POLITECNICO DI MILANO

Facoltà di Ingegneria Industriale

Corso di Laurea in
Ingegneria Aeronautica

Free Form Deformation Techniques for 3D Shape Optimization Problems

Relatore: Prof. Alfio QUARTERONI

Correlatore: Dr. Ing. Gianluigi ROZZA

Tesi di Laurea di:

Anwar KOSHAKJI Matr. 720668

Anno Accademico 2009 - 2010

Wherever your heart is,
there you will find your treasure.

— *The Alchemist*, P. Coelho

..to my family..

Acknowledgments

Questa pagina non dovrebbe essere una semplice pagina ma un capitolo vero e proprio, tante sono le persone che vorrei ringraziare.

A cominciare dal mio relatore, Prof. Dr. A. Quarteroni, che per primo mi ha offerto l'opportunità di lavorare a questo progetto e di vivere un'esperienza così stimolante come quella di Losanna, e dal mio correlatore, Dr. G. Rozza, per la sua pazienza, attenzione e rigore nel leggere questo lavoro e per il suo costante sforzo nell'applicare le sue conoscenze per correggerlo, migliorarlo e perfezionarlo. Voglio ringraziare inoltre il Prof. L. Formaggia che ha sostenuto la mia partecipazione al progetto di scambio. Un grande grazie va anche a tutto il gruppo del CMCS: Matteo Lombardi, Paolo Crosetto, Andrea Manzoni, Laura Iapicino, Matteo Lesinigo, Cristiano Malossi, Fabrizio Gelsomino, per il loro supporto e per aver contribuito a rendere unica e piacevole l'esperienza di Losanna. Un grazie va anche a quei professori, che spero siano sempre di più, che in questi anni di formazione hanno saputo andare oltre la semplice lezione, trasmettendomi non solo la pura pragmaticità della loro materia di insegnamento, ma anche lo stimolo della ricerca e il piacere del confronto aperto e rispettoso. Qualcosa a mio parere molto più importante delle semplici nozioni.

Poi voglio ringraziare chi ho conosciuto in questi anni, con cui ho condiviso magari anche solo un semestre, un solo un esame, chi invece ho perso lungo il tragitto e chi alla fine è rimasto. Le prove sono state molte e dure, ma proprio a causa di queste difficoltà sono nati dei legami che mi auguro continueranno a durare e a rafforzarsi anche una volta finito questo percorso. A Roby, Andre, il Leo, Alessio, Marco, Enri, Gabro, Ale Alo, Dvd, Umbe, Alessia, Stefano, i "Gionnis" Alberto e Davide, Ema e Ivan, Anto e potrei continuare ancora. Grazie per tutte le chiaccherate, i consigli, le risate insieme, le birre, le pizze e le sclerate di questi anni, le porterò sempre con me.

Un grazie speciale va ai miei amici di sempre, alla mia famiglia allargata, Manu, Miky, Albi, Ale, il Raffo e Miky di guanza. Per tutto il bene che vi voglio

e per aver condiviso con me alcuni dei momenti più belli e unici che io ricordi. Per esserci sempre e comunque e perché, pur per strade diverse, siamo parti di un unico percorso. Per citare gli Articolo 31 in una canzone che canto sempre a squarciagola "..realizzi che lo stile è quando siamo uniti..". Ogni volta che sento quella canzone penso a voi. In particolare un grazie speciale va a te sorellina, semplicemente per averti conosciuta e per il dono della tua vera, incrollabile, insostituibile amicizia.

Grazie a te Sam, ormai sempre più uomo, pronto a superarmi non solo in altezza, per tutti i consigli, le discussioni e perché no anche le litigate (in fondo siamo due arieti), perché sono convinto contribuiscano a farci crescere entrambi e perché so che, comunque vada, oggi come domani, possiamo sempre contare l'uno sull'altro. Sono fortunato ad averti come fratello. A big big thanx anche a mio zio Henry per il suo aiuto, per tutto il suo affetto, ma soprattutto per essere lo zio migliore che esista e il primo amico che io abbia mai avuto. Le persone come te sono rare purtroppo.

Ma coloro che più di ogni altro devo ringraziare sono quelle persone meravigliose che sono i miei genitori, Joseph e Nicoletta. A loro devo più di chiunque altro. Devo semplicemente tutto. Probabilmente nessuna parola o frase sarebbe all'altezza per dire loro quanto veramente vorrei esprimere....grazie di cuore. Per tutto l'Amore, il sostegno, il supporto che mi avete sempre dato da 26 anni a questa parte. Senza di voi non solo questo lavoro non sarebbe stato possibile, ma io stesso non sarei l'uomo che sono adesso, nel bene e nel male. Questo lo devo a voi. Grazie.

Abstract

The purpose of this work is to analyse and study an efficient parametrization technique for a 3D shape optimization problem. After a brief review of the techniques and approaches already available in literature, we choose to use the Free Form Deformation parametrization, a recent technique which proved to be efficient and at the same time versatile, allowing to manage complex shapes even with few parameters. We tested and studied the technique by developing a link among different specialized softwares, in order to establish a path, from the geometry definition, to the method implementation, and finally to the simulation and to the optimization of the problem. In particular, we have studied a bulb and a rudder of a race sailing boat as model problems.

Sommario

Lo scopo di questo lavoro è quello di analizzare e studiare una tecnica di parametrizzazione applicandola ad un problema 3D di ottimizzazione di forma. Dopo una rassegna delle principali tecniche e approcci già presenti in letteratura, si è scelto di utilizzare la parametrizzazione Free Form Deformation (FFD), una tecnica recente che si è dimostrata essere alquanto potente e allo stesso tempo versatile, consentendo di gestire forme complesse anche con un numero esiguo di parametri. È stato studiata e sviluppata un'interfaccia tra diversi software in modo da stabilire un percorso standard utilizzabile per costruire qualsiasi problema generico, dalla definizione della geometria, all'implementazione del metodo, fino alla simulazione e alla soluzione del problema di ottimizzazione. Nello specifico, sono stati utilizzati come modelli un bulbo e un timone di una barca a vela da competizione.

Outline

Sempre più frequentemente, in molti problemi di interesse ingegneristico è richiesto di risolvere un sistema di equazioni alle derivate parziali. Quando è necessario richiamarle molte volte nello stesso ciclo, ad esempio in un problema di ottimizzazione di forma, a maggior ragione è necessario avere a disposizione un modo per ridurre i costi computazionali che facilmente possono diventare proibitivi. Due possono essere le direzioni verso cui agire: la riduzione del modello matematico e la parametrizzazione efficiente della geometria. Questo lavoro è rivolto a studiare proprio questo secondo aspetto, e a questo scopo sono state analizzate diverse tecniche di parametrizzazione da utilizzare in un contesto di *shape optimization*. In particolare, sono stati approfonditi aspetti e potenzialità del metodo di parametrizzazione di forma *Free Form Deformation* (*FFD*), il quale è risultato essere molto versatile e al tempo stesso potente ed efficace, utilizzabile anche per le forme alquanto complesse. Si è partiti da una FFD bidimensionale e la si è estesa al caso tridimensionale, applicandola successivamente in un processo di ottimizzazione di forma a due componenti idrodinamiche di un'imbarcazione da competizione: un bulbo e un timone. Le equazioni utilizzate sono quelle di Navier-Stokes e per la loro risoluzione è stata usata una discretizzazione agli *Elementi Finiti*. Per l'ottimizzazione è stato usato il metodo di ottimizzazione *Sequential Quadratic Programming* (SQP). Al fine di ottenere un ciclo automatico di progetto, stabilendo così un percorso standard da poter utilizzare per i problemi più diversi, è stata studiata una sinergia e una portabilità tra diversi softwares. A cominciare dalla costruzione del modello CAD (SOLIDWORKS), all'implementazione di FFD e dell'algoritmo di ottimizzazione (MATLAB) e alla risoluzione delle equazioni di Navier-Stokes (COMSOL Multiphysics).

Il capitolo 1 illustra le principali tecniche di parametrizzazione, in particolare soffermandosi su quelle più usate e conosciute: le curve di Bezier, B-Splines e NURBS. Il capitolo 2 è interamente dedicato alla descrizione della FFD, le

sue proprietà, le sue applicazioni e la sua estensione al caso tridimensionale. Nel capitolo 3 sono illustrate le tecniche e gli strumenti utilizzati per creare e gestire i modelli geometrici. Il capitolo 4 è dedicato alla formulazione del modello matematico, ossia delle equazioni di Navier-Stokes e della loro formulazione debole, le approssimazioni adottate, la discretizzazione agli elementi finiti, le proprietà delle mesh, le proprietà geometriche, la descrizione dei solutori utilizzati e l'algoritmo di ottimizzazione. Infine nel capitolo 5 si riportano e illustrano i risultati delle ottimizzazioni di forma effettuate su bulbo e timone, mentre nel capitolo 6 vengono fatte alcune considerazioni conlusive e proposte di possibili sviluppi futuri.

Contents

List of Figures

Remark

This master thesis has been developed within the EU Socrates Programme with an exchange period of six months at the École Polytechnique Fédérale de Lausanne, Mathematics Institute of Computational Sciences and Engineering. Written with LaTeX.

Politecnico di Milano,
École Polytechnique Fédérale de Lausanne,

Milan and Lausanne, 3 March 2011.

Chapter 1

Introduction

An important problem in computational science and engineering is to solve partial differential equations in domains involving arbitrary shapes, more particularly in shape optimization [14, 45]. In an optimization context, one needs to solve the same equations several times, however in general this procedure is time consuming and inefficient. This calls for an improvement of the approach to the problem, which depends on the selected discretization model (e. g. *Finite Elements method* [59]) and on the description and parametrization of the domain geometry and its possible perturbation.

Let us consider a domain and apply a perturbation to it. Some questions may arise: what would the range of reachable shapes be? Which level of complexity could be achieved? How much would that cost in computational terms? Is there an easy and intuitive method to use? Developing such a strategy can be useful in many applications in shape optimization, due to the high flexibility and complexity required for this kind of problem and for the high number of iterations that can be needed to reach the convergence. Examples of practical applications can be found, for instance, in industrial applications and in particular in Aeronautics, for example in the shape optimization of an airfoil, or an entire wing (see figure 1.1), when attempting to obtain a drag reduction or an efficiency improvement, obeying to some specific optimization laws [14, 16, 30, 31, 40, 43, 45].

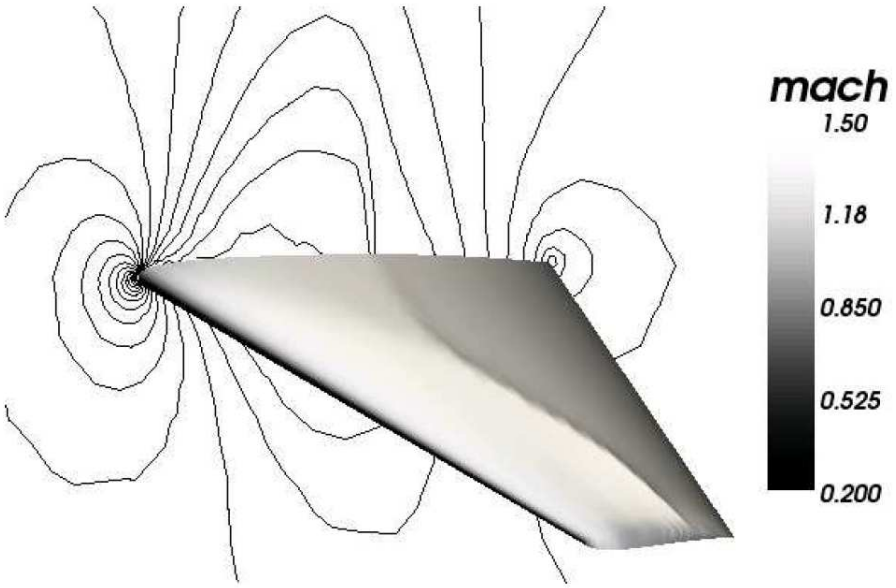

Figure 1.1: Mach number field on a wing shape optimized [30].

Other fields of interest can be found in medical applications, such as the study of aorto-coronaric bypass configuration [60, 66] (figure 1.2), and in naval engineering [47, 49, 54] (see figure 1.3).

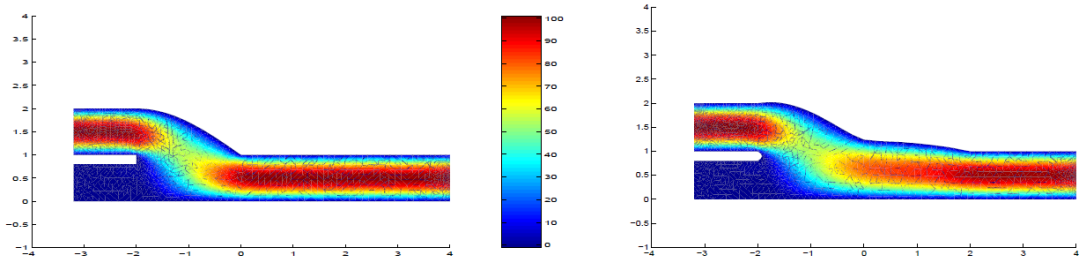

Figure 1.2: A bypass configuration before (left) and after the shape optimization [66].

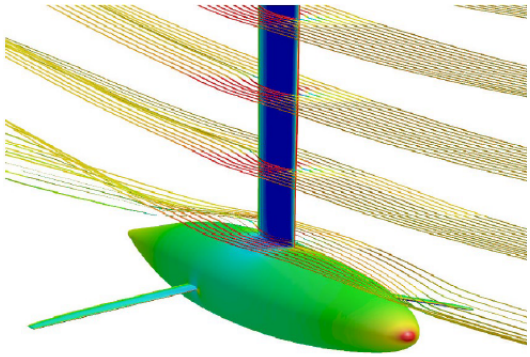

Figure 1.3: Streamlines and pressure field of around a bulb [54].

The aim of this work is to find a simple and powerful method for the management of a large variety of complex and smooth tridimensional deformations, and test it in order to deform some sample geometries. The method should be flexible to describe a wide range of shapes with minimum geometrical constraints. Applications will be made to deform the geometry of a rudder and that of a bulb, two appendages of a sailing yacht, and insert it in a shape optimization process.

1.1 Different approaches

The way to describe shape perturbations can be divided in two categories: variational or parametric. In this work we considered only the parametric shape variation: the perturbed domain is a function of a finite number of real parameters. There are several methodologies that will be described in the following sections, relevant aspects are efficiency, effectiveness and easiness of implementation and generalization.

Let us mention briefly some of the most typical approaches for parametric domains [72], and the motivations that have led us to choose the most appropriate one:

Basis shapes approach: By using this approach there is a well-chosen set of shape perturbations, which is used to model the geometry by a linear combination. An example of perturbation functions that parametrize an airfoil geometry can be seen in figure 1.4:

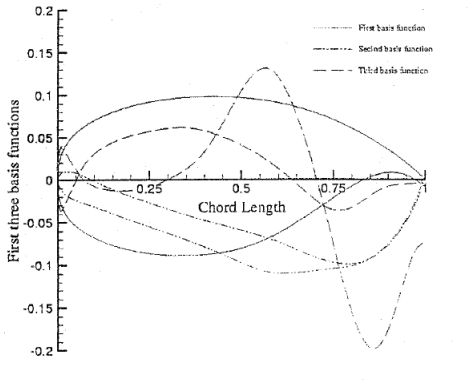

Figure 1.4: The first three orthogonal basis functions for parametrization of an airfoil [76].

The shape changes can be expressed as

$$R = r + \sum_i v_i U_i, \tag{1.1}$$

where R is the design shape, r is the baseline shape, v_i is the design variable vector and U_i is the design perturbation based on several proposed shapes.

This kind of parametrization is usually low-dimensional, so the method will only explore a limited set of variety of shapes [45]. Besides it is difficult to generate a set of consistent basis vectors for multiple disciplines, that is for multidisciplinary shape parametrization [72]. For instance in case of an airplane we must not only treat the external geometry but also the internal structural elements, such as spars, ribs and fuel tanks. As a result, this method can be applied only to problems involving a single discipline aspects with relatively simple geometry changes.

Discrete approach: This is the most intuitive approach, based on using the coordinates of the boundary points as design variables (see figure 1.5).

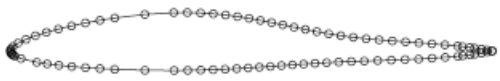

Figure 1.5: Airfoil designed by a set of points [72].

This method is easy to implement, and the available shapes are limited only by the number of the boundary points. However, it is difficult to maintain a smooth geometry, and to do this, the number of the design variables becomes very large, which leads to a high computational cost and a difficult optimization problem to solve [51].

Polynomial and Spline approach: This approach is based on the use of polynomial and spline representations for shape parametrization (see figure 1.6). This can greatly reduce the total number of design variables. The control of the shape is handled by few special points, called *control points*, which by modifying their positions, the value of the polynomial which describes the curve changes with them.

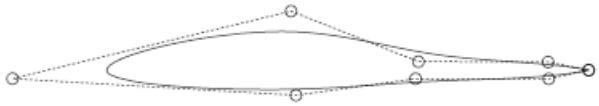

Figure 1.6: Splines curve and its control points [72].

Those methods are very popular in CAD and in design applications in general. According to the different features that will be described in details in the sections 1.2, 1.3 and 1.4, these are called Bezier, B-splines and NURBS (Non-Uniform Rational B-Spline) curves [12, 75]. Those types of curves are well suited for shape optimization, as shown in several works [14, 16, 23, 30, 72].

As it will be mentioned in the next sections, some definitions of such curves are limited (that is not all the geometries are representable with all these curves) but they are connected with each other in such a way that one

definition goes beyond the limit imposed by the other. The relationship among these types of curves and the range of shapes that they can reach is shown in figure 1.7: it is then clear that NURBS curves are the most general, followed by Rational Bezier curves, which are slightly more limited, then by B-spline curves and finally by Bezier curves.

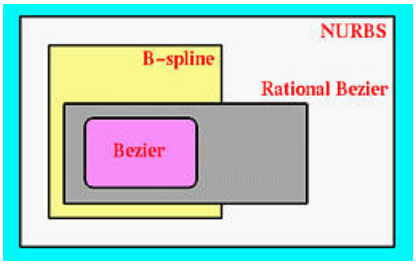

Figure 1.7: Relationship between Bezier, B-splines and NURBS curves [75].

This approach has several drawbacks, see the next sections, basically due to the fact that the more complex the curve, the higher the number of degrees of freedom in the parametrization that is required.

Free Form Deformation approach: This method operates on the whole space that embeds the deformed objects, through the definition of a reference domain and by moving some suitable control points. This is very interesting, especially because the user can manipulate the control points of trivariate Bezier[1] volumes.

From what we have highlighted, we can see that in all these methods there are some pros and cons. The one which seems to be the most promising, because of its power and simplicity, is Free Form Deformation method (or FFD method). This method has been considered in this work and will be described more precisely in chapter 2.

A description of the most important features of some of the spline techniques will first be introduced [21, 55, 56, 62, 64, 75] because they are at the base of FFD method. We will review Bezier, B-spline and NURBS curves.

[1] Also B-splines or NURBS can be used to produce the deformation volume.

1.2 Bezier curves

Bezier curves are widely used in computer graphics to model smooth curves. The points can be visualized graphically and used to instinctively manipulate the curve. The most important Bezier curves are the quadratic and cubic ones, higher degrees are too expensive.

1.2.1 Definition

Given $n+1$ *control points* $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, ... and $\mathbf{P}_n$ in space, with $\mathbf{P}_i = (x_i, y_i, z_i)$ and a local coordinate $u \in [0,1]$, such that the Bezier curve is defined as [75]:

$$\mathbf{C}(u) = \sum_{i=0}^{n} B_{n,i}(u)\,\mathbf{P}_i, \tag{1.2}$$

where the coefficients $B_{n,i}$ are defined as follows:

$$B_{n,i}(u) = \frac{n!}{i!\,(n-i)!} u^i (1-u)^{n-i}. \tag{1.3}$$

Therefore, the point that corresponds to u on the Bezier curve is the weighted average of all control points, where the weights are the coefficients $B_{n,i}(u)$. These coefficients are referred to as the *Bezier basis functions* or *Bernstein polynomials* [61].

1.2.2 Properties

Here are some important properties of a Bezier curve [75]:

- $n+1$ control points define a Bezier curve of n degree.
- $\mathbf{C}(u)$ passes through $\mathbf{P}_0$ and $\mathbf{P}_n$, or better, through the first and the last control point. Besides, Bezier curves are tangent to their first and last legs.
- All basis functions are non-negative.
- Partition of unity property: the sum of the basis functions at a fixed u is 1.

- Convex hull property: the Bezier curve is set completely in the convex hull of the given control points, that is the smallest convex set that contains all points (see figure 1.8).

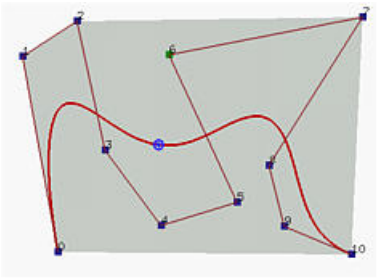

Figure 1.8: The convex hull is the grey zone [75].

This guarantees that the generated curve will be in a clear and computable region and will not go beyond its limit.

- Variation diminishing property: the control polyline[2] is more complex than the shape of generated curve. This means that if the curve is in a plane, no straight line intersects a Bezier curve more times than it intersects the curve control polyline (see figure 1.9).

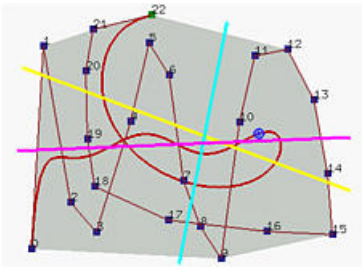

Figure 1.9: Line in the convex hull intersecting a Bezier Curve [75].

- Affine invariance: in geometry, an affine transformation or affine map between two vector spaces, or two affine spaces, consists of a linear transfor-

[2] A polyline is a term that indicates a series of connected line segments and arcs that are treated as a single entity.

mation followed by a translation:

$$x \longmapsto Ax + b, \tag{1.4}$$

A simple practical example is shown in figure 1.10 where this transformation can be seen applied to a fern and its smaller parts, which could be seen as its affine image:

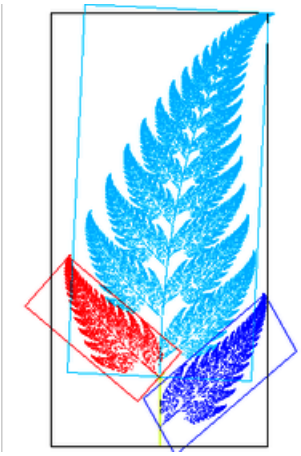

Figure 1.10: An image of a fern with self-similarity parts [10].

For a Bezier curve, the result of an affine transformation applied to it can be constructed from the affine images of its control points. So, the affine transformation can be applied to the control points, easier than applying the transformation to the curve.

- Changing the position of a control point causes the shape of a Bezier curve to change globally, as shown in figure 1.11.

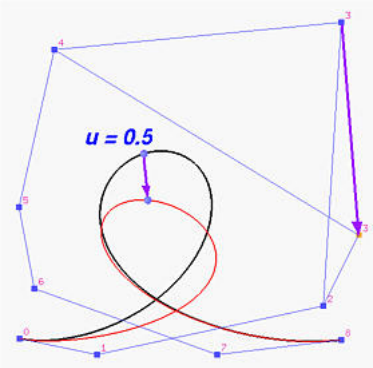

Figure 1.11: Displacement of a control point [75].

1.2.3 De Casteljau's algorithm

A problem that can arise by calculating $\mathbf{C}(u)$ from the equation (1.2) is a numerical instability due to the introduction of numerical errors during the evaluation of the Bernstein polynomials [75]. To overcome this problem the *De Casteljau's algorithm* is introduced.

Let the control points be **00** for $\mathbf{P}_0$, **01** for $\mathbf{P}_1$, ..., $\mathbf{0}\boldsymbol{i}$ for $\mathbf{P}_i$, ..., $\mathbf{0}\boldsymbol{n}$ for $\mathbf{P}_n$. The zeros in these numbers indicate the initial or the $0-th$ iteration. The fundamental concept of De Casteljau's algorithm is to choose a point **C** in line segment **AB** such that **C** divides the line segment **AB** in a ratio of $u : 1-u$, that is the ratio of the distance between **A** and **C** and the distance between **A** and **B** is u, as shown in figure 1.12:

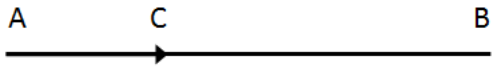

Figure 1.12: Segment **AB** and the point **C**.

The idea of De Casteljau's algorithm goes as follows: suppose we want to find $\mathbf{C}(u)$, where u is in $[0,1]$. Starting with the first polyline, **00-01-02-03**-...-$\mathbf{0}\boldsymbol{n}$, use the above method to find a point $\mathbf{1}\boldsymbol{i}$ on the leg, that is a line segment, from $\mathbf{0}\boldsymbol{i}$ to $\mathbf{0}(\boldsymbol{i}+\mathbf{1})$ which divides the line segment $\mathbf{0}\boldsymbol{i}$ and $\mathbf{0}(\boldsymbol{i}+\mathbf{1})$ in a ratio of $u : 1-u$. In this way, we will obtain n points **10**, **11**, **12**, ..., $\mathbf{1}(\boldsymbol{n}\text{-}\mathbf{1})$. They define a new polyline of $n-1$ legs.

In figure 1.13, where $u = 0.4$, **10** is in the leg of **00** and **01**, **11** is in the leg of **01** and **02**, ..., and **14** is in the leg of **04** and **05**, all the new points are in blue.

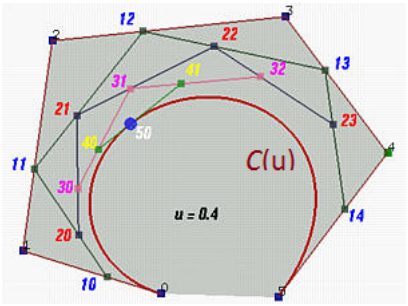

Figure 1.13: Geometrical interpretation of De Casteljau's algorithm [75].

Applying the procedure to this new polyline, we obtain a second polyline of $n-1$ points **20**, **21**, ..., **2(*n*-2)** and *n-2* legs. Starting with this polyline, we can construct a third one of $n-2$ points **30**, **31**, ..., **3(*n*-3)** and *n*-3 legs. Repeating this process n times yields a single point ***n*0** (in figure 1.13 it is the **50** point). De Casteljau proved that this is the point $\mathbf{C}(u)$ on the curve which corresponds to u.

So the geometric interpretation of the De Casteljau's idea is to subdivide the polyline in $n-1$ legs. Then applying once more the above procedure to this new polyline, we shall get a second polyline of $n-2$ legs. Repeating this process n times yields a single point.

From a computational point of view, let $\mathbf{P}_{0,j}$ be $\mathbf{P}_j$ for $j = 0, 1, \ldots, n$. That is, $\mathbf{P}_{0,j}$ is the j-th entry on column 0. The computation of entry j on column i is the following:

$$\mathbf{P}_{i,j} = (1-u)\,\mathbf{P}_{i-1,j} + u\mathbf{P}_{i-1,j+1} \qquad \begin{cases} i = 1, 2, \ldots, n, \\ j = 0, 1, \ldots, n-1. \end{cases} \tag{1.5}$$

Figure 1.14 shows this computational process, until the $\mathbf{P}_{n,0}$ is reached.

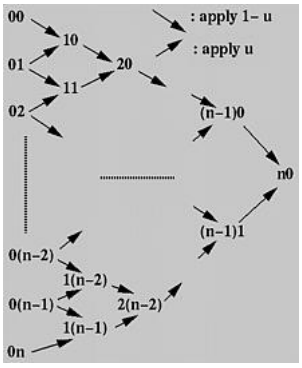

Figure 1.14: Computational passages to the single point on $\mathbf{C}(u)$ [75].

With this approach it is possible to find the Bezier curve $\mathbf{C}(u)$ using a numerical stable way.

1.2.4 Subdividing a Bezier curve

When there is a need to achieve a more complex curve, instead of increasing the polynomial degree, thus increasing the computational cost, it is possible to join more curves of lower degree, obeying continuity and smooth conditions [75].

Given a set of $n+1$ control points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, ... and $\mathbf{P}_n$ and a parameter value $u \in [0,1]$, two sets of $n+1$ control points $\mathbf{Q}_0$, $\mathbf{Q}_1$, $\mathbf{Q}_2$, ..., $\mathbf{Q}_n$ and $\mathbf{R}_0$, $\mathbf{R}_1$, $\mathbf{R}_2$, ..., $\mathbf{R}_n$ can be found such that the Bezier curve defined by $\mathbf{Q}_i$'s (respectively $\mathbf{R}_i$'s) is the piece of the original Bezier curve on $[0,u]$ (respectively $[u,1]$).

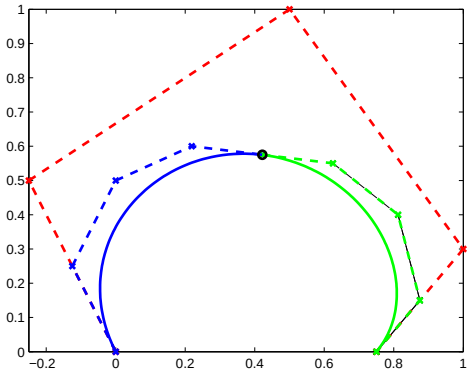

Figure 1.15: Subdivision of Bezier curve at $u = 0.5$.

A curve subdivision has many applications, for instance it can be used to make curve design easier, rendering Bezier curves. If for example there are two parts of the curve, one satisfactory and one not, it is possible to ignore the former and concentrate on the latter in order to improve it.

Subdivision can be applied as many times as one wishes, but the only thing to keep in mind is that the joining point and its two adjacent neighbors must be kept co-linear, to preserve the smoothness of the curve [75].

1.3 B-spline curves

To improve the control of the curve's shape, a new concept is introduced: instead of subdividing the curve directly, the domain of the curve is subdivided. Thus, if the domain of a curve is $[0, 1]$, this closed interval is subdivided by points called *knots* (figure 1.16). Consequently, modifying the subdivision of $[0, 1]$ changes the shape of the curve.

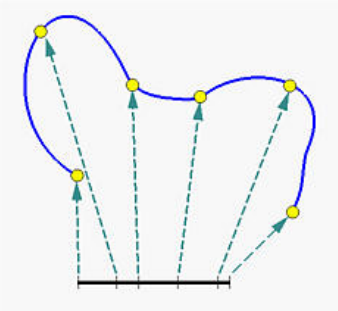

Figure 1.16: Knots of a B-spline curve [75].

This is what is called *B-spline curve* [21, 39, 56]. To summarize, in order to design a B-spline curve we need a set of control points, a set of knots and a set of coefficients, that is one for each control point, so that all curve segments are joined together satisfying certain continuity conditions.

So B-spline basis function will be used and will have the same function of Bezier basis functions, i.e. weight, but they are much more complex, because:

- the domain is subdivided by knots.
- basis functions are not non-zero on the entire interval. In fact, each B-spline basis function is non-zero on a few adjacent subintervals and, as a

result, B-spline basis functions are quite "local".

1.3.1 Definition

Let $\mathbf{U}$ be a set of $m+1$ non-decreasing numbers $u_0 \leq u_2 \leq u_3 \leq \ldots \leq u_m$. The u_i's are called *knots*, the set $\mathbf{U}$ the *knot vector*, and the half-open interval $[u_i, u_{i+1})$ the *i-th knot span*. Note that since some u_i's may be equal, some knot spans may not exist. If a knot u_i appears k times (i.e. $u_i = u_{i+1} = \ldots = u_{i+k-1}$), where $k > 1$, u_i is a *multiple knot* of multiplicity k. Otherwise, if u_i appears only once, it is a *simple knot*. If the knots are equally spaced, the knot vector or the knot sequence is said *uniform*, otherwise it is *non-uniform* [75].

To define B-spline basis function, it is necessary to introduce one more parameter, the degree of these basis functions, p. The i-th B-spline basis function of degree p, written as $N_{i,p}(u)$, is defined recursively as follows:

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u \leq u_{i+1}, \\ 0 & \text{otherwise,} \end{cases} \tag{1.6}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u). \tag{1.7}$$

So given $n+1$ control points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\ldots$, $\mathbf{P}_n$ and a knot vector $\mathbf{U} = \{u_0, u_1, \ldots, u_m\}$, the B-spline curve of degree p is defined as:

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u) \mathbf{P}_i. \tag{1.8}$$

As we can notice, the form of a B-spline curve is very similar to that of a Bezier curve, but it involves more information. Consequently, B-spline curves are more general than Bezier curves, as shown previously in figure 1.7. To change the shape of a B-spline curve, we can modify one or more of these control parameters: the position of control points, the position of knots and the degree of the curve.

1.3.2 Properties

Here are reported some important properties of a B-spline curve, many of which resemble those of Bezier basis functions [75]:

- $N_{i,p}(u)$ is degree p polynomial in u.

- Non-negativity property: for all i, p and u, $N_{i,p}(u)$ is non-negative.
- Local support: $N_{i,p}(u)$ is a non-zero polynomial on $[u_i, u_{i+p+1})$.
- On any knot span $[u_i, u_{i+1})$, at most $p+1$ degree p basis functions are non-zero, namely: $N_{i-p,p}(u)$, $N_{i-p+1,p}(u)$, $N_{i-p+2,p}(u)$, $\ldots$, $N_{i-1,p}(u)$ and $N_{i,p}(u)$.
- Partition of unity: the sum of all non-zero degree p basis functions on span $[u_i, u_{i+1})$ is 1.
- If the number of knots is $m+1$, the degree of the basis functions is p, and the number of degree p basis functions is $n+1$, then $m = n+p+1$.
- Basis function $N_{i,p}(u)$ is a composite curve of degree p polynomials with joining points at knots in $[u_i, u_{i+p+1})$.
- At a knot of multiplicity k, basis function $N_{i,p}(u)$ is C^{p-k}.

1.4 NURBS

B-spline and Bezier curves are polynomial curves [75]. While they are flexible and have many nice properties for curve design, they are not able to represent the simplest curve: the circle. Indeed, circles can only be represented with rational functions[3]. That is why it is necessary to extend the definition of B-spline curves.

In figure 1.17 four closed B-spline curves with 8 control points are shown. The degrees of the curves, from left to right, are 2, 3, 5 and 10, respectively.

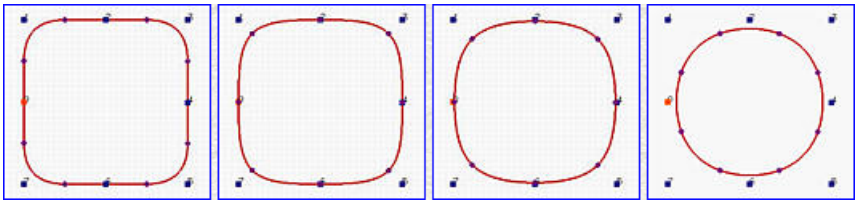

Figure 1.17: B-spline for the approximation of a circle [75].

As the degree increases, the roundness of the curve gets clearer and clearer. The last curve is a closed curve of order 10 and it is very similar to a circle,

[3]Functions that are quotients of two polynomials.

although obviously not a circle. Besides the degree of 10 is too high to be used to represent a 2 degree curve.

To solve this problem, B-splines shall be generalized to rational curves using homogeneous coordinates. There comes the name **N**on-**U**niform **R**ational **B**-**S**plines curves (or NURBS curves) [39, 55, 64, 75].

1.4.1 Definition

Given n+1 control points $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$ and knot vector $U = (u_0, u_1, \ldots, u_m)$ of $m + 1$ knots, the B-spline curve of degree p defined by previously introduced elements is the following:

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\,\mathbf{P}_i. \tag{1.9}$$

Let control point $\mathbf{P}_i$ be rewritten as a column vector with *four* components with the fourth one being 1:

$$\mathbf{P}_i = \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}, \tag{1.10}$$

$\mathbf{P}_i$ can be treated as a homogeneous coordinate. Since multiplying the coordinates of a point (in homogeneous form) with a non-zero number does not change its position, the coordinates of $\mathbf{P}_i$ are multiplied with a weight w_i to obtain a new form in homogeneous coordinates.

$$\mathbf{P}_i^w = \begin{bmatrix} w_i x_i \\ w_i y_i \\ w_i z_i \\ w_i \end{bmatrix}. \tag{1.11}$$

Note that $\mathbf{P}_i^w$ and $\mathbf{P}_i$ represent the same point in homogeneous coordinate. Plugging this new homogeneous form into the equation of the B-spline curve, $\mathbf{C}^w(u)$ can be obtained:

$$\mathbf{C}^w(u) = \sum_{i=0}^{n} N_{i,p}(u)\,\mathbf{P}_i^w = \begin{bmatrix} \sum_{i=0}^{n} N_{i,p}(u)\,(w_i x_i) \\ \sum_{i=0}^{n} N_{i,p}(u)\,(w_i y_i) \\ \sum_{i=0}^{n} N_{i,p}(u)\,(w_i z_i) \\ \sum_{i=0}^{n} N_{i,p}(u)\,w_i \end{bmatrix}. \tag{1.12}$$

Therefore, point $\mathbf{C}^w(u)$ is the original B-spline curve in homogeneous coordinate form. Now $\mathbf{C}^w(u)$ is converted back to Cartesian coordinates by dividing it with the fourth coordinate:

$$\begin{aligned}\mathbf{C}(u) &= \sum_{i=0}^{n} \frac{N_{i,p}(u)\, w_i}{\sum_{j=0}^{n} N_{j,p}(u)\, w_j} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = \dots \\ \dots &= \frac{1}{\sum_{i=0}^{n} N_{i,p}(u)\, w_i} \sum_{i=0}^{n} N_{i,p}(u)\, w_i \mathbf{P}_i.\end{aligned} \tag{1.13}$$

This is the NURBS curve of degree p defined by control points $\mathbf{P}_0$, $\mathbf{P}_1$, ..., $\mathbf{P}_n$, knot vector $U = (u_{0,}u_1, \dots, u_m)$, and weights w_0, w_1, ..., w_n. Note that since weight w_i is associated with control point $\mathbf{P}_i$ as its fourth component, the number of weights and the number of control points must match.

There are two results available immediately from the above definition:

- If all weights are equal, a NURBS curve reduces to a B-spline curve.
- NURBS curves are rational.

These two results indicate that B-spline curves are special cases of NURBS curves. Moreover, since NURBS curves are rational, circles, ellipses and may other curves which are impossible to represent with B-spline curves are instead possible with NURBS curves.

From a geometric interpretation point of view, NURBS curves are not special kind of curves, instead they are simply another face of B-spline curves. The control point (1.11) has four components and can be considered as a point in the four-dimensional space, and, consequently, $\mathbf{C}(u)$ becomes a B-spline curve in the four-dimensional space.

1.4.2 Properties

Given a set of $n+1$ control points $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_n$, each of which is associated with a non-negative weight w_i and a knot vector $U = (u_0, u_1, \dots, u_m)$ of $m+1$ knots, the NURBS curve of degree p is defined as follows:

$$\mathbf{C}(u) = \sum_{i=0}^{n} R_{i,p}(u)\, \mathbf{P}_i, \tag{1.14}$$

where $R_{i,p}(u)$ is defined as follows:

$$R_{i,p}(u) = \frac{N_{i,p}(u) w_i}{\sum_{j=0}^{n} N_{j,p}(u) w_j}. \tag{1.15}$$

This notation is useful because it rewrites the definition of a NURBS curve in a form as close to that of a B-spline curve as possible. In the above notation, $R_{i,p}(u)$'s are NURBS basis functions.

Since NURBS is a generalization of B-spline, it should have all properties of B-splines. The following are some of the most important properties for NURBS basis functions:

- $R_{i,p}(u)$ is a degree p rational function in u.
- Non-negativity property: for all i and p, $R_{i,p}(u)$ is non negative.
- Local support: $R_{i,p}(u)$ is a non-zero on $[u_i, u_{i+p+1})$.
- On any knot span $[u_i, u_{i+p+1})$, at most $p+1$ degree p basis functions are non-zero.
- Partition of unity: the sum of all non-zero degree p basis functions on span $[u_i, u_{i+1})$ is 1.
- If the number of knots is $m+1$, the degree of the basis functions is p, and the number of degree p basis functions is $n+1$, then $m = n+p+1$.
- Basis function $R_{i,p}(u)$ is a composite curve of degree p rational functions with joining points at knots in $[u_i, u_{i+p+1})$.
- At a knot of multiplicity k, basis function $R_{i,p}(u)$ is C^{p-k} continuous.
- If $w_i = c$ for all i, where c is a non-zero constant, $R_{i,p}(u) = N_{i,p}(u)$.

Besides there are some more important properties of NURBS curves that are worth to be taken into consideration:

- NURBS curves can be open, clamped and closed.
- $\mathbf{C}(u)$ is a piecewise curve with each component a degree p rational curve.
- A clamped NURBS curve $\mathbf{C}(u)$ passes through the two end control points $\mathbf{P}_0$ and $\mathbf{P}_n$.

- Strong convex hull property: the NURBS curve is contained in the convex hull of its control points. Moreover, if u is in knot span $[u_i, u_{i+1})$, then $\mathbf{C}(u)$ is in the convex hull of control points $\mathbf{P}_{i-p}$, $\mathbf{P}_{i-p+1}$, $\ldots$, $\mathbf{P}_i$.
- Local modification scheme: changing the position of control point $\mathbf{P}_i$ only affects the curve $\mathbf{C}(u)$ on the interval $[u_i, u_{i+p+1})$.
- Projective invariance: if a projective transformation is applied to a NURBS curve, the result can be constructed from the projective images of its control points. Note that Bezier curves and B-spline curves only satisfy the affine invariance property.

Another interesting property is that increasing or decreasing the value of weight w_i pulls or pushes the curve toward or away from control point $\mathbf{P}_i$, as showed in figure 1.18.

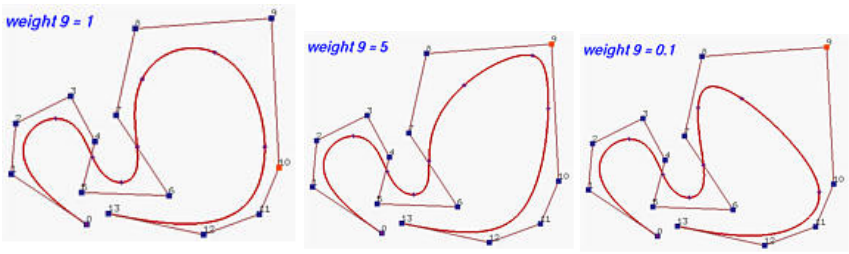

Figure 1.18: Changing w_9 [75].

1.5 Surfaces

The methods examined so far can also be applied also to the treatment of surfaces in tridimensional domain.

There are two types of surfaces that are commonly used in modeling systems: *parametric* and *implicit*. The focus of this work is to be able to manipulate the first kind of surface.

In general, parametric surfaces are defined by a set of three functions, one for each coordinate, as follows [75]:

$$\mathbf{f}(u,v) = (x(u,v), y(u,v), z(u,v)), \tag{1.16}$$

where parameters u and v are in certain domain. For simplicity, they are assumed both in the range of 0 and 1. Thus, (u,v) is a point in the square defined by

$(0,0)$, $(1,0)$, $(0,1)$ and $(1,1)$ in the uv-coordinate plane, as shown in the next figure:

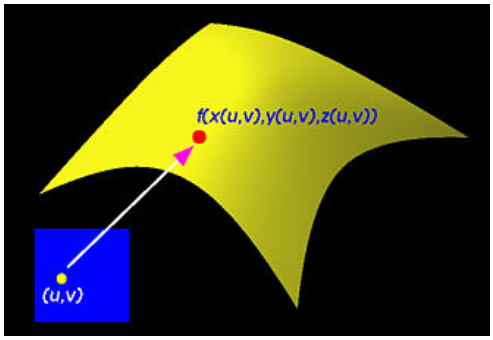

Figure 1.19: Relation between the uv-coordinate plane and the surface [75].

A parametric surface patch can be considered as a union of infinite number of curves, as can Bezier curves or other spline curves be.

There are many ways to form these unions of curves, but the simplest one being the so-called *isoparametric curves*: given a parametric surface $\mathbf{f}(u,v)$, if u is fixed to a value and let v vary, this will generate a curve on the surface whose u coordinate is a constant. Similarly, fixing v to a value and letting u vary, we obtain an isoparametric curve whose v direction is a constant.

Thus a *Bezier surface* $\mathbf{S}(u,v)$ is defined by a grid of control points $\mathbf{P}_{i,j}$, where $0 \leqslant i \leqslant m$, $0 \leqslant j \leqslant n$ and $0 \leqslant u \leqslant 1$, $0 \leqslant v \leqslant 1$.

$$\mathbf{S}(u,v) = \sum_i \sum_j B_{m,i}(u)\, B_{n,j}(v)\, P_{i,j}, \tag{1.17}$$

where $B_{m,i}(u)$ and $B_{n,j}(v)$ are the Bezier basis functions for the surface (see section 1.2). Since they are degree m and degree n, we shall say this is a Bezier surface of degree (m,n). The set of control points is usually referred to as the Bezier net or control net [75]. *De Casteljau's algorithm* can be extended easily to Bezier surface as well. This is a tensor product, which constructs surfaces by multiplying two curves, that can be either two Bezier curves or two B-spline or NURBS curves. That is the tensor product method constructs a surface by multiplying the basis functions of the first curve with the basis functions of the second, and use the results as the basis functions for a set of two-dimensional control points. Surfaces generated in this way are called *tensor product surfaces* [75]. This concept will be extended in the next chapter, because we will be

talking about the reference domain where FFD is defined, which is in effect a volume, not a surface, but it is defined by a tri-tensorial product.

The expression of the surface can be rearranged as follows [75]:

$$\begin{aligned}\mathbf{S}(u,v) &= \sum_{i=0}^{m}\sum_{j=0}^{n} B_{m,j}(u)\, B_{n,j}(v)\, \mathbf{P}_{i,j} = \dots \\ &\cdots = \sum_{i=0}^{m} B_{m,i}(u) \left[\sum_{j=0}^{n} B_{n,j}(v)\, \mathbf{P}_{i,j}\right] = \sum_{i=0}^{m} B_{m,i}(u)\, \mathbf{Q}_i(v)\,.\end{aligned} \tag{1.18}$$

If v is fixed, this is a Bezier curve in u defined by $m+1$ control points $\mathbf{Q}_0(v)$, $\mathbf{Q}_1(v)$, $\dots$, $\mathbf{Q}_m(v)$. Therefore, we conclude that any isoparametric curve with v fixed is a Bezier curve defined by a set of control points that can be computed from the equation of the surface. Interchanging the role of u and v, we will have the same conclusion for isoparametric curves in the v direction. Figure 1.20 shows isoparametric curves on a Bezier surface in both directions.

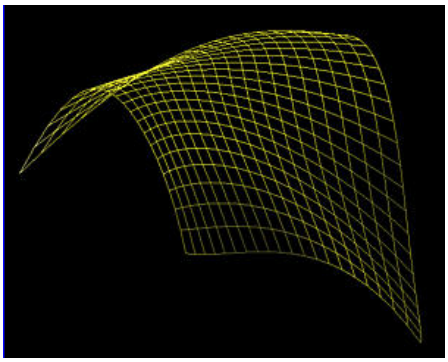

Figure 1.20: Isoparametric curves on a Bezier surface [75].

1.6 Summarizing

As we have seen, there are several methods used to parametrize a surface or a tridimensional body. Those we have examined are the most powerful ones, because they reduce the total number of design variables. On the other hand, the more the complexity of the shape is growing, the more the degree of the polynomials increases dramatically, making the calculations very expensive and even harder and harder to carry out the shape optimization process.

As we mentioned, *Free Form Deformation*, or more briefly *FFD*, can be considered as a valid alternative. As it will be described in the next chapter, FFD can be a very flexible and powerful method and, at the same time, very simple to be used and implemented.

Chapter 2

Free Form Deformation

One of the most important steps in the correct definition of a shape parametrization problem is the choice of the parametrization technique. As seen in chapter 1, several parametrization techniques are available. The problem of defining a solid geometric model of an object bounded by a complex surface has long been identified as an important research problem [22]. In recent years, new and versatile parametrization techniques have emerged, among them: *Free Form Deformation* (*FFD*). Initially FFD was used with solid modeling system [74], recently it has been proposed in a more general context, for example the parametrization of airfoils and wings in a shape optimization context for potential flows [14], in thermal flows [69] and in viscous flows [70], for instance for cardiovascular devices [50].

While other commonly used techniques directly manipulate an object, FFD deforms a lattice that is built around the object itself, and consequently, manipulates the whole space in which the object is embedded. Here are some examples found in literature [45, 74].

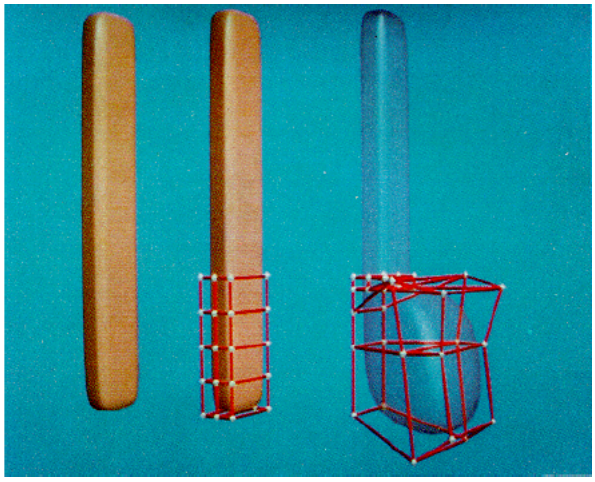

Figure 2.1: Example of local FFD [74].

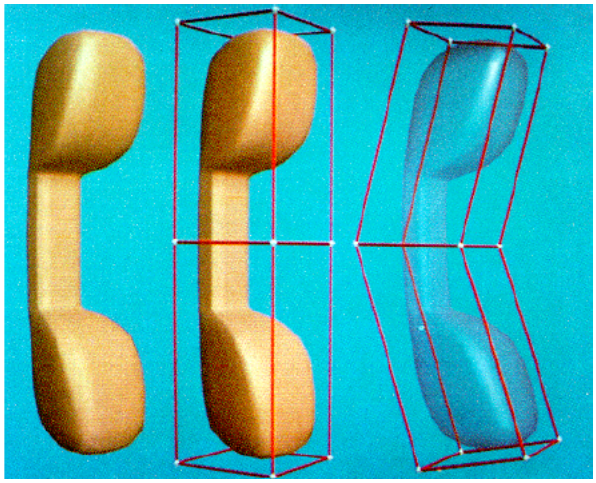

Figure 2.2: Example of global FFD [74].

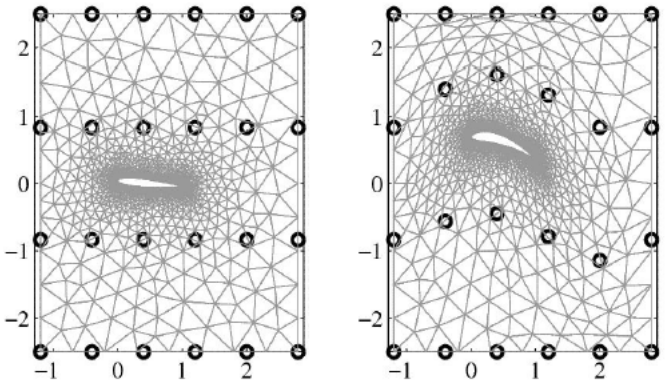

Figure 2.3: Shape optimization using FFD on an airfoil [45].

The lattice has the topology of a cube when deforming 3D objects or a rectangle when deforming two-dimensional objects.

One of the advantages concern the choice of the parameters, which is up to the user. Experiments [14] show that FFD is a low dimensional parametrization that gives a good accuracy even with few parameters and it has a good sensitivity.

FFD can treat surfaces of any formulation or degree and it is independent from the domain or the mesh which is applied to.

A distinguishing aspect of this method is that, by deforming the whole volume around (or inside) the object, the computational grids are also being automatically deformed with the object, which is a valuable characteristic for automated design optimization procedures.

FFD can be applied locally or globally and preserves the shape smoothness (and derivative continuity). In the next sections, after formulating FFD method, some examples will be presented and some properties will be described [14, 16, 31, 45, 72, 74, 79].

Another benefit of using FFD is that the computation is subdivided into an *offline* stage, which is more time consuming, and an *online* part, which can be computed several times once the product of the offline part is stored and it is much cheaper. This fact matches the need of the model reduction method, such as the *Reduced basis method* (*RB*) [27, 28, 44, 50, 67, 68, 69, 70]. Without giving details, RB method is based on an offline and on an online part, as well. So the two methods can be connected, remarkably improving the computational time and the efficiency of the computation.

A possible drawback of FFD is that the design variables may have no physical significance: for instance giving a value of 0.3 to a parameter, this does not correspond to a physical displacement of the corresponding control point of 0.3 unit length, and also the boundary of the shape cannot be controlled because it is a free-boundary value problem. But for the purpose of doing shape optimization, this is not a serious drawback, because the variation of the parameters is directly executed by the optimization process.

Just to mention it, there is an alternative parameterization method that could be coupled with RB method: that is the so-called *radial basis function,* or briefly *RBF*, which uses more control points and is more expensive, but permits to have the control of the boundary displacement [34, 52, 57].

In sections 2.1 and 2.2 the mathematical formulation of FFD method and its implementation will be described, in section 2.3 some of the FFD properties will be treated with a simple bidimensional case, while in the last section 2.4 the extension to the tridimensional case will be shown.

2.1 Formulating FFD

A good physical interpretation for FFD is to consider a rectangle (in 2D) or a parallelepiped (in 3D) of clear, flexible plastic in which an object or several objects are embedded, which are intended to be deformed [45]. We have studied the 3D method, so here follows the tridimensional case.

Defined a reference domain Ω_0 and a subset that we wish to perturb $D_0 \subset \Omega_0$, a differentiable and invertible map is introduced $\Psi : (x_1, x_2, x_3) \to (s, t, p)$, so that $\Psi : (D) \to (0,1) \times (0,1) \times (0,1)$. The FFD is defined in the reference coordinates (s, t, p) of the unit cube. Let us select a regular grid of unperturbed control points $\mathbf{P}^0_{l,m,n}$, where $l = 0, \ldots, L$, $m = 0, \ldots, M$ and $n = 0, \ldots, N$ so that

$$\mathbf{P}^0_{l,m,n} = \begin{bmatrix} l/L \\ m/M \\ n/N \end{bmatrix}. \tag{2.1}$$

A parameter vector $\boldsymbol{\mu}_{l,m,n}$ is introduced, whose dimension is $3 \times (L+1) \times (M+1) \times (N+1)$, because for each control point we consider the possibility to move in three different directions (s, t and p). In figure 2.4 we can show how they are distributed.

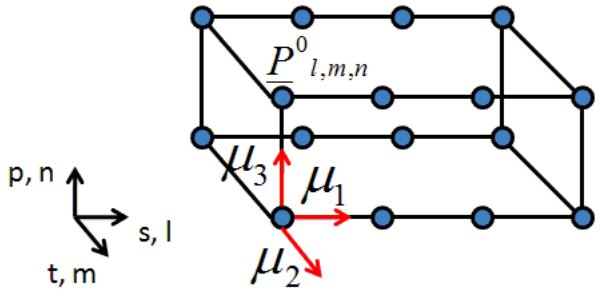

Figure 2.4: Unperturbed control points and parameters vector.

Each control point is perturbed by the corresponding value of the parameters vector:

$$\mathbf{P}_{l,m,n}\left(\boldsymbol{\mu}_{l,m,n}\right) = \mathbf{P}^0_{l,m,n} + \boldsymbol{\mu}_{l,m,n}. \tag{2.2}$$

After which the parametric domain map is constructed $\mathbf{T}: D_0 \to D(\boldsymbol{\mu})$ as

$$\mathbf{T}\left(\Psi\left(\boldsymbol{x}\right);\boldsymbol{\mu}\right) = \Psi^{-1}\left(\sum_{l=0}^{L}\sum_{m=0}^{M}\sum_{n=0}^{N} b^{L,M,N}_{l,m,n}\left(s,t,p\right)\mathbf{P}_{l,m,n}\left(\boldsymbol{\mu}_{l,m,n}\right)\right), \tag{2.3}$$

where

$$\begin{aligned} b^{L,M,N}_{l,m,n}\left(s,t,p\right) &= b^L_l\left(s\right) b^M_m\left(t\right) b^N_n\left(p\right) = \ldots \\ \cdots &= \left(\begin{array}{c} L \\ l \end{array}\right)\left(\begin{array}{c} M \\ m \end{array}\right)\left(\begin{array}{c} N \\ n \end{array}\right)\left(1-s\right)^{(L-l)} s^l \left(1-t\right)^{(M-m)} t^m \\ &\quad \cdot \left(1-p\right)^{(N-n)} p^n, \end{aligned} \tag{2.4}$$

are tensor products of the 1-d Bernstein basis polynomials

$$\begin{aligned} b^L_l\left(s\right) &= \left(\begin{array}{c} L \\ l \end{array}\right)\left(1-s\right)^{(L-l)} s^l, \\ b^M_m\left(t\right) &= \left(\begin{array}{c} M \\ m \end{array}\right)\left(1-t\right)^{(M-m)} t^m, \\ b^N_n\left(p\right) &= \left(\begin{array}{c} N \\ n \end{array}\right)\left(1-p\right)^{(N-n)} p^n, \end{aligned} \tag{2.5}$$

defined on the unit square with local variables $(s,t,p) \in [0,1]\times[0,1]\times[0,1]$, and the function Ψ maps $(x_1,x_2,x_3) \mapsto (s,t,p)$. In figure 2.5 we present the transformation $\mathbf{T}$ and its operational function.

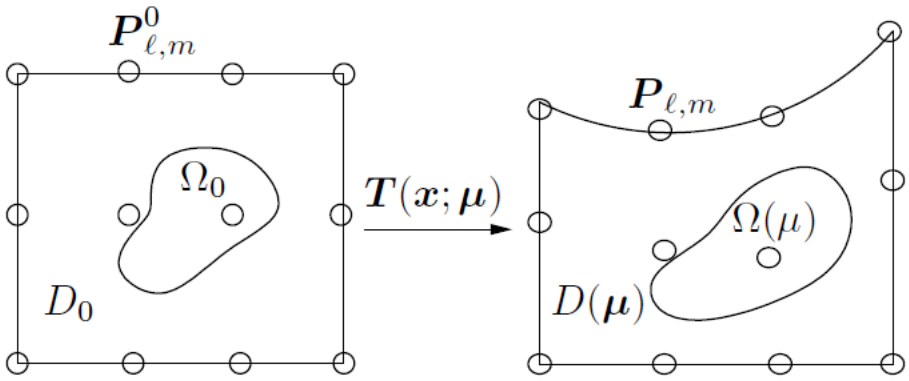

Figure 2.5: The transformation map $\mathbf{T}(\boldsymbol{x};\boldsymbol{\mu})$ and its effect [45].

Properties of Bernstein polynomials or Bezier basis functions have been described previously in section 1.2. It is possible to extend FFD to NURBS or B-spline basis functions [45]. For the purpose of this work, Bernstein basis functions are enough to obtain good and efficacious results and simplify the topic.

As mentioned, in order to effectively calculate the global map $\mathbf{T}$ there are two parts: one *offline*, that is the precomputation of the transformation by the use of a symbolic expression, and the other is *online*, that is the evaluation of the function for the parameters and the coordinates of the real system. This second part is cheaper than ever, even in the 3D case. So once the offline part is completed, that is the part which takes the majority of the cost, the map $\mathbf{T}$ is calculated, so it is enough to evaluate it. For optimization problems, which implies to reiterate the calculus several times, it is an excellent tool.

2.2 Implementing FFD

For the implementation of FFD method the numerical computing environment *MATLAB* has been used. MATLAB supplies a large variety of tools and it is a very powerful computing instrument due to its high intuitivity. It allows the use of symbolic expressions and, least but not last, supplies interfacing with other programs, such as *COMSOL Multiphysics* [25], as it will be described in chapter 3.

The inputs are the boundaries of the domain $[a,b]\times[c,d]\times[e,f]$ to allow the

transformation in reference domain $[0,1] \times [0,1] \times [0,1]$, the number of control points in each direction and finally which parameters decide to influence the transformation. The parameter order is arranged in order to increase, from the first control point to the last, counting the directions (two in 2D, three in 3D). The offline part can start, creating the symbolic expression of $\mathbf{T}$ and relate it from the reference domain to the original, and then storing it. So it is enough to pass the geometry to $\mathbf{T}$, which can be a mesh or the points defining a CAD object, and define the value of the parameter μ. Having stored the $\mathbf{T}$ as a string, which is just a function evaluation manner: that is the online part.

In chapter 3 the interaction between difference programs will be shown.

2.3 FFD Properties

Before considering the extension to the 3D case, some tests have been carried out on a simple 2D case to better highlight the FFD's potential properties. In figure 2.6 the original meshed domain is represented.

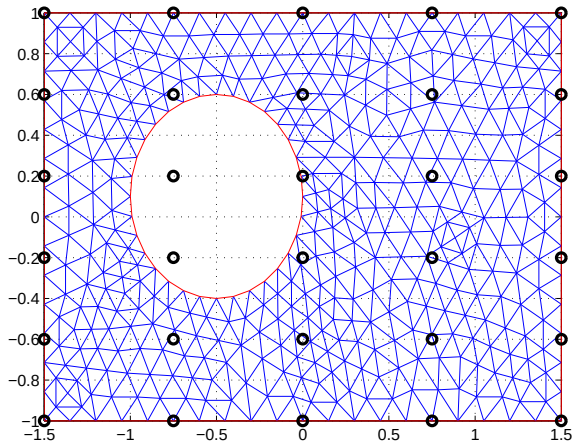

Figure 2.6: Undeformed mesh in 2D domain with control points.

Black points are the control points, and as we mentioned each one has two degrees of freedom, the longitudinal and the transverse direction. Chose which direction and how many parameters consider is up to the user.

In figure 2.7 the control point in red has been moved along the x direction from its unperturbed position, or we can better say that a perturbation of $\mu = 0.6$ has been imposed through the longitudinal axis at the unperturbed control point using the (2.2) equation, and the result in terms of shape perturbation due to the transformation map is shown.

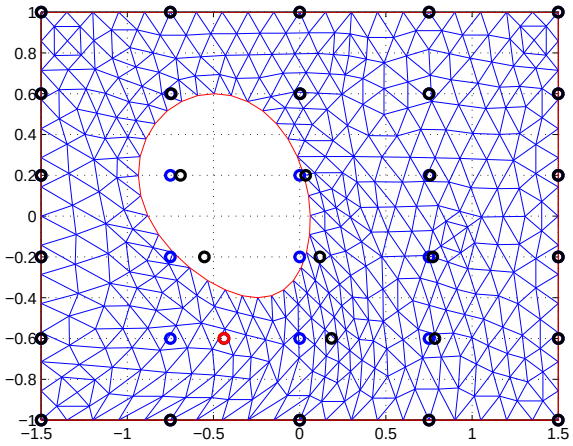

Figure 2.7: Deformed mesh in 2D domain by moving one control point ($\mu = 0.6$).

FFD is a method that involves all the domain, as a lattice created by Bernstein polynomials where all the objects inside are deformed and follow the deformation rule imposed by them. This is demonstrated in figure 2.7, where by just moving one control point in one direction implies the deformation of all the domain. The blue points are the undeformed control points $\mathbf{P}^0_{l,m,n}$, the black ones are the new deformed control points $\mathbf{P}_{l,m,n}$. As we can see, not only the red one has moved, but everything in the domain is moving accordingly with this displacement, assuring a smooth and continuous deformation, no matter how it could be complex. The value of the parameter μ is 0.6, but as we previously mentioned it is absolutely neither physical nor correlated with any unit length, which means that it does not correspond to a physical displacement of 0.6, because the control points and the (2.2) equation are defined in the reference domain $(0, 1) \times (0, 1)$, and not in the physical one after the imposition of the (2.3) transformation.

Another remark arising while looking at figure 2.7 and regarding one important property of Bernstein polynomial, is that it becomes equal to zero on

the boundary. In fact the point just near the red one that lies on the boundary of the domain is not moving. This means that all the deformation takes place only inside the boundary domain, unless we do not exaggerate with the parameter's values, which are usually small. But in that case there would be nodes superimposition, so the deformation would not be acceptable in any case.

The only way to move control points on the boundary is to impose over them the displacement, as shown in figure 2.8. As we have mentioned, the other method that permits to move boundary control points is RBF method [34, 52, 57].

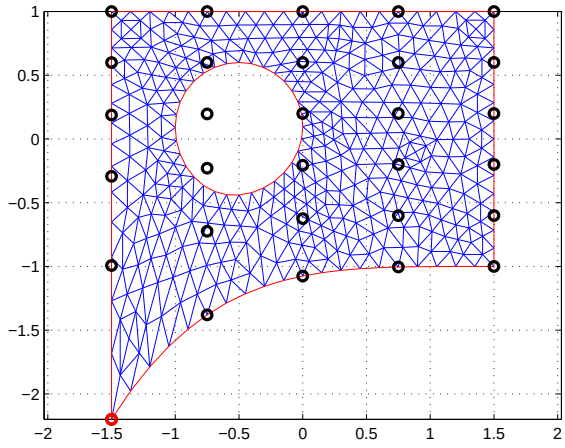

Figure 2.8: Deformed mesh in 2D domain by moving one control point on the boundary ($\mu = -0.6$).

This is the only way to move them. And all the domain follows the deformation, just like a lattice. In this case, the first control point, the red one, is moved in y direction, by imposing a parameter's value of $\mu = -0.6$. Note that also the other control points on the boundary are moving in this case too. This is due to the Bernstein polynomial properties describing the boundary and influenced by the parameter of the control point moved.

But moving control points on the boundary can be more dangerous than moving the ones inside. It is better to do two local FFDs if it is required. An example is showed in figure 2.9. The control point is the same as the one of figure 2.8, but moved by a value of $\mu = 0.6$ instead of -0.6 in y direction. In

this case there is superimposition of mesh points. Note that the deformation of the domain has the behavior of a paper that coils on itself.

It is easier to run into this type of error using boundary control points, that is why we take into consideration only the ones inside the lattice.

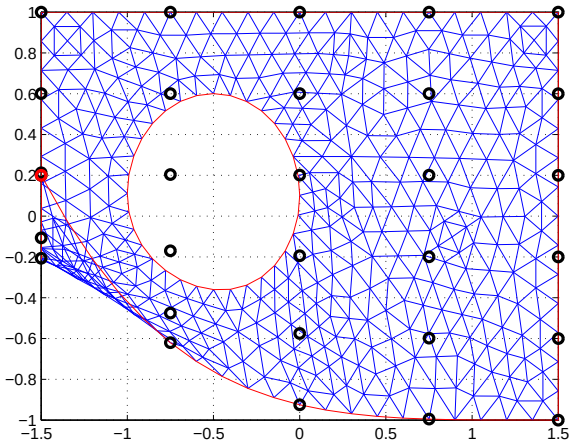

Figure 2.9: Deformed mesh in 2D domain by moving one control point on the boundary ($\mu = 0.6$).

Another thing to highlight is that the use of FFD also reduces the number of shape parameters: in [69] it has been estimated that compared to a local boundary variations approach by moving individual mesh nodes we can obtain a reduction of 238:1 in the number of geometric parameters. This is another strong point in favor of this method, which makes it still more complete and efficient at the same time.

Last but not least, FFD can be applied also locally, in order to deform just a part of the domain and to focus only on it. In figure 2.10 an example of local FFD is presented.

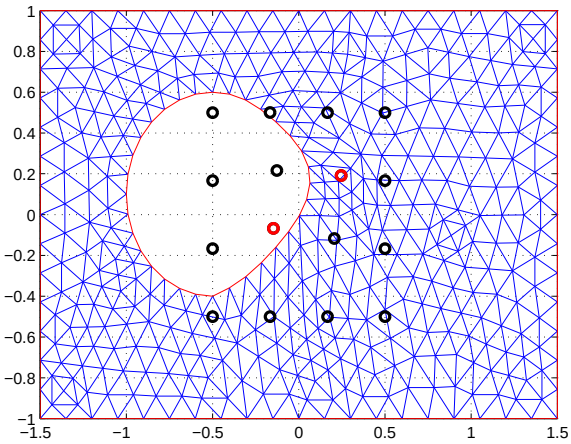

Figure 2.10: Example of local FFD ($\mu_1 = 0.5$ and $\mu_2 = 0.3$).

It has been obtained by moving two control points, which are in red in the figure, imposing on the first control point a vertical perturbation $\mu_1 = 0.5$ and on the second a horizontal perturbation of $\mu_2 = 0.3$. The figure shows the sum of the two deformations. Also, following this idea, multiple FFD blocks can be chained together to enhance the deformation of an object, the control points where the two FFD blocks are joined cannot be deformed. So FFD can be adapted according to the type of the problem and to what one looks to obtain, and from this point of view it is a very versatile method.

As we have seen, it may also allow to maintain the same mesh during the optimization process, due to the fact that the smooth deformation involves all the domain where the FFD lattice is defined, including the mesh points. This is another important feature, because one does not need to remesh for every iteration, but the new mesh follows the deformation. In other words, since FFD is a technique to deform the space, it can be used to deform the mesh and the shape simultaneously [31]. However, one should proceed carefully, in order to ensure that a smooth deformation is imposed, which does not generate overlapping effects in the mesh. The smoothness of the deformation is ensured inside the lattice, since deformation is ruled by Bernstein polynomials. That is why we have chosen to operate just with the internal point of the FFD domain, as we have mentioned.

Another feature that can be taken into consideration is the possibility of implementing a rotation of the FFD lattice. This could be helpful for example in

case the object to be deformed is rotated over a certain angle or it is necessary to deform this object in order to maintain a symmetry not aligned with the ortogonal axes. The mentioned rotation can be obtained by applying an additional rotational matrix $\mathbf{R}$ to compute the global map $\mathbf{T}$, which for a 3D problem is defined as

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.6}$$

where θ is the angle we want to rotate and, in this case, the rotational axis is the third one[1]. In figure 2.11 an example of a rotated local FFD is shown.

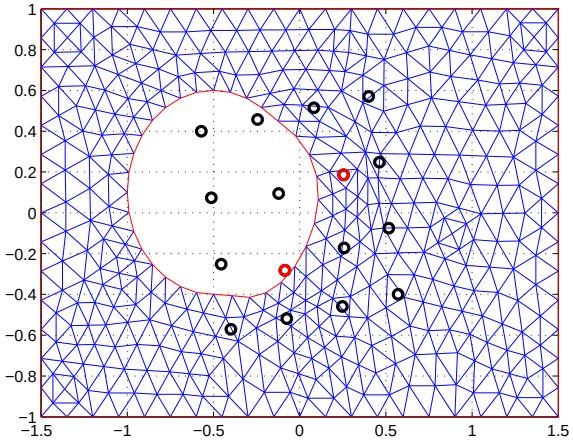

Figure 2.11: Example of a rotated local FFD ($\theta = 10°$, $\mu_1 = -0.5$ and $\mu_2 = 0.6$).

With this feature, the number of applications grew even more and this makes FFD become more general.

2.4 Extension to the 3D case

We have started with a MATLAB code that have been implemented previously by Dr. T. Lassila, EPFL researcher who works on a two-dimensional FFD. The program has been very useful to study the property of this powerful

[1]To obtain $\mathbf{R}$ for the 2D case it is sufficient not to consider the third column and the third row of the matrix (2.6).

methodology and it has also been a good base from where to start. Then we have added the third dimension, and the result was that all the properties that have been tested and mentioned before, continued to be valid. It is a direct consequence of the tensorial product technique, as explained in 1.5 and drove it to the tridimensional case, not speaking about a surface anymore but of a volume, defined by the third-order Bezier tensorial product.

The only difference between the two is that the number of control points drastically increases and, with them, the number of degree of freedom, due to the direction added to the parameters. But the computational cost of the calculus still remains not so onerous, once the offline part is done. Just to give an idea of the gap between the 2D and the 3D case, in figure 2.12 the increase of the computational time for the global **T** map is presented, obtained by varying the number of the parameter chosen to move[2].

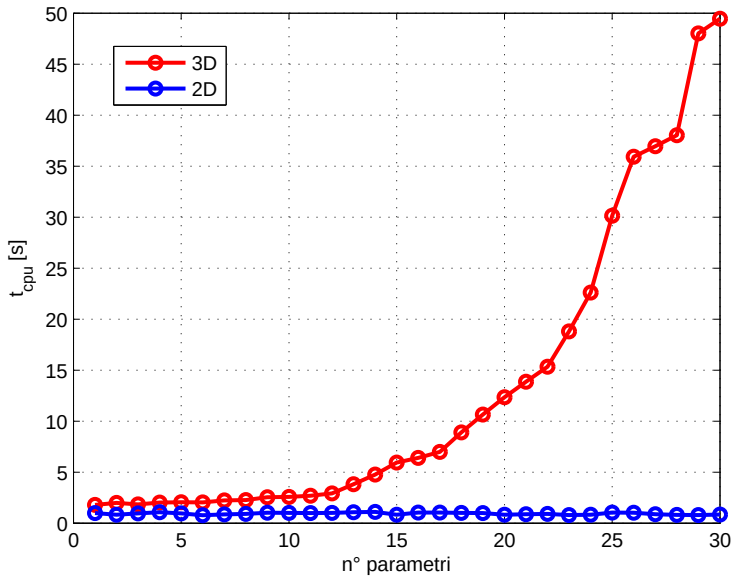

Figure 2.12: Computational time of 2D and 3D FFD.

As can be seen, in 2D it does not matter how many parameters are chosen to be moved, the computational time of **T** remains very small and constant, that is the offline part. Instead, in 3D case, the more are the parameters chosen,

[2]Curves otained using the same computer machine.

the more is the time that is required to obtain $\mathbf{T}$. However, even if the time increases, we can still obtain a rather good range of parameters in a relatively limited period of time.

In figures 2.13 and 2.14 an ellipsoid is presented with diameters $a = 1, b = 0.5$ and $c = 0.3$. In figure 2.15 and 2.16 it is shown deformed by using two control points, one on the upper side and lower side.

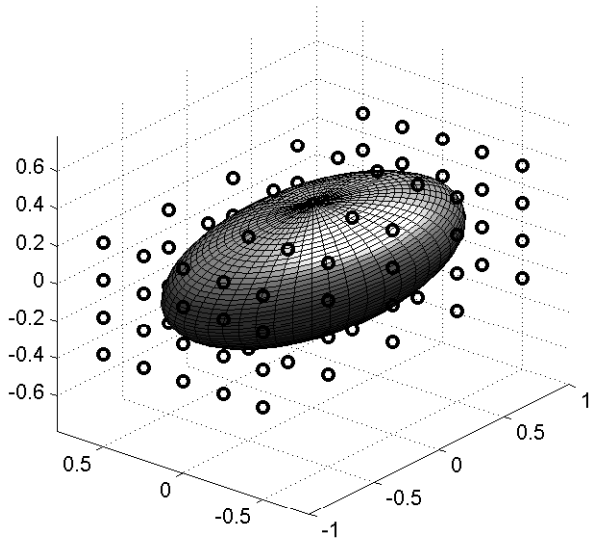

Figure 2.13: Ellipsoid embedded in the FFD lattice.

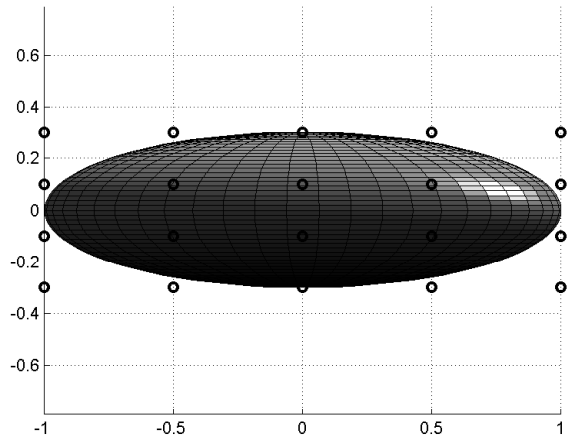

Figure 2.14: The ellipsoid embedded in the FFD lattice from a lateral point of view.

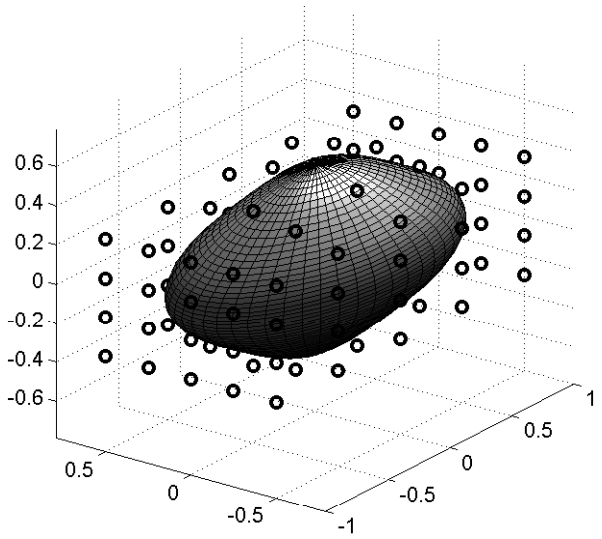

Figure 2.15: Deformed ellipsoid moving two points, one on the top and one on the bottom.

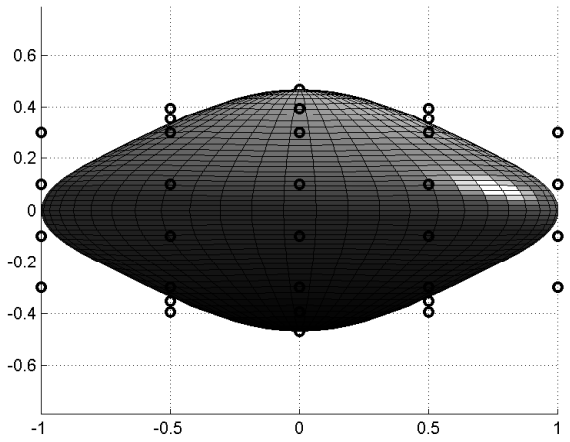

Figure 2.16: Deformed ellipsoid by a lateral sight.

In figures 2.17 and 2.19 more complex objects are presented, a sport car and a civil modern airplane, imported in MATLAB as STL file, and for example they have been deformed in figures 2.18 and 2.20, just to point out that the complexity of the shape is not relevant and that FFD can be applied to every kind of object.

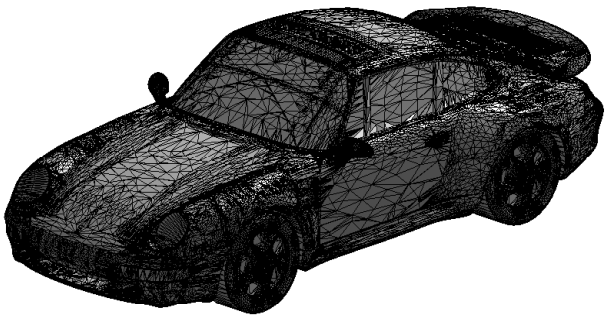

Figure 2.17: A car imported as STL file in MATLAB.

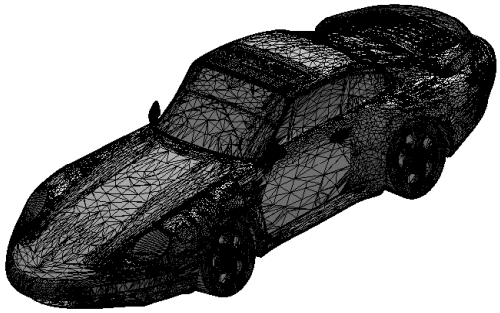

Figure 2.18: The deformed "stretched" car using FFD.

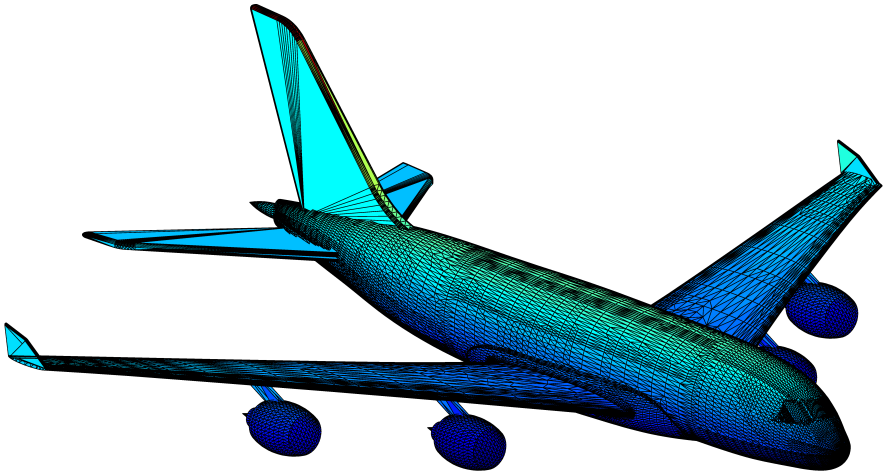

Figure 2.19: An A380 imported as STL file in MATLAB.

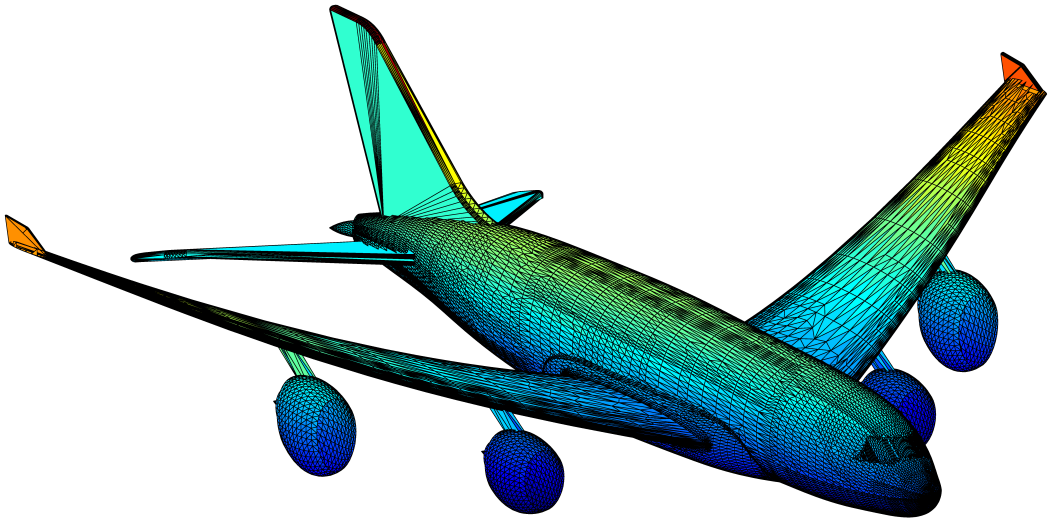

Figure 2.20: Deformed “filled” A380 using FFD.

Chapter 3

Geometrical modeling tools

In order to achieve the purposes of this work, that is studying and applying an efficient method of deformation and using it in an automatic shape optimization process, it is necessary to have the appropriate tools to work with and have efficient interactions among them. So there is the necessity to create an initial geometry, define the problem we want to solve and implement by the FFD method. For this reason, many options and softwares have been used and explored, and it has been necessary to study a connection between them, so that they can interact and communicate with proper interfaces and consistent data.

A CAD software (Computer-Aided Design) has been necessary for the setting up and design of the models and their geometries that we want to optimize. For this purpose, many CAD softwares are available, such as *Rhino* [8], *AutoCad* [2], *Maya* [3]. *SOLIDWORKS* [1], used in this work, is the starting point, where all the initial design decisions are taken. This aspect will be discussed in section 3.1, where the models that we have taken into consideration are also presented.

Secondly, the geometries are exported in a suitable format and imported into the equation solver program. *COMSOL Multiphysics* has been used to define the problem and to solve it at every iteration of the optimization process. COMSOL Multiphysics has also a natural attitude to be linked with the numerical computing environment *MATLAB*, where FFD method is implemented and inserted it in an optimization algorithm. An overview of COMSOL Multiphysics will be given in section 3.2.

In figure 3.1 we show the interaction between the softwares considered and the available connections that has been studied and extended in this work.

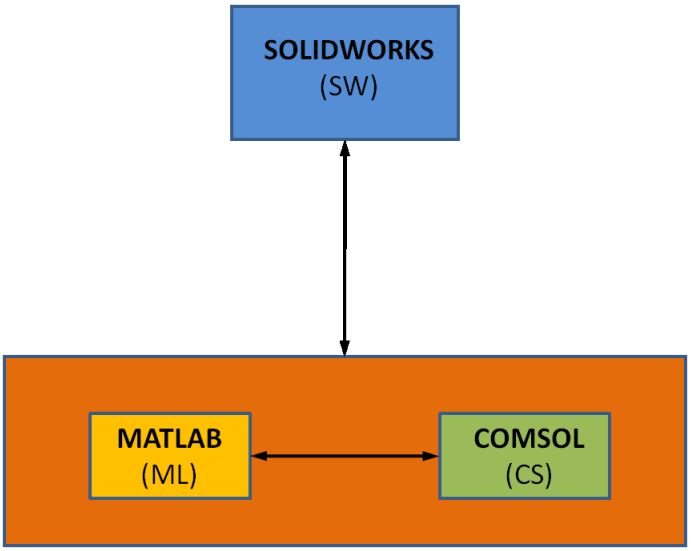

Figure 3.1: Interaction diagram between the softwares used.

These interactions cast the whole cycle of the shape design in an optimization contest possible: every block represents a different action on the process. We can see it from figure 3.2, where the colors of the blocks are associated to the program that fulfills the operation and the relative program is indicated in brackets, same as figure 3.1.

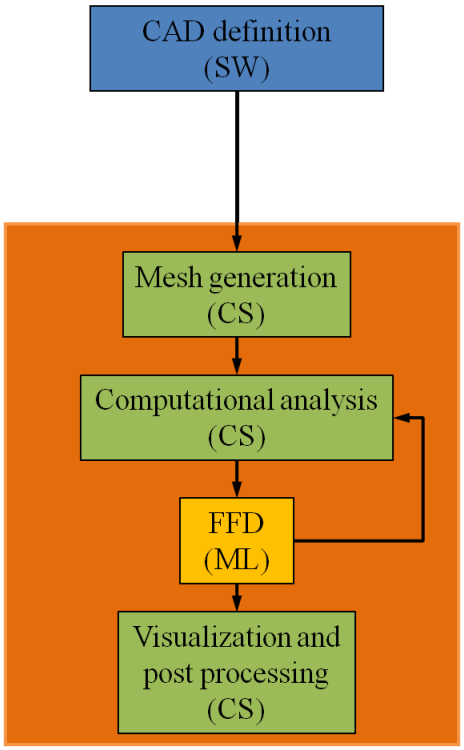

Figure 3.2: Operations carried out by the softwares.

3.1 SOLIDWORKS modeling

SOLIDWORKS is a 3D mechanical CAD program. It uses a parametric feature-based approach to create models and to assemble than. It is very intuitive and handful to use.

SOLIDWORKS has a very high number of formats available in which they could be saved. The ones that we highlight are:

STL is a widely used file format, native to the stereolothography CAD software. This kind of file describes only a raw unstructured triangulated surface geometry of a three dimensional object from the unit normal and vertices (ordered by the right-hand rule) of the triangles using a three-dimensional Cartesian coordinate system. STL files are supposed to be closed and connected like a combinatorial surface, where every edge is part of exactly two triangles, and not self-intersecting [71].

IGS/IGES the Initial Graphics Exchange Specification (IGES or IGS) defines a neutral data format that allows the digital exchange of information among CAD systems, different from one another, and so having different internal data set and representation. It is one of the most adopted format which is used to save CAD models [48], others can be VDA-FS [9], STEP [5] or DXF [4].

These are the main two formats that we have used to save the geometrical models and to import them to the other programs. The STL files can be imported both in MATLAB and COMSOL Multiphysics, however it is preferable to work with the IGES files [48], because with these files the geometry is not reduced to a surface approximating the original one by triangles, which is a limit for simulations and can compromise the results. In COMSOL Multiphysics it is possible to import the IGES files, and work directly on the geometry as it were in SOLIDWORKS. Once the geometry is imported into COMSOL Multiphysics, it is possible to create the mesh and to define the mathematical problem, designating the solving equations and the boundary conditions on the domain. These aspects will be treated in chapter 4.

In conclusion the model is created in SOLIDWORKS and it is saved as an IGES file. Consequently the file is imported to COMSOL Multiphysics, which can be used from a MATLAB interface, in order to combine the tools and the good features and versatility of both.

STL files still can be imported directly in MATLAB, thanks to an M-file that converts the information inside the STL file into loadable points that can be used in MATLAB. For this reason and thanks to the fact that they are compatible with many different softwares, STL files are easier to export and to use. However, as we have mentioned, the drawback of this kind of geometrical representation of the model is the approximation of the real geometry by triangulation. For a more detailed explanation, in appendix A there is an example of how this can be a limit to properly describe a deformed shape. Through the STL format a lot of information is lost, and it is impossible to save an object with its internal volume, but just its surface [71].

3.1.1 CAD Models

In figure[1] 3.3 a complete underwater part of a sailboat, with all its components, is shown.

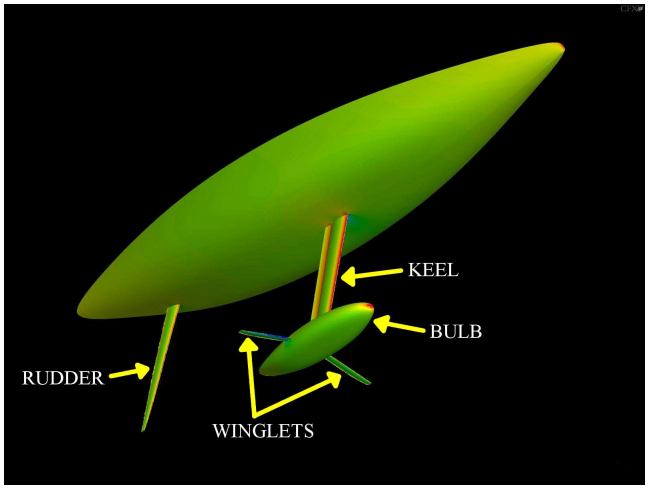

Figure 3.3: A whole sailboat with its parts.

In particular, in this work we have focused on the shape optimization of two of its parts: the bulb and the rudder, which have both been created in SOLIDWORKS. First in figure 3.4 the bulb is presented, which has a much less complex shape than the shape of the rudder. It is just an initial surface, which will be given as an initial guess in the optimization process. Keel and winglets (see figure 3.3) are not present because usually a shape optimization of a bulb under a uniform flow includes just the geometry of the bulb itself [32, 49].

[1]Image courtesy of CMCS (Chair of Modelling and Scientific Compacting - EPFL - Lausanne).

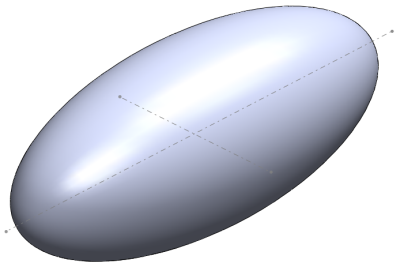

Figure 3.4: Bulb geometrical model created in SOLIDWORKS.

Secondly, in figure 3.5 the construction scheme of the rudder is shown. In figure 3.6 and 3.7 the final version of the rudder is represented from different sides.

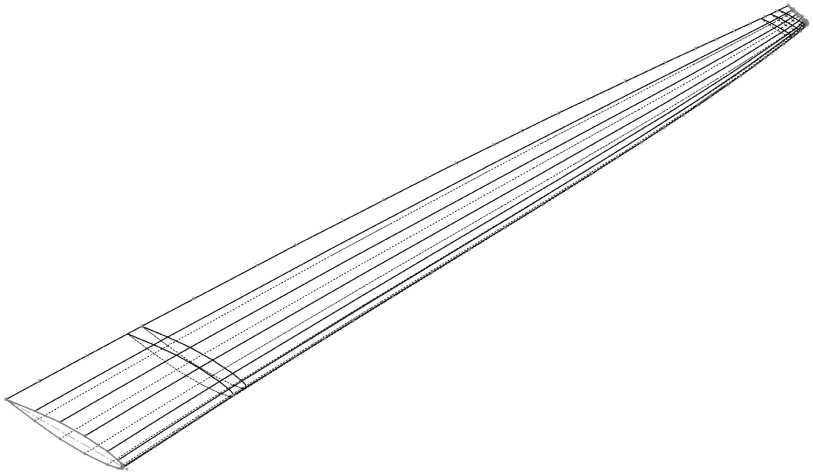

Figure 3.5: Rudder scheme.

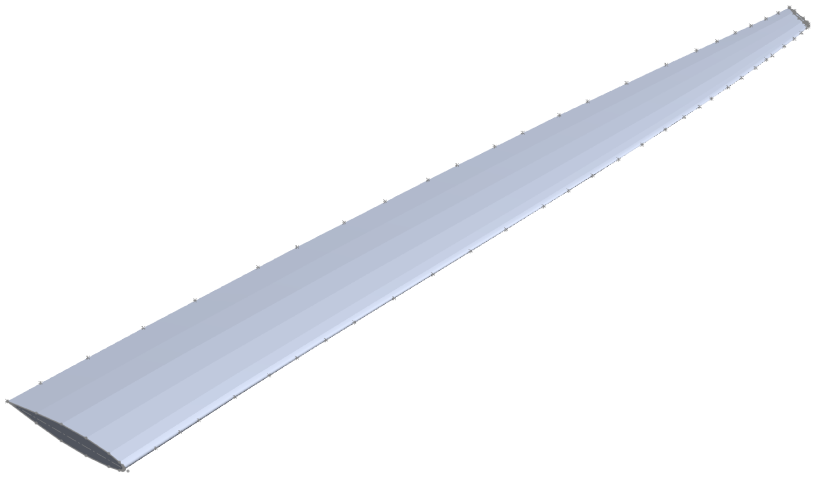

Figure 3.6: Rudder final form.

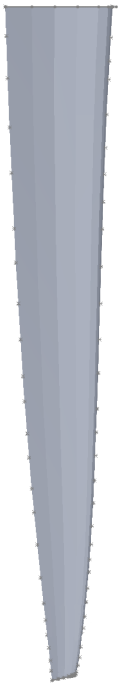

Figure 3.7: Rudder from a lateral side.

In order to achieve this, a mean spanwise airfoil NACA 63012 [11], a root

cord of 0.5 m and a total length of 3.02 m have been used.

As we have said, it is possible to save the models both in STL and in IGS. After importing them in COMSOL Multiphysics, it is necessary to create the mesh in order to solve the problem. If it is a STL file, it can be imported directly to COMSOL Multiphysics as a mesh file, but in this case, it is harder to refine or set a better mesh, and it is not possible to create another one. On the other hand, if the IGES file is imported, in this case only the geometry object is available, consequently the mesh can be created and made suitable for the particular geometry, using the instruments provided by COMSOL Multiphysics.

In the next figures we show a comparison between the mesh of the two formats of the rudder geometry imported to COMSOL Multiphysics. As we mentioned before, in the case of the STL file (figure 3.8) it is already the mesh and it cannot be significantly modified. On the contrary, with the IGS file (figure 3.9) the mesh is created by the user, refined in the areas which requires refinements.

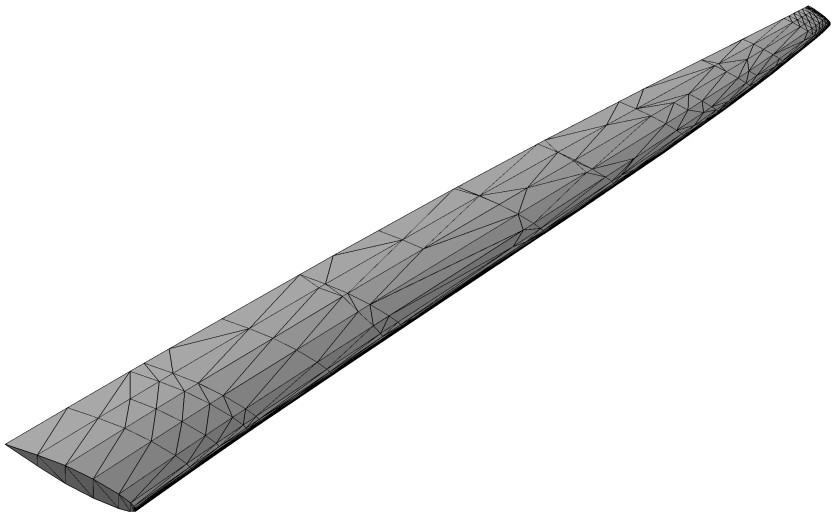

Figure 3.8: STL file of the rudder imported to COMSOL.

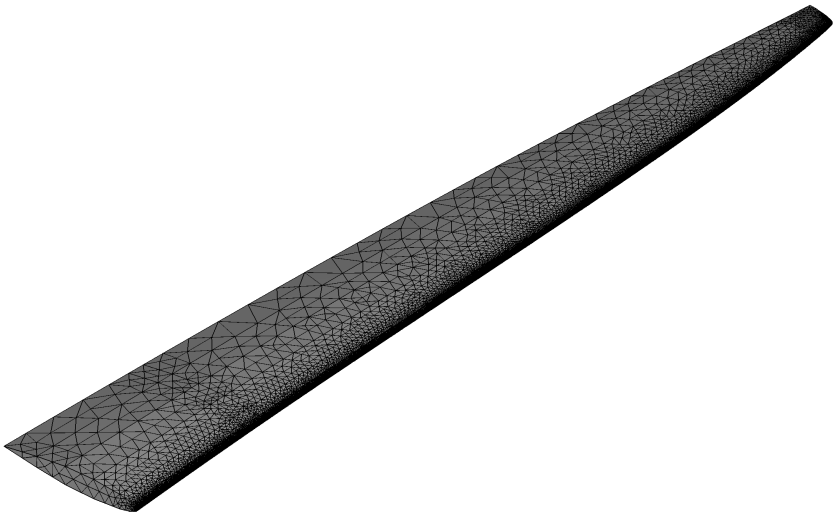

Figure 3.9: IGS file of the rudder imported to COMSOL and meshed.

This is another reason why we have chosen to operate with IGS file in order to guarantee a certain versatility in the mesh generation and management.

3.2 COMSOL Multiphysics

After creating the geometry in SOLIDWORKS, we are ready to import it to *COMSOL Multiphysics*, which is a powerful interactive environment in modeling and solving all kinds of scientific and engineering problems based on partial differential equations (PDEs) [25].

Below there is a brief description of what this software can do and can be used for. Thanks to the built-in physics modes it is possible to build models by defining the relevant physical quantities, such as material properties, loads, constraints, sources, and fluxes, rather than by defining the underlying equations. One can always apply these variables, expressions, or numbers directly to solid domains, boundaries, edges, and points independently of the computational mesh. COMSOL Multiphysics internally compiles a set of PDEs representing the entire model. The mathematical formulation of our model and the PDEs involved will be described in chapter 4.

In creating models you also have a great flexibility in setting up various constants and variables using a number of variables as well as mathematical and

logical functions. When the geometry is complete and the parameters are defined, the geometry can be discretized to have a computational mesh. COMSOL Multiphysics can create mesh on the geometry imported and act with many kinds of mesh, such as free, mapped, extruded, revolved, swept and boundary layer meshes [25]. However, it is possible to operate directly on the mesh-generation process through a set of control parameters, such as maximum element size, or element growth rate. A more detailed description will be given at section 4.3.

When solving the models, COMSOL Multiphysics is based on the finite element method (FEM) [59, 61]. The software runs the finite element analysis together with adaptive meshing and error control using a variety of numerical solvers, such as UMFPACK, PARDISO, GMRES [25, 61]. These solvers can be employed for stationary, eigenvalue and time-dependent problems. In order to solve linear systems, the software features both direct and iterative solvers and a range of preconditioners are available for the iterative solvers [25]. A more detailed methodological description will be given in chapter 4.

COMSOL Multiphysics can be used in many application areas, just to give a few examples:

- Acoustics;
- Heat transfer;
- Chemical reactions;
- Optics;
- Electromagnetics;
- Fluid dynamics;

and many others. Many real-world applications involve simultaneous multiphysics couplings represented by a system of PDEs. COMSOL Multiphysics enables the solution of the former by coupling different problem applications.

And last but not least, as we have mentioned, COMSOL Multiphysics has a practical MATLAB interface [6, 25] so that the FFD algorithm can be applied directly on the model with the use of both softwares properly combined and allow to implement an efficient and automatic shape optimization process.

Chapter 4

Mathematical and numerical formulation of the model problems

In chapter 3 we have described the base of the FFD method and illustrated how the geometrical models have been created and imported for the simulation. The FFD has been tested on them and inserted in a shape optimization process. In this chapter this process will be described, together with the tools used to implement the problem, the mathematical formulation of the model problem and the description of the optimization method used.

4.1 Definition of the problem

We wanted to solve viscous flows around geometries of interest, which is a bulb and a rudder for a boat, for example. Given a domain $\Omega \subset \mathbb{R}^3$, the equations that we have considered are the incompressible steady *Navier-Stokes equations*, which are valid for a viscous flow modelled by a newtonian fluid [15, 20, 58, 59]:

$$\begin{cases} (\mathbf{u} \cdot \nabla)\,\mathbf{u} + \nabla p - \nabla \cdot \left[\nu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right)\right] = \mathbf{f}, & \mathbf{x} \in \Omega \\ \nabla \cdot \mathbf{u} = 0, & \mathbf{x} \in \Omega, \end{cases} \tag{4.1}$$

where ρ is the fluid density, which is constant, $\mathbf{u}$ is the velocity field of the fluid, p is the pression divided by the density, ν is the kinematic viscosity, given by $\nu = \frac{\mu}{\rho}$, where μ is the dynamic viscosity and $\mathbf{f}$ is a forcing term per mass unit. As it is well known, the first equation of the system is the *momentum equation*, the second is the conservation of mass equation, which is known also

as *continuity equation*. The nonlinear term $(\mathbf{u}\cdot\nabla)\,\mathbf{u}$ describes the convective term, while $\nabla\cdot\left[\nu\left(\nabla\mathbf{u}+(\nabla\mathbf{u})^T\right)\right]$ is the molecular diffusion term.

In case of ν constant, applying the continuity equation, we can write:

$$\nabla\cdot\left[\nu\left(\nabla\mathbf{u}+(\nabla\mathbf{u})^T\right)\right]=\nu\left(\Delta\mathbf{u}+\nabla\left(\nabla\cdot\mathbf{u}\right)\right)=\nu\Delta\mathbf{u}, \tag{4.2}$$

so the system (4.1) can be rewritten in a more compact way as

$$\begin{cases}(\mathbf{u}\cdot\nabla)\,\mathbf{u}+\nabla p-\nu\Delta\mathbf{u}=\mathbf{f}, & \mathbf{x}\in\Omega\\ \nabla\cdot\mathbf{u}=0, & \mathbf{x}\in\Omega.\end{cases} \tag{4.3}$$

In order to have a well posed problem, boundary conditions must be assigned (see section 4.1.3). From a generical point of view, we can summirize them into the following expressions:

$$\begin{cases}\mathbf{u}\left(\mathbf{x}\right)=\boldsymbol{\varphi}\left(\mathbf{x}\right), & \forall\mathbf{x}\in\Gamma_D\\ \left(\nu\dfrac{\partial\mathbf{u}}{\partial\mathbf{n}}-p\mathbf{n}\right)\left(\mathbf{x}\right)=\boldsymbol{\psi}\left(\mathbf{x}\right), & \forall\mathbf{x}\in\Gamma_N,\end{cases} \tag{4.4}$$

where $\boldsymbol{\varphi}$ and $\boldsymbol{\psi}$ are assigned vector functions, while Γ_D and Γ_N are portions of the boundary $\partial\Omega$ of Ω, such that $\Gamma_D\cup\Gamma_N=\partial\Omega$ and $\mathbf{n}$ is the normal vector exiting from $\partial\Omega$. In the case where the equations used are the (4.1), the second equation in (4.4) is replaced by

$$\left[\frac{1}{2}\nu\left(\nabla\mathbf{u}+(\nabla\mathbf{u})^T\right)\cdot\mathbf{n}-p\mathbf{n}\right]\left(\mathbf{x}\right)=\boldsymbol{\psi}\left(\mathbf{x}\right),\ \forall\mathbf{x}\in\Gamma_N. \tag{4.5}$$

4.1.1 Weak formulation of the steady Navier-Stokes equations

In order to obtain a weak formulation of the problem, we proceed formally and multiply the first of the (4.3) by a function test $\mathbf{v}$, belonging to an appropriate space V which will be specified, and then integrate on Ω:

$$\int_\Omega\left[(\mathbf{u}\cdot\nabla)\,\mathbf{u}\right]\cdot\mathbf{v}\mathrm{d}\Omega+\int_\Omega\nabla p\cdot\mathbf{v}\mathrm{d}\Omega-\int_\Omega\nu\Delta\mathbf{u}\cdot\mathbf{v}\mathrm{d}\Omega=\int_\Omega\mathbf{f}\cdot\mathbf{v}\mathrm{d}\Omega. \tag{4.6}$$

Using the Green formula we find

$$-\int_\Omega \nu\Delta\mathbf{u}\cdot\mathbf{v}\mathrm{d}\Omega = \int_\Omega \nu\nabla\mathbf{u}\cdot\nabla\mathbf{v}\mathrm{d}\Omega - \int_{\partial\Omega}\nu\frac{\partial\mathbf{u}}{\partial\mathbf{n}}\cdot\mathbf{v}\mathrm{d}\gamma \tag{4.7}$$

$$\int_\Omega \nabla p\cdot\mathbf{v}\mathrm{d}\Omega = -\int_\Omega p\nabla\cdot\mathbf{v}\mathrm{d}\Omega + \int_{\partial\Omega} p\mathbf{v}\cdot\mathbf{n}\mathrm{d}\gamma. \tag{4.8}$$

Replacing these relations in the first equation of (4.3), we obtain

$$\begin{aligned}\int_\Omega \left[(\mathbf{u}\cdot\nabla)\,\mathbf{u}\right]\cdot\mathbf{v}\mathrm{d}\Omega - \int_\Omega p\nabla\cdot\mathbf{v}\mathrm{d}\Omega + \int_\Omega \nu\nabla\mathbf{u}\cdot\nabla\mathbf{v}\mathrm{d}\Omega \\ = \int_\Omega \mathbf{f}\cdot\mathbf{v}\mathrm{d}\Omega + \int_{\partial\Omega}\left(\nu\frac{\partial\mathbf{u}}{\partial\mathbf{n}} - p\mathbf{n}\right)\cdot\mathbf{v}\mathrm{d}\gamma \quad \forall\mathbf{v}\in V.\end{aligned} \tag{4.9}$$

Similarly, we can multiply the second equation of (4.3) by a test function q, belonging to space Q, which is the same space that p belongs to. Integrating on Ω we obtain:

$$\int_\Omega q\nabla\cdot\mathbf{u}\mathrm{d}\Omega = 0 \quad \forall q\in Q. \tag{4.10}$$

We choose V so that the test functions will be equal to zero on the portion of the boundary of the domain where the solution is known.

$$V = \left[\mathrm{H}^1_{\Gamma_D}(\Omega)\right]^3 = \left\{\mathbf{v}\in\left[\mathrm{H}^1(\Omega)\right]^3 :\ \mathbf{v}|_{\Gamma_D} = \mathbf{0}\right\}. \tag{4.11}$$

This will coincide with $\left[\mathrm{H}^1_0(\Omega)\right]^3$ if $\Gamma_D = \partial\Omega$. Let us suppose that $\Gamma_N \neq \emptyset$, then we can choose $Q = \mathrm{L}^2(\Omega)$. Also, we will search for $\mathbf{u}\in\left[\mathrm{H}^1(\Omega)\right]^3$, with $\mathbf{u} = \boldsymbol{\varphi}$ on Γ_D and $p\in Q$. With this space chosen, we can observe that

$$\int_{\partial\Omega}\left(\nu\frac{\partial\mathbf{u}}{\partial\mathbf{n}} - p\mathbf{n}\right)\cdot\mathbf{v}\mathrm{d}\gamma = \int_{\Gamma_N}\boldsymbol{\psi}\cdot\mathbf{v}\mathrm{d}\gamma \quad \forall\mathbf{v}\in V_{\Gamma_D}. \tag{4.12}$$

In [59] all the terms are demonstrated, even the nonlinear one, to be well defined and that all the integrals exist and are limited.

So the weak formulation of the equations is: find $\mathbf{u}\in\left[\mathrm{H}^1_{\Gamma_D}(\Omega)\right]^3$, $p\in Q$ such that

$$\begin{aligned}\int_\Omega \left[(\mathbf{u}\cdot\nabla)\,\mathbf{u}\right]\cdot\mathbf{v}\mathrm{d}\Omega - \int_\Omega p\nabla\cdot\mathbf{v}\mathrm{d}\Omega + \nu\int_\Omega \nabla\mathbf{u}\cdot\nabla\mathbf{v}\mathrm{d}\Omega \\ = \int_\Omega \mathbf{f}\cdot\mathbf{v}\mathrm{d}\Omega + \int_{\Gamma_N}\boldsymbol{\psi}\cdot\mathbf{v}\mathrm{d}\gamma \quad \forall\mathbf{v}\in V \\ \int_\Omega q\nabla\cdot\mathbf{u}\mathrm{d}\Omega = 0 \quad \forall q\in Q.\end{aligned} \tag{4.13}$$

with $\mathbf{u}|_{\Gamma_D} = \boldsymbol{\varphi}_D$ and $V = \left[\mathrm{H}^1_{\Gamma_D}(\Omega)\right]^3$ and $Q = \mathrm{L}^2(\Omega)$ if $\Gamma_N \neq \emptyset$ has been imposed, otherwise $Q = \mathrm{L}^2_0(\Omega)$ if $\Gamma_N = \emptyset$. The existence and the uniqueness of the solution is proven in [35, 37, 59].

By eliminating the pressure, we can rewrite the Navier-Stokes equations in a *reduced form* only in the variable $\mathbf{u}$. This can be obtained, starting from the weak formulation and using the following subspaces of the functional space $\left[\mathrm{H}^1(\Omega)\right]^3$

$$\mathrm{V}_{\mathrm{div}} = \left\{\mathbf{v} \in \left[\mathrm{H}^1(\Omega)\right]^3 : \nabla \cdot \mathbf{v} = 0\right\}, \ \mathrm{V}^0_{\mathrm{div}} = \left\{\mathbf{v} \in \mathrm{V}_{\mathrm{div}} : \mathbf{v} = \mathbf{0} \text{ on } \Gamma_{\mathrm{D}}\right\}. \quad (4.14)$$

We require that the test function $\mathbf{v}$ in the momentum equation (4.13) belongs to the space $\mathrm{V}_{\mathrm{div}}$, the term associated to the pressure gradient vanishes. So we can find the following reduced problem for the velocity: find $\mathbf{u} \in \mathrm{V}_{\mathrm{div}}$ such that

$$\begin{aligned} \int_\Omega \left[(\mathbf{u} \cdot \nabla)\,\mathbf{u}\right] \cdot \mathbf{v}\, \mathrm{d}\Omega + \nu \int_\Omega \nabla \mathbf{u} \cdot \nabla \mathbf{v}\, \mathrm{d}\Omega \\ = \int_\Omega \mathbf{f} \cdot \mathbf{v}\, \mathrm{d}\Omega + \int_{\Gamma_{\mathrm{N}}} \boldsymbol{\psi} \cdot \mathbf{v}\, \mathrm{d}\gamma \qquad \forall \mathbf{v} \in \mathrm{V}^0_{\mathrm{div}}, \end{aligned} \quad (4.15)$$

with $\mathbf{u}|_{\Gamma_{\mathrm{D}}} = \boldsymbol{\varphi}_{\mathrm{D}}$.

If $\Gamma_D = \partial\Omega$ and $\Gamma_N = \emptyset$ then we can consider the two *Hilbert spaces* $V = \left[\mathrm{H}^1_0(\Omega)\right]^3$ and $Q = \mathrm{L}^2_0(\Omega)$ and the weak formulation can be stated as follows: given $\mathbf{f} \in \left[\mathrm{L}^2(\Omega)\right]^3$, find $\mathbf{u} \in V$, $p \in Q$ such that

$$\begin{cases} a(\mathbf{u}, \mathbf{v}) + c(\mathbf{u}; \mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) = (\mathbf{f}, \mathbf{v}) & \forall \mathbf{v} \in V \\ b(\mathbf{u}, q) = 0 & \forall q \in Q, \end{cases} \quad (4.16)$$

where $a : V \times V \to \mathbb{R}$, $b : V \times Q \to \mathbb{R}$ are the bilinear forms, and $\mathrm{F} : V \to \mathbb{R}$ a linear continuous functional. They are defined by

$$a(\mathbf{w}, \mathbf{v}) \doteq \nu(\nabla \mathbf{w}, \nabla \mathbf{v}) = \int_\Omega \nu \nabla \mathbf{w} \cdot \nabla \mathbf{v}\, \mathrm{d}\Omega \quad (4.17)$$

$$b(\mathbf{v}, q) \doteq -(q, \nabla \cdot \mathbf{v}) = -\int_\Omega q \nabla \cdot \mathbf{v}\, \mathrm{d}\Omega \quad (4.18)$$

$$\mathrm{F}(\mathbf{v}) \doteq (\mathbf{f}, \mathbf{v}) = \int_\Omega \mathbf{f} \cdot \mathbf{v}\, \mathrm{d}\Omega. \quad (4.19)$$

$(.,.)$ denotes the scalar product in $\mathrm{L}^2(\Omega)$ or $\left(\mathrm{L}^2(\Omega)\right)^3$. The trilinear form $c : V \times V \times V \to \mathbb{R}$ is defined by

$$c(\mathbf{w}; \mathbf{z}, \mathbf{v}) = \int_\Omega \left[(\mathbf{w} \cdot \nabla)\,\mathbf{z}\right] \cdot \mathbf{v}\, \mathrm{d}\Omega \qquad \forall \mathbf{w}, \mathbf{z}, \mathbf{v} \in V. \quad (4.20)$$

This is the trilinear form associated with the nonlinear convective term. In [35, 63, 59] it is demonstrated that if $a(\cdot,\cdot)$, $b(\cdot,\cdot)$ and $c(.,.,.)$ are continuous and $a(\cdot,\cdot)$ is coercive, under certain conditions (see (4.21)), and the solution $(\mathbf{u},p)$ is a solution of the problem exists and is unique.

Besides, let us recall, for the well posedness of the problem, that the following *inf-sup* condition must be fulfilled:

$$\inf_{q\in Q,q\neq 0}\sup_{\mathbf{v}\in V,\mathbf{v}\neq\mathbf{0}}\frac{b(\mathbf{v},q)}{\parallel \mathbf{v} \parallel_{\mathrm{H}^1(\Omega)}\parallel q \parallel_{\mathrm{L}^2(\Omega)}}\geq\beta>0. \tag{4.21}$$

In numerical discretization context (4.21) will involve an appropriate compatible choice of the finite element spaces for velocity and pressure [63].

4.1.2 Finite elements

In order to solve the problem, a numerical approximation of the Navier-Stokes equations is needed. It can be obtained by applying a *Galerkin method* to the variational formulation (4.15). So, considering two finite elements space $V_h\subset V$, $Q_h\subset Q$ which satisfies the discrete *inf-sup* condition, the Galerkin approximation of the state problem can be expressed in: find $(\mathbf{u}_h,p_h)\in V_h\times Q_h$ such that

$$\begin{array}{ll} a(\mathbf{u}_h,\mathbf{v}_h)+c(\mathbf{u}_h;\mathbf{u}_h,\mathbf{v}_h)+b(\mathbf{v}_h,p_h)=(\mathbf{f},\mathbf{v}_h) & \forall\mathbf{v}_h\in V_h \\ b(\mathbf{u}_h,q_h)=0 & \forall q_h\in Q_h. \end{array} \tag{4.22}$$

For the discretization of both problems, the pair of finite elements, Taylor-Hood elements, $(\mathbb{P}_2-\mathbb{P}_1)$ has been chosen [63], with continuous pressure which results stable and satisfies the discrete *inf-sup* condition. Spaces $V=\left[\mathrm{H}^1_{\Gamma_D}(\Omega)\right]^3$ and $Q=\mathrm{L}^2(\Omega)$ are approximated by the pair of finite elements space $(V_h,Q_h)=\left(\left[X_h^2\right]^3,X_h^1\right)$, where

$$X_h^r=\left\{v\in C^0(\Omega),v|_K\in\mathbb{P}_r(K)\ \forall K\in\mathcal{T}_h\right\}, \tag{4.23}$$

$\mathcal{T}_h$ denotes the triangulation of the domain Ω. Expanding the solution on an appropriate base and expressing

$$\mathbf{u}_h(\mathbf{x})=\sum_{j=1}^{N_u}u_j\varphi_j(\mathbf{x}),\quad p_h(\mathbf{x})=\sum_{j=1}^{N_p}p_k\psi_k(\mathbf{x}), \tag{4.24}$$

taking $\mathbf{v}_h = \varphi_i$ and $q_h = \psi_l$ as test functions, we obtain the following algebraic nonlinear system with $N_u + N_p$ unknowns:

$$\begin{cases} \mathbf{AU} + \mathcal{N}(\mathbf{U}) + \mathbf{B}^T\mathbf{P} = \mathbf{F} \\ \mathbf{BU} = 0, \end{cases} \tag{4.25}$$

where $\mathbf{A}_{i,j} = a(\varphi_j, \varphi_i)$, $\mathbf{B}_{li} = b(\varphi_i, \psi_l)$, $\mathbf{F}_i = (f, \varphi_i)$ and $\mathcal{N}(\mathbf{U}) = \sum_{s,j} u_s u_j$ $c(\varphi_s; \varphi_j, \varphi_i)$. This is the system of equations that we want to solve numerically.

4.1.3 Boundary conditions

As we have anticipated in section 4.1, in order to solve the problem that we have previously defined, boundary conditions have to be applied. In particular, the problem we are considering, by using the introduced Navier-Stokes equations, is to solve the flows around two geometries of a sailing boat, a bulb and a rudder (see figures 3.4 and 3.6). We assume the boundary conditions that correspond to a cruising situation, listed here below:

1. Inlet: uniform velocity $\mathbf{u} = -U_0\mathbf{n}$;
2. Open boundary: normal stress $\left[-p\mathbf{I} + \mu\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right)\right]\mathbf{n} = 0$;
3. Outlet: pressure $p = p_0$ and no viscous stress $\mu\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right)\mathbf{n} = \mathbf{0}$;
4. Wall: no slip $\mathbf{u} = \mathbf{0}$;

where $\mathbf{n}$ is the normal to the face of the domain considered, $\mathbf{u}$ is the velocity vector and p is the pressure. The constants μ, U_0 and p_0 have been introduced in section 4.1. In figures 4.1 and 4.2 we show the boundary conditions and on which face of the domain they have been imposed: the numbers on the domain correspond to the particular condition mentioned before.

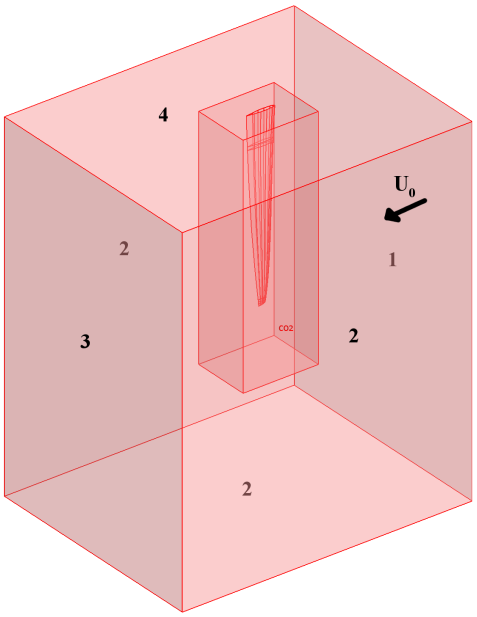

Figure 4.1: Boundary conditions for the rudder problem.

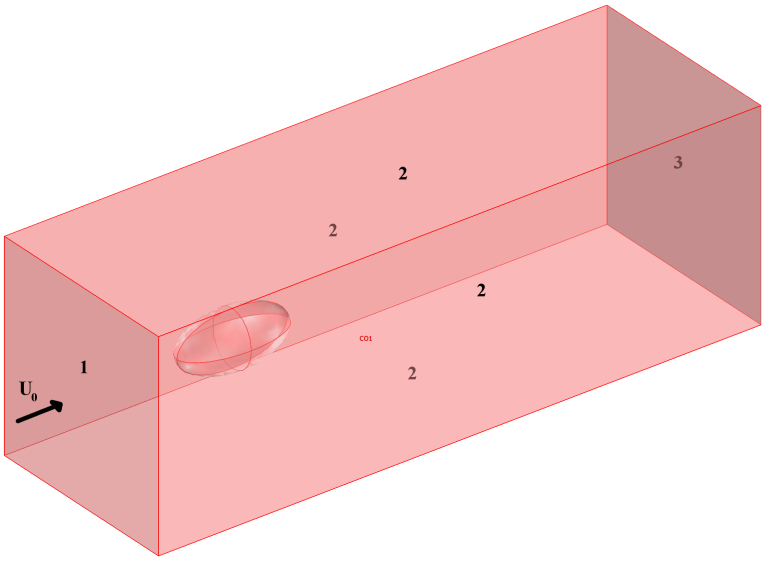

Figure 4.2: Boundary conditions for the bulb problem.

4.2 Approximations

The choice made is taking neither in consideration the instationarity nor any turbulence model, even if the objects we are analyzing are immersed in water. In fact, for these components of the sailboat, high Reynolds number can be easily reached. This is a strong approximation, however, the aim of this work is not to have a perfect real mathematical and physical model, but to analyze the potentialities of the geometrical deformation method, applying it into an optimization process. The approximation permits to relatively lighten the calculus for each iteration of the process, which is still very onerous, and to concentrate more on the FFD aspects.

Another approximation is given from the Reynolds number. Assuming density of water $\rho = 1000\frac{\text{kg}}{\text{m}^3}$, a root cord of the rudder of $c = 0.5$m and a minimum velocity of $V = 10\frac{m}{s}$, the Reynolds number is $\sim 3 \cdot 10^6$, with a dynamic viscosity of $\mu = 1.51$mPa·s. This means that the computational problem to solve the complete Navier-Stokes equation would be a hard task to solve, due to the fact that the power of a common computer machine is not adequately sufficient. It has been tested that to solve the problem with these equations, the maximum Re reachable with COMSOL Multiphysics on a single device is about $\text{Re}_{\text{lim}} \simeq 5000$. For this reason we have decided to contain the Reynolds number and not to introduce turbulence models, which allow an approximate description of this flow behavior through either algebraic or differential equations. This can be a topic for subsequent works.

All the simulations have been done using an *Intel* with a i3 processor of 2.4GHz. Disposing of a high sufficient computational power, such as parallel computing, and associating it with a model reduction technique, such as reduced basis method (see chapter 2) the proposed procedure could be used on several kinds of applications and with a great saving of time and costs. It could be a very interesting matter of research and a case for subsequent works.

In order to contain the Reynolds number and not to introduce (for test purposes) turbulence models, we have assigned the values for the physical variables, which are indicated in table 4.1.

Table 4.1: Constants set used.

	$\rho\,[\text{kg/m}^3]$	$\mu\,[\text{mPa}\cdot\text{s}]$	$U_0\,[\text{m/s}]$	$L\,[\text{m}]$	Re
Rudder	1	1	10	0.5	5000
Bulb	1	10	10	1	1000

where ρ is the fluid density, μ is the dynamic viscosity, U_0 is the free-stream velocity. The reference pressure p_0 has been imposed zero, assuming that it is a difference of pressure in respect to the undisturbed field.

4.3 Mesh and discretization of the problem

After having defined the problem and the mathematical model that we intend to apply to our case, and after having imported the geometries into COMSOL Multiphysics, we can proceed to mesh them.

COMSOL Multiphysics provides many possibilities to mesh the model and it can also fit in very many different needs, as we mentioned in section 3.2. However often there is a limit to the maximum number of the elements allowed, which is imposed by the computer memory, for this reason parallel computing setting would be useful.

We can use different meshing techniques in 3D, for example creating a free mesh which contains tetrahedral elements[1]. Or else one can even create a boundary layer mesh by inserting structured layers of elements along specific boundaries into an existing mesh. When one creates a free mesh, the number of mesh elements is determined from the shape of the geometry, and from various mesh parameters that can be controlled, which can be for instance local mesh-element sizes or element distribution, mesh curvature factor, the resolution of narrow regions.

The rudder (figure 4.7) has been more delicate to mesh because of its leading edge and its roundness which must be preserved. So the mesh has been refined in that region in order not to incur into meshing error problems. To avoid this problem and to have a relatively fine mesh close to the rudder, the main domain has been subdivided in two parts, an internal one where the mesh is intended to be finer, and an external one where the mesh is supposed to be coarser (see figure 4.8). For the bulb (figure 4.6) this operation has not been necessary to do, because its geometry is much less complex and it can be meshed without particular problems.

In figures 4.3 and 4.4 the geometries and the domains are shown, and in figure 4.5 a lateral vision of the rudder for clarity is proposed, while figures 4.6 and 4.7 represent their mesh.

[1]Even hexahedral and prismatic mesh elements are available.

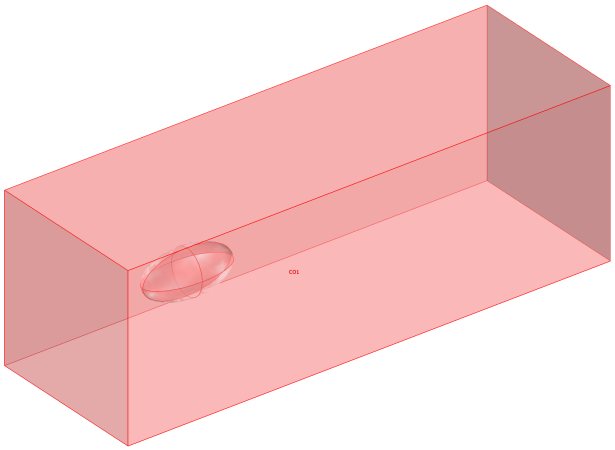

Figure 4.3: Bulb imported to COMSOL Multiphysics and the domain where the problem is defined.

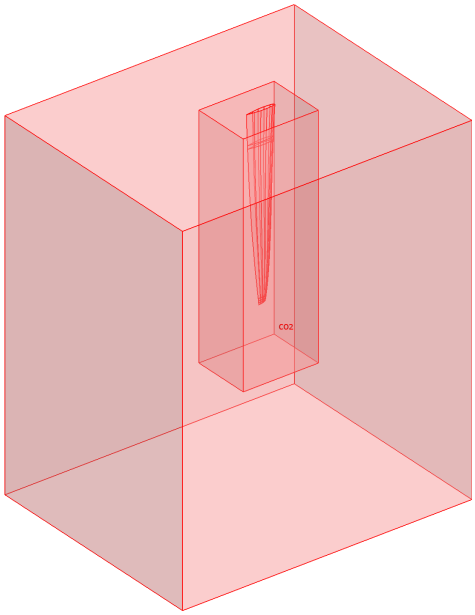

Figure 4.4: Rudder imported to COMSOL Multiphysics and the domain where the problem is defined.

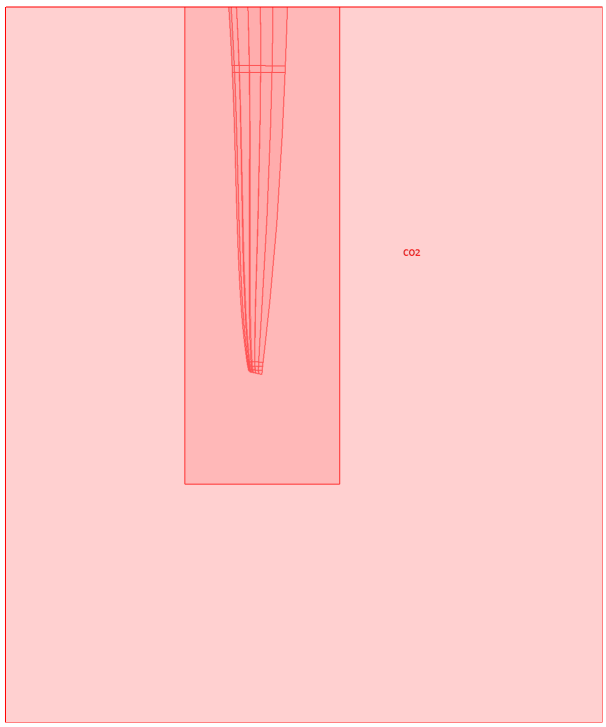

Figure 4.5: Lateral view of the rudder imported to COMSOL Multiphysics and the domain where the problem is defined.

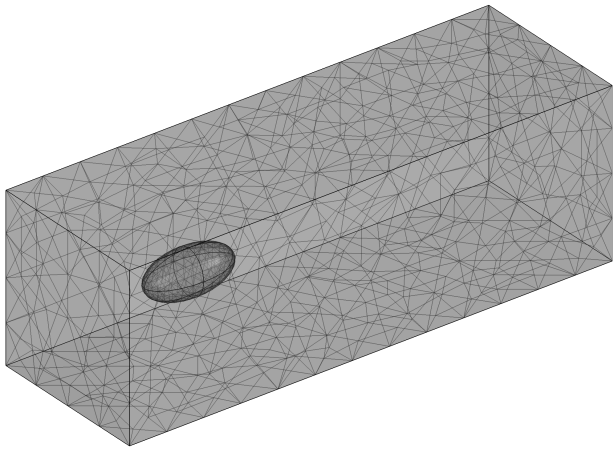

Figure 4.6: The bulb and the mesh of the domain.

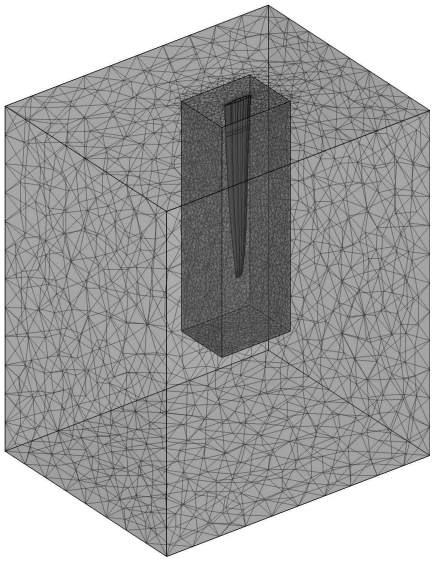

Figure 4.7: The rudder and the mesh of the domain.

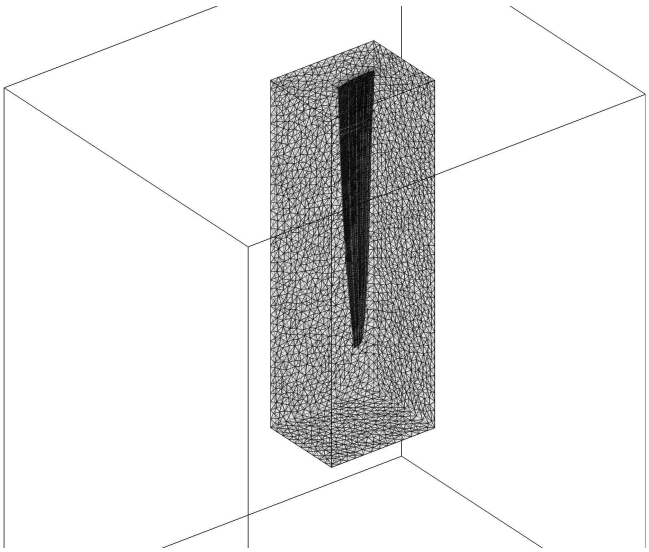

Figure 4.8: A particular of the rudder and the refined mesh of the inner subdomain.

In table 4.2 we can see the information on the meshes of the two models.

Table 4.2: Global mesh statistics.

	Bulb	**Rudder**
Number of mesh points	1857	42482
Number of elements	8904	222138
Number of boundary elements	1434	28124
Minimum element quality	0.364	0.2804
Element volume ratio	4.51E-4	8.01E-7

4.4 Solvers

We have seen what the set of equations to solve is and how it is reconducted in a weak formulation. By default, COMSOL Multiphysics solves both linear and nonlinear problems using the weak solution form. It comes with the nonlinearity of the NS equations that the solver will break the problem down into one or several linear systems of equations [24]. The linear solver solves a corresponding linearized model evaluated at a specified linearization point, which can be a given (already computed) solution. In this work we chose to use as specified given solution the one computed in the previous step.

In COMSOL Multiphysics different typology of solvers are available. Because the number of unknowns in these systems is usually very large, the most important solver parameter setting concerns the choice of a linear solver, which is the most effective linear system solver for the model under study. The linear solves are divided into *direct solvers* or *iterative solvers*. The formers are preferable for 1D and 2D problems, and for 3D problems with few degrees of freedom, which is not our case. Some of them are for example UMFPACK (Unsymmetric MultiFrontal method PACKage) [26], SPOOLES (SParse Object Oriented Linear Equation Solver) [18], PARDISO (PARallel sparse DIrect linear SOlver) [7]. The direct solvers solve a linear system by Gaussian elimination [59, 61]. This stable and reliable process is well suited for ill-conditioned systems [24].

For models with many degrees of freedom, which is our case, the direct solvers typically need too much memory. So, the more memory efficient iterative solvers GMRES (Generalized Minimum RESidual) [29, 78], FGMRES (Flexible GMRES) [29, 78] Conjugate gradient [29, 78], BiCGStab (BiConjugate Gradient Stabilized method) [29, 77, 78], can perform better. However, iterative solvers are less stable than direct solvers because they do not always converge. To improve the convergence, an appropriate *preconditioner* must be chosen, and

the ones available in COMSOL Multiphysics are several, such as Incomplete LU, Geometric multigrid, Incomplete Cholesky or Algebraic multigrid [25].

In the following a list and a valutation of these solvers is provided [25], with a brief explanation of their potential usage:

Table 4.3: Direct and iterative linear system solvers with their usage.

Direct solvers	
UMFPACK	A highly efficient direct solver for nonsymmetric systems.
SPOOLES	An efficient direct solver for symmetric and nonsymmetric systems. It uses less memory than UMFPACK.
PARDISO	A highly efficient direct solver fo symmetric and nonsymmetric systems. It often uses less memory than UMFPACK.
Iterative solvers	
GMRES	An iterative solver for nonsymmetric problems.
FGMRES	An iterative solver for nonsymmetric problems. It can handle more general preconditioners but also uses more memory than GMRES.
Conjugate gradient	An iterative solver for symmetric positive definite problems.
BiCGStab	An iterative solver for nonsymmetric problems. It uses a fixed amount of memory independent of the number of iterations. Therefore, it typically uses less memory than GMRES.

The recommended solver type for small and medium-sized fluid-flow problems is a direct solver [24], and the PARDISO direct solver is the default solver for 1D and 2D models. 3D problems are often too large to solve using a direct solver, and this is our case. The solver used for the simulations concerning the rudder is BiCGStab, an iterative one which can be seen as if using a BiCG and at every step applying a GMRES [77]. The preconditioner used is the geometric

multigrid (GMG) preconditioner. This choice permits to solve a problem saving memory compared to the other solvers mentioned.

For the bulb it has been possible to use the PARDISO solver, due to the fact that the bulb flux is simpler to solve than the rudder flux, due to its symmetry. As we have mentioned, PARDISO, or in general the direct solvers, guarantee a better numerical stability.

4.5 Optimization Process

After defining the problem and the way we intend to carry out the simulations, an introduction of some optimal control and shape optimization notions is recalled below. For further informations see [49, 51, 65]. In section 4.5.3 the optimization process used in this work will be described.

4.5.1 General notions of optimal control

An optimal control problem has three fundamental elements:

- an objective, represented by an appropriate functional, which is called *cost functional,*
- one or more control parameters (*design parameters*),
- a set of constraints which define the behavior of the system under study and it has to be satisfied by the state variables.

So, the purpose of the problem is to compute the state and design variables which minimize the cost functional and simultaneously satisfying the state equations. For that we define some mathematical entities.

- A set of design parameters $\boldsymbol{\mu} \in U$, where U is a space named *space of the eligible controls*. In our case, $\boldsymbol{\mu}$ is the set of displacements of the control points defined in chapter 2.1.
- A *state of the system* $\gamma(\boldsymbol{\mu})$, which has to be controlled properly, is associated at a given control $\boldsymbol{\mu}$. $\gamma(\boldsymbol{\mu})$ represents the solution of a differential problem of partial derivative equations, that is in our case the Navier-Stokes equations system:

$$\Lambda\gamma(\boldsymbol{\mu}) = f, \tag{4.26}$$

where Λ is the operator which specifies the problem. This equation describes the physical system to control and to optimize it. At this system you have to add appropriate boundary conditions (section 4.1.3) and initial conditions, if the problem is evolutive. We refer to this problem as *state problem*.

- An *observation function* $z(\boldsymbol{\mu})$ which is function of $\gamma(\boldsymbol{\mu})$ through an operator C: $z(\boldsymbol{\mu}) = Cy(\boldsymbol{\mu})$. The observation can be on the boundary or at the internal of the domain or in a part of it; for that reason, C is in general a restrictional operator and it collects physical information from the system.

- A *cost functional* or *objective* $J(\boldsymbol{\mu})$ to minimize, which is defined in the space of the function observed, which we indicate with Z, such that $z(\boldsymbol{\mu}) \rightarrow \Phi(z(\boldsymbol{\mu})) \geq 0$, that is
$$J(\boldsymbol{\mu}) = \Phi(z(\boldsymbol{\mu})). \tag{4.27}$$

A general formulation of an optimal control problem is: find a function $\boldsymbol{\mu} \in U$ such that the next inequality is valid [46]:

$$J(\boldsymbol{\mu}) \leq J(\boldsymbol{\xi}) \quad \forall \boldsymbol{\xi} \in U, \tag{4.28}$$

or, in equivalent terms: find a function $\boldsymbol{\mu} \in U$ such that:

$$J(\boldsymbol{\mu}) = \inf J(\boldsymbol{\xi}) \quad \forall \boldsymbol{\xi} \in U. \tag{4.29}$$

4.5.2 Shape optimization

When the control is applied on the boundary, the problem becomes of *shape optimization*. Indeed, the shape moves fulfilling optimization criteria represented by cost functionals, by respecting also certain constraints imposed by geometrical considerations. In our case (see chapter 5) these constraints will regard the maximum accepted variation of the volume of the object in consideration and the symmetries that should be preserved on the shape of the object.

The shape is represented directly by the FFD parametrization through its lattice definition in which the object is embedded, so that the control parameters involved are the μ_i described in chapter 2. At this point, one can choose a cost functional J to minimize appropriate quantities, for example the drag over a body as it has been done in this work. Thus, the functional J depends on $\{\mu_i\}$

parameters, and its sensitivity can be determined by considering the difference quotient in correspondence with small variations of $\delta\mu_i$ of each parameter:

$$\frac{\partial J}{\partial \mu_i} = \frac{J\left(\mu_i + \delta\mu_i\right) - J\left(\mu_i\right)}{\delta\mu_i}. \tag{4.30}$$

The gradient vector $\left\{\frac{\partial J}{\partial \mu_i}\right\}$ can be used to determine a direction of improvement of the shape for the next iteration, by a variation $\delta\boldsymbol{\mu}^n$ of the parameters, such that:

$$\boldsymbol{\mu}^{n+1} = \boldsymbol{\mu}^n + \delta\boldsymbol{\mu}^n, \tag{4.31}$$

and

$$\delta\boldsymbol{\mu}^n = -\lambda_n \frac{\partial J}{\partial \boldsymbol{\mu}^n}, \tag{4.32}$$

with λ_n an appropriate relaxation parameter (see [51]). There are some efficient methods to estimate a good value for λ_n, such as the *Armjio rule*, among many other [61].

The value of the cost functional at the next iteration is given by:

$$J^{n+1} = J^n + \delta J^n = J^n + \frac{\partial J^T}{\partial \boldsymbol{\mu}^n}\delta\boldsymbol{\mu}^n = J^n - \lambda_n \frac{\partial J^T}{\partial \boldsymbol{\mu}^n}\frac{\partial J}{\partial \boldsymbol{\mu}^n}. \tag{4.33}$$

The disadvantage of this approach is to demand, in order estimate the gradient, a number of numerical solutions for the flow around the object under analysis proportional to the number of the design variables. Computational costs can become unsustainable when the number of variables is significantly incremented, that is the reason to consider reduced order modelling methods, like *reduced basis*.

Another more sophisticated approach is to impose the problem of the project as a research of a shape that can generate a desired distribution of some fluid dynamics variables which are involved directly in the problem, that is *inverse problem* setting [40]. This approach needs only one numerical simulation, and not a number of simulations equal to the number of design variables. On the other hand, it is not sure that the shape optimized obtained is physically reachable, so the problem must be formulated very carefully.

A one possible automatic and versatile approach can be the Jameson's approach, which recurs to the formulation of an *adjoint problem* for the computation of the cost functional derivative [41]. The adjoint problem is made by a procedure of constrained minimization (using the state equations) which use the

Lagrange multiplier method [38]. For a flow around a bulb or a rudder, the cost functional would depend on the fluid dynamics variables involved $\mathcal{W}$ and on the physical position of the contour $\mathcal{F}$,

$$J = J\left(\mathcal{W}, \mathcal{F}\right), \tag{4.34}$$

and to every variation of $\mathcal{F}$, corresponds a variation for the functional

$$\delta J = \frac{\partial J}{\partial \mathcal{W}}^T \delta \mathcal{W} + \frac{\partial J}{\partial \mathcal{F}}^T \delta \mathcal{F}. \tag{4.35}$$

By using the control theory, the equations governing the flow field are introduced as constraints, such that the final expression of the gradient does not require the re-evaluation of the flow field. In this way $\delta\mathcal{W}$ must be eliminated from (4.35). We suppose that the governing equations (in our case the Navier-Stokes equations) are expressed on the domain by the following expression:

$$\mathcal{R}\left(\mathcal{W}, \mathcal{F}\right) = 0. \tag{4.36}$$

The quantity $\delta\mathcal{W}$ is determined by

$$\delta\mathcal{R} = \left[\frac{\partial \mathcal{R}}{\partial \mathcal{W}}\right] \delta\mathcal{W} + \left[\frac{\partial \mathcal{R}}{\partial \mathcal{F}}\right] \delta\mathcal{F} = 0. \tag{4.37}$$

By introducing the Lagrange multiplier ψ we have

$$\delta J = \frac{\partial J}{\partial \mathcal{W}}^T \delta\mathcal{W} + \frac{\partial J}{\partial \mathcal{F}}^T \delta\mathcal{F} - \psi^T \left(\left[\frac{\partial \mathcal{R}}{\partial \mathcal{W}}\right] \delta\mathcal{W} + \left[\frac{\partial \mathcal{R}}{\partial \mathcal{F}}\right] \delta\mathcal{F} \right). \tag{4.38}$$

By collecting the terms in $\delta\mathcal{W}$ and in $\delta\mathcal{F}$ we have

$$\delta J = \left(\frac{\partial J^T}{\partial \mathcal{W}} - \psi^T \left[\frac{\partial \mathcal{R}}{\partial \mathcal{W}}\right] \right) \delta\mathcal{W} + \left(\frac{\partial J}{\partial \mathcal{F}}^T - \psi^T \left[\frac{\partial \mathcal{R}}{\partial \mathcal{F}}\right] \right) \delta\mathcal{F}. \tag{4.39}$$

We choose ψ to satisfy the next adjoint equation

$$\frac{\partial J}{\partial \mathcal{W}} = \psi \left[\frac{\partial \mathcal{R}}{\partial \mathcal{W}}\right]^T, \tag{4.40}$$

and we find that

$$\delta J = \mathcal{G}\delta\mathcal{F}, \tag{4.41}$$

where

$$\mathcal{G} = \frac{\partial J}{\partial \mathcal{F}}^T - \psi^T \left[\frac{\partial \mathcal{R}}{\partial \mathcal{F}}\right]. \tag{4.42}$$

The equation (4.41) is independent from $\delta\mathcal{W}$ and so the gradient of J can be computed without the use of further numerical simulations of the flow around the body. If $\mathcal{R}(\mathcal{W}, \mathcal{F})$ is a system of PDEs, also the adjoint problem (4.40) is made up by PDEs.

4.5.3 Optimization algorithm

The next step is the description of the iterative optimization algorithm. In literature many optimization methods have been proposed [42, 51]. Among them, the most common ones are the gradient-like method [13], genetic algorithms [19] and neural networks [17]. Gradient-like methods require the gradient of the scalar cost function and constraints (dependent variables), respecting the shape design (independent) variables. The problem of these kinds of methods is that they may converge to local optimum and not to the global one. Besides, Genetic Algorithms (GAs) have proven their strength against local limits, however they may require a very high number of parameters evaluation to converge. In our case, the cost of GA methods is still not affordable, thus we will focus on the first, preferring also a deterministic approach to the problem.

In this work we exploited the interface between MATLAB and COMSOL Multiphysics, so it has been possible to implement the FFD using symbolic expressions, to solve the problem with COMSOL Multiphysics and in order to reiterate. The built-in MATLAB function *fmincon* comes to our purpose [6]. It is a function based on gradient-like method, which finds a constrained minimum of a scalar function of several variables, starting from an initial estimate.

For a smooth constrained problem, let g and h be vector functions representing all inequality and equality constraints respectively. The definition of the optimization process can be written as

$$\begin{aligned} &\min_{\boldsymbol{\mu}} J(\boldsymbol{\mu}), \\ \text{subject to } &g(\boldsymbol{\mu}) \leq 0, \\ &h(\boldsymbol{\mu}) = 0, \end{aligned} \tag{4.43}$$

where $\boldsymbol{\mu}$ is the set of FFD parameters defined in chapter 2, the design variables which are correlated with the control points displacement. A priori, one has to

decide which parameters to choose and in which direction they should move, and this is absolutely arbitrary.

Then, we define the Lagrangian function as

$$L(\boldsymbol{\mu}, \boldsymbol{\lambda}) = J(\boldsymbol{\mu}) + \sum \lambda_{g,i} g_i(\boldsymbol{\mu}) + \sum \lambda_{h,i} h_i(\boldsymbol{\mu}), \tag{4.44}$$

where $\boldsymbol{\lambda}$, which is the concatenation of $\boldsymbol{\lambda}_g$ and $\boldsymbol{\lambda}_h$, is the Lagrange multiplier vector. Its length is the total number of constraints. The Hessian of this function is shown below

$$W = \nabla^2_{\mu\mu} L(\boldsymbol{\mu}, \boldsymbol{\lambda}) = \nabla^2 J(\boldsymbol{\mu}) + \sum \lambda_{g,i} \nabla^2 g_i(\boldsymbol{\mu}) + \sum \lambda_{h,i} \nabla^2 h_i(\boldsymbol{\mu}), \tag{4.45}$$

where $\nabla^2_{\mu\mu}$ is the Laplacian in respect to vector $\boldsymbol{\mu}$. The function *fmincon* uses a *Sequential Quadratic Programming* (SQP) method, that is one of the most popular and robust algorithms for nonlinear continuous optimization [53], and it is appropriate for small or large problems. The method solves a series of subproblems designed to minimize a quadratic model of the objective function using a linearization of the constraints [36]. A non-linear program in which the objective function is quadratic and the constraints are linear is called a Quadratic Program (QP). An SOP method solves a QP at each iteration. In particular, if the problem is unconstrained, then the method reduces to *Newton's method* [61] to find a point where the gradient of the objective vanishes. If the problem has only equality constraints, then the method is equivalent to applying *Newton's method* to the first-order optimality conditions (or *Karush-Kuhn-Tucker* (KKT) conditions [53]) of the problem.

In order to define the k-th subproblem, both the inequality and equality constraints have to be linearized. If $\mathbf{p} = \boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k$, we obtain the local subproblem

$$\begin{aligned} \min \frac{1}{2} \mathbf{p}^T W_k \mathbf{p} + \nabla J_k^T \mathbf{p}, \\ \text{subject to } \nabla h_i(\boldsymbol{\mu}_k)^T \mathbf{p} + h_i(\boldsymbol{\mu}_k) = 0, \\ \nabla g_i(\boldsymbol{\mu}_k)^T \mathbf{p} + g_i(\boldsymbol{\mu}_k) \geq 0, \end{aligned} \tag{4.46}$$

A QP method is now used to solve this problem [53].

4.5.4 Cost functionals

Similarly to what have been done in previous works [16, 30, 31], we have chosen to minimize the following functionals:

$$J_b(\boldsymbol{\mu}) = \frac{D(\boldsymbol{\mu})}{D_0}, \tag{4.47}$$

$$J_r(\boldsymbol{\mu}) = \frac{1}{2}\left(\frac{D(\boldsymbol{\mu})}{D_0} + \frac{E_0}{E(\boldsymbol{\mu})}\right), \tag{4.48}$$

where J_b is the functional for the bulb, J_r is the functional for the rudder, D is the drag force that comes from the solution of the Navier-Stokes equations, $E = L/D$ is the efficiency where L is the lift of the rudder, and D_0 and E_0 are reference quantities (generally the values of the first step). D and L are obtained by making a pressure integration on the rudder surface (for the bulb it is done only for the drag) in a streamwise and spanwise direction, respectively.

In conclusion, all the operations involved in the optimization process are summarized in figure 4.9. Starting from the definition of the model problem and the design variables, through the choice of the cost functional and the iterations of the optimization process, till the final optimized shape.

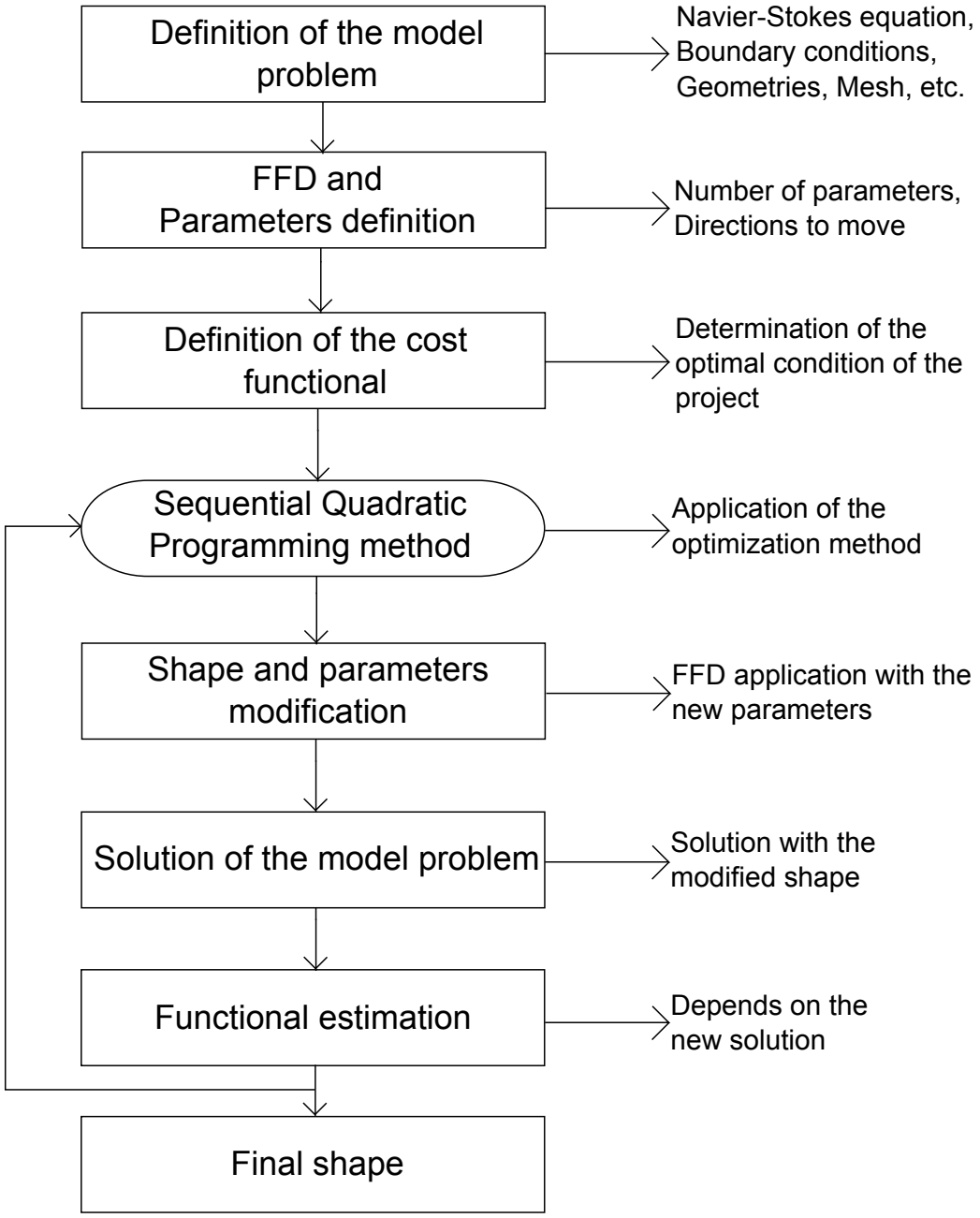

Figure 4.9: Scheme for the shape optimization process.

One last thing to note is that in all this procedure, there is no need of a

block to remesh after deformation. This is due to the fact that the deformation applied from the FFD techniques involves not only the geometry but also the mesh itself, and by remaining in small deformation fields, the mesh continues to maintain its validity.

Chapter 5

Simulations and results

In this chapter numerical results of the simulations carried out in this work are presented in section 5.1 and section 5.2 for the bulbs and the rudder, respectively. A local FFD has been applied on a subdomain in order to increase the deformation control and to reduce the parameters number. Besides, in the case of the rudder, it has been necessary to rotate the local FFD in order to maintain the spanwise symmetry of the mean airfoil. Constraints have been imposed on the displacements of the control points. Simulations have been done for a shape optimization process: at every step, the Navier-Stokes equations are solved (see chapter 4) and FFD is applied to minimize the specified cost functionals.

5.1 The bulb

First we present the results concerning the bulb. In the simulation, the cost functional described in section 4.5 has been used. A local FFD has been applied (see fig. 5.1, 5.2, 5.3). The total number of control points is 343 ($L = 6$, $M = 6$ and $N = 6$, referred to x, y, and z direction respectively), and the ones involved in the simulation are 12, with the use of 20 parameters.

Besides, the following constraints have been imposed:

- Concerning the possibility of the volume V to vary, we allow a variation of 20% of its initial value. This is to avoid the most obvious condition of minimum resistance, when the bulb degenerates to a point in the space.
- In order to maintain the symmetry along the z direction additional constraints have been imposed to displacements of control points indicated

as A and B in figure 5.2 and 5.3, such that $\mu_A = -\mu_B$. Instead, no constraints to the control points have been imposed to maintain the symmetry along the y direction, to test if the result still remain symmetric without imposing the symmetry.

- The parameters vary between $[-2, 2]$, to maintain the deformation contained and to avoid mesh problems.

In figures 5.2 and 5.3, the displacementes of the control points chosen as parameters for the optimization are shown.

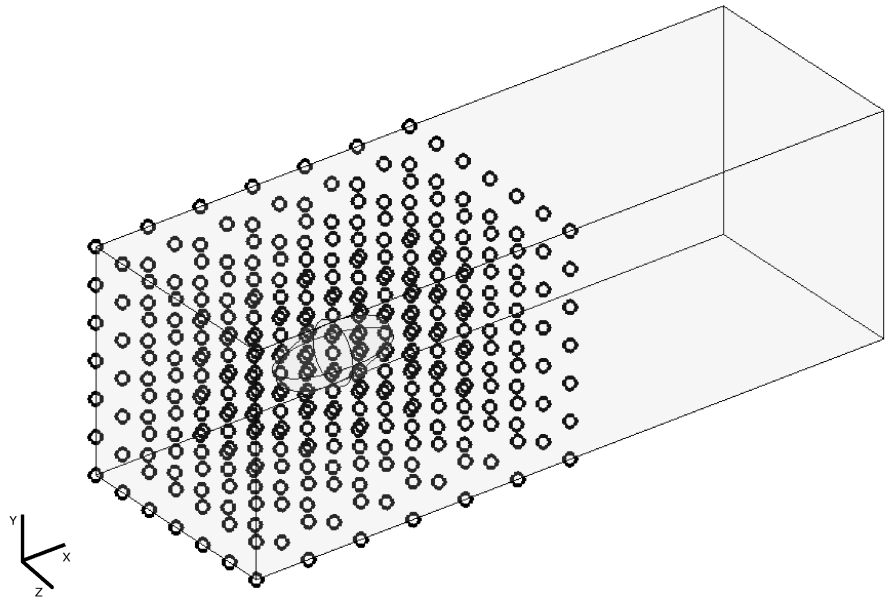

Figure 5.1: Domain where the bulb is inserted and definition of the "local" FFD.

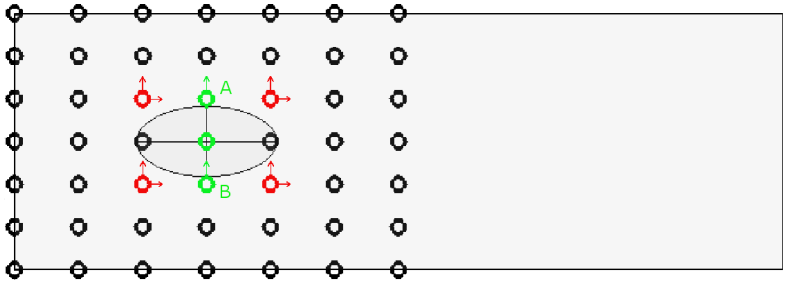

Figure 5.2: Lateral view of the domain and the local FFD.

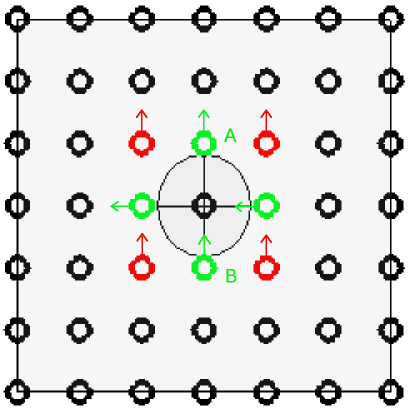

Figure 5.3: Frontal view of the domain and the local FFD.

Results after the optimization process follow. The boundary conditions are the ones exposed in section 4.1.3. For a better view of the results, we present just the flow field belonging to the XY plane, being a symmetric flow. In figures 5.4 and 5.5 the initial velocity and pressure field around the undeformed bulb are shown, while in figures 5.6 and 5.7 we report the results after the shape optimization, for the velocity and pressure field.

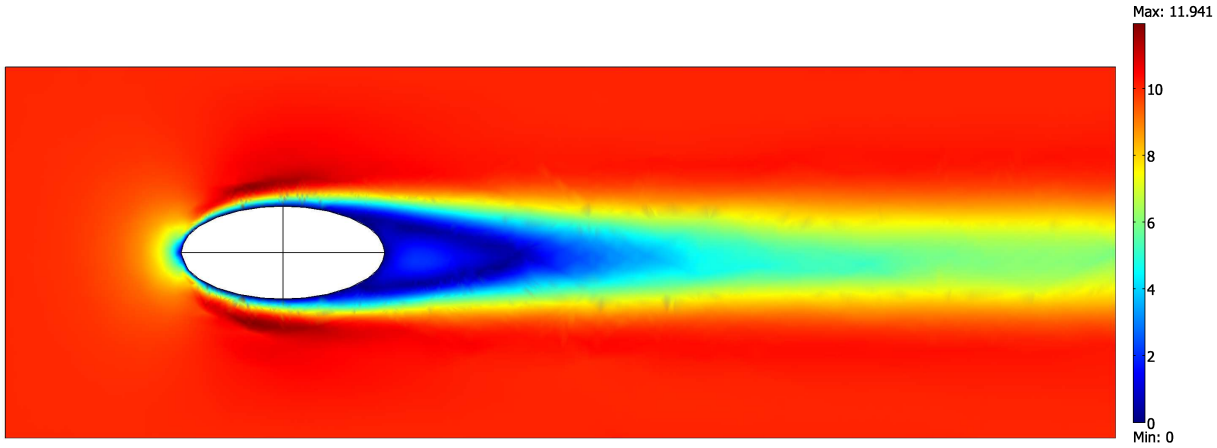

Figure 5.4: The initial shape of the bulb and the velocity in plane XY [m/s].

Figure 5.5: Pressure field of the initial shape of the bulb in plane XY [Pa].

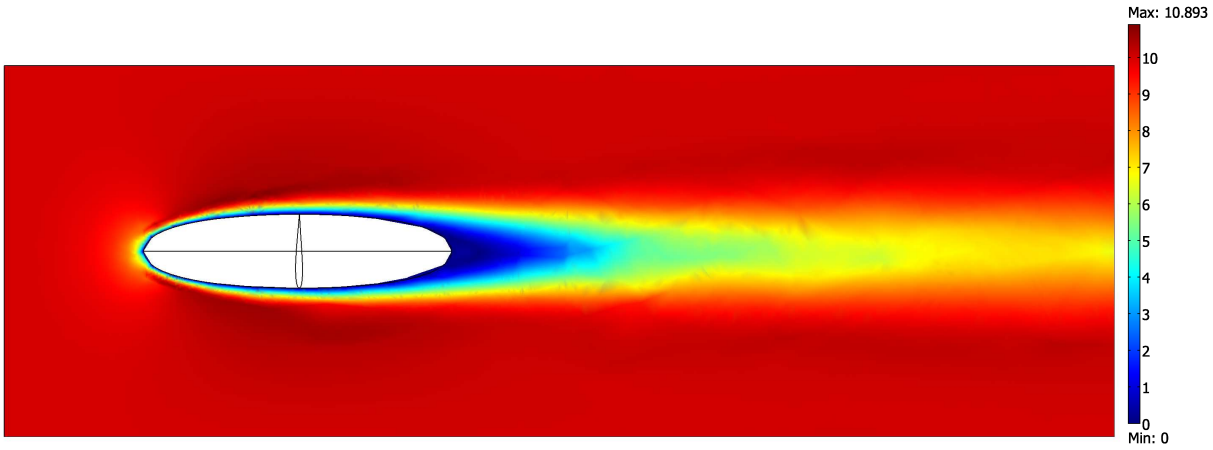

Figure 5.6: XY plane showing the simulation of the velocity field around the optimized bulb [m/s].

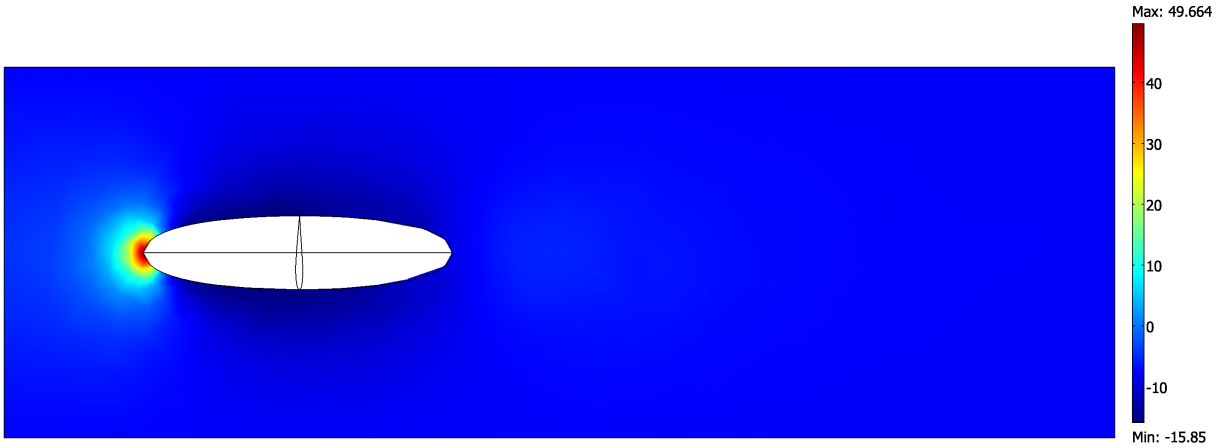

Figure 5.7: Pressure field around the optimized bulb in plane XY [Pa].

As it can be seen from the previous figures, the optimization headed to reduce the wake past the bulb. In fact, the drag force D is composed by two contributions: one given by skin friction force and the other given by the pressure force [15]. Depending on the shape of the object, and of course on the Reynolds number as well, one contribution becomes more important than the other one. In the case of a bulb, or a blunt body, and in presence of a sufficient high Re, as in our case the major contribution derives from the pressure force. So by

trying to contain this contribution, the frontal area of the bulb is reduced, as we expected, and it becomes more and more similar to an airfoil, where the skin friction drag, or the viscous one, is predominant.

In table 5.1 the values of the parameters and of the cost functional J obtained as results, which are defined in section 4.5.4, are reported. To recall, D_0, V_0 and J_0 are the drag, the volume and the cost functional, respectively, referred to the undeformed bulb, while D, V and J are the ones obtained at the end of the optimization. The percentual gain of drag decrease is denoted with $\%\Delta$.

Table 5.1: Value obtained before and after the shape optimization for the bulb.

D_0 [N]	D [N]	V_0 [m^3]	V [m^3]	J_0	J	$\%\Delta$
1.389	1.005	0.7156	0.5725	1	0.724	27.6

As it can be observed, there is an important reduction of the drag with the new shape, a gain of 27.6% respect to the initial shape. The final volume is the 80% of the initial one, and this indicates that the optimization has stopped because it has reached the limit of volume reduction. The FFD parameters values obtained at the end of the optimization are indicated here below.

Table 5.2: FFD parameters values obtained after the shape optimization process of the bulb.

μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8	μ_9	μ_{10}
-1.926	0.051	2.000	0.418	1.331	-1.176	0.920	2.000	0.529	1.278

μ_{11}	μ_{12}	μ_{13}	μ_{14}	μ_{15}	μ_{16}	μ_{17}	μ_{18}	μ_{19}	μ_{20}
-1.278	-0.783	-0.082	2.000	-0.329	-1.006	-1.499	-1.051	2.000	-0.651

Another test that has been conducted, to verify if the optimization is going in the right direction, is to consider the opposite situation, that is to consider a very low Reynolds number. With Re = 1, the main contribution to the drag force becomes the skin friction force, so the total wet surface should decrease. In figure 5.8 the final optimized shape of the bulb is shown.

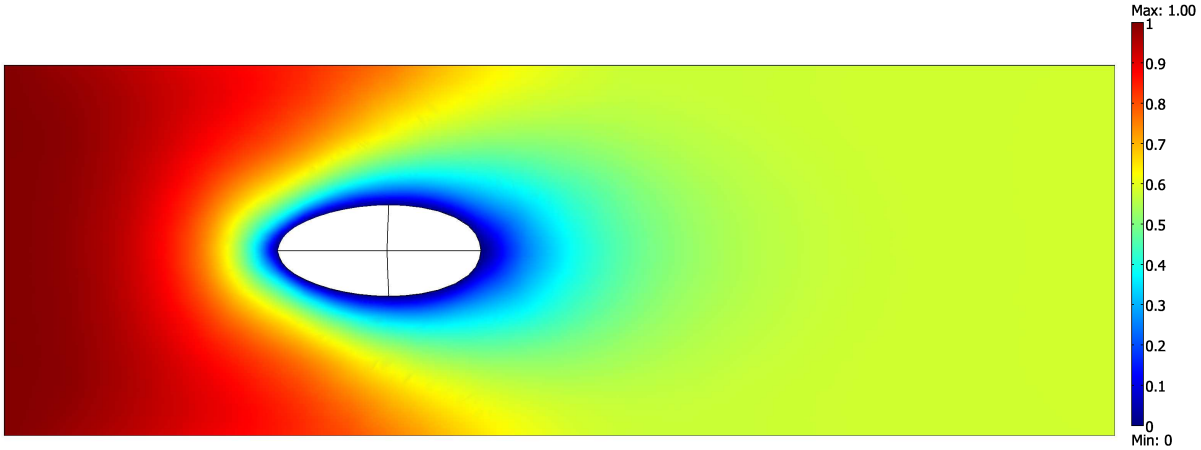

Figure 5.8: Velocity field with low Re number around an optimized bulb [m/s].

What is interesting, and predicted by the theory, is that the bulb gets smaller, decreasing its total wet surface in order to reduce the skin friction drag. This is the opposite result compared with the former and it is a confirmation that the optimization process is working properly.

5.2 The rudder

We now show the results for the rudder. The simulations have been more difficult, due to the fact that the rudder has a more complex and refined geometry. It has also been rotated by an angle of $\alpha = 5°$ with respect to the flow field around the z axis. Also in this case, a local FFD has been applied, as shown in figures 5.9, 5.10 and 5.11.

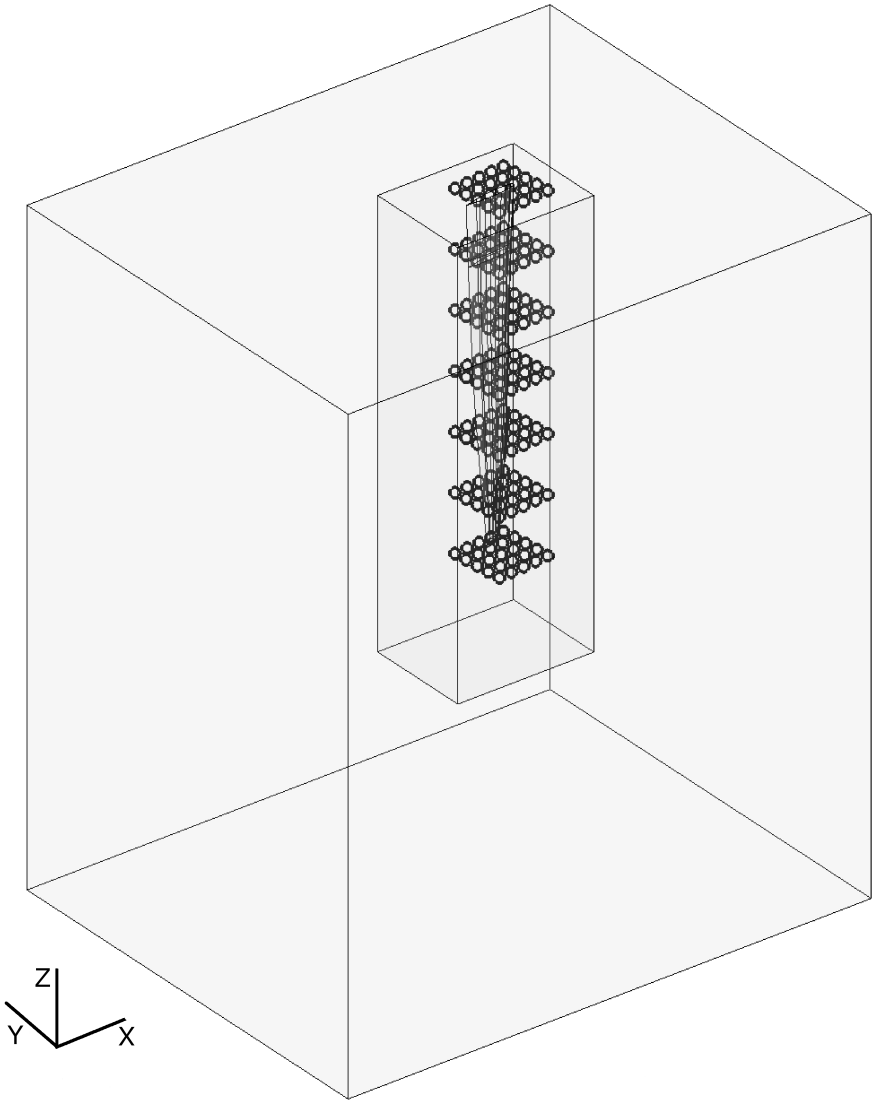

Figure 5.9: The rudder and the subdomains where the rudder is placed, with the lattice of the rotated local FFD.

As we anticipated in section 3.1, the mean airfoil is a NACA 63012, which is a symmetric one. To maintain this symmetry, a constraint on the displacements of the parameters has been imposed, such that all the displacements are proportionally connected along the z axis. Thus the constraints imposed are:

- As in the case of the bulb, the volume of the rudder cannot diminish more

than 20% of its initial volume.

- $\mu_{i_{zk}} = \frac{k}{5}\mu_{i_{z5}}$, where i is the i-th parameter, k is the layer of each plane XY formed by the control points considered along the z axis. In this case we have considered 5 layers, so k can vary from 1 to 5, where 5 is the highest XY plane of control points taken into consideration (see figure 5.11). So the displacements at the inferior layers will be proportional to the highest one.
- At every level k, it has been imposed that $\mu_A = -\mu_B$ (figure 5.10) to maintain the symmetry in the plane of the airfoil.
- Deformations are more delicate, so the range of the parameters is restricted to $[-1, 1]$.

As the rudder is rotated, it has been necessary to rotate also the local FFD, in order to respect the symmetry condition, as shown in figure 5.10.

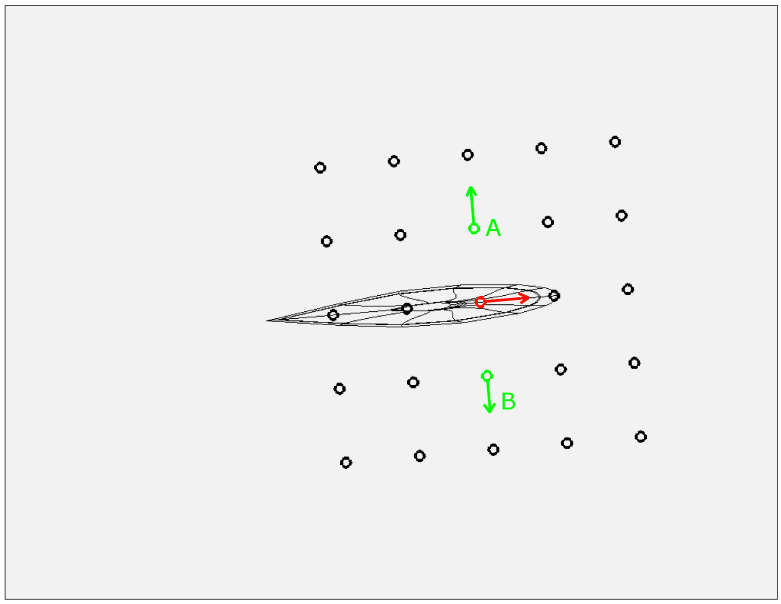

Figure 5.10: An upside view of the rudder in the XY plane and the displacements considered.

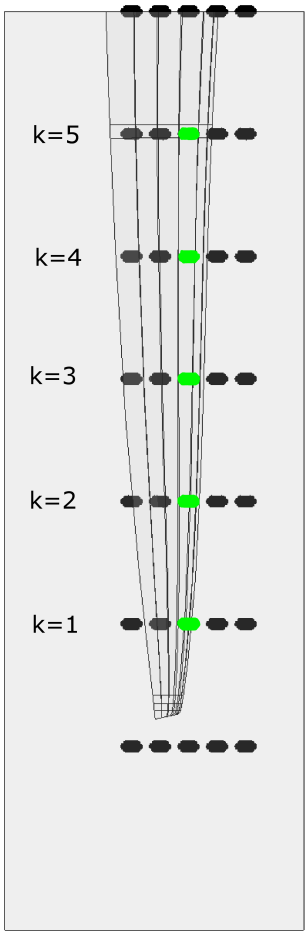

Figure 5.11: A lateral vision of the rudder in XZ plane and the displacements considered.

The lattice has 175 control points ($L = 4$, $M = 4$ and $N = 6$, referred to x, y, and z direction respectively), and the ones chosen for the optimization and the parameters are 15, which are all indicated in figures 5.10 and 5.11. In the next figures the initial shape of the rudder is shown from two different planes, the XZ and the XY, first for the velocity field (figures 5.12, 5.13, 5.14) and then for the pressure field (figures 5.15, 5.16, 5.17). The results correspond to the situation in the center of the domain, that is one is taken at $y/2$, the others at $z/6$ and $z/3$ respectively.

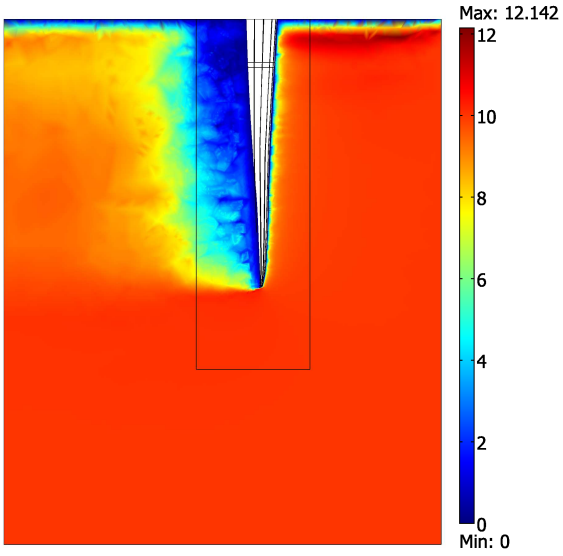

Figure 5.12: The initial shape of the rudder and the velocity flow field around it in the plane XZ at $y/2$ [m/s].

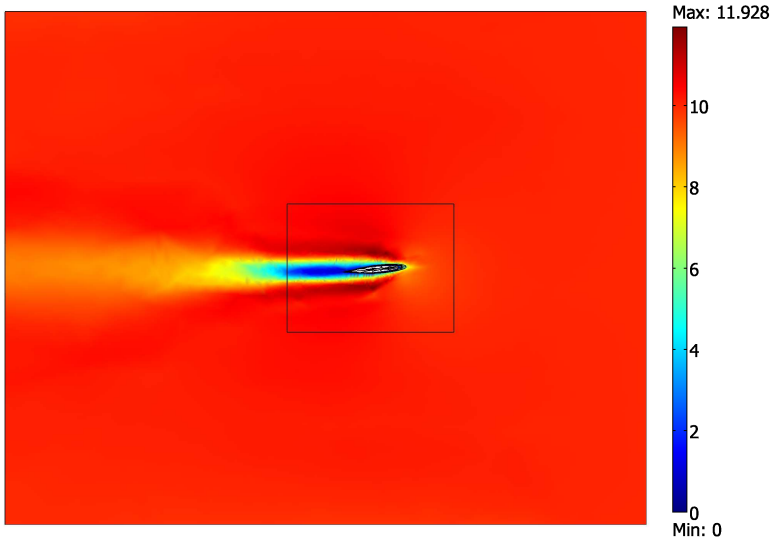

Figure 5.13: The initial shape of the rudder and the velocity flow field around it in the plane XY at $z/6$ [m/s].

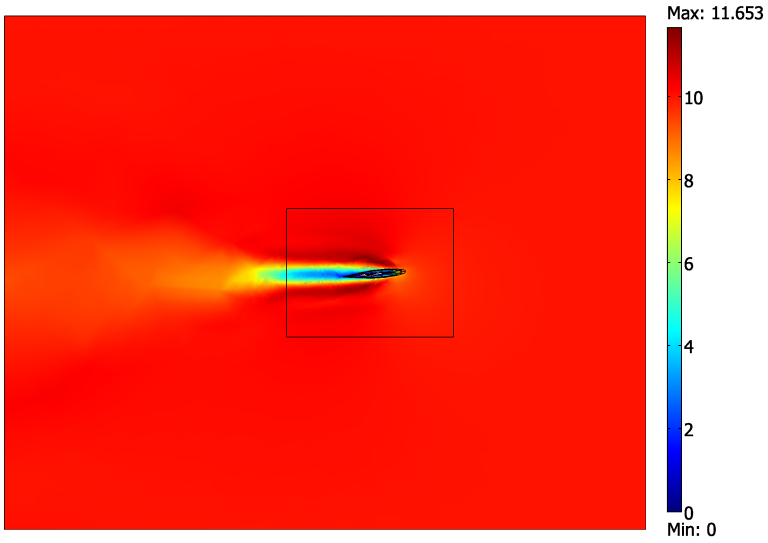

Figure 5.14: The initial shape of the rudder and the velocity flow field around it in the plane XY at $z/3$ [m/s].

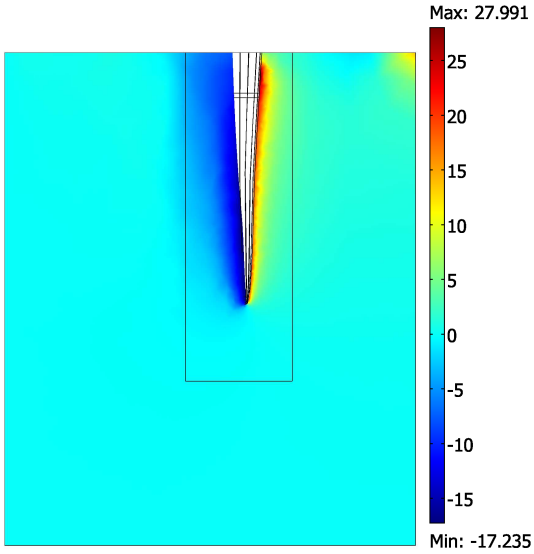

Figure 5.15: Pressure field of the initial shape of the rudder in plane XZ at $y/2$ [Pa].

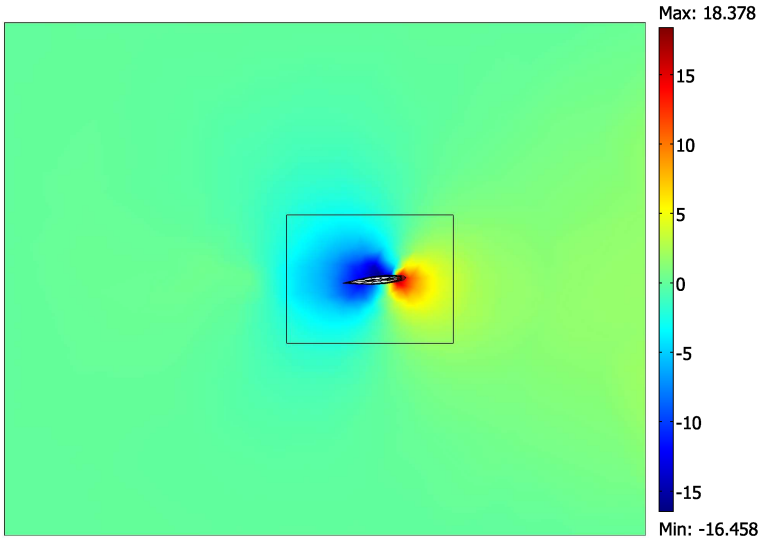

Figure 5.16: Pressure field of the initial shape of the rudder in plane XY at $z/6$ [Pa].

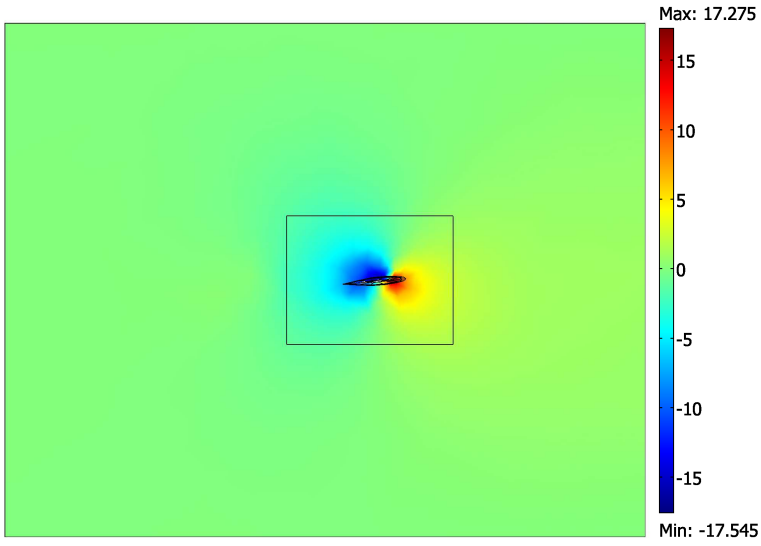

Figure 5.17: Pressure field of the initial shape of the rudder in plane XY at $z/3$ [Pa].

In figures 5.18, 5.19 and 5.20 the results for the velocity field after the opti-

mization are shown, always in the same planes as the previous figures, instead in figures 5.21, 5.22 and 5.23 the results for the pressure field are shown.

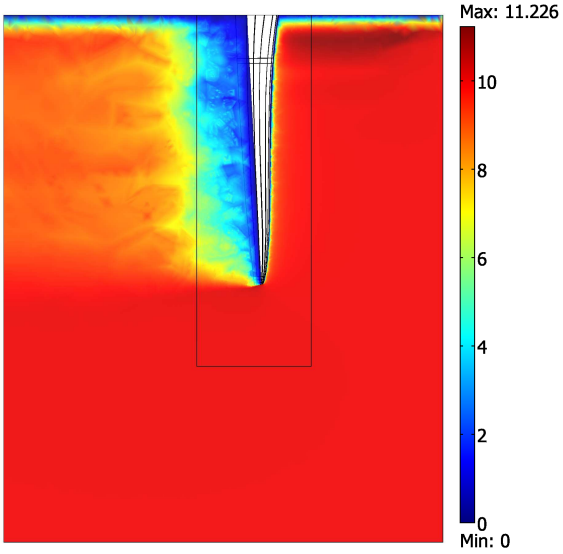

Figure 5.18: XZ plane showing the velocity field around the optimized rudder at $y/2$ [m/s].

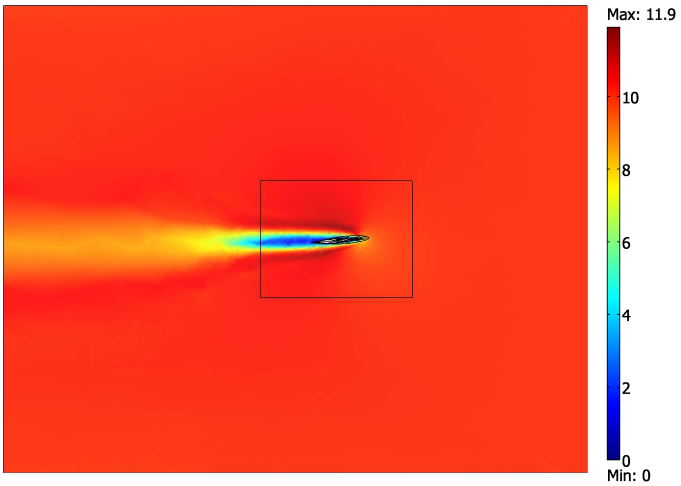

Figure 5.19: XY plane showing the velocity field around the optimized rudder at $z/6$ [m/s].

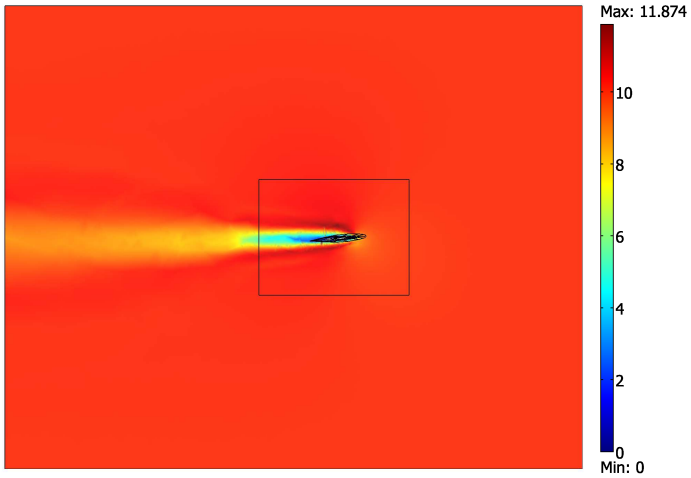

Figure 5.20: XY plane showing the velocity field around the optimized rudder at $z/3$ [m/s].

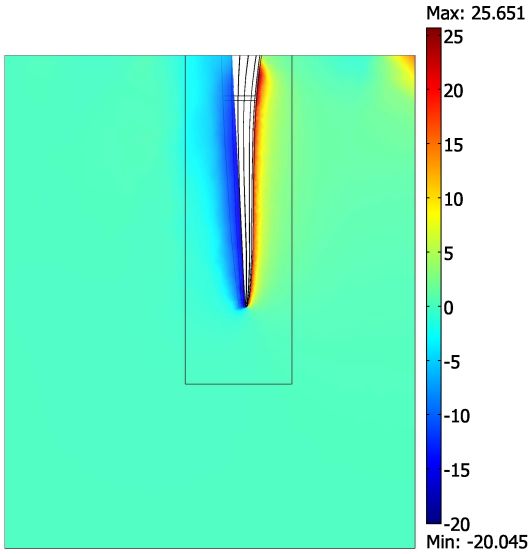

Figure 5.21: XZ plane showing the pressure field around the optimized rudder at $y/2$ [Pa].

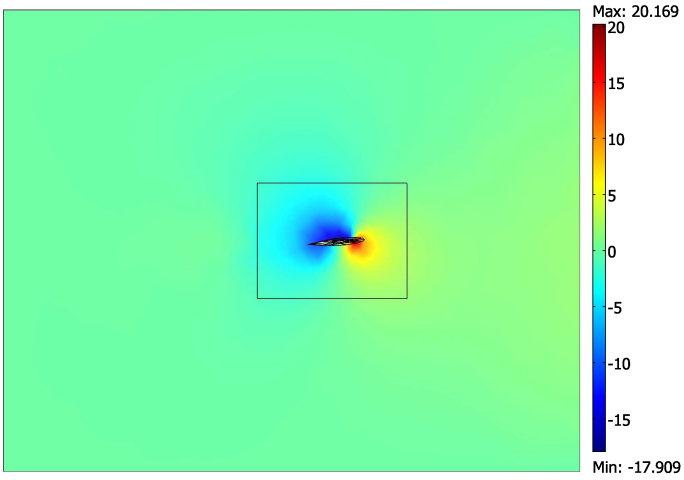

Figure 5.22: XY plane showing the pressure field around the optimized rudder at $z/6$ [Pa].

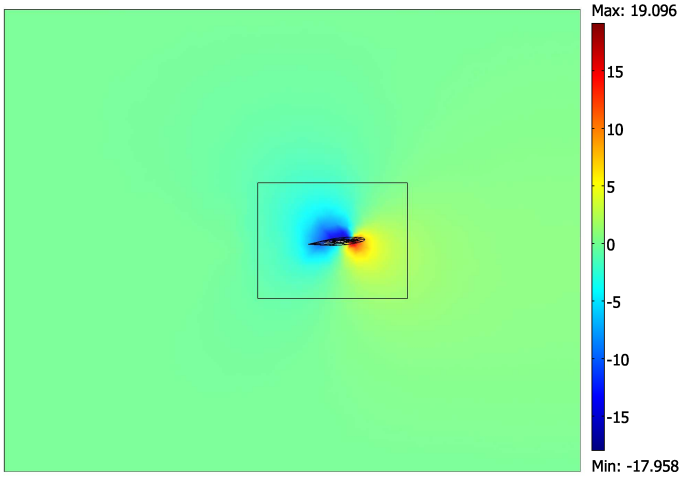

Figure 5.23: XY plane showing the pressure field around the optimized rudder at $z/3$ [Pa].

The shape is modified, however the deformation is not large enough to appreciate the entity of the variations. In figure 5.24 we show an amplification of twice the deformations obtained, just to give an idea of their magnitude.

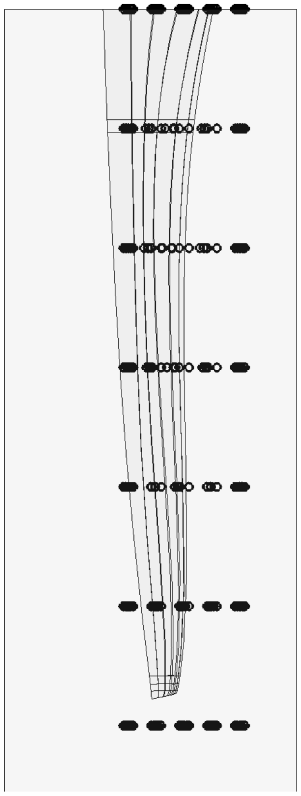

Figure 5.24: XZ plane with the amplified deformed rudder.

The values of the physical variables, which have been defined in section 4.5.3, are shown in table 5.3. D_0, E_0, V_0 and J_0 are the variables referred to the initial shape of the rudder, then D, E, V and J are the ones referred to the final optimized shape, which are the drag, the efficiency and the volume respectively (see section 4.5.4). $\%\Delta_{\mathrm{tot}}$ is the relative gain obtained with the new shape, which includes the contributions of the combination of drag D and efficiency E.

Table 5.3: Value obtained before and after the shape optimization for the rudder.

D_0 [N]	D [N]	E_0	E	V_0 [m^3]	V [m^3]	J_0	J	$\%\Delta_{\mathrm{tot}}$
2.993	2.915	2.559	3.7464	0.0298	0.0207	1	0.829	17.1

In this case the optimization has stopped after reaching the maximum range value of the parameters of the highest layer ($k = 5$). But even if the deformation

involved on the final form of the rudder is small, the wake after it looks smaller than the original one and there is a total gain of 17.1%, which corresponds to a 2.6% gain in drag and a 31.7% gain in efficiency E, maintaining the construction symmetries. The parameters values obtained after the shape optimization are indicated in table 5.4.

Table 5.4: FFD parameters values obtained after the shape optimization process of the rudder.

μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7
0.2	0.2	-0.2	0.4	-0.4	-0.4	0.6

μ_8	μ_9	μ_{10}	μ_{11}	μ_{12}	μ_{13}	μ_{14}	μ_{15}
-0.6	-0.6	0.8	-0.8	-0.8	1	-1	-1

In order to sum up, we can conclude that FFD versatility suits well the optimization process, adapting without problems to the different kinds of geometries and constraints, improving the performance of the object taken into consideration. The major cost of the computational process is the solution of the Navier-Stokes equations, and it can become very high by the use of more FFD parameters, since for the optimization process described in section 4.5.3 they are solved as many times as the parameters number. The use of some reduced order modelling techniques, such as reduced basis method [44, 67, 69], can help diminishing the computational cost of the solution of the Navier-Stokes equations.

However, the test cases considered can be a good starting point for a shape optimization design process of a generic CAD object, adapting the model equations and the quantities present in the cost functional, which can be weighted differently in function of the optimization that one wants to obtain, to the case taken into consideration.

Chapter 6

Conclusions

The FFD has proved to be a powerful and efficient parametrization method which could be used in several applications, such as the shape optimization of a wing or an airfoil or a bypass conduct or a part of a sailing boat as well. In this work, FFD has been tested on 3D cases (all the properties owned by the 2D FFD have been tested and are still valid). We have considered two shape optimization processes dealing with a bulb and a rudder on a yacht. It has been applied locally to just one part of the domain in order to have a better control of the deformation around the object and not in the part of the domain that is not taken into consideration. Moreover, regarding the rudder, FFD has also been rotated/distorted in order to maintain the symmetry constraint of the deformation along its spanwise direction.

The FFD method has been tested on various shapes (see chapter 2) and as a result, it fits very well to every kind of form, going from the easiest to the most complex one, permitting the deformation of the global domain (and local domain too, if the FFD is made locally). One aspect that could be tested in order to improve the control of the deformation is to subdivide the domain into two or more FFD settings defined only locally, so that we may have two or more different deformations sets/regions.

A strong point of the FFD method is that the deformation involves also the mesh defined in the inside of the lattice of points (bounding box), and for small and smooth deformation, there is no particular need to make a new mesh at each iteration of the shape optimization problem. Thus FFD has demonstrated to be independent of the mesh, geometry and even the PDE model to which it is applied. FFD is also a technique suitable to be used in multiple contexts, e. g. for aeronautical problems, in the case of the preliminary design phase of a

complete aircraft, that is a multidisciplinary shape optimization problem.

These features make FFD a very flexible and efficient method. The results presented in chapter 5 show that, applying the Navier-Stokes equations to solve the flow in a cruising condition, a new optimized shape is obtained both for the bulb and for the rudder, with an improvement of the fluid dynamics performances and indexes related with state variables chosen properly to minimize/maximize, which is drag and a combination of drag and efficiency, respectively, obtaining a percentual gain of 27.6% drag reduction for the bulb and a 17.1% of improvement of the combination of drag and efficiency for the rudder.

The choice of the degree of freedom of the admissible deformations and the number of the parameters are all up to the user, after carrying out proper investigations. One aspect that can make the object of further investigation is the setup of a method that, by identifying where the shapes need to be deformed, allows the choice of control points to improve the shape optimization with the least number and maximize the efficiency of the deformation. This could reduce even more the computational costs needed for the shape optimization process, which is determined by the number of chosen variables. To reach our goal we have developed a platform by combining several capabilities already available in order to combine different tools for geometrical modelling, shape variation management, mathematical and numerical modelling and then simulation and optimization.

Moreover, there are many other versatile aspects that could be dealt with in a future work. The main topic could be the use of parallel computing, which can permit to solve a more realistic problem with larger Re numbers and with the implementation of appropriate turbulence models. A better improvement of the computational capacity permits also to perform mesh, refinement and adaptivity.

Anyway, the cost of the optimization by solving Navier-Stokes equations discretized by finite elements at every iteration and for every design variables may become prohibitive. In this perspective, an aspect of great interest could be to couple FFD method with reduced order modelling techniques, such as the reduced basis method [44, 67, 69]. This will simplify the complexity problem and gain even more in efficiency of computational performance. An alternative geometrical parametrization method called *Radial Basis Function* (*RBF*) could be taken into consideration and compared with the FFD techniques [34, 52, 57]. This would permit to localize the deformation and not to involve the whole domain where FFD is defined (globally or locally), but having the control of a

single displacement of a control point.

Another aspect of investigation can be the comparison of FFD based on different parametrization polynomials: in this work we have chosen to operate with FFD based on Bezier curves, but FFD can be based also on B-Spline or NURBS curves [33, 79]. It would be interesting to see the difference between the results obtained with different parametrizations.

Last but not least, as we said, a drawback of the FFD is the lack of the physical meaning associated to the displacements of its parameters, which is neither physical nor correlated with any unit length. In the aerodynamics case, an alternative parametrization named *MASSOUD* (*Multidisciplinary Aero/Struc Shaper Optimization Using Deformation*), a sort of evolution of the FFD, has been proposed by Samareh in [73], which modifies the FFD method in order to avoid this problem by parameterizing the shape perturbations rather than the geometry itself. This can be a starting point in order to make FFD more general, adding more physics to the optimization process.

Shape optimization problems have met an increasing interest in many industrial problems to optimize/improve productions, processes and performances. Aeronautics has always provided a very interesting ground for these kind of problems, but also applications in hydrodynamics and life sciences are growing thanks to availability of powerful tools to face heavy computational loads.

Appendix A

STL format problem

In chapter 3 we have mentioned that there are some problems with STL file format. Here is an example to better show what we mean.

A geometrical object has been created which could be a part of a more complex system. The object is shown in the figures A.1, A.2, A.3, A.4:

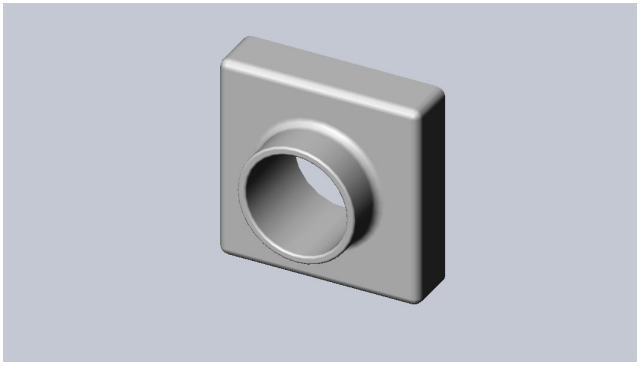

Figure A.1: Example of a tridimensional object created in SOLIDWORKS.

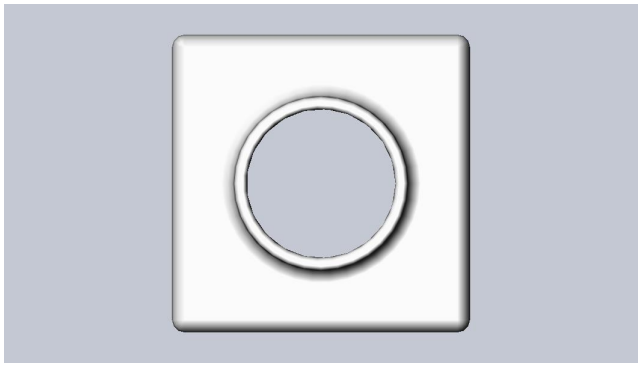

Figure A.2: Frontal vision of a tridimensional object created in SOLIDWORKS.

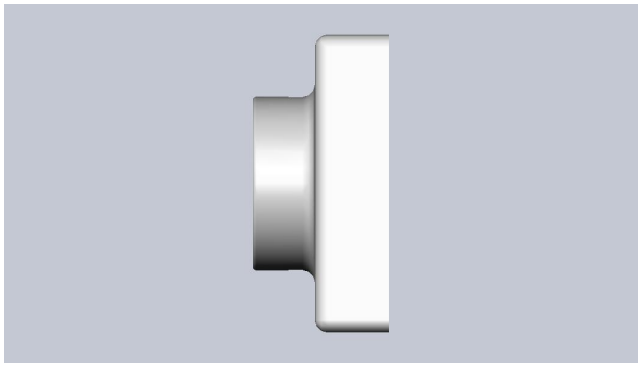

Figure A.3: Lateral vision a tridimensional object created in SOLIDWORKS.

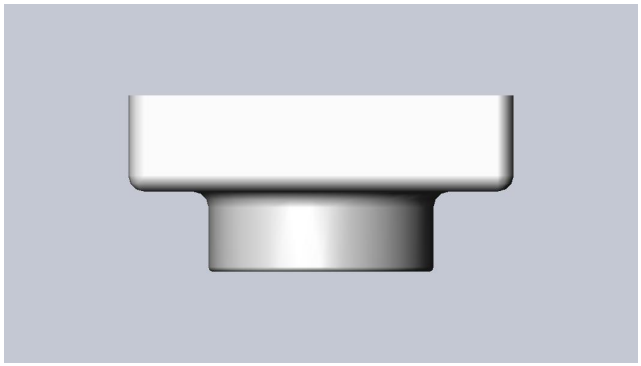

Figure A.4: Upside vision of a tridimensional object created in SOLIDWORKS.

The geometry has been saved as STL file and imported to MATLAB, where FFD (see chapter 2) could have been applied to. The test has been done by moving an external control point, stretching the geometry in order to see the effect of a significant deformation. After having applied the FFD to the object, the result, which is presented in the next figures, have been saved as an STL file and reimported in SOLIDWORKS:

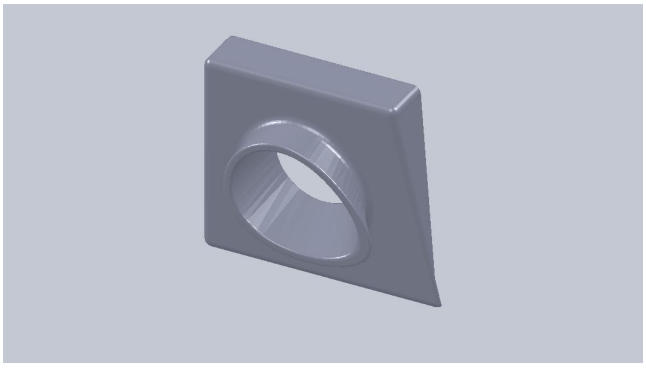

Figure A.5: Deformed tridimensional object imported in SOLIDWORKS.

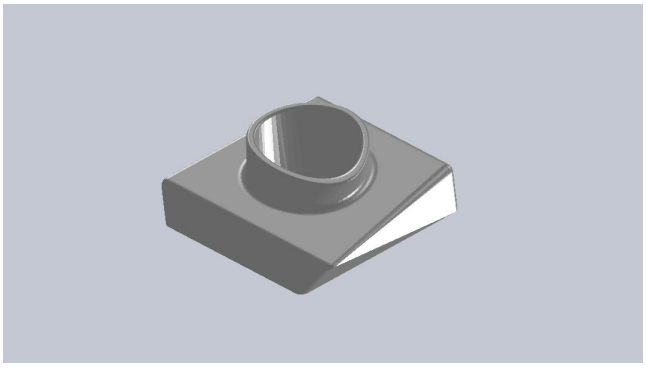

Figure A.6: Lateral vision of a deformed tridimensional object imported in SOLIDWORKS.

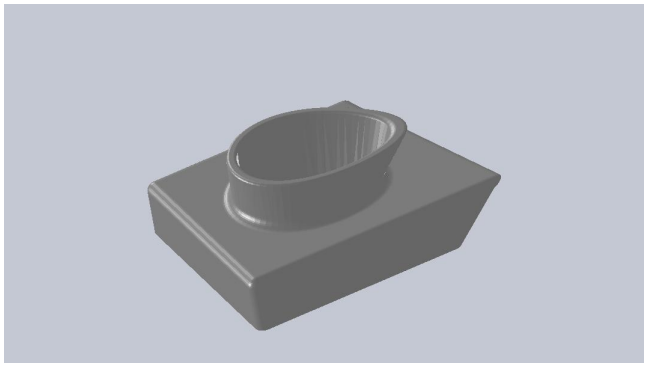

Figure A.7: Another lateral vision a deformed tridimensional object imported in SOLIDWORKS.

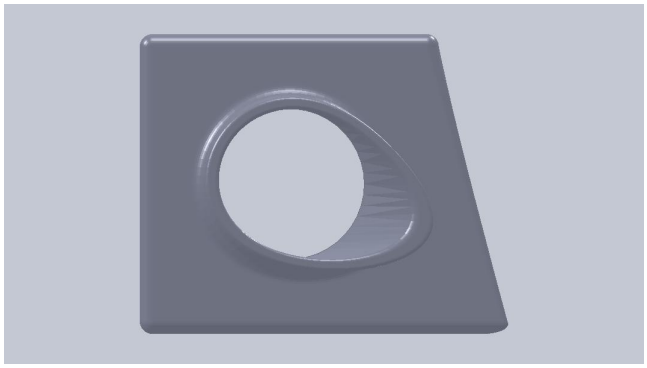

Figure A.8: Upside vision of a deformed tridimensional object imported in SOLIDWORKS.

So we can say that the object can be created in SOLIDWORKS, imported to MATLAB, deformed, then reimported to SOLIDWORKS. But the drawback of the STL format, the only format file directly importable and exportable to MATLAB, is evident in these figures and, in order to see it better, figure A.9 shows a detail of a lateral face, being the most deformed one.

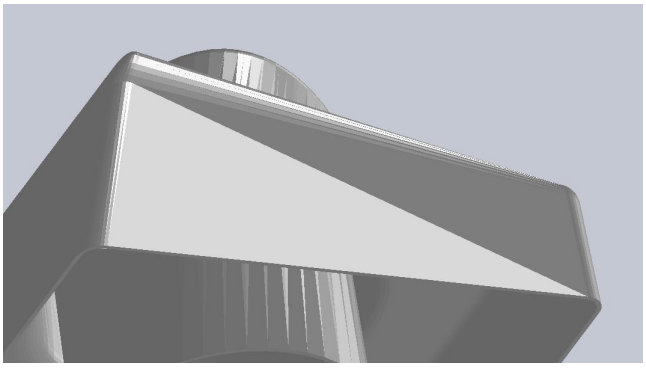

Figure A.9: A detail of the deformed object.

It is a clear evidence of the STL approximation and its limit: automatically, the plane face has been composed by two triangles. This is not enough to represent properly the deformation and the new geometry, as can be seen in figure A.9. So, what we have deformed is just an approximation of the real geometry by moving the triangles which composed it. So the fact that deformation may be good or not depends on how many triangles are used, and this generally is not up to the user. Typically, on planar surface, there are very few triangles, because they are enough to represent it, from an undeformed point of view.

This problem could be avoided by using of appropriate programs that are able to edit and better perform the STL files, allowing the user to decide the number of triangles to use. But this can cause an increase in the complexity of the approximation which is not very practical. An easier way to proceed is to use the IGS files, as done in this work, which are more universal among CAD programs, as pointed out in chapter 3.

Bibliography

[1] Available from: http://help.solidworks.com.

[2] Available from: http://www.autodesk.it/adsk/servlet/pc/index?siteid=457036&id=14626681.

[3] Available from: http://www.autodesk.it/adsk/servlet/pc/index?siteid=457036&id=14646075.

[4] Available from: http://www.faqs.org/faqs/graphics/fileformats-faq/part3/section-45.html.

[5] Available from: http://www.iso.org/iso/home.html.

[6] Available from: http://www.mathworks.com/help/techdoc/index.html.

[7] Available from: http://www.pardiso-project.org.

[8] Available from: http://www.rhino3d.com.

[9] Available from: http://www.vda-qmc.de.

[10] Available from: http://www.wikipedia.org.

[11] I. H. Abbot and A. E. Von Doenhoff. *Theory of Wing Sections*. Dover Publications Inc., New York, 1959.

[12] R. A. Adams and C. Essex. *Calculus: a complete course*. Pearson Education, Canada, 7 edition, 2009.

[13] V. I. Agoshkov. *Optimal Control Methods and Adjoint Equations in Mathematical Physics Problems*. Institute of Numerical Mathematics, Russian Academy of Science, Moscow, 2003.

[14] E. I. Amoiralis and I. K. Nikolos. Freeform Deformation Versus B-Spline Representation in Inverse Airfoil Design. *J. Comput. Inform. Sci. Eng.*, 8(2):024001–1–024001–13, June 2008.

[15] J. D. Anderson Jr. *Fundamentals of Aerodynamics*. McGraw-Hill, 2001.

[16] M. Andreoli, A. Janka, and J. A. Désidéri. Free-form-deformation parameterization for multilevel 3d shape optimization in aerodynamics. *INRIA Research Report no. 5019*, November 2003.

[17] M. A. Arbib (Ed.). *The Handbook of Brain Theory and Neural Networks*. 1995.

[18] C. Ashcraft, R. Grimes, J. Liu, J. Patterson, D. Pierce, Y. Pierce, P. Schartz, J. Schulze, W. P. Tang, D. Wah, and J. Wu. SPOOLES 2.2: SParse Object Oriented Linear Equation Solver, http://www.netlib.org/linalg/spooles/spooles.2.2.html, January 1999.

[19] W. Banzhaf, R. E. Nordin, P. Keller, and F. D. Francone. *Genetic Programming - An introduction*. Morgan Kaufmann, San Francisco, CA, 1998.

[20] A. Baron. Fluid dynamics. Course material, Politecnico di Milano, 2001.

[21] R. H. Bartels, J. C. Beatty, and B. A. Barsky. *An introduction to Splines for use in computer graphics and geometric modeling*. Morgan Kaufmann, 1995.

[22] M. Botsch and L. Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Trans. on Graphics*, 23(3):630–634, 2004. Proc. ACM SIGGRAPH.

[23] A. Buffa, G. Sangalli, and R. Vazquez. Isogeometric analysis in electromagnetics: B-splines approximation. *Comp. Meth. Appl. Mech. Eng.*, 199(17-20):1143–1152, 2010.

[24] Comsol. *COMSOL Multiphysics Modeling Guide*. COMSOL AB, 3.5a edition, 2007.

[25] Comsol. *COMSOL Multiphysics User's Guide*. COMSOL AB, 3.5a edition, 2007.

[26] T. A. Davis. UMFPACK version 4.1, http://www.cise.ufl.edu/research/sparse/umfpack, April 2003.

[27] L. Dedé. Reduced basis method for parametrized elliptic advection-reaction problems. *Journal of Comp. Math.*, 28(1):122–148, 2010.

[28] M. D'Elia, L. Dedé, and A. Quarteroni. Reduced basis method for parametrized differential algebraic equations. *Boletin de la Sociedad Española de Matematica Aplicada*, 46:45–73, 2009.

[29] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM Publication, Philadelphia, 1998.

[30] J. A. Désidéri, R. Duvigneau, B. Abou El Majd, and Z. Tang. Algorithms for efficient shape optimization in aerodynamics and coupled disciplines. In *42nd AAAF Congress on Applied Aerodynamics*, Sophia-Antipolis, France, March 2007.

[31] R. Duvigneau. Adaptive parameterization using Free-form deformation for aerodynamic shape optimization. *INRIA Research Report RR-5949*, July 2006.

[32] C. Fassardi and K. Hochkirch. Sailboat design by response surface optimization. In *High Performance yacht Design Conference*, Auckland, New Zealand, 2006.

[33] J. Feng. B-spline free-form deformation of polygonal objects through fast functional composition. In *Geometric Modeling and Processing, Theory and Applications, Proceedings*, pages 408 – 414, Hong Kong , China, 2000.

[34] K. C. Giannakoglou. Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence. *Progress Aerospace Sci.*, 38:43–76, 2002.

[35] V. Girault and P. A. Raviart. *Finite Element Methods for Navier-Stokes Equations*. Springer-Verlag, Berlin-Heidelberg, 1986.

[36] M. S. Gockenbach. Introduction to sequential quadratic programming. Course material, Michigan Technological University.

[37] M. D. Gunzburger and H. Kim. Existence of an optimal solution of a problem for the stationary Navier-Stokes equations. *SIAM J. Control Optim.*, 36(3):895–909, 1998.

[38] G. F. Hadley. *Nonlinear and dynamic programming.* Addison-Wesley, 1964.

[39] T. J. R. Hughes, A. Reali, and G. Sangalli. Efficient quadrature for NURBS-based isogeometric analysis. *Comp. Meth. Appl. Mech. Eng.*, 199(5-8):301–313, 2010.

[40] A. Jameson. Optimum Aerodynamic Design using CFD and Control Theory. In *AIAA Paper 95-1729*, 12th AIAA Computational Fluid Dynamics Conference, 1995.

[41] A. Jameson. Aerodynamic Design via Control Theory. *Journal of Scientific Computing*, 3:233–260, 1998.

[42] A. Jameson and L. Martinelli. *Aerodynamic shape optimization techniques based on control theory*, volume 1739/2000 of *Lecture Notes in Mathematics*. Computational Mathematics Driven by Industrial Problems, Springer Berlin / Heidelberg, 2000.

[43] A. Jameson, N. Pierce, and L. Martinelli. Optimum Aerodynamic Design using the Navier-Stokes Equations. *AIAA Paper 97-0101*, 1997.

[44] T. Lassila, A. Quarteroni, and G. Rozza. A reduced basis model with parametric coupling for fluid-structure interaction problems. *Submitted to SIAM Journal of Scientific Computing*, 2010.

[45] T. Lassila and G. Rozza. Parametric free-form shape design with PDE models and reduced basis method. *Comp. Meth. Appl. Mech. Eng.*, 199:1583–1592, 2010.

[46] J. L. Lions. *Optimal Control of Systems Governed by Partial Differential Equations*. Springer-Verlag, 1971.

[47] M. Lombardi, N. Parolini, G. Rozza, and A. Quarteroni. Numerical simulation of sailing boats dynamics and shape optimization. In preparation, 2011.

[48] G. I. Magoon and C. Pfrommer. Ironing out IGES. *Computer-Aided Engineering*, 8(1):52–54, 1989.

[49] A. Manzoni. Ottimizzazione di forma per problemi di fluidodinamica: analisi teorica e metodi numerici. Master degree thesis, Politecnico di Milano, 2008.

[50] A. Manzoni, A. Quarteroni, and G. Rozza. Shape optimization for viscous flows by reduced basis method and free form deformation. Submitted, 2010.

[51] B. Mohammadi and O. Pironneau. *Applied Shape Optimization for Fluids*. Oxford University Press, Oxford, 2001.

[52] A. M. Morris, C. B. Allen, and T. C. S. Rendall. CFD-based optimization of aerofoils using radial basis functions for domain element parameterization and mesh deformation. *Int. J. Numer. Methods Fluids*, 58:827–860, 2008.

[53] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999.

[54] N. Parolini and A. Quarteroni. Mathematical models and numerical simulations for the America's Cup. *Comp. Meth. Appl. Mech. Eng.*, 194:1001–1026, 2005.

[55] L. Piegl and W. Tiller. *The NURBS book*. Springer-Verlag, 1997.

[56] H. Prautzsch, W. Boehm, and M. Paluszny. *Bézier and B-Spline techniques*. Springer-Verlag, 2002.

[57] C. Praveen and R. Duvigneau. Low cost PSO using metamodels and inexact pre-evaluation: Application to aerodynamic shape design. *Comp. Meth. Appl. Mech. Eng.*, 198:1087–1096, 2009.

[58] L. Quartapelle and F. Auteri. Fluid dynamics. Course material, Politecnico di Milano, 2004.

[59] A. Quarteroni. *Numerical Models for Differential Problems*, volume 2 of *MS&A*. Springer, Milano, 2009.

[60] A. Quarteroni and G. Rozza. Optimal control and shape optimization in aorto-coronaric bypass anastomoses. *Mathematical Models and Methods in Applied Sciences (M3AS)*, 13(12):1801–23, 2003.

[61] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. Springer, Milano, 2007.

[62] A. Quarteroni and F. Saleri. *Scientific Computing with MATLAB and Octave*. Springer-Verlag Berlin, 2006.

[63] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations*. Springer-Verlag, 2 edition, 1997.

[64] D. F. Rogers. *An introduction to NURBS*. Morgan Kaufmann, 2001.

[65] G. Rozza. Controllo Ottimale e ottimizzazione di forma in fluidodinamica computazionale. Master degree thesis, Politecnico di Milano, 2002.

[66] G. Rozza. On Optimization, Control and Shape Design for an arterial bypass. *International Journal Numerical Methods for Fluids*, 47(10-11):1411–1419, 2005. Special issue for ICFD Conference, University of Oxford.

[67] G. Rozza. An introduction to reduced basis method for parametrized PDEs. In World Scientific, editor, *Applied and Industrial Mathematics in Italy*, volume 3 of *Series on Advances in Mathematics for Applied Sciences*, Vol. 82, pp. 508-519, Singapore, 2009. Proceedings of SIMAI Conference, Italian Society for Applied and Industrial Mathematics, Rome, Italy, 14-18 September 2008. EPFL-IACS report 01.2009.

[68] G. Rozza, D. B. P. Huynh, C. N. Nguyen, and A. T. Patera. Real-time reliable simulation of heat transfer phenomena. In *ASME - American Society of Mechanical Engineers - Heat Transfer Summer Conference Proceedings*, S. Francisco, CA, US, July 2009, Paper HT 2009-8812.

[69] G. Rozza, T. Lassila, A. Manzoni, E. Ronquist (ed.), and J. Hesthaven (ed.). Reduced basis approximation for shape optimization in thermal flows with a parametrized polynomial geometric map. In Springer Heildeberg, editor, *Spectral and High Order Methods for Partial Differential Equations, Lectures Notes in Comp. Science and Engineering*, volume 76, pages 307–315, 2010. Selected papers from the ICOSAHOM 09 Conference, NTU Trondheim, Norway, 22-26 June 2009.

[70] G. Rozza, A. Manzoni, J. Pereira (ed.), and A. Sequeira (ed.). Model order reduction by geometrical parametrization for shape optimization in computational fluid dynamics. In *Proceedings of ECCOMAS 2010 CFD Conference*, Lisbon, Portugal, June 2010.

[71] D. Rypl and Z. Bittnar. Triangulation of 3D surfaces reconstructed by interpolating subdivision. *Computers and Structures*, 82:2093–2103, 2004.

[72] J. A. Samareh. A survey of shape parameterization techniques. In *CEAS AIAA ICASE NASA Langley International Forum on Aeroelasticity and Structural Dynamics*, June 1999.

[73] J. A. Samareh. A novel Shape Parameterization Approach. In *Tech. Rep. NASA-TM-1999-209116*, March 1999.

[74] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. In *Proceedings of SIGGRAPH - Special Interest Group on GRAPHics and Interactive Techniques*, volume 20, pages 151–159, August 1986.

[75] C.-K. Shene. CS3621 Introduction to Computing with Geometry Notes. Michigan Technological University, 1997-2008.

[76] W. Song and A. J. Keane. A study of shape parameterisation methods for airfoil optimization. In *Proceedings 10th AIAA/ISSMO Multidisciplinary Anal. Optim. Conf.*, volume 6, 2004.

[77] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13(2):631–633, 1992.

[78] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear systems*. Cambridge University Press, Cambridge, ISBN: 0521818281, April 2003.

[79] J. Wang and T. Jiang. Nonrigid registration of brain MRI using NURBS. *Journal Pattern Recognition Letters*, 28(2), 2007.