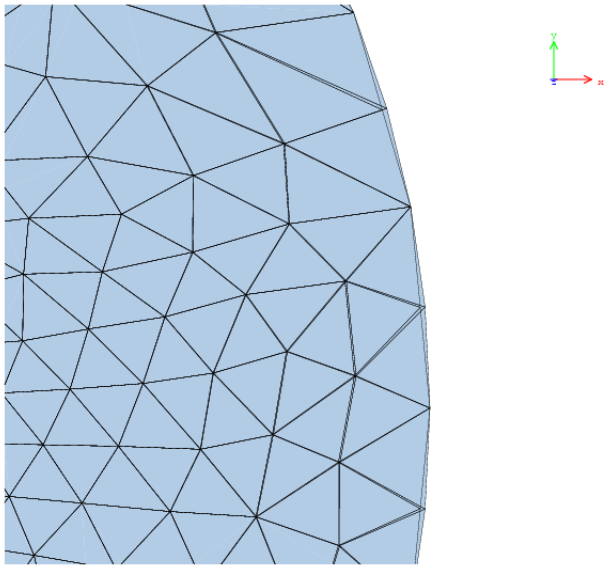


Figure 4.20: NACA 0012 angle of attack 1°, far field downstream detail with superimposed grids with and without subdivision.

4.4.2 3D: Concept aircraft

Flow conditions for this case were set to Mach number 0.85, and an angle of attack of $\alpha = 2°$. The aircraft body is approximately 1250 units long with origin at the tip of the wing section cutting the symmetry plane, and a wingspan of 490 units. The radius of the far field is approximately 25 times the body length, with centre at body mid-length. The original mesh is described with 129 865 nodes, 676 524 tetrahedral elements, and 65 770 triangular surface elements, as shown in figure 4.21.

The first two steps are computations carried out to set up the problem, with a first order solution in step one, and a second order solution in step two. The grid is then adapted with respect to Mach number, using the different subdivision schemes described above, and an a posteriori error estimator that will be described in a later chapter. The adaptation is performed on the solution obtained from the second step, which is the starting point for all the trials of the different types of subdivisions that follow. Hence the segments selected for adaption will be the same for all examples, and the resulting grids will look very much alike. An example of one of the adapted meshes is shown in figure 4.22. Note that although structural optimisation is selected, no smoothing is performed, in order to verify the methods differences and effectiveness. The details of some critical zones of the

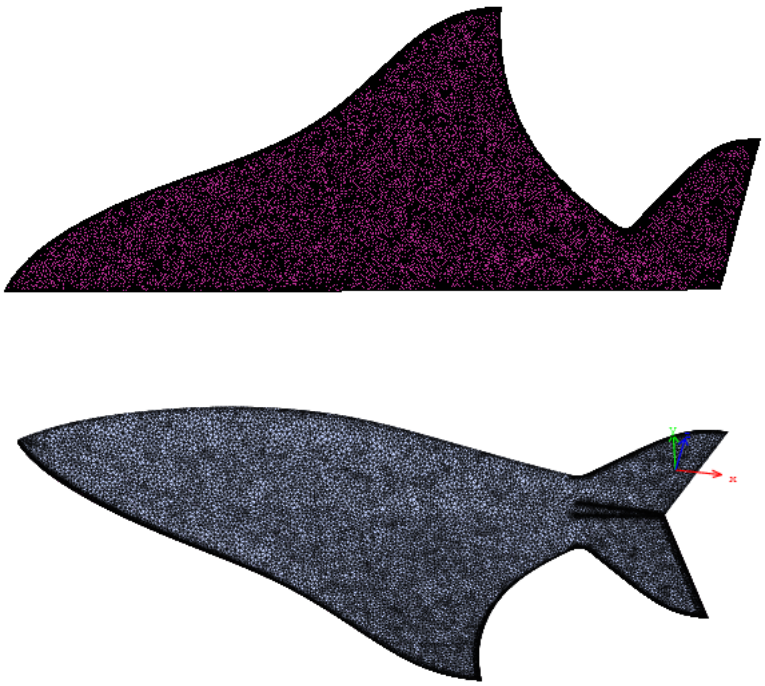


Figure 4.21: Initial grid of SmartFish, view from below and above.

grid, on the other hand, will show some differences.

First an initial detail of the tip and the tail is shown in figure 4.23, with the adapted mesh without subdivision. Then, in order to evaluate the subdivision process, figures 4.24 to 4.26 show details of the tip and tail with and without the superimposed original grid. In all subdivisions the larger segments, that tend to be further from the underlying geometry, are correctly divided by moving the node away from the midpoint and towards the geometry. The first two subdivisions in particular seem to get a smoother grid, whilst the third is not so evident. However the lower straight line feature at the tail is maintained better with the diamond difference scheme, whilst the other two methods perform similarly, with a slightly better sharpness obtained by the multi-node. This marked difference is due to ϱ and ς. Changing these to higher values would cause less difference between the results of the three schemes, but a probable loss of control of sharp features by the last scheme. This is a compromise for the user to decide which direction to take from case to case.

Results displayed in figures 4.27 and 4.28 show the initial solutions and those after adaptations with the various strategies. The disturbance created by the grid, not describing well

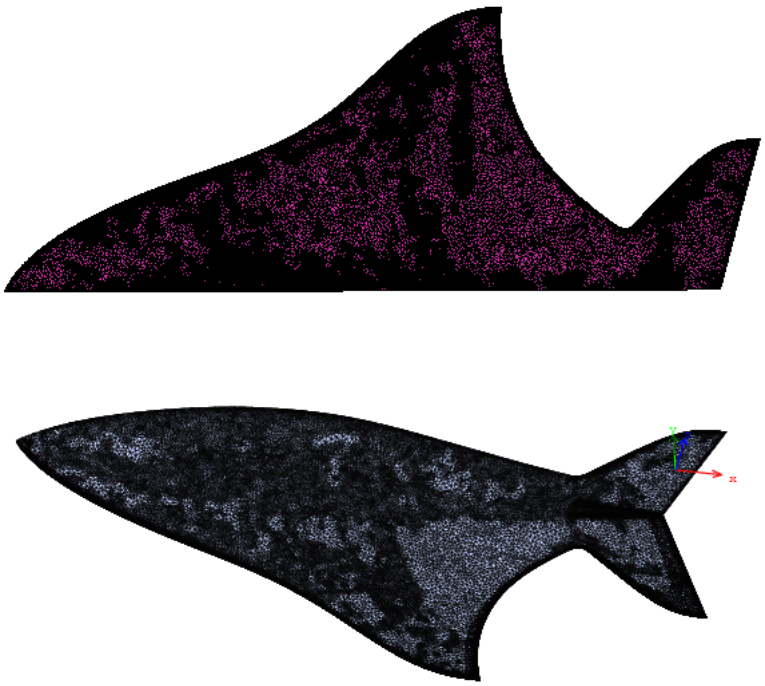
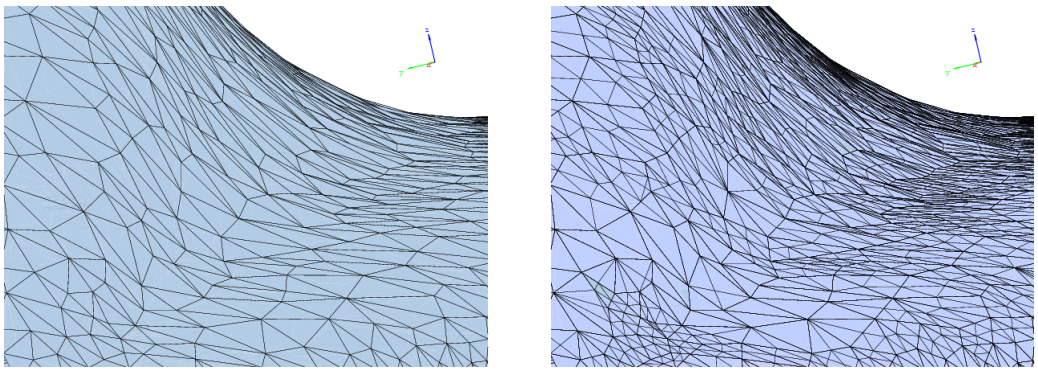
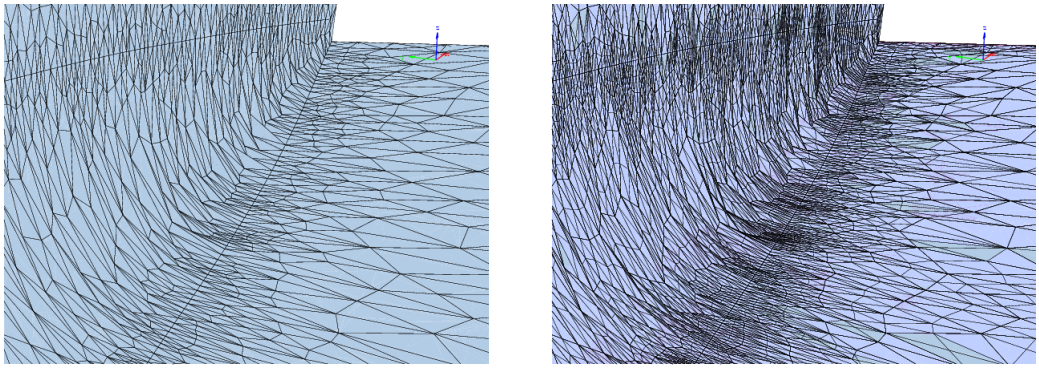


Figure 4.22: Adapted mesh, view from below and above.

enough the areas of the geometry with higher gradient, is clearly visible in the refinement without subdivision (fig. 4.27 (c)). The solutions obtained with the diamond difference, and the multi-node in particular, show a marginal improvement overall, whilst the distance based butterfly doesn't seem to ameliorate much the solution.

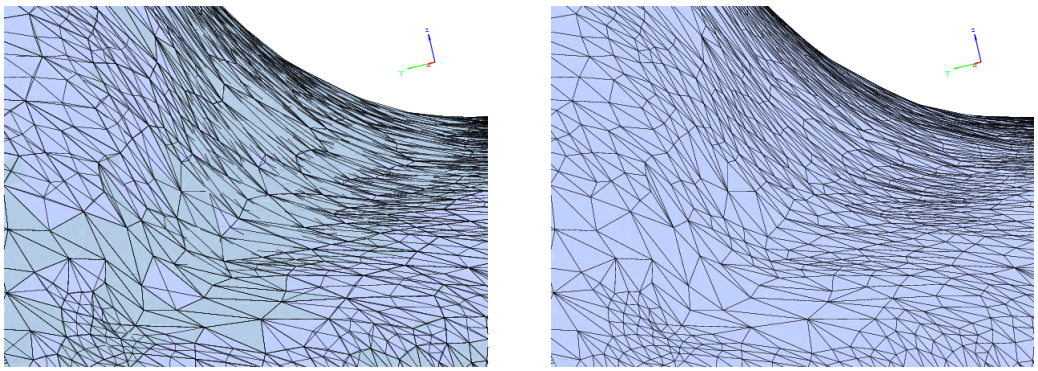


(a)

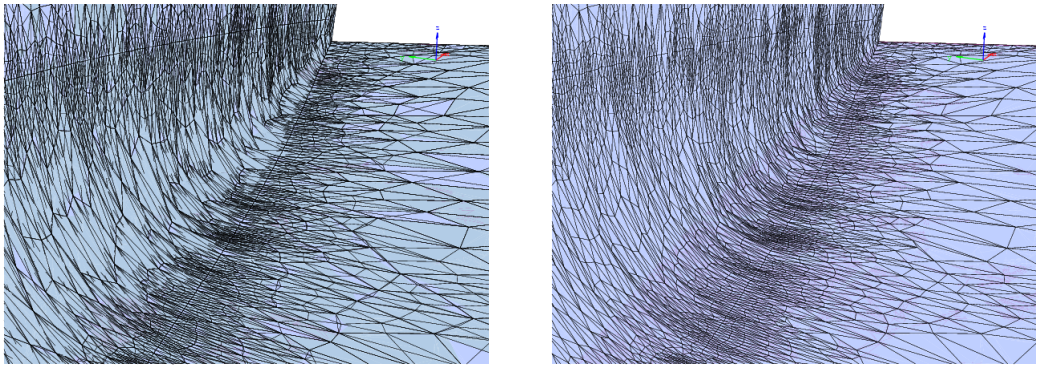


(b)

Figure 4.23: Original mesh detail of tip (a) and tail (b) on the left, and standard midpoint subdivision on the right.

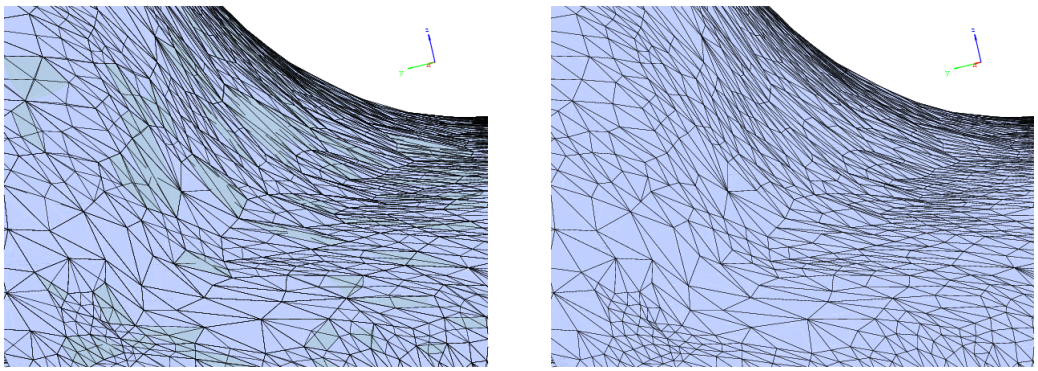


(a)

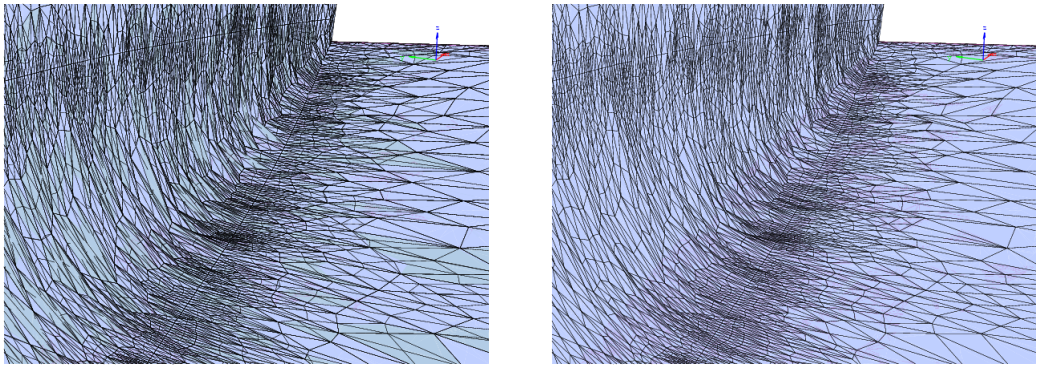


(b)

Figure 4.24: Distance based butterfly subdivision, detail of (a) tip and (b) tail, with and without original grid superimposed.

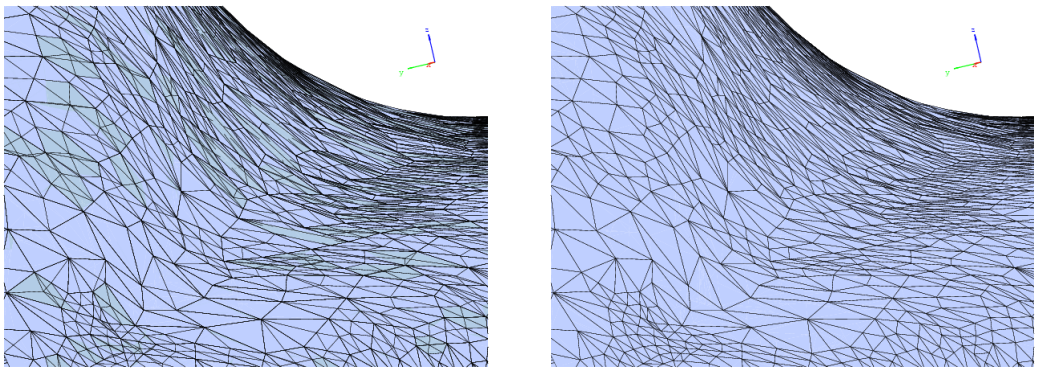


(a)

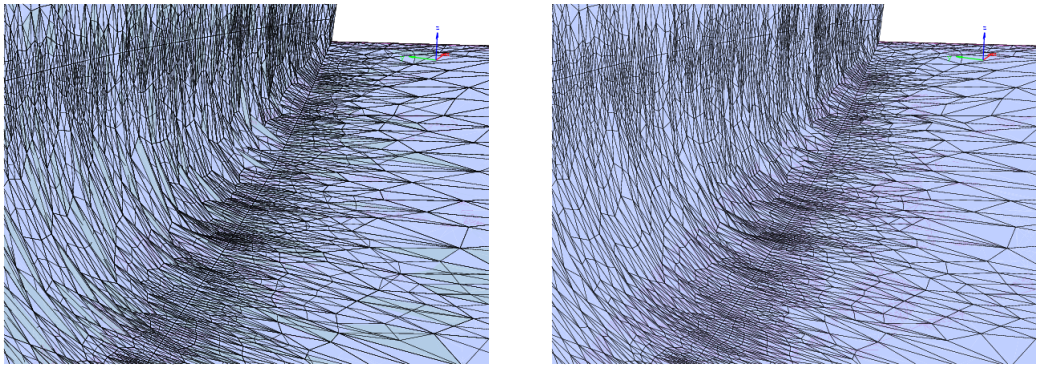


(b)

Figure 4.25: Multi-node subdivision, detail of (a) tip and (b) tail, with and without original grid superimposed.

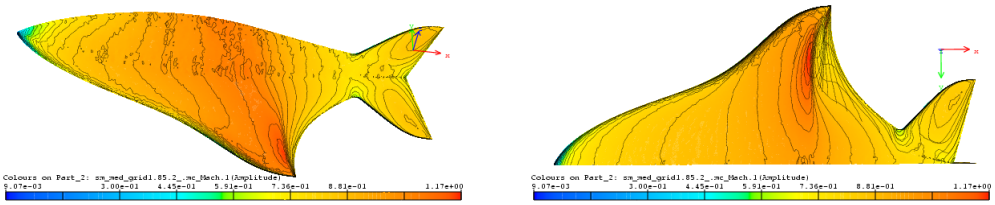


(a)

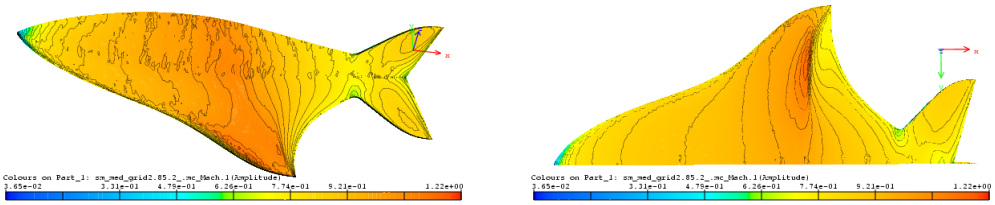


(b)

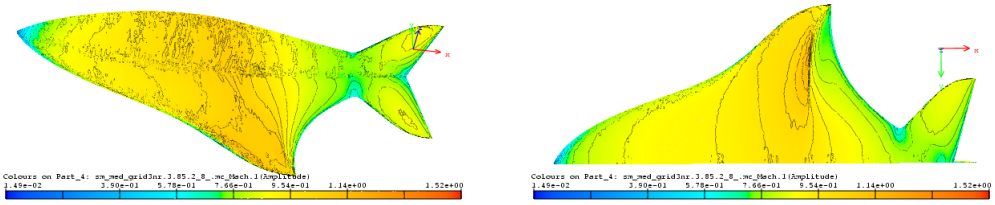
Figure 4.26: Diamond difference scheme subdivision, detail of (a) tip and (b) tail, with and without original grid superimposed.



(a)

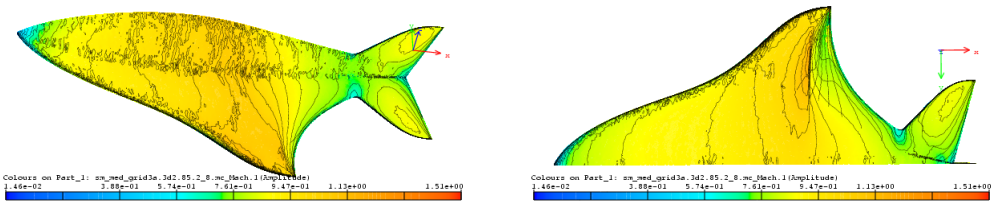


(b)

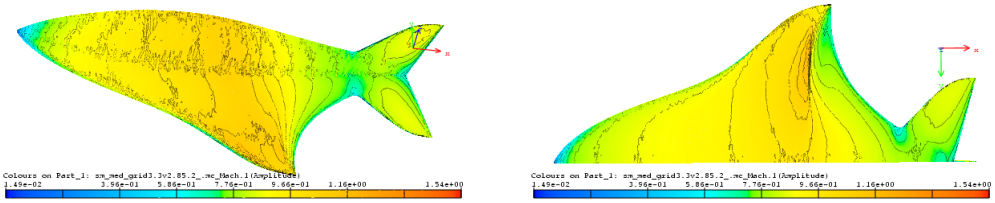


(c)

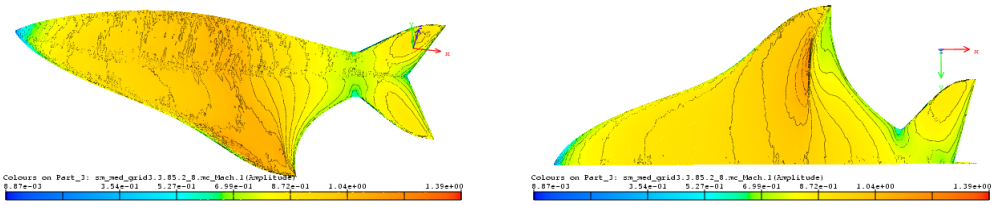
Figure 4.27: First (top) and second (middle) step soultions. At the bottom, solution for the adapted mesh without subdivision. Solution field is Mach number.



(a)



(b)



(c)

Figure 4.28: Distance based butterfly, multi-node, and diamond difference subdivisions from top to bottom. Solution field is Mach number.

5
Error Estimation

In this chapter a posteriori error estimation along the lines of the Zienkiewicz and Zhu (ZZ)-philosophy is discussed and implemented as mesh adaptation criteria. One of the main reasons for choosing these developments come from the interest of simulating problems in aeronautics with shocks and discontinuities in the solution field, and be able to capture these phenomena with element based criteria. In order to efficiently and effectively adapt a mesh to a solution field, the criteria used for the adaptation process needs to be as accurate as possible. Due to the nature of the solution, which is obtained by discretisation of a continuum model, numerical error is intrinsic in the calculation. A posteriori error estimation allows us to somehow assess the accuracy by using the computed solution itself. Various a posteriori error estimation techniques have been developed over the years, the most popular, being widely used and presented in literature, are reported thoroughly in reviews such as [19, 94, 18] and many more. Among these primes an estimator developed two decades ago by Zienkiewicz and Zhu [15, 125, 16, 17, 20, 21], hence it's most popular nickname *ZZ*. The reasons for this diffused use is its simplicity and solidity. Initially conceived for structural mechanics finite element codes, the method proved valid for many fields as it is independent of the problem and governing equations. The concept driving the method is that the exact solution $\boldsymbol{u}$ can be replaced by a recovered one $\boldsymbol{u}^*$. The approximation of the error estimation can then be measured as the difference between the recovered solution and the numerical solution:

$$\|\boldsymbol{e}^*\| = \|\boldsymbol{u}^* - \boldsymbol{u}_h\| \tag{5.1}$$

in a suitable norm.

From the above, and considering that fields such as the gradient $\boldsymbol{\sigma}_h$ are derived from the solution u_h and therefore less accurate, the improved gradient $\boldsymbol{\sigma}_h^*$ is reconstructed [16, 126, 22]. In this way the gradient is projected onto a richer space, thereby allowing for an estimation of the solution error in the energy norm. Although the method is common

amongst the research community, theoretical properties are not yet very well understood. Research has being done over the years but it still remains an unfinished topic of discussion [127, 128, 129, 130, 131, 132, 133].

Here we focus on an a posteriori recovery-based error estimator developed by Maisano et al. [23], in particular the first of three proposed, where the gradient is re-interpolated based on a choice of weighting coefficients. This can be used in the adaptation procedure to refine the mesh in those areas where the local error exceeds a set tolerance, hence further increasing the accuracy of the solution in those regions during the next calculation step (sec. 2.2). Variants of this type of error estimates are also considered.

5.1 Recovery procedures

Let us set up the problem first. Consider the governing system of partial differential equations written in a generalised form as:

$$\begin{cases} Lu = f & \text{in } \Omega, \\ Bu = g & \text{on } \partial\Omega, \end{cases} \tag{5.2}$$

where L and B are suitable differential operators, possibly nonlinear, f and g are the data of the problem, Ω the computational domain, with $\partial\Omega$ its boundary, an open bounded subset of $\mathbb{R}^d$, with $d = 1, 2, 3$. Problem (5.2) may be considered as the standard Poisson problem or the linear elasticity problem which fit directly into this framework, whilst for incompressible fluid dynamics an extension is also directly possible. For compressible fluid dynamics equations, the operator L is no longer symmetric positive definite, and the system can be written under the form:

$$\begin{cases} \mathcal{L}u = \dfrac{\partial u}{\partial t} + A(u,x,t)u + B(u,x,t) = f, \\ + \text{ Boundary Conditions}, \end{cases} \tag{5.3}$$

where $A(t)$ and $B(t)$ represent respectively the non-linear advection and diffusion operators. This generic system of non-linear PDE's applies to general compressible flow and other systems in computational flows. To simulate steady state compressible flow it is usual to pseudo-time iterate, to steady state, as equation (5.3) is of well defined type (parabolic, hyperbolic) whereas the steady state equations are not [134, 135]. In the discretised system, the underlying matrix systems are better conditioned.

To propose "a posteriori" recovery error estimators, it is useful to recall the underlying finite element (FE) type scheme used to solve the system (5.3). Assuming that our model is the system of equations for compressible fluid flow, a re-organisation of the operators allows a so-called conservative formulation for the Euler equation part which is grouped on the LHS of the following system:

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathcal{F}(u) = f + \mathcal{B}(u, \nabla u, x, t) \ . \tag{5.4}$$

Neglecting the RHS, the system becomes hyperbolic, the exact solution of which can allow discontinuities (shocks, slip lines etc..). The mathematical and discretised formulation must take these characteristics into account. In order to maintain conservativity and consistency it is usual to discretise the Euler equations using equivalently upwinded techniques. On FE grids the approximate solution u_h is represented by a liner function over each element T:

$$u_h^T(x,t) = \sum_{i=1}^{N_e} u_{h,i}(t)\phi_i(x) \ , \tag{5.5}$$

where N_e is the number of nodes per element T, and $\phi_i(x)$ the basis function associated to the node i. As is well known, [136], FE Galerkin procedures are unstable for hyperbolic problems. Upwinded FE approximations can be devised by performing biasing of the basis functions as in Streamline Upwind Petrov Galerkin (SUPG) techniques, and adding dissipation terms for controlling discontinuities, [137]. This is equivalent to performing finite volume methods (FVM) on structured grids. Alternatively, finite volume numerical schemes can be adapted to unstructured grids by considering equivalent control volumes on the FE mesh - made up by joining the barycentre of the elements and agglomerating the partial volumes around each node as shown in figure 5.1.
Then the numerical scheme is defined by flux difference schemes across the boundary between two cells V_i and V_j:

$$\nabla \cdot \mathcal{F}(u_h) = \sum_{\eta_{ij}} \mathcal{F}(u_h) \cdot \vec{n}_{ij} \ .$$

These schemes have been widely developed from 1984 to 1996; they are based on a one dimensional flux evaluation across the face η_{ij} between the neighbouring cells V_i and V_j (like V_i and V_{i+1} in structured meshes) and are hence highly mesh and directional dependent.
In order to obtain schemes that relate the multidimensionality of the wave propagation, multidimensional upwinded schemes [138] and other weighted residual schemes, such as discontinuous Galerkin, are devised.

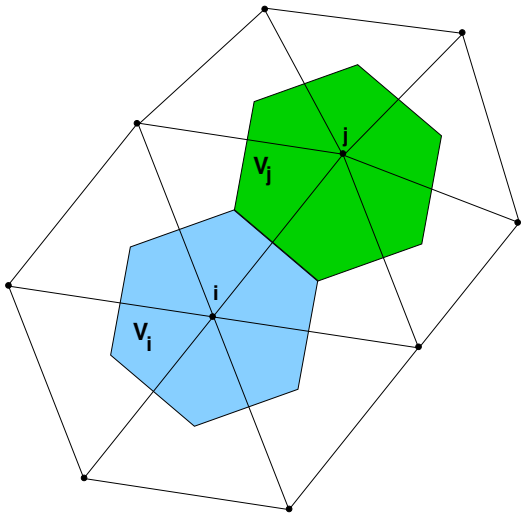


Figure 5.1: Equivalent control volumes on FE mesh.

To construct a recovery based error estimator, we must consider the numerical algorithm rendering the approximate solutions u_h. We will proceed taking a weighted residual finite element method (FEM). Considering the Euler part of equation (5.4) and discretising using a finite element space V_h, consisting of continuous piecewise linear functions, and $u_h \in V_h$ be the Galerkin approximation to the solution u in (5.2): we obtain a system like

$$\sum_{T_h} \int_T \frac{\partial u_h}{\partial t} dx = -\sum_{T_h} \int_T \nabla \cdot F(u_h) dx = -\sum_{T_h} \int_{\partial T_h} \frac{\partial F(u_h)}{\partial u} \frac{\partial u_h}{\partial x} \vec{n}_i \ , \tag{5.6}$$

using the diagonalised form of the Euler fluxes. The RHS is the residual over element T_h, and the $\vec{n}_i$ are the inward normals to the element for node i. This residual is then redistributed to the nodes using distribution coefficients such that summing up over all elements surrounding i renders the nodal residual. The a posteriori error between the solution and its approximation is related to its gradient over the elements via this residual scheme. For linear base functions, the low order gradient over an element is constant on each element T:

$$\nabla u_h|_T = \sum_i u_i \nabla \phi_i|_T = \frac{1}{|T|} \sum_{i=1}^{N_e} u_{h,i} \vec{n}_i \ .$$

Then noting $\boldsymbol{\sigma}_h$ as the piecewise constant gradient of u_h, and $\boldsymbol{\sigma}$ the corresponding exact value, the recovery based estimators work on the value $\boldsymbol{\sigma} - \boldsymbol{\sigma}_h$.
Note that the methods described in the following sections are applicable in both two and three dimensions regardless, for triangles and tetrahedra, with due modifications pointed

out where necessary.

5.1.1 Zienkiewicz-Zhu like procedure

Following the procedure devised in [139], we start by describing the first method. The approach consists in recovering the gradient $\boldsymbol{\sigma}_h^*$ at each nodal point $\boldsymbol{\zeta}_i$. This is done by using the sample points $\boldsymbol{s}_j$ of a patch of n elements enveloping the nodal point, as shown in figure 5.2.

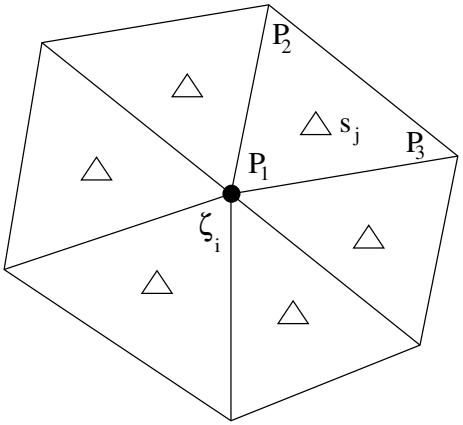


Figure 5.2: Sample and recovery points in a patch.

Hence we must first find the position of these sample points, which is taken as the average of the positions of the nodes that make up the element:

$$\boldsymbol{s}_j = \frac{1}{N_e} \sum_{i=1}^{N_e} \boldsymbol{P}_i \tag{5.7}$$

which is valid for any element type.

The computation of the constant gradient of the solution $\boldsymbol{\sigma}_h$ over the surrounding elements, is the next step. In order to do this we consider the affine n-simplex as shown in figure 5.3, with $n = 2$ for the case of triangular elements, i.e. considering only the plane xy in the figure, and $n = 3$ for the tetrahedra [140].

The element being considered can now be mapped onto the simplex by the following coordinates transformation:

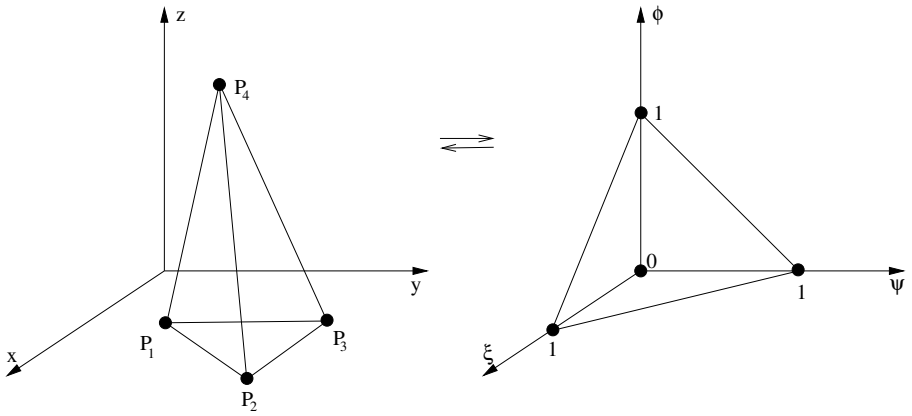


Figure 5.3: Affine map between triangle/tetrahedron and standard 2/3-simplex.

$$\begin{array}{rclcrcl} x & = & x_1 & + & (x_2 - x_1)\xi & + & (x_3 - x_1)\psi \ , \\ y & = & y_1 & + & (y_2 - y_1)\xi & + & (y_3 - y_1)\psi \ , \end{array} \tag{5.8}$$

in the two dimensional case, and:

$$\begin{array}{rclcrcl} x & = & x_1 & + & (x_2 - x_1)\xi & + & (x_3 - x_1)\psi & + & (x_4 - x_1)\phi \ , \\ y & = & y_1 & + & (y_2 - y_1)\xi & + & (y_3 - y_1)\psi & + & (y_4 - y_1)\phi \ , \\ z & = & z_1 & + & (z_2 - z_1)\xi & + & (z_3 - z_1)\psi & + & (z_4 - z_1)\phi \ , \end{array} \tag{5.9}$$

in the three dimensional case. Whilst the vertices of the simplex are noted by the points:

$$\begin{array}{rcl} e_0 & = & (1,0,0,0) \\ e_1 & = & (0,1,0,0) \\ e_2 & = & (0,0,1,0) \\ e_3 & = & (0,0,0,1) \end{array} \tag{5.10}$$

where e_3 is not considered in the two dimensional case, as is the last column of the first three points.

Given the Jacobian matrix for the triangle:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \psi} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \psi} \end{pmatrix} \ , \tag{5.11}$$

and for the tetrahedron:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \psi} & \frac{\partial x}{\partial \phi} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \psi} & \frac{\partial y}{\partial \phi} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \psi} & \frac{\partial z}{\partial \phi} \end{pmatrix} , \tag{5.12}$$

this results, in the two dimensional case to:

$$\mathbf{J} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} , \tag{5.13}$$

and in 3D to:

$$\mathbf{J} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_3 - y_1 & y_4 - y_1 \\ z_2 - z_1 & z_3 - z_1 & z_4 - z_1 \end{pmatrix} . \tag{5.14}$$

The linear weighting functions are then taken as:

$$\begin{aligned} N_1 &= 1 - \xi - \psi - \phi \\ N_2 &= \xi \\ N_3 &= \psi \\ N_4 &= \phi \end{aligned} \tag{5.15}$$

where in the 2D case N_4 and ϕ in N_1 are not considered. The gradient of u is then defined as:

$$\vec{\nabla} u = \frac{\partial u(x,y)}{\partial x}\vec{e}_x + \frac{\partial u(x,y)}{\partial y}\vec{e}_y , \tag{5.16}$$

and

$$\vec{\nabla} u = \frac{\partial u(x,y,z)}{\partial x}\vec{e}_x + \frac{\partial u(x,y,z)}{\partial y}\vec{e}_y + \frac{\partial u(x,y,z)}{\partial z}\vec{e}_z , \tag{5.17}$$

for 2D and 3D respectively. The linear approximation of u_h over the element may now be taken, for the 2D case:

$$\vec{\nabla} u_h(\xi,\psi) = \begin{pmatrix} -u_1 + u_2 \\ -u_1 + u_3 \end{pmatrix} , \tag{5.18}$$

and for the 3D:

$$\vec{\nabla} u_h(\xi, \psi, \phi) = \begin{pmatrix} -u_1 + u_2 \\ -u_1 + u_3 \\ -u_1 + u_4 \end{pmatrix} \ . \tag{5.19}$$

It suffices now to apply the inverse of the transposed Jacobian matrix computed earlier, to the equation of $\vec{\nabla} \boldsymbol{u}_h$ to obtain the constant gradient over the element.

$$\vec{\nabla} u_h(x, y) = (\mathbf{J}^T)^{-1} \cdot \vec{\nabla} u_h(\xi, \psi) \ , \tag{5.20}$$

$$\vec{\nabla} u_h(x, y, z) = (\mathbf{J}^T)^{-1} \cdot \vec{\nabla} u_h(\xi, \psi, \phi) \ , \tag{5.21}$$

where $\vec{\nabla} \boldsymbol{u}_h$ is equivalent to $\boldsymbol{\sigma}_h$. The recovered gradient is then found by means of a weighted average of the surrounding elements constant gradient. The weights are based on the distance from the sampling points to the node:

$$w_j = \frac{1}{\|\boldsymbol{\zeta}_i - \mathbf{s}_j\|} \ , \tag{5.22}$$

where $\| \cdot \|$ stands for the Euclidean norm, and

$$W = \sum_{j=1}^{n} w_j \ . \tag{5.23}$$

Note that the above weight system is not the only available. Other widely used methods rely on the area A_j (or volume V_j) of the surrounding elements, or more simply the number of surrounding elements:

$$w_j = |A_j| \ , \ w_j = |V_j| \quad \text{or} \quad w_j = \frac{1}{n} \ .$$

From the above equations we now have all that is needed to recover the gradient:

$$\boldsymbol{\sigma}_h^*(\boldsymbol{\zeta}_i) = \frac{1}{W} \sum_{j=1}^{n} \boldsymbol{\sigma}_h(\mathbf{s}_j) w_j \ , \tag{5.24}$$

which is the same for both two and three dimensions.
Once all this procedure is carried out for every node in the mesh, the following error estimator can be devised:

$$\eta_1 = \|\boldsymbol{\sigma}_h^* - \boldsymbol{\sigma}_h\|_{L_2(\Omega)} \simeq |||u - u_h||| \ , \tag{5.25}$$

with $|||\cdot|||$ standing for the suitable energy norm.

Since we are interested in using the error estimator for adapting the mesh, we need to write this in a local fashion. Let $\mathcal{T}_h$ be the finite representation of our domain and $\{K_j\}$ the elements discretising it. The error estimator can then be expressed as a sum of the local elements error indicators:

$$\eta_1 = \Big(\sum_{K\in\mathcal{T}} \eta_{1,K}^2\Big)^{(1/2)} \ , \tag{5.26}$$

where the local error indicator represents the difference between the piecewise linear recovered gradient field, and the piecewise constant gradient field of the element

$$\eta_{1,K} = \|\boldsymbol{\sigma}_{h,i,K}^* - \boldsymbol{\sigma}_{h,K}\|_{L_2,K} \ , \tag{5.27}$$

which can be rewritten as:

$$\eta_{1,K} = \left[\int_K (\boldsymbol{\sigma}_{h,K}^* - \boldsymbol{\sigma}_{h,K})^2 \, dV\right]^{1/2} \ , \tag{5.28}$$

$$\eta_{1,K} = \left[(\boldsymbol{\sigma}_{h,K}^* - \boldsymbol{\sigma}_{h,K})^T \mathbf{M}_K (\boldsymbol{\sigma}_{h,K}^* - \boldsymbol{\sigma}_{h,K})\right]^{1/2} \ , \tag{5.29}$$

where $\mathbf{M}_K$ is the mass matrix of the element K. This is evaluated from:

$$M_K^{lm} = \int_K \varphi_l \varphi_m \ , \tag{5.30}$$

and for the 2D and 3D respectively

$$\int_{A_K} \varphi_l \varphi_m \, dA = \begin{cases} \frac{1}{6} A_K & \text{if } l = m, \\ \frac{1}{12} A_K & \text{if } l \neq m. \end{cases} \tag{5.31}$$

$$\int_{V_K} \varphi_l \varphi_m \, dV = \begin{cases} \frac{1}{10} V_K & \text{if } l = m, \\ \frac{1}{20} V_K & \text{if } l \neq m. \end{cases} \tag{5.32}$$

which gives us the explicit form of the mass matrices for the linear triangle and tetrahedron:

$$M_K = \frac{1}{12} A_K \left[\begin{array}{cc|cc|cc} 2 & 0 & 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 & 0 & 1 \\ \hline 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 2 \end{array}\right] , \tag{5.33}$$

$$M_K = \frac{1}{20} V_K \left[\begin{array}{ccc|ccc|ccc|ccc} 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 \end{array}\right] . \tag{5.34}$$

Now that we have the mass matrix we pass on to the difference between the reconstructed gradient and the constant one:

$$(\boldsymbol{\sigma}^*_{h,K} - \boldsymbol{\sigma}_{h,K})^T = \left[\hat{\boldsymbol{\sigma}}_{h,P_1}, \quad \hat{\boldsymbol{\sigma}}_{h,P_2}, \quad \hat{\boldsymbol{\sigma}}_{h,P_3}, \quad \hat{\boldsymbol{\sigma}}_{h,P_4}\right] , \tag{5.35}$$

for the tetrahedron, and without $\hat{\boldsymbol{\sigma}}_{h,P_4}$ for the triangle keeping in mind that the vectors change also in size.
The single element of this vector is described as:

$$\hat{\boldsymbol{\sigma}}_{h,P_i} = \boldsymbol{\sigma}^*_{h,P_1,K} - \boldsymbol{\sigma}_{h,K} . \tag{5.36}$$

Boundary nodes

For nodes on the boundary a slightly different approach is required when reconstructing the gradient. Rather than using the original recovery procedure suggested in [125], we use the first alternative suggested in [139]. The starting point here is at the end of the

previous step, with all gradients having been reconstructed on internal nodes $\boldsymbol{\zeta}_j^k$. Then our boundary nodes become $\boldsymbol{\zeta}_i^B$, as shown in figure 5.4.

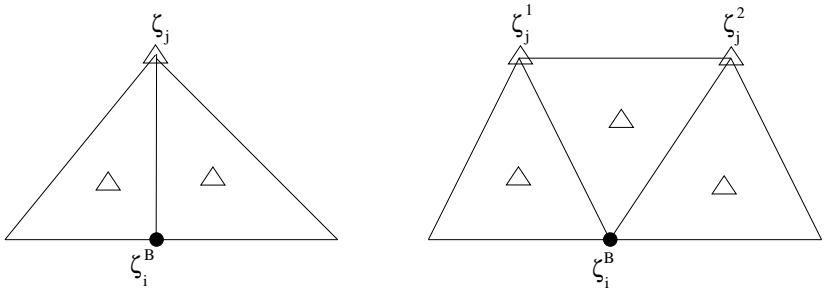


Figure 5.4: Recovery procedure for the boundary node $\boldsymbol{\zeta}_i^B$: case of one internal adjacent node (on the left), and of multiple adjacent internal nodes (on the right).

Once all the neighbouring internal nodes have been identified, they are added to the set of sampling points. The gradient on the boundary nodes can now be reconstructed using the constant gradients of the elements they belong to and the recovered gradients at the internal nodes that they are connected to. Equation (5.24) is then applied to the new set of sampling points for each boundary node, and the gradient is recovered.

Shock Capturing

This is a particularly interesting feature that well adapts to the problems studied here. The idea behind this is to make use of the freshly recovered gradients and the element constant gradients. The conditions to capture the shock are stated as:

$$\begin{cases} \|\boldsymbol{\sigma}_h^*(\boldsymbol{\zeta}_j)\| < \|\boldsymbol{\sigma}_{h,K}\| \ , & j = i, N_e \ , \\ \\ \dfrac{|\,\|\bar{\boldsymbol{\sigma}}_{h,j}^*\| - \|\boldsymbol{\sigma}_{h,K}\|\,|}{\|\boldsymbol{\sigma}_{h,K}\|} \geq \Phi \ , \end{cases} \tag{5.37}$$

where the absolute values and Euclidian norm are intended, and

$$\bar{\boldsymbol{\sigma}}_{h,i}^* = \frac{1}{N_e} \sum_{i=1}^{N_e} \boldsymbol{\sigma}_h^* \tag{5.38}$$

whilst Φ is the tolerance value of the percentage in difference between the average recovered gradient over the element, and the constant gradient. The concept behind these inequalities is that, for a shock to be present in an element, both the following should be true:

1. All the element nodal values of the recovered gradient are less than the element constant gradient;

2. The jump between the averaged recovered gradient over the element and the constant gradient of the same element has to be at least 25% of the latter.

If these conditions, also known as shock capturing conditions (SCC), are both satisfied, then the element could possibly have a shock inside it. If this is not the case then it should not undergo further refinement in a following adaptation step. For further details on the mathematical background of the above conditions the reader is referred to the work by Maisano et al.[23].

5.1.2 Face gradient error estimator

Here the procedure above is followed up to the definition of the error estimator η_1 (5.25). The piecewise constant gradient and the linear reconstructed one are used in a different manner. To construct the error estimator we make use of the element constant gradient only, and we shall see in a later section how the reconstructed gradient is used to devise an error indicator from the estimator. Let us consider equation (2.7) in section 2.2.1, and the discrete second derivative there defined as D_h^2. This can be estimated as[111]:

$$D_h^2 u_h|_{T_k} = \max_{T_{k'} \in V(T_k)} \frac{|\vec{\nabla} u_h(G_{k'}) - \vec{\nabla} u_h(G_k)|}{G_{k'} - G_k} \ , \tag{5.39}$$

where $V(T_k)$ is the set of elements sharing a common interface (edge or face) with element T_k, and G_k is the centroid of element T_k. The discrete second derivative D_h^2 can also be written as:

$$D_h^2 u_h|_{T_k} = \max_{edges \in T_k} h_{edge} \left| \left[\frac{\partial u_n}{\partial n_{edge}} \right] \right| \ , \tag{5.40}$$

where $[\cdot]$ is the jump of the gradient across the element interface, which will either be an edge in 2D, or a face in 3D. The above correspond to discrete H^2 norms, therefore (2.7) can be used.

Then the error estimator rather than being computed on the element, it is done at face level. Similarly to the above equation, this is achieved by using just the constant piecewise gradients of the elements, giving:

$$\eta_{2,i} = \| \boldsymbol{\sigma}_{h,K_l} - \boldsymbol{\sigma}_{h,K_m} \| \ , \tag{5.41}$$

where i is the face in common between elements K_l and K_m, and the norm is Euclidean. The boundary procedure and that for capturing the shock remains the same as for the previous method.

5.1.3 Simple error estimator

In this method we proceed following the basic idea behind the first error estimator, but rather than reconstruct the gradient on the node, we reconstruct a solution field to obtain a reconstructed solution at the node. If we consider figure 5.2 again, and u_h at the P_i nodes, we can take the average value on the element of u_h as:

$$\bar{u}_{h,j} = \frac{1}{N_e} \sum_{i=1}^{N_e} P_i \ . \tag{5.42}$$

We can now find the reconstructed u_h^* at node $\boldsymbol{\zeta}$:

$$u_h^*(\boldsymbol{\zeta}_i) = \frac{1}{W} \sum_{j=1}^{n} \bar{u}_{h,j} w_j \ . \tag{5.43}$$

The simple local error estimator can then be derived:

$$\eta_{3,i} = \| u_h^* - u_h \| \ , \tag{5.44}$$

with the norm being Euclidean.
This indicator was developed as a simple improvement to the physical criteria described in section 2.2.2. Boundary conditions are treated in the same manner as the above methods, replacing the reconstructed gradient at $\boldsymbol{\zeta}_j^k$ with u_j^*.
The shock capturing procedure on the other hand, is modified in the following way:

$$\begin{cases} \| u_h^*(\boldsymbol{\zeta}_j) \| < \| u_{h,K} \| \ , & j = i, N_e \ , \\ \\ \dfrac{|\, \| \bar{u}_{h,j}^* \| - \| u_{h,K} \| \,|}{\| u_{h,K} \|} \geq \Phi \ , \end{cases} \tag{5.45}$$

and

$$\bar{u}_{h,i}^* = \frac{1}{N_e} \sum_{i=1}^{N_e} u_{h,i}^* \ . \tag{5.46}$$

5.2 Implementation

ZZ like estimator

As we have seen from the previous section we can find the error indicator $\eta_{i,K}$ for both triangular and tetrahedral grids. This however is not particularly useful for adapting the mesh locally. In order to obtain a useful measure for adaptation, the local percentage error needs to be defined. This is given by the following relationship:

$$\eta^{\%}_{1,K} = \frac{\eta_{1,K}}{\frac{1}{2}(\|u_h\|_{L_2,K} + u_{m,K})} \ , \tag{5.47}$$

where

$$u_{m,K} = \frac{1}{N_K}\sum_{j=1}^{N_K} \|u_h\|_{L_2,j} \ , \tag{5.48}$$

is the mean value of $\|u_h\|_{L_2,K}$ in the N_K neighbouring cells of element K. The procedure to compute $\|u_h\|_{L_2,K}$ is the same followed in the previous section to find the local error indicator $\eta_{1,K}$, whereby:

$$\|u_h\|_{L_2,K} = \left[(\hat{\boldsymbol{u}}_{h,P_i})^T \mathbf{M}_K (\hat{\boldsymbol{u}}_{h,P_i})\right]^{1/2} \ , \tag{5.49}$$

where $\hat{\boldsymbol{u}}_{h,P_i}$ is the vector of our solution at the nodes of the element. We make again use of our element mass matrices for the triangular and tetrahedral elements, which are now smaller since u_h in our case is normally a scalar:

$$M_K = \frac{1}{12} A_K \left[\begin{array}{c|c|c} 2 & 1 & 1 \\ \hline 1 & 2 & 1 \\ \hline 1 & 1 & 2 \end{array}\right] \ , \tag{5.50}$$

$$M_K = \frac{1}{20} V_K \left[\begin{array}{c|c|c|c} 2 & 1 & 1 & 1 \\ \hline 1 & 2 & 1 & 1 \\ \hline 1 & 1 & 2 & 1 \\ \hline 1 & 1 & 1 & 2 \end{array}\right] \ . \tag{5.51}$$

Finally for a given test-case, let ϵ_1 be the tolerance associated with the error indicator $\eta_{1,K}$. Then the adaptive procedure consists of refining the elements satisfying

$$\eta^{\%}_1 > \epsilon_1 \ . \tag{5.52}$$

Since however the refinement is carried out at a segment level rather than element-wise, the way we proceed is to assign to the nodes the highest $\eta^{\%}_{1,K}$ of the elements it belongs to. Then, if both $\eta^{\%}_{1,K}$ at the nodes of the segment satisfy (5.52), the segment is marked for refinement.

Notice that the error indicator may also be modified by the shock capturing procedure if the SCC are satisfied, by increasing it's $\eta^{\%}_{1,K}$ value above ϵ_1.

Face gradient error indicator

The local percentage error indicator developed here makes use of the recovered gradients at the nodes to give a measure of the $\eta_{2,i}$. A first step is to find the nodes of the face being evaluated, and averaging the recovered gradient such that:

$$\bar{\boldsymbol{\sigma}}^*_{h,i} = \frac{1}{d}\sum_{j=1}^{d} \boldsymbol{\sigma}^*_{h,j} \ , \tag{5.53}$$

where $\bar{\boldsymbol{\sigma}}^*_{h,i}$ is the average over face i, and d is the dimension of the case, since in 2D the face of a triangle is composed by the edge (which has two nodes), and in 3D by the triangular face of the tetrahedron with three nodes defining it. The percentage error indicator takes the form:

$$\eta^{\%}_{2,i} = \frac{\eta_{2,i}}{\|\bar{\boldsymbol{\sigma}}^*_{h,i}\|} \ . \tag{5.54}$$

As in the previous case, the highest value of $\eta^{\%}_{2,i}$ of the faces to which a node belongs to, is assigned to it. The procedure for finding which edges to refine is the same as that described just above.

Simple error indicator

Following on from 5.1.3, and in a similar manner to the previous paragraphs, from the error indicator $\eta_{3,i}$ we can derive the local percentage error indicator $\eta^{\%}_{3,i}$, which is simply:

$$\eta^{\%}_{3,i} = \frac{\eta_{3,i}}{|u_{h,j}|} \tag{5.55}$$

where $|\cdot|$ is the absolute value.

5.3 Applications

The error estimators are then tested with the some of the geometries seen in the previous chapters. The flow solver used to obtain the solutions is, as previously, THOR. In particular we examine the test cases of the ONERA M6 wing and the wedge.
A two dimensional example of the simple error estimator is also available in section 2.6.1. There we can notice how the difference method tends to refine the mesh in the areas from the tip to the shocks. The simple error estimation concentrates on the tip and shocks, although the shock capturing procedure had not been introduced yet at the time, as well as around the profile. The effect of these differences can be clearly seen in the solution fields shown in fig. 2.20, where the shocks are much more defined in the error estimation case.

5.3.1 3D Wedge

The initial grid used for this test case is shown in figure 2.21 in section 2.6.2. The solution here is obtained for free-stream Mach number 2.0, and is adapted with the simple error estimator with respect to density. The number of nodes and elements was kept similar between this computation and that of section 2.6.2, in order to be able to compare the two methods. The adaptation steps were also the same, and are resumed in figures 5.5 to 5.5:

Figure 5.5 (a) Refined only mesh 7 176 NE, 1 390 NN, 1 324 BF and (b)&(c) solution;

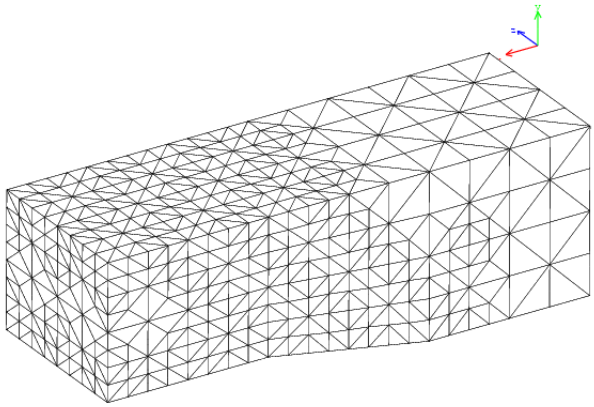
Figure 5.6 (a) Refined only mesh 42 963 NE, 7 558 NN, 4 382 BF and (b)&(c) the solution obtained;

Figure 5.7 (a) Mesh after 1 step refinement followed by 1 step derefinement 148 688 NE, 24 335 NN, 9 270 BF and (b)&(c) the solution obtained;

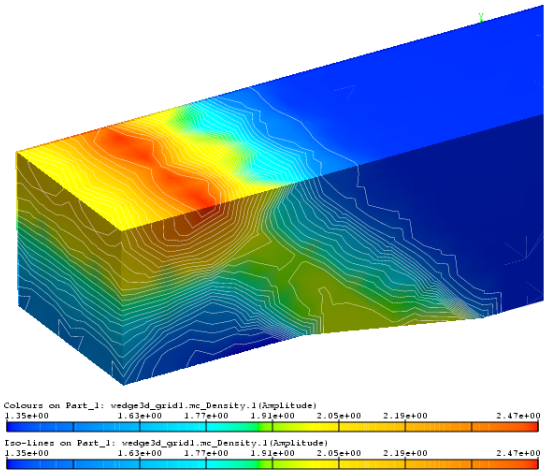
Figure 5.8 (a) Mesh after 1 step refinement followed by 1 step derefinement 404 002 NE, 64 266 NN, 18 636 BF and (b)&(c) the solution obtained;

Figure 5.9 (a) Mesh after 1 step refinement followed by 1 step derefinement 1 025 857 NE, 159 675 NN, 34 986 BF and (b)&(c) the solution obtained.

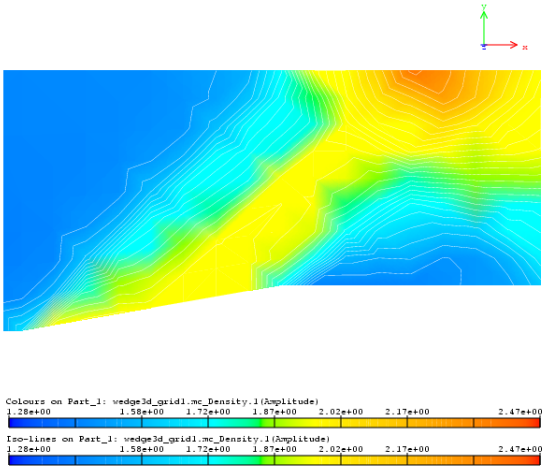
Although the first two steps are very much the same if compared to the computation of section 2.6.2, the first differences can be noticed starting from the third step. The boundary faces in particular are increasing using the simple error estimation. This is clear to see by the last step of the test, where the boundary faces are double in the results obtained with the simple error estimation.



(a)

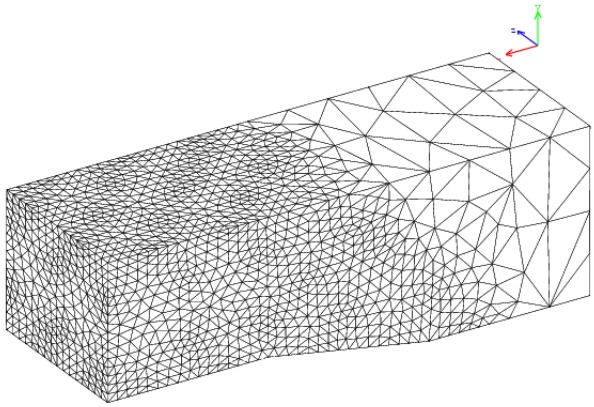


(b)

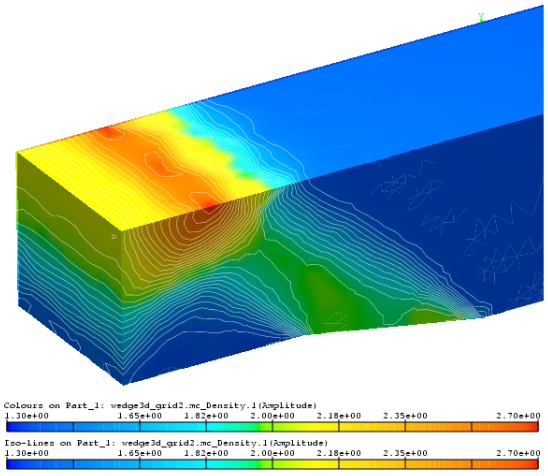


(c)

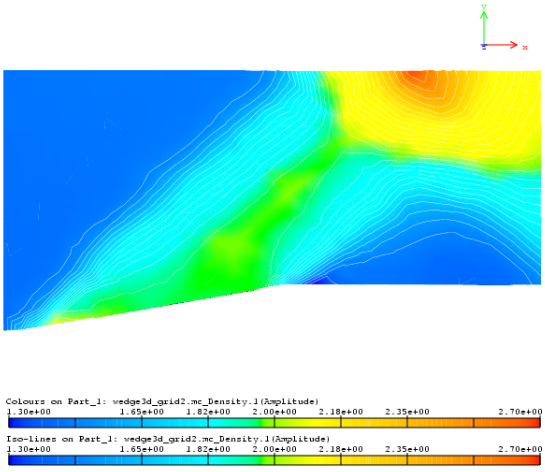
Figure 5.5: 3D wedge mesh and relative solution. 1 step adaptation (no smoothing).



(a)

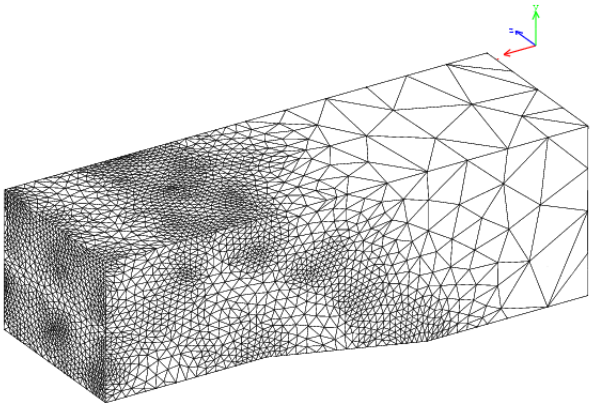


(b)

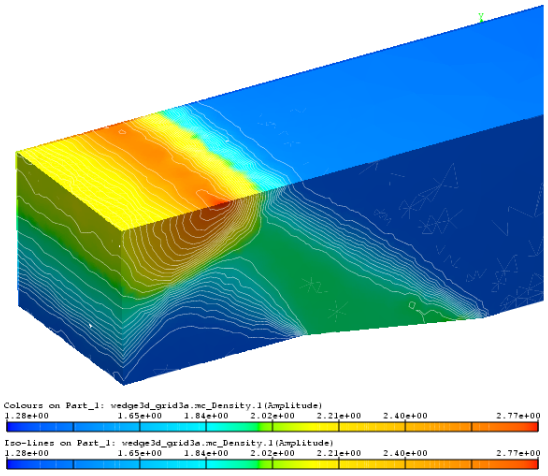


(c)

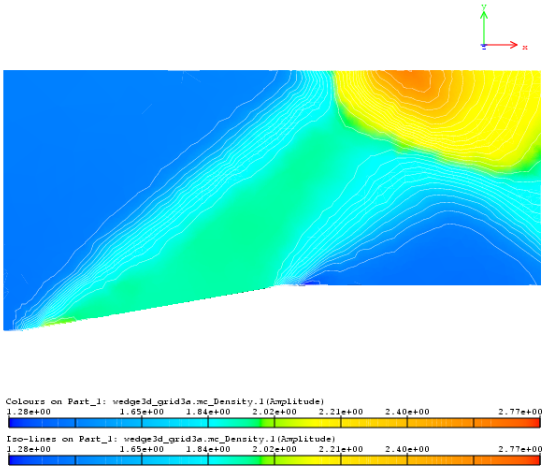
Figure 5.6: 3D wedge mesh and solution. 1 step adaptation (following from figure 5.5).



(a)

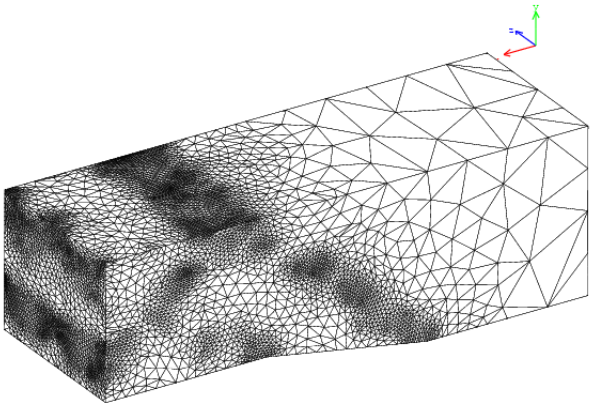


(b)

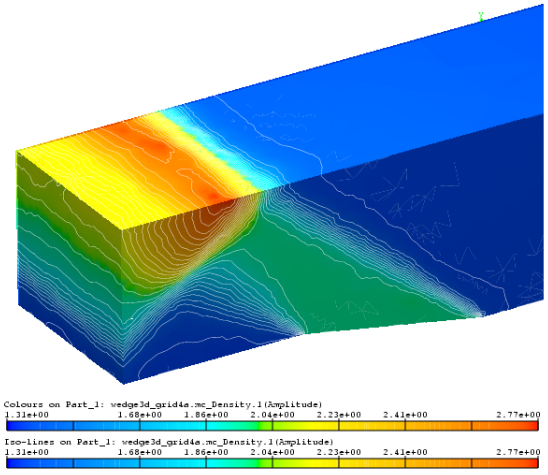


(c)

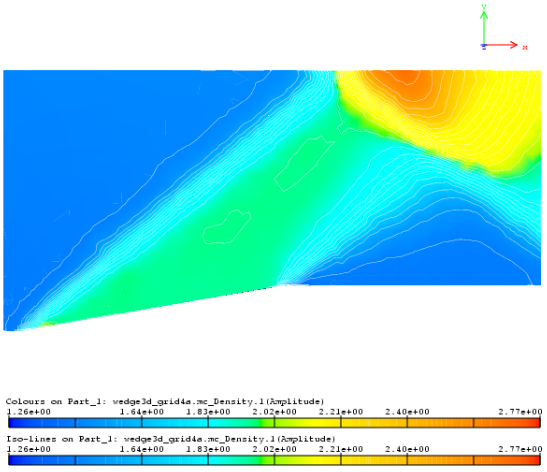
Figure 5.7: 3D wedge mesh and solution. 2 step adaptation (following from figure 5.6).



(a)

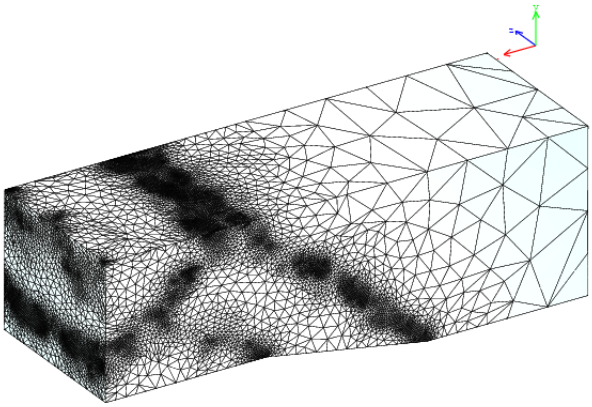


(b)

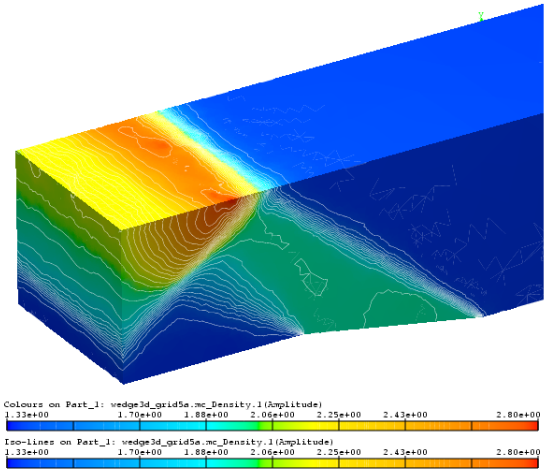


(c)

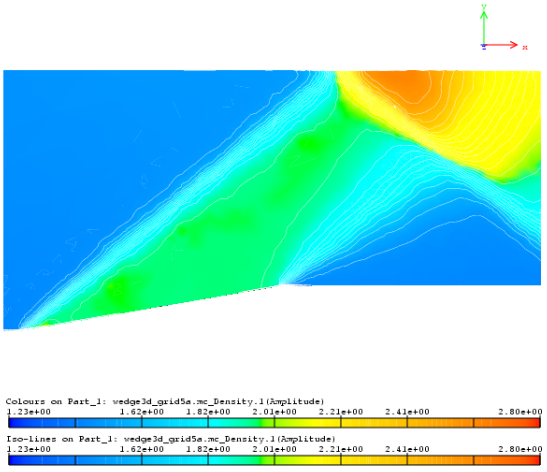
Figure 5.8: 3D wedge mesh and solution. 2 step adaptation (following from figure 5.7).



(a)



(b)



(c)

Figure 5.9: 3D wedge mesh and solution. 2 step adaptation (following from figure 5.8).

5.3.2 ONERA M6

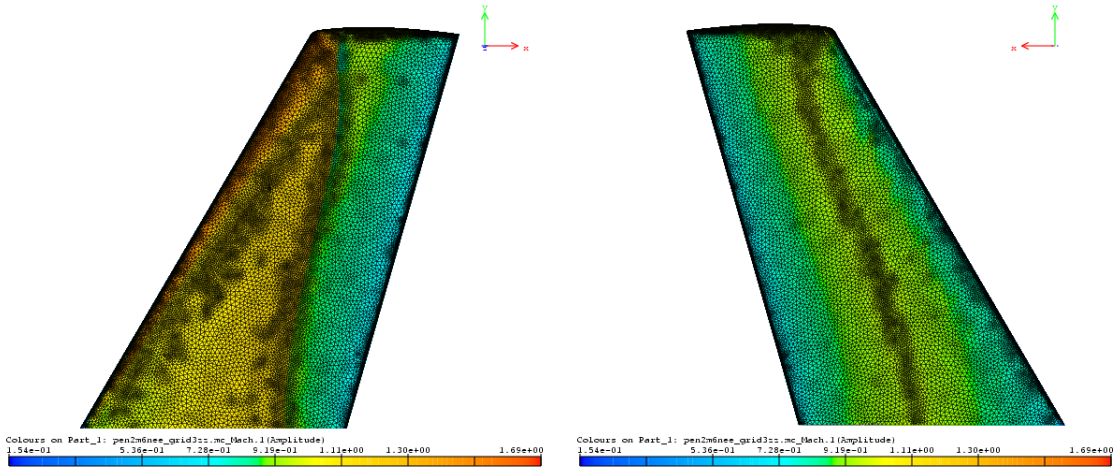
The results obtained with this geometry are generally satisfactory in the case of the first two error estimators, whilst less optimal for the simple error estimator. However the solutions are relatively good in all three cases, given the initial grid coarseness. This in fact could pose a problem in the initial adaptation steps, due to small perturbations caused by the mesh size, whereby the SCC might be satisfied in areas where no shock is present. The initial mesh and conditions are the same described in section 3.5.3, with transonic flow over the wing. Also in this case, the solution was restarted after two initial steps to set up the problem correctly.

In order to do a reliable comparison, all adaptation parameters are kept the same for the computations with the different error estimators. In particular the mesh was adapted with respect to Mach number. As we can see from table 5.1, the mesh sizes are relatively similar, with a slightly less populated mesh in the case of the simple error estimator.

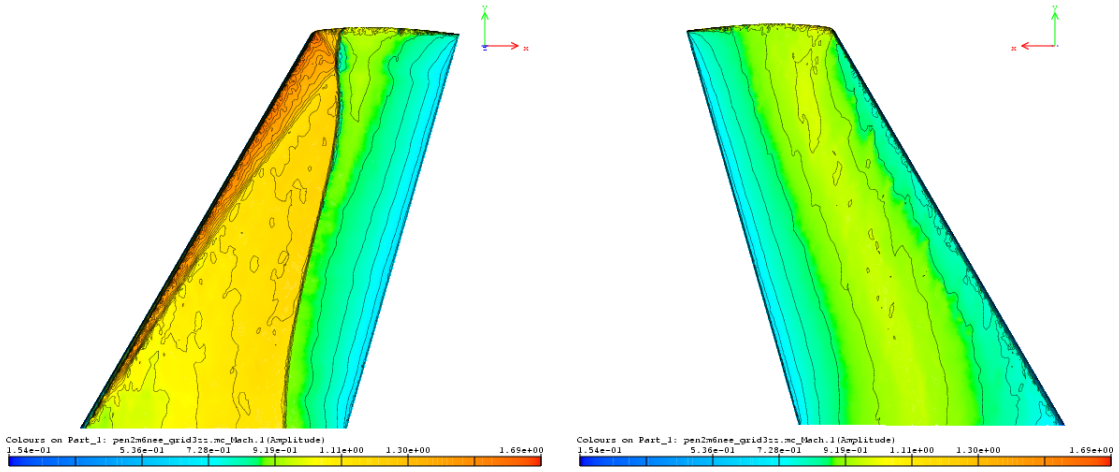
Characteristic number	**Error estimator**		
	ZZ-like	*Face-grad*	*Simple*
N_nodes	266 714	264 013	226 049
N_elements	1 495 724	1 480 961	1 279 895
N_surf_el	82 488	81 930	63 630

Table 5.1: Grid sizes for ONERA M6 computations with the relative error estimator.

The results are presented in figures 5.10 to 5.12. The first illustrates the grid obtained with the ZZ-like error estimator, and its relative solution. Here we can see how the typical λ shock wave is well captured, and how the mesh is correctly refined in that area. It is also very interesting to notice how the much more feeble shock on the lower side is evidently captured by the technique. The similarities with the Face-grad estimator are due to the shock capturing procedure that is the same as for the ZZ-like estimator. The Simple estimator on the other hand did not capture the first part of the shock wave, and refined the mesh in areas of little or no interest. Although the performance seemed to be good for other test cases, more work is needed on its formulation to improve it.

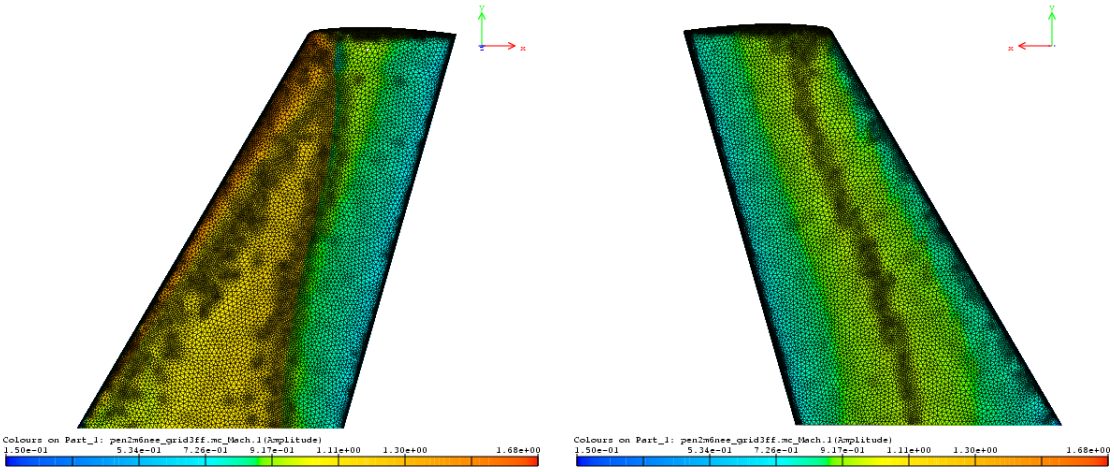


(a)

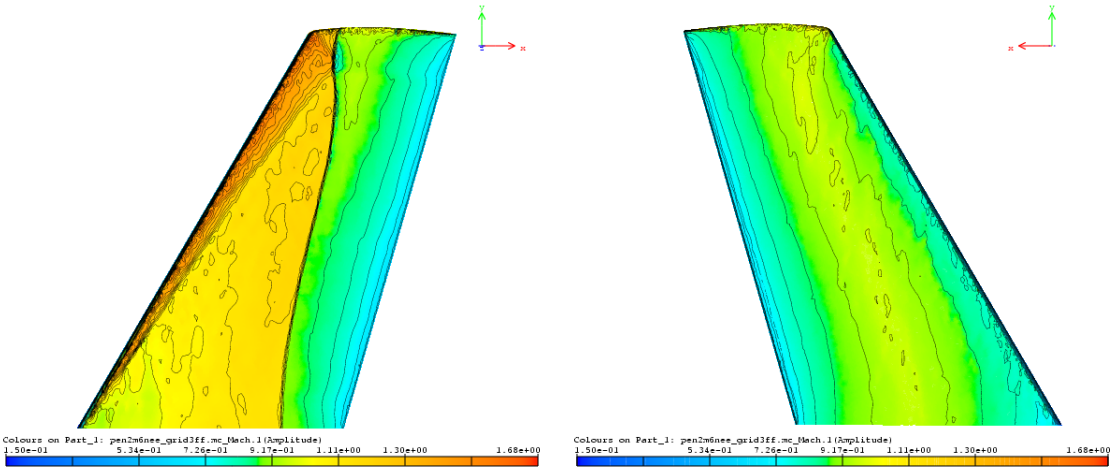


(b)

Figure 5.10: Grid and solution on the ONERA M6 wing, with the ZZ like error estimator.

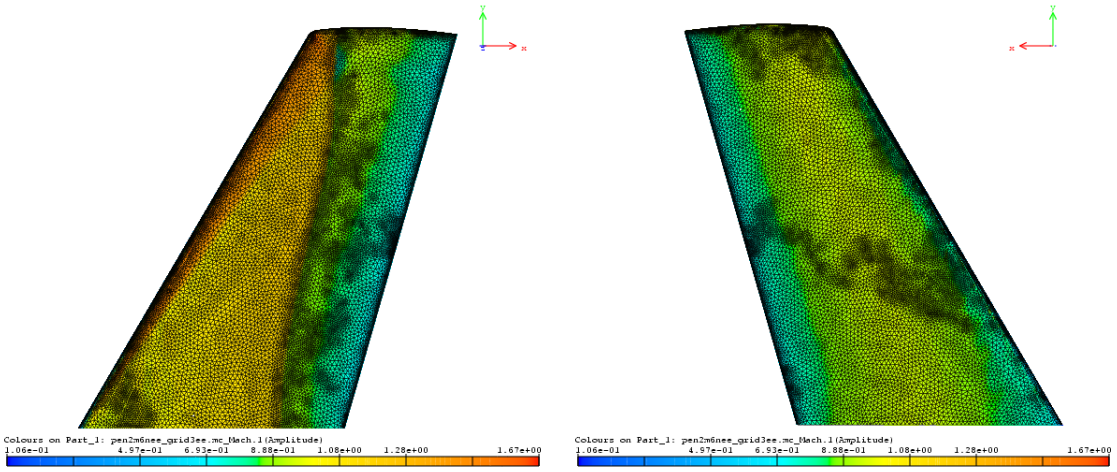


(a)

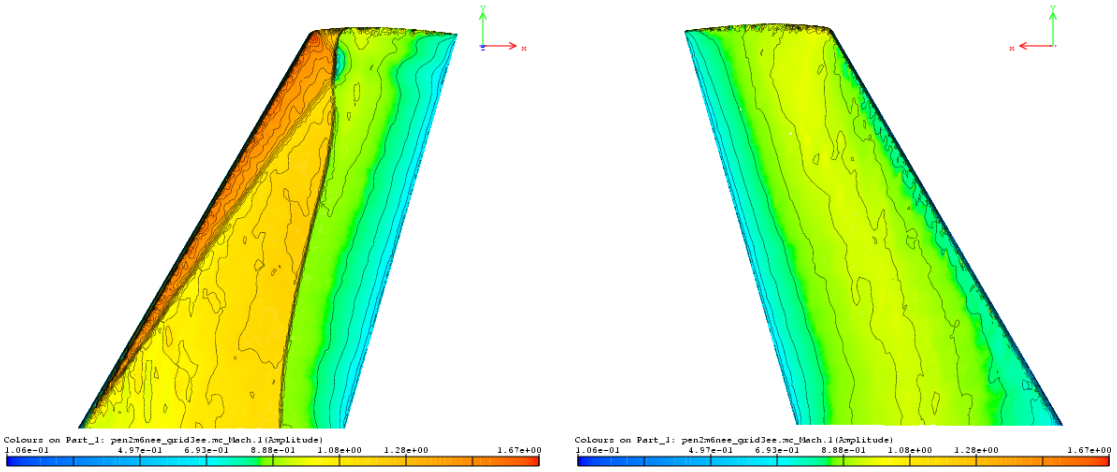


(b)

Figure 5.11: Grid and solution on the ONERA M6 wing, with the face gradient error estimator.



(a)



(b)

Figure 5.12: Grid and solution on the ONERA M6 wing, with the simple error estimator.

6 Parallelisation

The aim of this chapter is to provide an overview of the parallelisation issues of automatic adaptive meshing of unstructured grids methods.
For an efficient and scalable implementation of calculation techniques on unstructured grids, parallel codes have to be developed. Data, including the grid's geometry description, the successive adapted grid structures, the solution solver's Jacobian matrix, the solution state variable and work vectors, must be partitioned among the processors in order to minimise the amount of intra-processor communications. The definition of a distributed data structure, well-suited for all the computational phases, is the first problem to be addressed. The meshes change during the solution progress, hence a dynamic partitioning is required to re-balance the separate partitions on the chosen number of processors. Some algorithms of mesh partitioning and their modification required and obtained here are shown.

6.1 Parallelisation of the solution method

Although in this thesis the work is focused on mesh adaptation, a short summary of the parallelisation of the solution solver used is given here. More details can be found in [141]. The solution techniques used are based on either finite element type solvers or equivalent finite volume type solvers on unstructured grids. The first type computes a *residual* at element level (5.6), defined as

$$\mathbf{R}^{(T)}(\mathbf{u}) = \int_T \left\{ G_1(\mathbf{u}) \frac{\partial \mathbf{u}}{\partial x_1} + G_2(\mathbf{u}) \frac{\partial \mathbf{u}}{\partial x_2} + G_3(\mathbf{u}) \frac{\partial \mathbf{u}}{\partial x_3} \right\} dT \ .$$

Then, the main idea is to redistribute this quantity to the nodes of the element, according to the choice of the scheme. The second type has many variants. Some of these variants take as control volumes the cell itself, and particular care has to be taken for the cell face normals in the finite volume flux which defines the numerical scheme. The parallelisation

issues are similar to finite element ones, with an additional difficulty of the inter-cell (control volume) face normals that have to be communicated and exchanged when the cell faces are shared between two adjacent processors.

The more usual equivalent finite volume method, that renders coherent finite volume schemes with the structured mesh finite volume world, uses control volumes that are constructed on the geometrical (homological) dual of the P1-finite element simplex. These control volumes - or dual cells - are constructed by joining the barycentre of the neighbouring elements around a node (fig. 5.1). The communication information of the fluxes through the faces of these cells requires very well laid out inter-processor information. Therefore the mesh partitioning algorithm and the choice of overlapping/non-overlapping inter-processor information has a significant effect.

The spatial discretisation of the equation set results in a system of nonlinear equations of type

$$S\frac{d\mathbf{U}}{dt} + \mathbf{R}(\mathbf{U}) = \mathbf{0} \ , \tag{6.1}$$

with a convenient initial and boundary conditions. In (6.1), S is a non-singular (lumped) mass-matrix.
System (6.1) is discretised in time using an implicit scheme such as a backward Euler scheme. Starting from a given $\mathbf{U}_0$, the solution at the time step $k = 1, 2, \ldots$, is found by solving the non-linear system of equations

$$S\frac{\mathbf{U}_k - \mathbf{U}_{k-1}}{\delta_k} + \mathbf{R}(\mathbf{U}_k) = \mathbf{0}$$

or, equivalently,

$$S\delta_k^{-1}\mathbf{U}_k - S\delta_k^{-1}\mathbf{U}_{k-1} + \mathbf{R}(\mathbf{U}_k) = \mathbf{0} \ , \tag{6.2}$$

where δ_k is the time increment at step k.

Taking Newton iterations at time levels k for the solution of (6.2) would then read:

$$\mathbf{U}_{k,l} = \mathbf{U}_{k,l-1} + \left[S\delta_k^{-1} + \frac{\partial \mathbf{R}}{\partial \mathbf{U}}(\mathbf{U}_{k,l-1})\right]^{-1} (-S\delta_k^{-1}\mathbf{U}_{k,l-1} - \mathbf{R}(\mathbf{U}_{k,l-1}) + S\delta_k^{-1}\mathbf{U}_{k-1}) \tag{6.3}$$

with $l = 0, 1, \ldots$, being the index associated to an iterative Newton procedure, and $\mathbf{U}_{k,0} = \mathbf{U}_{k-1}$. The iterations should stop when either a certain tolerance level is reached, or after a fixed number of iterations. At each intermediate Newton iteration a sub problem of linear form is to be solved:

$$\mathbf{U}_k = \mathbf{U}_{k-1} + \left[S\delta_k^{-1} + \frac{\partial \mathbf{R}}{\partial \mathbf{U}}(\mathbf{U}_{k-1}) \right]^{-1} \left(-\mathbf{R}(\mathbf{U}_{k-1}) \right) , \tag{6.4}$$

that can be written, dropping the index k, as

$$A\mathbf{u} = \mathbf{f}, \tag{6.5}$$

where $A \in \mathbb{R}^{n \times n}$ and $\mathbf{u}$, $\mathbf{f} \in \mathbb{R}^n$. More precisely,

$$\begin{aligned} A &= \left[S\delta_k^{-1} + \frac{\partial \mathbf{R}}{\partial \mathbf{U}}(\mathbf{U}_{k-1}) \right] , \\ \mathbf{u} &= (\mathbf{U}_k - \mathbf{U}_{k-1}) , \\ \mathbf{f} &= -\mathbf{R}(\mathbf{U}_{k-1}) . \end{aligned}$$

This system is solved by parallel Krylov iterations with Schwarz-type preconditioners. To overcome the locality of these preconditioners, they are made up of local corrections and global components. The local part, acting at the subdomain level, captures the strong couplings that appear between neighbouring subdomains, while the global part provides an overall communication among the subdomains.

The global component is usually referred to as a "coarse space correction", since usually it is defined on a space that is coarse with respect to the fine space containing the solution. The complexity of this auxiliary problem is much lower than that of the original problem, and its role is to diffuse information among the subdomains. In an analogous manner to multigrid methods, this coarse space is used to correct the "smooth" part of the error, whereas the local preconditioner is used to dump the "high-frequency" part of the error.
In all generality, the global correction term has the form $B_0 = R_0^T A_0^{-1} R_0$, where A_0 corresponds to the discretisation of the original problem on a coarse space V_0, and R_0 is the restriction operator from the fine space to the coarse space. The resulting preconditioner reads

$$P_{C,add}^{-1} = B_0 + P_S^{-1} = \sum_{i=0}^{M} B_i. \tag{6.6}$$

This preconditioner is fully additive since all the corrections on the subdomains and on the coarse space are added together. Alternatively, an hybrid preconditioner can be used:

$$P_{C,hybrid}^{-1} = P_S^{-1} + B_0 - P_S^{-1} A B_0. \tag{6.7}$$

The preconditioner (6.7) is called hybrid because the corrections on the subdomains are treated in an additive way (the term $P_S^{-1} + B_0$), as well as in a multiplicative way (the term $P_S^{-1}AB_0$). For more details, the reader is referred to [142].

The key element to obtain a scalable and efficient preconditioner is the proper definition of the coarse space. The general approach is to discretise the original problem on a coarse grid. However, the construction of the coarse grid and of the corresponding restriction operator R_0 can be difficult or computationally expensive for problems defined on unstructured grids in domains of complex shape, as typical in aeronautical applications. For this reason, agglomeration procedures are used to construct the coarse matrix for a two-level Schwarz preconditioner. The procedure is completely algebraic and very well-suited for unstructured grids.

6.1.1 Definition of distributed data structure

The development of a scalable parallel code requires the definition of an efficient distributed data structure. This data structure must efficiently handle the preconditioning phase, as well as all the other computational kernels of the scheme. The architecture type of the parallel computer is a distributed memory architecture. MPI message passing is used.

Usually the starting grid, of moderate size, and the underlying geometrical description (CAD) has been generated on a sequential computer like a linux workstation. The first step is the partition of the grid among the M processors. To that aim, we have used the *k-way* graph partitioning algorithms [73]. Given a graph $G = (V, E)$ with $|V| = n$, we partition V into k subsets $V_1, V_2, \ldots, V_k$ such that $V_i \cap V_j = \emptyset$, $|V_i| = n/k$, and $\bigcup_i V_i = V$, and the number of edges of E whose incident vertexes belong to different subsets is minimised. A k-way partition is commonly represented by a partition vector $\mathbf{p}$ of length n, such that for every vertex $v \in V, P[v]$ is an integer between 1 and k, indicating the partition at which vertex v belongs to. Given a partition P, the number of edges whose incident vertexes belong to different subsets is called the *edge-cut* of the partition.

The k-way partition problem is frequently solved by recursive bisection. That is, we first obtain a $2-$way partition of V_m and then we further subdivide each part using $2-$way partitions. After $\log k$ phases, the graph G is partitioned into k parts. Thus, the problem of performing a k-way partition can be solved by performing a sequence of $2-$way parts or bisections. Even though this scheme does not necessarily lead to optimal partitioning, it is used extensively due to its simplicity.
A $2-$way partitioning using a multilevel graph bisection algorithm, can be written as

follows, as described in detail in [73]

1. Coarsening phase. The graph G is transformed into a sequence of smaller graphs $G_1, \ldots, G_m$ such that $|V| > |V_1| > |V_2| > \ldots > |V_m|$;

2. Partition phase. A 2-way partition P_m of the graph $G_m = (V_m, E_m)$ is computed that partitions V_m into two parts, each containing half the vertexes of G;

3. Uncoarsening phase. The partition P_m of G_m is projected back to G by going through intermediate partitions $P_{m-1}, P_{m-2}, \ldots, P_1, P$.

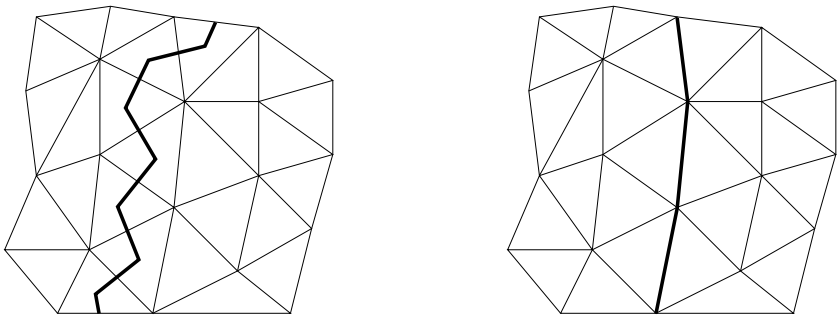


Figure 6.1: Example of vertex-oriented (left) and element-oriented domain decomposition (right).

The k-way algorithm can be used to decompose the direct graph or the dual graph of the grid, leading to vertex-oriented (VO) or element-oriented (EO) decompositions shown in figure 6.1. In the former, each vertex of the grid is assigned to a different processor, while in the latter the decomposition is done element-wise. A numerical comparison between the two approaches for the compressible Euler equations using explicit time-marching scheme can be found in [77], while results concerning implicit time-marching schemes can be found in [143]. VO decomposition allows a very suitable parallel data structure for the preconditioned type solver described briefly above. An example of a partitioned surface mesh is shown in figure 6.2. Each vertex of the computational grid is assigned to a unique processor, whereas there exist elements shared by more processors. These elements are stored on all processors which have at least one node of the element.

From the point of view of the parallel application, grid generation and graph partitioning result in a description of the processor M file; each M file contains a list of elements, their grid connectivity and coordinates, a list of the boundary elements and nodes, and their type (wall type - wing, fuselage, engine inlet,etc; symmetry plane, infinite inflow, outflow

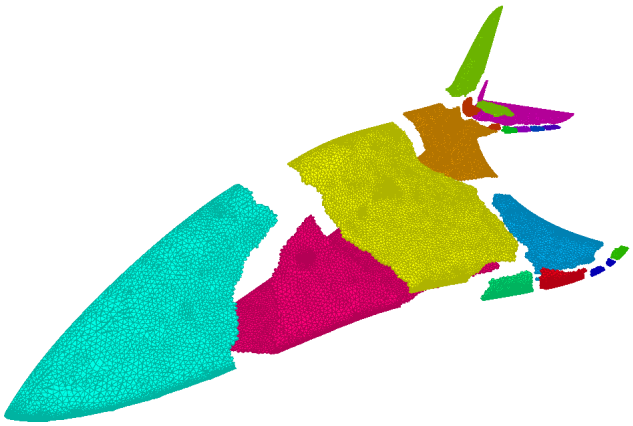


Figure 6.2: SmartFish partitioned mesh exploded view.

plane..etc). Also, a local-to-global mapping must be provided, to enable communication among the processors.

6.2 Parallelisation of the grid adaptation techniques

The adaptation techniques dynamically adapt the grid by localised refinement and derefinement during the calculation. Grid quality optimisation and moving by elastic spring analogies are performed. The multi-processor distribution of the mesh requires to be continuously repartitioned to maintain load balancing and manage the modifications of the internal grid data structure. This whole procedure requires repeated internal data structure modifications to track down the inter-processor boundaries, the new (old) nodes, elements and connectivity, and local re-ordering and re-numbering operations are required.

In figure 6.3 the main operations to be performed during the parallel mesh adaption are given. The renumbering phase is repeated twice per adaptation sweep, since, to perform the operations of structural changes like swapping and collapsing, and also the smoothing operations, the solver must exchange data, therefore requiring a new global numbering.
The solution algorithm gives an approximate solution of problem (6.1), for a discretisation on a given grid, say $\mathcal{T}_h^{(0)}$. At adaptation cycle i, the solution $\mathbf{U}^{(i)}$ corresponding to the solution of the transient problem on the grid $\mathcal{T}_h^{(i-1)}$, is projected on the grid $\mathcal{T}_h^{(i)}$, and then

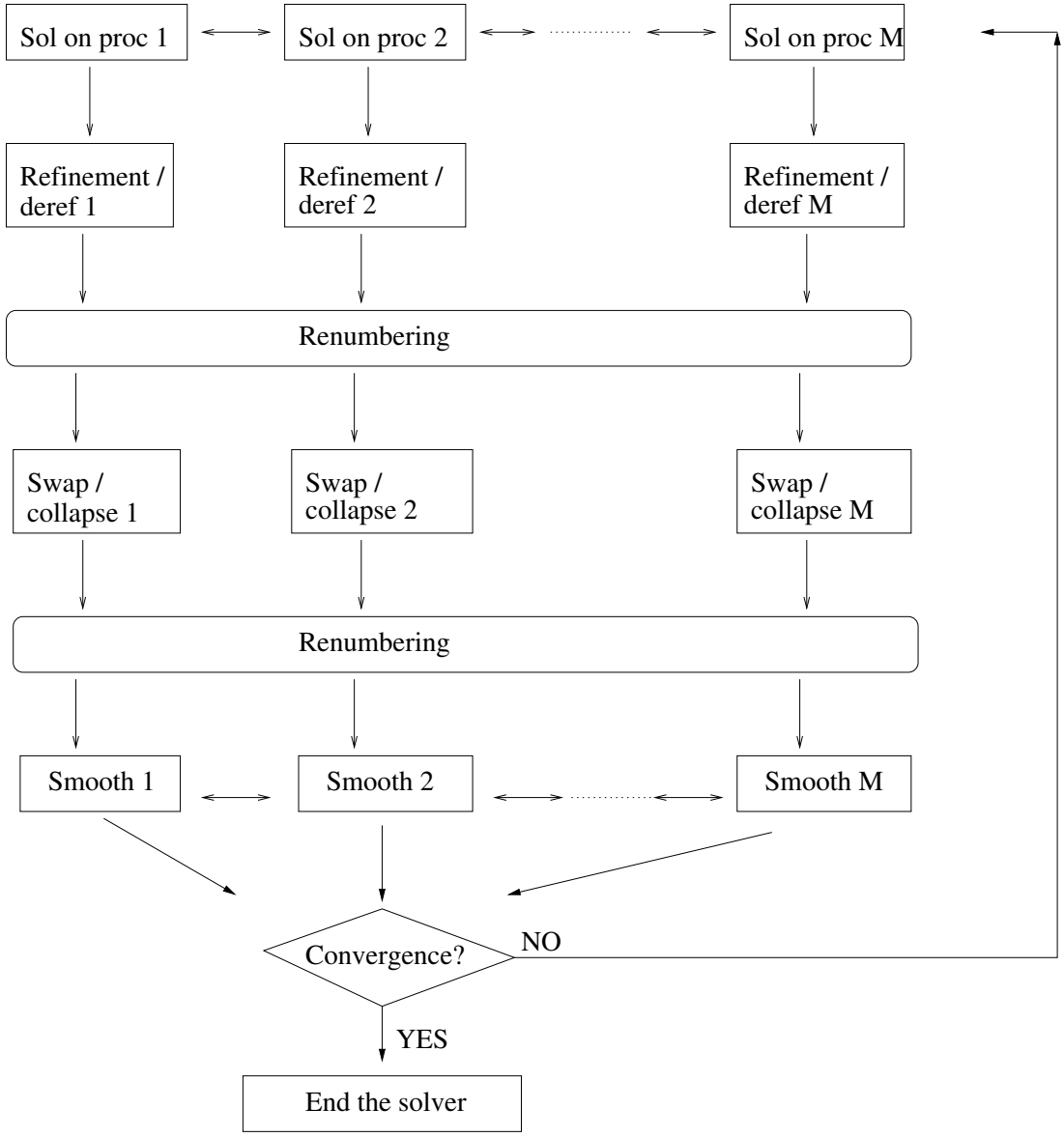


Figure 6.3: Flow-chart for the parallel mesh adaption.

used as a starting solution for problem (6.1), discretised on $\mathcal{T}_h^{(i)}$.

The goal of grid adaptation is to increase the accuracy of the solution process by locally enforcing the h-adaptivity using smaller discretisation elements. This process tends to uniformly equidistribute the local error η_h throughout the grid, $\mathcal{T}^h$. The first step in a grid adaptation algorithm is therefore to locally evaluate criteria corresponding to the solution error estimate, and mark out the zones to be modified in order to minimise globally the error. The criteria used should be as close as possible to the error estimations of the underlying discretisation scheme. There are several derivations of adaptation criteria. As described in section 2.2 adaptation requires in all cases a local error estimate per grid cell, $\eta(T_k)$, moderated by some tolerance levels.

Here the local estimates are all based on *a-posteriori* criteria, which require a solution on the starting grid. Then, grid refinement and derefinement operations are performed, taking as adaptation criteria functions of the current solution field (local error), and geometrical properties of the current grid (optimisation). Then, an optimisation step follows, based on geometrical properties of the current grid, followed by repartition, reordering and renumbering.

The incorporation of parallel grid adaptation within the solution process requires load balancing partitioning techniques to obtain well balanced subdomains. This introduces other algorithmic concepts such as parallel sorting and renumbering techniques.
The grid adaptation techniques are applied globally throughout a pre-partitioned mesh, and require careful renumbering and re-ordering internally per processor (local) and globally of the addresses of the entities, cells, shells, faces, edges, nodes... in order that the adaptation renders a global mesh that is in turn re-partitioned again. All this is dynamic and needs to have the partitioning procedure as an integral part of the adaptation procedure.
Let us outline some of the key steps of adaptation and how to install a parallel data structure to allow this in parallel.

6.2.1 Smoothing

The solution of the smoothing procedure depends strongly on the connectivity of the mesh. Following the notation of the chapter on mesh adaptation procedures, in the case of $\mathcal{N}_i \neq \mathcal{N}_{\text{opt}}$, the node neighbour number $\mathcal{N}_i$, affects the overall node displacements. The structural optimisation described in section 2.5 allows again each node to have $\mathcal{N}_{\text{opt}}$ neighbouring nodes. The smoothing procedures do not modify significantly the internal

mesh topology, the parallelisation is hence straightforward as long as a coherent numbering of the nodes, segments, faces and cells is employed.

6.2.2 Refinement/Derefinement

The parallelisation of the refinement and derefinement stages leads to the tracking of nodes created on an updated segment, which are considered as new border (interface) nodes. Refinement and derefinement phases will first mark out candidate border segments. Then the deletion phase is applied on the nodes, and the border segment re-constructed. The parallelisation of the above concepts requires careful renumbering and re-ordering techniques internally per processor, locally and globally. Also the choice of the number of overlapping within the subdomains (partitions) is crucial. One of the main difficulties was due to an external choice of overlapping partitioning to fit into a pre-written multigrid algorithm. The segments in the interface layer are distributed on one processor, and are considered as external on the other (see figure 6.4).

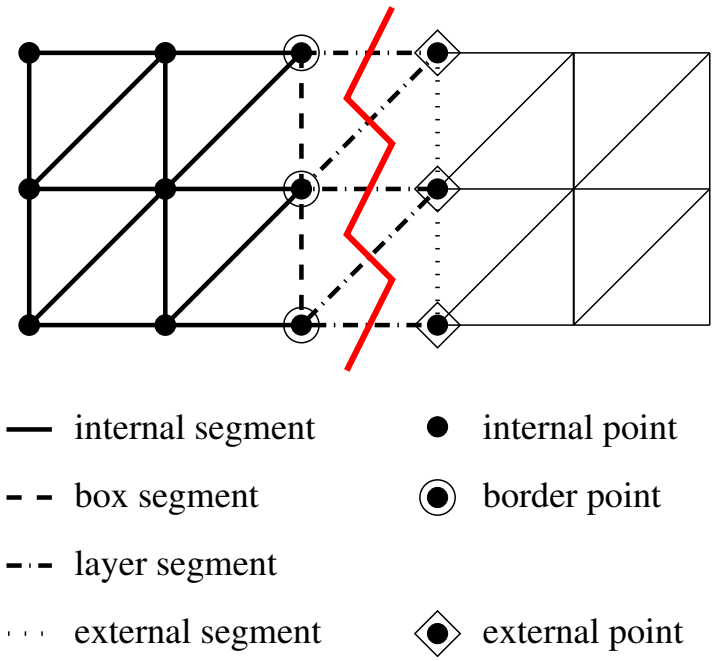
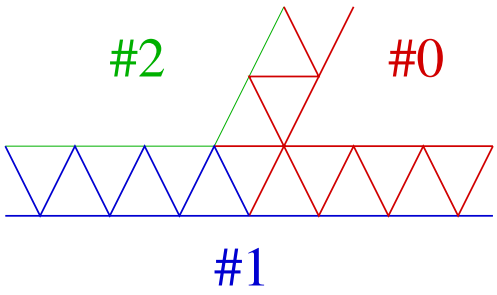
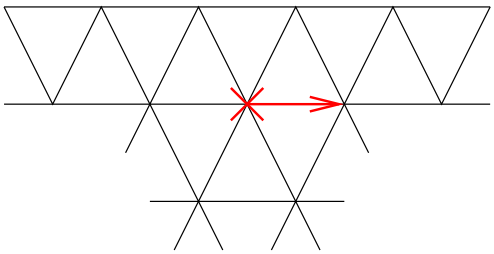


Figure 6.4: Layers of the segments during parallel adaptation

In particular, for **refinement** : Nodes created on an updated segment become new processor interface border nodes :

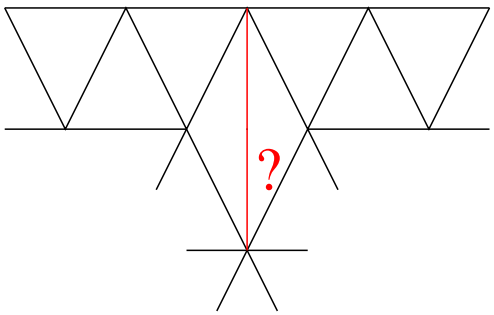


For **coarsening**, when attempting to delete a border node, a border segment must be chosen :



6.2.3 Structural changes

Diagonal swapping and **Edge collapsing** work as shown in the following sketch :



The parallelisation of such structural changes is one of the hardest points, especially for the choice of overlapping partitions. For these reasons the swapping and collapsing works most efficiently on the internal segments. However, diagonal swapping or face swapping is still straightforward across partition interfaces. Collapsing is often harder to control.
The basic stages encountered with completely parallel mesh adaptation are

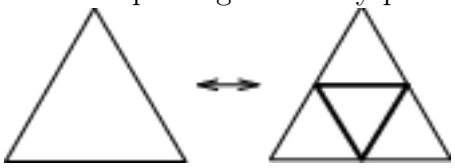
1. **Initial input file**. This file is transferred continuously onto the parallel network, filling up a first layer of neighbouring processors with the mesh entities - nodes, element connectivities, boundary patch definitions - until completion.

2. **Domain decomposition, reordering, renumbering**. Once the raw data has been distributed, the grid is then partitioned using ParMetis. The mesh entities are marked out together with the appropriate boundary interface entities. Then all entities are re-ordered and renumbered according to the new internal structure.

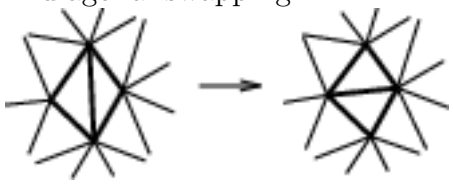
3. **Initialisation and solution**. A first solution is calculated throughout the processors. The different criteria for mesh adaptation are applied and the selected edges marked for adaptation.

4. **Mesh adaptation phase**. The adaptation phase is non-hierarchical (i.e. no memory of filiation). Consistent renumbering and reordering after *every* structural or geometrical modification within a single adaptation cycle, (criteria marking, derefinement, refinement, smoothing and stretching) is hence required to flag out the mesh entities. The different stages are

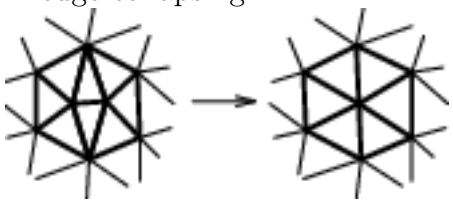
   - refinement and derefinement respecting boundary patches



   - reorder and renumber
   - structural optimisation - diagonal swapping



   - reorder and renumber
   - structural optimisation - edge collapsing



   - reorder and renumber
   - stretching, smoothing, regularisation

5. **Partitioning - return to [2]**. The resultant mesh is now re-partitioned for redistribution and further calculation. The partitioning algorithms in [73] are usually based on Recursive Bisection (RCB or RGB) and are parallel sorting algorithms

see figure 6.5. Each processor keeps part of the original data and redistributes the remainder to its neighbours.

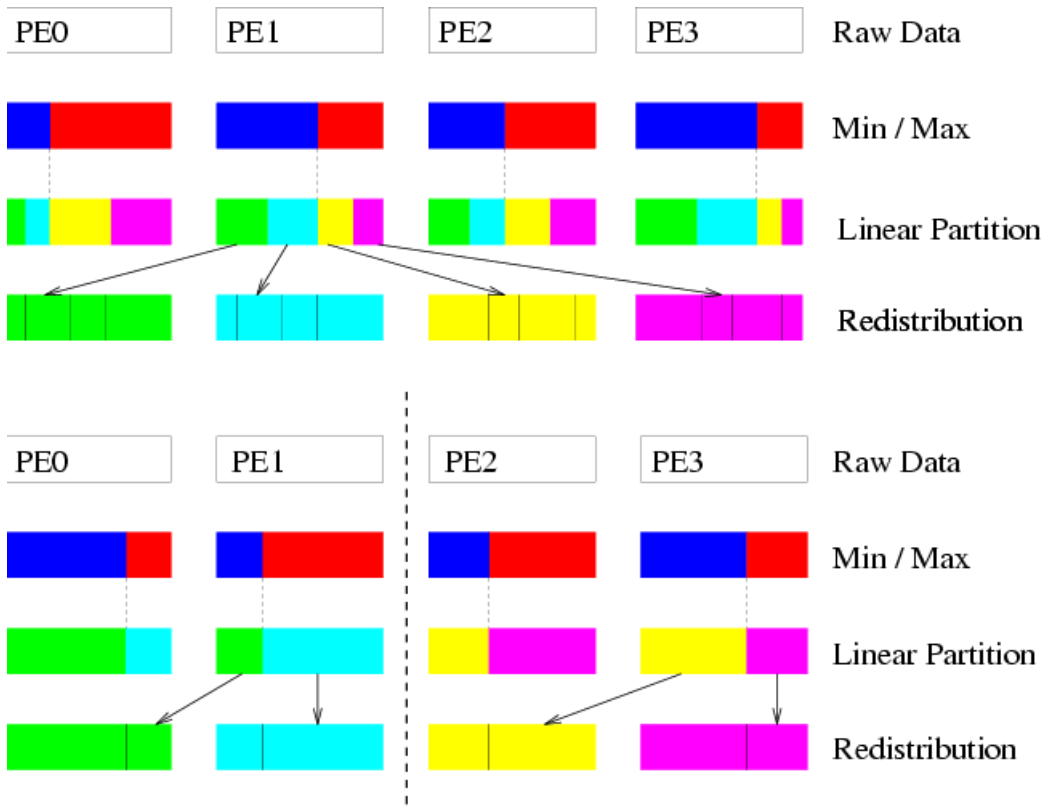


Figure 6.5: Parallel sorting algorithm for dynamic domain partitioning with RCB [77]

The elements entities maintains the book-keeping of the localisation using partitioning algorithms provided by an integrated use of ParMETIS, (i.e. directly integrating the package into the code) allowing for the use of efficient and optimal partition techniques.

6.2.4 Partitioning and Repartitioning

From the point of view of parallel computing, statically partitioned grid adaptation procedure may result in an unbalanced distribution of the workload among the processors. Here the updated adapted grids are repartitioned dynamically within the parallel adaptation procedure using a parallel graph partitioning algorithm. For these purposes, the library ParMETIS [72] can be used dynamically within the source code, as well as home-made partitioners as described above and in [77]. ParMETIS is called concurrently by each process and gives the new destination for each internal node. Then, data structure corresponding

to moved nodes are sent using MPI calls.

Two main partitioning types are proposed in ParMETIS: diffusing and remapping. Here we mainly use the remapping schemes as in figure 6.6, which redistribute the reference decomposition, and lead to partitions with smaller edge cuts.

6.2.5 Reorder and Renumber

As can be seen in figure 6.3, the reordering and renumbering operations are multiple and crucial in the parallel mesh adaptation technique. As nodes are reordered and renumbered all the vectors and matrices used in the code may have to be re-allocated in memory. In particular, the data structure for the parallel matrix-vector product must be recomputed. Fast and efficient multiple renumbering techniques are necessary for the grid entities: elements, segments, faces and nodes. To achieve this, explicit MPI library routines are used and a fast dynamic binomial search tree is developed to sort during the renumbering procedures. This sorting algorithm is based on a balanced binomial search tree algorithm AVL (Adelson-Velskii and Landis), [144].

Let us briefly describe the search tree algorithm. An AVL tree is a dynamically balanced binary search tree that is balanced according to its height H (see figure 6.7), defined by the number of nodes in the longest path from the root to any leaf.
For any node in the tree, the height of the left and right subtrees differ by at most one. To implement such a tree, a recursive operation on interlinked nodes is applied.
When a new node is inserted into the tree, it appears at the root, then moves along the branches until it finds an attachment to the tree. Once the node is inserted, the tree balance is checked. If no imbalance is found, another node is inserted and the process continues. If an imbalance is found, the heights of some nodes are fixed and the process repeated. When a node is deleted, the root becomes unbalanced. The look-up is performed to balance again.
Pairs of subtrees of each node hence differ by at most 1, which keeps the difference in height H between different branches minimal, and $H \leq n \leq 2^H - 1$.

1. Evaluate the difference of the height between different branches H
2. Balance requirement: sub-trees of every node differ in height by at most one
3. Every sub-tree is an AVL tree.
4. new nodes are inserted into the tree at the root and move along branches until they find an attachment
5. Imbalance? - No $\Longrightarrow$ insert another node
6. Imbalance? - Yes $\Longrightarrow$ fix H and repeat process

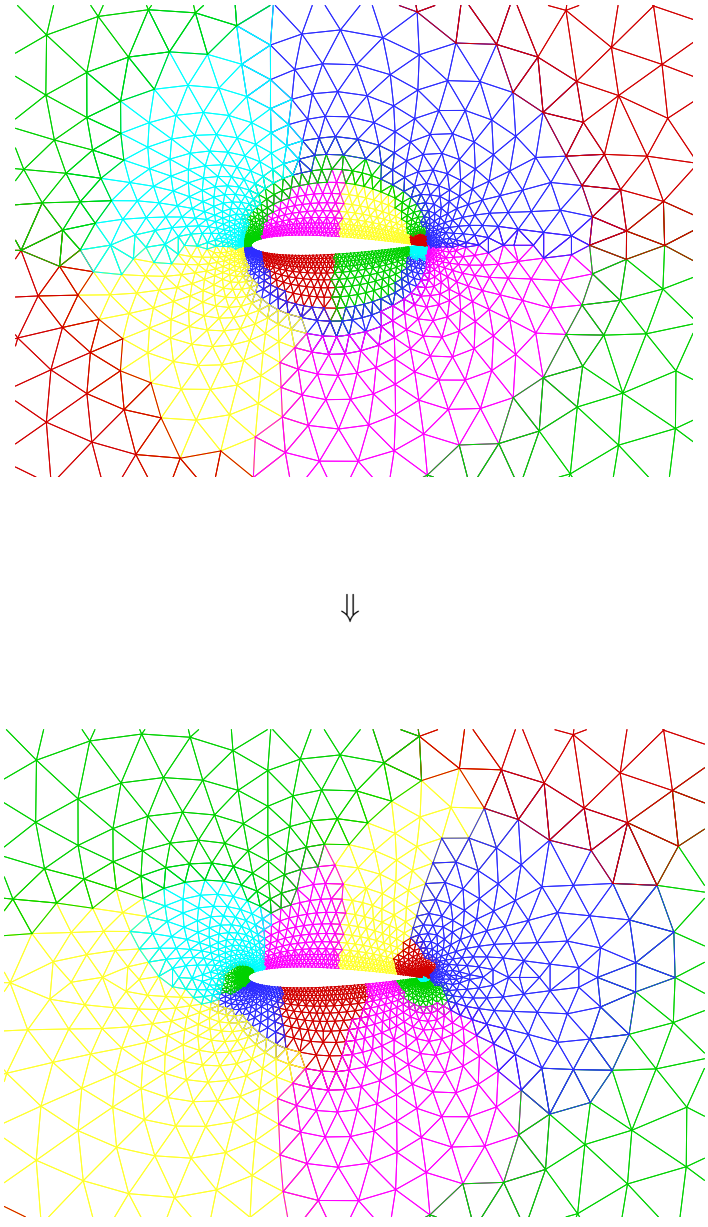
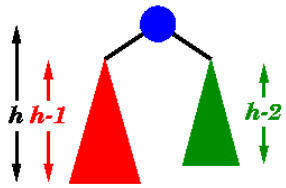
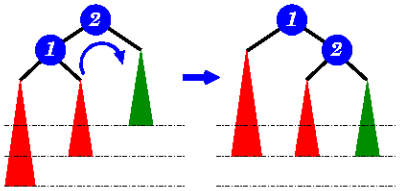


Figure 6.6: Effect of re-partitioning scheme using the remapping algorithm illustrated on an airfoil problem. On the top the original partition remapped below.



Balance requirement : left and right sub-trees differ by at most 1 in height

Insertion:



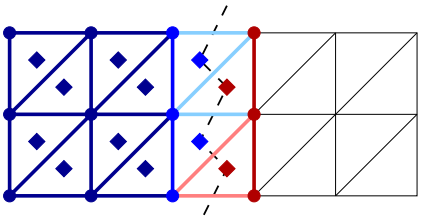
A new item is added to left sub-tree of node 1, causing its height to become greater than 2's *right sub-tree* - right-rotation is performed to correct imbalance

Figure 6.7: AVL binary tree insertion

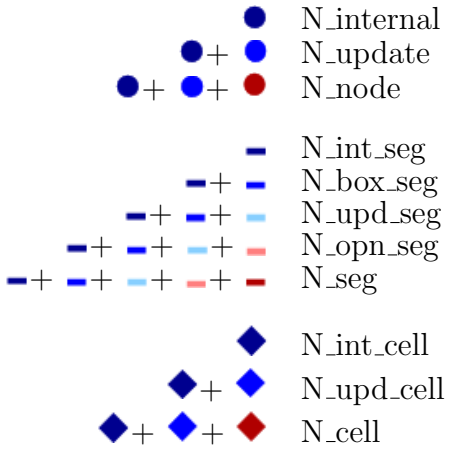
7. delete node $\Longrightarrow$ root is unbalanced – repeat look-up procedure to re-balance.

The operations of look-up, insertion and deletion are of $O(\log n)$, where n is the number of nodes in the tree, when the tree is balanced. Search steps $S(n)$ needed to find an item, are bounded by $\log(n) \leq S(n) \leq n$.

The renumbering details across a partition can be schematically represented as follows ;



where



where N_* denotes the node entities in the adaptation procedure following the nomenclature of the AZTEC iterative library [145], which is used to assure the liason between adaptation and solver.

6.3 Performance aspects

In order to verify the performance of the adaptive procedures, some of the test cases presented in the previous chapters have been re-run. First a 2D NACA 0012 airfoil is used to test the code on the Linux cluster (*Pleiades*[1]) used for the computations, as well as measuring the CFD code and the adaptation parts elapse time. In particular the nodes used for the computations reported here are bi-processor, bi-core.
Then various 3D test cases, are used to measure the total time of the adaptation process with respect to the CFD time, and the breakdown of the adaptation process stages.

6.3.1 2D results

In order to test the charge of the adaptation process, with respect to the total time of the CFD computation, a same NACA 0012 airfoil case was executed with a different number of processors. In particular the adaptation process was run with refinement and derefinement procedures, adaptation with respect to Mach gradient, reprojection as described in the chapter on geometry approximation, 20 optimisation cycles (swapping and collapsing), and 20 smoothing cycles. Four adaptation steps were carried out, hence from an initial grid of 2 355 nodes, 4 537 triangular elements, and 173 boundary faces, the flow solver is run and the solution adapted in turn four times, and a final solution obtained from the final adapted grid. The final grid characteristics for the computations with different number of processors are reported in table 6.1. In figure 6.8 instead we report the total computational execution time of the flow solver and that of the adaptation process. As we can notice, the total time of adaptation can be considered negligible compared to the solution time, being at most, less than 1.2% of the total CFD computation time.

Processors	*N_nodes*	*N_elements*	*N_surf_el*
1	35 130	69 615	645
2	35 082	69 527	637
4	34 335	68 031	639
8	34 839	69 035	643

Table 6.1: Final grid sizes for different number of processors after 4 adaptation steps. 2D NACA 0012 airfoil.

[1] http://pleiades.epfl.ch/

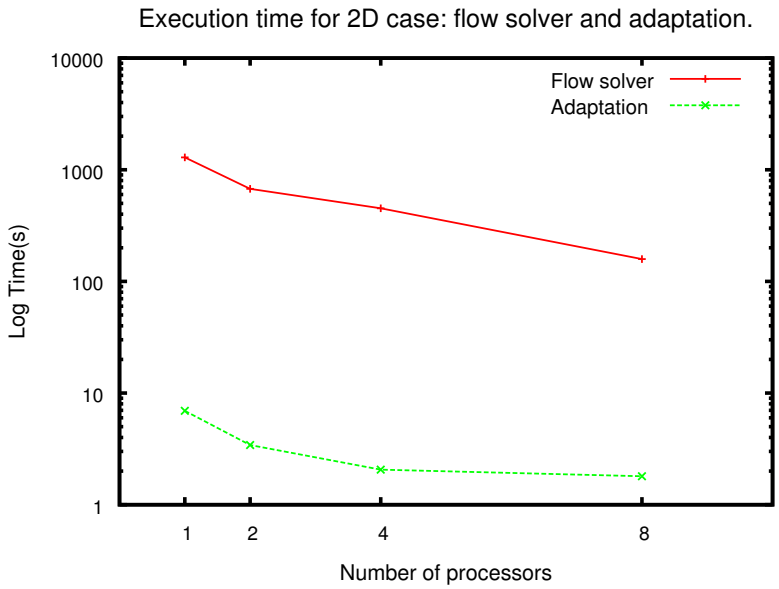


Figure 6.8: Execution time for flow solver and adaptation process on multiple processors. 2D NACA 0012, four adaptation steps.

6.3.2 3D results

In a similar way to that of the 2D case above, we first compare the execution time of the flow solver and that of the adaptation with a different number of processors. The test is carried out with the 3D Wedge examples of the previous chapters, with an initial mesh of 306 415 tetrahedral elements, 54 370 nodes and 12 906 boundary faces. The adaptation process was run with refinement only, with respect to the difference in density on the segments, 30 optimisation passes and 10 smoothing cycles. Three adaptation steps were performed, each one after obtaining a partial solution with the flow solver, and a final solution obtained from the final adapted grid. This was carried out for 8, 16 and 32 processors as shown in table 6.2 with the final adapted grids characteristics. The final mesh and solution obtained with 32 processors was then used for a single adaptation step, using 64 and 128 processors. In this last case only one solution step is required, as for the adaptation, since the grid is immediately adapted to the solution obtained with the previous computation.

Here the computations start from 8 processors, rather than 1, due to memory requirements for the grid obtained after the third adaptation step. Figures 6.9 and 6.10 show the computational times plotted for the three adaptation steps, and for the restart, on a

Processors	*N_nodes*	*N_elements*	*N_surf_el*
8	1 118 350	6 721 070	50 714
16	1 123 326	6 753 716	50 963
32	1 130 577	6 801 332	51 073
64	3 843 209	23 302 721	85 652
128	3 842 653	23 290 638	85 641

Table 6.2: Final grid sizes for different number of processors after 3 adaptation steps, and after 1 adaptation step for 64 and 128 restarting from 32 final solution and grid. 3D Wedge.

different number of processors. Once again the total adaptation time is negligible compared to that of the flow solver, reaching at most 2% of the total solution time.

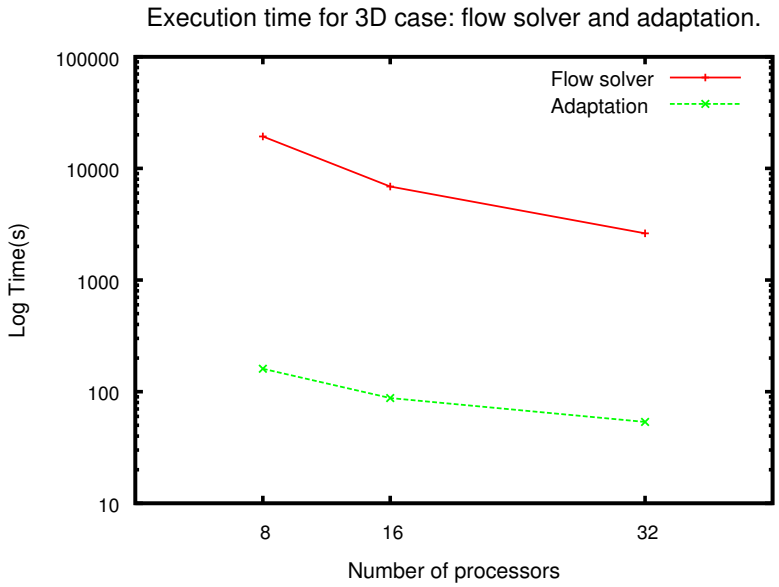


Figure 6.9: Execution time for flow solver and adaptation process on multiple processors. 3D wedge, three adaptation steps.

Adaptation execution breakdown

Although the adaptation execution times are far less than the total computation, where the flow solver is accounted for, it is interesting to examine the various stages of the adaptation cycle and see the impact these have on the use of computational resources used. Therefore what follows is a breakdown of the adaptive cycle in three main blocks, as shown

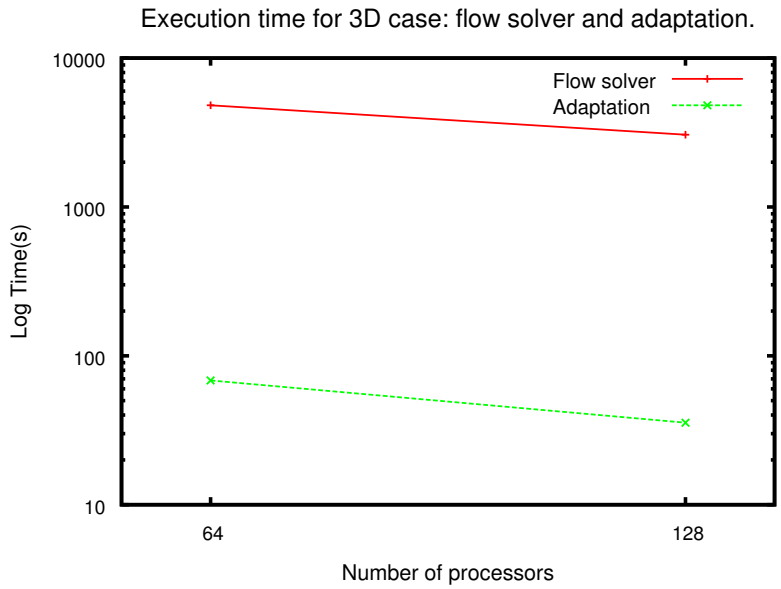


Figure 6.10: Execution time for flow solver and adaptation process on multiple processors. 3D wedge, one adaptation step after restart.

in figure 6.3, with the refinement/derefinement and renumbering as a first block, swapping/collapsing and renumbering a second block, and smoothing being the third and last block.

The previous 3D wedge initial mesh was used to start a computation with two adaptation steps on 4 processors, and a third adaptation step for 8, 16 and 32 processors. The reason for this choice is that it is not possible to run three adaptation steps on 4 processors, due to memory constraints. The final solution and mesh of this last computation were once again used as a starting point for an adaptation step carried out with 64 and 128 processors. Adaptation conditions were maintained the same as for the previous case. Grids for all steps and number of processors are given in table 6.3. Figures 6.11 and 6.12 show the breakdown of the adaptation process time for the first two steps with multiple processors, and figure 6.13 that of the third step. Figure 6.14 instead shows the breakdown for the restarted case.
As we can see from the above examples, the two optimisation procedures are far more time consuming than the refinement/derefinement block for the case where the difference physical criteria is chosen.

Steps	Processors	*N_nodes*	*N_elements*	*N_surf_el*
Step 1	4	140 180	811 918	20 902
	8	140 129	811 258	20 868
	16	140 133	810 941	20 915
	32	140 108	810 176	20 941
Step 2	4	362 718	2 145 897	31 844
	8	364 891	2 158 249	31 728
	16	365 409	2 161 629	31 911
	32	367 912	2 176 392	31 923
Step 3	8	1 118 350	6 721 070	50 714
	16	1 123 896	6 756 786	50 931
	32	1 131 649	6 806 732	50 949
restart	64	3 843 363	23 310 788	85 648
	128	3 842 828	23 298 937	85 647

Table 6.3: Step by step grid sizes for different number of processors after 1, 2 and 3 adaptation steps, and after 1 adaptation step for 64 and 128 restarting from 32 final solution and grid. 3D Wedge.

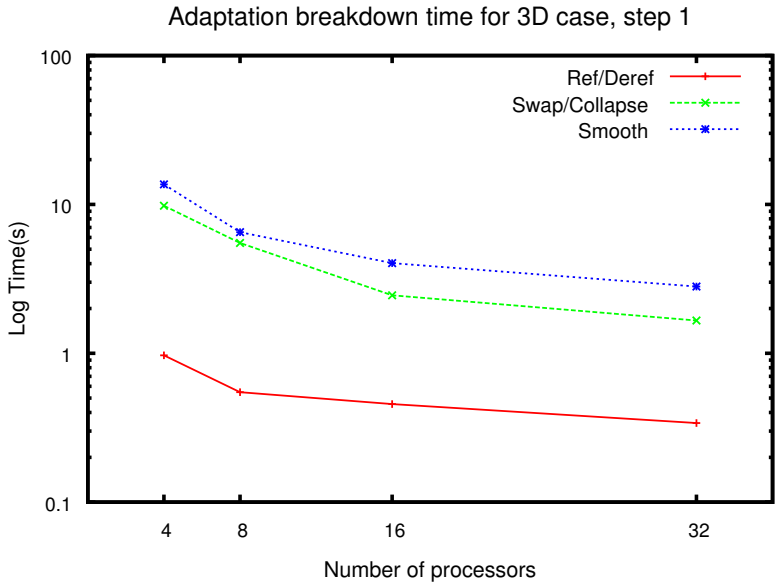


Figure 6.11: Execution time for adaptation blocks on multiple processors. 3D wedge, first adaptation step.

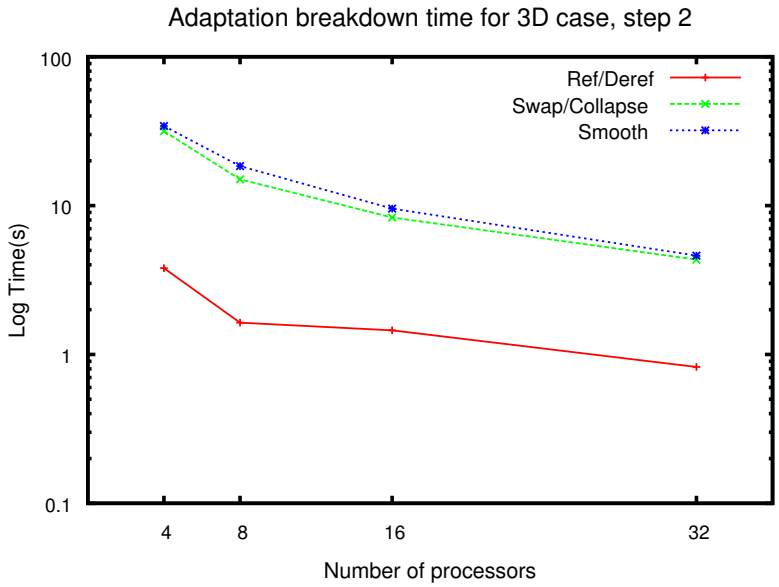


Figure 6.12: Execution time for adaptation blocks on multiple processors. 3D wedge, second adaptation step.

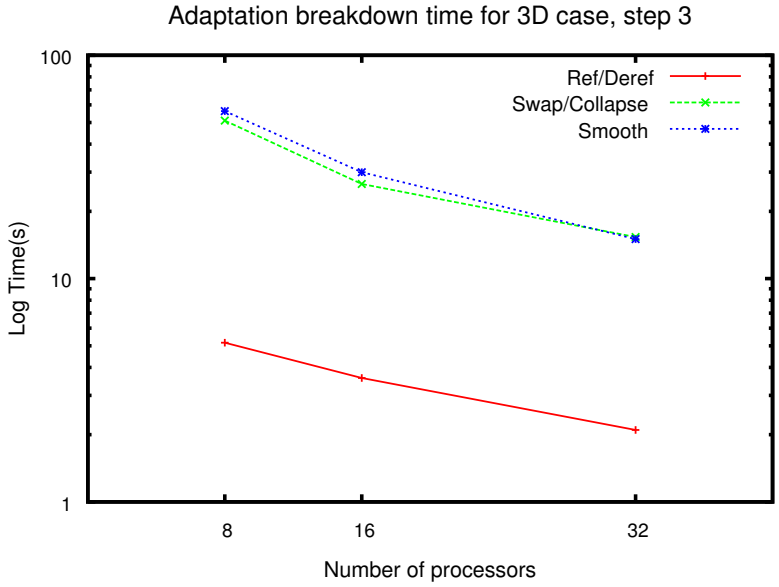


Figure 6.13: Execution time for adaptation blocks on multiple processors. 3D wedge, third adaptation step.

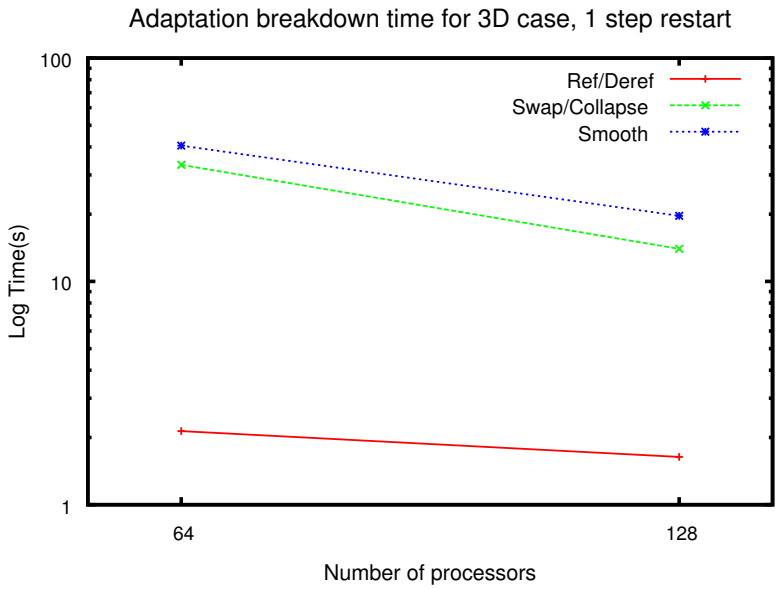


Figure 6.14: Execution time for adaptation blocks on multiple processors. 3D wedge, one adaptation step after restart.

Subdivision schemes

We now consider the adaptation time when using the different subdivision schemes as described in the geometry approximation chapter. The test case used here is the SmartFish concept aircraft, with an initial grid of 676 524 tetrahedral elements, 129 865 nodes and 65 770 boundary faces. The idea here is to measure the difference between the various schemes, hence rather than using a varying number of processors, we fix this to 8. For these tests we turned the smoothing off and the adaptation process was carried out for one step with refinement and derefinement, 20 optimisation passes, using the ZZ-like indicator with respect to Mach number as criteria. Table 6.4 shows the different grids obtained for the subdivision schemes, where the differences are due to the swapping/collapsing procedure. In figure 6.15 the time in seconds for the adaptation process blocks, when using different subdivision schemes, is given. As we can see all times are of the same magnitude, with the diamond difference and distance based butterfly schemes performing slightly worse than the multi-node version.

Subdivision type	*N_nodes*	*N_elements*	*N_surf_el*
Mid-point	554 750	3 080 199	199 666
Multi-node	554 760	3 080 450	199 654
Diamond difference	554 762	3 081 941	199 630
Distance butterfly	554 623	3 080 210	199 368

Table 6.4: Grid sizes for different subdivision schemes and after 1 adaptation step with 8 processors. SmartFish test case.

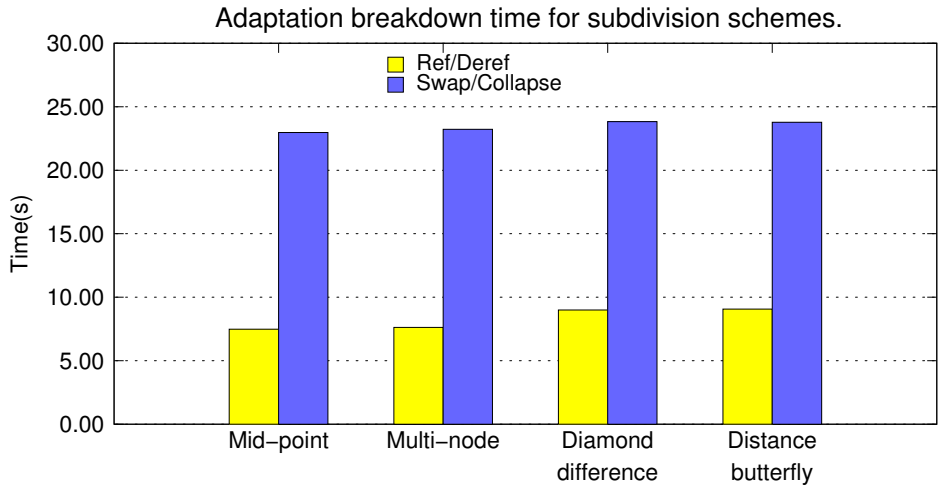


Figure 6.15: Adaptation blocks times for different subdivision schemes and after 1 adaptation step with 8 processors. SmartFish test case.

CAD projection

We now examine the execution times of the various adaptation blocks as described above when CAD projection is used, as described in the chapter on geometry conservation. The test case used here is that of the ONERA M6 wing. Here the initial grid is composed by 477 262 tetrahedral elements, 91 379 nodes and 38 926 boundary faces. Once again only a set number of processors is used for this computation, which will be 16, since we are only interested in observing the overhead when using CAD projection. Here a single adaptation step was carried out, with respect to the Mach number, using the simple error estimator for the refinement and derefinement and with the addition of the physical difference criteria for the refinement. Smoothing iterations were set to 10 and optimisation steps to 20. Furthermore the case was run with reprojection of new nodes only in a first computation, and of all nodes in-between smoothing for another one. Table 6.5 shows the different grids obtained for the different cases, whilst figure 6.16 shows the execution times for the three adaptation blocks. Smoothing time in figure 6.16, for the case when CAD projection is used during the smoothing operation, can be broken down into two parts to obtain the real smoothing time and the projection time, as shown in figure 6.17. As we can notice from this last figure, the projection time becomes more important when carried out for all surface nodes, increasing threefold the whole smoothing time block. However, it is also interesting to notice how the actual smoothing time decreases, due to re-projection, when compared to the smoothing times of figure 6.16.

Projection type	*N_nodes*	*N_elements*	*N_surf_el*
No Projection	171 219	955 543	57 700
CAD new nodes	171 194	955 793	57 660
CAD new and smoothing	171 192	955 863	57 658

Table 6.5: Grid sizes for CAD projection after 1 adaptation step with 16 processors. ONERA M6 wing.

Error indicators

Finally we consider the execution times of the different adaptation stages when using the different error indicators discussed in the previous chapter. The ONERA M6 wing described earlier is used, and the number of processors is again fixed at 16 since the objective here is to evaluate the different error indicators performances. A single adaptation step is performed with respect to Mach number for the three different indicators, for

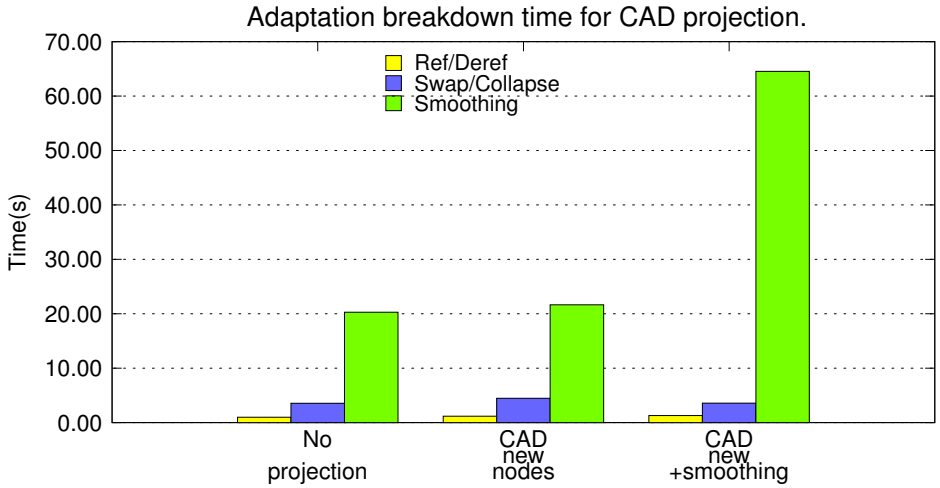


Figure 6.16: Adaptation blocks times for CAD projection after 1 adaptation step with 16 processors. ONERA M6 wing.

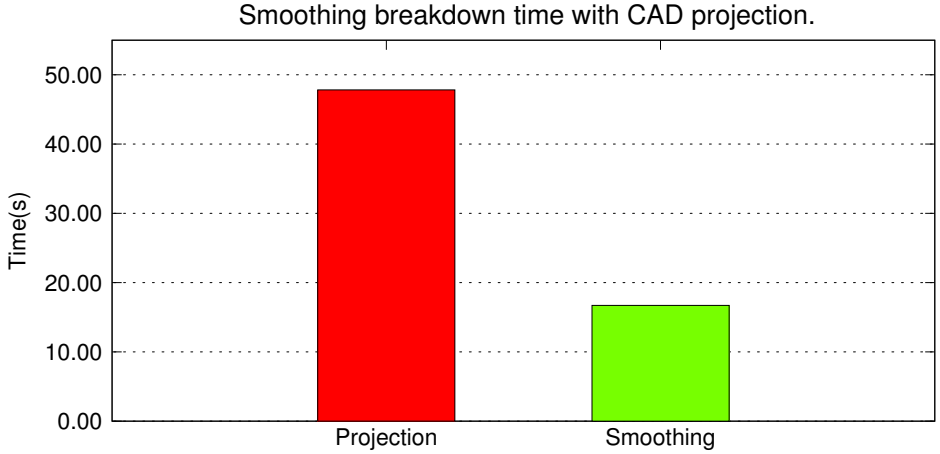


Figure 6.17: Projection and smoothing time breakdown for CAD projection during smoothing. ONERA M6 wing.

both refinement and derefinement. One case was run using the simple error indicator for derefinement with respect to Mach number and a mixture of ZZ-like indicator and the physical difference criteria with respect to density. In all cases CAD projection for new nodes only was used, and 20 cycles of both optimisation and smoothing were carried out. Table 6.6 shows the grid sizes obtained with the different error indicators, which is more important here than in other cases, as it must be considered when examining figure 6.18. In fact the different mesh sizes obtained have an effect on the overall adaptation times. What is interesting to point out here, is that despite mesh sizes and results (as shown at the end of the previous chapter) being very similar for the face gradient and ZZ-like indicators, the latter is more time consuming. Note that times for error indicators are measured during the refinement/derefinement stage. For completeness all times of the adaptation are presented in figure 6.18.

Error indicator	*N_nodes*	*N_elements*	*N_surf_el*
Mixed	246 084	1 382 446	75 096
Simple	226 061	1 280 510	63 622
Face-grad	263 766	1 478 357	81 962
ZZ-like	264 392	1 483 283	82 010

Table 6.6: Grid sizes for different error indicators and after 1 adaptation step with 16 processors. ONERA M6 wing.

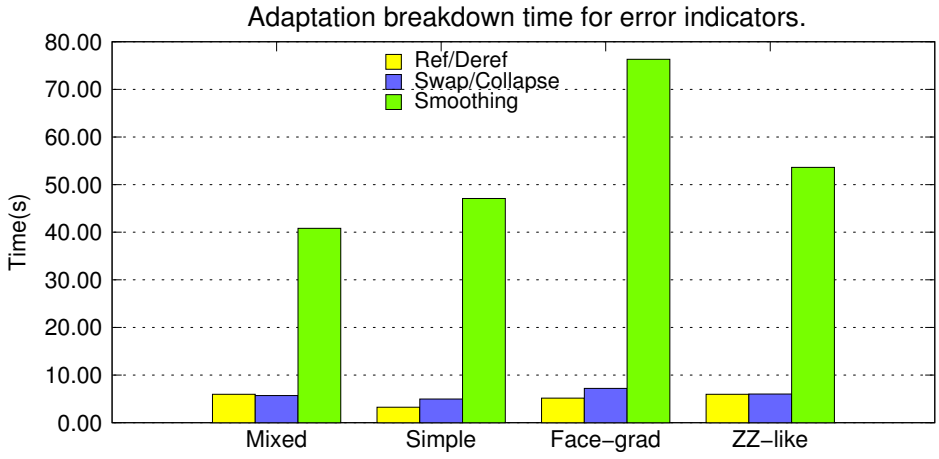


Figure 6.18: Adaptation blocks times for different error indicators and after 1 adaptation step with 16 processors. ONERA M6 wing.

7

Summary, Conclusions and Perspectives

The work presented in this thesis addresses several important aspects in the field of mesh adaptation. The main concepts of the adaptive techniques used are described in detail, and some original methods for improving the whole procedure, by conforming to the geometry, are introduced. This is completed by the addition of a posteriori error indicators, and a description of the parallel algorithms used.

When creating a computational discretisation of an object, the precise geometrical definition is required in order to generate a grid. This is also true when refining the mesh in locations where the stresses will be higher, or where flow features are going to appear. However, it is rare to know the exact location of these stresses and flow features a priori, hence the mesh is adapted manually in large portions of a computational domain. The use of dynamic integrated mesh adaptation avoids the need of prior knowledge of the solution features and their locations. Given an initial coarse grid, the mesh adaptation process will automatically detect the critical zones of the solution, and adapt the mesh accordingly, enhancing the accuracy and rendering an improved solution.

There remains a problem in the process described above. Just as one cannot discretise a geometry without the set of curves and surfaces defining it, the adaptation algorithm cannot place the refined elements on the underlying geometry without its computational representation. This was the first goal achieved here, by inserting a dedicated library for the treatment of the most commonly used geometric representations in present day CAD software. The results presented clearly show the importance of maintaining the geometric representation, with undoubted benefits to the acquisition of a precise solution.

Encouraging results have also been obtained from the proposed use of interpolating subdivision in the cases where no geometrical description is available. In particular three alternative schemes have been proposed, and all provided satisfactory solutions for a com-

plex geometry. These methods also proved to be very efficient, in a parallel environment.

Furthermore, a posteriori error estimators based on recovery procedures have been implemented, with interesting results. Three alternatives were proposed, two based on the recovered gradient of the solution and one based on the solution itself. The adaptations obtained from these error estimators, showed in particular, good capabilities of capturing shock waves, using an included shock capturing procedure.

In order to carry out all the above processes on realistic simulations with computational meshes of the order of one million nodes, mono processor-serial calculations are not feasible. It is hence mandatory that all the above techniques be developed in parallel. Numerical investigations on the parallel performances of these methods proved the efficiency of all parts of the adaptation process. Execution time of the adaptation is low compared to that of the physics solver. Moreover, the various techniques implemented are also computationally efficient as shown from the results presented in the chapter on parallel aspects.

The original contributions of this work can be summarised in:

- Inclusion of geometry description in a parallel mesh adaptation environment;
- Study on reprojection algorithms for representing surface geometries;
- A posteriori error estimates for adaptation criterion, blending physical gradients and mathematical error estimation;
- Test cases of challenging geometries using the above methods in a completely parallel environment.

7.1 Future work

The perspectives for this type of domain are multiple and continuosly changing, since it covers a wide range of active research topics. Here we try to identify those which could be most interesting to apply to the work proposed here.

Isogeometric analysis

This is a particularly interesting subject which has been further developed in recent years by the group of Prof. Hughes [68]. It is a very promising project which makes further use of NURBS, other than just as CAD description. The idea behind it is to use the

geometry definition of the bounding surfaces in such a way to replace finite elements with "NURBS elements" leading to Isogeometric Analysis. This has the advantage of losing the CAD-CFD interface, since the geometry itself is used as the computational mesh.

Hybrid grids

Use of hybrid meshes for the discretisation of the domain is a topic that has attracted a lot of interest in recent years [82, 83, 84, 86, 87, 88, 90, 146, 147, 148]. The scope of the use of hybrid grids is straightforward: take the advantages of both structured and unstructured grids and incorporate them to overcome the disadvantages. In general this mixture is done in order to better capture the features of the flow field in different areas. With hexahedra or prisms being used for boundary layers and wakes, where the gradients are high and the flow is strongly directional following the body surface, and tetrahedra to cover the rest of the domain and other features, such as shocks and vortices [149]. The adaptation schemes however increase vastly in complexity due to the new topology of the mesh which makes refinement and de-refinement much more difficult. A first step in reducing this difficulty is achieved by using the same type of elements in specific areas as described above, and confine the problem to an interface with specific elements such as pyramids. Complex and memory consuming data structures are also an issue that should not be underestimated. Division rules for refinement/coarsening that minimise the number of pyramids and simplify the implementation of adaptation are required.

Bibliography

[1] R. Löhner. Mesh adaptation in fluid mechanics. *Engrg. Fract. Mech.*, 50:819–847, 1995.

[2] D.J. Mavriplis. Unstructured mesh generation and adaptivity. Technical Report TR-95-26, ICASE-NASA, 1995.

[3] M. Shephard, J. Flaherty, H.L. de Cougny, C. Ozturan, C. Bottasso, and M. Beall. Parallel automated adaptive procedures for unstructured meshes. *AGARD Report R-807*, 1995.

[4] T.J. Baker. Mesh adaptation strategies for problems in fluid dynamics. *Finite Elements in Analysis and Design*, 25:243–273, 1997.

[5] R. Richter. *Schémas de capture de discontinuités en maillage non-structuré avec adaptation dynamique: Applications à l'aérodynamique*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1993.

[6] N.P. Weatherill, M.J. Marchant, O. Hassan, and D.L. Marcum. Grid adaptation using a distribution of sources applied to inviscid compressible fluid simulations. *Int. J. Numer. Meth. Fluids*, 19:739–764, 1994.

[7] S.Z. Pirzadeh. An adaptive unctructured grid method by grid subdivision, local remeshing, and grid movement. AIAA Paper 99-3255, 1999.

[8] P.L. George, P.J. Frey, and F. Alauzet. Automatic generation of 3D adapted meshes. Fifth World Congress on Computational Mechanics, Vienna, Austria, July 2002.

[9] F. Alauzet, P.L. George, B. Mohammadi, P. Frey, and H. Borouchaki. Transient fixed point-based unstructured mesh adaptation. *Int. J. Numer. Methods Fluids*, 43:729–745, 2003.

[10] Y. Ito, A.M. Shih, R.P. Koomullil, and B.K. Soni. A solution-based adaptive redistribution method for unstructured meshes. In *Proc. 15th int. meshing roundtable*, pages 147–161, 2006.

[11] I. Babuvška and W. C. Rheinboldt. *A-posteriori* error estimates for the finite element method. *Int. J. Numer. Meth. Engng.*, 12(10):1597–1615, 1978.

[12] M. Ainsworth and J.T. Oden. A procedure for a posteriori error estimation for h-p finite element methods. *Comput. Methods Appl. Mech. Engrg.*, 101:73–96, 1992.

[13] M. Ainsworth and J.T. Oden. A unified approach to a posteriori error estimation using element residual methods. *Numerische Mathematik*, 65:23–50, 1993.

[14] O.C. Zienkiewicz. The background of error estimation and adaptivity in finite element computations. *Comput. Methods Appl. Mech. Engrg.*, 195:207–213, 2006.

[15] O.C. Zienkiewicz and J.Z. Zhu. A simple error estimator and adaptive procedure for practical engineering analysis. *Int. J. Numer. Meth. Engng.*, 24:337–357, 1987.

[16] O.C. Zienkiewicz and J.Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. I: The recovery technique. *Int. J. Numer. Meth. Engng.*, 33:1331–1364, 1992.

[17] O.C. Zienkiewicz and J.Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. II: Error estimates and adaptivity. *Int. J. Numer. Meth. Engng.*, 33:1365–1382, 1992.

[18] R. Verfürth. *A Review of A Posteriori Error Estimation and Adaptive Mesh-Refinement Techniques*. Wiley-Teubner, New York, 1996.

[19] M. Ainsworth and J.T. Oden. A posteriori error estimation in finite element analysis. *Comput. Methods Appl. Mech. Engng.*, 142:1–88, 1997.

[20] O.C. Zienkiewicz, B. Boroomand, and J.Z. Zhu. Recovery procedures in error estimation and adaptivity, part I: adaptivity in linear problems. *Comput. Methods Appl. Mech. Engrg.*, 176:111–125, 1999.

[21] B. Boroomand and O.C. Zienkiewicz. Recovery procedures in error estimation and adaptivity, part II: adaptivity in nonlinear problems of elasto-plasticity behaviour. *Comput. Methods Appl. Mech. Engrg.*, 176:127–146, 1999.

[22] M. Möller and D. Kuzmin. Adaptive mesh refinement for high-resolution finite element schemes. *Int. J. Numer. Meth. Fluids*, 52:545–569, 2006.

[23] G. Maisano, S. Micheletti, S. Perotto, and C.L. Bottasso. On some new recovery-based a posteriori error estimators. *Comput. Methods Appl. Mech. Engng.*, 195:4794–4815, 2006.

[24] Y. Kallinderis and P. Vijayan. Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. *AIAA Journal*, 31(8):1440–1447, 1993.

[25] R. Richter and P. Leyland. Auto-adaptive finite element meshes. In *NASA. Langley Research Center, ICASE/LaRC Workshop on Adaptive Grid Methods*, pages 219–232, oct 1995.

[26] R. Bank, A. Sherman, and A. Weiser. Some refinement algorithms and data structures for regular local mesh refinement. Scientific Computing (Applications of Mathematics and Computing to the Physical Sciences), pages 3–17. 1983.

[27] R. Löhner. Regridding surface triangulations. *J. Comput. Phys.*, 126:1–10, 1996.

[28] T.J. Baker. Adaptive modification of time evolving meshes. *Comput. Methods Appl. Mech. Engrg.*, 194:4977–5001, 2005.

[29] T. Baker. Identification and preservation of surface features. In *Proc. 13th int. meshing roundtable*, pages 299–310, 2004.

[30] G.F. Carey. *Computational Grids: Generation, Adaptation and Solution Strategies*. Taylor & Francis, 1997.

[31] T.J. Baker. Mesh deformation and modification for time dependent problems. *Int. J. Numer. Meth. Fluids*, 43:747–768, 2003.

[32] M.A.T. Walter, A.A.Q. Abdu, L.F. Figueira da Silva, and J.L.F. Azevedo. Evaluation of adaptive mesh refinement and coarsening for the computation of compressible flows on unstructured meshes. *Int. J. Numer. Meth. Fluids*, 49:999–1014, 2005.

[33] Y. Savoy and P. Leyland. Parallel mesh adaptation for unstructured grids within the idemas project. Technical report, IMHEF-DGM EPFL, January 2001.

[34] L.A. Freitag and C. Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *Int. J. Numer. Meth. Engng*, 40:3979–4002, 1997.

[35] L.A. Freitag and P. Plassmann. Local optimization-based simplicial mesh untangling and improvement. *Int. J. Numer. Meth. Engng*, 49:109–125, 2000.

[36] L.A. Freitag and P.M. Knupp. Tetrahedral mesh improvement via optimization of the element condition number. *Int. J. Numer. Meth. Engng*, 53:1377–1391, 2002.

[37] J.T. Batina. Unsteady euler airfoil solutions using unstructured dynamic meshes. *AIAA Journal*, 28(8):1381–1388, 1990.

[38] R.D. Rausch, J.T. Batina, and H.T. Yang. Spatial adaptation procedures on tetrahedral meshes for unsteady aerodynamic flow calculations. AIAA Paper 93-0670, 1993.

[39] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Comput. Methods Appl. Mech. Engrg.*, 163:231–245, 1998.

[40] C. Degand and C. Farhat. A three-dimensional torsional spring analogy method for unstructured dynamic meshes. *Comput. Struct.*, 80:305–316, 2002.

[41] F.J. Blom. Considerations on the spring analogy. *Int. J. Numer. Meth. Fluids*, 32:647–668, 2000.

[42] V. Selmin and L. Formaggia. Simulation of hypersonic flows on unstructured grids. *Int. J. Numer. Meth. Engng.*, 34:569–606, 1992.

[43] M.J. Castro-Díaz, F. Hecht, B. Mohammadi, and O. Pironneau. Anisotropic unstructured mesh adaption for flow simulations. *Int. J. Numer. Meth. Fluids*, 25:475–491, 1997.

[44] J. Peraire and K. Morgan. Unstructured mesh generation including directional refinement for aerodynamic flow simulation. *Finite Elements in Analysis and Design*, 25:343–356, 1997.

[45] W.G. Habashi, J. Dompierre, Y. Bourgault, D. Ait-Ali-Yahia, M. Fortin, and M.G. Vallet. Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. Part I: general principles. *Int. J. Numer. Meth. Fluids*, 32:725–744, 2000.

[46] D. Ait-Ali-Yahia, G. Baruzzi, W.G. Habashi, M. Fortin, J. Dompierre, and M.G. Vallet. Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. Part II. Structured grids. *Int. J. Numer. Meth. Fluids*, 39:657–673, 2002.

[47] 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. *Int. J. Numer. Meth. Fluids*, 39:675–702, 2002.

[48] A. Tam, D. Ait-Ali-Yahia, M.P. Robichaud, M. Moore, V. Kozel, and W.G. Habashi. Anisotropic mesh adaptation for 3D flows on structured and unstructured grids. *Comput. Methods Appl. Mech. Engrg.*, 189:1205–1230, 2000.

[49] K.G. Siebert. New anisotropic a priori error estimates. *Numerische Mathematik*, 96(3):373–398, 1996.

[50] T. Apel. Anisotropic finite elements: Local estimates and applications. Advances in Numerical Mathematics. Teubner, Stuttgart. Habilitationsschrift., 1999.

[51] L. Formaggia and S. Perotto. New anisotropic a priori error estimates. *Numerische Mathematik*, 89(4):641–67, 2001.

[52] L. Formaggia, S. Micheletti, and S. Perotto. Anisotropic mesh adaptation in computational fluid dynamics: application to the advection-diffusion-reaction and the stokes problems. Technical Report MOX15, MOX-Politecnico di Milano, 2003.

[53] S. Micheletti, S. Perotto, and M. Picasso. Stabilized finite elements on anisotropic meshes: a priori error estimates for the advection-diffusion and stokes problems. Technical Report MOX2, MOX-Politecnico di Milano, 2002.

[54] S. Micheletti and S. Perotto. Anisotropic mesh adaptivity in CFD. Technical Report MOX29, MOX-Politecnico di Milano, 2003.

[55] P.J. Frey and F. Alauzet. Anisotropic mesh adaptation for CFD computations. *Comput. Methods Appl. Mech. Engrg.*, 194:5068–5082, 2005.

[56] M. Picasso. Adaptive finite elements with large aspect ratio based on an anisotropic error estimator involving first order derivatives. *Comput. Methods Appl. Mech. Engrg.*, 196:14–23, 2006.

[57] Y. Bourgault, M. Picasso, F. Alauzet, and A. Loseille. On the use of anisotropic *a posteriori* error estimators for the adaptative solution of 3D inviscid compressible flows. *accepted for publication in Int. J. Numer. Meth. Fluids*, 2008.

[58] P. Persson, M.J. Aftosmis, and R. Haimes. On the use of loop subdivision surfaces for surrogate geometry. In *Proc. 15th int. meshing roundtable*, pages 375–392, 2006.

[59] D. Zorin and P. Schröder. Subdivision for modeling and animation: Chapters 1-4. In *ACM SIGGRAPH 2000 Conference Course Notes #23*, pages 13–92, 2000.

[60] D. Zorin, P. Schröder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *Computer Graphics Proceedings (SIGGRAPH 96)*, pages 189–192, 1996.

[61] P. Krysl and M. Ortiz. Extraction of boundary representation from surface triangulations. *Int. J. Numer. Meth. Engng.*, 50:1737–1758, 2001.

[62] S.J. Owen and D.R. White. Mesh-based geometry: a systematic approach to constructing geometry from a finite element mesh. In *Proc. 10th int. meshing roundtable*, pages 83–96, 2001.

[63] J. Peiró, J. Peraire, and K. Morgan. *FELISA system reference manual*. University of Wales Swansea, report c/r/821/94 edition, 1994.

[64] W.G. Habashi, C.Y. Lepage, G.S. Baruzzi, and I. Akel. Optimesh: Anisotropic mesh adaptation with CAD integrity for verifiably accurate CFD solutions over complete aircraft. NATO Research and Technology Organisation, RTO-MP-089, April 2002.

[65] S. Merazzi, E.A. Gerteisen, and A.A. Mezentsev. A generic cad-mesh interface. In *IMR*, pages 361–370, 2000.

[66] SINTEF ICT, Applied Mathematics. *SISL The SINTEF Spline Library*, 4.4 edition, August 2005.

[67] T. Dokken. *Aspects of Intersection Algorithms and Approximation*. PhD thesis, University of Oslo, 1997.

[68] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric Analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Methods Appl. Mech. Engng.*, 194:4135–4195, 2005.

[69] Y. Bazilevs, L. Beir ao da Veiga, J.A. Cottrell, T.J.R. Hughes, and G. Sangalli. Isigeometric Analysis: Approximation, stability and error estimates for h-refined meshes. *Math. Models Methods Appl. Sci.*, 16(7):1031–1090, 2006.

[70] R. Löhner, J.R. Cebral, F.E. Camelli, S. Appanaboyina, J.D. Baum, E.L. Mestreau, and O.A. Soto. Adaptive embedded and immersed unstructured grid techniques. *Comput. Methods Appl. Mech. Engrg.*, 197:2173–2197, 2008.

[71] F. Perazzo, R. Löhner, and L. Perez-Pozo. Adaptive methodology for meshless finite point method. *Adv. Eng. Softw.*, 39:156–166, 2008.

[72] G. Karypis and V. Kumar. ParMETIS: Parallel graph partitioning and sparse matrix ordering library. *Technical Report 97-060, Department of Computer Science, University of Minnesota*, 1998.

[73] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput*, 20(1):359–392, 1998.

[74] C. Walshaw and M. Cross. Mesh partitioning: a multilevel balancing and refinement algorithm. *SIAM J. Sci. Comput*, 22(1):63–80, 2000.

[75] R.E. Bank and R.K. Smith. An algebraic multilevel multigraph algorithm. *SIAM J. Sci. Comput*, 23(5):1572–1592, 2002.

[76] H.L. de Cougny M. Shephard. Parallel refiniment and coarsening of tetraherdal meshes. *Comput. Methods Appl. Mech. Engng.*, 46:1101–1125, 1999.

[77] P. Leyland and R. Richter. Completely parallel compressible flow simulations using adaptive unstructured meshes. *Comput. Methods Appl. Mech. Engng.*, 184:467–483, 2000.

[78] L. Oliker, R. Biswas, and H.N. Gabow. Parallel tetrahedral mesh adaptation with dynamic load balancing. *Parallel Computing*, 266(12):1583–1608, 2000.

[79] J. Waltz. Parallel adaptive refinement for unsteady flow calculations on 3D unstructured grids. *Int. J. Numer. Meth. Fluids*, 46:37–57, 2004.

[80] Y.M. Park and O.J. Kwon. A parallel unstructured dynamic mesh adaptation algorithm for 3D unsteady flows. *Int. J. Numer. Meth. Fluids*, 48:671–690, 2005.

[81] J.A. Shaw. Hybrid grids. In J.F. Thompson, editor, *CRC Handbook of Grid Generation*. CRC Press, Boca Raton, FL, 1999.

[82] Y. Kallinderis, A. Khawaja, and H. McMorris. Hybrid prismatic/tetrahedral grid generation for viscous flows around complex geometries. *AIAA Journal*, 34:291–298, 1996.

[83] V. Parthasarathy and Y. Kallinderis. Adaptive prismatic-tetrahedral grid refinement and redistribution for viscous flows. *AIAA Journal*, 34:707–716, 1996.

[84] Y. Kallinderis. A 3-D finite-volume method for the Navier-Stokes equations with adaptive hybrid grids. *Appl. Numer. Math.*, 20:387–406, 1996.

[85] Y. Kallinderis. Prismatic/tetrahedral grid generation for complex geometries. In *Computational Fluid Dynamics, Lecture Series 1996-06*. von Karman Institute for Fluid Dynamics, Rhode Saint Genése, Belgium, 1996.

[86] A.J. Chen and Y. Kallinderis. Adaptive hybrid (prismatic-tetrahedral) grids for incompressible flows. *Int. J. Numer. Methods Fluids*, 26:1085–1105, 1998.

[87] Y. Kallinderis. Hybrid grids and their applications. In J.F. Thompson, editor, *RC Handbook of Grid Generation*. CRC Press, Boca Raton, FL, 1999.

[88] A. Khawaja, T. Minyard, and Y. Kallinderis. Adaptive hybrid grid methods. *Comput. Methods Appl. Mech. Engrg.*, 189:1231–1245, 2000.

[89] T. Minyard and Y. Kallinderis. Parallel load balancing for dynamic execution environments. *Comput. Methods Appl. Mech. Engrg.*, 189:1295–1309, 2000.

[90] Y. Kallinderis and C. Kavouklis. A dynamic adaptation scheme for general 3-D hybrid meshes. *Comput. Methods Appl. Mech. Engng.*, 194:5019–5050, 2005.

[91] D.J. Mavriplis. Adaptive meshing techniques for viscous flow calculations on mixed element unstructured meshes. *Int. J. Numer. Meth. Fluids*, 34:93–111, 2000.

[92] M.B. Giles and E. Süli. Adjoint methods for PDE's: *a posteriori* error analysis and postprocessing by duality. *Acta Numerica*, pages 145–326, 2002.

[93] R. Becker and R. Rannacher. A feed-back approach to error control in finite element methods: basic analysis and examples. *East-West J. Numer. Math.*, 4:237–264, 1996.

[94] R. Becker and R. Rannacher. An optimal control approach to a posteriori error estimation in finite element methods. *Acta Numerica*, 10:1–102, 2001.

[95] M.B. Giles. On adjoint equations for error analysis and optimal grid adaptation in CFD. Technical Report NA97/11, Oxford University Computing Laboratory, 1997.

[96] M.B. Giles. On the use of runge-kutta time-marching and multi-grid for the solution of steady adjoint equations. Technical Report NA00/10, Oxford University Computing Laboratory, 2000.

[97] M.B. Giles and N.A. Pierce. Adjoint equations in CFD: duality, boundary conditions and solution behaviour. AIAA Paper 97-1850, 1997.

[98] M.B. Giles and N.A. Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion*, 65:393–415, 2000.

[99] M.B. Giles and N.A. Pierce. Improved lift and drag estimates using adjoint euler equations. Technical Report NA00/02, Oxford University Computing Laboratory, 2000.

[100] M.B. Giles, M.G. Larson, J.M. Levenstam, and E. Süli. Adaptive error control for the finite element approximations of the lift and drag coefficients in viscous flow. Technical Report NA97/06, Oxford University Computing Laboratory, 1997.

[101] D.A. Venditti and D.L. Darmofal. A multilevel error estimation and adaptive strategy for improving the accuracy of integral outputs. AIAA Paper 99-3292, 1999.

[102] D.A. Venditti and D.L. Darmofal. A grid adaptive methodology for functionl outputs of compressible flow simulations. AIAA Paper 2001-2659, 2001.

[103] D.L. Darmofal and D.A. Venditti. Output-based error estimation and adaptation for aerodynamics. WCCM V, Vienna, Austria, July 2002.

[104] D.A. Venditti and D.L. Darmofal. Grid adaptation for functional outputs: application to two-dimensional inviscid flows. *J. Comput. Phys.*, 176:40–69, 2002.

[105] D.A. Venditti and D.L. Darmofal. Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows. *J. Comput. Phys.*, 187:22–46, 2003.

[106] J.D. Müller and M.B. Giles. Solution adaptive mesh refinement using adjoint error analysis. AIAA Paper 2001-2550, 2001.

[107] M.A. Park. Adjoint-based, three-dimensional error prediction and grid adaptation. AIAA Paper 2002-3286, 2002.

[108] L. Formaggia and S. Perotto. Error estimation for finite element methods. In *31th Computational Fluid Dynamics Lecture Series*, 2000.

[109] A. Casagrande, P. Leyland, L. Formaggia, and M. Sala. Parallel mesh adaptation. In M. Primicero, R. Spigler, and V. Valente, editors, *Applied and Industrial Mathematics in Italy*, volume 69 of *Series on Advances in Mathematics for Applied Sciences*, pages 201–212. World Scientific, September 2004.

[110] G. Warren, W.K. Anderson, J.L. Thomas, and S.L. Krist. Grid convergence for adaptive methods. AIAA Paper 91-1592, June 1991.

[111] P. Leyland, F. Benkhaldoun, N. Maman, and B. Larrouturou. Dynamical mesh adaptation criteria for accurate capturing of stiff phenomena in combustion. Technical Report N^o1876, INRIA, Avril 1993.

[112] Y. Savoy and P. Leyland. Adaptive module: Technical report task 5.1. Technical Report TR5.1, IDeMAS, 2000.

[113] R. Richter and P. Leyland. Distributed CFD using auto-adaptive finite elements. ICASE/LaRC Workshop on Adaptive Grid Methods, November 1994.

[114] M. Sala and P. Leyland. A parallel adaptive Newton-Krylov-Schwarz method for the 3D compressible Euler equations. Technical report, IDeMAS, 2002.

[115] G. Cartes and K. Sermeus. Thor user's guide. Technical Report V.1.0, IDeMAS, 1999.

[116] G. Farin. *Curves and surfaces for computer aided geometric design, a practical guide.* Computer Science and Scientific Computing. Academic Press, fourth edition, 1997.

[117] G. Farin. *NURBS, from Projective Geometry to Practical Use.* A. K. Peters, Ltd, second edition, 1999.

[118] G. Farin, J. Hoschek, and M.-S.Kim, editors. *Handbook of computer aided geometric design*. Elsevier Science B.V., first edition, 2002.

[119] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Technical Report 04-51, ICES, October 2004.

[120] I. J. Anderson, M. G. Cox, A. B. Forbes, J. C. Mason, and D. A. Turner. An efficient and robust algorithm for solving the foot point problem. In *Proceedings of the international conference on Mathematical methods for curves and surfaces II Lillehammer, 1997*, pages 9–16, Nashville, TN, USA, 1998. Vanderbilt University.

[121] M. Aigner and B. Jüttler. Robust computation of foot points on implicitly defined curves. In *Mathematical methods for curves and surfaces: Tromsø2004*, Mod. Methods Math., pages 1–10, Brentwood, TN, 2005. Nashboro Press.

[122] E. Hartmann. On the curvature of curves and surfaces defined by normalforms. *Computer Aided Geometric Design*, 16, 1999.

[123] N. Dyn, J.A. Gregory, and D. Levin. A four-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, 4:257–268, 1987.

[124] N. Dyn, D. Levin, and J.A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. Gr.9*, 2:160–169, April 1990.

[125] O.C. Zienkiewicz and J.Z. Zhu. The superconvergent patch recovery (spr) and adaptive finite element refinement. *Comput. Methods Appl. Mech. Engrg.*, 101:207–224, 1992.

[126] D.J. Mavriplis. Revisiting the least-square procedure for gradient reconstruction on unstructured meshes. NASA/CR Report, 2003-06, 2003.

[127] M. Ainsworth and J.T. Oden. *A Posteriori Error Estimation in Finite Element Analysis*. John Wiley & Sons, New York, 2000.

[128] C. Carstensen. All first-order averaging techniques for a posteriori finite element error control on unstructured grids are efficient and reliable. *Math. Comp.*, 73:1153–1165, 2004.

[129] C. Carstensen. Some remarks on the history and future of averaging techniques in a posteriori finite element error analysis. *ZAMM Z. Angew. Math. Mech.*, 84:3–21, 2004.

[130] C. Carstensen and S. Bartels. Each averaging technique yields reliable a posteriori error control in FEM on unstructured grids. I: Low order conforming, noncomforming, and mixed FEM. *Math. Comp.*, 71:945–969, 2001.

[131] S. Micheletti and S. Perotto. Reliability and efficiency of an anisotropic Zienkiewicz-Zhu error estimator. Technical Report MOX37, MOX-Politecnico di Milano, 2004.

[132] M. Picasso. Numerical study of the effectivity index for an anisotropic error indicator based on Zienkiewicz-Zhu error estimator. *Comm. Numer. Methods Engrg.*, 19:13–23, 2003.

[133] N. Yan and A. Zhou. Gradient recovery type a posteriori error estimation for finite element approximations on irregular meshes. *Comput. Methods Appl. Mech. Engrg.*, 190:4289–4299, 2001.

[134] G. Geymonat and P. Leyland. Transport and propagation of a perturbation of a flow of a compressible fluid in a bounded region. *Archive for Rational Mechanics and Analysis*, 100(1):53–81, 1987.

[135] E. Feireisl. *Dynamics of viscous compressible fluids*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford, 2003.

[136] A. Brooks and T.J.R. Hughes. Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Comput. Methods Appl. Mech. Engrg.*, 32(1-3):199–259, September 1982.

[137] T.J.R. Hughes and M. Mallet. A new finite element formulation for computational fluid dynamics: III. the generalized streamline operator for multidimensional advective-diffusive systems. *Comput. Methods Appl. Mech. Engrg.*, 58(3):305–328, November 1986.

[138] H. Deconinck, C. Hirsch, and J. Peuteman. Characteristics decomposition methods for the multidimensional Euler equations. In *10th Int. Conf. in Num. Meth. in Fluid Dyn.*, volume 264 of *Lecture Notes in Physics*, pages 216–221, Beijing, China, June 1986. Springer Berlin / Heidelberg.

[139] C.L. Bottasso, G. Maisano, S. Micheletti, and S. Perotto. New recovery based a posteriori error estimators. Technical Report MOX52, MOX-Politecnico di Milano, 2004.

[140] W. Wessner. *Mesh Refinement Techniques for TCAD Tools*. PhD thesis, Technischen Universität Wien, 2006.

[141] M. Sala. *Domain Decomposition Preconditioners: Theoretical Properties, Application to the Compressible Euler Equations, Parallel Aspects*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2003.

[142] B.F. Smith, P. Bjorstad, and W.D. Grop. *Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, Cambridge, 1996.

[143] M. Sala and L. Formaggia. Parallel Schur and Schwarz based preconditioners and agglomeration coarse corrections for CFD problems. Technical report, DMA-EPFL, 2001.

[144] M. A. Weiss. *Data Structure and Algorithm Analysis*. The Benjamin Cummings Publishing Company, Inc., 1992.

[145] R.S. Tuminaro, M. Heroux, S.A. Hutchinson, and J.N. Shadid. *Official Aztec User's Guide Version 2.1*. Sandia National Laboratories, Albuquerque, NM, November 1999.

[146] J. Müller and T. Schönfeld. A comparison of the treatment of hanging nodes for hybrid grid refinement. AIAA Paper 97-1859, 1997.

[147] D. Sharov and K. Nakahashi. Hybrid prismatic/tetrahedral grid generation for viscous flow applications. *AIAA Journal*, 36:157–162, 1998.

[148] A. Athanasiadis and H. Deconinck. An improved strategy for semi-structured layer generation in hybrid. In *Paper T1-I-57-0967, Applied Mathematics and Simulation*. 17^{th} IMACS World Congress in Scientific Computation, Paris, France, July 2005.

[149] J.F. Thompson, B.K. Soni, and N.P. Weatherhill, editors. *Handbook of grid generation*. CRC Press, 1999.

Curriculum Vitæ

Angelo Casagrande

Personal Information:

Date of birth:	18$^{\text{th}}$ January 1975.
Place of birth:	Milano, Italy.
Nationality:	Italian

Education:

2003-2008:	PhD student at LIN (Laboratoire d'ingénierie numérique) at EPFL.
1999-2001:	M.Sc. Maritime Engineering Science (Maritime CFD & Applied Mechanics) at University of Southampton (UK).
1996-1999:	B.Eng. (Hons) Ship Science at University of Southampton (UK).

Teaching Experience:

2005:	Assistant for the course "Introduction à la mécanique des milieux continus", Prof. M. Deville, LIN/EPFL.

Publications:

Parallel Mesh Adaptation, A. Casagrande, P. Leyland, L. Formaggia and M.Sala, In *APPLIED AND INDUSTRIAL MATHEMATICS IN ITALY Proceedings of the 7th Conference*, 2004, ed: M. Primicerio, R. Spigler and V. Valente; World Scientific.

Parallel Mesh Adaptive Techniques for Complex Simulation, A. Casagrande, P. Leyland and L. Formaggia, In *EUA4X@IAC 06 Proceedings*, 2006, ed: F. Pistella and R.M. Spitaleri; IMACS.