ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
A MASTER'S DEGREE WITH THESIS IN AEROSPACE ENGINEERING
M.A.Sc.

BY
Jonathan LANDRY

# ROBUST MOVING-MESH ALGORITHMS FOR HYBRID STRETCHED MESHES: APPLICATION TO MOVING BOUNDARIES PROBLEMS

MONTREAL, JULY 10, 2015

**BOARD OF EXAMINERS**

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS:

M. Azzeddine Soulaïmani, Thesis Supervisor
Département de Génie Mécanique, École de technologie supérieure

M. Amine Ben Haj Ali, Thesis Co-supervisor
Advanced Aerodynamics, Bombardier Aerospace

M. François Morency, Chair, Board of Examiners
Département de Génie Mécanique, École de technologie supérieure

M. Kurt Sermeus, Aerodynamicist
Advanced Aerodynamics, Bombardier

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON JUNE 5, 2015

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

## ACKNOWLEDGEMENTS

I would like to express my gratitude:

- To my thesis supervisor Azzeddine Soulaïmani Ph.D., Eng. Professor Soulaïmani has given me the chance to explore a complex subject with freedom and trust. I have also gained a lot from his mentoring, reputation, experience and our long discussions for which I am truly grateful;
- To Amine Ben Haj Ali Ph.D. for his help on defining the test cases and his experiences in meshing generation;
- To Professor Edward Luke Ph.D. who shared with us his mesh-mover algorithm which was crucial to generate great results and study deeper the subject of moving boundaries problems;
- To Aziz Madrane Ph.D. for his recommendation on the use of the ADT search algorithm.

Also, a special thanks to the École de technologie supérieure for entrusting me the excellence scholarship which allowed me to dedicate my energy completely to this research with the support of this growing institution.

I dedicated this work to my dear fiancée Alicia who allowed me to push myself every day and to produce proudly this thesis thanks to her love and her motivation; à ma mère Colette, mon père Georges, mon frère Nicholas et ma soeur Maude pour leur soutien inconditionnel; to all my friends who have help me grow; and to Botas and Rizpita whom entertain me during my redaction.

**MÉTHODES ROBUSTES DE MOUVEMENT DE MAILLAGES HYBRIDES: APPLICATION AUX PROBLÈMES DE FRONTIÈRES MOBILES**

Jonathan LANDRY

## RÉSUMÉ

L'objectif principal de ce mémoire est de développer un Algorithme de Mouvement de Maillage (AMM) suffisament robuste qui sera capable d'adapter les maillages de la majorité des problèmes de frontières mobiles. Ainsi, une nouvelle méthodologie est implémentée à partir de la meilleure combinaison provenant d'algorithmes connus dans le but de mieux préserver la qualité des maillages initiaux.

Dans la majorité des simulations, les AMMs standards sont capables de déplacer le maillage en conservant une bonne qualité de maillage, toutefois quand le mouvement est complexe et/ou qu'il y a une interaction multi-corps ils génèrent des maillages invalides. Alors, la nouvelle approche est constituée de trois algorithmes: la fonction pondérée de la distance inverse (PDI) pour produire le champ de déplacements, les algorithmes de lissages de la Méthode de Transformation Géométrique d'Élément (MéTGÉ) pour améliorer la qualité du maillage résultant et un nouvel algorithme basé sur la MéTGÉ qui est capable de réparer les maillages invalides. Il a été prouvé qu'avec cette méthodologie des problèmes de frontières mobiles très difficiles peuvent être résolus.

L'approche proposée a été démontrée efficace pour adapter les maillages de différentes situations aéroélastiques réalistes: une aile symétrique a subi une grande flexion et une grande torsion induites au bout de l'aile; et les dispositifs hypersustentateurs de deux ailes (une rectangulaire et une avec une flèche) ont été déplacés vers différentes positions de vol.

Finalement, il a été prouvé qu'avec la méthode proposée, la majorité des maillages pour les problèmes d'IFS peuvent être adaptés. Toutefois, pour les situations où les surfaces mobiles sont très proches, des améliorations devront être appliquées ou une toute autre direction de résolution devrait être adopté tel que la méthode Chimera.

**Mots-clés:** mouvement de maillage, éléments finis, frontières mobiles, PDI, MéTGÉ, lissage, algorithmes

# ROBUST MOVING-MESH ALGORITHMS FOR HYBRID STRETCHED MESHES: APPLICATION TO MOVING BOUNDARIES PROBLEMS

Jonathan LANDRY

## ABSTRACT

A robust Mesh-Mover Algorithm (MMA) approach is designed to adapt meshes of moving boundaries problems. A new methodology is developed from the best combination of well-known algorithms in order to preserve the quality of initial meshes.

In most situations, known MMAs distribute mesh deformation while preserving a good mesh quality. However, invalid meshes are generated when the motion is complex and/or involves multiple bodies. After studying few MMAs limitations, we propose the following approach: use the Inverse Distance Weighting (IDW) function to produce the displacements field, then the Geometric Element Transformation Method (GETMe) smoothing algorithms to improve the resulting mesh quality and use an untangler to revert negative elements.

The proposed approach has been proven efficient to adapt meshes for various realistic aerodynamic motions: a symmetric wing has suffered large tip bending and twisting and the high-lift components of two wings (one rectangular and one swept) have moved to different flight stages.

Finally, the fluid flow has been solved on adapted meshes and their results are close to experimental ones. However, for situations where moving boundaries are close to contact more improvements need to be made or other approaches should be considered such as the overset grid method.

**Keywords:** moving mesh, finite element, moving boundaries, GETMe, smoothing, untangler, algorithms

## TABLE OF CONTENTS

Page

## LIST OF TABLES

## LIST OF FIGURES

Page

## LIST OF ABREVIATIONS

| | |
|---|---|
| ADT | Alternating Digital Tree |
| AIAA | American Institute of Aeronautics and Astronautics |
| ALE | Arbitrary Lagrangian Euleurian |
| BiCGStab | Bi-Conjugate Gradient Stabilized |
| CFD | Computer Fluid Dynamics |
| DLR | Deutsches zentrum für Luft- und Raumfahrt or German Aerospace Centre |
| FEM | Finite element method |
| FSI | Fluid-Structure Interaction |
| GETMe | Geometric Element Transformation Method |
| GMRES | Generalized Minimal Residual method |
| GUI | Graphic User Interface |
| HiLift-PW | High-Lift Prediction Workshop |
| IDW | Inverse Distance Weighting |
| MMA | Mesh-Mover Algorithm |
| MMARS | Mesh-Mover Algorithms Robust Strategy |
| MPI | Message Passing Interface or Multi-Processors Interface |
| MSA | Moving Submesh Approach |
| NASA | National Aeronautics and Space Administration (United States' space agency) |
| NGU | Novel GETMe Untangler |

| | |
|---|---|
| PDE | Partial Differential Equation |
| PSM | Pseudo-Structural Method |

## LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

| | |
|---|---|
| $\nabla^2$ | Laplacian operator |
| $\nabla$ | Gradient operator |
| $T$ | Temperature field |
| $f$ | Source term |
| $\boldsymbol{\sigma}$ | Cauchy stress |
| $\mathbf{F}$ | Vector of external body forces |
| $\mathbf{C}$ | Fourth order stiffness tensor |
| $\boldsymbol{\varepsilon}$ | Strain vector |
| $\mathbf{u}$ | Displacement vector |
| $\mathbf{w}$ | Mesh velocity vector |
| $\mathbf{x}$ | Position vector |
| $[K]$ | Stiffness matrix |
| $t$ | time |
| $\Omega$ | Domain of calculation |
| $\mathbf{w}_m$ | Prescribed mesh velocity vector |
| $\Gamma_m$ | Moving boundaries |
| $\Gamma_f$ | Fixed boundaries |
| $\mathbf{g}$ | Prescribed displacement vector |
| $\tau_m$ | Non-dimensional function of mesh size |

| | |
|---|---|
| $\{U\}$ | Nodes displacements vector |
| $\{F\}$ | Vector of external forces |
| $[B^e]$ | First derivative of the element shape function in the real coordinate system |
| $[D]$ | Matrix of pseudo-material properties |
| $[J]$ | Jacobian matrix |
| $\Omega^e$ | Element's domain |
| $\Omega^r$ | Reference element's domain |
| $E$ | Pseudo Young's modulus |
| $\nu$ | Pseudo Poisson's ratio |
| $p$ | Inverse volume stiffening exponent |
| $pq$ | Inverse quality stiffening exponent |
| $i,\ j,\ k,\ l$ | Indices |
| $k_{ij}$ | Linear spring stiffness of edge $ij$ |
| $L_{ij}$ | Length of edge $ij$ |
| $C_i^{ijk}$ | Torsional spring stiffness of vertex $i$ in triangle $ijk$ |
| $A_{ijk}$ | Area of the triangle $ijk$ |
| $s(\mathbf{x})$ | Function of the approximate displacements field |
| $\mathbf{x}_b$ | Position vector of a boundary node |
| $\mathbf{g}_j$ | Prescribed displacement vector of a boundary node |
| $P(\mathbf{x})$ | An arbitrary polynomial function |

| | |
|---|---|
| $C0$, $C1$, $C2$, $C3$ | Constants of an arbitrary polynomial function |
| $m$ | The quantity of elements for summation |
| $\phi$ | A radial basis function |
| $\lambda_j$ | Coefficients for the radial basis function |
| $\|\|...\|\|$ | The Euclidean norm |
| $a$, $b$, $c$ | Exponents for the inverse distance weighting function |
| $\beta$ | User defined for exponentials: $exp^{\beta...}$ |
| $M_\infty$ | Mach number for the far-field |
| $\mathbf{x}_n$ | Position vector of a neighbouring node |
| $qtN$ | Quantity of nodes in the mesh |
| $V^e$ | The volume of an element |
| $w_i(\mathbf{x})$ | Inverse distance weighting function |
| $A_i$ | Average area of the boundary faces around the node $i$ |
| $L_{ref}$ | Radius of influence for the inverse distance weighting function |
| $\alpha$ | Factor of near body influence |
| $\mathbf{x}_{fine}$ | Position vector of a fine mesh node |
| $N_i$ | Interpolation functions |
| $x$, $y$, $z$ | Real frame coordinates |
| $\xi$, $\eta$, $\zeta$ | Reference frame coordinates |
| $\boldsymbol{\varsigma}$ | Position vector of a fine mesh node in the reference frame |

| | |
|---|---|
| $\mathbf{R}$ | Residual matrix |
| *coord* | Matrix containing all coordinates of an element |
| *nnel* | Quantity of node of an element |
| $\mathbf{n}_k$ | Real element coordinates vector |
| $\mathbf{F}$ | Real element face matrix connectivities |
| $\mathbf{d}_k$ | Dual element coordinates vector |
| $\overline{\mathbf{F}}$ | Dual element face matrix connectivities |
| $\mathbf{n}'_k$ | Real element new coordinates vector |
| $\mathbf{nv}_k$ | Normal vector of a dual element's face |
| $\sigma$ | The geometric transformation scaling factor |
| $\mathbf{a}_k$ | Specific base points for triangular faces |
| $\mathbf{b}_k$ | General base points for faces |
| $\mathbf{c}_k$ | Centroid base points for faces |
| $q$ | Quality metric |
| **Nt** | Nodal tetrahedron matrix connectivities |
| **Diff** | Difference matrix |
| $\mathbf{W}$ | Element target matrix |
| $J$ | Index set of all elements |
| $I$ | Index set of all nodes |
| $\psi$ | Mean edge length of an element |

| | |
|---|---|
| $\gamma_k$ | Vector of relaxation values |
| $q_{chg}$ | Metric of quality change |
| $q_{chgAbs}$ | Metric of absolute quality change |
| $q_{pre}$ | Metric of preserved quality |
| $t$ | Simulation time ($t \in [0..1]$) |
| $MAC$ | Mean aerodynamic chord |
| $Re$ | Reynolds number |
| $P_\infty,\ \nu_\infty, T_\infty$ | Pressure, dynamic viscosity and temperature at the far field |
| $U_\infty$ | Velocity at the far field |
| $C_p$ | Pressure coefficient |
| $C_D$ | Drag coefficient |
| $C_L$ | Lift coefficient |

## LIST OF ALGORITHMS

# INTRODUCTION

## 0.1. Preface

Computational methods are very often used in design and analysis phases of engineering products. Numerical methods consist in finding a solution to Partial Differential Equations (PDEs) which define a physical process. The Finite Element Method (FEM) is a popular numerical method used for solving problems in engineering and science. The FEM solves the PDEs of a physical domain divided in elements. This physical decomposition is called a mesh and the mesh quality has a big effect on the numerical solution. A major field of interest for numericians nowadays concerns moving boundary problems, particularly Fluid Structure Interaction (FSI) problems and in the studies of stability and control of moving aerodynamic surfaces (such as ailerons).

Usually, in fluid mechanics the mesh is fixed (Eulerian description); however in structural mechanics the mesh follows the material (Lagrangian description). To solve FSI problems, the Arbitrary Lagrangian Eulerian (ALE) formulation is used to define the position of the mesh nodes: the shared interfaces between the fluid and structural domains are moved in Lagrangian manner and fluid nodes are moved progressively from Lagrangian to Eulerian as their distance to the interfaces becomes greater.

The ALE is a combination of the Lagrangian and Eulerian descriptions, thus it has the ability to track interfaces easily and the capability to handle large particle material motions (Donea *et al.* (1982)). The mesh moves according to the ALE description, however this approach alone can lead to a distorted mesh invalid for numerical analysis. Thus, many tools have been developed to assist ALE methods in order to move the mesh and reduce its distortion. This work will focus on the study of Mesh-Mover Algorithms (MMAs) which distribute the displacements from moving or fixed boundaries to the whole domain.

## 0.2. Objective

In the study of the aeroelastic behaviour of wings, the mesh is moved according to wing surfaces' motions. The mesh has to maintain a good level of quality during its motion and avoid generating highly distorted or tangled elements. Classical algorithms to move the mesh are based on spring (Robinson *et al.* (1991)) or pseudo-structural (Tezduyar *et al.* (1992)) analogies. However, those analogies do not guarantee a good mesh quality, especially for fine meshes in the viscous boundary layers. In this work, algorithms are developed to enhance the performance and robustness of mesh-movers for hybrid meshes composed of tetrahedron, pyramids, triangular prisms and hexahedron.

The specific objectives of this thesis are to:

- Develop a code which can be easily improved and integrated with existing FEM codes. It should be able to read mixed meshes generated from commercial software such as Pointwise®;
- Implement and improve MMAs which are the core of the code;
- Test the limits of each MMAs with simple geometry simulations;
- Use the MMAs to distribute displacements of static aeroelastic situations while keeping the mesh valid;
- Validate that fluid solvers can produce valid results with deformed meshes.

## 0.3. Plan of the thesis

First, a review of current developed MMAs will be presented in Chapter 1, followed by detailed descriptions of those chosen to be implemented, in Chapter 2. Then, the structure and methodology of the code used to solve the displacements will be explained, in Chapter 3. In Chapter 4, the MMAs parameters' influence on mesh-motion will be analysed, as well as the usability of MMAs to distribute large distortions and the workability of MMAs for multi-body interaction

problems. Finally in Chapter 5, various meshes surrounding aerodynamic geometries will be moved according to the moving boundaries displacements:

- A symmetric NACA0012 wing will be bent and twisted similarly as in flight situation;
- High-lift components of two wings will move through all the positions encountered in full flight: take-off, cruise then landing. The first wing is a section extrusion of the DLR-F6 wing and the second is the trapezoidal wing configuration which is composed of a swept wing and its half fuselage. Those geometries are taken from the first and second American Institute of Aeronautics and Astronautics (AIAA) High-Lift Prediction Workshops (HiLift-PW) of Rumsey (2014);
- The flow field for the trap. wing deformed mesh and undeformed mesh will be computed to assure that complex mesh deformations do not affect fluid flow results.

# CHAPTER 1

# LITERATURE REVIEW

MMAs have been used in a large number of different applications from character animation (de Aguiar and Ukita (2013)), to electromagnetic simulation (Miwa *et al.* (2011)) and all kinds of Computer Fluid Dynamics (CFD) problems. They all use similar algorithms which are divided in two families:

- PDEs based methods;
- Interpolations from moving boundaries to internal domain nodes.

PDEs methods solve the system of equations of the element to obtain the mesh displacements, thus these methods consume more computer resources than the interpolation schemes which calculate algebraically the nodes displacements.

The literature review will cover:

- The presentation of major trends of both families in section 1.2 to 1.5;
- Two approaches for overlapping moving and computational meshes;
- Smoothing approaches which improve mesh quality.

This review will help in choosing which algorithms, or combinations, are sufficiently robust to allow most mesh motion while preserving a good mesh quality.

## 1.1 Definitions

A robust mesh-mover algorithm is defined as an MMA which distributes moving boundaries displacements in a mesh by generating no (or rarely) elements with negative volume.

Then, a smooth MMA is defined as an algorithm which distributes displacements of moving boundaries on a mesh evenly through all elements making the average of element's deformation small.

In terms of calculated quality values (see Sections 2.4 and 2.5) it means that a smooth MMA has a high average quality and a robust MMA has high minimal quality which is always positive.

Finally, the performance of an MMA is defined as the combination of the robustness, smoothness and computational time to adapt a mesh.

## 1.2 Partial differential equations based methods

The PDEs based MMAs are an adaptation of the linear elasticity and steady state heat diffusion equations which are:

1. Steady State diffusion

$$\nabla^2 T + f = 0, \tag{1.1}$$

where $\nabla^2$ is the second order spatial derivative operator called Laplacian, $T$ is the temperature field and $f$ is a source term.

2. Linear elasticity

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{F} = 0, \tag{1.2}$$
$$\textit{with } \boldsymbol{\sigma} = \mathbf{C} : \boldsymbol{\varepsilon} \textit{ and } \boldsymbol{\varepsilon} = \frac{1}{2}((\nabla \mathbf{u}) + (\nabla \mathbf{u})^T),$$

where $\nabla$ is the gradient operator, $\boldsymbol{\sigma}$ is the Cauchy stress tensor, $\mathbf{F}$ is the vector of external body forces, $\mathbf{C}$ is the fourth order stiffness tensor, $\boldsymbol{\varepsilon}$ is the strain vector and $\mathbf{u}$ is the displacement vector.

### 1.2.1 Laplacian review

From the author's knowledge, the first PDE based MMA has been used in Soulaïmani (1987) and Soulaïmani *et al.* (1991) where it had been applied to free surface flow problems. The method used to update the mesh consists in solving a Poisson's equation for the grid velocity ($\mathbf{w}$) which is distributed from the free surface:

$$\begin{aligned} \pi(\mathbf{w}(\mathbf{x},t)) = \boldsymbol{\nabla} \cdot ([K]\boldsymbol{\nabla}\mathbf{w}(\mathbf{x},t)) = 0 \ \text{ in } \Omega, \\ \mathbf{w}(\mathbf{x},t) = \mathbf{w}_{\mathrm{m}} \ \text{ on } \Gamma_m, \\ \mathbf{w}(\mathbf{x},t) = 0 \ \text{ on } \Gamma_f, \end{aligned} \tag{1.3}$$

where $\mathbf{x}$ is the position vector, $[K]$ is the stiffness matrix, $t$ is the time, $\mathbf{w}_m$ is the velocity at the free surface, $\Gamma_m$ is the moving boundary and $\Gamma_f$ is the fixed boundary. This method was used with a Navier-Stokes equations solver in an ALE description and with a finite element discretization to solve free surface flow problems. The mesh moves solely vertically ($\mathbf{w} = (0,0,w_z)$), the free surface movements represent waves which cannot break and compared to subsequent approaches it solves the Laplacian of the mesh velocity instead of element displacements.

The Laplacian, or diffusion, based method (Equation 1.1) was first proposed as a mesh generation algorithm by Winslow (1963) where nodes and elements are inserted according to scalar or vector fields (for example temperature, pressure or velocity). This basic method refines the mesh where large gradients are encountered in the computed field, thus allowing better results. Although, if far-field boundaries are not enclosing totally the computational domain there is a risk of elements being generated outside of the domain. Afterwards, this approach has evolved and has become an adaptive mesh refinement technique based on the advancing front as used by Löhner (1988).

One of the first applications of the Equipotential technique (Winslow (1963) and Winslow (1981)) to distribute mesh velocity from moving boundaries has been done by Benson (1989). Then the method has been improved in Löhner and Yang (1996) by defining the diffusivity

proportional to the distance from moving boundaries. This method diminishes mesh distortion as well as improves quality of elements close to moving boundaries. This approach has shown good results and has been implemented in the open-source software OpenFOAM®(Jasak and Rusche (2009)).

Then, one of the last improvements, attempted by Helenbrook (2003), was to solve the fourth order derivative PDE for the displacement field: $\nabla^4 \mathbf{u} = 0$. This modification allows less deformation in boundary layers and gave similar results than the linear spring method (Robinson *et al.* (1991)), but with an increase in computational time.

An interesting version has been introduced in Masud and Hughes (1997), where the Laplacian equation has been formulated in the following general form:

$$\begin{aligned} \nabla \cdot ([1+\tau_m]\nabla)\mathbf{u} &= \mathbf{0} \quad \text{in } \Omega, \\ \mathbf{u} &= \mathbf{g} \quad \text{on } \Gamma_m, \\ \mathbf{u} &= \mathbf{0} \quad \text{on } \Gamma_f, \end{aligned} \tag{1.4}$$

where $\mathbf{u}$ is the mesh displacement field, $\tau_m$ is a non-dimensional function of mesh size which prevents negative elements from being generated and $\mathbf{g}$ is the prescribed displacements. This method is similar to the diffusion equation except for the scalar term $[1+\tau_m]$ which is proportional to the inverse of each element's size.

In Masud and Hughes (1997), the method is applied to a missile launch from a rectangular cavity of a submarine. The MMA is used to move the mesh until distortions are too large, then re-meshing is done. This method is general, although not robust since it needs several re-meshings to complete the tested simulation.

### 1.2.2 Pseudo-Structural Method (PSM) review

The first to have come up with the idea of modelling a moving mesh in an ALE reference frame as a pseudo material is Schreurs *et al.* (1986). This article introduces that physical mesh and moving mesh should share connectivities (see Figure 1.1) and that the pseudo or fictitious material behaviour is similar to an isotropic linearly elastic material.
The discretization by FEM of Equation 1.2 leads to solving a linear system:

$$[K]\{U\} = \{F\} \; or,$$
$$[\sum_{e} \int_{\Omega^r} [B^e]^T [D^e][B^e] det([J^e]) d\xi d\eta d\zeta]\{U\} = \{F\}, \tag{1.5}$$

where the summation over the elements generates the stiffness matrix $[K]$, the vector $\{U\}$ is the displacements nodal vector, $\{F\}$ is the vector of external forces, $[B^e]$ is the first derivative of the shape function in the real coordinate system, $[D^e]$ is called the matrix of pseudo-material properties and $det([J^e])$ is the determinant of the mapping between the physical ($\Omega^e$) and reference ($\Omega^r$) elements. This determinant is related to the volume of the element $e$.

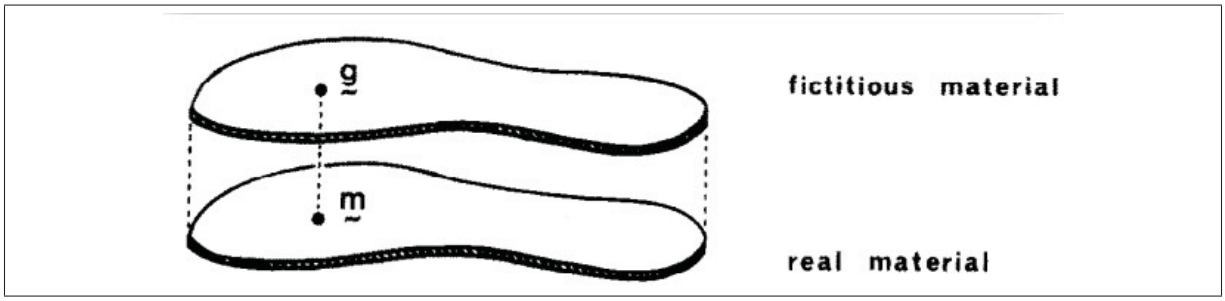



Figure 1.1 Real material and Pseudo material (Schreurs *et al.* (1986))

Schreurs *et al.* (1986) arrived at the following conclusions: the Pseudo Young's modulus $E$ has no impact on mesh deformation since all elements have the same value and the Pseudo Poisson's ratio $\nu$ does influence the distortion of the mesh for values of : $0 \leq \nu < 0.5$.

The PSM has been used by Johnson and Tezduyar (1994), as well as others, for multiple ALE problems such as:

- A viscous drop falling in a viscous fluid;
- A flow past an oscillating airfoil;
- A flow past two airfoils with one oscillating;
- A flow through a sluice gate.

It has been shown by Johnson and Tezduyar (1994) that using a mesh-update process instead of a re-meshing algorithm gives results faster and easier. Multiple FSI and two-liquid interaction simulations also have shown the generality of the method. Finally, the $E$ of each element is defined to be inversely proportional to its Jacobian determinant (or volume) which will increase the stiffness of small elements.

In Stein *et al.* (2003) an exponent has been added to their previous modified $E$ in order to increase the possible degree of stiffening. The three basic movements of translation, rotation and bending have been tested with different values of exponent. They concluded that the optimal exponent is problem dependent, but values greater or equal to one give good results.

Finally, similar test cases have been assessed with a small modification in Stein *et al.* (2004). They have increased the stiffening power of elements close to moving boundaries, since they are more at risk of being largely distorted. An increase of around 90% in mesh quality has been demonstrated for exponents equal to one for the domain and two close to moving boundaries.

## 1.3 Spring analogies

The analogy of modelling element stiffness by springs has been introduced by Robinson *et al.* (1991). This popular approach considers each element's edge $'ij'$ as a lineal spring with stiffness $k_{ij}$ inversely proportional to its length $L_{ij}$:

$$k_{ij} = \frac{1}{(L_{ij})^p} \tag{1.6}$$

Similar to the PSM, the material, or in this case springs, stiffness can be improved with the help of an exponent $p$. This approach has been used on structured hexahedral meshes where each edge is modelled as a spring and each face has a spring through its diagonal to protect the hexahedron from shearing. To validate that new model, they compared aeroelastic flutter experimental results to the spring deformed mesh coupled to an Euler flow and a modal solver.

Then, the method was evaluated to solve 2D compressible Navier-Stokes equations in Farhat and Lanteri (1994). The simulation concerns the flutter of a NACA0012 airfoil surrounded by unstructured triangles in a transonic flow.

To increase the robustness of this method a torsional spring analogy has been studied in two dimensions (Farhat *et al.* (1998)), where each vertex $'i'$ of each triangle $'ijk'$ has a torsional stiffness:

$$C_i^{ijk} = \frac{L_{ij}^2 L_{ik}^2}{4A_{ijk}^2}, \tag{1.7}$$

where $L_{ij}$ and $L_{ik}$ are the length of each edge and $A_{ijk}$ is the area of the triangle. $C_i^{ijk}$ has been added to the stiffness matrix of the lineal springs, to prevent vertices from crossing edges. Simulations of the supersonic inviscid flow over a vibrating plate and of the turbulent transonic aeroelastic analysis of a suspension bridge showed that the combined spring approaches yield valid mesh while the lineal spring analogy failed to complete the simulation.

The three dimensions version has been developed in Farhat *et al.* (1999) and Degand and Farhat (2002). One example to highlight from this improvement is a series of 5-G pull-up manoeuvres of the Langley fighter in transonic flow (Farhat *et al.* (2001)). This simulation shows the efficiency and robustness of the torsional spring analogy to move complex 3D meshes.

### 1.4 Radial Basis Functions (RBF)

In all moving boundaries problems, mesh quality close to moving boundaries needs to be kept unchanged which leads to distributing the nodal displacements relatively to the distance from moving boundaries. Thus, the Radial Basis Function (RBF) method is proposed which interpolates displacements from moving boundaries to fluid nodes. It consists in approximating the displacements of each fluid node by collecting each moving boundaries' displacements and scaling with a particular distance based function, named RBF.

The first application of RBFs was the interpolation of scattered data to generate a continuous field. As a starting point, Broomhead and Lowe (1988) have summarized the results and analysis of previous researchers into one document. This article is considered as a reference for RBF formulations. Let us define the approximate node displacements field $\mathbf{s}(\mathbf{x})$ :

$$\mathbf{s}(\mathbf{x}) = \begin{cases} \mathbf{g}_j, & \text{if } \mathbf{x} = \mathbf{x}_{b_j} \quad \text{, for j =1,2,...m} \\ \sum_{j=1}^{m} \lambda_j \phi(\left\|\mathbf{x} - \mathbf{x}_{b_j}\right\|), & \text{otherwise} \end{cases}, \tag{1.8}$$

where $\mathbf{x}_{b_j}$ are a set of known data points, $m$ is the quantity of known values, $\mathbf{g}_j$ are the known displacement vectors, $\|...\|$ is the Euclidean norm, $\phi(.)$ is the radial basis function and $\lambda_j$ are coefficients which are found from the following system of linear equations:

$$\left[A_{ij}\right]^{-1}\left\{g_j\right\} = \left\{\lambda_j\right\} \quad i,j = 1,2,...,m \quad \text{where,} \tag{1.9}$$

$$A_{ij} \stackrel{\Delta}{=} \phi(\left\|\mathbf{x}_{b_i} - \mathbf{x}_{b_j}\right\|). \tag{1.10}$$

Buhmann (2000) presented an improvement to allow the problem to be uniquely solvable with the addtion of a polynomial function $P(\mathbf{x})$ of a degree lower than the RBF:

$$\mathbf{s}(\mathbf{x}) = \sum_{j=1}^{m} \lambda_j \phi(\left\|\mathbf{x} - \mathbf{x}_{b_j}\right\|) + P(\mathbf{x}) \tag{1.11}$$

One of the first uses of an RBF interpolation for FSI has been depicted in Beckert and Wendland (2001), where the basics of an RBF interpolation for FSI is shown. First, the polynomial should be in the form:

$$P(\mathbf{x}) = C0 + C1x + C2y + C3z, \tag{1.12}$$

where the coefficients ($C0$, $C1$, $C2$ and $C3$) are calculated in a similar manner to $\lambda_j$ coefficients. Then, the authors studied different forms of RBFs that they designed. All those functions have in common the fact that the Euclidean distance is scaled by a support radius. This radius defines how far a boundary node has an impact. The RBF is applied to study the static aeroelastic simulation of a bending wing in a transonic fluid flow solved by compressible Euler equations. It has been shown that even with the hypothesis of inviscid flow, the results are close to the experimental measurements.

de Boer *et al.* (2007) tested 14 different RBFs on a mesh under the deformation of a rotating, translating block and compared the results to those of the semi-torsional spring formulation by Zeng and Ethier (2005) which is a simplified version of Farhat *et al.* (1999). Minimal and average quality of the mesh is evaluated to define which function gives better quality and it is shown that the semi-torsional analogy generates worse minimum and average quality than most of the RBFs. An airfoil flap configuration is validated as well as the coupling with a flow solver for a wing. The authors concluded in recommending two RBFs which can be used in most cases.

Rendall and Allen (2008) developed an algorithm to reduce the number of moving nodes to be used in the RBF interpolation. A modal analysis has been done on a wing to validate their proposed upgrade to the RBF method. Finally, they have shown the results of a parallel version for the simulation of a multi-bladed rotor under cyclic pitch motion(Rendall and Allen (2010)) and it has been stated that very good quality was kept through all the simulation.

## 1.5 Inverse Distance Weighting (IDW) function

The IDW function has been formulated by Shepard (1968) to interpolate surfaces from irregularly distributed data collections. It defines that a continuous data field $\mathbf{s}(\mathbf{x})$ for a known data point $\mathbf{x}(x,y,z)$ can be interpolated from known values $\mathbf{s}(\mathbf{x}_{b_i})$ of position $\mathbf{x}_{b_i}(x_{b_i},y_{b_i},z_{b_i})$ by a weighted average of the inverse distance:

$$\mathbf{s}(\mathbf{x}) = \frac{\sum_{i=1}^{m} w_i(\mathbf{x}) s(\mathbf{x}_{b_i})}{\sum_{i=1}^{m} w_i(\mathbf{x})}. \tag{1.13}$$

Let define the weighting function:

$$w_i(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_{b_i}\|^{-a}, \tag{1.14}$$

where $\|..\|$ is the Euclidean distance, $m$ is the quantity of known values and $a$ is an exponent that should be greater to 1 and exponent a=2 generate better results for 2D interpolations (Shepard (1968)). For large problems, the quantity of values in summation can be reduced to only those close to $\mathbf{x}$. Then for each vector $\overline{\mathbf{x}_{b_i}\mathbf{x}}$ of similar direction the nearest of $\mathbf{x}$ will have a stronger weight.

A lot of work has been done in interpolation of scattered data since Shepard (1968) and is summarized in Franke (1982). Multiple modified weighting functions are presented:

- Limit $w_i$ to have a value inside a sphere and zero elsewhere;
- Setting $w_i = exp(-\beta * \|\mathbf{x} - \mathbf{x}_{b_i}\|^a)$, for $\beta$ a user-defined parameter;
- Multiplying the numerator by an approximation of the field.

Significant analyses have been done and have shown that a combination of these three modifications generates smoother solutions. Although, simulations done in these articles are solely for two dimensions interpolation of a single variable. There were multiple applications of the IDW method, for example in the calculation of crack propagation in a meshless FEM formulation by Belytschko *et al.* (1996), but only few people have worked on the use of IDW to interpolate displacements on a moving mesh.

The first use of the IDW method as an MMA is done by Witteveen and Bijl (2009) for aeroelastic simulation. First, a NACA0012 airfoil is rotated and translated to validate large deformations of the domain. Then, for the same mesh, a flutter simulation at a Mach number of $M_\infty = 0.3$ for an inviscid flow has been performed and compared to the RBF mesh deformation algorithm. Afterwards, an aeroelastic-flutter simulation is done with the AGARD 445.6 wing. The RBF and IDW methods give almost the same lift coefficient through time, however IDW calculates faster for a lower average mesh quality. Although, no results have been shown concerning minimal quality to give information on the robustness of the method. Two small modifications have been done to reduce the computational time of the IDW method, but as a drawback the deformed mesh has a lower quality.

Witteveen and Bijl (2012) have continued their previous work by making different mathematical analyses of the IDW interpolation and its weight function. Important lessons to remember from this work are:

- An increasing exponent $a$ gives a better approximation of the displacements field;
- When $a \to \infty$ the interpolated results are coming from the closest moving boundary points.

Luke *et al.* (2012) optimized the IDW function to make it usable for large three dimensional problems with these modifications:

- The weighting function has been redefined into one part to scale displacements to a wanted region and another part which increase the weight of nodes close to moving boundaries;
- To allow faster computation the displacements field is calculated from the values of groups of moving boundary nodes;
- The summation is done by traversing a k-d tree, which is a space partitioning data structure, of the groups of boundary nodes until an acceptable error;
- The whole calculation has been implemented to allow the workload to be shared by multiple processors.

Multiple test cases have been presented:

- A rotating and translating rectangle to study performance under large deformations;
- A rotating rectangle in the shadow of a sphere surrounded by a mesh for viscous fluid solver;
- A beam is bent to calculate the approximation errors generated by the k-d tree optimisation;
- Two simulations of the transonic flow around aeroelastic wings are done to validate coupling with fluid and structural solvers and to compare the results with wind-tunnel data.

These simulations proved that the IDW scheme is better than the RBF method in computation time and for keeping initial quality of elements close to moving boundaries. Also, the IDW scheme distributes further its displacements than the RBF based MMA which explained the lower average quality depicted in Witteveen and Bijl (2009).

## 1.6 Moving Submesh Approach (MSA)

Lefrançois (2008) proposed a novel approach for mesh motion that should increase robustness and lower computational cost. It consists of solving the motion of a coarse mesh using a known MMA, in his work only the PSM approach was considered, then it interpolates the displacement to the fine computational mesh.

The coarse meshes considered are only composed of triangles and the study is limited to two dimensions. The basic concept of MSA is illustrated in Figure 1.2.

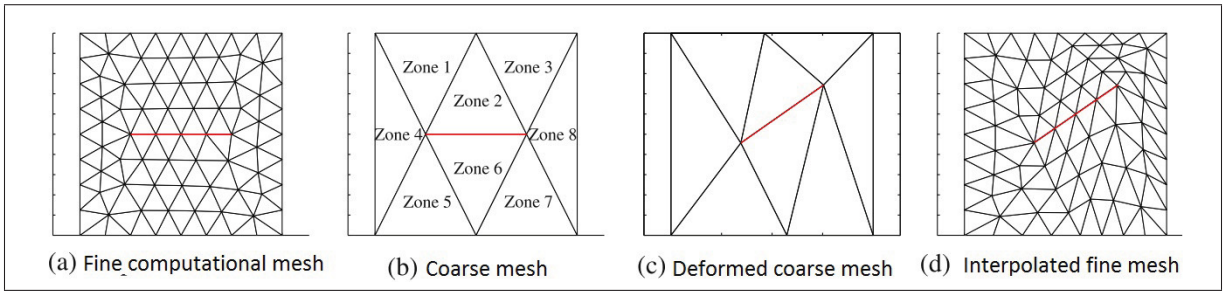



Figure 1.2 MSA basic representation (Lefrançois (2008))

The interpolation from a coarse mesh to a fine mesh needs two basic tools:

- Each fine mesh node are associated to the coarse element that contains it. To do so for all nodes a loop on all elements is done until some vectorial products prove the node is inside one of them;
- The interpolation process is done by a finite element approximation of the sub-triangle area difference (Dhatt *et al.* (2005), p. 112).

Then, Lefrançois (2008) proposed a possibility to encapsulate moving boundaries within a box which undergoes the same rigid-body motion as the moving boundaries. This method keeps unchanged elements which are close to moving boundaries, although no information is given about how to define that motion. Finally, through this work multiple basic proof-of-concept simulations are done without comparisons to different MMA.

He and Zhou (2012) used the MSA framework with the spring analogy to move the coarse mesh. Once again the simulations are done in two dimensions, thus no improvement has been done to how a MSA type algorithm is used.

The last proposed modification to the MSA is to use an RBF interpolation scheme to move the coarse mesh (Liu *et al.* (2012)). The simulations are in three dimensions, thus the interpolation equations are generalized to three dimensions, but for tetrahedron in the coarse mesh only. They stated that : "The scale and the distribution of background mesh are two crucial factors for the efficiency and robustness of the RBFs-MSA...and largely relied on the experience of the user." Three simulations are done to evaluate the new algorithm:

- A box rotated of 60 degrees;
- A wing moves vertically behind a second fixed wing;
- A wing under oscillated bending.

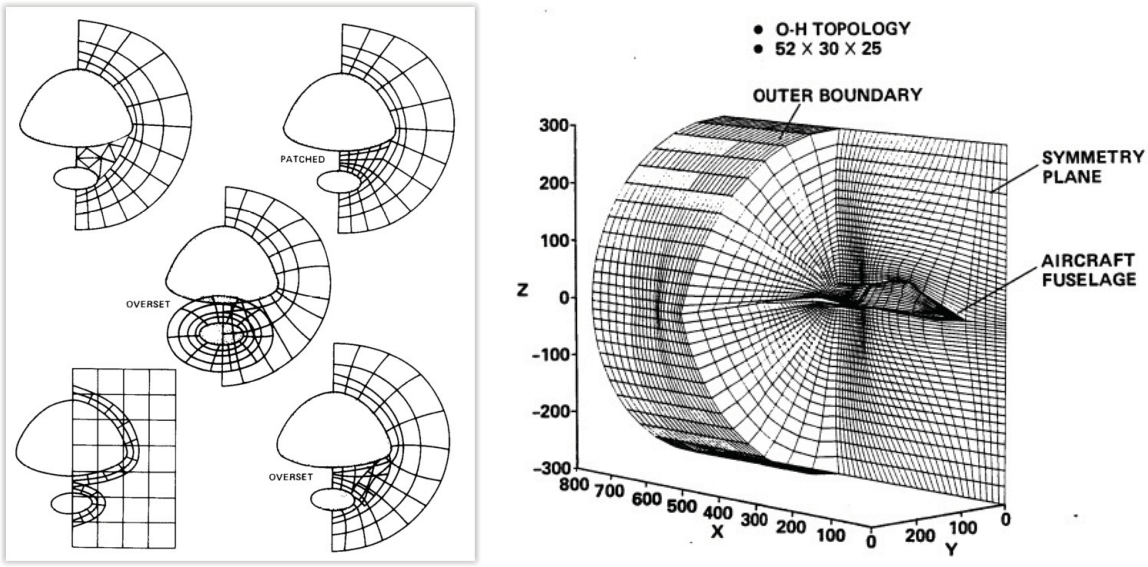
Every fine mesh is computed with the RBF, RBF-MSA and semi-torsional spring analogy and those results have taught us that:

- The semi-torsional spring analogy is less robust than RBFs and RBFs-MSA;
- The RBFs-MSA generate a similar or slightly better mesh than RBFs;
- The displacements field are calculated faster with RBFs-MSA than with RBFs or the semi-torsional spring analogy.

## 1.7 Chimera grid approach

To compute the solution to moving boundaries problems instead of adapting a mesh, a mobile mesh overlapping (or overset) a fixed mesh can be used. The Chimera grid approach of Steger *et al.* (1983) is a special type of overset grid. Overset grids were originally used to generate a structured mesh in a domain having multiple bodies and/or complex geometries.

As the name states, it consists of generating multiple structured grids around different curves of a body and/or different bodies and joining them together through different methods (patched or overset). The joining possibilities for overset grids are shown in Figure 1.3a. (Steger and Buning (1985), Steger and Benek (1987)) and a resulting joined mesh is shown in Figure 1.3b. (Reznick (1988)).




(a) Possible overset approaches

(b) Overset grid of fighter aircraft

Figure 1.3 Overset grids examples

A patched overset grid consists of joining overlapping grids to generate a continuous structured grid where the fluid flow solver will be applied. Patching overset grids for moving boundaries problem is computationally expensive since new nodes and elements are created at each move-

ment, making this method not efficient for mesh-movement. On the other hand, an overset grid concept keeps both overlapping grids, solves the fluid flow on each grid and the solutions are transferred between all overlapping grids. One method to transfer information between overset grids is defined by the "Chimera Grid Embedding technique" (Steger *et al.* (1983)).

A Chimera grid consists of a stationary base grid containing the main geometry and multiple minor unconstrained overlapping grids. To be able to interact with each other, holes are recognized on the base grid where a minor grid overlaps it. A base grid node is considered inside a hole when the vector made of this node and one of the nodes of the minor boundary under consideration is directed outward the minor grid. The hole recognition computation is illustrated in Figure 1.4 which is taken from Steger and Benek (1987). Those nodes of the base grid (called fringe points) are now defined as off; it means that the flow is not solved at those positions and the solution for those positions is obtained by solving the flow of each minor grid.

Afterwards, to be able to solve the flow of minor grids and allow the base grid to be influenced by them an overlapping region is used to interpolate between grids (see Figure1.5a. from Kao *et al.* (1993)). The overlap region solution from the minor grid is interpolated to the base grid and the base grid solution at the outward overlap boundary is used as the far field condition for the minor grid, as shown in Figure 1.5b. from Steger and Benek (1987). Obviously, the final solution is obtained after multiple relaxed iterations of information transfer between overset grids.

Further reading should be done concerning this subject although for this work we have considered that the work necessary to implement such a method, particularly the data structure management, would be too large for an academic research. Also, the task of post-processing overlapping grids to allow good visualisation of the flow without gaps is quite a job. In the future, this method should be used when simple MMA method failed to update meshes, since overset grid approaches seem to have no limitations. This may be why governmental organisations such as NASA and DLR are using overset grid methods to solve all kinds of moving boundaries problems.

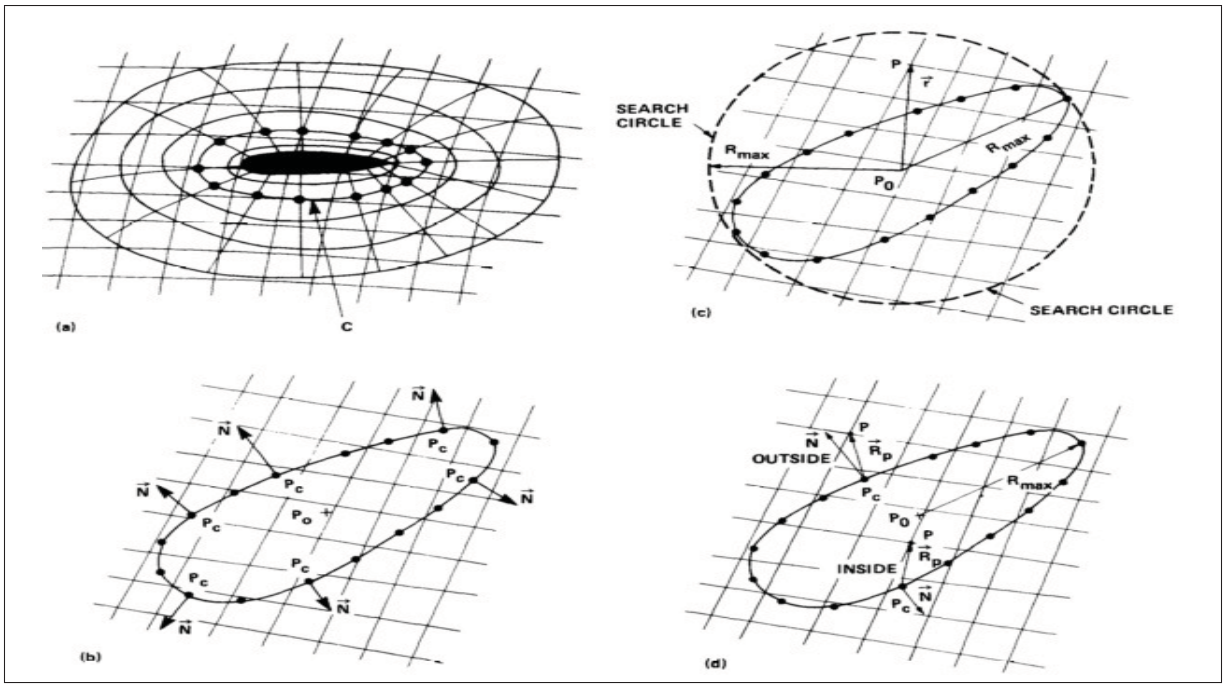
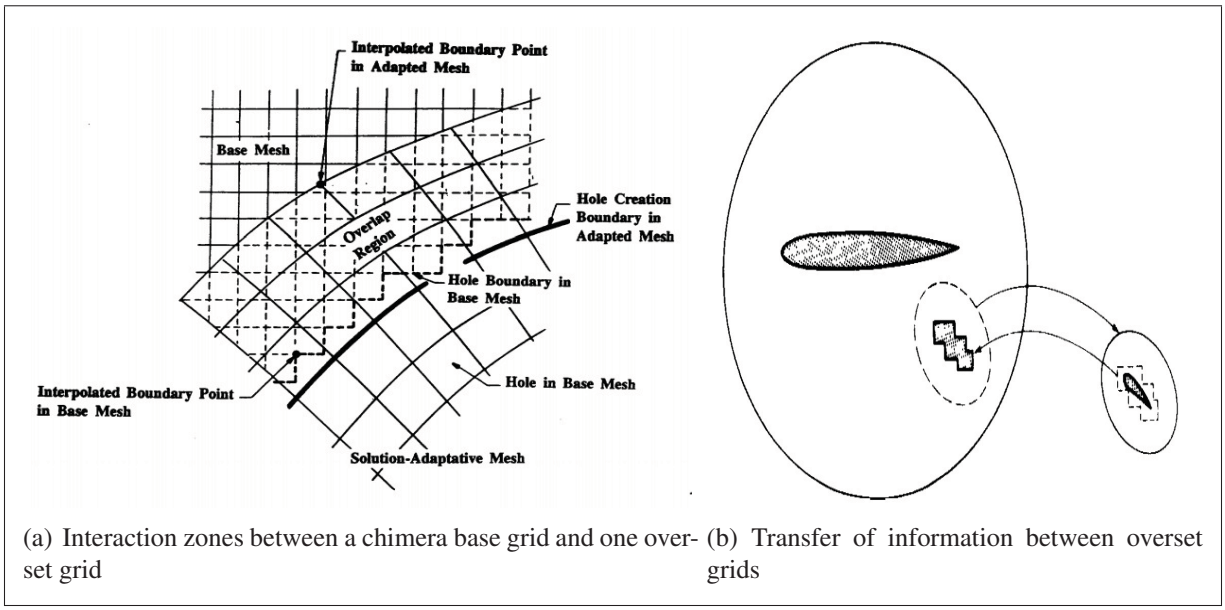



Figure 1.4 Hole generation in two dimensions: (a) Hole boundary defined by level curve C of the minor grid, (b) Construction of outward normals to curve C, (c) Construction of search ball or circle, to find the base grid nodes which are inside the hole, (d) Construction of position vector R and dot product test




(a) Interaction zones between a chimera base grid and one overset grid

(b) Transfer of information between overset grids

Figure 1.5 Chimera concept of information transfer

## 1.8 Improving robustness by smoothing

As stated earlier, a basic approach to mesh motion consists of smoothing the displacements from boundaries to the rest of the mesh. This method works well for small displacements, but can also be used before or after the MMA computation to reduce mesh distortions and allow the mesh to be kept valid longer or with a better quality.

The simplest way of smoothing a displacements field is by adjusting each node position $\mathbf{x}_0$ to be in the middle of their neighbours $\mathbf{x}_{n_i}$ ($i = 1,2..m$), for $m$ the quantity of neighbours (Jones (1974)) and $qtN$ the quantity of nodes in the mesh:

$$\mathbf{x}_0^j = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_{n_i}^j - \mathbf{x}_0^j) \ \ , for\ j = 1..qtN \tag{1.15}$$

However, this Laplacian smoothing operator does not guarantee that elements will have improved quality since the calculation is done per node.

Various modifications to the Laplacian smoothing operator and alternatives have been proposed, although we have decided to consider solely the most recent ones which are considered more promising according to the analysis done by Wilson (2011). First, an objective function (Bank and Smith (1997)) can be defined from elemental quality metrics (Knupp (2003)) and by minimizing it a better quality is expected through the mesh. The objective function does not work when elements are inverted, thus a modification to the objective function has been proposed in Escobar *et al.* (2003).

From those quality metrics was defined a reference undistorted element and by moving nodes of a real element in the direction of the reference one, the mesh can be improved globally or locally. This approach has been presented for tetrahedron in Vartziotis *et al.* (2009), then for hexahedrons in Vartziotis and Wipper (2011) and for mixed elements mesh (tetrahedron, pyramids, prisms and hexahedrons) in Vartziotis and Wipper (2012). This method is called the Geometric Element Transformation Method (GETMe).

## 1.9 Conclusions

Following the previous review some choices are made to deeply study the use and development of a mesh-mover algorithm. Our methodology begins by improving one method of each family: one PDE based method and one interpolation based method.

Since there is no comprehensive comparison of PDEs methods we will assume that all of them have equivalent robustness and we will choose to implement a PSM (Stein *et al.* (2004)) approach for its simplicity, even if spring analogies are more popular.

The interpolation method selected is the IDW since it performs better than the RBF, as it was stated in Luke *et al.* (2012). Also, the proposed modifications from Franke (1982) will not be implemented since the version of Luke *et al.* (2012) already produces great results.

The use of overlapping mesh capability combined to any MMA is an excellent improvement as seen in Liu *et al.* (2012). Thus, the MSA methodology will be applied in this work even if the Chimera grid technique seems to give better results. This decision is made because the implementation of a Chimera state of the art method and its numerical data structure will take much more time. Also, concerning the type of MSA-X combinations, we decide to implement the MSA-IDW and MSA-PSM of Stein *et al.* (2004) which were never, from our knowledge, studied before (MSA-spring and MSA-RBFs were studied in He and Zhou (2012) and Liu *et al.* (2012)).

Following the conclusions of Wilson (2011), the best smoothing algorithm compromise is to use the GETMe smoothing algorithm designed for mixed elements mesh. However, we have decided to implement an untangler based on the GETMe method instead of one which optimises an objective function (Escobar *et al.* (2003)). As future work, the untangler from the literature should be compared to the proposed novel GETMe untangler.

# CHAPTER 2

# TWO APPROACHES FOR MOVING-MESH ALGORITHMS

From the literature review, four methods (PSM, IDW, MSA and GETMe) have been selected to be studied and optimised; they will be detailed in this chapter. Our contributions to the study of MMA can be summarized in the following two global improvements:

- Combining the MSA to mesh-mover algorithms, which gives in total four algorithms to study: PSM, IDW, MSA-PSM and MSA-IDW;
- Smoothing the mesh before mesh-movement and/or at each small movement of large boundary motions with the use of quality metrics designed for moving boundaries problems.

In the subsequent sections, the tools which will achieve the global improvements are :

- The Pseudo-Structural Method (PSM) of Stein *et al.* (2004);
- The Inverse Distance Weighting method (IDW) of Luke *et al.* (2012);
- The Moving Submesh Approach (MSA) of Lefrançois (2008);
- The complete Geometric Transformation Method (GETMe) smoothing approach described by Vartziotis and Wipper (2012);
- The definition of quality metrics to use for mesh-movement;
- The improvements to PSM and IDW based on quality;
- The Novel GETMe Untangler (NGU).

## 2.1 Pseudo-Structural Method (PSM)

The nodes displacements are obtained by solving the equilibrium Partial Differential Equations (PDEs) using the classical finite element formulation for a linear elastic isotropic material, with a Pseudo Young's modulus $E$ and a Pseudo Poisson's coefficient $\nu$:

$$\sum_e (\frac{1}{V^e})^p [K]^e \{U\}^e = \sum_e \{F\}^e, \tag{2.1}$$
$$\text{where } [K]^e = \int_{\Omega^e} [B^e]^T [D^e][B^e] dxdydz,$$

with $V^e$ the element volume (obtained from the integral of the $det([J^e]))$ and $p$ a stiffening power. The stiffening power $p$ is set to one for all elements except for those close to moving boundaries, where the value is doubled as described in Stein *et al.* (2004). To define if an element is close or not to a moving boundary we have decided to divide the fluid domain in 3 different sub-domains: close (volID=30), fluid (volID=31) and far field (volID = 39). This division is done during the mesh generation, in our case, with Pointwise V17.2-R2. The close sub-domain elements will be stiffened, our recommendation is to define this sub-domain as being approximately the height of the boundary layer around each moving boundary.

We present two different isotropic pseudo-materials: the standard which allows shear and Poisson's effect (Equation 2.2), and one which limits the element displacements to be axial only( Equation 2.3).

$$[D] = [D_{shear}] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{2.2}$$

$$[D] = [D_{no\ shear}] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.3}$$

## 2.2 Inverse Distance Weighting (IDW) function

The version of Luke *et al.* (2012) is the one used for this work and it consists of the following: First, the moving boundaries nodal displacements field is defined as $\mathbf{s}(\mathbf{x}_{b_i})$ where $(\mathbf{x}_{b_i})$ is the position vector of the boundary node $i$. Then the fluid mesh nodal displacements field is found by the weighted average of the moving boundaries displacements:

$$\mathbf{s}(\mathbf{x}) = \frac{\sum_{i=1}^{m} w_i(\mathbf{x})\mathbf{s}(\mathbf{x}_{b_i})}{\sum_{i=1}^{m} w_i(\mathbf{x})}, \tag{2.4}$$

where $w_i(\mathbf{x})$ is a two-exponent weighting function of the reciprocal distance:

$$w_i(\mathbf{x}) = A_i \left[ \left( \frac{L_{ref}}{\|\mathbf{x} - \mathbf{x}_{b_i}\|} \right)^a + \left( \frac{\alpha L_{ref}}{\|\mathbf{x} - \mathbf{x}_{b_i}\|} \right)^b \right], \tag{2.5}$$

where $A_i$ is the average area of all moving boundary faces containing the node $i$, $L_{ref}$ is the distance between the mesh centroid and the farthest point of the domain, $a$ is the exponent for the domain, $\alpha$ defines the near body region and $b$ is the exponent for the near body region.

The computation of $\alpha$ reads to:

$$\alpha = \frac{5}{L_{ref}} \max_{i=1}^{m} \left\| \mathbf{s}(\mathbf{x}_{b_i}) - \mathbf{s}_{mean} \right\|, \tag{2.6}$$

$$\text{where } \mathbf{s}_{mean} = \sum_{i=1}^{m} a_i \cdot \mathbf{s}(\mathbf{x}_{b_i})$$

$$\text{and } a_i = \frac{A_i}{\sum_{j=1}^{m} A_j}$$

The weighting function is designed such that it preserves a rigid body motion of nodes close to boundaries, as well as ensures a smooth deformation transition through the mesh. The variables default value of the IDW function are shown in Table 2.1. Since, the IDW mesh-mover has been kindly offered to us by professor E. Luke [1], it has been modified slightly to work in the object oriented presented framework.

Table 2.1 IDW standard parameters

| **Parameters** | **Default Value** |
|---|---|
| $L_{ref}$ | Calculated |
| $a$ | 3 |
| $b$ | 5 |
| $\alpha$ | Calculated, but $\alpha_{min} = 0.1$ |

[1] Dr. E. Luke publications website: http://www.cse.msstate.edu/ luke/publications/index.html

## 2.3 Moving Submesh Approach (MSA)

The procedure presented in Lefrançois (2008) has been generalized for three dimensions and a rapid algorithm has been included to search through the coarse mesh with the help of a k-dimensional tree (Bentley (1975)) which will be explained in Section 2.3.1.

The main steps of the MSA are described:

- A separate coarse mesh is generated composed mostly of tetrahedral elements;
- Each node $\mathbf{x}_{fine}$ of the fine mesh is then associated to the element that contains it in the coarse mesh as shown in Figure 2.1;
- The coarse mesh displacement field is interpolated on the fine mesh.

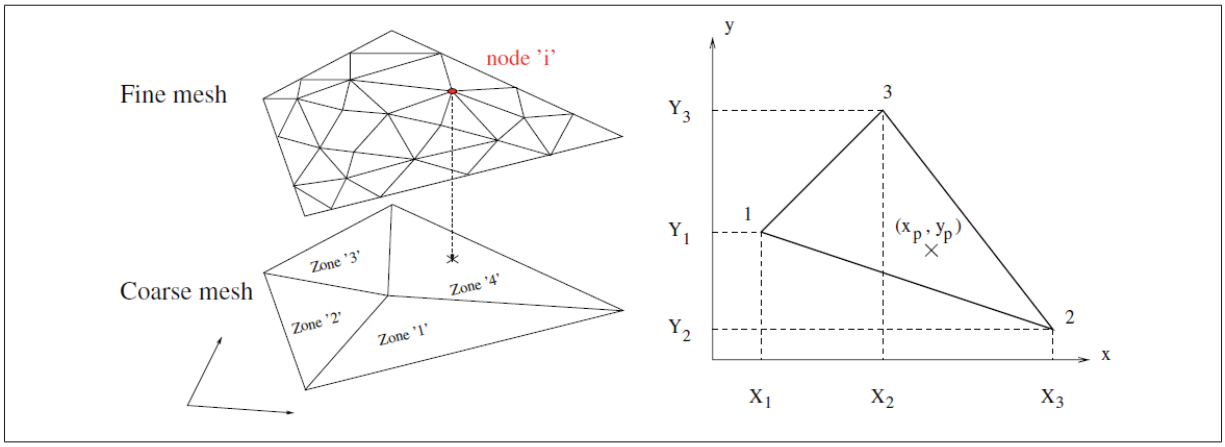



Figure 2.1 MSA definition and fine node in coarse element

The interpolation functions $N_i$, for each $i$ node of the coarse element, are the standard FEM linear interpolations expressed in terms of reference coordinates ( $\xi, \eta, \zeta$). When a fine mesh node ($\mathbf{x}_{fine}$) is located in an element belonging to the coarse mesh, the coordinates ($\boldsymbol{\varsigma} = (\xi, \eta, \zeta)$ ) have to be computed first in order to use the interpolation functions. The general method consists in solving $\mathbf{R} = \mathbf{x}_{fine} - \sum_{i=1}^{nnel} [N_i(\boldsymbol{\varsigma}) \cdot coord] = 0$ with an iterative method. As an example, the simple Newton method is used in Algorithm 2.1 until the relative residual norm gets close to zero with $\mathbf{R}_0$ the initial residual matrix.

Algorithm 2.1 Calculate Interpolation Values

**Calculate Interpolation Values**
**Input** : A fine node ($\mathbf{x}_{fine}$) and an element coordinates matrix ($coord$)
**Output**: Interpolation coefficients $v_i$

$\boldsymbol{\varsigma} = [0,0,0]$
**while** $\frac{\|\boldsymbol{R}\|}{\|\boldsymbol{R}_0\|} > tolerance$ **do**
  $\mathbf{R} = \mathbf{x}_{fine} - \sum_{i=1}^{nnel} [N_i(\boldsymbol{\varsigma}) \cdot coord]$
  $\boldsymbol{\Delta} = -[J]^{-1} \cdot R$
  $\boldsymbol{\varsigma} = \boldsymbol{\varsigma} + \boldsymbol{\Delta}$
**end while**
Set $v_i = N_i(\boldsymbol{\varsigma})$

In Algorithm 2.1, *nnel* is the number of nodes for the current coarse element, *coord* is the matrix composed of the coarse element nodes coordinates and $[J]$ is the Jacobian matrix of the residual equation. This algorithm is applicable to any kind of element, but an exact analytical solution can be found in the case of tetrahedral elements (Lefrançois (2008) and Liu *et al.* (2012)). Then, the operation of projecting the fine mesh onto the coarse mesh essentially provides the set of $v_i$ coefficients and is done only once. For each boundaries movement, the displacement vectors of the fine mesh is: $\boldsymbol{\delta}(\mathbf{x}_{fine}) = \sum_{i=1}^{nnel} v_i \cdot \boldsymbol{\delta} el_i$, where $\boldsymbol{\delta} el_i$ is the nodal displacements of the coarse mesh element.

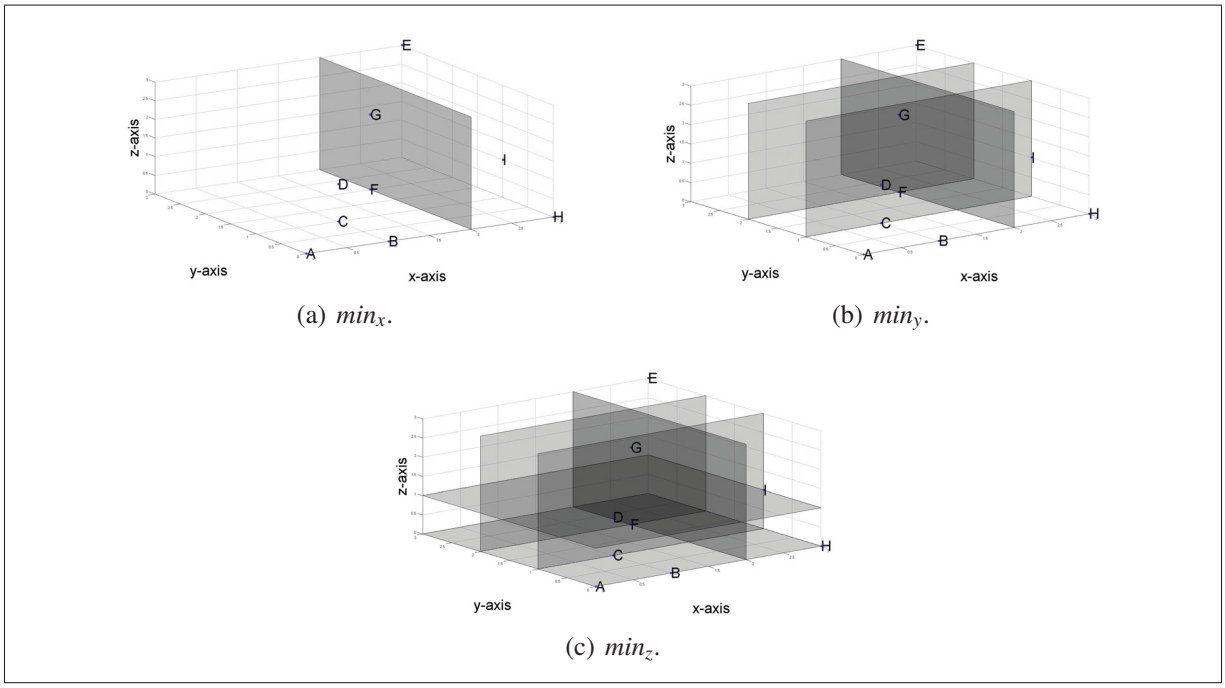
### 2.3.1 Alternating Digital Tree (ADT)

In order to link a fine node to a coarse element a powerful search algorithm is used and described in the following section. To construct an ADT (Bonet and Peraire (1991)) the coarse mesh is distributed in a tree data structure where every node of the tree contains an element. To populate those tree-nodes, the group of elements is divided recursively into smaller sub-groups, according to an element variable, until all elements are contained by a node.

When a group is split, the element in the middle is called the splitting plane. This element is stored in the current tree node and the rest of the group goes to the left, if their value is smaller, or to the right, if their value is bigger or equal to the splitting plane.

In our algorithm, the alternating splitting variables are the minimum coordinates of the bounding boxes. A bounding box is defined for each element by two nodes: the minimum and the maximum values from each coordinates.

An example of the ADT splitting process is shown for nine elements (A,B...H,I) represented by their minimum node in Figure 2.2 and the relevant ADT is shown in Figure 2.3.




(a) $min_x$. (b) $min_y$.

(c) $min_z$.

Figure 2.2 Three subsecant cut of the ADT

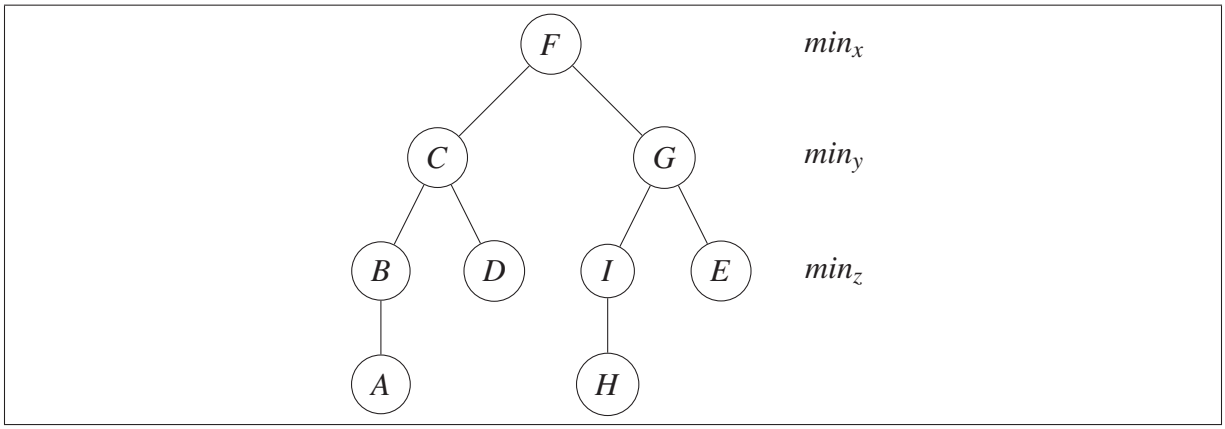



Figure 2.3 Alternating digital tree example

Finally, the search in an ADT to find which element contains each fine mesh nodes is explained in Algorithm 2.2. In Algorithm 2.2, a $fineNode$ is considered inside a $treeNode.element$ if it is inside one of the tetrahedron composing the element. The connectivity of each tetrahedral per type of element is defined by Dompierre *et al.* (1999).

Algorithm 2.2 Recursive search in an ADT

**Recursive search in an ADT**
**Input** : A fine node ($fineNode$) and a tree node ($treeNode$)
**Output**: The element containing the fine node ($fineNode.element$)

```
1  i = current splitting plane
2  if treeNode.min_i ≤ fineNode_i ≤ treeNode.max_i then
3  |  if fineNode is inside the treeNode.element then
4  |  |  fineNode.element = treeNode.element
5  |  |  break
6  |  end if
7  else
8  |  if treeNode.left.min_{i+1} ≤ fineNode_i ≤ treeNode.left.max_{i+1} then
9  |  |  Recursive search in treeNode.left
10 |  else
11 |  |  Recursive search in treeNode.right
12 |  end if
13 end if
```

The tools for finding a coarse element and calculating interpolations functions can be also used for different applications such as: to initialize a fine mesh fluid flow from a coarse mesh fluid flow solution; or it can be used to define holes and to interpolate variables between overlapping grids for an overset grid method.

## 2.4 Geometric element transformation method (GETMe)

The GETMe method of Vartziotis *et al.* (2009) is used to maintain the orthogonality of the mesh especially close to boundary layers. Hence, it allows the deformed mesh to maintain its initial quality. The GETMe algorithm is defined for mixed meshes composed of standard FEM elements: tetrahedron, hexahedron, pyramids and prisms. In complex meshes there are

always layers of prisms or hexahedrons close to boundaries, then the rest of the domain is composed of tetrahedron and pyramid elements allow the transition between quadrilateral faces and triangular faces. Thus, since all type of elements are needed for complex FEM problems the smoothing algorithm needs to be able to handle all of them.

### 2.4.1 GETMe definition

The method consists of making an element, composed of $\mathbf{n}_k$ nodes, closer to its undistorted standard element. The node numbering and the faces connectivity matrix $\mathbf{F}$ for each element's type are presented in Figure 2.4 and in Table 2.2, where $\mathbf{d}_k$ are the dual element nodes and $\overline{\mathbf{F}}$ is the dual element faces connectivity matrix. A dual element is a geometry of reference enclosed in an element and its faces are used to define the transformation.

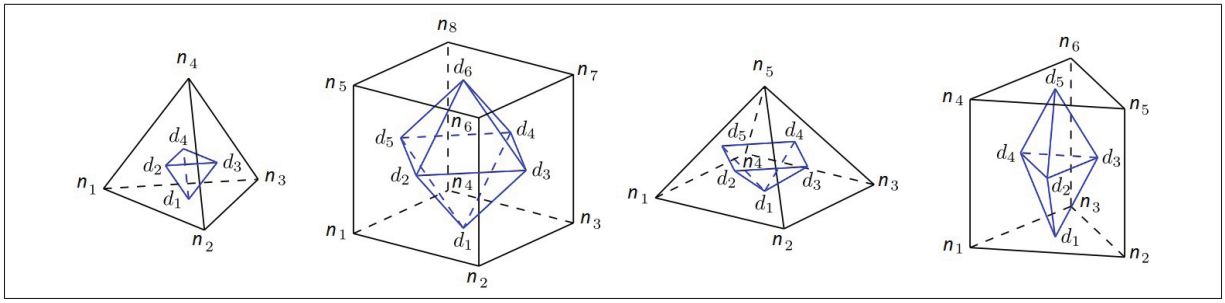



Figure 2.4 Elements and dual elements connectivities from Vartziotis and Wipper (2012)

Each dual element node position is calculated as follows:

$$\mathbf{d}_k = \frac{1}{|\mathbf{F}_k|}\sum_{i=1}^{|\mathbf{F}_k|}\mathbf{n}_{\mathbf{F}_{k,i}},\ k \in 1,...,|\mathbf{F}|, \tag{2.7}$$

where $|\mathbf{F}_k|$ is the number of nodes in the face $k$ and $|\mathbf{F}|$ is the number of element faces.

The geometric transformation consists of orienting all nodes $\mathbf{n}_k$ of an element from a base point $\mathbf{b}_k$ in the direction of its dual element face to the new nodal positions $\mathbf{n}_k^{'}$:

$$\mathbf{n}_k^{'} = \mathbf{b}_k + \frac{\sigma}{\sqrt{|\mathbf{nv}_k|}}\mathbf{nv}_k, \qquad k \in \{1, ..., |\mathbf{n}_k|\}, \tag{2.8}$$

where $\sigma$ is a scaling factor (of order $3/2$ as recommended by Vartziotis and Papadrakakis (2013)), $|\mathbf{n}_k|$ is the number of nodes of the element, $\mathbf{nv}_k$ are the normals defined as:

$$\mathbf{nv}_k = \begin{cases} (\mathbf{d}_{\overline{\mathbf{F}}_{k,2}} - \mathbf{d}_{\overline{\mathbf{F}}_{k,1}}) \times (\mathbf{d}_{\overline{\mathbf{F}}_{k,3}} - \mathbf{d}_{\overline{\mathbf{F}}_{k,1}}) & if\ |\mathbf{F}_k| = 3, \\ \frac{1}{2}(\mathbf{d}_{\overline{\mathbf{F}}_{k,3}} - \mathbf{d}_{\overline{\mathbf{F}}_{k,1}}) \times (\mathbf{d}_{\overline{\mathbf{F}}_{k,4}} - \mathbf{d}_{\overline{\mathbf{F}}_{k,2}}) & if\ |\mathbf{F}_k| = 4 \end{cases} \tag{2.9}$$

and the base points are defined as:

$$\mathbf{b}_k = \begin{cases} \mathbf{c}_k & if\ |\mathbf{n}_k| \in \{4, 8\}\ or\ (|\mathbf{n}_k| = 5\ and\ k = 5), \\ \mathbf{a}_k(\tau); \tau = \frac{1}{2} + \sigma & if\ |\mathbf{n}_k| = 5\ and\ 1 \leq k \leq 4, \\ \mathbf{a}_k(\tau); \tau = \frac{4}{5}(1 - \frac{\sqrt{2}\sigma}{\sqrt[4]{39}}) & if\ |\mathbf{n}_k| = 6, \end{cases} \tag{2.10}$$

for $\mathbf{c}_k$ being the dual element faces centroids and $\mathbf{a}_k$ the base points for triangular faces. Base points are calculated as follows:

$$\mathbf{c}_k = \frac{1}{\overline{\mathbf{F}}_k} \sum_{i=1}^{\overline{\mathbf{F}}_k} \mathbf{d}_{\overline{\mathbf{F}}_{k,i}} \quad \text{and} \tag{2.11}$$

$$\mathbf{a}_k(\tau) = (1 - \tau)\mathbf{d}_{\overline{\mathbf{F}}_k,1} + \tau\frac{1}{2}(\mathbf{d}_{\overline{\mathbf{F}}_k,2} + \mathbf{d}_{\overline{\mathbf{F}}_k,3}). \tag{2.12}$$

The operation of directing the node of a real element in the direction of its dual element faces normals is represented in Figure 2.5.

Table 2.2 Connectivities of GETMe geometries

| **Geometries** | **Tetrahedra** | **Hexahedra** | **Pyramid** | **Prism** |
|---|---|---|---|---|
| **F** | $\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 4 \\ 2 & 3 & 4 \\ 3 & 1 & 4 \end{bmatrix}$ | $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 6 & 5 \\ 2 & 3 & 7 & 6 \\ 3 & 4 & 8 & 7 \\ 4 & 1 & 5 & 8 \\ 5 & 8 & 7 & 6 \end{bmatrix}$ | $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 5 & \\ 2 & 3 & 5 & \\ 3 & 4 & 5 & \\ 4 & 1 & 5 & \end{bmatrix}$ | $\begin{bmatrix} 1 & 2 & 3 & \\ 1 & 2 & 5 & 4 \\ 2 & 3 & 6 & 5 \\ 3 & 1 & 4 & 6 \\ 4 & 6 & 5 & \end{bmatrix}$ |
| $\overline{\mathbf{F}}$ | $\begin{bmatrix} 1 & 2 & 4 \\ 1 & 3 & 2 \\ 1 & 4 & 3 \\ 2 & 3 & 4 \end{bmatrix}$ | $\begin{bmatrix} 1 & 2 & 5 \\ 1 & 3 & 2 \\ 1 & 4 & 3 \\ 1 & 5 & 4 \\ 6 & 5 & 2 \\ 6 & 2 & 3 \\ 6 & 3 & 4 \\ 6 & 4 & 5 \end{bmatrix}$ | $\begin{bmatrix} 1 & 2 & 5 & \\ 1 & 3 & 2 & \\ 1 & 4 & 3 & \\ 1 & 5 & 4 & \\ 2 & 3 & 4 & 5 \end{bmatrix}$ | $\begin{bmatrix} 1 & 2 & 4 \\ 1 & 3 & 2 \\ 1 & 4 & 3 \\ 5 & 4 & 2 \\ 5 & 2 & 3 \\ 5 & 3 & 4 \end{bmatrix}$ |

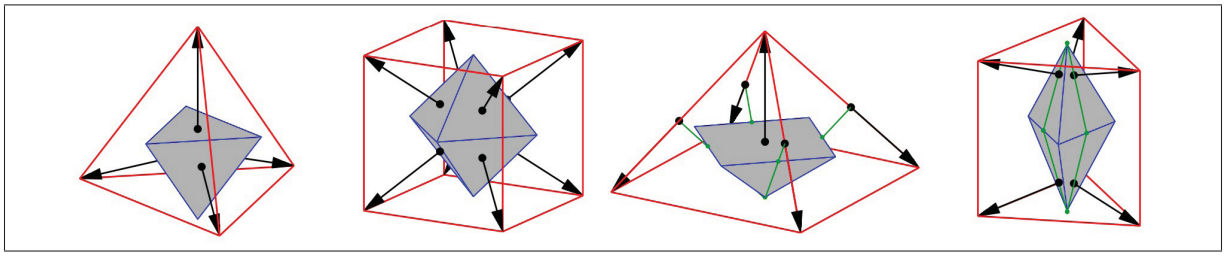


Figure 2.5 Reference elements and their normals used to transform distorted real elements (Vartziotis and Wipper (2012))

### 2.4.2 GETMe quality definition

The quality of any element is defined by the average quality value of the tetrahedron composed of each element node and its connected three edges:

$$q(\mathbf{n}_k) = \frac{1}{|\mathbf{Nt}|}\sum_{k=1}^{|\mathbf{Nt}|}\frac{3det(\mathbf{S}_k)^{2/3}}{tr(\mathbf{S}_k^T\mathbf{S}_k)}, \quad \mathbf{S}_k = \mathbf{Diff}(\mathbf{Nt}_k)\mathbf{W}^{-1}, \tag{2.13}$$

where **Nt** is the tetrahedron connectivity matrix, $|\mathbf{Nt}|$ is the number of tetrahedron of the element, $\mathbf{Diff}(\mathbf{Nt}_k)$ is the difference $3x3$ matrix defined as:

$$\mathbf{Diff}(\mathbf{Nt}_k) = \left[(\mathbf{n}_{\mathbf{Nt}_{k,2}} - \mathbf{n}_{\mathbf{Nt}_{k,1}}), (\mathbf{n}_{\mathbf{Nt}_{k,3}} - \mathbf{n}_{\mathbf{Nt}_{k,1}}), (\mathbf{n}_{\mathbf{Nt}_{k,4}} - \mathbf{n}_{\mathbf{Nt}_{k,1}})\right] \tag{2.14}$$

and **W** is an element type dependant target matrix. An element is valid if $det(\mathbf{S}_k) > 0$ which implies of positive volume and an equiangular element has a quality of $q(\mathbf{n}_k) = 1$. The tetrahedron **Nt** and reference **W** connectivity matrices per element type are defined in Table 2.3.

Table 2.3 Connectivities for quality calculation

| **Geometries** | **Tetrahedra** | **Hexahedra** | **Pyramid** | **Prism** |
|---|---|---|---|---|
| **Nt** | $\begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix}$ | $\begin{bmatrix}1 & 4 & 5 & 2\\ 2 & 1 & 6 & 3\\ 3 & 2 & 7 & 4\\ 4 & 3 & 8 & 1\\ 5 & 8 & 6 & 1\\ 6 & 5 & 7 & 2\\ 7 & 6 & 8 & 3\\ 8 & 7 & 5 & 4\end{bmatrix}$ | $\begin{bmatrix}1 & 2 & 4 & 5\\ 2 & 3 & 1 & 5\\ 3 & 4 & 2 & 5\\ 4 & 1 & 3 & 5\end{bmatrix}$ | $\begin{bmatrix}1 & 2 & 3 & 4\\ 2 & 3 & 1 & 5\\ 3 & 1 & 2 & 6\\ 4 & 6 & 5 & 1\\ 5 & 4 & 6 & 2\\ 6 & 5 & 4 & 3\end{bmatrix}$ |
| **W** | $\begin{bmatrix}1 & \frac{1}{2} & \frac{1}{2}\\ 0 & \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{6}\\ 0 & 0 & \sqrt{\frac{2}{3}}\end{bmatrix}$ | $\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$ | $\begin{bmatrix}1 & 0 & \frac{1}{2}\\ 0 & 1 & \frac{1}{2}\\ 0 & 0 & \frac{\sqrt{2}}{2}\end{bmatrix}$ | $\begin{bmatrix}1 & \frac{1}{2} & 0\\ 0 & \frac{\sqrt{3}}{2} & 0\\ 0 & 0 & 1\end{bmatrix}$ |

### 2.4.3 Smoothing algorithms

The application of the GETMe on a mesh is divided in a global smoothing approach, which smooths all elements, and a local smoothing which smooths worst quality elements of the mesh as explained in Vartziotis and Papadrakakis (2013). The simultaneous algorithm (global smoothing) loop is defined in Algorithm 2.3; it stops when the average mesh quality difference has reached the input tolerance.

Algorithm 2.3 Global Smoothing

**Global Smoothing**
**Input** : A mesh and a tolerance ($tol$)
**Output**: A mesh with improved $q_{avg}$

**while** $q_{avg}$ *improvement is less than* $tol$ **do**
  **for** $j \in J$ **do**
    Transformation according to Equation (2.8) to obtain $\mathbf{n}'_{i\in I(j)}$
    Scaling $\mathbf{n}'_{i\in I(j)}$ to get $\mathbf{n}_{i\in I(j),scaled}$ with:

$$\mathbf{n}_{i\in I(j),scaled} = \mathbf{q} + \psi_j(\mathbf{n}'_{i\in I(j)} - \mathbf{q}') \tag{2.15}$$

    Store the element contribution per node $i \in I(j)$ in the matrix $\mathbf{n}'_{i\in I,j}$
  **end for**
  New nodes $\mathbf{n}'_{i\in I}$ are obtained by the weighted average of all element contribution :

$$\mathbf{n}'_{i\in I} = \frac{\sum_{j\in J(i)} w_j \mathbf{n}'_{i,j}}{\sum_{j\in J(i)} w_j}, \;\; with\, w_j = \sqrt{\frac{\sum_{k\in J(j)} q(\mathbf{n}_{l\in I(k)})}{|J(j)|\, q(\mathbf{n}_{l\in I(j)})}} \tag{2.16}$$

  **foreach** *Values of the relaxation vector* $\gamma_k$ **do**
    Relaxation of unrelaxed nodes according to:

$$\mathbf{n}_{i\in I,relaxed} = (1-\gamma_k)\mathbf{n}_{i\in I} + \gamma_k \mathbf{n}'_{i\in I} \tag{2.17}$$

    Reset each node $\mathbf{n}_{i\in I,relaxed}$ to $\mathbf{n}'_{i\in I}$ if it produces a negative volume element
  **end foreach**
  Set the element nodes $\mathbf{n}_{i\in I}$ to $\mathbf{n}_{i\in I,relaxed}$
  Calculate quality of the new mesh with Equation (2.13)
**end while**

In Algorithm 2.3, $\mathbf{q}$ and $\mathbf{q}'$ are respectively the centroids of $\mathbf{n}_i$ and $\mathbf{n}'_i$, $J$ is the index set of all elements, $J(i)$ is the index set of elements associated to the node $i$, $J(j)$ is the index set of neighbouring elements of element $j$, $|J(j)|$ is the number of neighbours of element $j$, $I$ is the index set of all nodes and $I(j)$ or $I(l)$ is the index set of nodes associated to the element $j$ or $l$. Then, $\psi_j$ is equal to the mean edge length of $\mathbf{n}_{i\in I(j)}$ divided by the mean edge length of $\mathbf{n}'_{i\in I(j)}$ and the vector of relaxation values is $\gamma_k = [1, 1/8, 1/16, 0]$.

Also, Equation 2.15 scales the transformation to make it dependant to the element's centroid and the relaxation step (Equation 2.17) controls the transformation to be applied only if it improves the mesh quality.

Global smoothing increases average quality, but may decrease the minimal quality of the mesh. To correct this situation a similar process called the sequential GETMe smoothing, or local smoothing, is used and defined in Algorithm 2.4 with the relaxation values $\gamma_k = [1/2, 1/10, 1/100, 0]$.

Algorithm 2.4 Local Smoothing

**Local Smoothing**
**Input** : A mesh and a minimal quality (*minMesh*)
**Output**: A mesh with improved $q_{min}$

**while** $q_{min}$ *has not increased for* 5 *iterations* **do**
  **for** $j \in J$ *if* $q(\boldsymbol{n}_{i\in I(j)}) \leq minMesh$ **do**
    Transformation according to Equation (2.8) to obtain $\mathbf{n}'_{i\in I(j)}$
    Scaling $\mathbf{n}'_{i\in I(j)}$ with Equation (2.15) to get $\mathbf{n}'_{i\in I(j),scaled}$
    Store the element contribution per node $i \in I(j)$ in the matrix $\mathbf{n}'_{i\in I,j}$
  **end for**
  New nodes $\mathbf{n}'_{i\in I}$ are obtained by the weighted average defined in Equation (2.16)
  **foreach** *Values of the relaxation vector* $\gamma_k$ **do**
    Relaxation of unrelaxed nodes according to Equation (2.17)
    Reset each node $\mathbf{n}_{i\in I,relaxed}$ to $\mathbf{n}'_{i\in I}$ if it produces a negative volume element
  **end foreach**
  Set the element nodes $\mathbf{n}_{i\in I}$ to $\mathbf{n}_{i\in I,relaxed}$
  Calculate quality of the new mesh with Equation (2.13)
**end while**

## 2.5 Quality metrics for mesh-movement

Since, the goal of the smoothing algorithm is to keep the initial quality of the mesh we propose quality metrics adapted to this goal. They are $q_{chg}$which has been used by Luke *et al.* (2012), among others, and $q_{chgAbs}$ which is an adaptation of the former.

Let us define these quality metrics:

$$q_{chg} = \frac{min(q_{orig}, q_{def})}{max(q_{orig}, q_{def})}, \tag{2.18}$$

$$q_{chgAbs} = \frac{q_{def}}{q_{orig}}, \tag{2.19}$$

where $q_{orig}$ is the original element quality and $q_{def}$ is the quality of the deformed element. The metric $q_{chg}$ is not usable with smoothing algorithms since for a better $q_{def}$, $q_{chg}$ will decrease.

In the equations for the MMAs and the smoothing algorithms one of the three quality metrics ($q$, $q_{chg}$ *or* $q_{chgAbs}$) needs to be chosen. In our simulations we prefer to use $q_{chgAbs}$ because the mesh should be kept with the same or higher quality as the original one and the smoothing algorithms criteria are to increase mesh quality. This means that we assume the mesh at $t = 0$ is already of good quality, thus with $q_{chgAbs}$ as quality criteria the mesh should be kept with good quality after deformation.

Moreover, the proposed quality metrics ($q_{chg}$ *and* $q_{chgAbs}$) are not suitable for elements in boundary layer regions which are purposely stretched in the parallel direction of the connected boundary. Smoothing those elements with $q_{chgAbs}$ as criteria will result in more equiangular and equilateral elements which is not desired.

Thus, the approach based on preserving initial quality with a Size-Shape metric of Knupp (2012) will be used. In this article the mesh is optimized to generate better shaped elements except for elements close to boundaries where stretched elements are kept unchanged.

Thus, the new equation of $q_{chgAbs}$ for stretched elements is:

$$q_{pre} = \frac{1}{|nt|}\sum_{k=1}^{|nt|}\frac{3}{\sqrt{trace(S2_k)}\sqrt{trace(S2_k^{-1})}+(det(S2_k)-1)^2}, \tag{2.20}$$

where $S2_k = D(nt_k)W2(nt_k)^{-1}$ and $W2(nt_k)$ is the initial difference matrix ($D(nt_k)$).

We propose to use Equation 2.20 to calculate $q_{chgAbs}$ for all elements under a desired distance from any boundary. The distance calculation is done with an ADT structure where each tree-node contains a moving boundary node instead of an element.

For simple problem, the use of $q_{pre}$ can be replace by not applying the smoothing algorithms to elements inside boundary layer regions and this preserving metric is strongly recommended for simulations involving multiple bodies.

## 2.6 Improving MMAs with quality stiffener

MMAs presented in previous sections are designed to maintain a smooth displacements field while keeping elements close to boundaries less deformed. For the IDW scheme, the criterion is based on the distance to the solid wall and for the PSM the criteria is related to element size. Although, it should be remembered that these deformed meshes will be eventually used to solve a physical problem and it is well known that a mesh with highly distorted elements can give bad results. Thus, we propose to protect elements from being largely distorted with the implementation of a quality stiffener.

For the PSM the inverse of the quality is added to Equation 2.1:

$$\sum_e (\frac{1}{V^e})^p (\frac{1}{q^e})^{pq} [K]^e \{U\}^e = \sum_e \{F\}^e, \tag{2.21}$$

where $pq$ is the quality rigidity power factor. That modification allows the use of three different methods of stiffening: the original approach ($p = 1$ and $pq = 0$), stiffening the mesh according

to its quality ($p = 0$ and $pq = 1$), or the two factors can be combined to define the element stiffness ($p = 1$ and $pq = 1$).

A similar modification is done to the IDW algorithm to allow less deformation for low quality elements. Since, the current algorithm is interpolating nodes and the nodes do not possess a quality value, the value is computed from the lowest value of the elements constructed by each node. To stiffen nodes of distorted elements the IDW method's weight function is modified to use an exponent ($c$) for quality values smaller than one:

$$w_i(\mathbf{x}) = A_i \cdot (q(\mathbf{x})_{min}) \left[ \left( \frac{L_{ref}}{\|\mathbf{x} - \mathbf{x}_{b_i}\|} \right)^a + \left( \frac{\alpha L_{ref}}{\|\mathbf{x} - \mathbf{x}_{b_i}\|} \right)^b \right] + A_i \cdot (1 - q(\mathbf{x})_{min}) \left[ \left( \frac{L_{ref}}{\|\mathbf{x} - \mathbf{x}_{b_i}\|} \right)^c + \left( \frac{\alpha L_{ref}}{\|\mathbf{x} - \mathbf{x}_{b_i}\|} \right)^c \right], \tag{2.22}$$

where $q(\mathbf{x})_{min}$ is the quality of the current node and $c$ the exponent that controls the impact of quality on mesh-movement. In other words when $q(\mathbf{x})_{min} = 1$ the original weighting function is used and when $q(\mathbf{x})_{min} < 1$ the weighting function is relaxed with exponent $c$. This option should reduce the deformations of elements with decreasing quality. The modified expression is only used when $c > 0$.

## 2.7 Novel GETMe Untangler (NGU)

For complex moving boundaries problems there is a risk of elements being inverted, thus making the mesh unusable for physical problem solving. To improve the robustness of our methodology we propose an untangler based on the GETMe smoothing methods with small changes in order to be able to revert to positive volume a collapsed element. Thus, the Algorithm 2.5 is developed similar to the local smoothing algorithm, but with a modified weight calculation and with only one relaxation value.

The proposed algorithm (Algorithm 2.5) should be seen as a tool to help in keeping a deformed mesh valid but not as an MMA to be used alone. Thus, the more robust MMA configuration should be used and the Novel GETMe Untangler (NGU) will be the "air-bags" which protect the mesh from being invalid . Also, it is a good practice to use the smoothing algorithms after an untangling operation, since elements are guaranteed to be of positive volume but they may be of low quality.

Algorithm 2.5 Novel GETMe Untangler

**Novel GETMe Untangler**
**Input** : A mesh with negative elements
**Output**: A mesh without negative elements

**while** *There is negative elements* **do**
  **for** $j \in J$ *if* $q(\boldsymbol{n}_{i\in I(j)}) \leq 0.0$ **do**
    Transformation according to Equation (2.8) to obtain $\mathbf{n}'_{i\in I(j)}$
    Scaling $\mathbf{n}'_{i\in I(j)}$ with Equation (2.15) to get $\mathbf{n}'_{i\in I(j),scaled}$
    Store the element contribution per node $i \in I(j)$ in the matrix $\mathbf{n}'_{i\in I,j}$
  **end for**
  New nodes $\mathbf{n}'_{i\in I}$ are obtained by Equation (2.16), but with:

$$w_j = \begin{cases} 10.0, & \textit{if } q(\mathbf{n}_{i\in I(j)}) \leq 0.0 \\ 1.00, & \textit{otherwise} \end{cases} \tag{2.23}$$

  Set $\gamma_k = [0.01]$
  Relaxation of unrelaxed nodes according to Equation (2.17)
  Reset each node $\mathbf{n}_{i\in I,relaxed}$ to $\mathbf{n}'_{i\in I}$ if it produces more negative volume element
  Set the element nodes $\mathbf{n}_{i\in I}$ to $\mathbf{n}_{i\in I,relaxed}$
  Calculate quality of the new mesh with Equation (2.13)
**end while**

## 2.8 Methodology summary

The summary of the discussed robust approach to mesh motion is presented in Algorithm 2.6.

Algorithm 2.6 Mesh-mover algorithms robust strategy (MMARS)

**Mesh-mover algorithms robust strategy (MMARS)**
**Input** : A mesh and prescribed boundary displacements **g**
**Output**: A deformed mesh

```
1 The boundaries are moved according to g
2 The displacements field is solved with: PSM, IDW, MSA-PSM or MSA-IDW
3 if There is negative elements then
4 |   The mesh is repaired with the NGU
5 end if
6 if Mesh quality is under desired value then
7 |   The mesh is smoothed with the GETMe algorithms
8 end if
```

# CHAPTER 3

# AN OBJECT-ORIENTED FEM METHODOLOGY

Appropriating a computer code design is mandatory to solve complex and/or large scale problems. Multiple methods for code design can vary from simple to sophisticated. The strategy proposed is the most common for software design (but not for FEM solvers): the object-oriented paradigm of programming. This paradigm is defined by Standardization (1999) as being a programming language that supports objects, classes, and inheritance. This way of thinking, while programming, provides the capacity to modulate through code generalisation which makes easy the solving of any problems. Moreover as expressed by Booch (1986):

> Perhaps the greatest strength of an object-oriented approach to development is that it offers a mechanism that captures a model of the real world. This leads to improved maintainability and understandability of systems whose complexity exceeds the intellectual capacity of a single developer or a team of developers.

This paradigm allows developers to concentrate on the mathematical and physical abstractions without the intellectual barrier created by computer languages. In the current section will be described the code structure, programming strategies to solve problems and some detailed implementations. This description of the code should help new developers or users of the proposed code to be able to use, modify and understand it. The recommended steps of Booch (1986) specific to code development will be followed to describe the code:

- Objects and attributes (Section 3.1);
- Functions needed by each object (Section 3.2);
- Data structure and visibility (Section 3.3);
- Interfaces (Section 3.4);
- Implementation (Section 3.5).

## 3.1 Objects and attributes

The core of the design is composed of FEM entities, since we consider that, for any user, knowledge of this field is required. FEM entities can be expressed in this fashion: a numerical analysis consists in solving a domain; the domain is represented by a mesh of nodes; nodes are connected to each other to create faces and volumes (elements). We have divided FEM entities, called classes, in four natural groups: one dimension, two dimensions, three dimensions and global. The abstract classes and their children are shown in Figure 3.1.

An abstract class consists of a type of object that is undefined to be used without clarification, but its definition helps collecting similarities (functions, attributes, and interfaces) between real classes. Each child's class is connected to their parent class with an arrow (Figure 3.1), this means they inherit functions and attributes from their connected parent class.

For example, a FEM analysis is not possible until we know if it is steady state or not and if the physics under inquiry is linear or not, but they all need a mesh and boundary conditions in order to compute solutions. Also, fluid nodes and structural nodes possess different species, but both are basically defined with their coordinates and number.

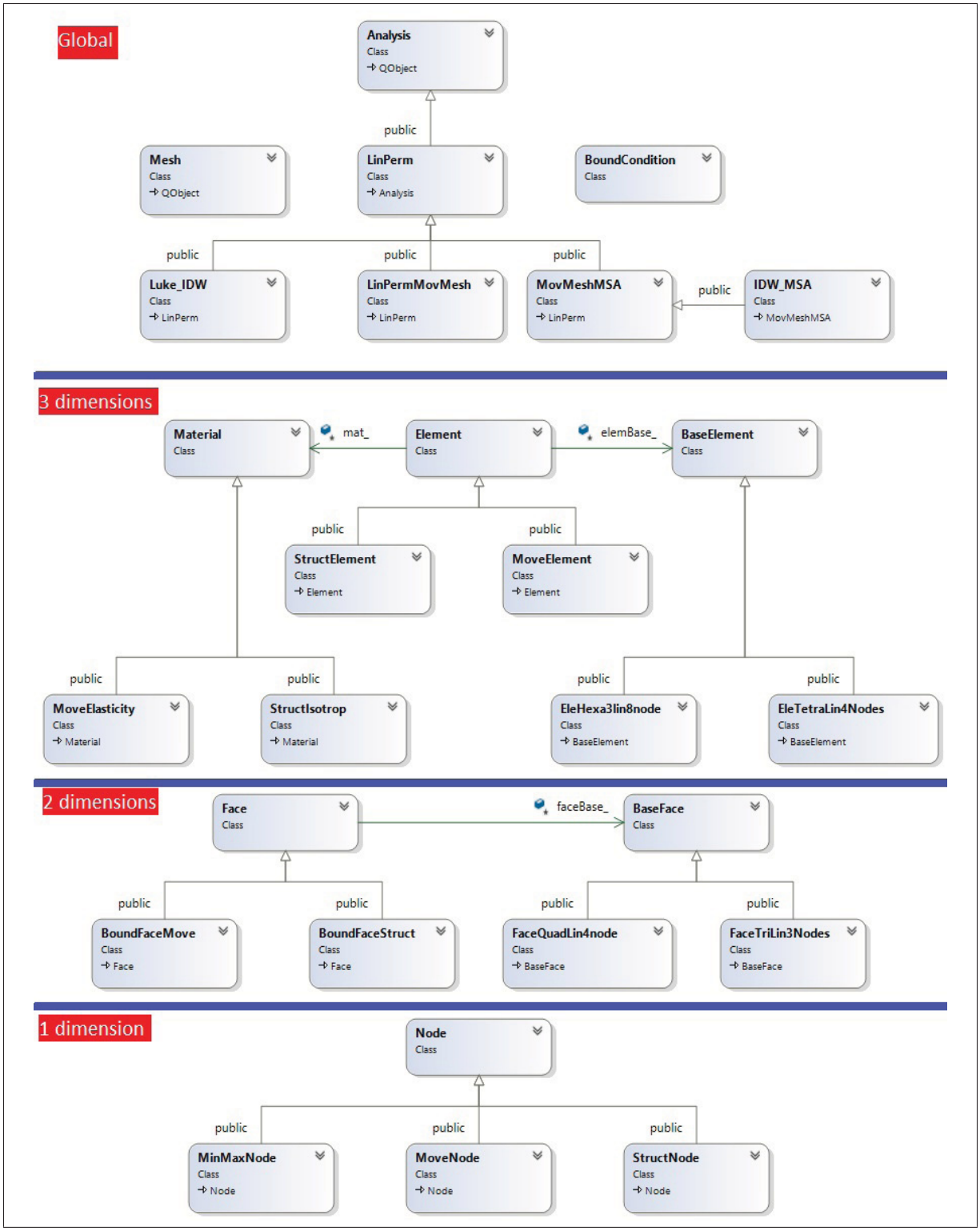



Figure 3.1 Classes of the code.

The FEM objects are defined by their attributes which are shown in Table 3.2 in the following manner:

Table 3.1 Example of presentation of attributes

| **(Name of the class)**: |
| --- |
| *(type of attribute) (name of attribute)* (short description) |

Usually child classes do not possess different type of attributes than their parents and to ease the lecture of the code each abstract class defines an *enum* which is a list of integers identified by a word. For this code, these *enum* are used to distinguish the type of children created from abstract classes. Also, some attributes which are not standard C++ object are from the downloadable Eigen library (Guennebaud *et al.* (2010)), in particular this library is used for matrix operation. Objects from the standard and Eigen library are used by calling their namespace, for example: std::vector<double> and Eigen::Vector3d.

Table 3.2 Abstract classes attributes

| **Analysis**: |
| --- |
| *enum analysisType (LINEARS = 0, LINEARUS, NLINEARS, NLINEARUS = 3)*<br>This enum allows the selection of different type of solvers: linear/non-linear and steady/unsteady. Each value is used to select the correct process to solve the PDEs system.<br>*analysisType type*<br>The identifier of the child class type.<br>*Mesh* Mesh*<br>This object gives information about connectivities, coordinates, gives access to related objects physical properties and the functions related to the mesh.<br>*Eigen::VectorXd matFGlobal*<br>The right hand side of the PDEs to be solved which is also called the external nodal forces vector. |

Table 3.2 Abstract classes attributes – (cont'd)

| |
|---|
| *Eigen::SparseMatrix<double> matKGlobal*:<br>The left hand side of the PDEs to be solved, also called the stiffness matrix which is assembled from all elements' stiffness matrix.<br>*Eigen::BiCGStab<Eigen::SparseMatrix<double>, Eigen::IncompleteLUT<double»* solver*:<br>The object which possesses all the functions needed to compute the solution of the FEM problem. The solver used here is the Bi-Conjugate Gradient Stabilized method to solve a preconditioned ILUT matrix. |
| **Node**: |
| *enum NODETYPE (MINMAX = -1, STRUCT = 0, THERM=10, ELECTMAGNET=20, MOVMESH = 30, ENCAP = 40)*<br>The possible type of node associated to the physical equations to be solved by the software. The MINMAX is a type of node created for the ADT structure, the MOVMESH is the type which adapt to moving boundaries displacements and ENCAP is a special MOVMESH nodes inside the encapsulation zone.<br>*NODETYPE type*<br>This identifier specifies which children class this object is.<br>*int index, index4solve*<br>The first index is from the mesh file and is the same one used while exporting resulting meshes. Then, *index*4*solve* is the index which regroups nodes with the same coordinates and those which are not used in the current analysis are skipped from that numbering. This renumbering process is useful to reduce the size of matrices to be solve.<br>*Eigen::Vector3d* coord, originCoord, middleCoord*<br>Those vectors of doubles contain the current coordinates, the coordinates at $t = 0$ and at $t = 0.5$, where $t = \frac{\text{current iteration}}{\text{last iteration}}$.<br>*std::map<std::string, double> ddl*<br>This map stores the unknowns of the current iteration and identifies them by a name. All types of nodes have different unknown, thus this map makes implementations versatile. |

Table 3.2 Abstract classes attributes – (cont'd)

| |
|---|
| *std::vector<int> indexOfElements*<br>There are the indices of all the elements which are composed or connected to the node. After an element is smoothed during the local smoothing we validate that no negative elements are generated through this list.<br>*int volID*<br>The mesh generator allows to give an identifier to each group of elements, the same index can be transferred to nodes and the selection is done by prioritizing the lowest index for volume interface nodes.<br>*double minQuality*<br>The minimal quality of connected elements. That value is used in the IDW mesh-mover to improve stiffness of nodes surrounded by low quality elements. |
| **BaseElement**: |
| *int gaussIntegPoint*<br>The quantity of Gauss points for the numerical integration of elemental matrices.<br>*std::vector<Eigen::MatrixXd> Bkez*<br>The matrices of the elemental reference derivatives of the shape functions for each Gauss points.<br>*std::vector<std::vector<double» N*<br>The vectors of the reference element shape functions for each Gauss points.<br>*Eigen::VectorXd wFromGauss*<br>The vector of weight for each Gauss points. |
| **Material**: |
| *int index*<br>The number identifying the material.<br>*std::map<std::string, double> constant*<br>The list of each constant name and value.<br>*Eigen::Vector3d vecFvol* |

Table 3.2 Abstract classes attributes – (cont'd)

| |
|---|
| The vector of volumetric forces values depending on the physical equations (ex: gravity). |
| *std::map<std::string,std::string> unitForUI* |
| The list of constants name and unit which are shown in the graphical user interface. |
| *Eigen::SparseMatrix<double> matD* |
| The material matrix assembled with the constants relative to each physical equation. |
| **Element**: |
| *enum NODES_PER_ELEMENT(EleTetra = 4, ElePyr = 5, ElePrism = 6, EleHexa = 8)* |
| The enum which allows to define the type of BaseElement according to the quantity of nodes. |
| *int meshIndex* |
| The identifier of the mesh domain containing the current element, also called volID. |
| *Node::NODETYPE type* |
| The type of node inside the element which defines the type of Element child class to use. Only one type is allowed per element. |
| *std::vector<Node*> nodes* |
| The vector of nodes which composes the element. |
| *Material* mat* |
| The material object which contains the constants and the material matrix used to construct the elemental stiffness matrix. |
| *BaseElement* elemBase* |
| The object which contains shape functions and derivatives of the reference element. |
| *int qtDDLelement* |
| The quantity of unknowns of the elements which is a sum of the unknowns of all nodes. It allows to define the size of the element matrices and vectors. |
| *Node* minNode;Node* maxNode* |
| The nodes created to define the bounding box of each element. They are used in the ADT structure. |

Table 3.2 Abstract classes attributes – (cont'd)

| |
|---|
| *double *quality, origQuality, q_chg, q_chgAbs*<br>The values of quality metrics as defined in the previous chapter.<br>*double stretchedTet_*<br>The value which indicates if the element should be smooth according to its original shape and size($t = 0$) or to be equiangular (see Section 2.5). |
| **BaseFace**: |
| *int gaussIntegPoint*<br>The quantity of Gauss points for the numerical integration of face matrices |
| **Face**: |
| *enum NODES_PER_FACES(FaceTri = 3, FaceQuad = 4)*<br>The enum which allows to define the type of BaseFace according to the quantity of nodes.<br>*int meshIndex*<br>The identifier of the parent mesh domain containing the current face, also called volID.<br>*Node::NODETYPE type*<br>The type of node inside the face which defines the type of child class to use. Only one type is allowed per face.<br>*std::vector<Node*> nodes*<br>The vector of nodes which composes the face.<br>*Material* mat*<br>The object material which contains the constants and the material matrix used to construct the face stiffness matrix.<br>*BaseFace* faceBase*<br>The object which contains the shape functions and the methods to generate matrices depending on the reference face.<br>*int qtDDLface*<br>The quantity of unknown of the face which is a sum of the unknowns of all nodes. It allows to define the size of faces matrices and vectors. |

Table 3.2 Abstract classes attributes – (end)

| |
|---|
| *double fSurf*<br>The value of the normal force per area applied on the face, the signification of this force depends on the physical equations (ex: pressure). |

The classes *Mesh* and *BoundCondition* have been omitted from the previous table to distinguish abstract classes from normal classes. These two classes inherit or share properties and functions from no other classes. Their attributes are shown in Table 3.3.

Table 3.3 Other classes attributes

| **Mesh**(standard attributes): |
|---|
| *struct type_star (fstream conec, noeud, cL, comm)*<br>When the input mesh format is STAR-CD there is four files with different extension to be read :(*.cel, *.vrt, *.bnd, and *.inp). Each contains different information: elements connectivity, nodes coordinates, boundary faces connectivity and the names of each mesh group (boundary and volume).<br>*int numberThread*<br>The amount of processors used for the OpenMP functions.<br>*Node::NODETYPE type*<br>The physical type of the mesh.<br>*int index*<br>The numerical identifier of the mesh.<br>*std::map<int,int> qtElementPerZone*<br>The list of the quantity of elements for each subdomain (volID) per index. This saved list is needed to save the output meshes in Tecplot format.<br>*std::map<int,std::string> blockNameAndIndex*<br>The list of names for each sub-domain (volume ID) by their index. |

Table 3.3 Other classes attributes – (cont'd)

| |
|---|
| *std::vector< Element* > elements*<br>The vector of elements of the mesh.<br>*std::map< int, Node* > nodes*<br>The list of nodes sorted by the mesh generator indices read.<br>*std::map<int, BaseElement*> baseElementPTR*<br>The list of all *BaseElement* possible for the mesh sorted by the quantity of nodes. Since, for each element the matrices of each *BaseElement* are the same, we compute them once and link each real element to its respective *BaseElement*.<br>*std::map<int, BoundCondition* > boundList*<br>The list of boundary groups sorted by their read indices.<br>*std::string path*<br>The complete input file path to access the mesh file.<br>*std::fstream logFile*<br>The file where output messages are printed. |
| **Mesh**(attributes for smoothing): |
| *enum scalingPreservProp(MEANEDGE = 0, VOLUME=1, MAXEDGE = 2, MINEDGE = 3, NOSCALE=4 )*<br>The possible types of element preserving quantity for the smoothing scaling step.<br>*std::vector< Element* > negativeElement*<br>The list of all negative elements generated.<br>*scalingPreservProp scalingType*<br>The selected smoothing preserving quantity.<br><br>**BoundCondition**: |
| *enum typeBoundCondition(NONE = -1, DIRICHLET = 99, FSURF = 1, FNODE = 2, CAUCHY = 3, SYMMETRY = 4,INTERFACE = 5)* |

Table 3.3 Other classes attributes – (end)

| |
|---|
| Boundary condition types which are used for each solving processes, they are read from the input mesh.<br>DIRICHLET represents imposing a value for the unknowns of the problem; FSURF represents the application of a force normal to surfaces and FNODE to nodes; CAUCHY means applying a flux to surfaces; SYMMETRY means that the value of unknowns parallel to symmetry faces are zero; INTERFACE is the identifier for faces used in the IDW method to represent a moving boundaries.<br>*std::string name*<br>The name of the boundary as it was read from the input mesh.<br>*bool onOff*<br>The Boolean which is *true* if a condition has been defined, if not the boundary is not considered in the solving.<br>*int type*<br>The type of boundary condition.<br>*std::vector<double> value*<br>A list of values which have different meaning for each type of boundary condition.<br>*std::map< int, Face*> boundFace*<br>The list of faces under this group of boundary condition. |

## 3.2 Functions needed from each object

Let us define what each class can do in terms of computation and delivering of information. First of all, it is imperative to define and retrieve attributes of each class; it is done with functions called getX() and setX(), where X is the attribute. The other less basic and primary functions are shown in Table 3.4, for all important classes. Child classes' functions are not shown since they are inherited, even if their implementation may differ.

Table 3.4 Major classes and their main functions

| **Analysis**(Implemented in all child classes): |
| --- |
| *Analysis(int type, Mesh* mesh)* |
| Constructor of the abstract class from a mesh and the type of child class. |
| *void removeNotActiveNode()* |
| Most meshes contain unnecessary nodes, thus a renumbering process is done. The renumbering consists in deleting duplicate nodes, deleting nodes not part of the physical domain and attributing other nodes a new index called $index4solve$. |
| *void assembleElements(Material* tempMat)* |
| The assembly of all element stiffness matrix is done to populate the $matKGlobal$ sparse matrix. |
| *void assembleFaces()* |
| The assembly of Cauchy and surface boundary conditions are added to the global system (matKGlobal and vecFGlobal). |
| *void setBoundariesValues()* |
| The imposition of nodal forces and unknowns are applied to the vector $vecFGlobal$. |
| *void linkNodeMoving2Deformed(Mesh* meshWithDef)* |
| For each moving mesh nodes a link to its similar node in the physical mesh is made. |
| *void setDeformationForMeshSolve(Mesh* meshWithDef)* |
| The contribution from the moving boundary to the moving mesh is imposed in the vector $vecFGlobal$ through the physical mesh. |
| *void solveAmesh()* |
| The function that solve the system $[K]\{U\} = \{F\}$ for $\{U\}$, then export values from vector $\{U\}$ to each nodes. |

Table 3.4 Major classes and their main functions – (cont'd)

| **Analysis**(Only for IDW classes): |
|---|
| *void prepareIDW(double gridMotionLref, double alphaFactor,double alphaFloor,double gridMotionAlpha)*<br>This function computes the rotation, translation and area of each moving boundary surface. Then, it computes $L_{ref}$ and $\alpha$ according to the function inputs or from the domain boundaries.<br>*void solveIDWExact()*<br>The displacements field is calculated from the inverse distance weighted average of all boundary nodes.<br>*void solveIDWApprox(double& qAvg, gridMotion::Symmetry sym = gridMotion::NONE)*<br>The displacements field is computed from the inverse distance weighted average approximation method proposed by Luke *et al.* (2012). |
| **Analysis**(Only for MSA classes): |
| *void findCoarseZone4Fine()*<br>Each fine mesh node is associated to a coarse mesh element that contains it. The search is done with a k-d tree based on the ADT approach.<br>*void calculateInterpolationFunction()*<br>The weights of each coarse element node relative to each fine mesh node are computed and saved for further use.<br>*void addDeformationCoarse2Fine()*<br>The displacements field from the coarse mesh is interpolated to the fine mesh. |
| **Mesh**: |
| *Mesh(Node::NODETYPE type, int index)*<br>A mesh is constructed from a type and index. Also in the constructor is initialized the *baseElementPTR* vector which contains all possible *BaseElement* for the mesh. |

Table 3.4 Major classes and their main functions – (cont'd)

| |
|---|
| *void readMesh(std::string inputfile)*<br>This function reads the mesh given as an input file. After the use of this function all the mesh attributes are initialized.<br>*void printTecplot(std::string ouputfile, int currentIt, int totalIt, std::string title = "Result" );*<br>This function saves the solved mesh under the standard Tecplot format. It contains all nodes coordinates, each node solved variables values, each element connectivity divided by group as it was in the input file and each element quality metrics.<br>*void setElementConnected2Nodes()*<br>Each node vector $indexOfElements$ are populated. These vectors are used for the local smoothing.<br>*void smoothMeshGlobally(const double& tolerance,const double& minMesh,const int maxIt, std::vector<double>& minQ, std::vector<double>& avgQ, int& finalIt, int typeQuality )*<br>This function smooths the mesh according to the method shown in Section 2.4. The method smooths until the input tolerance quality is attained, or until the maximum input iteration. The different types of quality are returned as minimal and average quality. The amount of iteration done is returned under $finalIt$. The type of quality to be improved is the last input parameter.<br>*void smoothMeshLocally(const double& tolerance,const double& minMesh, const int maxIt, std::vector<double>& minQ, std::vector<double>& avgQ,int& finalIt, int typeQuality )*<br>The smoothing of the mesh worse quality elements, as presented in Section 2.4, is done until the minimum input quality is attained, or until the maximum input iteration. The input parameters are the same as the global smoothing. |

Table 3.4 Major classes and their main functions – (cont'd)

| |
|---|
| *void meshUntanglerGETMe( const int maxIt, std::vector<double>& minQ, std::vector<double>& avgQ, int typeQuality )*<br>The vector of negative elements are smoothed, as presented in Section 2.7, until all elements are untangled, or until the maximum input iteration. The input parameters are the same as the global smoothing.<br>*void calculateQualityOfElements(std::vector<double>& minQ, std::vector<double>& avgQ, int typeQuality)*<br>The quality of each element is calculated as well as the average and minimal mesh values according to the metrics defined in Section 2.4. |
| **Node**: |
| *Node(NODETYPE type, int& index, Eigen::Vector3d& coord)*<br>Each node is created from their type, index and respective coordinates.<br>*void addMovement2node()*<br>(Only for moving mesh nodes) Current coordinates are modified by adding the displacements of each coordinate. This function is called when the displacements field is known.<br>*double distanceBTW2node(Node* toSub)*<br>The calculated Euclidean distance between the current node and the one given in parameter is returned. |
| **BaseElement**: |
| *BaseElement(int gauss)*<br>The construction of each *BaseElement* object or derived object is done by giving the number of Gauss points wanted for the numerical integration.<br>*void setBkezANDn()*<br>This function computes the values for each variables of the *BaseElement* (*Bkez*, *N*, and *wFromGauss*.) |

Table 3.4 Major classes and their main functions – (cont'd)

| **Material**: |
| --- |
| *Material(int index)*<br>The construction of each *Material* object or derived object is done by giving the index of the material.<br>*void initialiseDefMat()*<br>This function computes the matrix *matD*, which contains the material properties. Its use is not defined for the *IDWprop* material, since there is no PDEs system to solve with the IDW scheme. |
| **Element**: |
| *Element(Node::NODETYPE type, int& meshIndex,int& qtNode, std::vector<Node*>& nodes,BaseElement* elemBase)*<br>An element is constructed by its type of node, its volID, the quantity of nodes, the vector of nodes and the link to its *BaseElement*.<br>*void getMatXElement(Eigen::SparseMatrix< double >& matKelement, Eigen::VectorXd& matFelement)*<br>Returns the computed elemental matrix and vector according to its physical type.<br>*bool inBoundingSphere(Node* x)*<br>This function returns *true* if the node $x$ is inside the bounding box of the current element. This function is called in the ADT search algorithm before calculating if the node is exactly inside. This approximation is done according to the equation: $minNode \leq x \leq maxNode$ and it helps to speed up the MSA algorithm search.<br>*void calculateQuality()*<br>This function calculates the quality metrics of the current element. This function is called in $Mesh :: calculateQualityOfElements()$. |
| **BaseFace**: |
| *BaseFace(int gauss)* |

Table 3.4 Major classes and their main functions – (end)

| |
|---|
| The construction of each *BaseFace* object or derived object is done by giving the number of Gauss points wanted for the numerical integration.<br>*Eigen::SparseMatrix<double> getBKsiEta(const int currentGaussNode)*<br>Returns and computes the matrix of interpolation derivatives *Bke* for the number of Gauss node given in parameter.<br>*double * getNnum(const int currentGaussNode)*<br>Returns and computes the matrix of interpolations *N* for the number of Gauss node given in parameter. |
| **Face**: |
| *Face(Node::NODETYPE type, int& meshIndex, int& qtNode, std::vector<Node*>& nodes)*<br>A face is constructed by its type of node, its volID, a quantity of nodes and a vector of nodes. Also, for each face a *BaseFace* is created in the constructor.<br>*void getFeFace(Eigen::VectorXd& vectFeFace, std::vector<double>* valuesOfBoundCond)*<br>Returns and computes the face vector according to its physical type and the input boundary values. |
| **BoundCondition**: |
| *BoundCondition(std::string name, bool onOff, int type)*<br>Each boundary condition is constructed with their name and type. |

Some functions were not depicted, since their usages are parts of other functions which make them private. A private function cannot be used outside the class thus their definitions are not presented in this work.

### 3.3 Data structure and visibility

Each object functions presented previously can only be called by the current object or the object's owner, this concept is called visibility. In other words, a class can only access its attributes and its attributes functions. For example, an *Analysis* object only sees its associated *Mesh* and *Material* objects only have access to their attributes and functions. The visibility of each major class is shown in Figure 3.2.

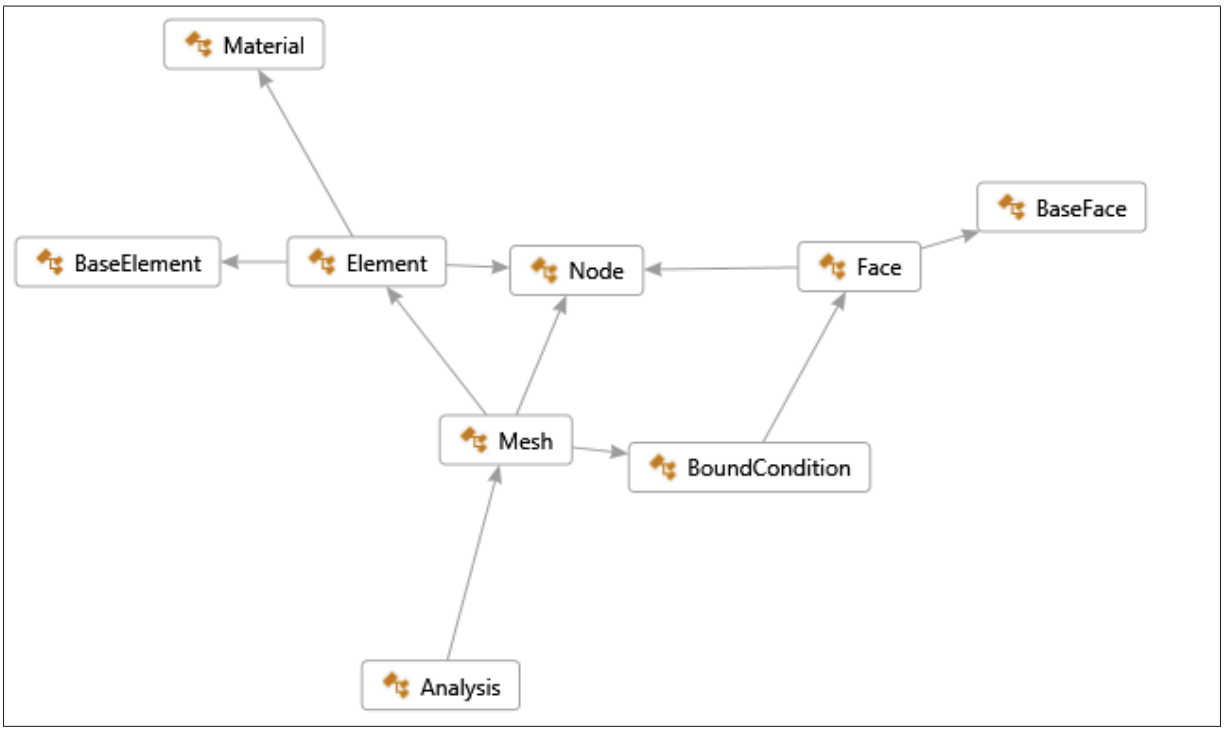



Figure 3.2 The visibility of each abstract class.

From this tree, the central classes are those of interest: *Analysis*, *Mesh* and *Node*. Where an *Analysis* is the mathematical method to solve a simple or complex problem, a *Mesh* is the grouping of all geometrical objects under analysis and a *Node* represents the data sought from the analysis. Classes around this core are merely tools to help in working with those classes.

As stated before the main goal of this code is to be understandable by users with basic knowledge of the FEM. Thus, we believe that the visibility tree and the information given in previous

sections make simple the understanding, usage and modification of the code content. Although, more specific information will be provided in subsequent sections.

## 3.4 Interfaces

### 3.4.1 Interface between objects

Each class interacts with the other in a different manner, but the standard is to regroup attributes and functions under three groups:

- Public: what is accessible by each class which sees the object;
- Protected: what is accessible by child classes;
- Private: what is accessible solely by the current class.

The programming strategy is based on the encapsulation paradigm of the object oriented programming which says that all attributes are hidden, thus private, and that the public functions are the interface to their attributes. However, since the inheritance is largely used in this work, we have decided to move the attributes to the protected group to ease programming and still respect the encapsulation rule.

### 3.4.2 User interface

To use such a code it is necessary to make it flexible for variable situations and options. Thus, two possibilities to use the program have been designed: a text file input or a Graphical User Interface (GUI). For the two proposed interfaces each process is divided in sections, or tabs which are:

- Physics: Structural tab (0), Thermal tab (10), and Electro tab (20);
- Moving Mesh tab (30);

- Message Passing Interface (MPI), or Multi-Processors Interface, tab (40);
- Options tab (50).

#### 3.4.2.1 Graphical User Interface Usage

Unless the analysis concerns solely one physical behaviour, we should always start by filling the Moving Mesh tab as seen in Figure 3.3. First, the coupling of physics is selected to allow the creation of the analysis for each physical problem. Concerning the study of MMA, selecting the structural coupling is sufficient because the mesh-movement is imposed through the structural tab. Then the mesh file is set by browsing through the computer, this allows to define the boundary conditions read from the mesh file. Finally, we select the MMA and fill its properties section which will be stored in a *Material* family class.

The next step is to configure each coupled physic for the current simulation. As an example, in Figure 3.4 is shown the structural tab configuration. Similarly to the moving-mesh tab, the type of analysis is selected, the boundary conditions are imposed and the properties of the material are set. Also, in this tab the moving boundary movements are defined per *volID* (box and fluid in this example). The possible movements are rotations around x, y, z or a custom axis, a translation, or specific movements defined in the code (for example the *NACA0012 - Bending*).

Additional options are defined as shown in Figure 3.5. The options to be set are the smoothing parameters, the number of iterations and the PDEs solver properties. Then, when everything is set, the relevant solve button in the bottom of the window should be pushed. After, each iteration the deformed mesh file is created in the same folder as the input mesh and a file containing calculation information is produced.

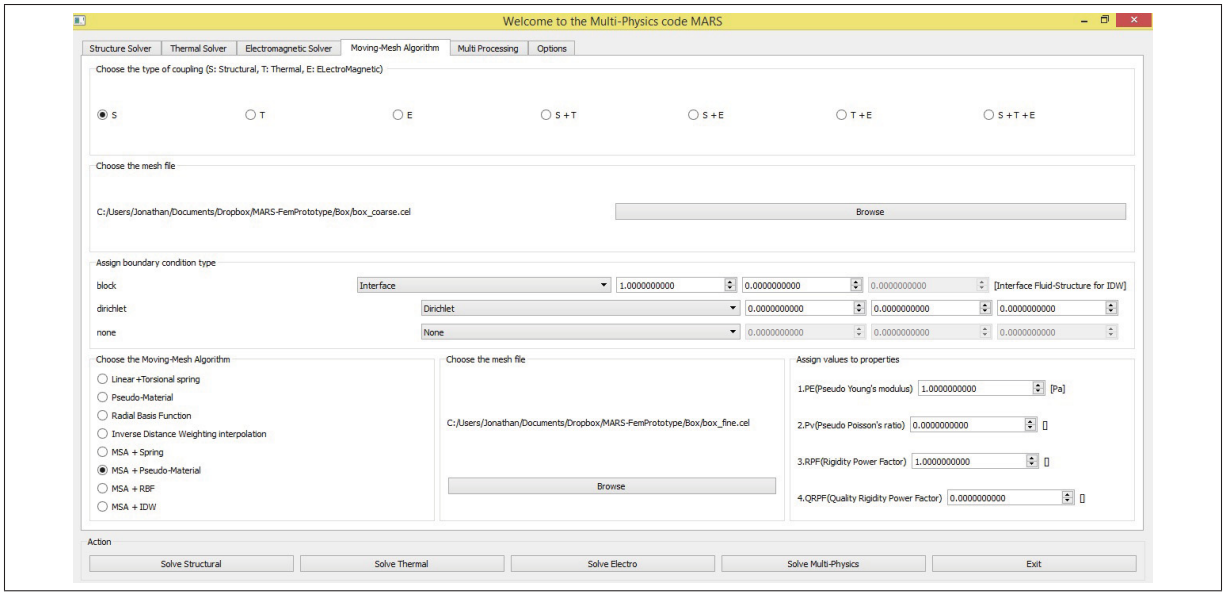



Figure 3.3 GUI: Moving Mesh Tab example.

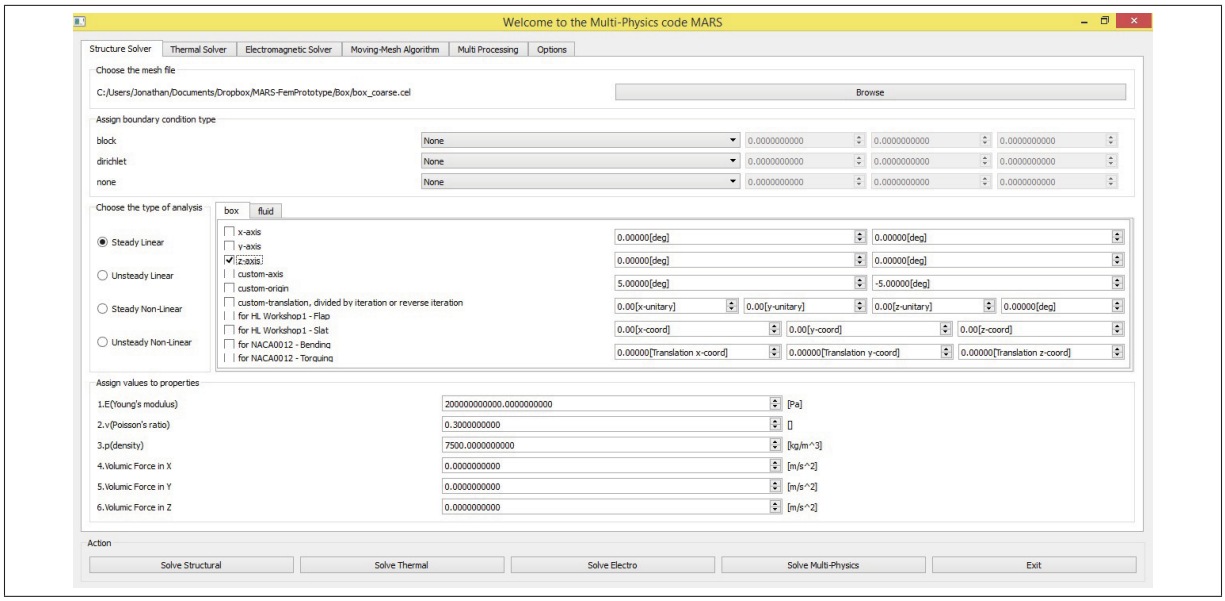



Figure 3.4 GUI: Structural Tab example.

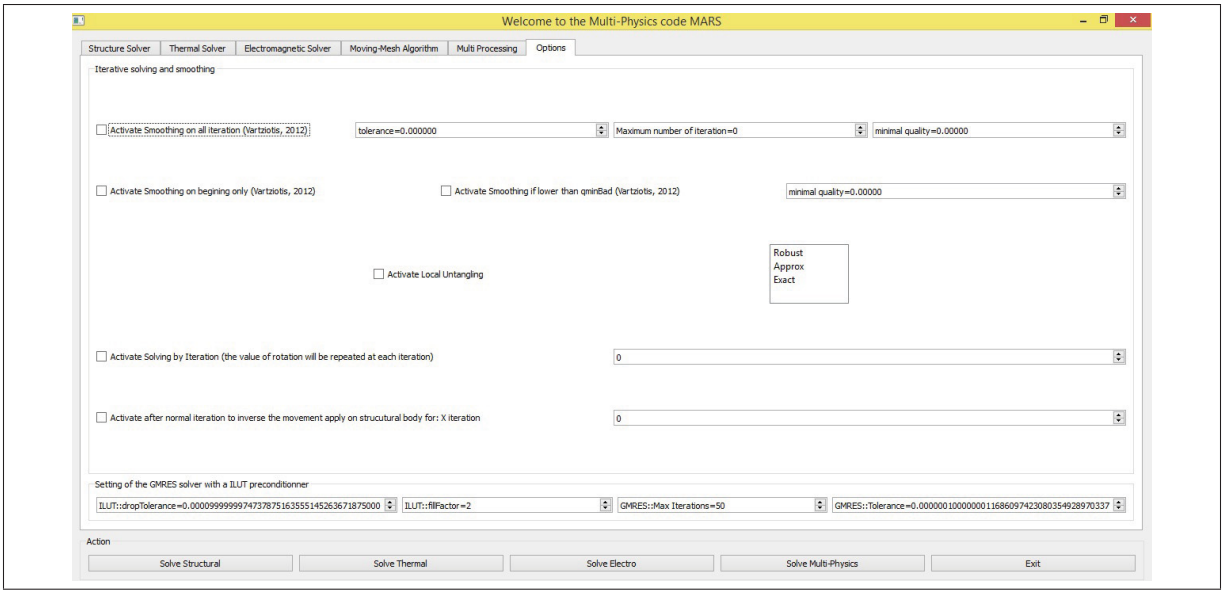



Figure 3.5 GUI: Options Tab example.

#### 3.4.2.2 Cluster Interface

Most complex problems are solved on clusters of computers which allow faster computations at the cost of more implementation development. In this optic, we have designed a text file template that can be read by the proposed program and be used in the context of MPI solving.

The structure is similar to the GUI except that the parameters are read from a single text file. In this text file, each tab is divided by identifiers *start* and *stop* and the options are called by their names before the new assigned value. An example of this file is represented in Figure 3.6. Each line that contains new information starts with identifier followed by the underscore ('_') character and when comments are written they begin with a double slash ('//').

```
// Template file for send a calculation of MMARS to the cluster//
start_ 0 //tab Structure
type_ 0 //for steady linear, others are UL(1), SNL(2), UNL(3)
mesh_  /sb/project/sks-412-aa/jlandry/hiLift4/HiLiftPW-Fine.cel //meshFile it cannot have space in the file
mat_ Isotrop 1 //the Type of Material and the index, by default it is not needed to enter the properties after
stop_ 0 //
start_ 10 //tab Thermo
stop_ 10 //
start_ 20 //tab Electro
stop_ 20 //
start_ 30 //tab Mesh-Mover
type_ 0 // for Struct coupling, other are T[1],E,ST,SE,TE,STE[6]
mesh_  /sb/project/sks-412-aa/jlandry/hiLift4/HiLiftPW-Fine.cel //meshFile
bnd_ 1 5 1.25 0.0 0.0 //interface slat
bnd_ 2 5 1.125 0.0 0.0 //interface wing
bnd_ 3 5 1.0 0.0 0.0 //interface flap
bnd_ 4 99 0.0 0.0 0.0 //symmetry faces, is dirichlet for meshMovement
bnd_ 5 99 0.0 0.0 0.0 //dirichlet, farField
//bnd_ 6 4 0.0 0.0 0.0 //symmetry aircraft body, creates too much neg elements if turn symmetry
bnd_ 7 99 0.0 0.0 0.0 //dirichlet, farField
bnd_ 8 99 0.0 0.0 0.0 //dirichlet, farField
    //NONE = -1,//DIRICHLET = 99, //FSURF = 1,//FNODE = 2,
    //CAUCHY = 3, //SYMMETRY = 4, //INTERFACE = 5
    //then values
//custom rot of slat
bndMov_ 1 3 0.559533629881678 0.005607428973512 0.828788678597711 -0.6 //custom rot Slat
bndMov_ 1 4 7.84097857343 -0.939174490713 6.90158652818 //custom origin Slat
bndMov_ 1 5 0.0 0.0 0.5 //translation to do not collide with fuselage
bndMov_ 2 5 0.0 0.0 0.5 //translation to do not collide with fuselage
bndMov_ 3 5 0.0 0.0 0.5 //translation to do not collide with fuselage
bndMov_ 3 3 0.346604875658787 7.607201571169524e-04 0.938010917566741 0.4 //custom rot Flap
bndMov_ 3 4 45.5600604621 -0.989323704892 6.90135121536 //custom origin Flap
//MMA type SPring[0],PSM,RBF,IDW[4], MSA-SP,MSA-PSM,MSA-RBF, MSA-IDW[8]
//algo_ 2 1 //with PSM 0 or nothing means material without poisson and 1 with poisson&shear effect,
algo_ 4 //idw
//algo_ 6 1  C:\Users\aj08230\Documents\3.DLR-F11\DLR-Fine-MSA-PSM.cel //exemple msa-PSM
//algo_ 8  C:\Users\aj08230\Documents\3.DLR-F11\DLR-Fine-MSA.cel //exemple msa

//props to modify PSM
//props_ 1.E 1.0 //no effect it will just multiply all rigidity, usefull if multiple material
props_ 2.v 0.0 //tranfer axial to transversal, but does not converge
props_ 3.RPF 1.0 //exponent that increase stiffness of smaller element
props_ 4.QRPF 2.0 //exponent that increase stiffness of distorded element
//props to modify IDW
props_ 1.ExpA 4.0 //
props_ 2.ExpB 5.0 //
props_ 10.ExpC 0.0 //for quality Dependent movement
props_ 4.Lref 20.0 //lref=half meanchord
props_ 6.alphaFloor 0.001 //
props_ 7.gridMotionAlpha 0.001 //
props_ 3.GridMotionErrorTol 0.1 //property of the algorithm,example GridMotionErrorTol
//props_ 9.ExactorApprox 0 //for exact 1, 0 for off
props_  11.UpdateIDW 1  //1 for updateIDW on, 0 for off
stop_ 30 //
start_ 40 //tab MPI
//np_ 16 // number of processor for solve, options solely for OPEN-MPI version
stop_ 40 //
start_ 50 //tab Options
//for smooth:smoothItBool, then tol(==0.0 skip global mesh),
//it,minQc, smoothBeginBool, badBool, qBad, typeOfQuality2Smooth(qchg[0],q,qchgAbs[2]
smooth_ 0 0.0 2 0.25 1 0.125 2.0 //
//for smooth begin of q[0]: tol, it and qmin
//smoothB_ 1e-3 10 0.25 //no begin because it makes the mesh more fragile to become invalid
untgl_ 3 100 2 //untangling tool, Robust[0], Approx and Exact[2], getMe based[3], untIterations
//stretchedAngles_ 6.0 20.0 //min included angle to recognize t-rex layers stretched tets
//stretchedQuality_ 0.0625 0.0625 //the quality value to define t-rex layers stretched tets
stretchedDistance_ 0.05 0.05 //for recognized stretched elements of t-rex
it_ 50 //number of iteration
rit_ 50 //number of reverse iteration
//printStarCD_ 3 0 50 100 //for print coordinates after mesh-motion: qtPrint, list of iterations to print
printCFX_ 3 0 50 100 //for print coordinates after mesh-motion: qtPrint, list of iterations to print
printTecPlot_ 3 0 50 100 //for print mesh in tecplot format put nothing print at all iterations
BiCGStab_ 1e-4 2 5000 1e-7 //GMRES-ILUT setting: dropTol,FillFactor, maxIt, Tol
stop_ 50 //
calcul_ 3 //0 for struct, 1 for therm, 2for electro, 3 for multi
stopFile_ //
```

Figure 3.6 Cluster text input file example.

## 3.5 Implementation

The last section of this chapter will cover details and comments on the implementation of the code which will be useful for future developers. As well some implementations from the proposed interface will be shown.

### 3.5.1 General information

In previous chapters and sections all the necessary equations, algorithms, variables and strategies to understand the computer code was presented; however few additional details of our design will be shown.

Firstly, objects which could be used by multiple entities are created in the pointer form. This means that each object entity knows the address where each pointed attribute is stored in the memory without possessing directly the attribute. This abstraction is represented by a star (*) in programming language. The advantage of using pointers is the possibility of using polymorphism which allows pointers of a parent class to represent any of its children class. Thus, it is mandatory to use this method to generalize the code and facilitate its understanding.

### 3.5.2 Finite Element Method

All classes, used for solving PDEs, are implemented with an approach similar to FEM solvers. To show our idea, in Algorithm 3.1 is solved the displacements field with the PSM, also called *LinPermMovMesh*, and a linear solving strategy.

The Algorithm 3.1 is part of the interface class *Principale* under the function *SolverLINEARS*. It consists of creating an object *Analysis* for linear steady problem. Then, the number of processes used by the functions of this class is set. Afterwards, the nodes which are duplicated or not necessary are removed.

Algorithm 3.1 PSM linear solver

**PSM linear solver**
**Input** : A mesh ($*mesh2Solve$) and a material object ($*mat$)
**Output**: An integer

```
1 Analysis* solver
2 solver = new LinPermMovMesh (mesh2solve)
3 solveur.setNumberThread(numberofthreadforsolving)
4 solveur.removeNotActiveNode()
5 solveur.assembleElements(mat)
6 solveur.assembleFaces()
7 solveur.setBoundariesValues()
8 solveur.solveAmesh()
9 return 0
```

Then, as done in all finite element algorithms, each element contributions, each flux per faces and each imposed degree of freedom from boundaries are assembled into global system matrices. Finally, the system is solved before returning the zero integer.

### 3.5.3 Eigen Library

To be able to solve systems of PDEs various objects and functions are needed to be designed and they are from the widely used Eigen library of Guennebaud *et al.* (2010) version 3.2. The principal objects used in our project are:

- MatrixXd: a matrix of variable dimensions for doubles.
- VectorXd: a vector of variable dimensions for doubles.
- SparseMatrix: a vector of the non-zero coefficients and a vector of their indices in the matrix.
- BiCGStab: the object to solve systems with sparse non-symmetric matrices systems of equation using the bi-conjugate gradient stabilized algorithm (Sleijpen and Fokkema (1993)).

The popular GMRES algorithm (Saad and Schultz (1986)) is provided but not used because it is not supported from the Eigen library developers. Finally, to understand better the Eigen library it is recommended to read the documentation concerning each of the above objects.

### 3.5.4 Parallelisation

For large scale problems, repetitive operations cause the resolution to be slow. However, it is recommended to distribute work load of repetitive task onto different computers, or processors, to accelerate the generation of result. The paradigm which allows calculation loads to be distributed and to share information between different processors is called the Message Passing Interface (MPI).

Thus, the code is written to work in conjunction with the standard MPI library. A user who wishes to modify this version of the code should at least have basic knowledge of this library since the MPI version is totally parallelized. The particularity of this version is that the mesh is divided on each processor, except for mesh boundaries which are possessed by all. Also, concerning the MSA implementation each processors possess its own coarse mesh; this approach is memory consuming and should be modified in a future version of the code.

### 3.5.5 Interface implementation

The first part of the interface, as shown in Algorithm 3.2, is the main file where the *Principale* object is created as a GUI or cluster interface. This algorithm activates the MPI methods and it creates the proper *Principale* object according to the quantity of arguments *argc* used to start the application. When $argc == 1$, only the name of the program is sent, thus the GUI object is created and the program is not ended until the exit of the GUI window. Although, if an input file is sent ($argc == 2$) this file is read and its information is used to start the calculation.

Algorithm 3.2 Main function of the code

**Main function of the code**
**Input** : An integer (*argc*) and a chain of character (argv[])
**Output**: An integer

```
1  MPI_Init(&argc,&argv)
2  proc_info.init()
3  QApplication app(argc,argv)the application is created
4  if argc = 1 then
5  |   Principale fenetrePrincipale
6  |   return app.exec()
7  end if
8  else
9  |   if argc = 2 then
10 |   |   Principale fenetrePrincipale(1)
11 |   |   fenetrePrincipale.readInput4Cluster(argv[1])
12 |   |   fenetrePrincipale.launchCalcul(fenetrePrincipale.getParam4Cluster().calcul)
13 |   end if
14 end if
15 MPI_Finalize()
16 return 0
```

The last implementation to present is the most complex: the function launchCalcul. This function, for the GUI interface, is launched by pushing a button and for the cluster interface by the main file (as seen in Algorithm 3.2). In Algorithm 3.3 is presented the launchCalcul function which represents the whole proposed solving process, where solely the section related to the mesh movement loop: the physical solvers which can be inside or outside the loop, the necessary variable initializations and the various window printing or file recording steps are not shown.

The function *smoothingLoop*() consists of smoothing globally, then locally for *itSmooth* iterations. Each global smoothing step stops when the average quality change is less than the input tolerance, usually around $10^{-5}$, and each local smoothing step stops when the wanted minimal quality is recovered or after five iterations without improving the minimal quality.

Moreover, after the untangling section, we should note the importance of recalculating the interpolation functions to ensure good interpolations while using MSA type algorithms. Also, the function *solverMeshMove*() is the direct use of the MMAs presented in Section 2, thus it creates the object *Analysis* moveSolver* of the desired type.

Finally, all functions not covered in this section should be seen as black boxes to most users since they have been validated for the various test cases presented in the following sections.

Algorithm 3.3 Mesh-movement resolution

**Mesh-movement resolution**
**Input** : An integer *calculType* which informs on the physical coupling
**Output**: No output

```
1  (Initialization section)
2  for iteration = 1 to lastIteration do
3      (Initialization section related to smoothing and untangling)
4      if iteration = 1 and smoothOnStart = true then
5          smoothingLoop(0)  0 for quality, 1 for q_chg and 2 for q_chgAbs
6      end if
7      SolverMeshMove(deformedMesh, materialMMA)
8      smoothAfterUntangle = false
9      if untanglerOn =true and qMin < 0 then
10         coarseMesh- > meshUntangler()
11         moveSolver- > calculateInterpolationFunction()
12     end if
13     if meshFine ≠ 0 then
14         if untanglerOn and qMin < 0 then
15             meshFine- > meshUntangler()
16             moveSolver- > calculateInterpolationFunction()
17         end if
18     end if
19     if smoothAllIterations = true or (smoothIfBad = true andqMin < qBad) then
20         if qMin < qMinRequired then
21             smoothingLoop(qualityType2smooth)
22         end if
23     end if
24     (The results and meshes are recorded.)
25 end for
```

# CHAPTER 4

# MESH-MOVERS ANALYSIS AND VALIDATION TESTS

Two simple test cases are used to validate and investigate limitations of the presented MMAs. In the first, large rotation and translation motions are imposed to a fluid immersed solid block to study the influence of MMAs parameters, to ensure that meshes under large displacements are kept valid and to investigate the influence of coarse mesh elements size for MSA type algorithms. In the second, the case of a cylinder approaching a 'Γ' shape is used to explore the limitations of multi-body interaction and to evaluate how boundary layers elements quality can be preserved with a rigid zone called 'encapsulating zone'. Finally, the smoothing algorithms and the Novel GETMe Untangler (NGU) are tested to validate and quantify their improvement for MMAs.

## 4.1 Rotating and translating box

A block of $2.5 \times 1.0$ is rotated 60 degrees around the 'z' axis and translated -10.0 units in both x and y directions ($t = 0.5$), then it is moved back to its original position ($t = 1$). The whole simulation, from $t = 0$ to $t = 1$, is divided in 20 steps. The elements directly on the moving block are of $1.7 \times 10^{-2}$ height and elements are grown from this first layer with a ratio of 1.3 for 20 layers until the end of the $50 \times 50 \times 1$ domain. The mesh is solely composed of tetrahedron, thus boundary layers for this mesh are structured tetrahedron. The original mesh is shown in Figure 4.1.

### 4.1.1 PSM parameters

There are two parameters to study concerning the standard PSM algorithms: the pseudo Poisson's ratio $\nu$ and the rigidity power factor $p$. Modifying $E$ results in multiplying the system of equations with the same constant, thus it makes no change at all since all elements are considered having the same $E$.

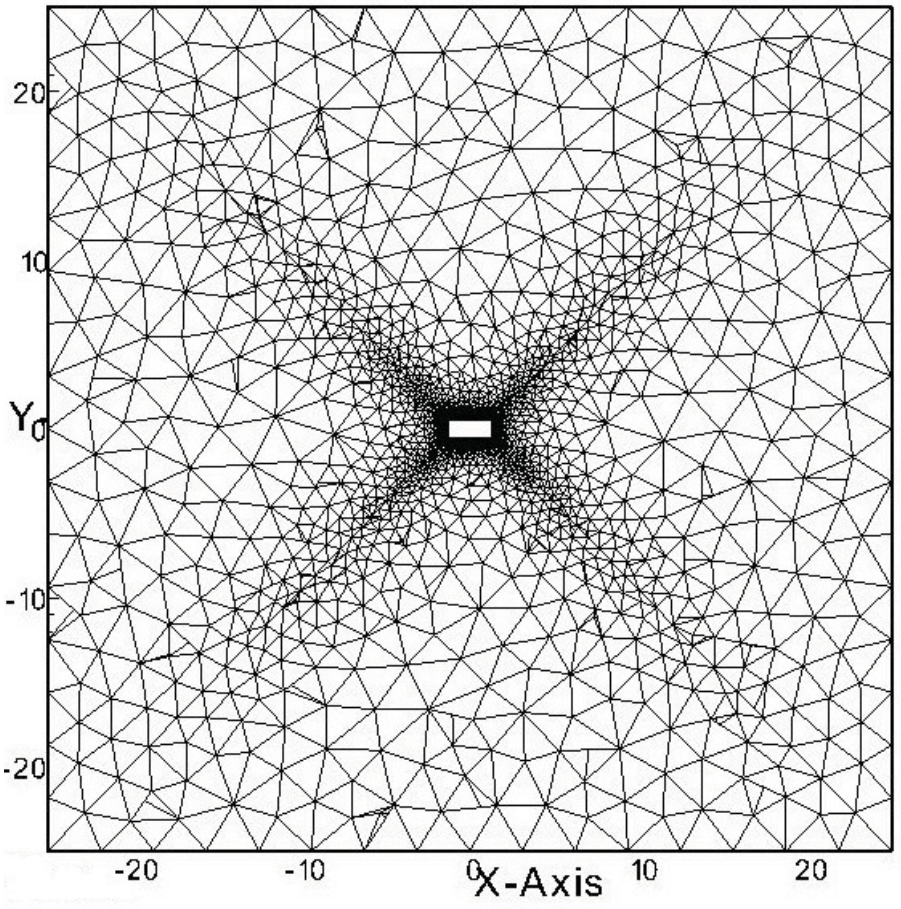
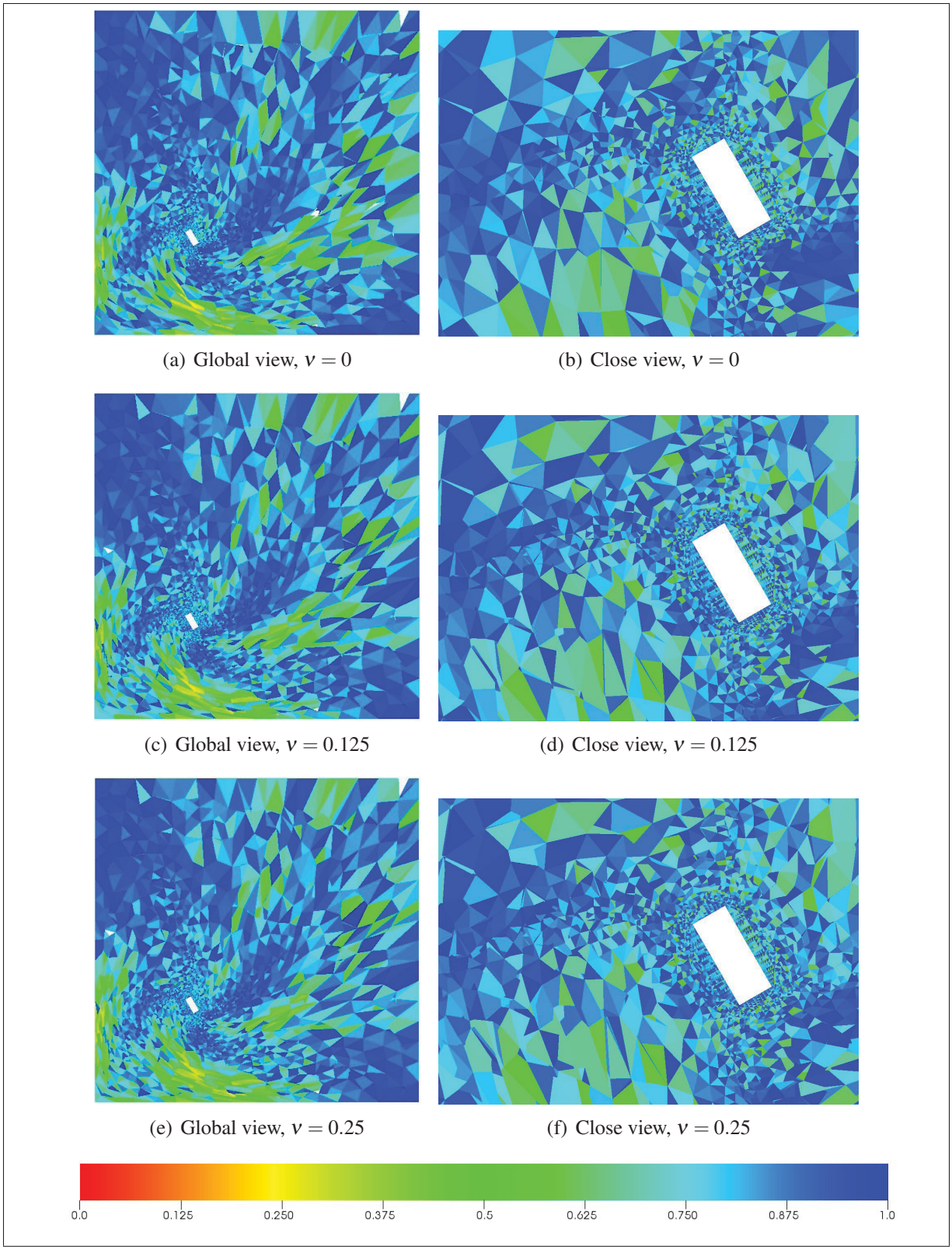



Figure 4.1 Large motion problem: mesh at t=0

The PSM study starts by changing $\nu$ from 0 to 0.45 (with $p = 1$), but only for the material (matrix $[D]$) which allows shear strains since the material without shear strains does not converge. Materials with values of $\nu$ close to 0.5 are called incompressible and those materials with the current formulation are unsolvable since there is a division by zero. To solve this situation a penalty method could be used (Masud and Hughes (1997)), although it is not necessary since meshes are kept sufficiently good with values up to 0.45. In Figures 4.2 and 4.3 are shown fields of $q_{chg}$ and their respective close views for different $\nu$ at final position. Then, in Figure 4.4 is shown the evolution of quality.

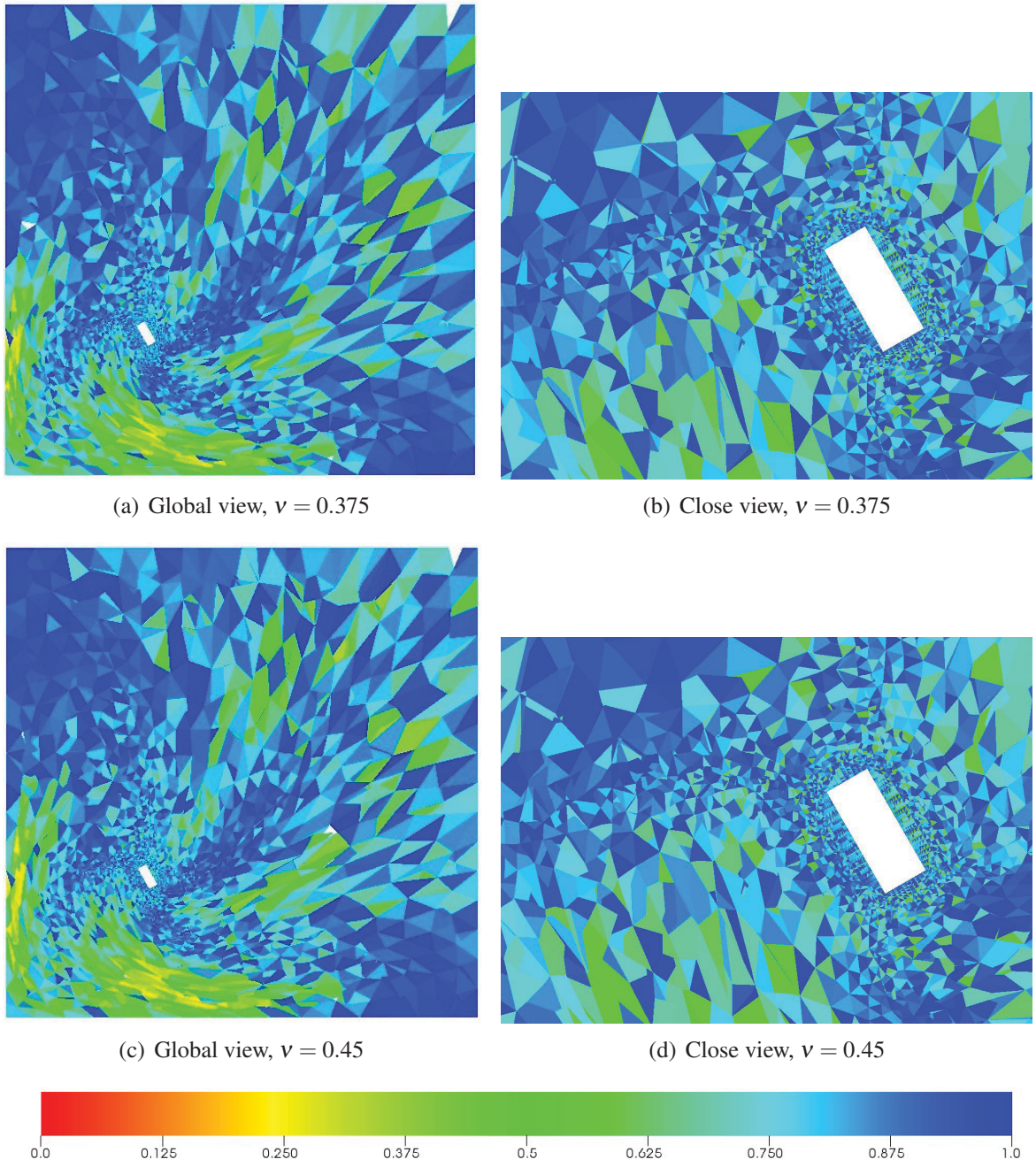



(a) Global view, $\nu = 0$ (b) Close view, $\nu = 0$

(c) Global view, $\nu = 0.125$ (d) Close view, $\nu = 0.125$

(e) Global view, $\nu = 0.25$ (f) Close view, $\nu = 0.25$

Figure 4.2 Large motion problem: $q_{chg}$ field for various $\nu$ at $t = 0.5$.




(a) Global view, $\nu = 0.375$

(b) Close view, $\nu = 0.375$

(c) Global view, $\nu = 0.45$

(d) Close view, $\nu = 0.45$

Figure 4.3 Large motion problem: $q_{chg}$ field for various $\nu$ at $t = 0.5$ (cont'd).

Figures 4.2 and 4.3 show that few elements have been highly deformed ($q_{chg} < 0.25$) and most of them have between the same or half their original quality ($0.5 < q_{chg} < 1.0$). Especially, the highly deformed elements are far from the moving boundaries, thus close to static boundaries. This is caused by the facts that highly deformed elements are of larger volume and on domain limits. This method with $\nu > 0$ allows the displacements of moving boundaries to be transferred in all directions. We can notice, from Figure 4.4, that the average quality is similar for the tested values of $\nu$, but the minimum is higher for $\nu = 0.25$. This better mesh is caused by the fact that displacements are distributed equally in all directions with $\nu = 0.25$.

Afterwards, the comparison of results for different values of the inverse volume stiffening exponent $p$ ($p = \{1,2,3\}$) are studied and results are shown in Figure 4.5 with $\nu = 0.25$. Then, in Figure 4.6 is drawn the evolution of $q_{chg}$ for different $p$ ($p = \{1,2,3\}$) and the pseudo-material which does not allow shear strains. Simulations with $p = 0$ are skipped, since the solution diverges after few small movements.

The results show that an exponent $p = 3$ always generates a bad mesh that leads to inverted elements($q_{chg} < 0.0$). The mesh minimal quality for $p = 2$ is improved for a material without shear strain, but not for the other material proposed. Thus, the exponent $p = 1$ seems the best compromise for a robust MMA. Exponent $p$ controls the difference between stiffness of elements, large values can generate jump of displacements between neighbour elements which is undesirable, thus the exponent $p$ must stay equal to one to give good quality meshes. However, making a zone of the mesh stiffer can contribute in keeping a better mesh, especially for elements close to moving boundaries as proposed by Stein *et al.* (2004). This option will be evaluated in Section 4.2.1.

Our proposed modification to the standard linear isotropic material matrix $[D]$ has been studied and has been shown acceptable, but not better than the standard one(as seen in Figure 4.7). It can be imagined that different pseudo-materials can be designed to distribute better displacements, but we have decided to stop the pseudo-material investigation here, since the results are satisfactory.

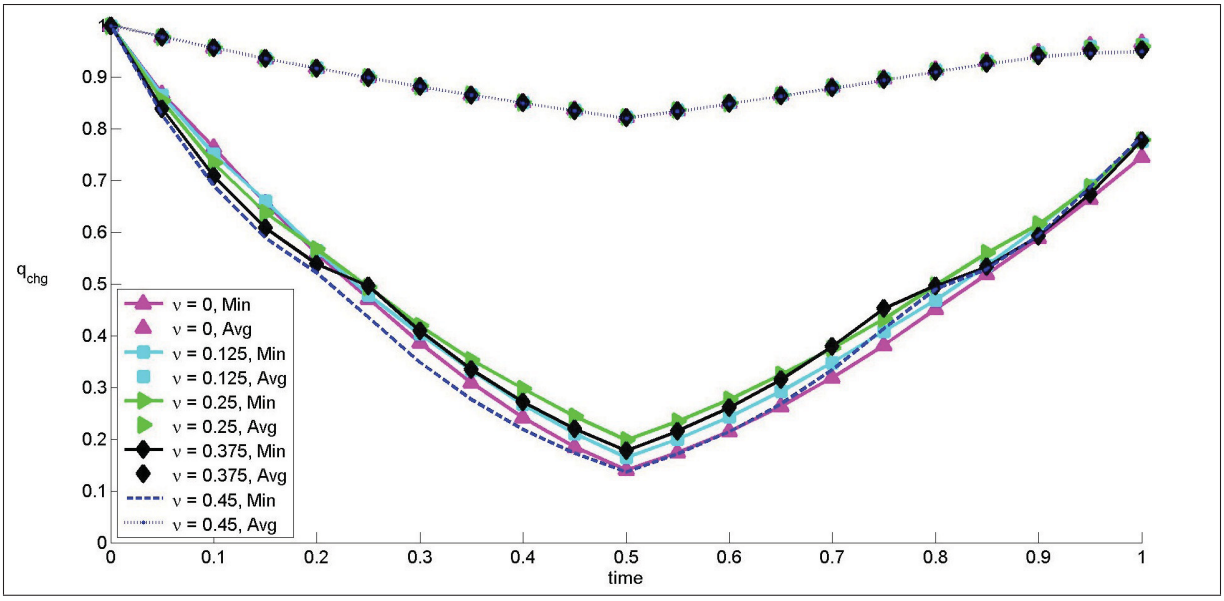



Figure 4.4 Large motion problem: $q_{chg}$ evolution for different $\nu$ with p=1.

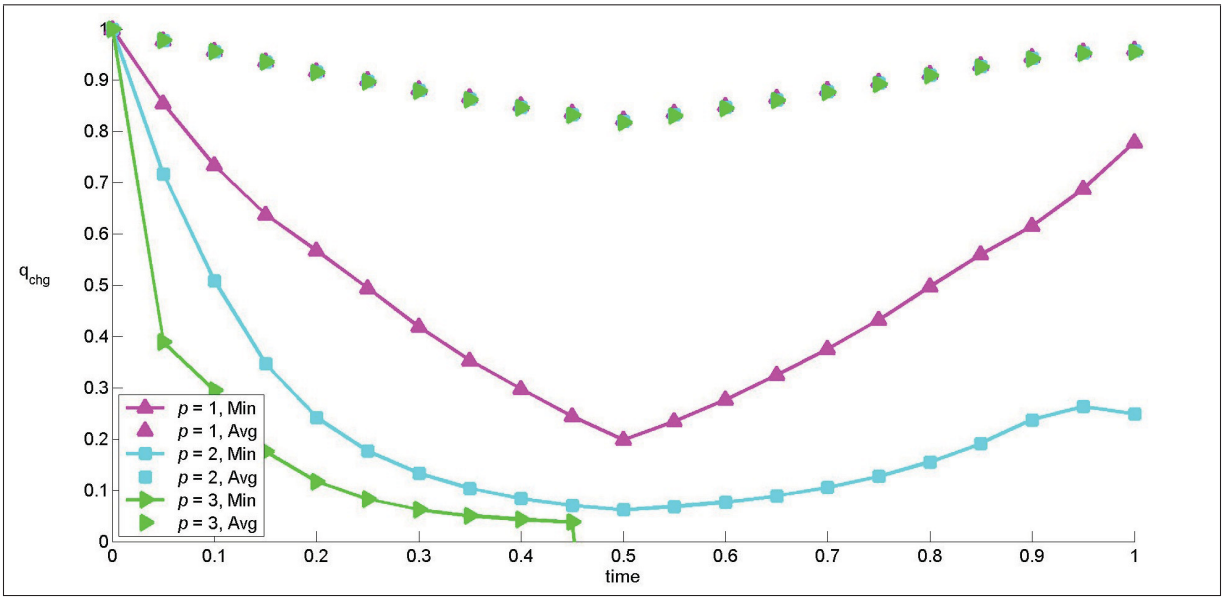



Figure 4.5 Large motion problem: $q_{chg}$ evolution for different $p$ with $\nu = 0.25$.

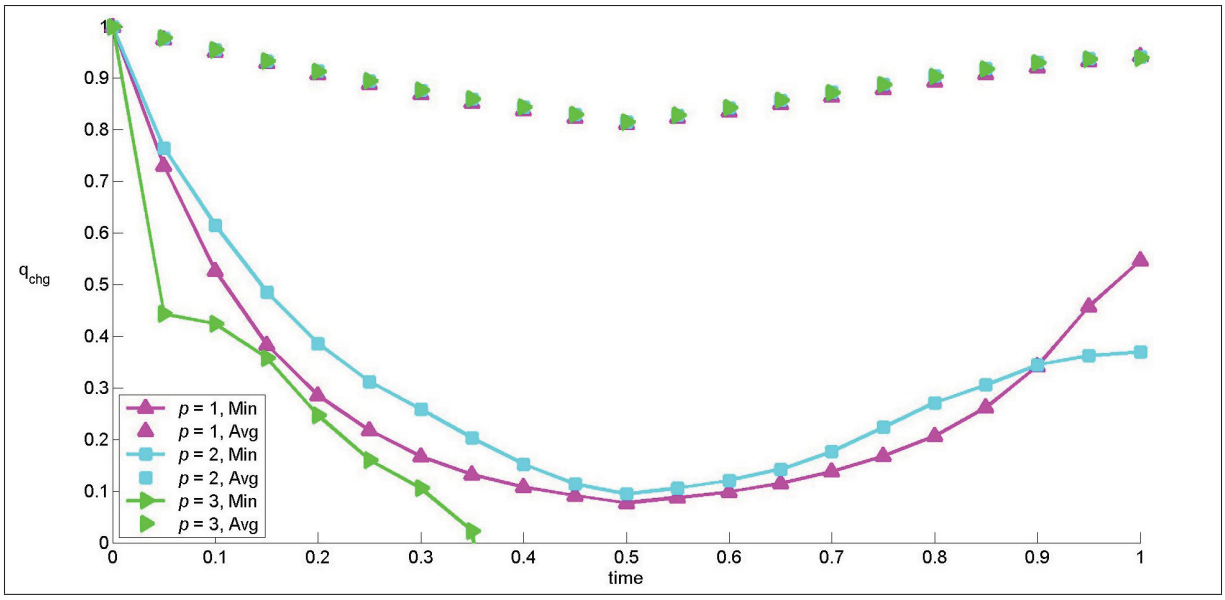



Figure 4.6 Large motion problem: $q_{chg}$ evolution for different $p$ and no shear strain allowed.

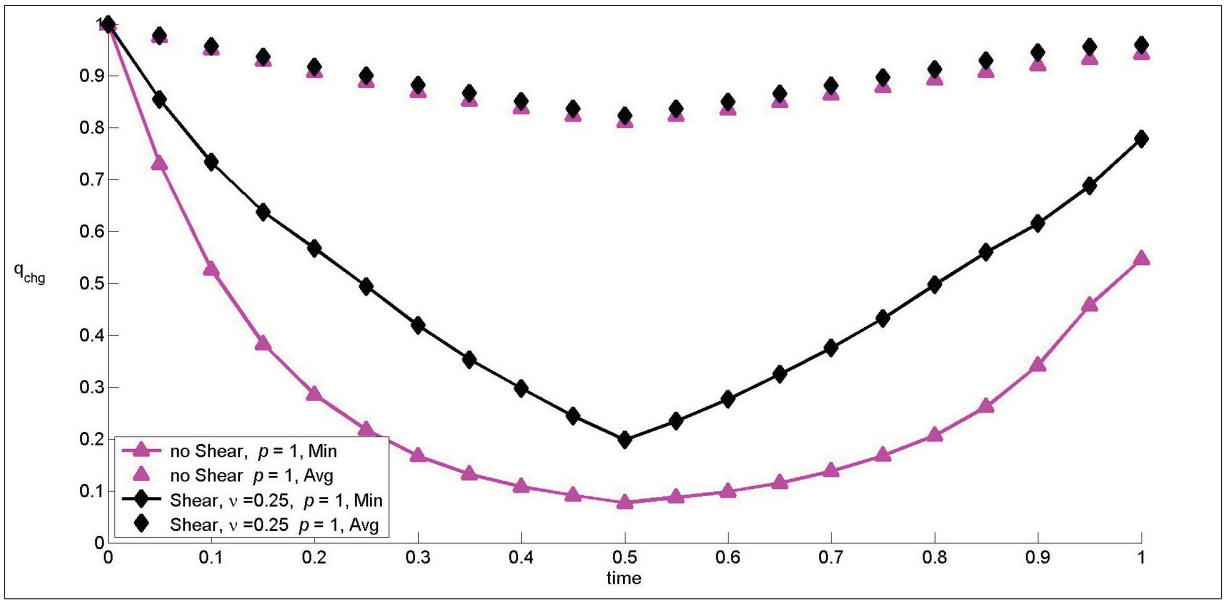



Figure 4.7 Large motion problem: Pseudo-material comparison.

### 4.1.2 PSM quality criteria

In Figure 4.8 is demonstrated the quality for different values of $pq$ (see Section 2.6), the exponent for the inverse quality multiplier, with the pseudo-material that allows shear, $\nu = 0.25$ and $p = 1$.

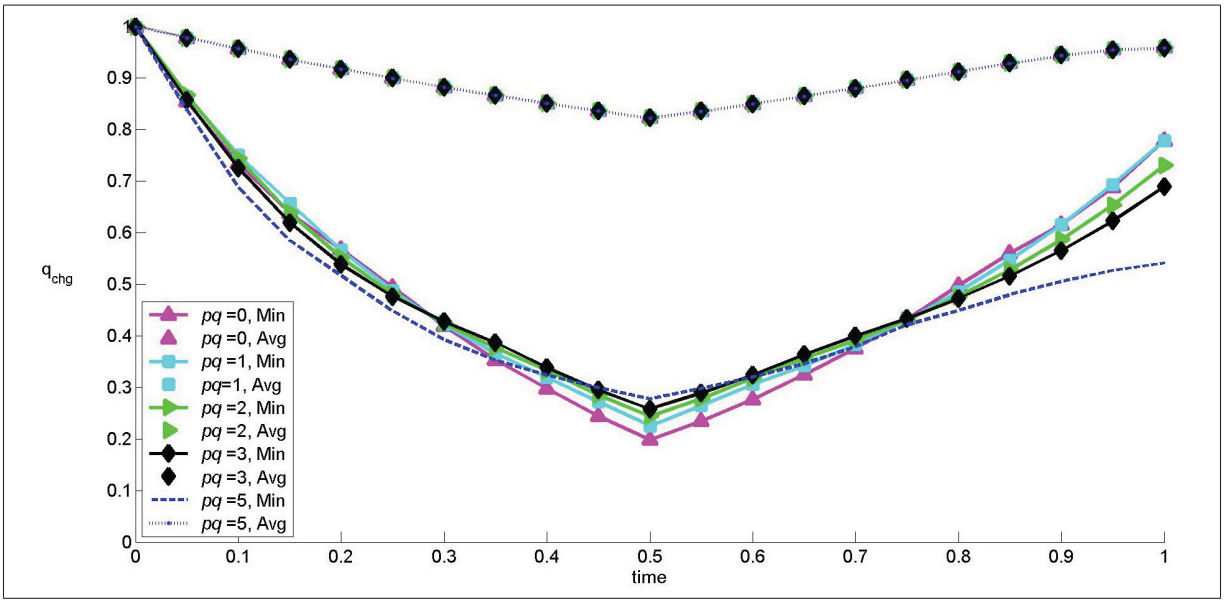



Figure 4.8 Large motion problem: $q_{chg}$ evolution for different $pq$ values.

The use of a quality stiffener exponent alone is not sufficient to move meshes (the solved mesh is invalid after one iteration), the volume dependent stiffener is always needed. Although, the addition of an inverse quality multiplier definitely helps in keeping elements from becoming inverted.

The proposed multiplier increases the minimal quality considerably with an increasing $pq$ exponent, but there is a drawback: it resists deformations that would increase quality (as seen when $t > 0.5$). This disadvantage is greater for a value of $pq = 5$ as it can be seen on Figure 4.8, where at final position(t=0.5) the minimum is considerably higher, but on the way back the minimum does not increase as much as for other exponent values; the minimum with $pq = 5$, at t=1, is even lower than with $pq = 0$.

**Recommended PSM settings:**

To summarize this section, optimal meshes are produces for $p$=1, $\nu = 0.25$ and a pseudo-material that allows shear. Also, we recommend values of $pq$ to be: $1 \leq pq \leq 3$. This recommendation increases robustness of the PSM method at the price of a small decrease in overall quality.

### 4.1.3 IDW parameters

The previous mesh has been tested for the IDW scheme. After, few iterations negative volume elements were generated. In order to correct this situation a larger domain of $500 \times 500$ has been used. This necessity of a larger domain for large displacements is caused by the fact that moving boundaries displacements are distributed on nodes, but stopped at the domain limits. If those limits are close to moving surfaces, some elements can become negative. Thus, increasing the distance of distribution contributes in keeping a valid mesh for the complete simulation.

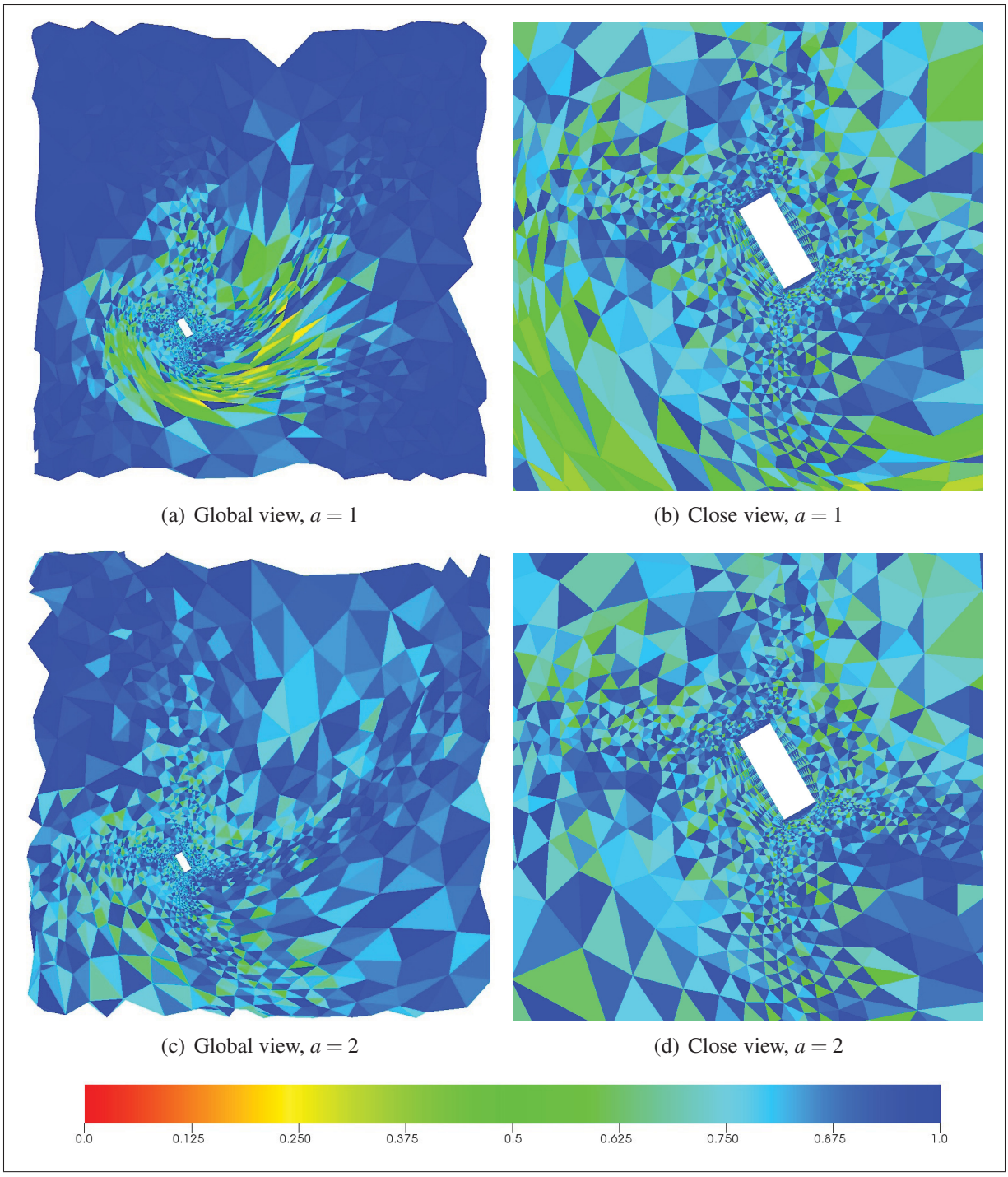
In Figures 4.9 and 4.10, the influence of exponent $a$ is shown for large displacements. Lower values of $a$ ($a < 3$) generate a displacements field concentrated around the moving boundaries and for higher values ($a > 2$) the displacements seem to be distributed more equally through the domain. In Figure 4.11, results of the minimal quality show that a value of $a = 1$ is not suitable for mesh movement and when exponent $a = 5$ a badly deformed mesh is produced. As for the other values, it is not possible to express a recommendation. However, in Figure 4.12 it can be seen that exponent $b = 5$ produces the worst mesh whereas other values produce similar results. This behaviour means that the exponent $a$ influences more the overall mesh quality than $b$, except for $b = 5$. Then, in Figure 4.13 it is presented that exponents equal to one or five generate bad meshes and that the deformed mesh is more dependent on the exponent $a$. It seems that exponent $b$, from those graphics, is not needed, but for complex cases it should help elements close to boundary from being largely distorted.

From those results, we recommend the exponents combinations:

- $3 < a < 4$;
- $3 < b < 4$.

It is possible to have a good mesh for $a = 2$ with $b = 5$ and $a = 4$ with $b = 5$. Also, we should keep in mind that Luke *et al.* (2012) has recommended the use of $a = 3$ with $b = 5$ for most cases, but exponents value may be modified to obtain optimal results for each specific case.

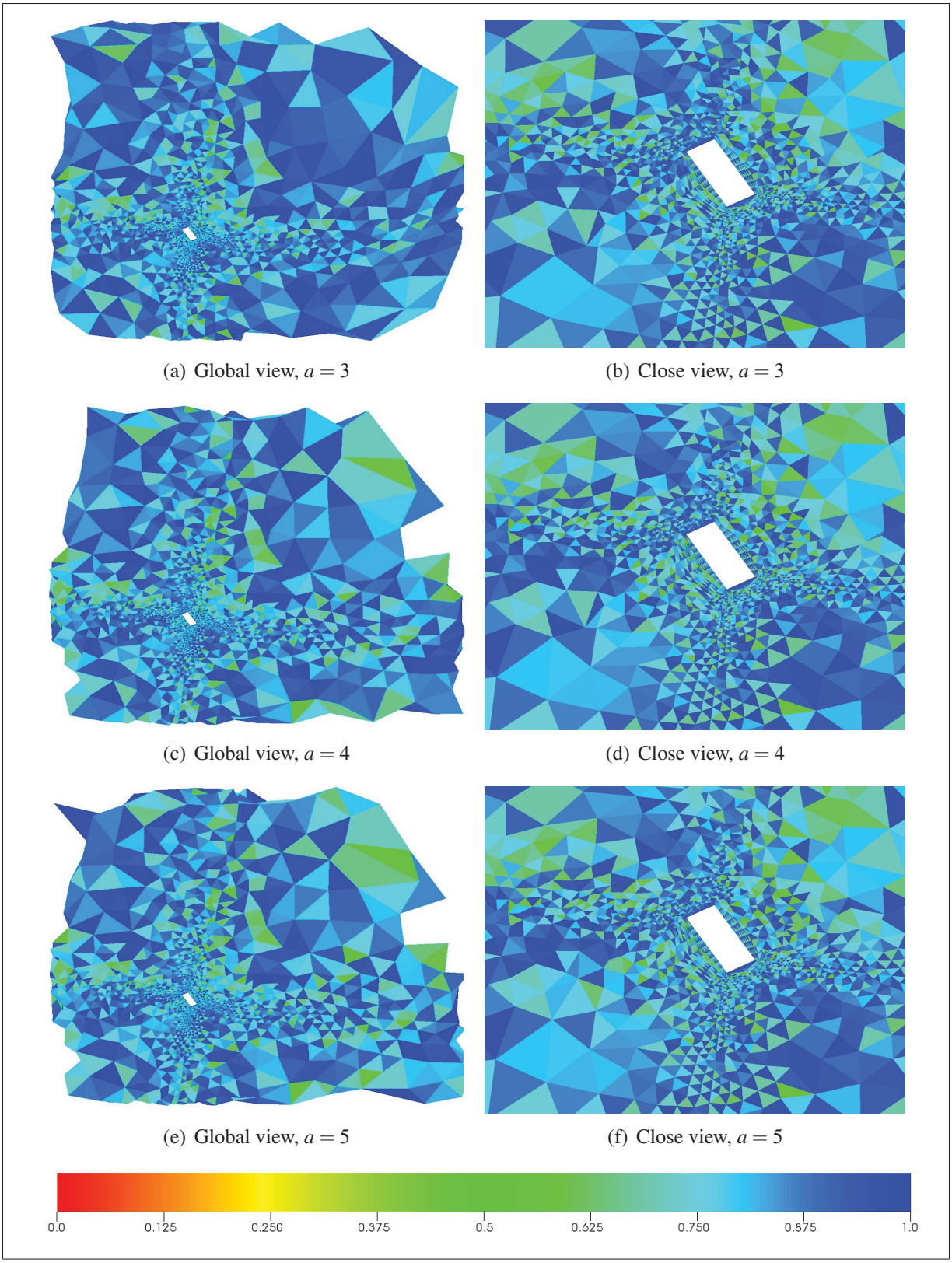
The values of exponents represent how smooth the displacements will be distributed from moving surfaces to the domain. A lower exponent will smooth the displacements field, while a higher one will generate more differences in displacements between neighbour nodes and keep elements close to boundary less deformed.




(a) Global view, $a = 1$ (b) Close view, $a = 1$

(c) Global view, $a = 2$ (d) Close view, $a = 2$

Figure 4.9 Large motion problem: $q_{chg}$ field for various exponent $a$ with $b = 5$ at t = 0.5.




(a) Global view, $a = 3$

(b) Close view, $a = 3$

(c) Global view, $a = 4$

(d) Close view, $a = 4$

(e) Global view, $a = 5$

(f) Close view, $a = 5$

Figure 4.10 Large motion problem: $q_{chg}$ field for various exponent $a$ with $b = 5$ at t = 0.5 (cont'd).

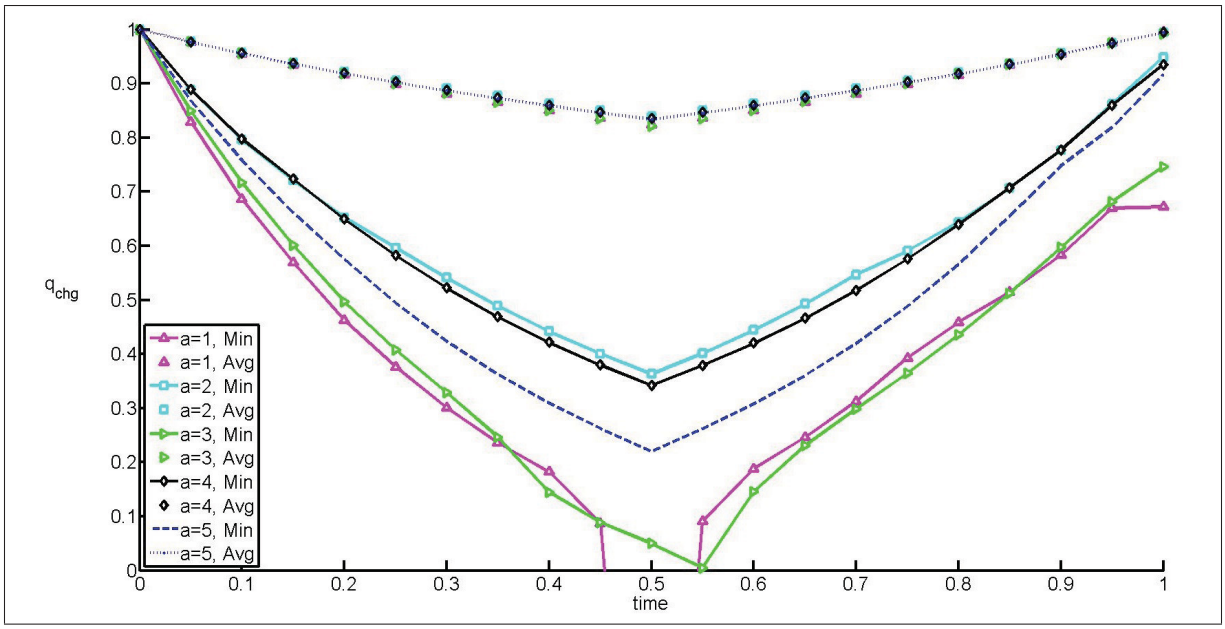



Figure 4.11 Large motion problem: $q_{chg}$ evolution for different exponent $a$ with $b = 5$.

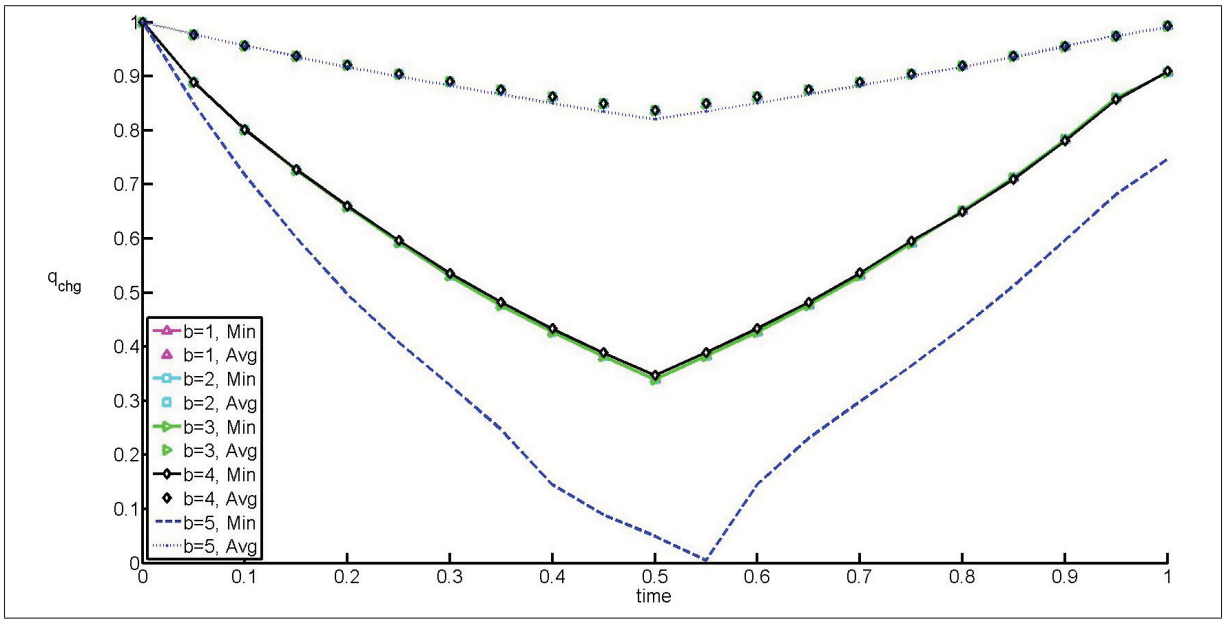



Figure 4.12 Large motion problem: $q_{chg}$ evolution for different exponent $b$ with $a = 3$.

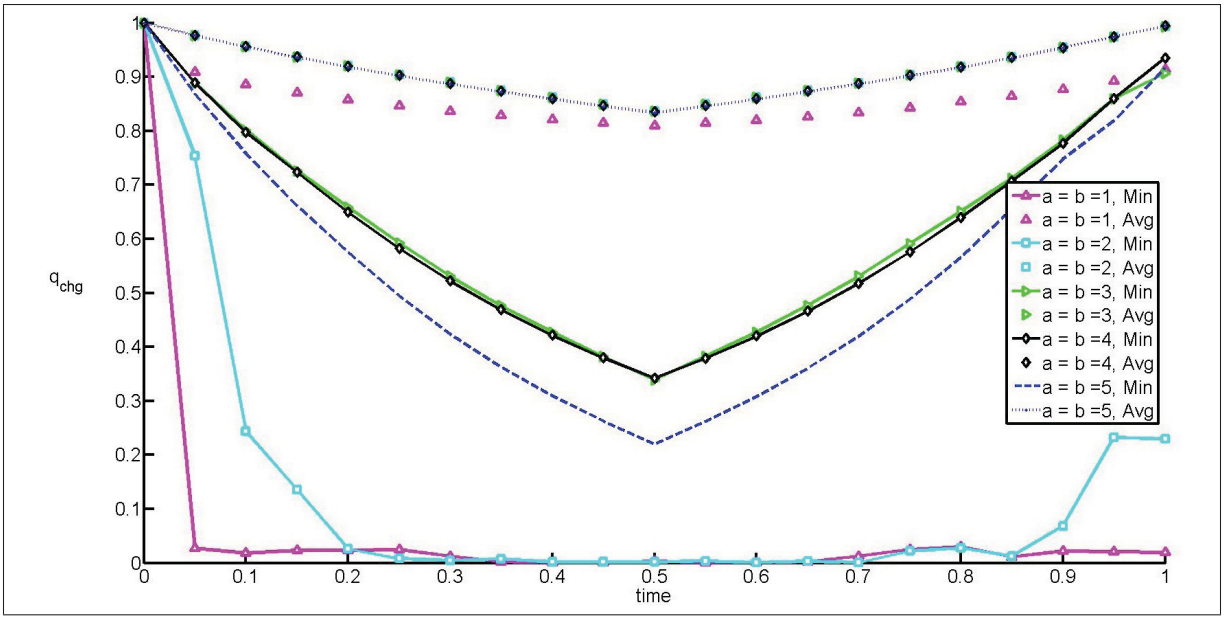



Figure 4.13 Large motion problem: $q_{chg}$ evolution of equal exponents ($a = b$).

Also, varying $L_{ref}$, the length of influence of moving boundaries, has a great impact on mesh quality after displacements. In Figure 4.14, values under the calculated value of $L_{ref} = 353.61$ always give better results, in particular for $L_{ref} = \frac{353.61}{2}$. Then for values of $L_{ref}$ from two to ten times higher, the resulting meshes are of distinguishable better quality as depicted in Figure 4.15. Changing the value of $L_{ref}$ is done to bring displacements fields closer or farther from moving boundaries. This study has shown us that it is better to modify the value of $L_{ref}$, but we believe the perfect multiplier is problem dependent. However, we can define a recommended range of $L_{ref}$ improving multiplier:

- To push displacements farther multiply the original $L_{ref}$ by 10 or;
- To bring displacements closer multiply the original $L_{ref}$ by 0.5.

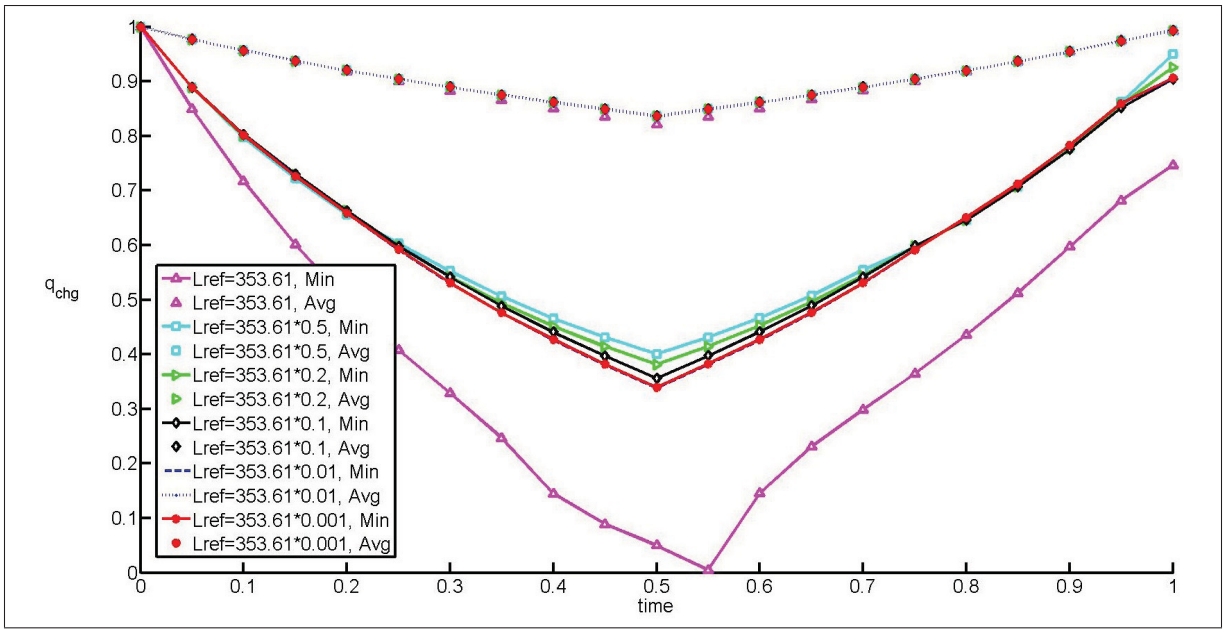



Figure 4.14 Large motion problem: $q_{chg}$ evolution for lower $L_{ref}$ with exponents $a = 3$ and $b = 5$.

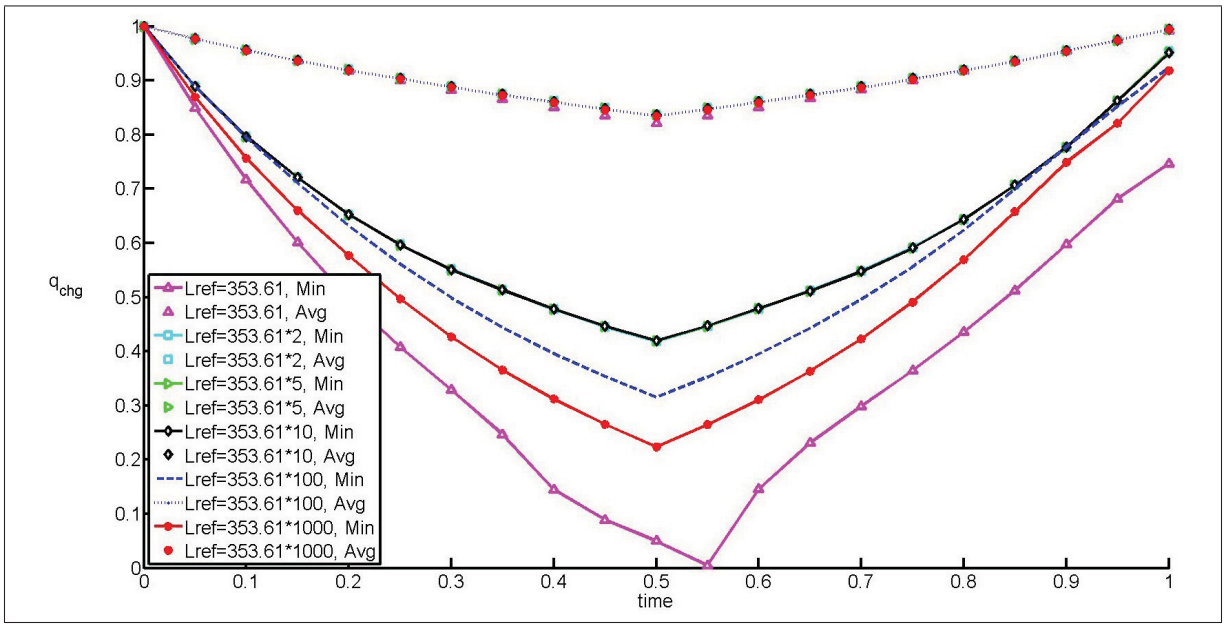



Figure 4.15 Large motion problem: $q_{chg}$ evolution for higher $L_{ref}$ with exponents $a = 3$ and $b = 5$.

### 4.1.4 IDW quality criteria

Figure 4.16 shows the variation of quality depending on the exponent $c$ (see Section 2.6). It is clear that a value of exponent $c$ between 1 and 3 creates a small improvement in the minimal quality at the final position. It is caused by the fact that when elements get distorted, their displacements are calculated by a smoother function for $1 < c < 3$. Where values of $c = \{1,2\}$ smooths both exponents, $a$ and $b$, and the value of $c = 3$ smooths only $b$. Finally, values of exponent $c$ that are greater than exponents $a$ or $b$ give bad results.

Even if we have presented a slight improvement of quality from the usage of exponent $c$, we will not use it for further simulations. Simply, we do not believe this exponent improves enough the mesh quality and we do not wish to have another parameter to control.

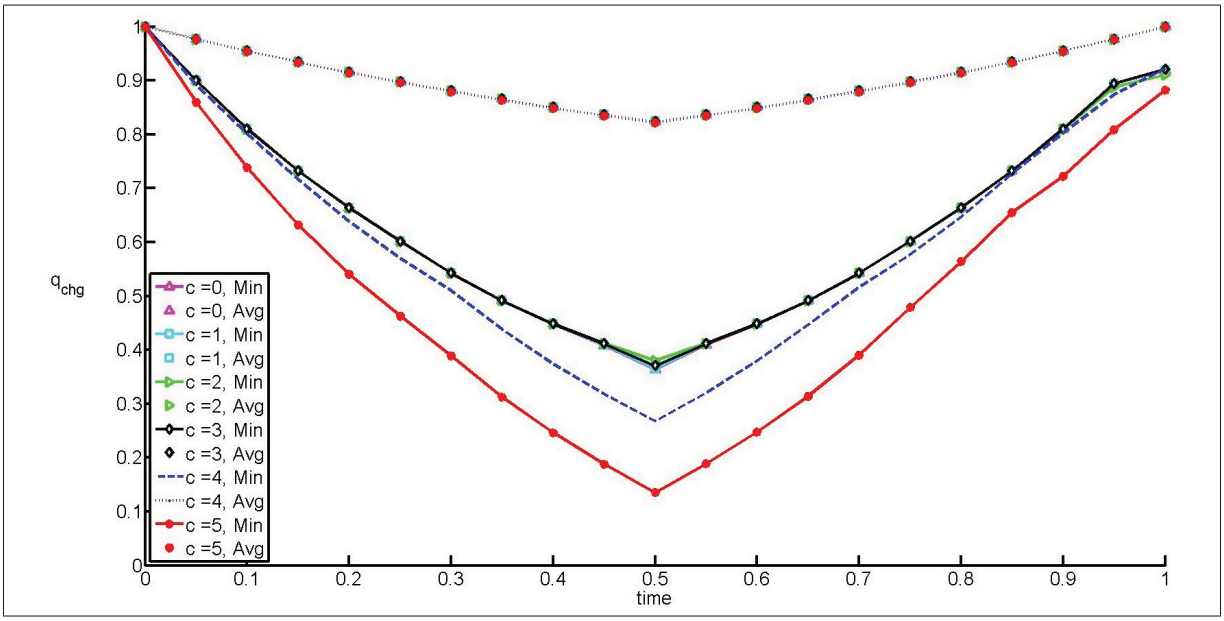



Figure 4.16 Large motion problem: $q_{chg}$ evolution for various exponent $c$ with exponents $a = 3$ and $b = 5$.

**Recommended IDW settings:**

The exponents $a$ and $b$ value should be between 3 to 4, then exponent $c$ should be equal to zero since the use of two exponents is robust enough and $L_{ref}$ should be the half or ten times the original calculated value (which is the maximum distance between a node and the mesh centroid). Also, the close body region factor $\alpha$ should be calculated as defined by Luke *et al.* (2012) or be imposed to make $\alpha \cdot L_{ref}$ equal to the boundary layers total height.

### 4.1.5 MSA-PSM/MSA-IDW results and meshes refinement study

As presented in Section 2.3, the MSA consists of deforming a coarse mesh and interpolating its displacements to the computational mesh. This method should allow less distortions in the coarse mesh than in the fine mesh, thus after interpolation the fine mesh should be valid longer. Also, solving the displacements field of a coarse mesh and interpolating it are less CPU consuming than solving the fine mesh displacements field especially for PDE type MMAs.

Since, there is no definition of how coarse a mesh should be, a preliminary study will be done. Three different coarse meshes (see Table 4.1) will be analysed for the case of the rotating translating box.

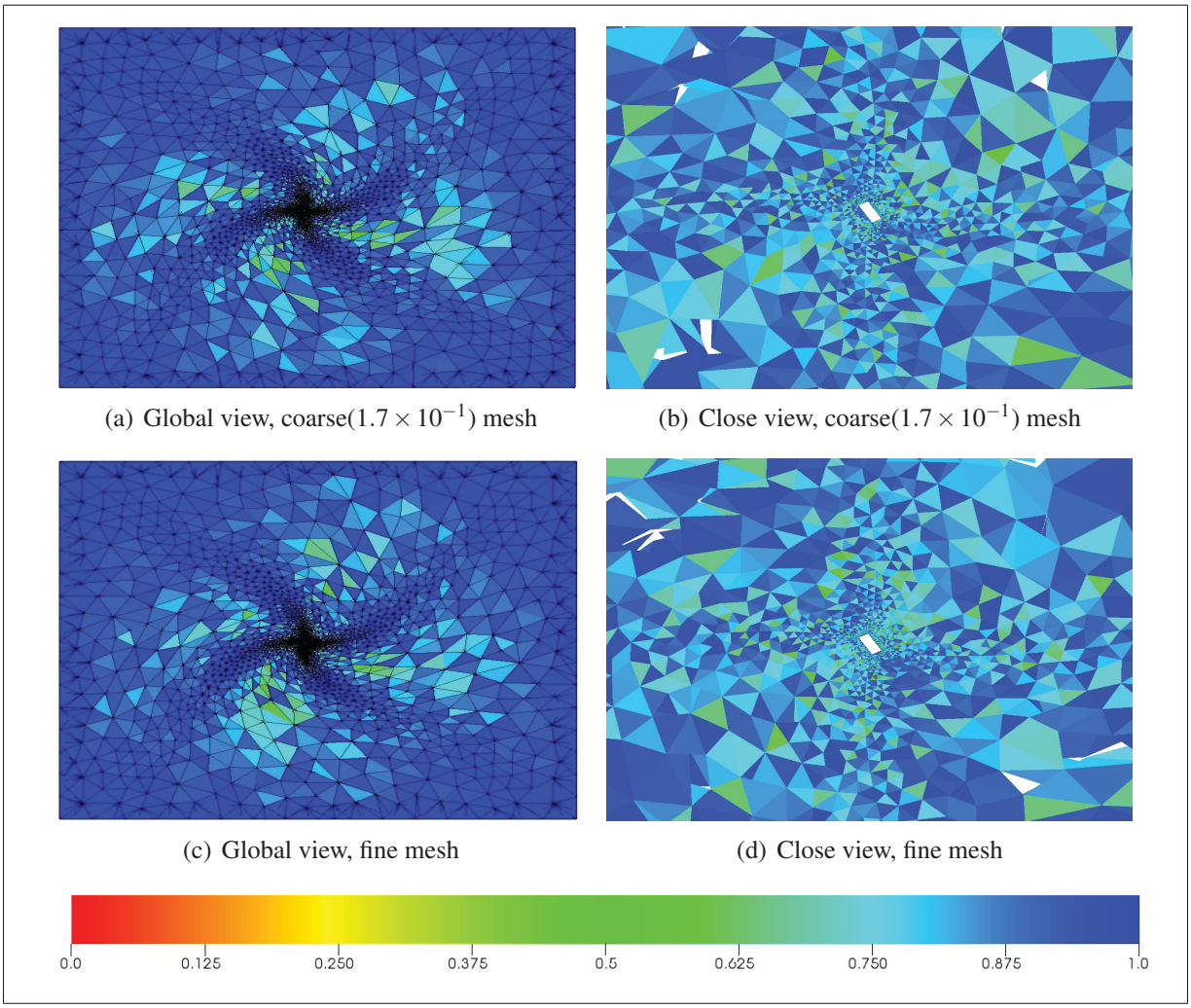
Table 4.1 Large motion problem: MSA meshes statistics

| Mesh | First Layer Height | Nodes | Elements |
|---|---|---|---|
| Coarser | $1.7 \times 10^{-1}$ | 21 212 | 103 277 |
| Coarse | $1.7 \times 10^{-2}$ | 38 808 | 206 269 |
| Coarse Finer | $1.7 \times 10^{-4}$ | 74 319 | 414 742 |
| Fine | $1.7 \times 10^{-6}$ | 109 969 | 624 658 |

In Figure 4.17 is shown the coarsest mesh and its resulting fine mesh at final position (t=0.5), then in Figure 4.19 is shown the comparison of $q_{min}$ and $q_{avg}$. All the simulations were done with the MSA-PSM method with $\nu = 0.25$, $p = 1$ and $pq = 2$. Visually, the resulting fine meshes from different coarse meshes are similar, thus they are not presented.

The noticeable difference for our various coarse meshes is that after attaining the final position ($t = 0.5$) the 'coarse' mesh generates better minimal quality. Also, after $t = 0.75$ the standard PSM has a better minimal quality than MSA-PSM. Similarly, in Figures 4.18 and 4.20 are shown results for the MSA-IDW method with exponents $a = b = 4$ and $L_{ref} = 353.61 \times 10$.

For all coarse meshes, the MSA-IDW and the standard IDW method generate similar quality evolutions. Except for $0.25 < t < 0.75$ the MSA-IDW, especially with the coarser mesh, gives a small increase in minimal quality, but not enough to consider that the refinement of the coarse mesh has an impact on the results. Also, from this simulation it is not shown the advantage of using the MSA-IDW over the IDW.



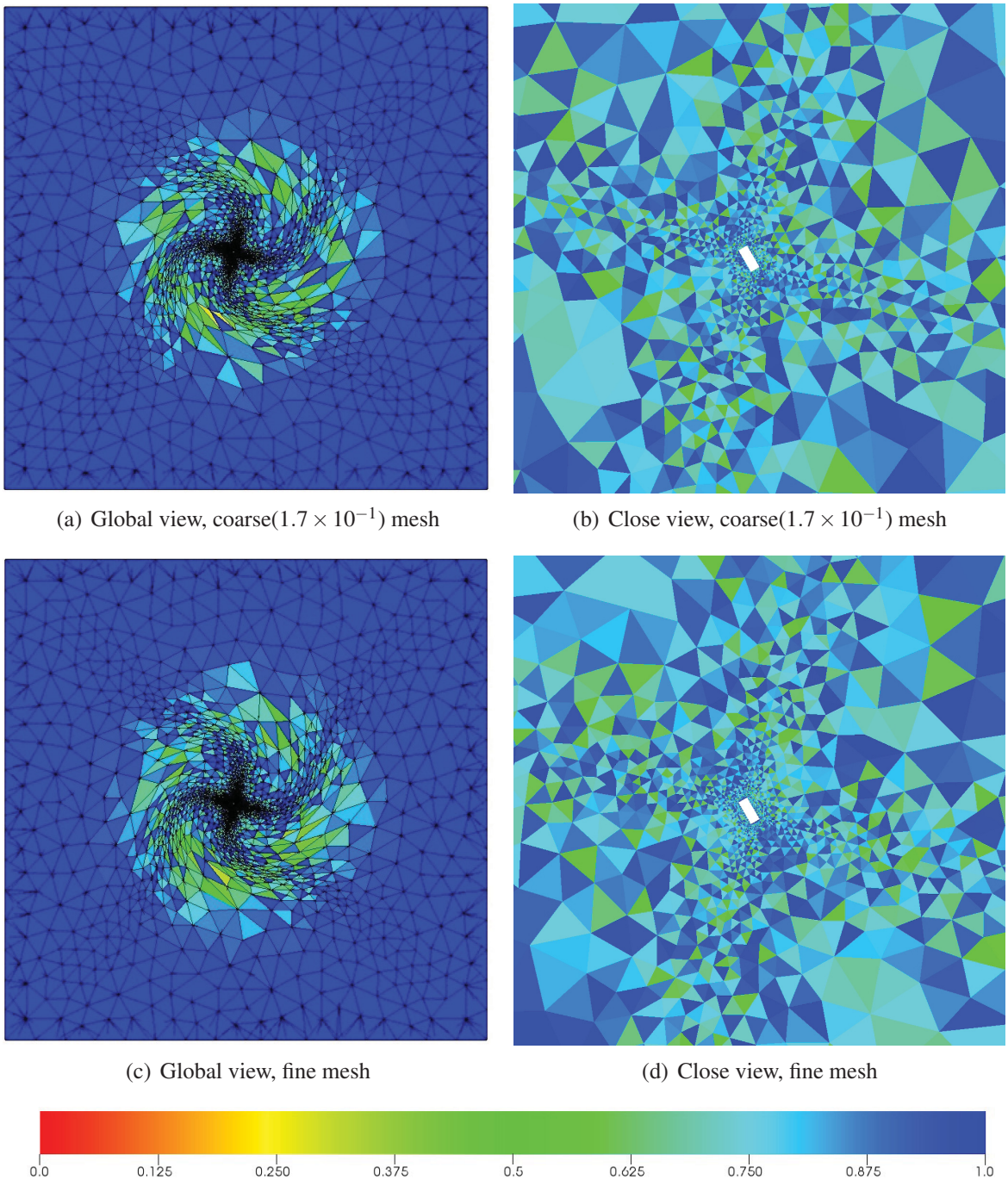

(a) Global view, coarse($1.7 \times 10^{-1}$) mesh

(b) Close view, coarse($1.7 \times 10^{-1}$) mesh

(c) Global view, fine mesh

(d) Close view, fine mesh

Figure 4.17 Large motion problem: $q_{chg}$ field at t = 0.5 moved by MSA-PSM.




(a) Global view, coarse($1.7 \times 10^{-1}$) mesh

(b) Close view, coarse($1.7 \times 10^{-1}$) mesh

(c) Global view, fine mesh

(d) Close view, fine mesh

Figure 4.18 Large motion problem: $q_{chg}$ field at t = 0.5 moved by MSA-IDW scheme.

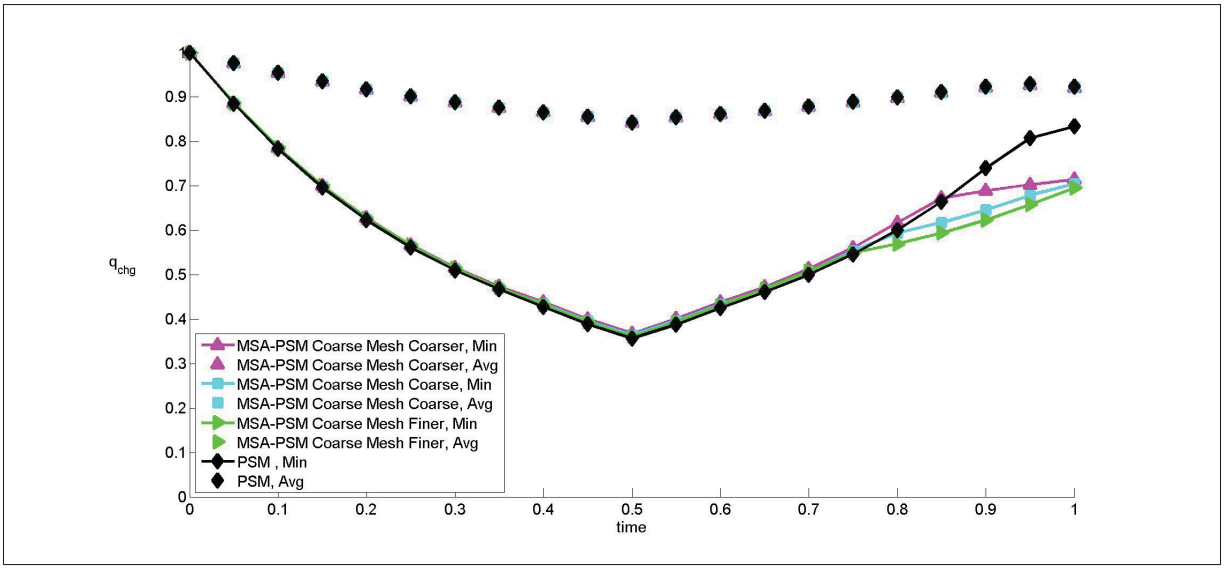



Figure 4.19 Large motion problem: $q_{chg}$ evolution moved by MSA-PSM for different coarse meshes.

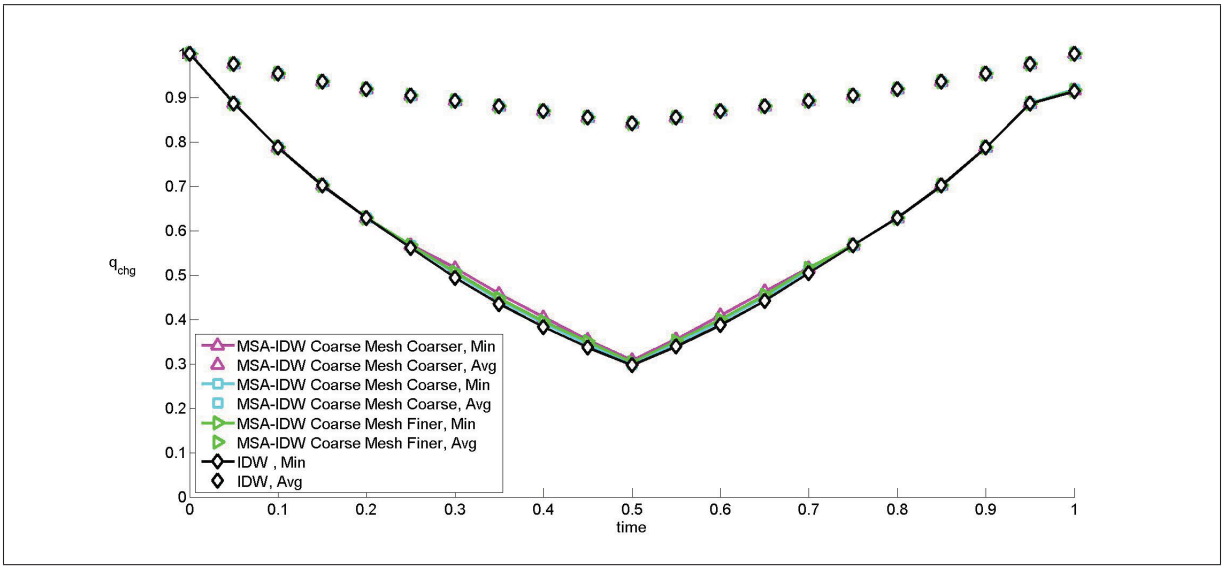



Figure 4.20 Large motion problem: $q_{chg}$ evolution moved by MSA-IDW for different coarse meshes.

### 4.1.6 MMAs comparison

To conclude this section, a simple comparison of the best MMAs is done in Figure 4.21 and it is shown that for $t > 0.7$ the PSM based methods produce lower quality than IDW based ones. Also, the MSA-IDW compared to IDW gives better minimal quality for the critical period of the simulation: $0.25 < t < 0.75$.

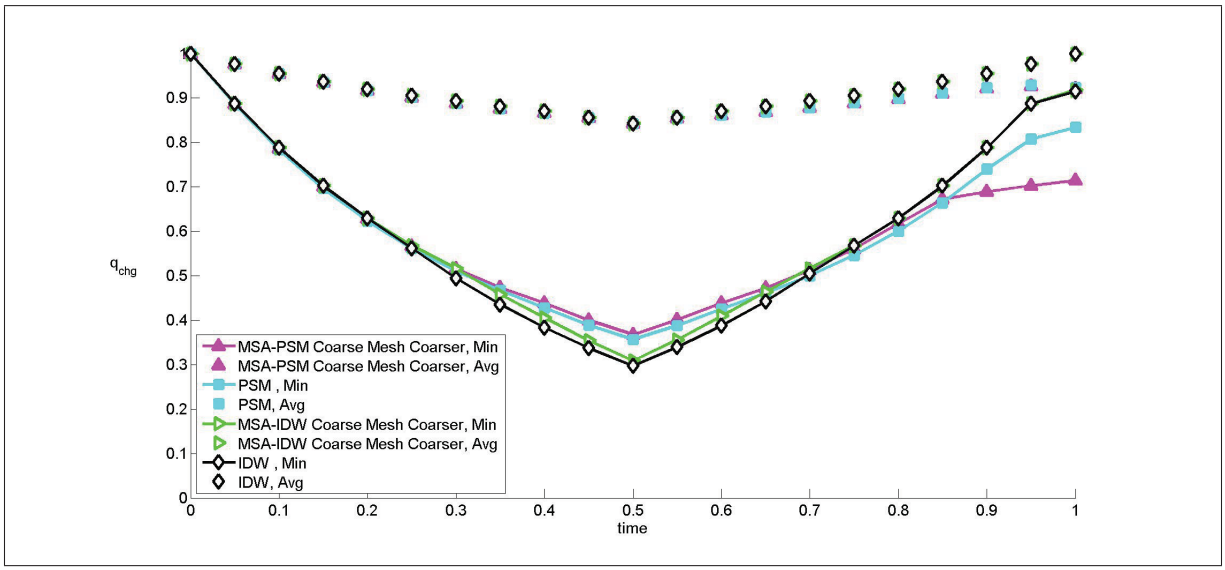



Figure 4.21 Large motion problem: MMAs comparison of $q_{chg}$ evolution.

**Recommended MMAs for large motions:**

For simple problem, the PSM distributes the displacements of moving boundaries while preserving better the initial mesh quality, but if the motion is reverse or oscillatory the quality will decrease. Thus, in such situation the IDW should be used. Both methods, can be combined to the MSA in order to improved the deformed mesh quality and reduce the computation time (only for MSA-PSM).

## 4.2 Multi-body validation

The following simple simulation will help to understand how multiple bodies interact, in particular when they get closer. In Figure 4.22 is shown a schematic, at scale, of the imposed moving boundaries displacements: a cylinder shape (diameter = 4.0 and lenght = 10.0) is getting closer to a Γ shape through rotation of 76 degrees around the 'z' axis.

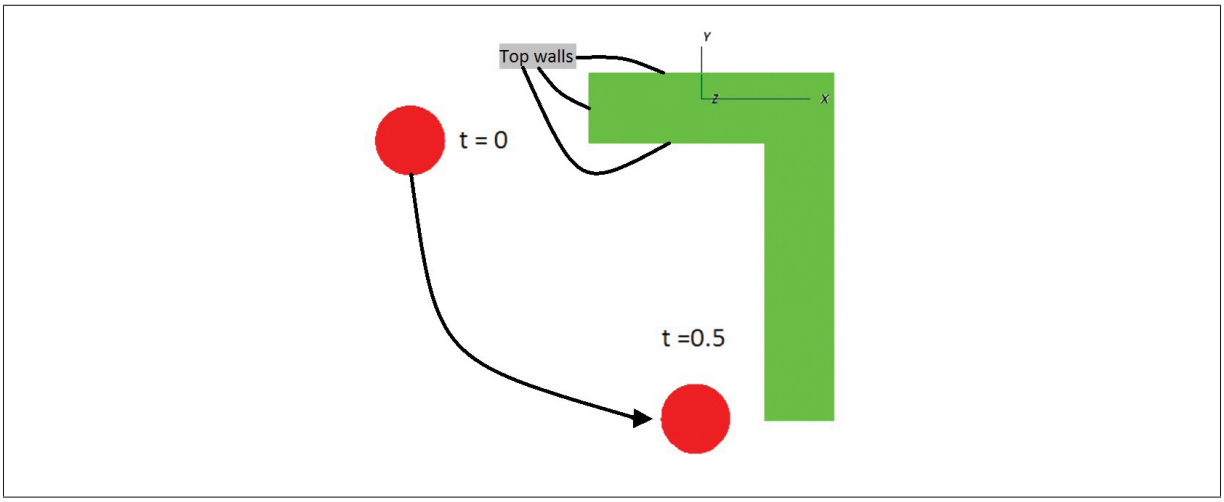
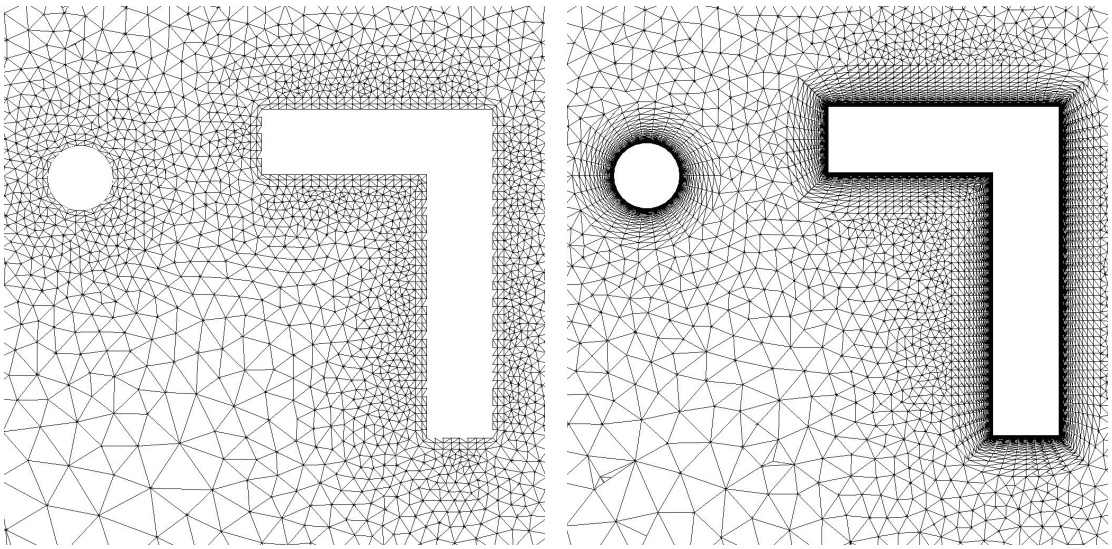



Figure 4.22 Multi-body problem: Schematic moving boundaries motion.

For the subsequent simulations two meshes of dimensions $1250 \times 1250 \times 10$ are used: a coarse mesh of 25 809 nodes with 98831 elements and a fine mesh of 98 175 nodes with 507 719 elements. The coarse mesh should be generated without pyramid because their interpolation of displacements are not accurate for distorted pyramids. The meshes are shown in Figure 4.23 for the section z= 5.0.



(a) Coarse mesh (b) Fine mesh

Figure 4.23 Multi-body problem: meshes at $t = 0$ at z=5.0.

Moreover, both meshes are composed of prism layers extruded from the moving boundaries, followed by structured tetrahedral layers and the balance of the domain is filled with unstructured tetrahedron. The interfaces between hexahedron and tetrahedron are generated without the need of pyramidal element. In the Table 4.2 is detailed the information concerning the mesh boundary layers. These particular meshes are generated to allow the IDW method to keep valid

Table 4.2 Multi-body problem: Meshes Statistics

| **Mesh** | **Prism boundary Layers** | | |
|---|---|---|---|
| | **1st Height** | **Growth** | **Quantity** |
| Coarse | $3 \times 10^{-1}$ | 1.3 | 1 |
| Fine | $1 \times 10^{-4}$ | 1.3 | 28 |

mesh longer, to reduce computation times and to allow a good smoothing.The parameters of the MMAs for the simulation are defined in Table 4.3.

Table 4.3 Multi-body problem: MMAs parameters

| Algorithm | Parameters | Value |
|---|---|---|
| PSM w\o shear | | |
| | $E$ | 1.0 |
| | $\nu$ | 0.0 |
| | $p$ | 1.0 |
| | $pq$ | 2.0 |
| IDW | | |
| | $a$ | 3.5 |
| | $b$ | 3.5 |
| | $c$ | 0.0 |
| | $L_{ref}$ | 10.0 |
| | $\alpha_{min}$ | 0.01 |
| | $\Gamma$ shape's three top walls $(A_i)$ | $0.5 \times A_i$ |
| Both | | |
| | Iterations | 20 to final position |
| | Reverse Iterations | 20 to go back to the original position |

The PSM parameters are set according to the best combination shown in previous simulations. However, shearing effects are removed because meshes generated are of better quality with $\nu = 0.0$. The IDW exponents $a$ and $b$ are set to three and zero to generate a smooth field. Since the value of $a = 3$ is the minimum there is no need to smooth the exponent according to quality, thus exponent $c$ is set to zero. Finally, the $\Gamma$ shape top wall (see Figure 4.22) faces area have been divided by four to minimize their influence on nodes displacements.

These simulations will validate the requirement of using an encapsulation zone, smoothing algorithms and a mesh untangler.

### 4.2.1 Encapsulation Zone

As said in Lefrançois (2008) and Stein *et al.* (2004) a zone with increased stiffness can be generated close to moving boundaries. This zone is created in the meshing process as a distinct fluid domain and possesses a specific identification number (*volID*). These *volIDs* represent standard moving mesh blocks ($31 < volID < 39$) or encapsulation zones ($volID = 30$). For encapsulation zones, the exponents for the PSM are doubled and for the IDW method modifying

nothing gives better result. In fact, the IDW distance functions already produce the effects of stiffening close elements, but with a smoother distribution.

In Figure 4.24 is shown the impact of using or not an encapsulation zone for the coarse mesh without smoothing. The zone is the layer of hexahedral elements extruded from moving boundaries. Using rigid layer should increase both algorithms minimum and average $q_{chgAbs}$, but it

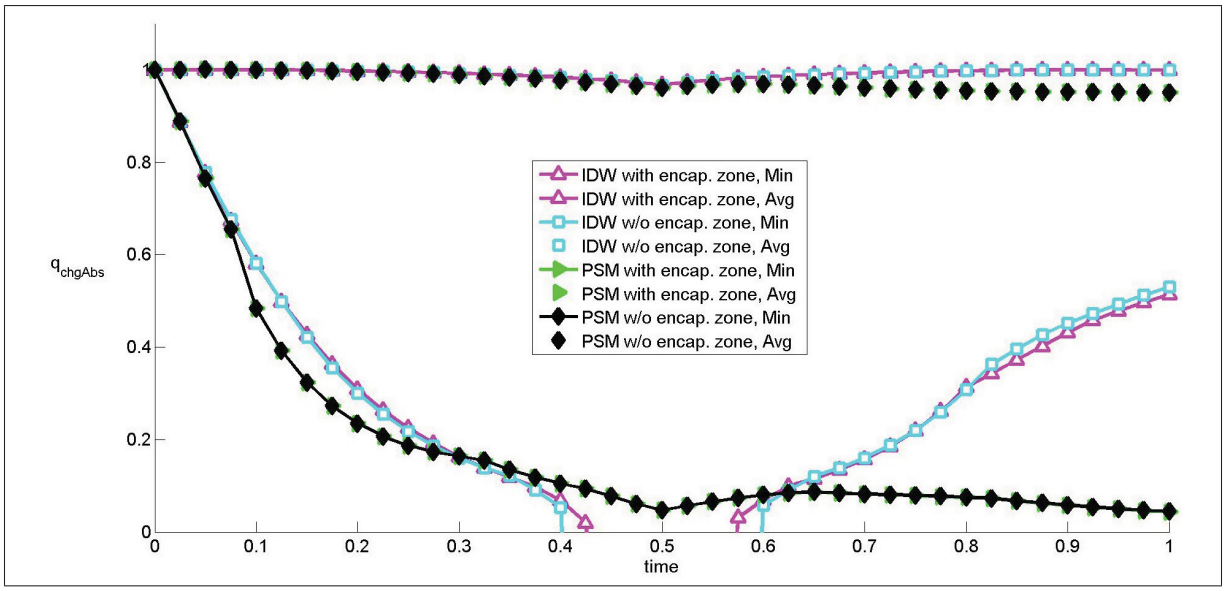



Figure 4.24 Multi-body problem: Impact of using an encapsulation zone for the $q_{chgAbs}$ evolution.

only allows the IDW method to keep a valid mesh longer ($q_{chgAbs}min > 0$). From this preliminary simulation, it is proven that the IDW algorithm is not robust for multi-body interaction simulations, but it will be with the help of smoothing and untangling tools.

**Recommendation on the use of encapsulation zone:**

The improvement of using an encapsulation zone is not visible by comparing global quality values, although it can be used to limit the smoothing algorithms impact on boundary layers.

### 4.2.2 Smoothing Parameters and Mesh Requirements

As stated in Section 2.4, mesh quality can be increased by smoothing elements in order to bring them closer to their reference shape. Several options are available to smooth a mesh and bad parameters value can result in an infinite loop or no improvement at all. Thus, we recommend the following parameters:

a. The quality metric to be increased by the smoothing algorithms is $q_{chgAbs}$ (as explained in section 2.5);

b. Smooth when $q_{min} < 0.25$;

c. Smooth globally (Algorithm 2.3) until the improvement is less than the tolerance of $10^{-4}$;

d. Smooth locally (Algorithm 2.4) elements of $q_{min} < 0.5$ five times;

These parameters permit good averaging of distortion through the domain and makes future meshes protected better against generation of negative elements. There are three additional options which are going to be tested:

a. Smooth according to $q$ before the first movement;

b. Smooth at each iteration according to $q_{chgAbs}$;

c. Options a) and b).

Thirdly, to ensure a good smoothed mesh we recommend the mesh to be composed of surface boundaries of face aspect ratios between one and fifty. These boundaries are fixed during the smoothing process, thus trying to smooth elements connected to boundaries of bad aspect ratio will result in bad quality elements. Also, in this simulation two domains are skipped in the smoothing process: the far-field ($volID = 39$) and the encapsulation zones ($volID = 30$). The choice of not smoothing these regions helps respecting the face aspect ratio requirement

and in keeping the boundaries unchanged; thus this approach is used in all simulations under smoothing.

In Figures 4.25 and 4.26 are shown the comparisons of those three options for each MMAs. Smoothing meshes at $t = 0$ generates worse deformed meshes for the IDW method while almost generating no changes with the PSM. Then, smoothing when the quality is under 0.25 improves minimum quality considerably, but does not protect against generation of negative elements. In fact, it even perturbed meshes moved with the PSM to make them invalid for $t > 0.5$. For the IDW method it pushed further the moment when meshes become invalid. Also, the combination of both options gives a more stable evolution for both MMAs than solely smoothing when the quality is under 0.25.

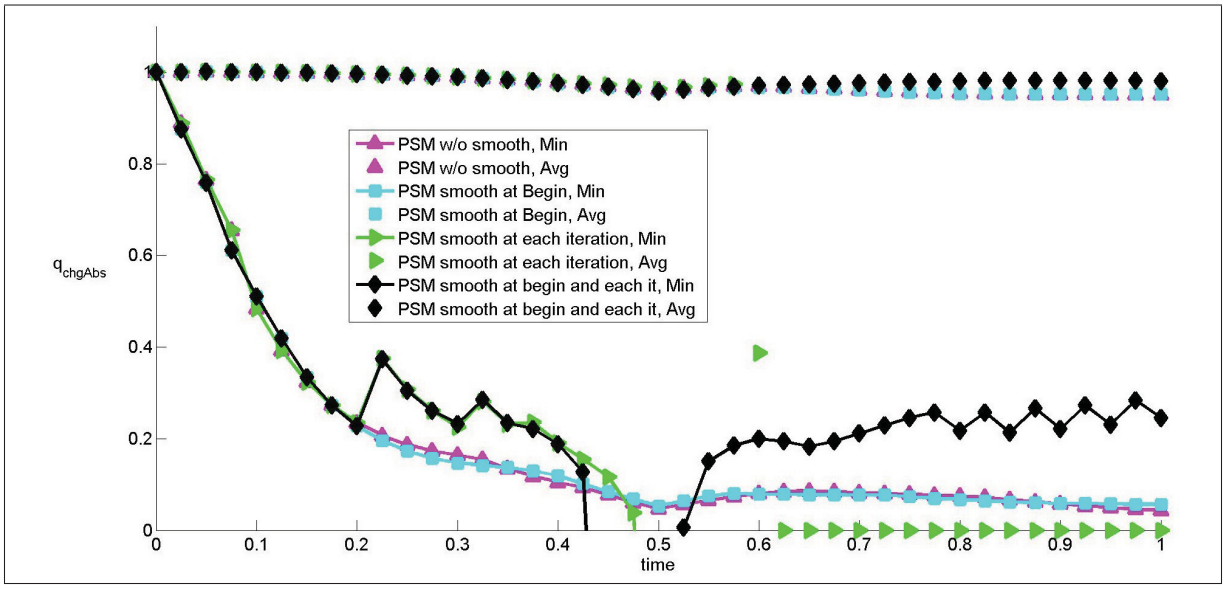



Figure 4.25 Multi-body problem: Impact of smoothing on the $q_{chgAbs}$ evolution for the PSM.

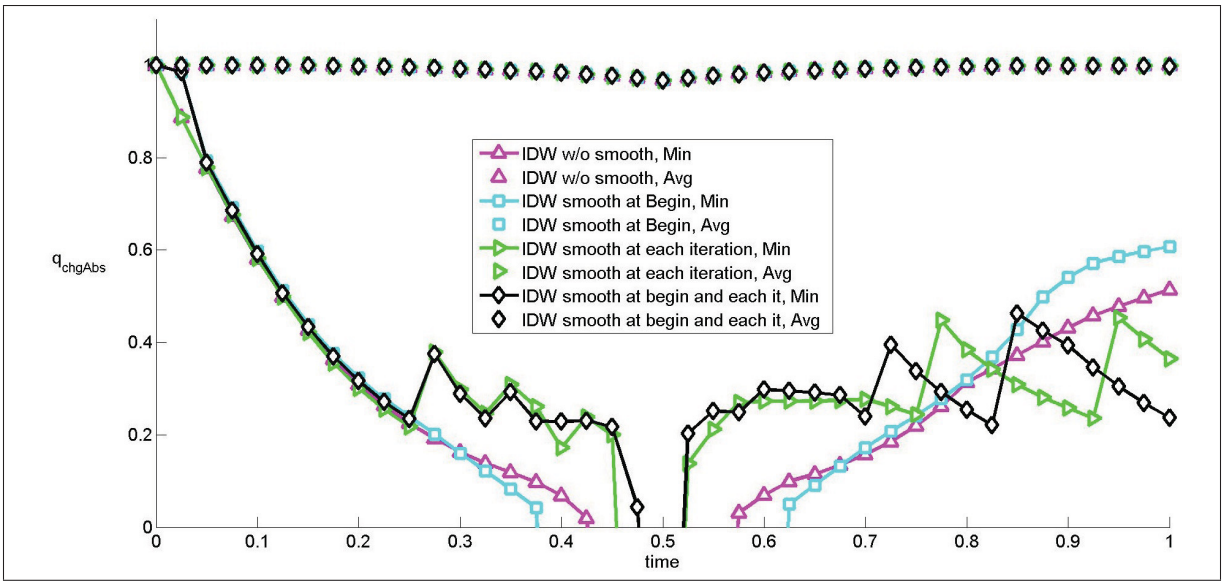



Figure 4.26 Multi-body problem: Impact of smoothing on the $q_{chgAbs}$ evolution for the IDW method.

In Figure 4.27 is shown a comparison of MMAs while using or not the smoothing process. The smoothing options are those which gave better results for each MMA. Smoothing a mesh while moving it with the PSM removes the stability of the algorithm which can easily result in invalid meshes. Although, smoothing before the first movement can improve the original mesh quality, as in the previous simulation $q_{min}$ was raised from 0.00787278 to 0.164369.

The difficulty with the IDW algorithm for interacting bodies is that elements between bodies are equally moved by each moving body which can generate easily distorted elements even if the mesh is smoothed.

**Recommendation on the use of smoothing:**

Smoothing algorithms help elements at risk, those between bodies, from being largely distorted and make the mesh valid longer. Although, it is not sufficient to make MMAs truly robust which proves the need for an untangler algorithm.

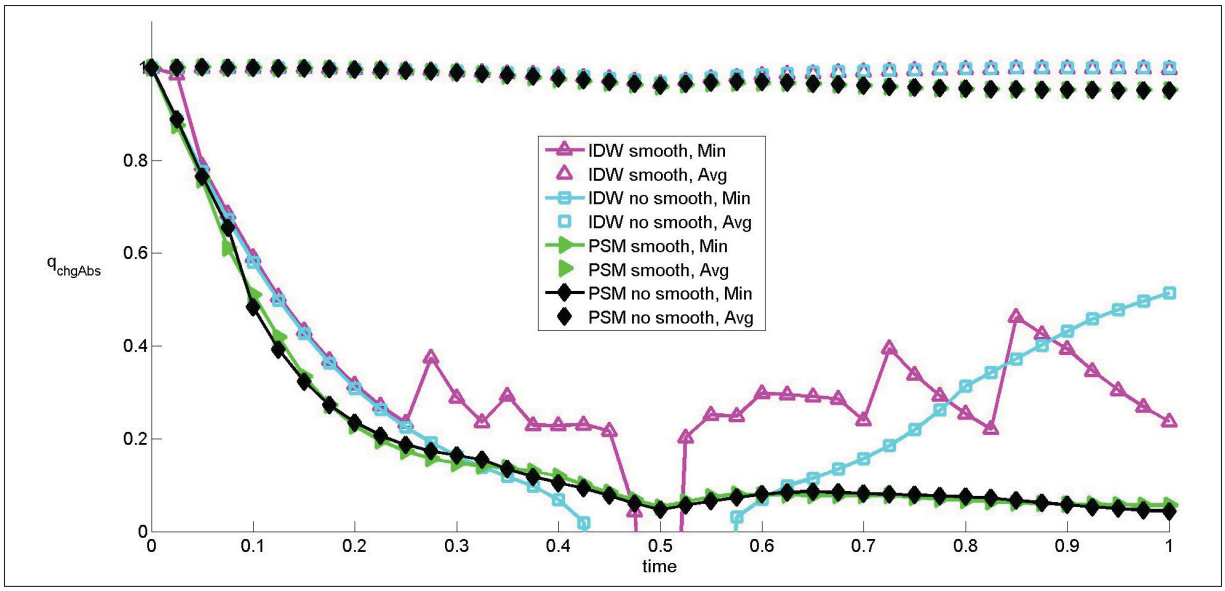



Figure 4.27 Multi-body problem:: Impact of smoothing on the $q_{chgAbs}$ evolution for different MMAs.

### 4.2.3 Validation of the Novel GETMe Untangler (NGU) algorithm

The NGU presented in Section 2.7 should resolve the problem of having negative elements, especially while using the IDW scheme. To assess its necessity, workability and capability in conserving mesh quality, we compare, in Figure 4.28, quality evolutions without the NGU or with it and with or without smoothing. It is clear that using the NGU allows the mesh to be kept valid through all the proposed simulation. Also, the smoothing algorithms and untangler are shown to be compatible tools. Thus, we are able to improve, or preserve, as wanted mesh quality and to keep valid meshes through complex movements.

### 4.2.4 Remarks on boundary geometries and boundary layer

There is some concerns from specialists about the risk of mesh adaptations not respecting the original geometries. Although, our algorithms are built in order to make them unalterable. Our approach consists in always imposing rigid-body motion to each moving boundary node. In other words, nodes displacements which are calculated or interpolated are those which are not on moving boundaries.

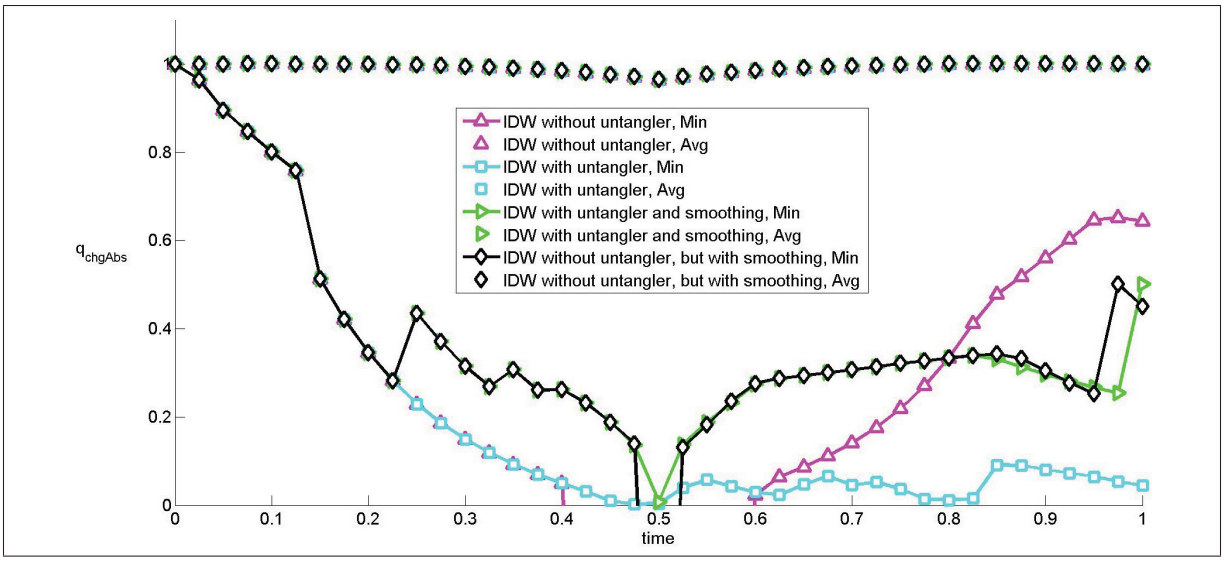



Figure 4.28 Multi-body problem: Impact of using the Novel untangler tool for mesh movement.

As well, elements close to moving boundaries for complex phenomenon, such as turbulent flows, are required to be of good quality. Thus, special care should be taken to generate and move the mesh, although all MMAs used in this work are designed to preserve boundary layer elements quality. Through our multiple simulations, we have learned that elements in boundary layer regions are kept unchanged for most situations, but little distortions can be found for multi-body test cases. This is not alarming since other elements in the mesh are more distorted, thus they would affect more the veracity of a physical solver results. This means that the need of re-meshing or mesh invalidity will not emerge from boundary layers elements.

### 4.2.5 MMAs comparison

Previously, to allow a better understanding of mesh smoothing, the loops were done before the mesh displacements was solved. However, in a real multi-physics application, the mesh will be smoothed after the displacements have been solved. Thus, before the mesh is used to solve other physic equations, from now-on, it will be smoothed. A detail not to forget is that the smoothing process is done after the untangling operation, since the smoothing algorithms are defined only for valid meshes.

The simulation of the fine mesh uses the following settings:

- There is no global smoothing;
- When the minimal quality is lower than 0.25, meshes are smoothed according to $q_{chgAbs}$;
- When negative elements are generated, the NGU makes them positive;
- Elements' distance to boundary smaller or equal to 1.0 use the $q_{pre}$ metric.

In Figure 4.29, is shown the evolution of quality for the fine meshes and in Figure 4.30 is shown the deformed meshes at $t = 0.5$. The PSM and MSA-PSM results are not shown because they failed at the first iterations, thus those methods are not recommended for mesh-movements involving multiple bodies.

From figures 4.29 and 4.30 is shown that the MSA-IDW has a higher minimal quality until t=0.7 and the average quality is higher with the IDW method around t=0.5. This can be explained by the fact that the coarse and fine meshes are generated analogous, except for the size of the first boundary layers. Thus, generating a coarse mesh with elements following the path of moving boundaries could improve the MSA-IDW performance. However, we will not study that alternative since we wish to design MMAs to be mesh independent. Also, it can be seen that the zone with the $q_{pre}$ metric shows values close to 1.0, since their shape is almost unchanged.

For the multi-bodies fine mesh, the solver for the PSM based method was diverging. We believe that the problem emanates from the facts that we were doing calculation on a single computer and the BiCGStab solver is not adequate for large problems. Thus, we should use a solver that can produce a solution for large problems and we should include the capability of decomposing the domain to use parallel solvers.

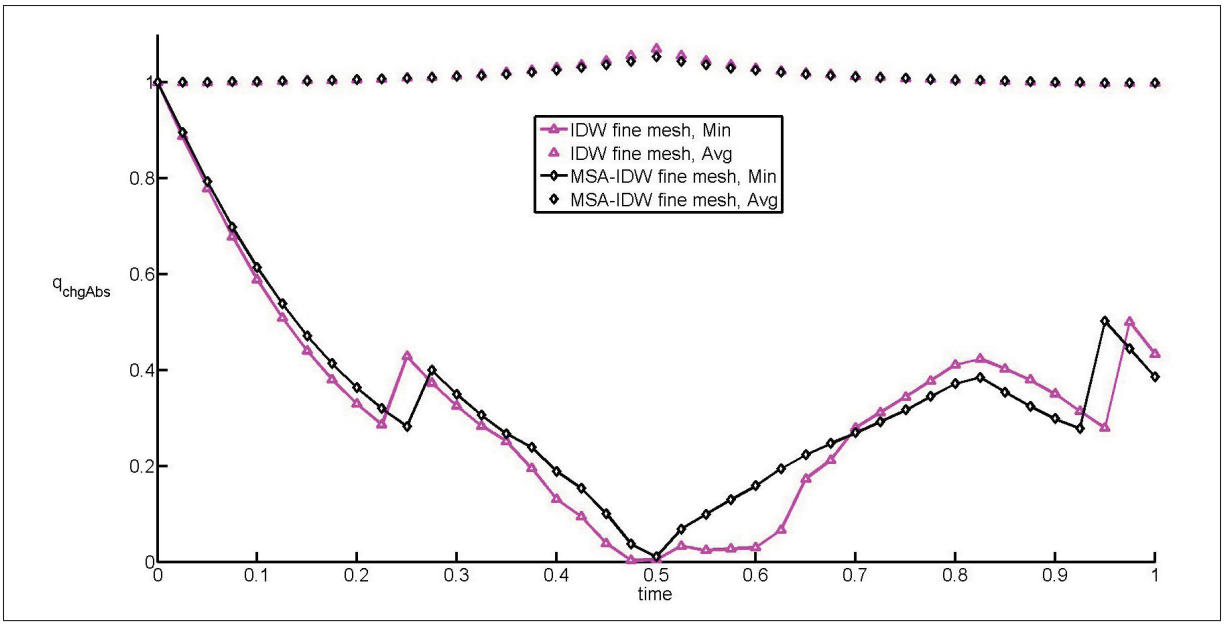
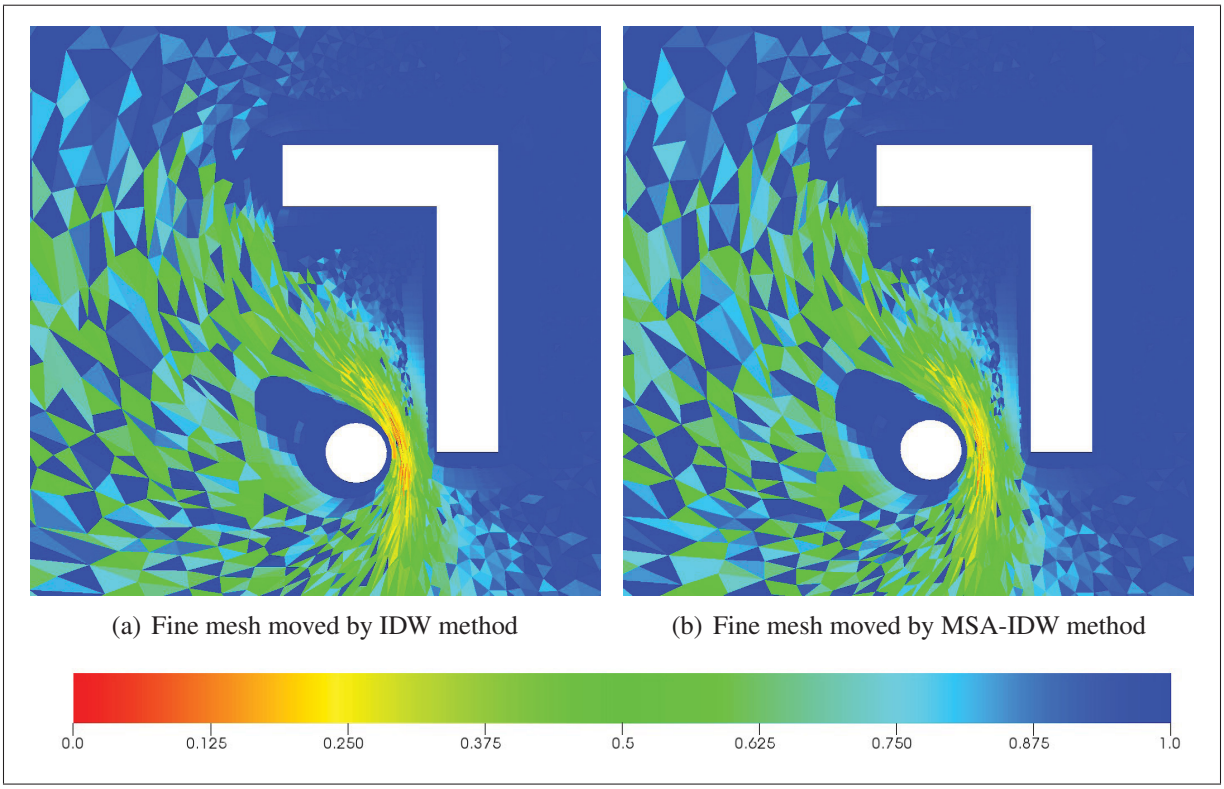



Figure 4.29 Multi-body problem: Fine mesh $q_{chgAbs}$ evolution per MMAs.




(a) Fine mesh moved by IDW method

(b) Fine mesh moved by MSA-IDW method

Figure 4.30 Multi-body problem: $q_{chgAbs}$ field at $t = 0.5$.

### 4.2.6 Recommendations

For simple one body moving-mesh problems the PSM or MSA-PSM generate better minimal quality than other MMAs. The parameters of the PSM to use are $p = 1$, $\nu = 0.25$ and $pq = 1$. However, the pseudo-material analogy should not be used for multi-body problems and complex motions.

For difficult problems, it is recommended to use the IDW or MSA-IDW methods with $a = [3,4]$, $b = [3,4]$, $c = 0$ and $L_{ref} = \frac{L_{ref,calculated}}{2}$. Also, to improve robustness the smoothing GETMe algorithms combined to the NGU are applied to the mesh to preserve mesh quality and validity. To ensure good smoothing, the boundaries faces aspect ratio are required to be close to one ($1 <$aspect ratio $< 50$), this criteria is also required for fluid flow solvers making the face aspect ratio a criteria to respect for any simulation.

### 4.2.7 Conclusions

Finally, we should understand from the previous sections that various cases of moving boundaries can be handled with moving-mesh algorithms and proposed improvements. For some strong movements the mesh can be repaired or improved to preserve good mesh quality. Although, these methods have limits and do not allow all movements especially contact or close to contact situations. The greatest advantage from our strategies to mesh movement is the gain in calculation time and ease of implementations.

To make the current tool perfect it should be able to locally re-mesh when the MMAs are close to generating an unusable mesh. Since, a lot of work has already been done on the subject it should be quite straight-forward to combine to the proposed approach a re-meshing algorithm.

# CHAPTER 5

# REALISTIC MESH-MOVEMENT SIMULATIONS

The final objective of our research is to adapt meshes of moving aerodynamic surfaces. In this section, aeroelastic geometries of meshes for viscous flow solvers will undergo large movement. The prescribed movements are the twisting-bending of a wing and the movements of wings lifting components to different flight stages.

## 5.1 NACA0012 bending, twisting and combined

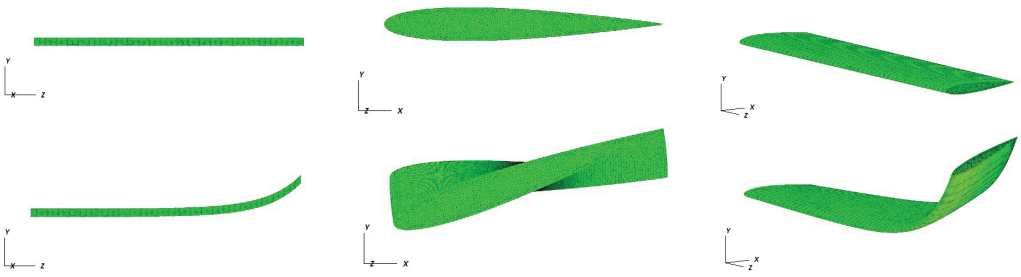
### 5.1.1 Motions Definition

The airfoil used is the symmetric NACA0012 of chord length equal to one which is extruded in the z direction four units to generate a wing. In flight, flexible wings are under pressure produced twisting and bending which are both maximal at the tip and null at the root where the wings are fixed to the fuselage. The bending and twisting displacements for this problem are expressed as follows:

$$\text{Bending: } \delta y = e^{2.0 \times z}/scaling \tag{5.1}$$

$$\text{Twisting: Rotation around z of } \delta\theta = 10^{\circ} \times z/span, \tag{5.2}$$

where the $span = 4.0$ and $scaling = 6000.0$. The span variable is there to distribute the twisting linearly from root (z = 0) to tip and the scaling variable to scale down the value of the exponential to be of 0.5 unit at the tip. Three simulations will be done: bending, twisting and the combined movement (bending + twisting). In Figure 5.1 is shown the boundary surfaces three movements.



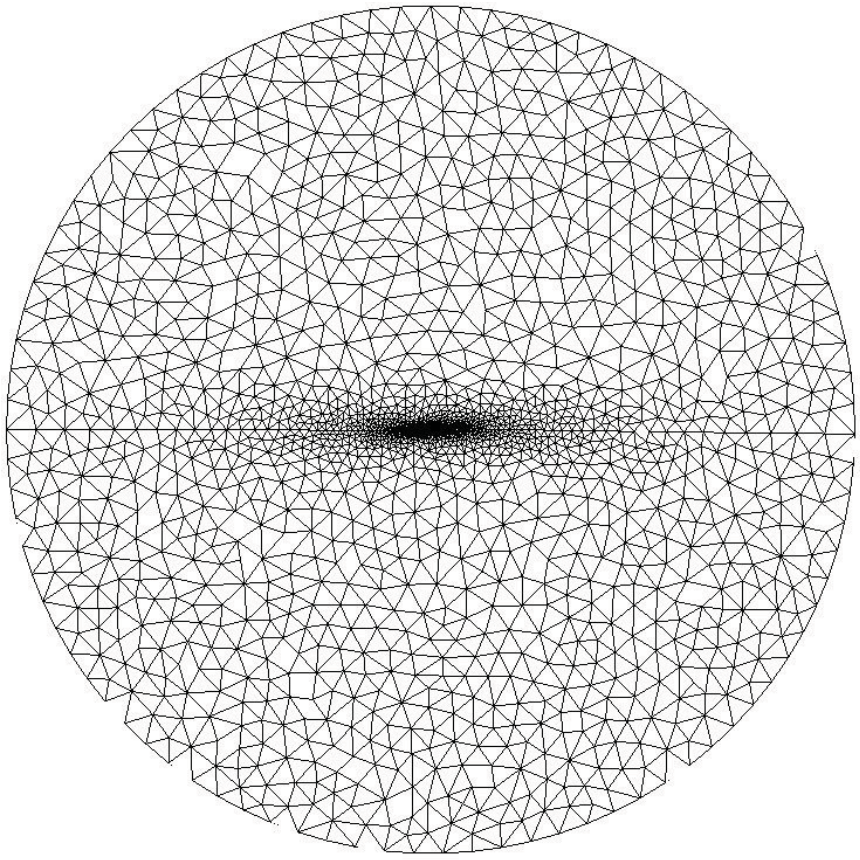
(a) Bending, (b) Twisting, (c) Combined.

Figure 5.1 NACA0012 simulations: boundary surfaces movements.

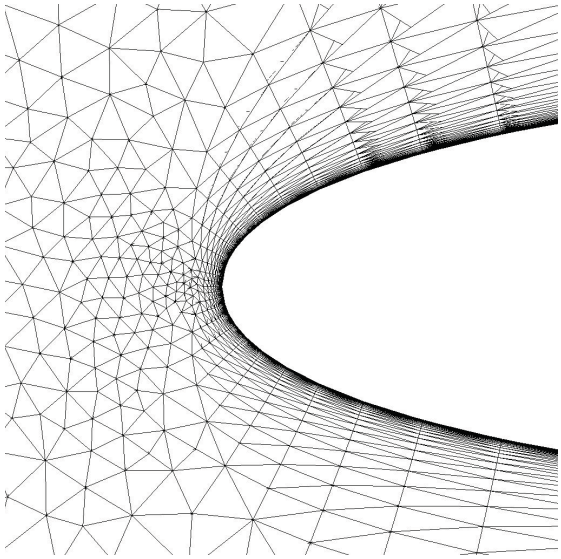
### 5.1.2 Meshes Description and MMA Configuration

The mesh around the boundary surfaces is generated with 42 tetrahedral structured layers extruded from the moving surfaces, of faces aspect ratio $\leq 12.49$, where the first 28 layers are only prism elements. The first layer is of $1.0 \times 10^{-6}$ with a growth ratio of 1.3 and the domain limits are defined by a cylinder of $radius = 155.0$. A cut of the mesh at z=2.0 is shown in Figure 5.2. The mesh is composed of 2 036 382 elements (150 718 prisms, 7465 pyramids and 523 199 tetrahedron) and 859 096 points. Similarly, the coarse mesh is obtained from the same boundaries with the first layer height from the wing of $1.0 \times 10^{-3}$, the growth ratio is equal to 2.0 and there is 8 layers where the first one is composed only of prisms. The coarse mesh has 1 049 224 elements(39 880 prisms and 1 009 344 tetrahedron) and 201 850 nodes.

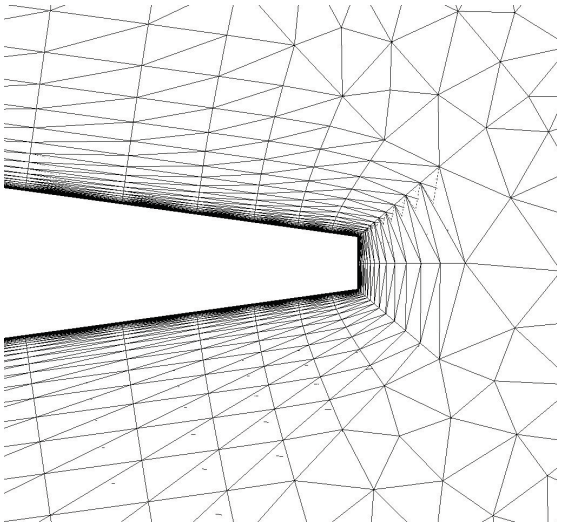
The parameters used for the large movements of the NACA0012 wing are defined according to the recommended default values of Chapter 4, as seen in Table 5.1 and the preserving quality metric is not necessary for this simulation since there is only one body.



(a) Global mesh



(b) Close view of tip leading edge



(c) Close view of tip trailing edge

Figure 5.2 NACA0012 simulations: original mesh views.

Table 5.1 NACA0012 simulations: MMA parameters

| Algorithm | Parameters | Value |
|---|---|---|
| IDW | | |
| | $a$ | 4.0 |
| | $b$ | 4.0 |
| | $c$ | 0.0 |
| | $L_{ref}$ | calculated |
| | $\alpha_{min}$ | 0.1 |
| Smoothing | | |
| | Quality minimal | until $q_{min} \geq 0.5$, but start smoothing when $q_{min} < 0.25$ |
| | Type quality | $q_{chgAbs}$ |
| | Iterations | 20 to final position |
| | Reverse Iterations | 20 to go back to the original position |

### 5.1.3 Results

In Figures 5.3 and 5.4 are shown the evolution of $q_{chgAbs}$ for each movement and MMA. Since the quality for this simulation is always higher than 0.25 the was not smoothed. For the single motions, both methods produce similar quality evolution and the combined motion is better solved with the IDW where the minimal quality around $t = 0.5$ is a little better.

Then, in Figures 5.5 and 5.6 are presented the deformed meshes for the NACA combined motion at $t = 0.5$ for the tested MMAs. From these results, it is shown that even for complex motions of a single body the proposed MMAs generate displacements fields of good quality. In this particular simulation, most of the elements have preserved their original quality except at the tip of the wing where elements are slightly distorted ($q_{chgAbs} \geq 0.4$ ). Elements close to moving boundaries of the tip have their quality reduced down to 0.4 their original which is considered acceptable especially since the boundary layers elements are mostly deformed parallel to the boundaries.

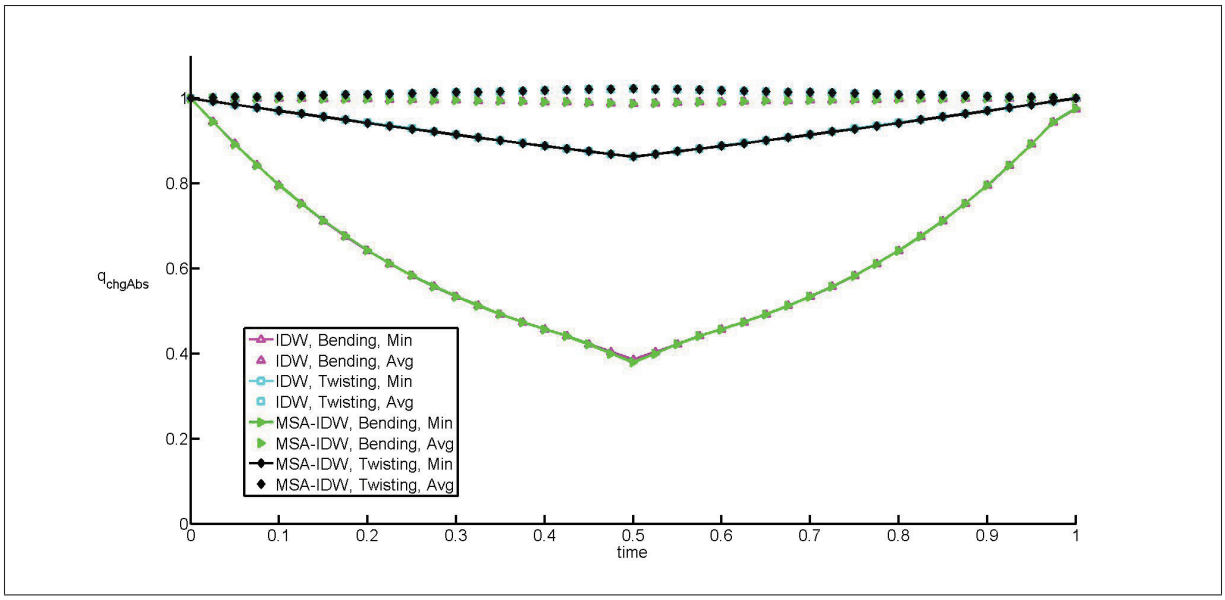



Figure 5.3 NACA0012 simulations: $q_{chgAbs}$ evolution for single motions.

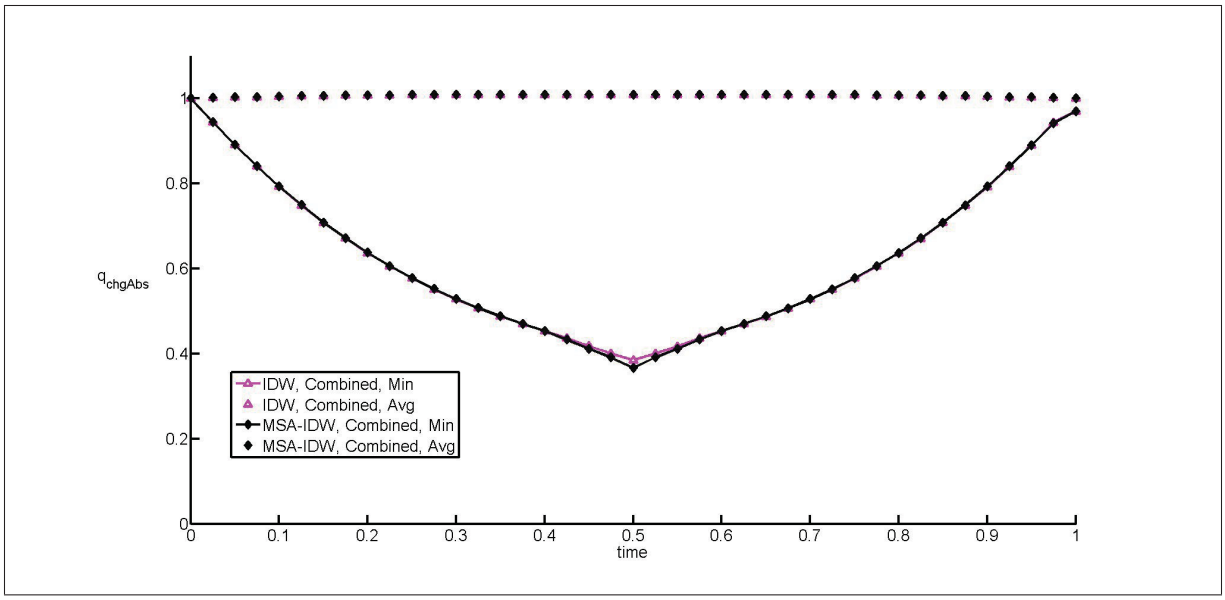
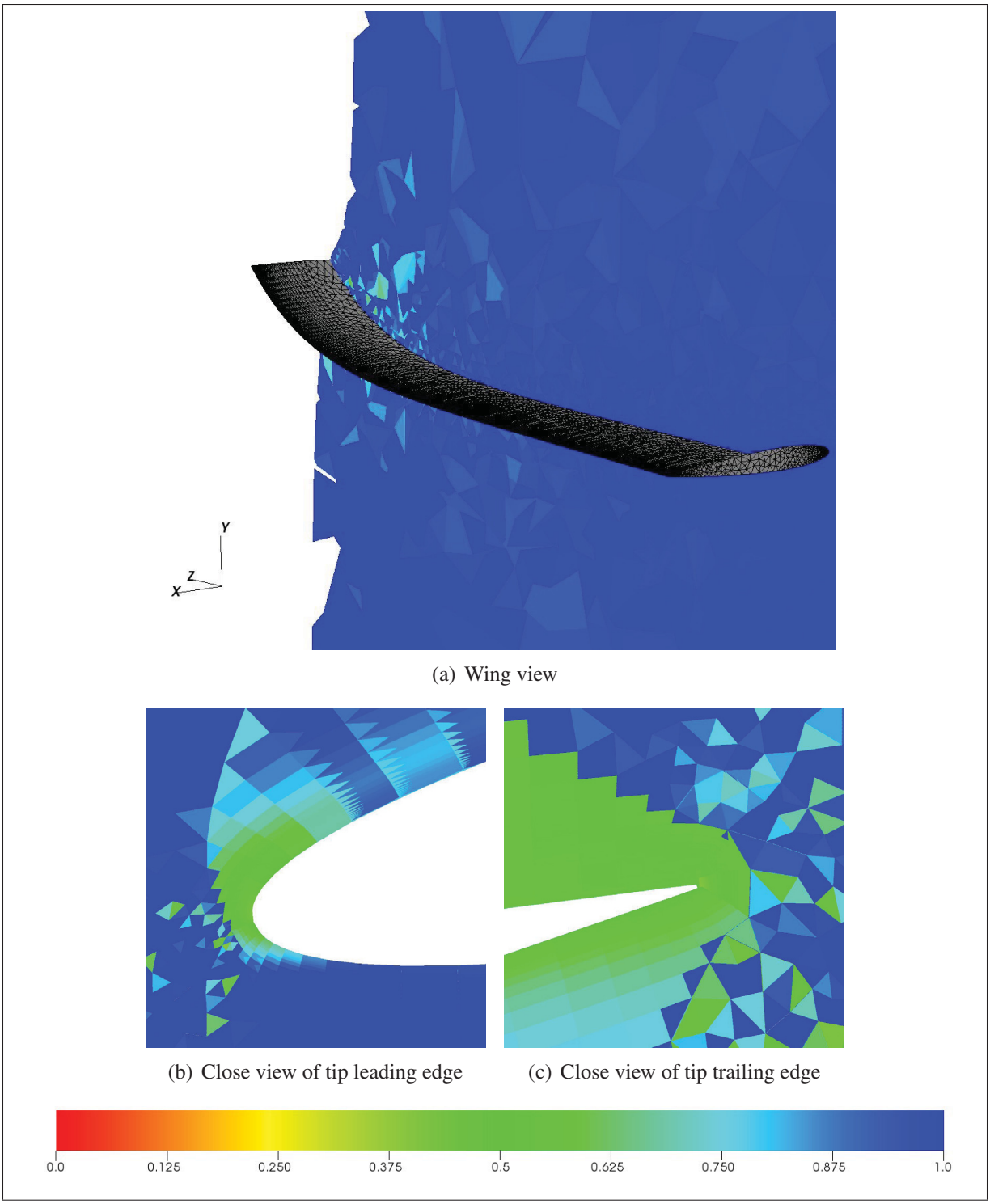



Figure 5.4 NACA0012 simulations: $q_{chgAbs}$ evolution for combined motion.

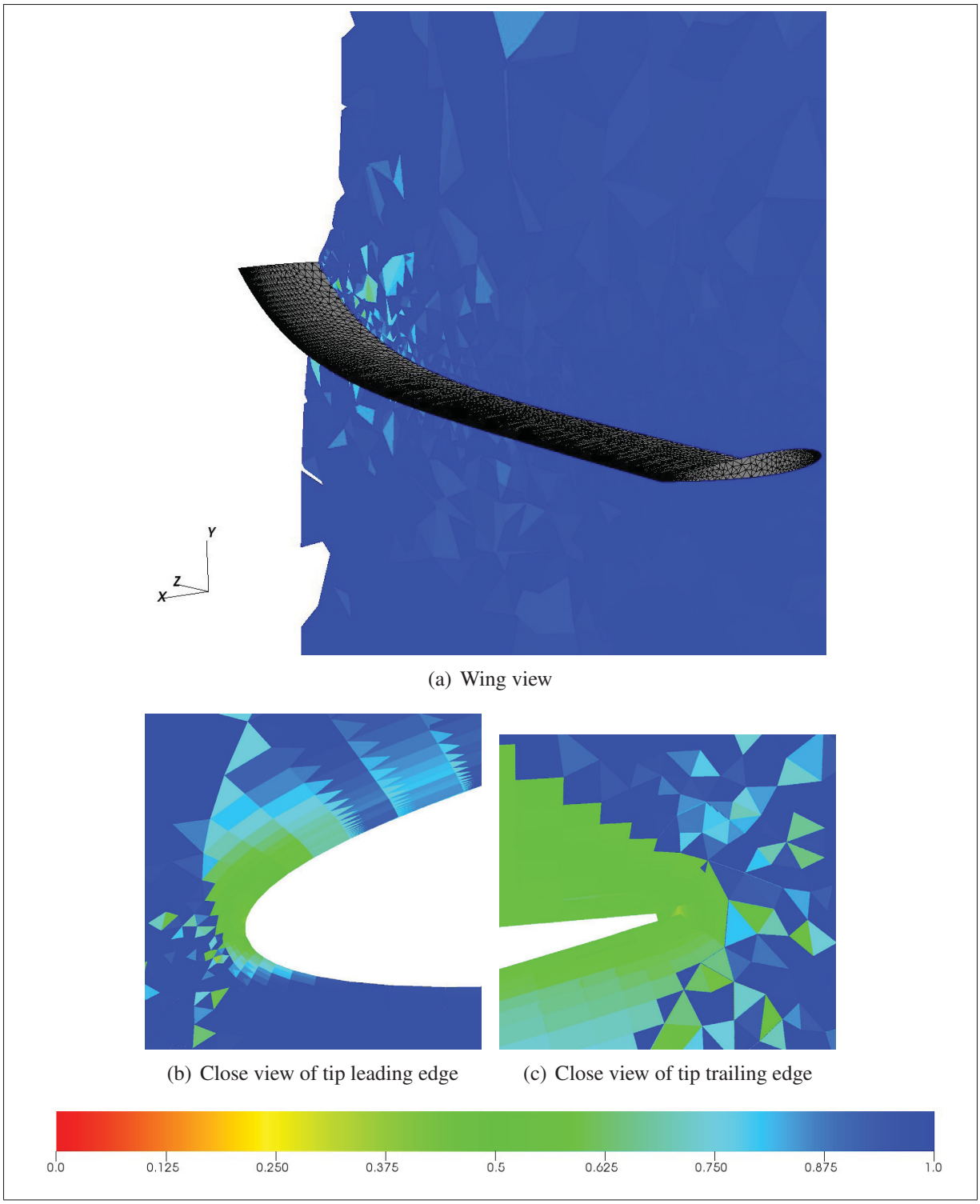



(a) Wing view

(b) Close view of tip leading edge

(c) Close view of tip trailing edge

Figure 5.5 NACA0012 simulations: $q_{chgAbs}$ field with IDW for the combined motion at $t = 0.5$.




(a) Wing view

(b) Close view of tip leading edge

(c) Close view of tip trailing edge

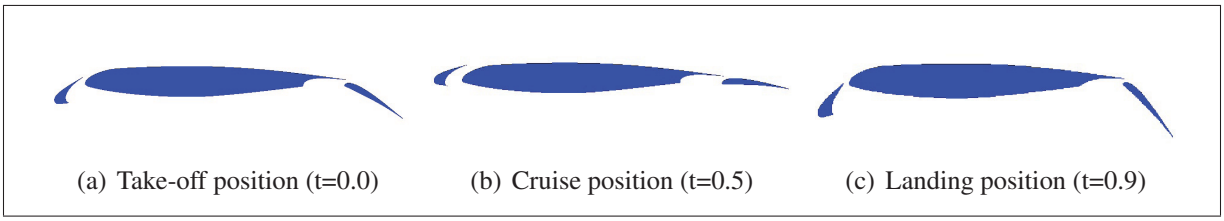
Figure 5.6 NACA0012 simulations: $q_{chgAbs}$ field with MSA-IDW for the combined motion at $t = 0.5$.

## 5.2 DLR-F11-HL profile extrusion from take-off to landing

The geometry is a wing section of the DLR-F11-HL which was used for the second AIAA High-Lift Prediction Workshop (Rumsey (2014)). The wing chord length is one unit and the span length is 0.5 unit. This simulation will show that the mesh validity is preserved after the application of an MMA for complex multi-body problem in pseudo-two dimensions.

### 5.2.1 Motions Definition

The flap and slat have imposed movements while the wing is fixed; these movements represent the stages of a full flight from take-off to landing (see Figure 5.7). The simulation starts at the take-off position, after 50 iterations it is in the cruise position and after 40 more iterations the wing body is ready for landing. A large number of iteration is used to apply the motion for high-lift components motion in order to allow a smoother quality evolution, to reduce the generation of negative elements and to increase the effect of smoothing algorithms on the deformed mesh.



(a) Take-off position (t=0.0) (b) Cruise position (t=0.5) (c) Landing position (t=0.9)

Figure 5.7 DLR-F11-HL simulation: Flight positions.

### 5.2.2 Mesh Description

From each moving boundary is extruded 34 layers of structured tetrahedron elements, where the first layer height is of $1.27 \times 10^{-5}$ and the growth rate is of 1.3. Moving boundary surfaces have their faces aspect ratios between 1.3819 and 23.92 which is necessary to ensure small impact from the smoothing algorithms on the surface mesh.

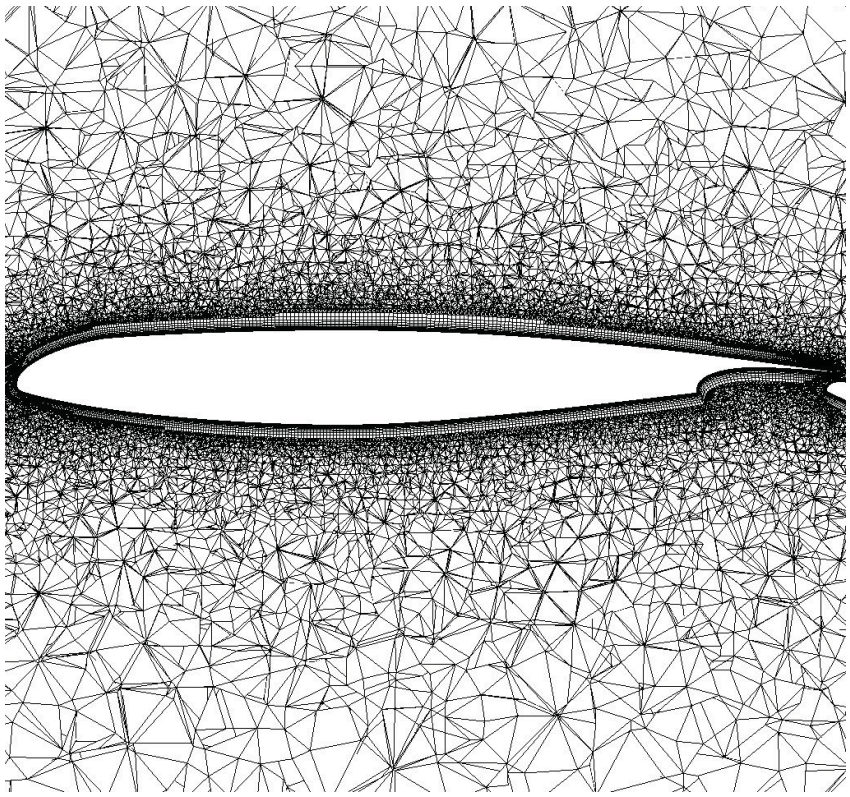
The domain limit is a box of dimensions $10.0 \times 10.0 \times 0.5$ and the complete mesh is composed of 6 370 052 elements (3 277 643 hexahedron, 18 487 prisms, 577 775 pyramids and 2 496 147 tetrahedron)and 3 993 494 nodes. In Figure 5.8 is presented the undistorted mesh around moving boundaries. The coarse mesh is composed of 18 layers of structured tetrahedron growing from the moving boundaries with a initial height of $1.27 \times 10^{-5}$ and a growth rate of 1.3 where the first 9 layers are only of prisms. Thus, the mesh is composed of 3 732 958 elements (960 008 prisms and 2 772 950 tetrahedron) and 995 359 points.

### 5.2.3 Mesh-mover algorithms parameters setting

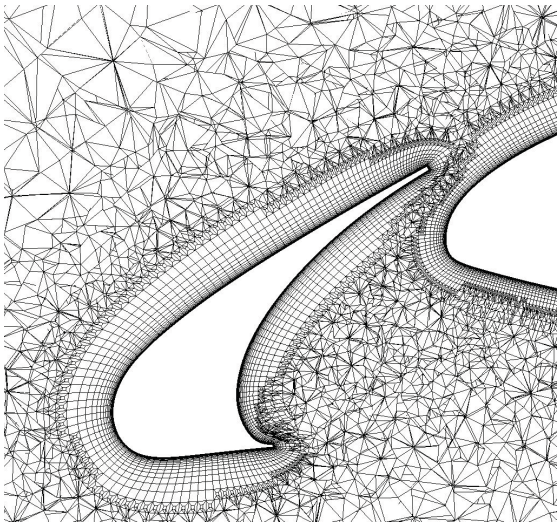
In Table 5.2 is presented the MMA parameters for the DLR-F11-HL simulation. The exponents are set to the values which produce the smoothest displacements field and $L_{ref} = 0.25$ to restrain the displacements field to be inside the quarter of the wing chord from each moving boundary. Then, $\alpha$ is being equal to 0.02 to make sure all moving boundaries have the same close boundary region definition which is equal to the boundary layers total height. Also, the slat and flap surfaces weight are raised which increase their influence on the displacements field.

Table 5.2 DLR-F11-HL simulation: MMA parameters

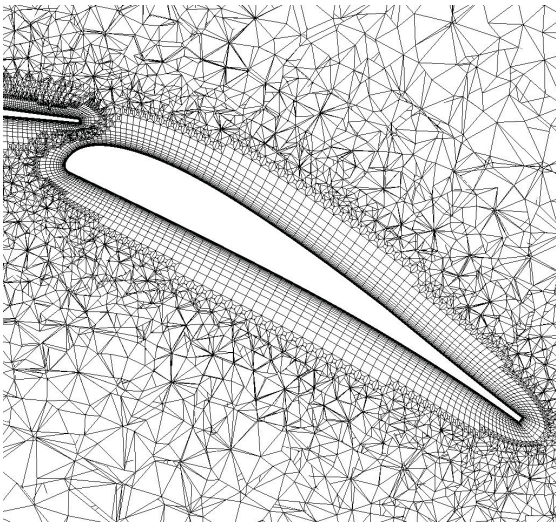
| **Algorithm** | **Parameters** | **Value** |
|---|---|---|
| IDW | | |
| | $a$ | 3.0 |
| | $b$ | 3.0 |
| | $L_{ref}$ | 0.25 |
| | $\alpha$ | 0.001 |
| | Slat Surfaces weight $A_i$ | $2.0 \times A_i$ |
| | Wing Surfaces weight $A_i$ | $1.5 \times A_i$ |
| Smoothing | | |
| | Quality minimal | smoothing when $q_{min} < 0.25$ until $q_{min} \geq 0.5$ |
| | Type quality | $q_{chgAbs}$ |
| | Stretched element | wall distance $\leq 1 \times 10^{-4}$ [m] |
| | Iterations | 50 to final position |
| | Reverse Iterations | 40 to go back to the original position |



(a) Wing view



(b) Slat view



(c) Flap view

Figure 5.8 DLR-F11-HL simulation: initial mesh around moving boundaries at $t = 0$.

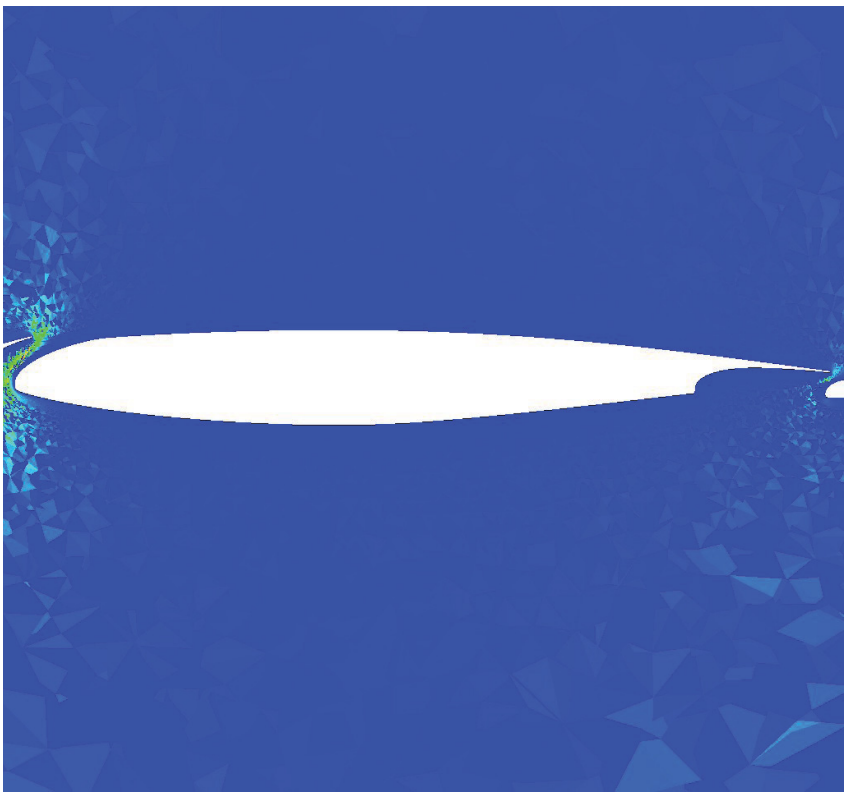
### 5.2.4 Results

Smoothing algorithms and the proposed NGU are proven to be necessary to solve such complex mesh motions. Without our proposed improvements to the IDW of Luke *et al.* (2012) such problems will not be solvable.

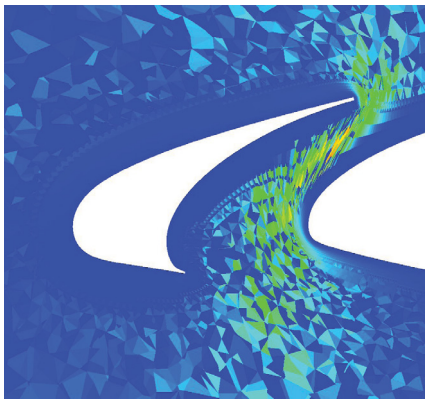
In Figures 5.9 to 5.11 is presented the deformed mesh at various time instant ($t = [0.5, 0.75, 0.90]$) with the IDW method combined to the smoothing GETMe and NGU algorithms. The element with the lowest $q_{chgAbs}$, especially at t=0.90, are between the slat and the wing, but their distortion is reduced as wanted by the GETMe smoothing algorithm. Elements around the flap do not suffer large deformation since the flap is not getting closer to the wing. Thus, the wing trailing edge and flap boundary layers elements' quality are almost unchanged. It is quite different for the wing leading edge boundary layers where the displacements are diffused really close to it, but the first layers of elements around the surfaces have a good quality of $q_{chgAbs} > 0.75$.

In Figure 5.12 is presented that from t=0.23 to t=0.65 the smoothing algorithm is not able to maintain the requested minimum quality, it is around $q_{chgAbs} = 0.02$ instead. Afterwards, up to retaliation of the original position (t=0.75) the proposed approach respects the quality criteria. Then, for the movement to landing position it is difficult to keep a good minimum; however the minimal quality is always positive thus the flow field may be solved.

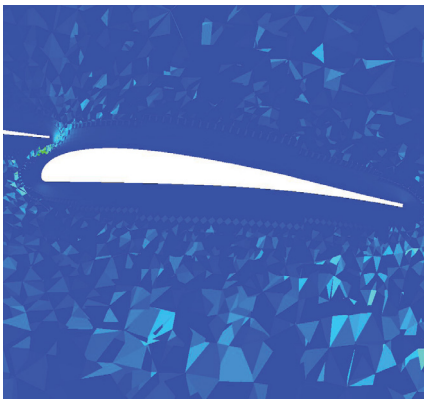
The solution from the MSA-IDW shows a similar quality evolution until t=0.30 where it decreases drastically, for then generate an unrecoverable mesh at t=0.50. Effectively, for such complex problem the coarse mesh interpolation produces a displacements field less smooth than using the IDW on the fine mesh. This difficulty may be corrected by using a finer coarse mesh or with a different interpolation approach.



(a) Wing view



(b) Slat view



(c) Flap view

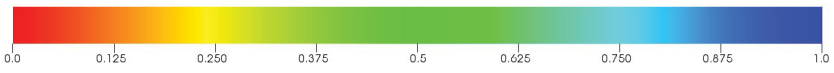
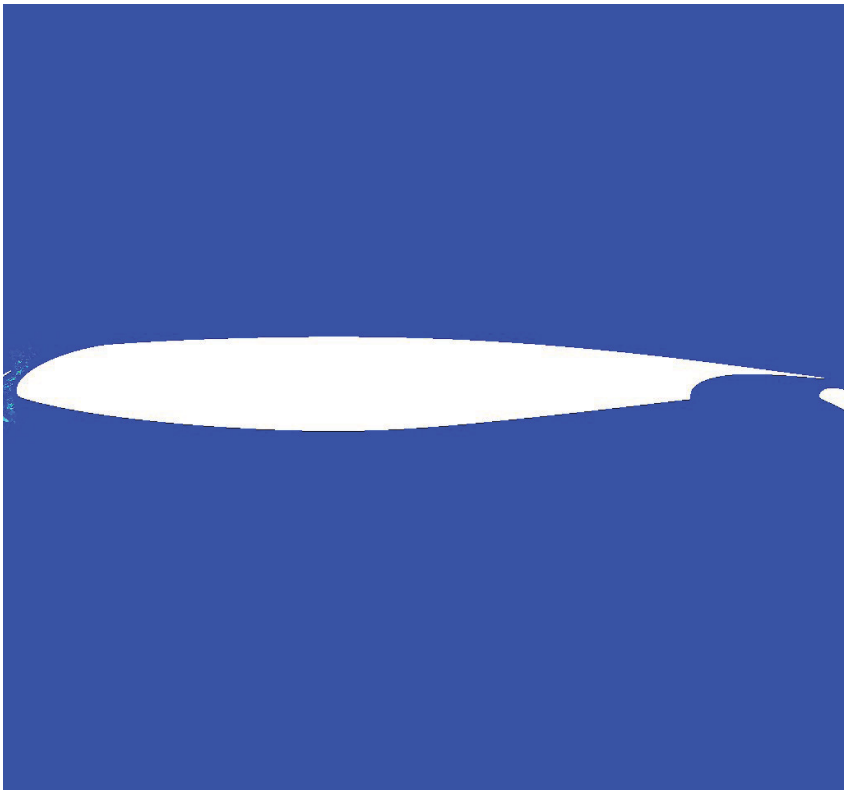



Figure 5.9 DLR-F11-HL simulation: $q_{chgAbs}$ field around moving boundaries at $t = 0.5$.



(a) Wing view

(b) Slat view

(c) Flap view

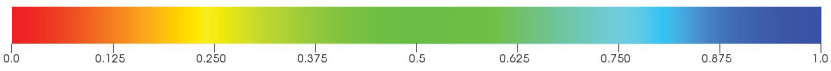
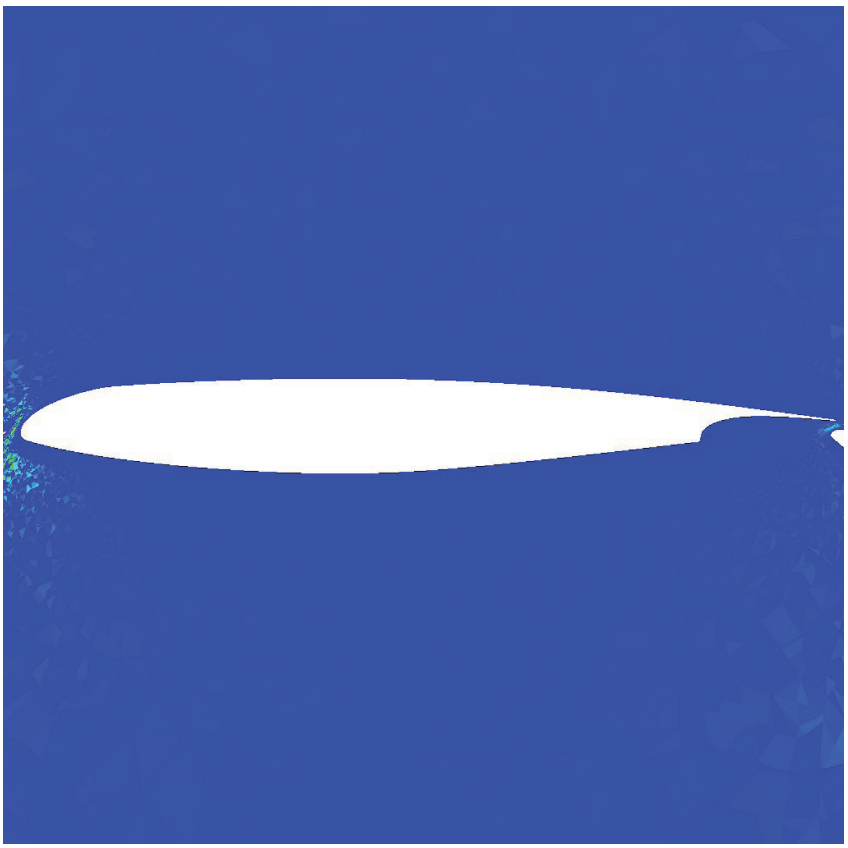



Figure 5.10 DLR-F11-HL simulation: $q_{chgAbs}$ field around moving boundaries at $t = 0.75$.



(a) Wing view

(b) Slat view

(c) Flap view

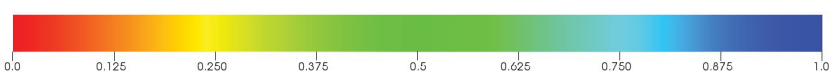



Figure 5.11 DLR-F11-HL simulation: $q_{chgAbs}$ field around moving boundaries at $t = 0.90$.

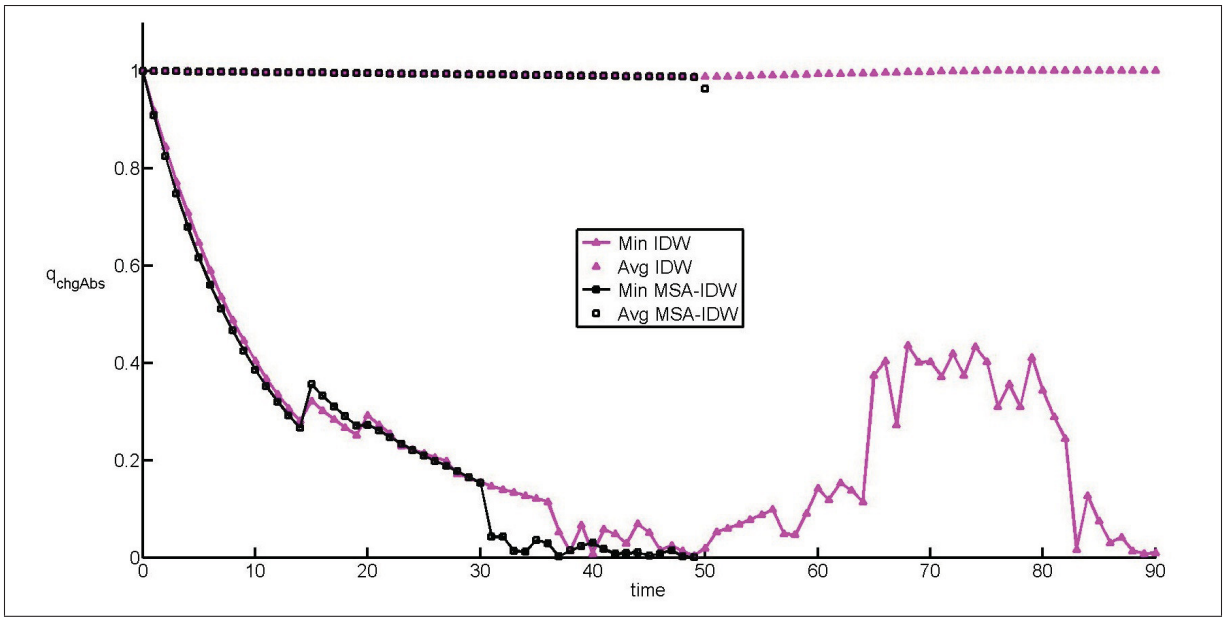



Figure 5.12 DLR-F11-HL simulation: $q_{chgAbs}$ evolution with the proposed improvements.

## 5.3 Wing-body motion from take-off to landing

In the two previous simulations, results of meshes under large three dimensional motions and complex multi-body interactions have been presented. Our last simulation will be to investigate an advanced moving boundaries problem: a multi-body interaction of a three dimensional motion for a complex geometry with CFD solving.

### 5.3.1 Motions Definition

The trapezoidal wing (see Figure 5.13) geometry is defined by the HiLift-PW1 (Rumsey (2014)) and the motion of each body is: a rotation of -30 degrees of the flap around the wing leading edge, the wing is fixed and the flap is rotated 20 degrees around its leading edge. A translation in the z-axis of 12.7[mm] is added to the motion of all bodies to protect the mesh from penetrating the fuselage surface. These manufactured motions represent the positioning of each high-lift component to different flight stages and are demonstrated in Figure 5.14 at the wing tip ($z = 84.0$).

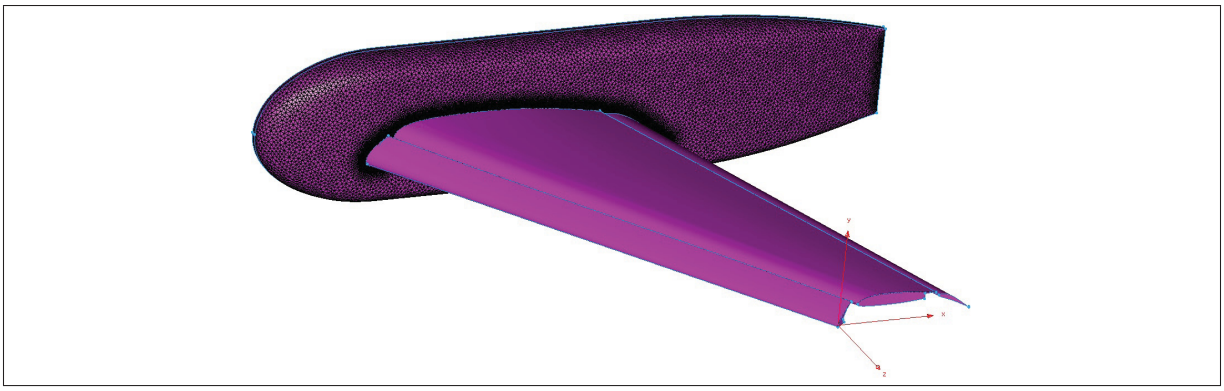
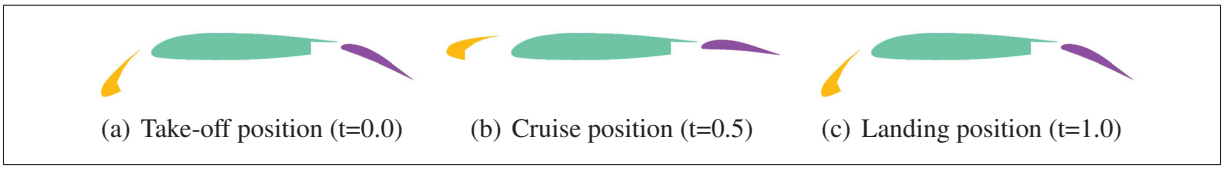


Figure 5.13 Wing-body simulation: Geometry.



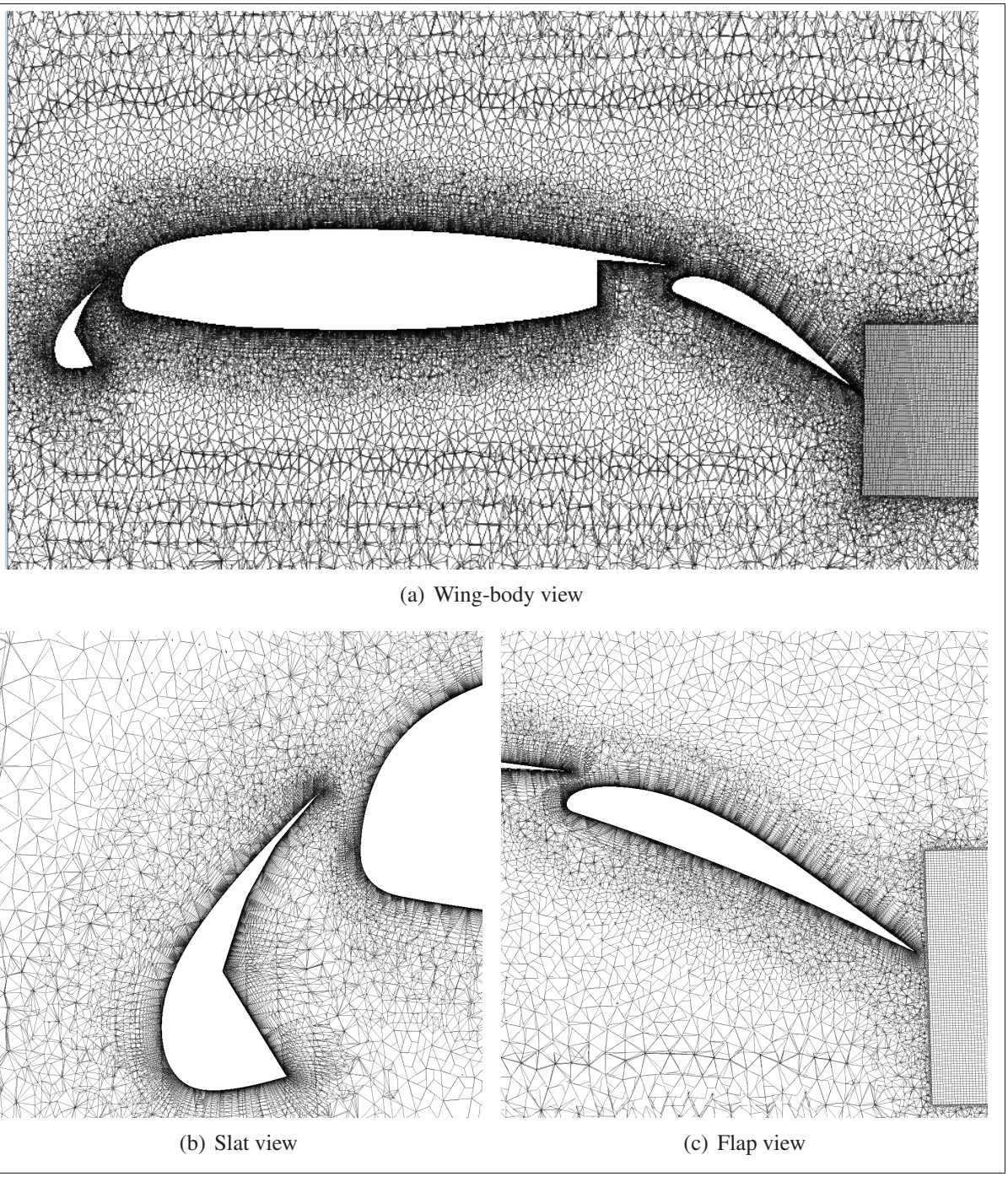
(a) Take-off position (t=0.0) (b) Cruise position (t=0.5) (c) Landing position (t=1.0)

Figure 5.14 Wing-body simulation: Flight positions at tip (z=84.0).

### 5.3.2 Mesh Description

49 structured tetrahedral layers are extruded from each moving boundary to generate the fluid field and most of them will be combined into prism elements at exportation. The first layer height is of $5 \times 10^{-6}(2 \times 10^{-4}$ [in]), the growth rate is of 1.2 and moving boundary surfaces have faces aspect ratios between 1.00 and 37.89. Additionally, a block of hexahedron behind the flap has been inserted to allow the wake flow to be resolved.

The domain limit is a box of dimensions $250 \times 250 \times 100$ and the complete mesh is composed of 40 593 682 elements (3 504 600 hexahedron, 29 511 057 prisms, 10 607 197 tetrahedron and 475 428 pyramids) and 20 420 367 nodes. The undistorted mesh around moving boundaries is presented in Figures 5.15 and 5.16. The MSA-IDW is not tested since the current implementation replicate the coarse mesh on each processors which uses more memory than the amount available.

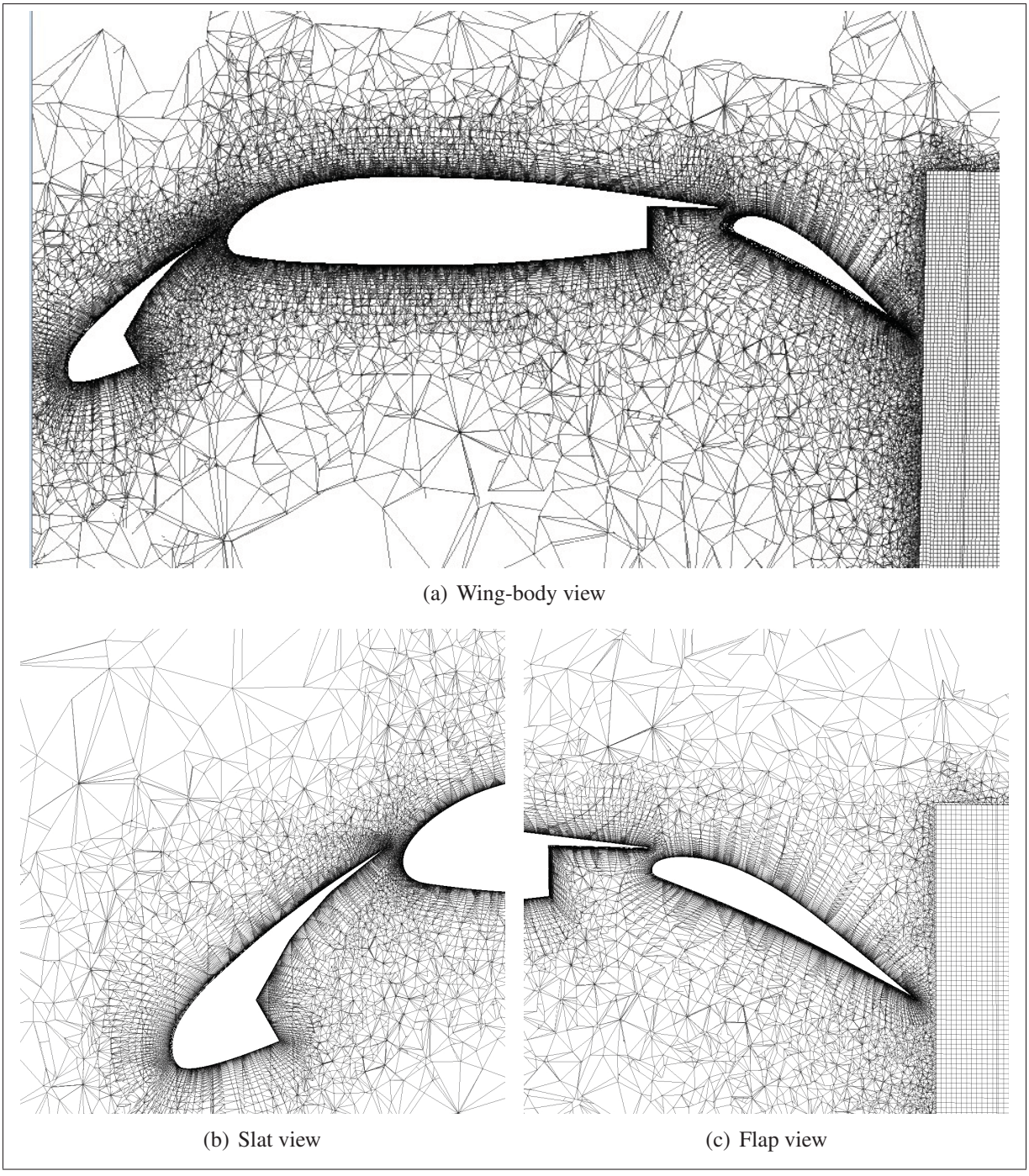


(a) Wing-body view

(b) Slat view

(c) Flap view

Figure 5.15 Wing-body simulation: mesh around moving boundaries at root (z=8.0) for $t = 0$.



(a) Wing-body view

(b) Slat view

(c) Flap view

Figure 5.16 Wing-body simulation: mesh around moving boundaries at tip (z=84.0) for $t = 0$.

### 5.3.3 Mesh-mover algorithm parameters

In the Table 5.3 is presented the parameters configuration for the trapezoidal wing simulation. The exponents which produce a better mesh quality are $a = 4$ and $b = 5$, $L_{ref} = 0.5[m]$ in order to reduce the shearing of elements between bodies and the other parameters are set to make the mesh-movement computation robust and fast.

Table 5.3 Wing-body simulation: MMA parameters

| **Algorithm** | **Parameters** | **Value** |
|---|---|---|
| IDW | | |
| | $a$ | 4.0 |
| | $b$ | 5.0 |
| | $L_{ref}$ | 0.5[m] |
| | $\alpha$ | 0.001 |
| Smoothing | | |
| | Quality minimal | smooth when $q_{min} < 0.125$ until $q_{min} \geq 0.25$ |
| | Stretched element | wall distance $\leq 0.00127$ [m] |

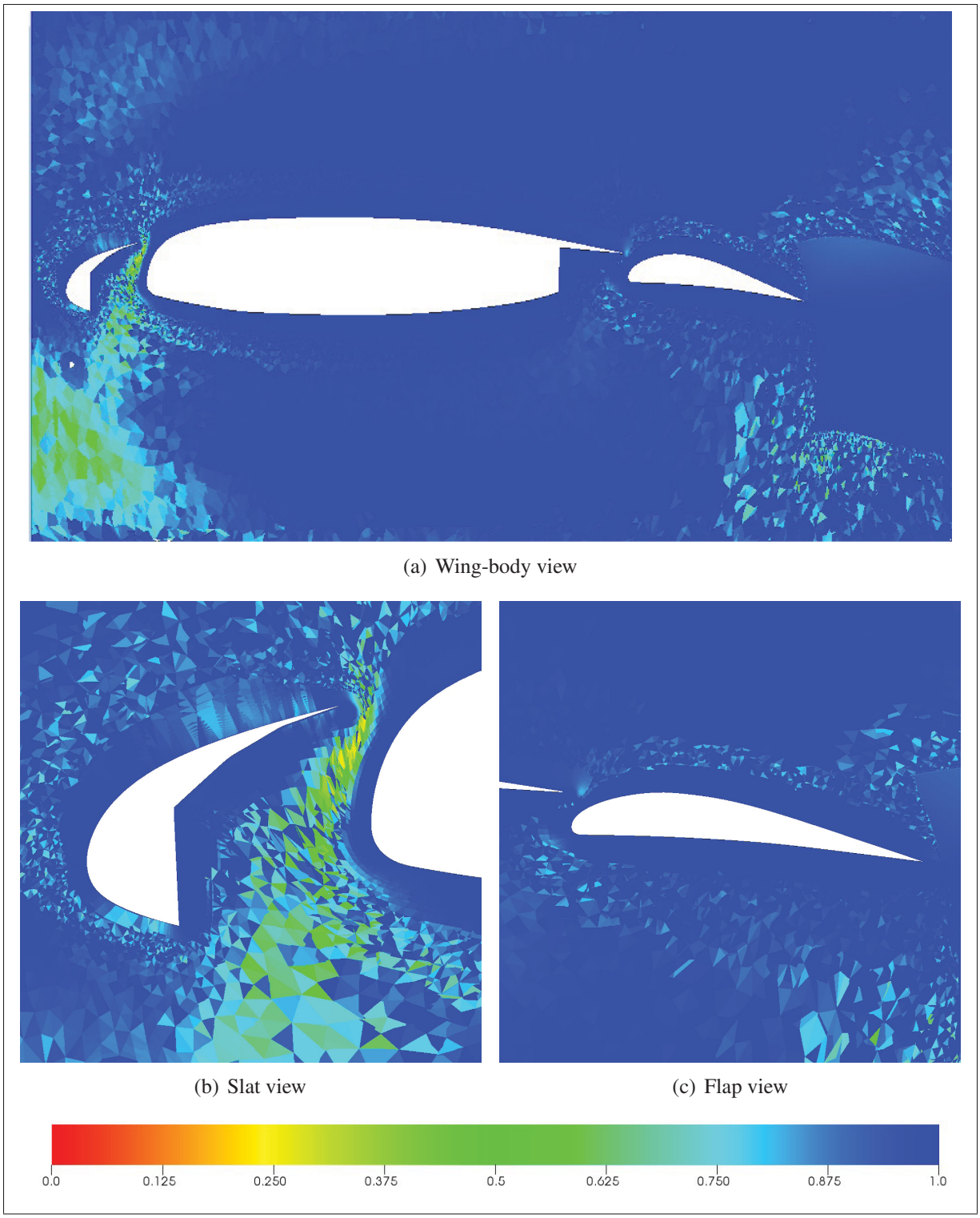
### 5.3.4 Results

In Figures 5.17 to 5.20 are presented the deformed mesh at cruise ($t = 0.5$) and landing (t=1.0) positions for the root and tip of the wing. It is seen that boundary layers elements have their quality unchanged and that between bodies is the zone of high sheared elements of poor quality that could affect the fluid calculation especially between the slat and the wing which is due to the proximity of the bodies. Also, the crucial zone is at the tip between bodies because the bodies are closer and it is the zone with the lowest quality. Thus, the mesh-movement could fail in this zone if the mesh is not generated with care. Then, in Figure 5.21 it is presented that the mesh quality is kept over $q_{chgAbs} = 0.125$ except from t=0.29 to t=0.7. This can be caused by the fact that the smoothing algorithms are not able to smooth bad quality elements or the quality metric is not defined well enough for this situation.

However, the mesh returns at the asked minimum quality of $q_{min} = 0.25$ for t=1.0 which means that this deformed mesh should performs almost as good as the initial one to solve the fluid flow, but it is difficult to state if the deformed mesh of t=0.5 will produce good flow results.

To complete this section, the computation time for the simulation per major operations is presented in Figure 5.22. The simulation has been performed on a cluster of computers by 80 processors in parallel. The first operation done is the initialisation which explains a higher computational time for the first iteration than the second. Then, the mesh displacement is computed in less than two minutes for almost ten million nodes and this operation computational time is constant for each iteration. The mesh is smoothed few times outside of the critical zone, t=0.29 to t=0.7, and depending on the quantity of loop it takes more or less time, but this operation takes 18 minutes on average which means there can be optimisation to do. However, this time is lower than generating, exporting and reading a new mesh.

The untangling operations similarly to the smoothing operation vary per iteration depending on the quantity of negative elements and the ease of untangling them, but it takes always less than 7 minutes to untangle the mesh which is fast enough. Finally, the untangling operation is shown to be necessary from time t=0.28 until the end since all elements are required to be of positive volume for the fluid flow solver.

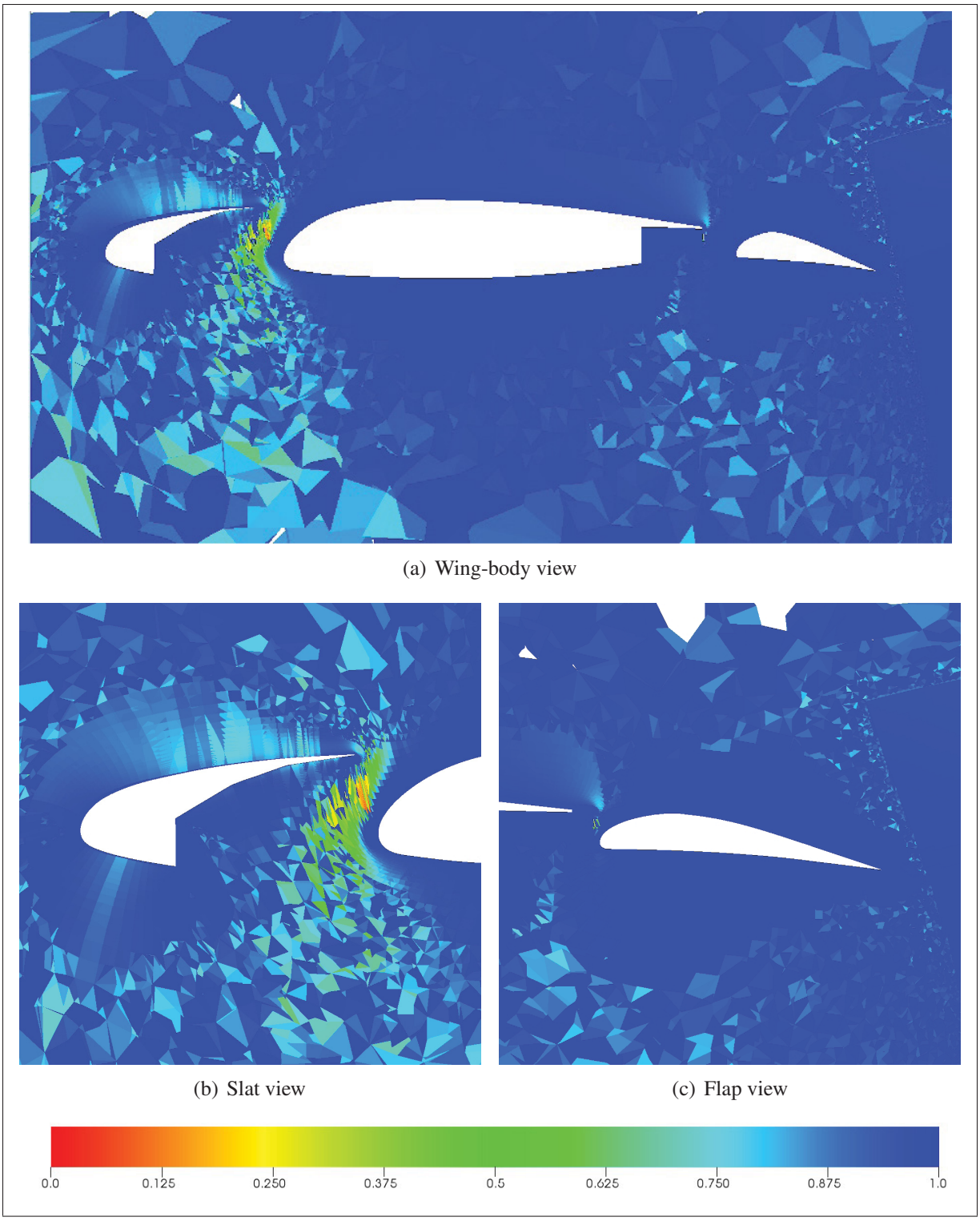



(a) Wing-body view

(b) Slat view

(c) Flap view

Figure 5.17 Wing-body simulation: $q_{chgAbs}$ field around moving boundaries at root (z=8.0) for $t = 0.5$.

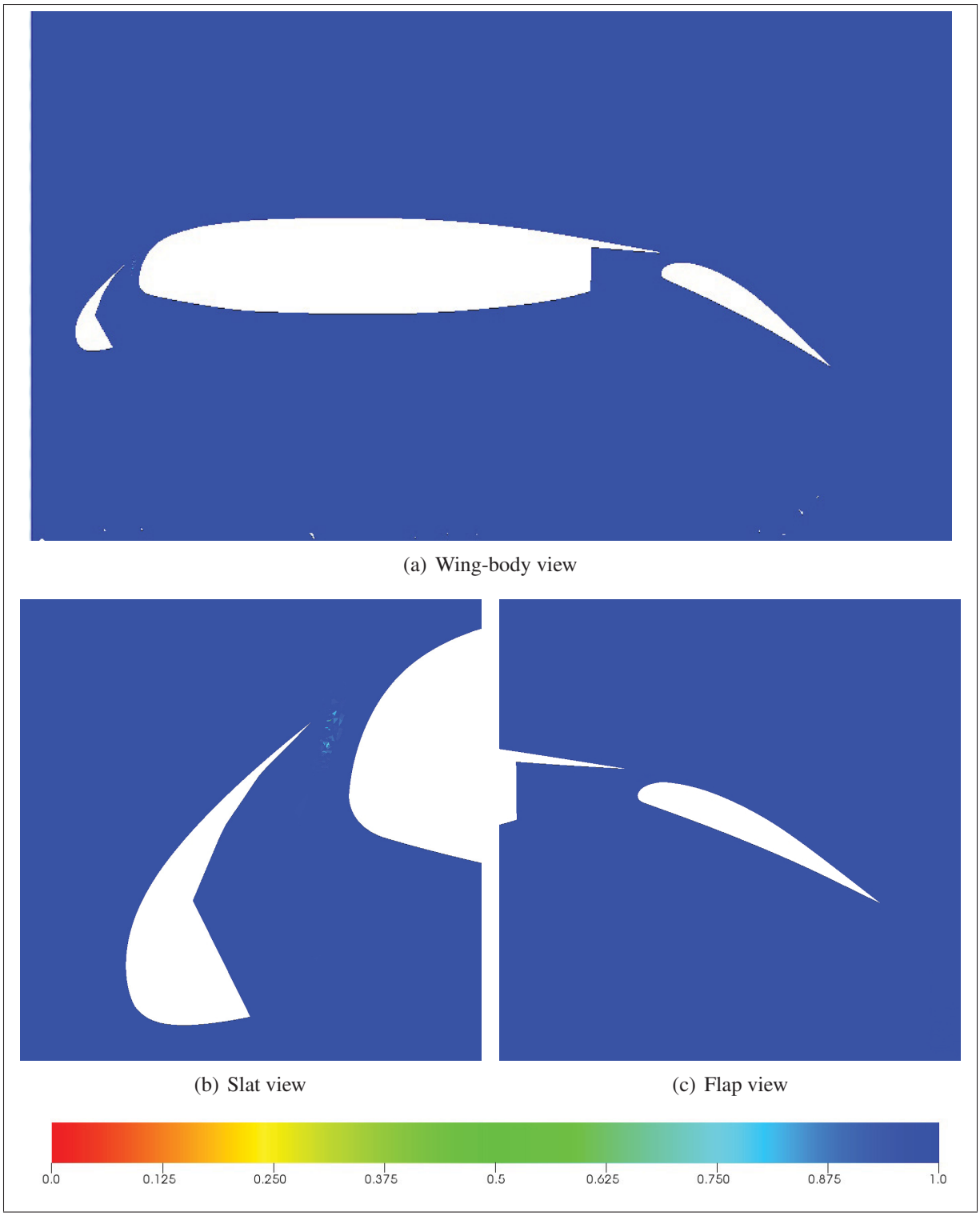



(a) Wing-body view

(b) Slat view

(c) Flap view

Figure 5.18 Wing-body simulation: $q_{chgAbs}$ field around moving boundaries at tip (z=84.0) for $t = 0.5$.

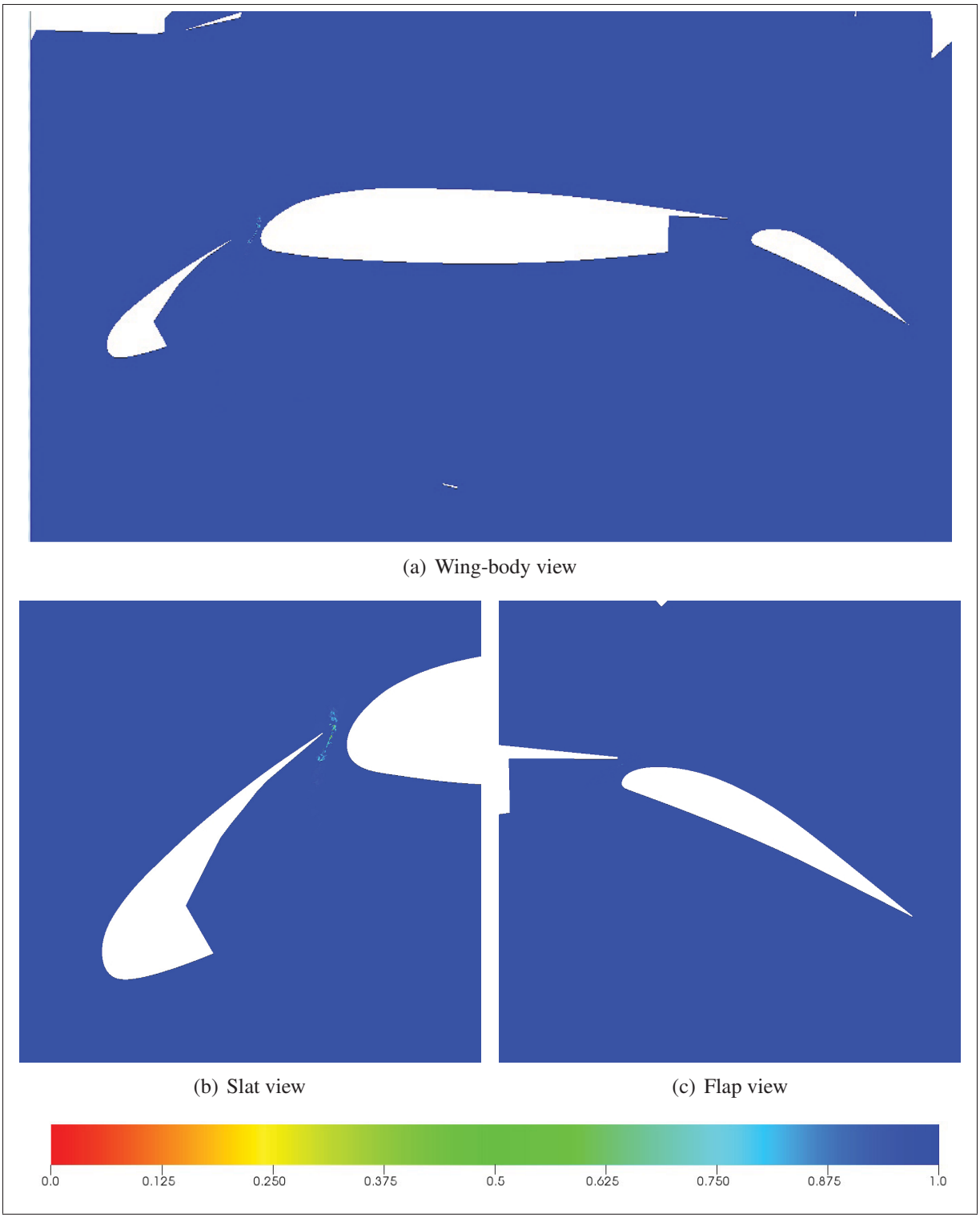



(a) Wing-body view

(b) Slat view

(c) Flap view

Figure 5.19 Wing-body simulation: $q_{chgAbs}$ field around moving boundaries at root (z=8.0) for $t = 1.0$.




(a) Wing-body view

(b) Slat view

(c) Flap view

Figure 5.20 Wing-body simulation: $q_{chgAbs}$ field around moving boundaries at tip (z=84.0) for $t = 1.0$.

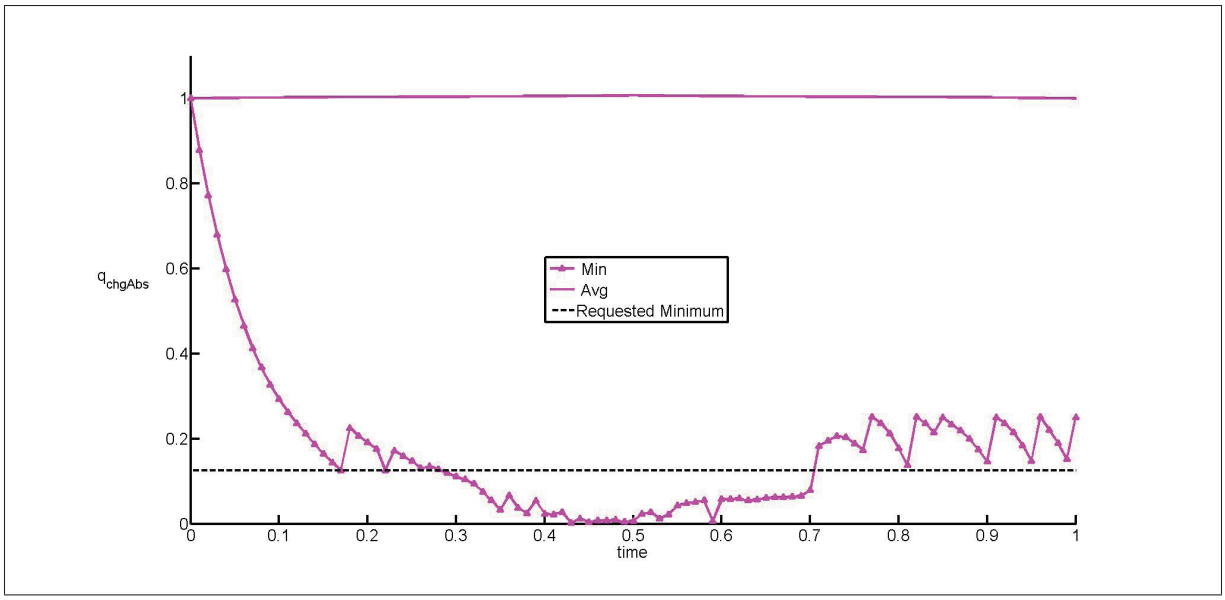



Figure 5.21 Wing-body simulation: $q_{chgAbs}$ evolution with the proposed improvements.

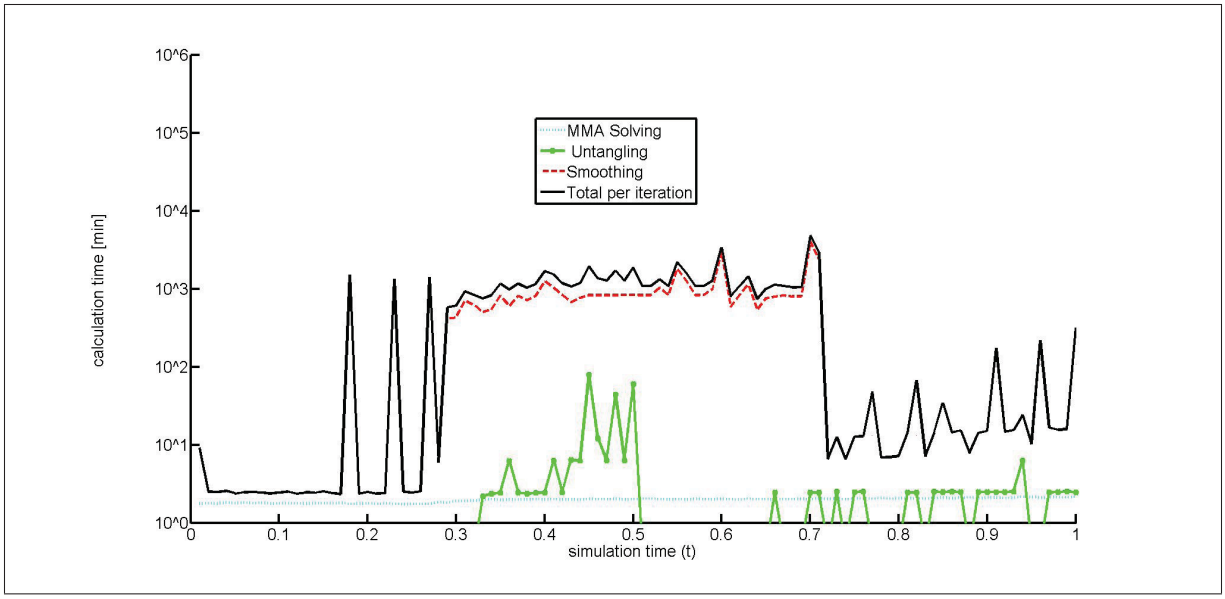



Figure 5.22 Wing-body simulation: Computation time per operations.

### 5.3.5 Fluid flow solution

This last section will prove that our methodology is suitable for the needs of the industry which is to solve fluid flows of wing-body. The flow characteristics are shown in Table 5.4 where *MAC* is the mean aerodynamic chord which is equal to 39.54 [in] (approx. 1[m]). The ANSYS CFX

Table 5.4 Wing-body simulation: fluid flow characteristics

| **Parameters** | **Value** |
|---|---|
| $M_\infty$ | 0.2 |
| Angle of attack | 13.0 [deg] |
| $Re_{MAC}$ | $4.3 \times 10^6$ |
| $P_\infty$ | 101 325 [Pa] |
| $\nu_\infty$ | $1.5743 \times 10^{-5} [\frac{m^2}{s}]$ |
| $T_\infty$ | 540 [R] |

software will be used to resolve the flow around the wing-body. The turbulence model used is the SST model, the resolution strategy for the advection and turbulence equations is high resolution and the physical timescale for convergence is equal to $\frac{MAC}{U_\infty \cdot 10} = .0015[s]$. The high resolution scheme of CFX tries to increase the discretization to second order when possible and if not it used the upwind discretization.

The simulations evaluated are the undeformed mesh ($t = 0.0$) flow, the flow of the deformed mesh at cruise position ($t = 0.5$) and the deformed mesh flow of $t = 1.0$. The solution for each simulation is observed after 1000 iterations and the convergence attained for all equations is under (or close to) $10^{-4}$. In Figure 5.23 is shown the convergence evolution for the initial position ($t = 0.0$) simulation and other simulations convergence evolution are similar thus they are not shown. The shown convergence is for the RMS residuals of variable of interest: the velocities (U,V,W) and the pressure (P). The flow is analysed by presenting various global results at $t = 0$ and $t = 0.5$ in Figures 5.24 to 5.31. In these figures, the turbulence intensity is defined as $I = \frac{\sqrt{\frac{2}{3} \cdot k_{turb}}}{U_L}$, where $k_{turb}$ is the turbulent kinetic energy and $U_L$ is the local mean velocity and the vorticity is the curl of the velocity.

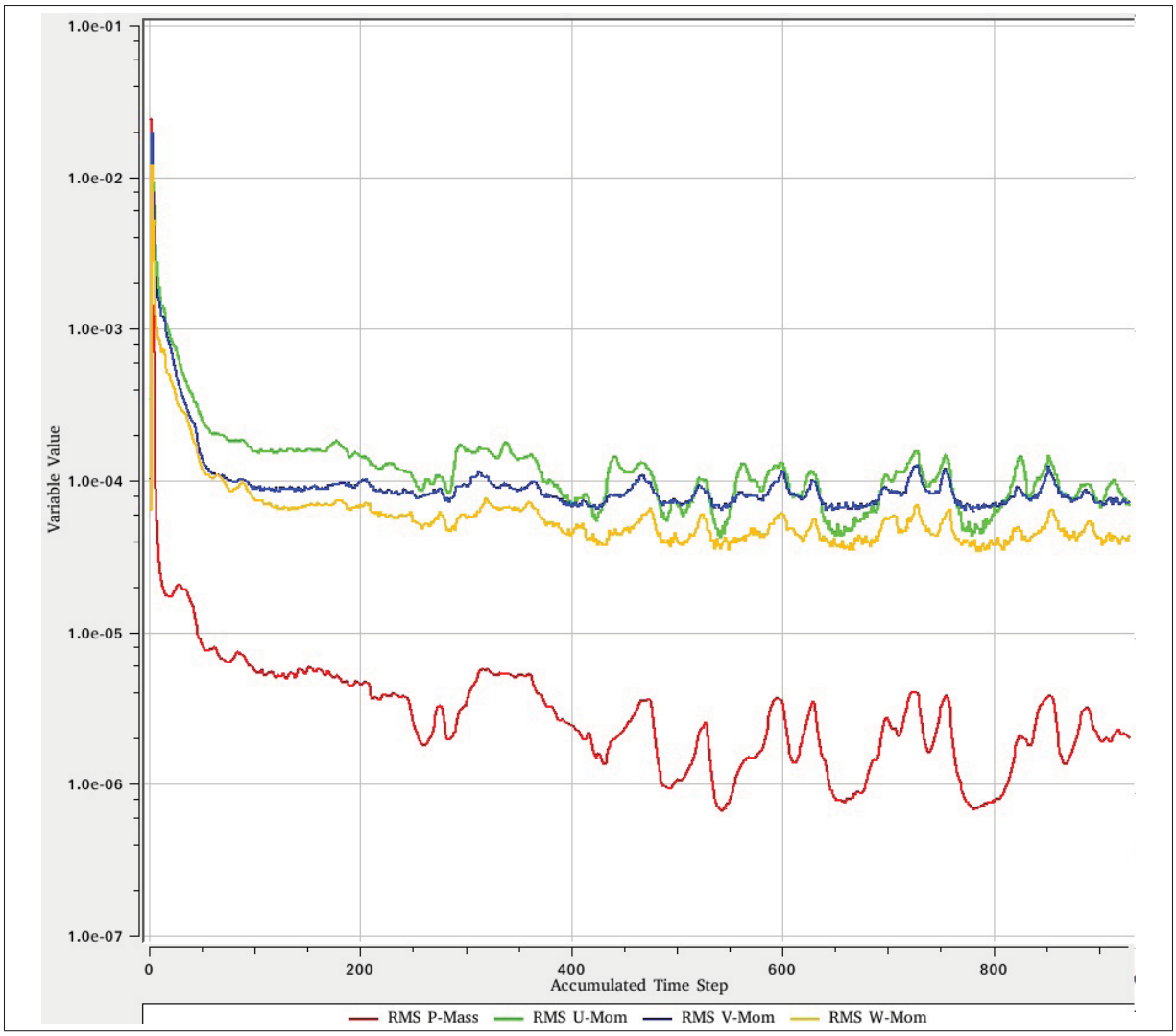



Figure 5.23 Wing-body simulation: ANSYS CFX fluid flow convergence for $t = 0.0$.

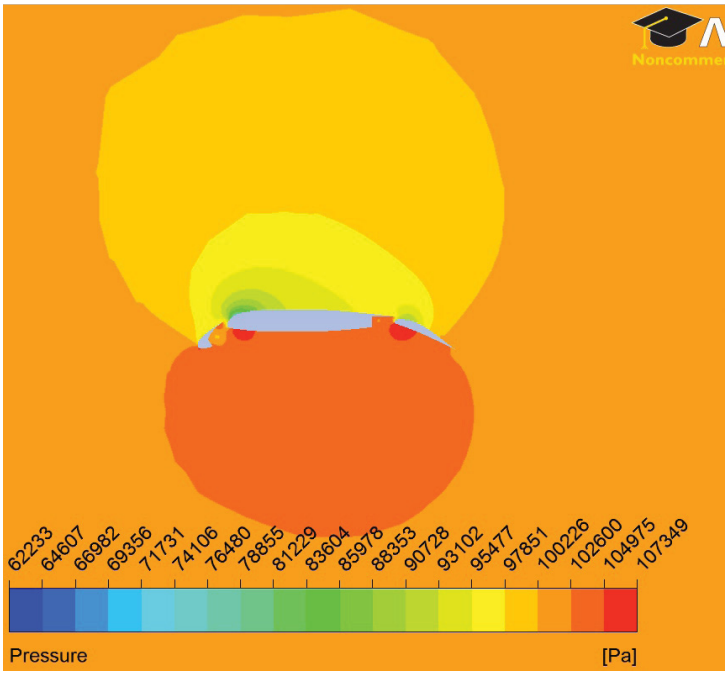



Figure 5.24 Wing-body simulation: Pressure field at z/span = 0.5 and t=0.0.

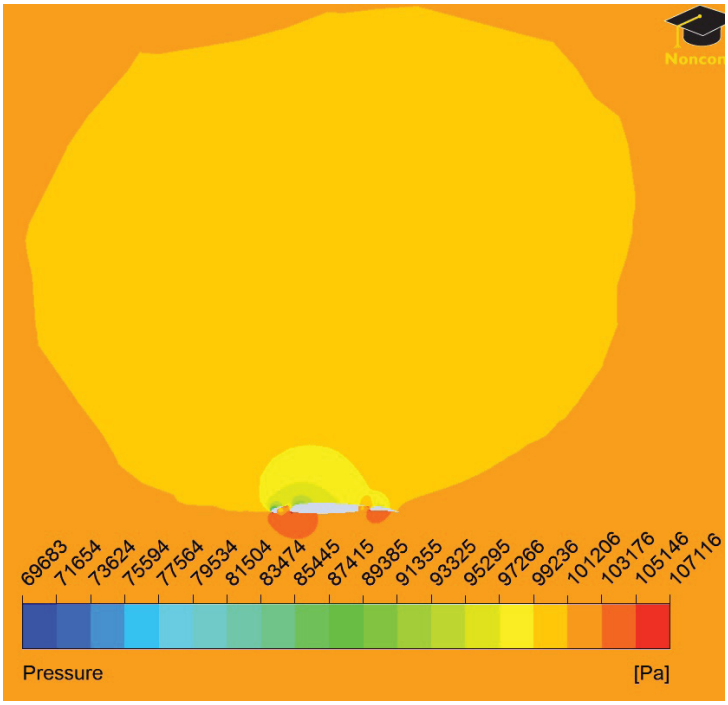



Figure 5.25 Wing-body simulation: Pressure field at z/span = 0.5 and t=0.5.

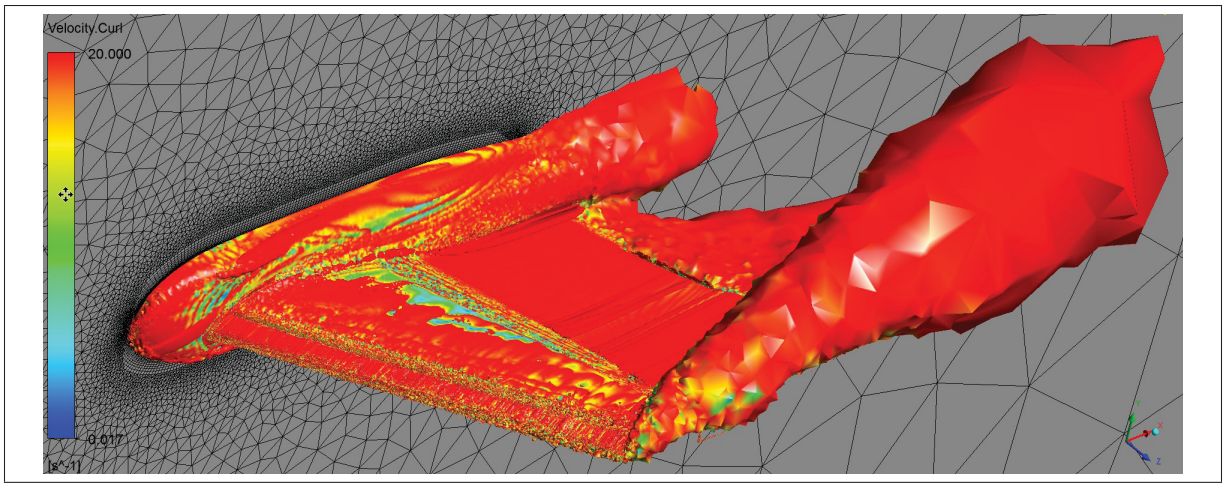



Figure 5.26 Wing-body simulation: Vorticity iso-surface of 20[1/s] at z/span = 0.5 and t=0.0.

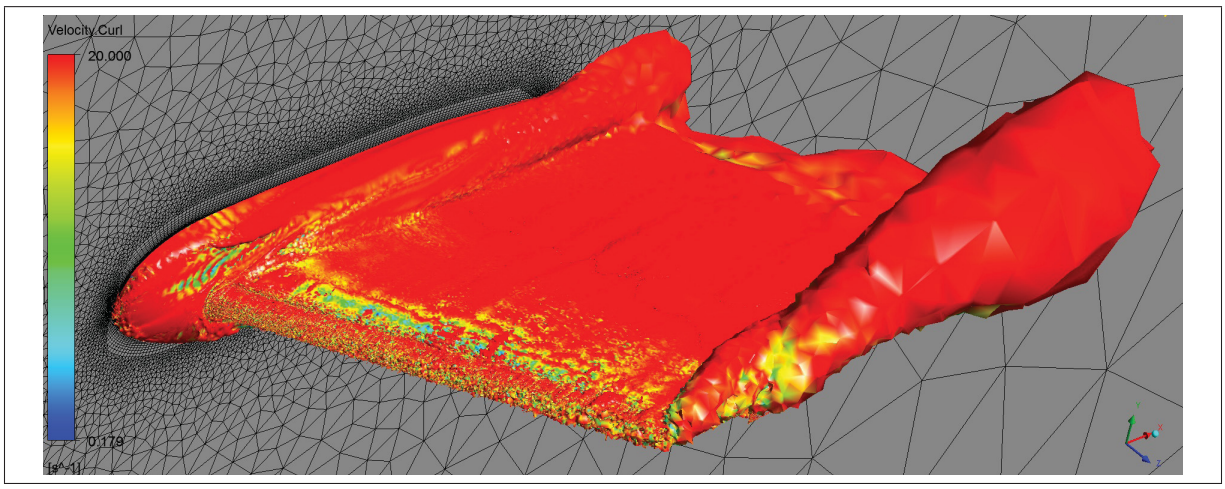



Figure 5.27 Wing-body simulation: Vorticity iso-surface of 20[1/s] at z/span = 0.5 and t=0.5.

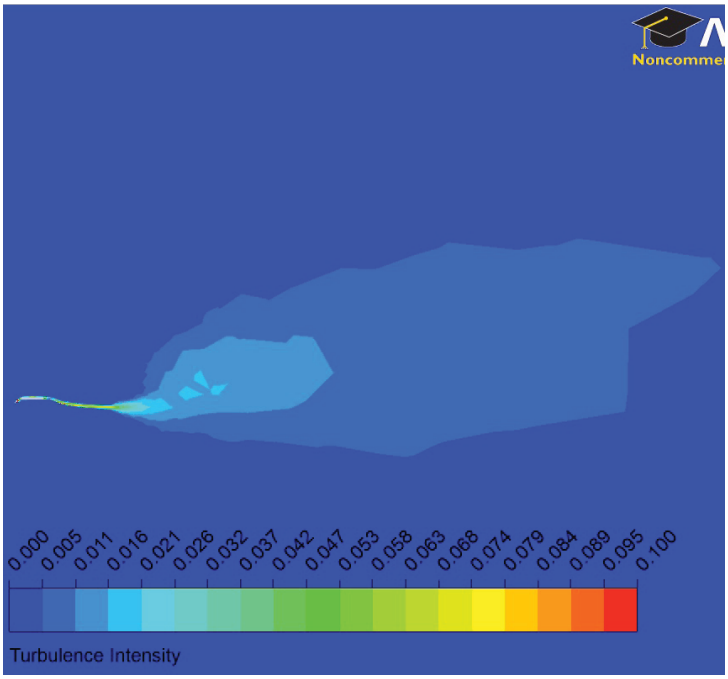



Figure 5.28 Wing-body simulation: Turbulence intensity field at z/span = 0.5 and t=0.0 (far view).

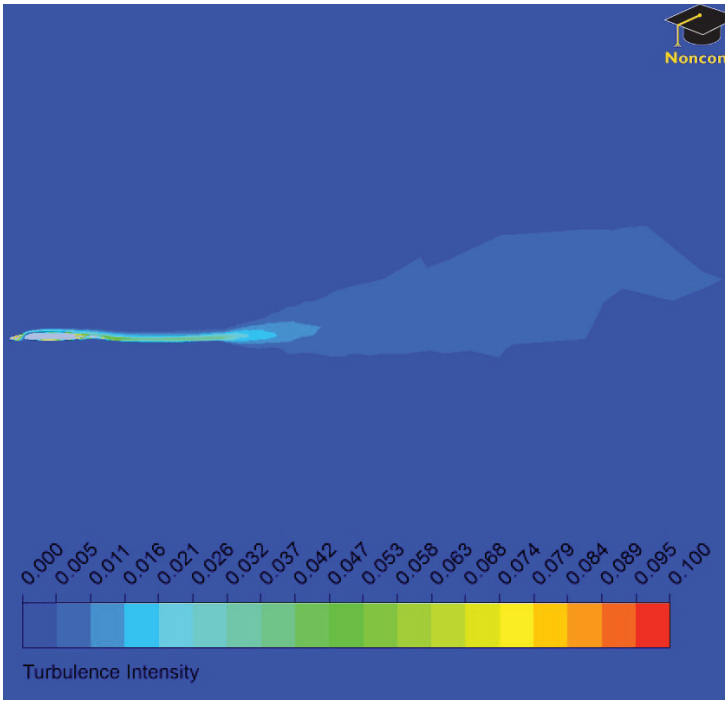



Figure 5.29 Wing-body simulation: Turbulence intensity field at z/span = 0.5 and t=0.5 (far view).

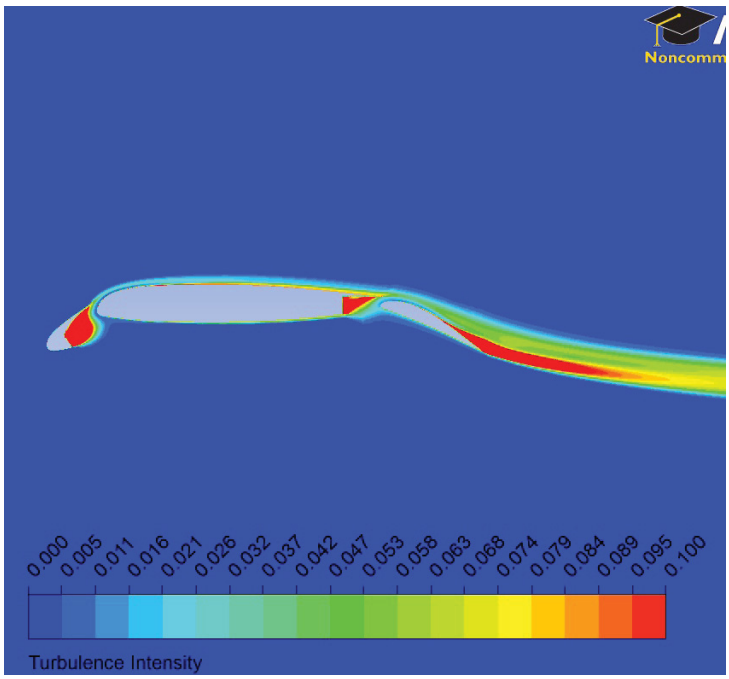



Figure 5.30 Wing-body simulation: Turbulence intensity field at z/span = 0.5 and t=0.0.

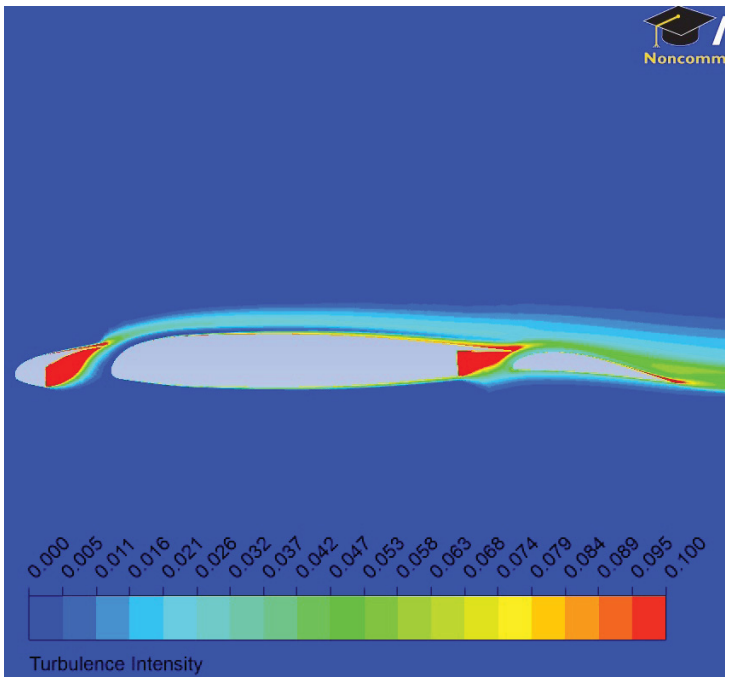



Figure 5.31 Wing-body simulation: Turbulence intensity field at z/span = 0.5 and t=0.5.

The previous figures are presented to give to others additional comparisons. Concerning pressure, the bubbles of depression on top of the wing are bigger compared to the overpressure under the lifting surfaces which explained the values of lift coefficient larger than one. From the vorticity surfaces it is seen that the vorticity is mainly produced parallel to the flow except at wing tip. Also the size of the vorticity region is more concentrated on the wing-body for t=0.5. These surfaces are the zones where the mesh need to be refined in order to improve the accuracy of the results. Then, the figures of turbulence show that the solution has not attained the steady state since the turbulence is not solely concentrated close to the surfaces. This can be explained by the fact that the problem to be solved is not steady, thus the solution will always show some sign of turbulence generation. Also, it can be caused by the fact that the mesh is considered coarse in the viscous layer (Rumsey (2014)).

In Figures 5.32 to 5.37 is presented the pressure coefficient ($C_p$) at different sections of interest. The graphics of $t = 0$ are compared to wind tunnel data and for $t > 0$ the $C_p$ behaviour is compared to current numerical results of $t = 0$.

The numerical results are close to the experimental results, for the flap and the wing the values are almost identical except for the tip sections ($z/span = [0.7, 0.9]$) which is acceptable to validate our mesh-movement approach. The pressure coefficient results for the slat section show a large offset of the data but the shape of the results is similar to experimental values, except at the tip where the depression of the extrado is not captured accurately.

As the flow field figures before, the pressure coefficient at $t = 0.5$ is similar to $t = 0$, except for the magnitude of values. The shape is slightly different at the slat tip which can be corrected by generating a better mesh in that zone, but this effect can be physical also. The mesh at t=1.0 has returned to its original position after deformation and it is proven validated without a doubt that a deformed mesh produces good results for such a difficult flow since the values of $C_p$ are identical.

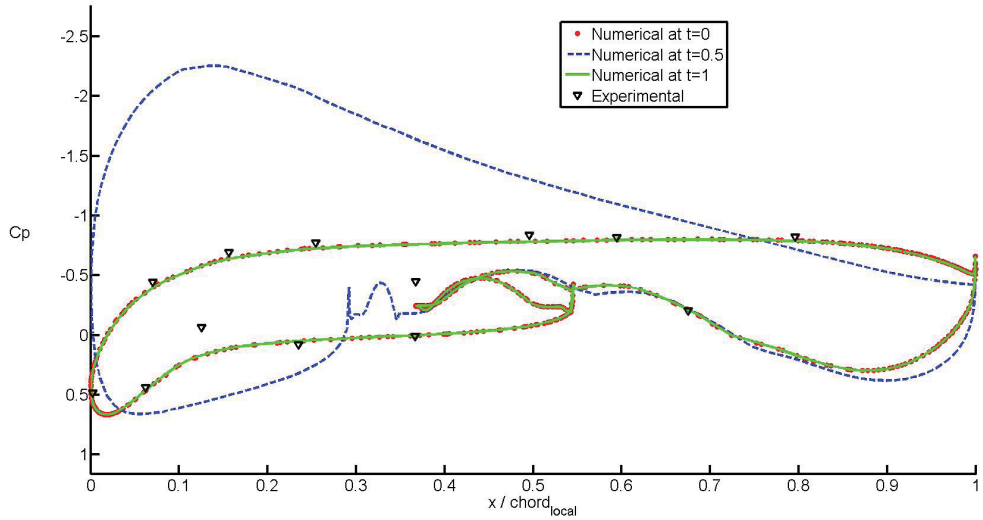
To complete the study, the lift and drag coefficients (pressure and viscous) have been calculated for all cases and compiled in Table 5.5. For t=0, the coefficients are really close to experimental values especially the lift. For t=0.5, as expected the values of both coefficients decrease of 46.85% and 27.15% since the angle of attack is unchanged. We consider that those values are realist, thus proving the workability of a deformed mesh with our method, but they should be compared to an undeformed mesh. Finally, the coefficients for t=1.0 are a little less close to experimental values than the original mesh. It means the smoothing algorithms have to be improve to reduce this discrepancy.

Table 5.5 Wing-body simulation: Drag and Lift coefficients

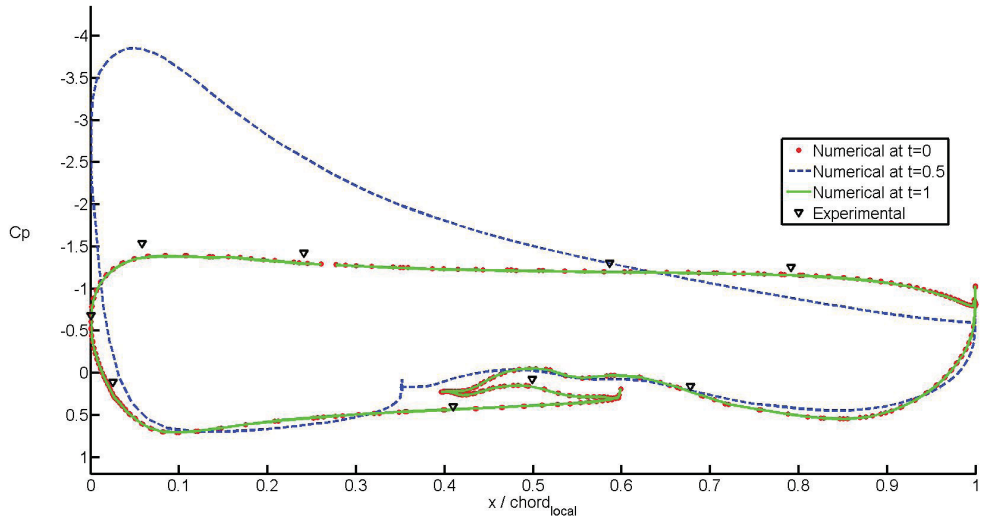
| **Test** | $\mathbf{C_D}$ | **Relative Error** $[\%]$ | $\mathbf{C_L}$ | **Relative Error** $[\%]$ |
|---|---|---|---|---|
| Experimental | .333 | 0.0 | 2.0468 | 0.0 |
| t=0.0 | 0.3057 | -8.194 | 1.954 | -4.552 |
| t=0.5 | 0.1567 | N/A | 1.394 | N/A |
| t=1.0 | 0.3045 | -8.562 | 1.951 | -4.68 |

**Comment on mesh quality:**

The quality results of the previous section have depicted multiple highly distorted elements between bodies, for t=0.5 and t=1.0, and it was not clear if those would have affected the fluid flow solver. However, we can state that the impact if there was on the flow calculation is minimal relative to the variables of interest ($C_p$, $C_L$ and $C_D$). As well, the solver before the first iteration did a quality verification and according to it all meshes have less than 3% of low aspect ratio elements. For CFX solver, this represent a good mesh quality, thus in order to integrate better the proposed approach to mesh-motion the quality metric should be similar to the one of the desired solver.

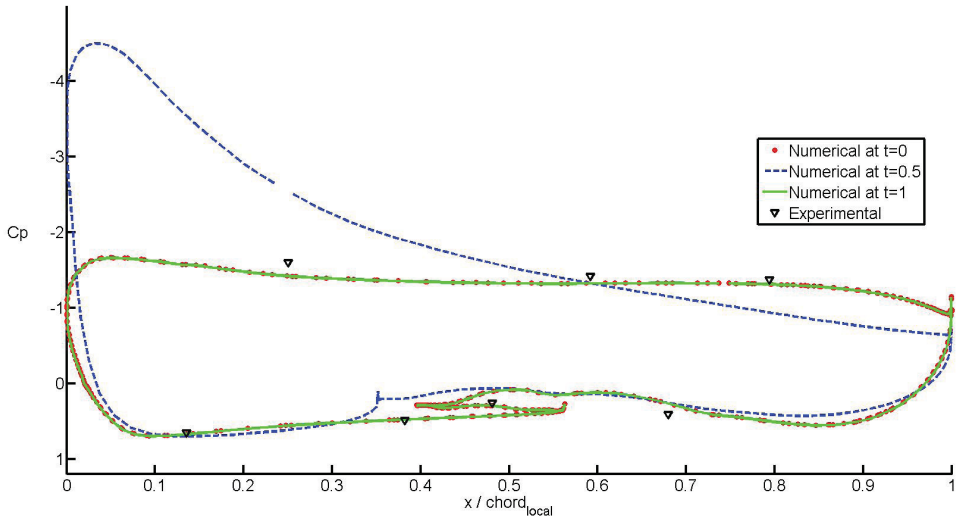



(a) z/span = 0.17

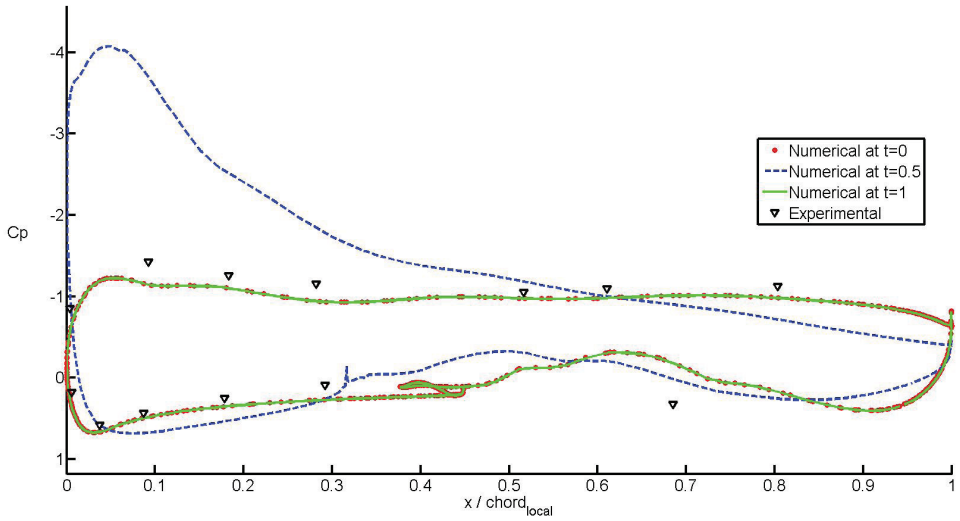



(b) z/span = 0.50

Figure 5.32 Wing-body simulation: Pressure coefficient on slat surfaces for $t = 1.0$.

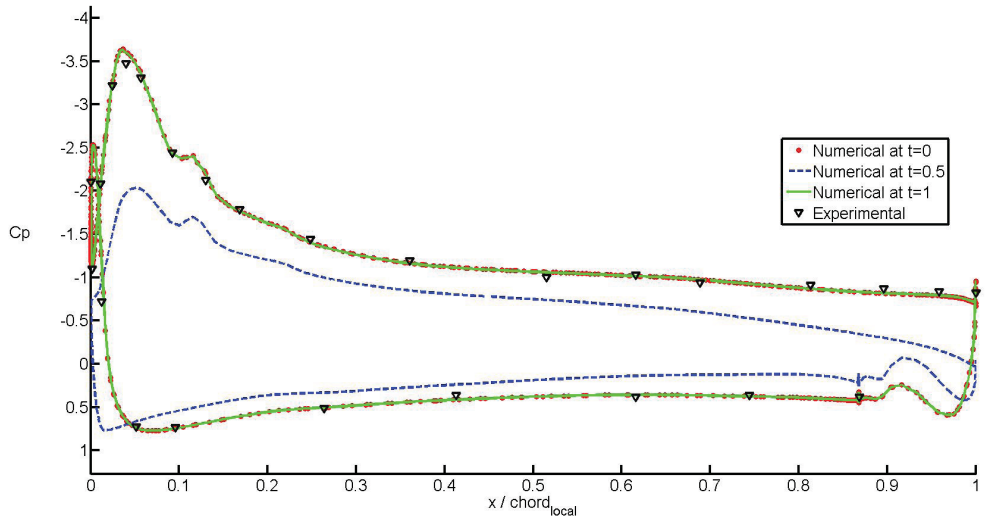



(a) z/span = 0.70

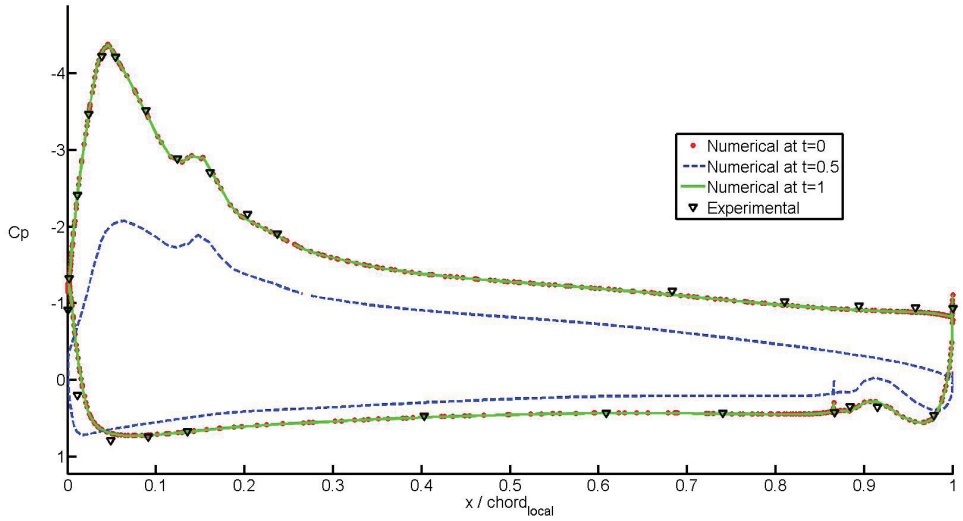



(b) z/span = 0.95

Figure 5.33 Wing-body simulation: Pressure coefficient on slat surfaces for $t = 1.0$ (cont'd).

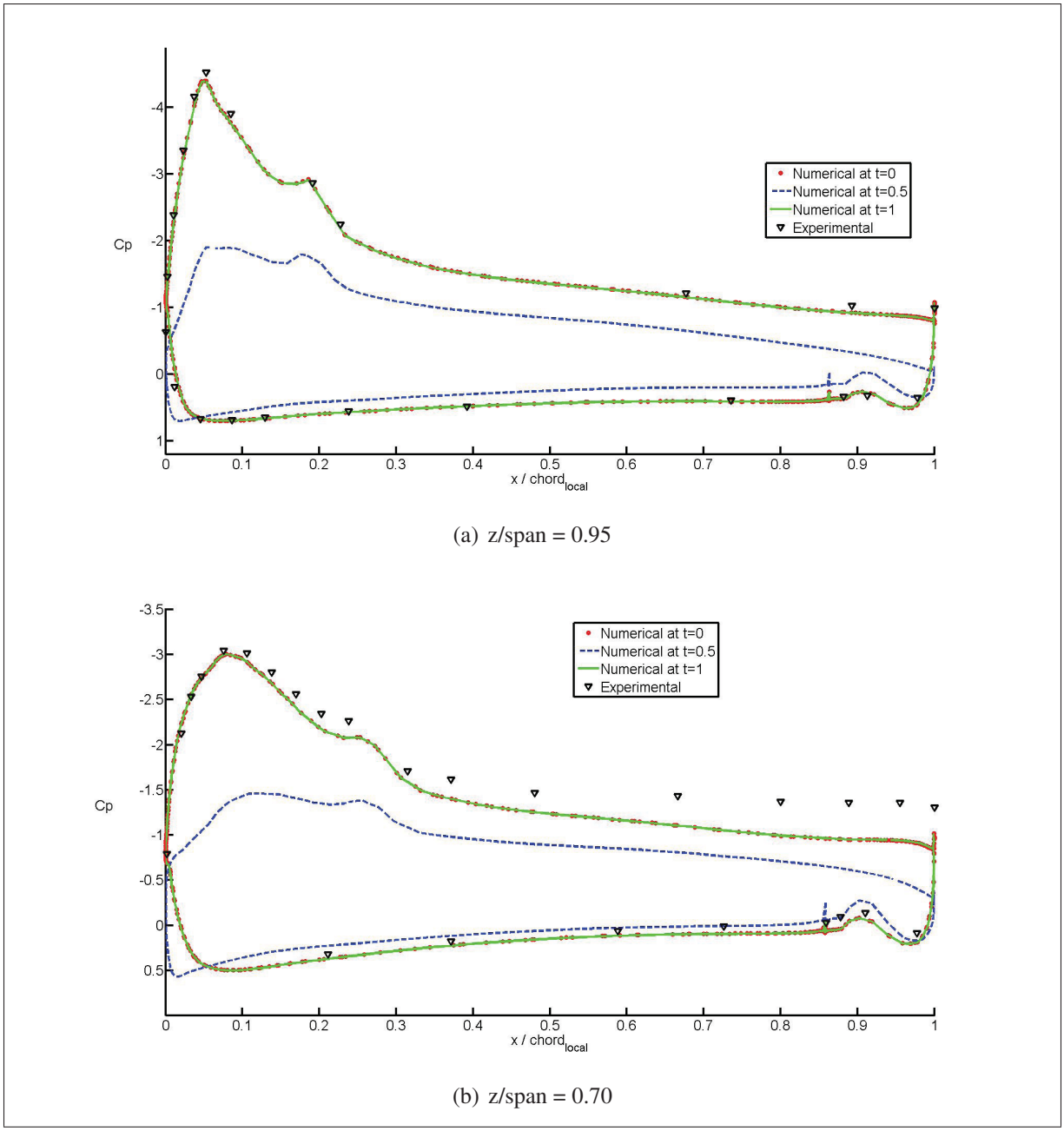



(a) z/span = 0.17




(b) z/span = 0.50

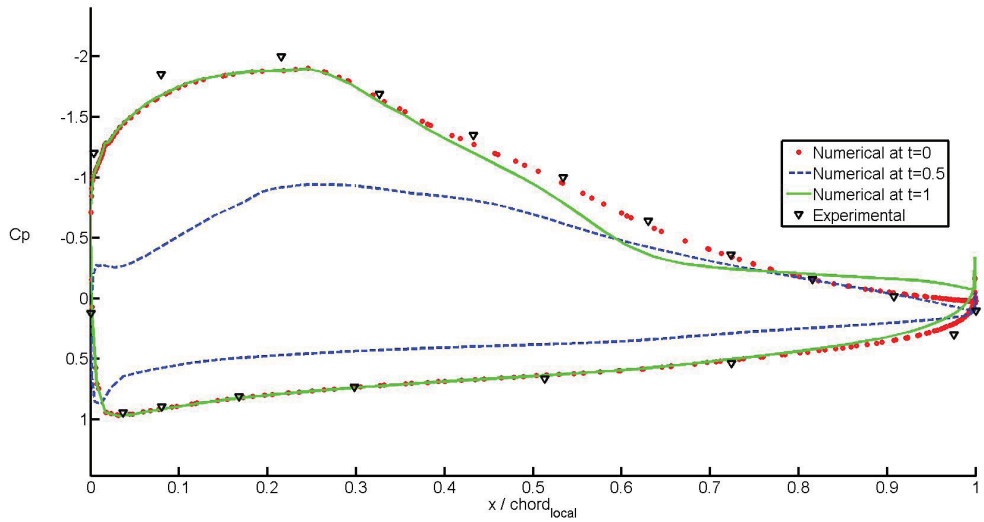
Figure 5.34 Wing-body simulation: Pressure coefficient on wing surfaces for $t = 1.0$.



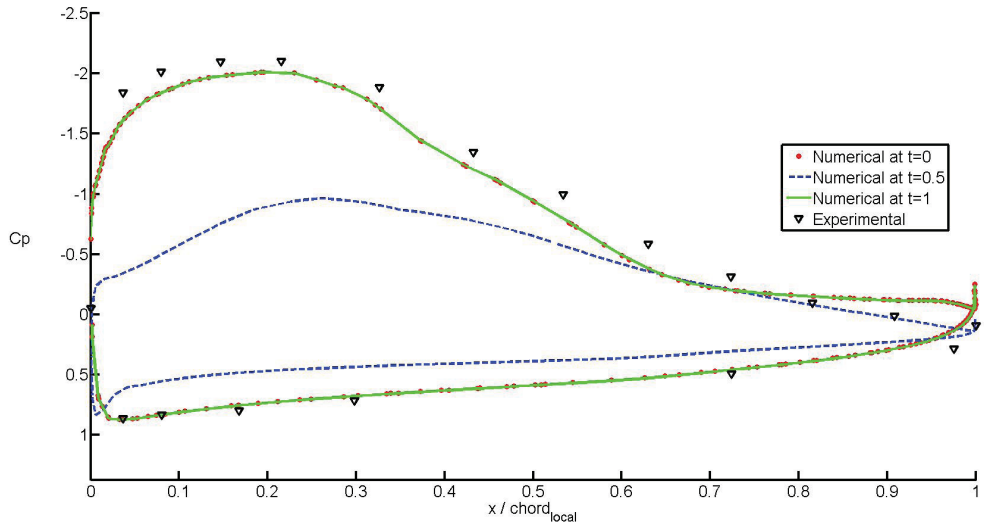

(a) z/span = 0.95

(b) z/span = 0.70

Figure 5.35 Wing-body simulation: Pressure coefficient on wing surfaces for $t = 1.0$ (cont'd).

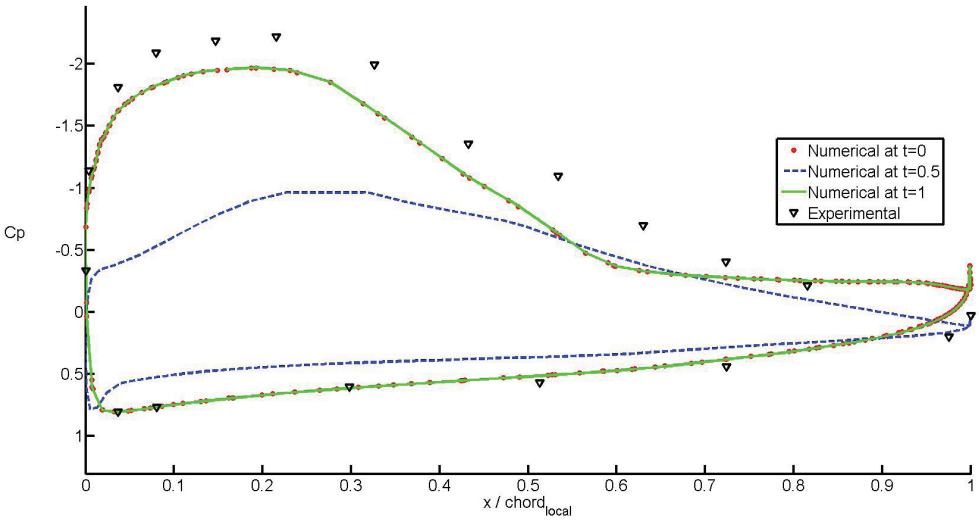



(a) z/span = 0.17

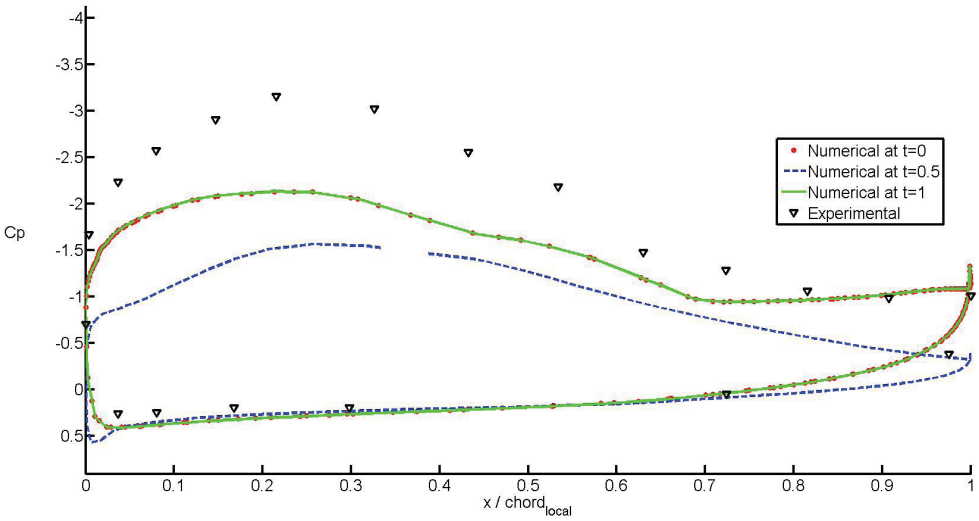



(b) z/span = 0.50

Figure 5.36 Wing-body simulation: Pressure coefficient on flap surfaces for $t = 1.0$.




(a) z/span = 0.70




(b) z/span = 0.95

Figure 5.37 Wing-body simulation: Pressure coefficient on flap surfaces for $t = 1.0$ (cont'd).

# CONCLUSION

For the numerical analysis of moving boundaries problems Mesh-Mover Algorithms (MMAs) are needed. MMAs allow fluid meshes to be adapted according to moving boundaries and keep them valid for fluid flow solving. The first requirement for a deformed mesh to be valid for the solvers is the positivity of each element volume. The second requirement is that the original mesh quality be preserved in order to obtain accurate results from the fluid flow resolution.

The first objective of this work was to design a state of the art Finite Element Method (FEM) module which can be used by any multi-physics program to solve challenging moving boundaries problems. The paradigm used was the well-known object-oriented method which reproduces in computer language a model of the reality. The design is centred on FEM objects: analysis, mesh, boundary condition, element, material, face and node. Each object is defined by its type such as fluid, structure or moving-mesh. Also, the attributes and functions of each object are presented. Then, the graphical and text-based interfaces structure and implementations are described.

The second objective was the implementation of improved MMAs for hybrid meshes in aerodynamics analysis. Thus, we have contributed in this subject with:

- The addition of a quality criteria to the Pseudo-Material (PSM) method of Stein *et al.* (2004);
- The generalization of the Multi-Submesh Approach (MSA) of Lefrançois (2008) for all elements and an implementation of a fast search algorithm;
- The combination of the MSA to the IDW and PSM;
- The consolidation of quality metrics designed for mesh-movements;
- The improvement of deformed mesh quality by combining MMAs to the powerful mixed mesh smoothing algorithms called the Geometric Element Transformation Method (GETMe) of Vartziotis and Papadrakakis (2013);

- The proposition of the Novel GETMe Untangler (NGU) which reverts negative volume elements.

To validate and quantify each proposed improvement, two simple test cases have been studied. After the study of each MMA parameter impact, the proposed improvements were studied. The quality parameters added to the PSM and IDW approaches are found able to increase the mesh quality, but with a lost in robustness thus those tools are not recommended for mesh-motion. Then, the generalization of the MSA is validated for mixed elements meshes, it was found to be less smooth than the IDW and that the current implementation should be improve to reduce its memory consumption. Afterwards, in a multi-body simulation the use of smoothing algorithms and of the NGU (since the mesh had to be repaired) was investigated. It was proven that both methods (NGU and GETMe) help in keeping validity of the mesh and in improving the mesh quality through mesh-motion. Thus, it is strongly recommended to use the NGU and GETMe algorithms together for complex problems after each mesh-motion.

In the last chapter, realistic displacements were imposed from wings to their surrounding fluid mesh (bending/twisting of a NACA0012 wing and high-lift components motions). These simulations have shown the following conclusions:

- PSM based methods with current implementations are not usable for large or complex motions;
- The GETMe and NGU algorithms combination is shown to be robust and performs sufficiently well to preserve mesh quality;
- The best methodology to solve mesh-motion is the IDW approach with the GETMe and NGU algorithms;
- The developed method is shown able to deform the mesh under the motion of high-lift components of rectangular and swept wings through flight stages;
- The distorted mesh flow field for a wing-body has been obtained and the results were similar to an undistorted mesh.

**Future works :**

From the proposed method future improvements and field of study are suggested:

- The proposed method should be integrated with a fluid flow solver to appreciate the work done in this research and improve the solver capability;
- Local re-meshing capabilities should be added to the MMA module to be able to handle more extreme moving boundaries problems and increase the mesh quality even further;
- Design a similar robust approach to refine meshes in order to improve a flow solution;
- An overset approach for mesh-motion treatment should be implemented and compared to the current approach. Then, the overset approach could be improved to be able to simulate multi-body close to contact problems by allowing secondary overset meshes to be deformed with the IDW, GETMe and NGU algorithms;
- Investigations should be made to the MSA framework, such as improving the accuracy of the search algorithm, increasing the degree of interpolation functions to quadratic or cubic and using the method to initialize a fluid solution from a coarse mesh;
- The PSM implementation performance should be increase to allow this method to be used for more difficult moving boundaries problem. To start, we would recommend to implement a GMRES solver, add the capability of domain decomposition and increase the capacity of boundary layers preservation with the use of quaternion to define moving boundaries motion Samareh (2002).

**General remark:**

The numerician who wishes to use a mesh-mover algorithm should use one related to his needs. If only few positions are needed for the numerical analysis, it is better to generate multiple meshes than use an MMA. Then, if the motion involves contact between bodies an overset approach is more suitable for such problems. However, in most moving boundaries simulations, robust MMAs have their place and should be used to save valuable time.

## BIBLIOGRAPHY


Bank, Randolph E and R Kent Smith. 1997. "Mesh smoothing using a posteriori error estimates". *SIAM Journal on Numerical Analysis*, vol. 34, n° 3, p. 979-997.

Beckert, Armin and Holger Wendland. 2001. "Multivariate interpolation for fluid-structure-interaction problems using radial basis functions". *Aerospace Science and Technology*, vol. 5, n° 2, p. 125-134.

Belytschko, Ted, Yury Krongauz, Daniel Organ, Mark Fleming and Petr Krysl. 1996. "Meshless methods: an overview and recent developments". *Computer methods in applied mechanics and engineering*, vol. 139, n° 1, p. 3-47.

Benson, David J. 1989. "An efficient, accurate, simple ALE method for nonlinear finite element programs". *Computer methods in applied mechanics and engineering*, vol. 72, n° 3, p. 305-350.

Bentley, Jon Louis. 1975. "Multidimensional binary search trees used for associative searching". *Communications of the ACM*, vol. 18, n° 9, p. 509-517.

Bonet, Javier and Jaime Peraire. 1991. "An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems". *International Journal for Numerical Methods in Engineering*, vol. 31, n° 1, p. 1-17.

Booch, Grady. 1986. "Object-oriented development". *IEEE Transactions on Software Engineering*, vol. SE-12, n° 2, p. 211-221.

Broomhead, David S and David Lowe, 1988. *Radial basis functions, multi-variable functional interpolation and adaptive networks*. Coll. "RSRE Memorandum", 4148. Malvern (GB) : Royal Signals and Radar Establishment, 34 p.

Buhmann, Martin D. 2000. "Radial basis functions". *Acta Numerica 2000*, vol. 9, p. 1-38.

de Aguiar, Edilson and Norimichi Ukita. 2013. "Representing mesh-based character animations". *Computers and Graphics*, vol. 38, p. 8.

de Boer, A., M. S. van der Schoot and H. Bijl. 2007. "Mesh deformation based on radial basis function interpolation". *Computers and Structures*, vol. 85, n° 11–14, p. 784-795.

Degand, Christoph and Charbel Farhat. 2002. "A three-dimensional torsional spring analogy method for unstructured dynamic meshes". *Computers and Structures*, vol. 80, n° 3–4, p. 305-316.

Dhatt, Gouri, Gilbert Touzot and Emmanuel Lefrançois, 2005. *Méthode des éléments finis*. Paris : Lavoisier, 601 p.

Dompierre, Julien, Paul Labbé, Marie-Gabrielle Vallet and Ricardo Camarero. 1999. "How to Subdivide Pyramids, Prisms, and Hexahedra into Tetrahedra". In *8th International Meshing Roundtable*. (Lake Tahoe, California, 10–13 octobre 1999), p. 195-204.

Donea, J, S Giuliani and JP Halleux. 1982. "An arbitrary Lagrangian-Eulerian finite element method for transient dynamic fluid-structure interactions". *Computer methods in applied mechanics and engineering*, vol. 33, n° 1, p. 689-723.

Escobar, JM, E Rodrıguez, R Montenegro, G Montero and JM González-Yuste. 2003. "Simultaneous untangling and smoothing of tetrahedral meshes". *Computer Methods in Applied Mechanics and Engineering*, vol. 192, n° 25, p. 2775-2787.

Farhat, C, B Koobus and H Tran. 1999. "Simulation of vortex shedding dominated flows past rigid and flexible structures". In *Computational Methods for Fluid-Structure Interaction*. (Trondheim, Norway, 1999), p. 1-30.

Farhat, Ch, C Degand, B Koobus and M Lesoinne. 1998. "Torsional springs for two-dimensional dynamic unstructured fluid meshes". *Computer methods in applied mechanics and engineering*, vol. 163, n° 1, p. 231-245.

Farhat, Charbel and Stéphane Lanteri. 1994. "Simulation of compressible viscous flows on a variety of MPPs: computational algorithms for unstructured dynamic meshes and performance results". *Computer Methods in Applied Mechanics and Engineering*, vol. 119, n° 1, p. 35-60.

Farhat, Charbel, Kendall Pierson and C Degand. 2001. "Multidisciplinary simulation of the maneuvering of an aircraft". *Engineering with Computers*, vol. 17, n° 1, p. 16-27.

Franke, Richard. 1982. "Scattered data interpolation: Tests of some methods". *Mathematics of computation*, vol. 38, n° 157, p. 181-200.

Guennebaud, Gaël, Benoît Jacob et al. 2010. "Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms". Online. <http://eigen.tuxfamily.org>. Accessed December 5, 2014.

He, Tao and Dai Zhou. 2012. "Vortex-induced vibrations of a flexible circular cylinder based on the CIBC method". In *The Seventh International Colloquium on Bluff Body Aerodynamics and Applications*. (Shanghai, China, September 2-6 2012), p. 601-610.

Helenbrook, Brian T. 2003. "Mesh deformation using the biharmonic operator". *International journal for numerical methods in engineering*, vol. 56, n° 7, p. 1007-1021.

Jasak, Hrvoje and Henrik Rusche. 2009. "Dynamic mesh handling in OpenFOAM". In *Proceeding of the 47th Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*. (Orlando, FL, January 5-8 2009), p. 1-10. AIAA.

Johnson, Andrew A and Tayfun E Tezduyar. 1994. "Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces". *Computer methods in applied mechanics and engineering*, vol. 119, n° 1, p. 73-94.

Jones, RE. 1974. "A self-organizing mesh generation program". *Journal of Pressure Vessel Technology*, vol. 96, n° 3, p. 193-199.

Kao, Kai-Hsiung, Meng-Sing Liou and Chuen-Yen Chow. 1993. "Grid adaption using Chimera composite overlapping meshes". In *AIAA 11th Computational Fluid Dynamics Conference*. (Orlando, Florida, July 6-9 1993), p. 1-34.

Knupp, Patrick. 2012. "Introducing the target-matrix paradigm for mesh optimization via node-movement". *Engineering with Computers*, vol. 28, n° 4, p. 419-429.

Knupp, Patrick M. 2003. "Algebraic mesh quality metrics for unstructured initial meshes". *Finite Elements in Analysis and Design*, vol. 39, n° 3, p. 217-241.

Lefrançois, E. 2008. "A simple mesh deformation technique for fluid–structure interaction based on a submesh approach". *International Journal for Numerical Methods in Engineering*, vol. 75, n° 9, p. 1085-1101.

Löhner, Rainald. 1988. "An adaptive finite element solver for transient problems with moving bodies". *Computers and Structures*, vol. 30, n° 1, p. 303-317.

Löhner, Rainald and Chi Yang. 1996. "Improved ALE mesh velocities for moving bodies". *Communications in numerical methods in engineering*, vol. 12, n° 10, p. 599-608.

Liu, Yu, Zheng Guo and Jun Liu. 2012. "RBFs-MSA Hybrid Method for Mesh Deformation". *Chinese Journal of Aeronautics*, vol. 25, n° 4, p. 500-507.

Luke, Edward, Eric Collins and Eric Blades. 2012. "A fast mesh deformation method using explicit interpolation". *Journal of Computational Physics*, vol. 231, n° 2, p. 586-601.

Masud, Arif and Thomas JR Hughes. 1997. "A space-time Galerkin/least-squares finite element formulation of the Navier-Stokes equations for moving domain problems". *Computer Methods in Applied Mechanics and Engineering*, vol. 146, n° 1, p. 91-126.

Miwa, Masahiko, Koji Tani, Takashi Yamada and Shinji Wakao. 2011. "A study of mesh deformation methods for magnetic field analysis". *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 6, n° 5, p. 497-502.

Rendall, TCS and CB Allen. 2008. "Unified fluid–structure interpolation and mesh motion using radial basis functions". *International Journal for Numerical Methods in Engineering*, vol. 74, n° 10, p. 1519-1559.

Rendall, TCS and CB Allen. 2010. "Parallel efficient mesh motion using radial basis functions with application to multi-bladed rotors". *International journal for numerical methods in engineering*, vol. 81, n° 1, p. 89-105.

Reznick, Steve, 1988. *Transonic Navier-Stokes computations of strake-generated vortex interactions for a fighter-like configuration*. Coll. "NASA Technical Memorandum", 100009. Moffett Field, CA, United States : NASA Ames Research Center, 124 p.

Robinson, Brian A, Henry TY Yang and John T Batina. 1991. "Aeroelastic analysis of wings using the Euler equations with a deforming mesh". *Journal of Aircraft*, vol. 28, n° 11, p. 781-788.

Rumsey, Christopher. 2014. "High Lift Prediction Workshop". Online. <http://hiliftpw.larc.nasa.gov/>. Accessed April 9, 2015.

Saad, Youcef and Martin H Schultz. 1986. "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems". *SIAM Journal on scientific and statistical computing*, vol. 7, n° 3, p. 856-869.

Samareh, Jamshid A. 2002. "Application of quaternions for mesh deformation". *NASA TM*, vol. 211646, 14 p.

Schreurs, PJG, FE Veldpaus and WAM Brekelmans. 1986. "Simulation of forming processes, using the arbitrary Eulerian-Lagrangian formulation". *Computer methods in applied mechanics and engineering*, vol. 58, n° 1, p. 19-36.

Shepard, Donald. 1968. "A two-dimensional interpolation function for irregularly-spaced data". In *Proceedings of the 1968 23rd ACM national conference*. p. 517-524. ACM.

Sleijpen, Gerard LG and Diederik R Fokkema. 1993. "BiCGstab (l) for linear equations involving unsymmetric matrices with complex spectrum". *Electronic Transactions on Numerical Analysis*, vol. 1, n° 11, p. 2000.

Soulaïmani, A. 1987. "Contribution à la résolution numérique des problèmes hydrodynamiques à surface libre". Ph.D Thesis, Québec, Faculté des sciences et de génie, Universite Laval, 241 p.

Soulaïmani, Azzeddine, Michel Fortin, G Dhatt and Y Ouellet. 1991. "Finite element simulation of two-and three-dimensional free surface flows". *Computer methods in applied mechanics and Engineering*, vol. 86, n° 3, p. 265-296.

Standardization, International Organization for, 1999. *Information technology: vocabulary. Programming languages*. ISO/IEC, 2382-15. Geneva: International Organization for Standardization.

Steger, Joseph L and John A Benek. 1987. "On the use of composite grid schemes in computational aerodynamics". *Computer Methods in Applied Mechanics and Engineering*, vol. 64, n° 1, p. 301-320.

Steger, Joseph L and Pieter G Buning. 1985. "Developments in the simulation of compressible inviscid and viscous flow on supercomputers". In *Progress and Supercomputing in Computational Fluid Dynamics*. p. 67-91. Springer.

Steger, Joseph L, F Carroll Dougherty and John A Benek. 1983. "A Chimera grid scheme". In *Advances in grid generation; Proceedings of the Applied Mechanics, Bioengineering,*

*and Fluids Engineering Conference*. (Houston, TX, January 1 1983). American Society of Mechanical Engineers.

Stein, K, T Tezduyar and R Benney. 2003. "Mesh moving techniques for fluid-structure interactions with large displacements". *Journal of Applied Mechanics*, vol. 70, n° 1, p. 58-63.

Stein, Keith, Tayfun E Tezduyar and Richard Benney. 2004. "Automatic mesh update with the solid-extension mesh moving technique". *Computer Methods in Applied Mechanics and Engineering*, vol. 193, n° 21, p. 2019-2032.

Tezduyar, T. E., M. Behr and J. Liou. 1992. "New strategy for finite element computations involving moving boundaries and interfaces. The deforming-spatial-domain/space-time procedure. I. The concept and the preliminary numerical tests". *Computer Methods in Applied Mechanics and Engineering*, vol. 94, n° 3, p. 339-351.

Vartziotis, Dimitris and Manolis Papadrakakis. 2013. "Improved GETMe by adaptive mesh smoothing". *Computer Assisted Methods in Engineering and Science*, vol. 20, p. 55-71.

Vartziotis, Dimitris and Joachim Wipper. 2011. "A dual element based geometric element transformation method for all-hexahedral mesh smoothing". *Computer Methods in Applied Mechanics and Engineering*, vol. 200, n° 9, p. 1186-1203.

Vartziotis, Dimitris and Joachim Wipper. 2012. "Fast smoothing of mixed volume meshes based on the effective geometric element transformation method". *Computer Methods in Applied Mechanics and Engineering*, vol. 201, n° 204, p. 65-81.

Vartziotis, Dimitris, Joachim Wipper and Bernd Schwald. 2009. "The geometric element transformation method for tetrahedral mesh smoothing". *Computer Methods in Applied Mechanics and Engineering*, vol. 199, n° 1, p. 169-182.

Wilson, Thomas James. 2011. "Simultaneous Untangling and Smoothing of Hexahedral Meshes". Master Thesis, Barcelona, Universitat Politècnica de Catalunya, 62 p.

Winslow, Alan M, 1963. *"Equipotential"zoning of two-dimensional meshes*. No. UCRL-7312. California Univ., Livermore (USA): Lawrence Livermore National Lab.

Winslow, Alan M, 1981. *Adaptive-mesh zoning by the equipotential method*. No. UCID-19062. California Univ., Livermore (USA): Lawrence Livermore National Lab. 14 p.

Witteveen, Jeroen AS and Hester Bijl. 2009. "Explicit mesh deformation using inverse distance weighting interpolation". In *19th AIAA Computational Fluid Dynamics Conference*. (San Antonio, Texas, June 22-25 2009), p. 2009-2019. AIAA.

Witteveen, Jeroen AS and Hester Bijl. 2012. "Transonic velocity fluctuations simulated using extremum diminishing uncertainty quantification based on inverse distance weighting". *Theoretical and Computational Fluid Dynamics*, vol. 26, n° 5, p. 459-479.

Zeng, Dehong and C. Ross Ethier. 2005. "A semi-torsional spring analogy model for updating unstructured meshes in 3D moving domains". *Finite Elements in Analysis and Design*, vol. 41, n° 11–12, p. 1118-1139.