

Master's Degree in Informatics Engineering

Dissertation

Quality-Mesh Generation and Reconstruction for Engineering Design Optimisation

July, 2015

Gustavo Alexandre Martins

gamart@student.dei.uc.pt

Advisors

Prof. Dr. Carlos M. Fonseca

Prof. Dr. Luís C. Santos

FCTUC DEPARTMENT
OF INFORMATICS ENGINEERING
FACULTY OF SCIENCES AND TECHNOLOGY
UNIVERSITY OF COIMBRA



Quality-Mesh Generation and Reconstruction for Engineering Design Optimisation

Master's Degree in Informatics Engineering
Dissertation

July, 2015

Gustavo Alexandre Martins

Department of Informatics Engineering
University of Coimbra
gamart@student.dei.uc.pt

Advisors

Prof. Dr. Carlos M. Fonseca
Department of Informatics Engineering
University of Coimbra
cmfonsec@dei.uc.pt

Prof. Dr. Luís C. Santos
Institute of Mathematics and Statistics
University of São Paulo
lsantos@ime.usp.br

Jury

Prof. Dr. Luís C. Paquete
Department of Informatics Engineering
University of Coimbra
paquete@dei.uc.pt

Prof. Dr. Jorge Sá Silva
Department of Informatics Engineering
University of Coimbra
sasilva@dei.uc.pt

Abstract

Numeric simulation of physical phenomena of interest is currently a must in the development process of new products and systems, namely in the aerospace and automotive industries, among many others. However, greater simulation fidelity inevitably leads to computationally heavier simulators, strongly limiting the extent to which simulation-based numerical optimisation of physical systems may be employed in practice. In this dissertation we propose a new method for mesh reconstruction under domain boundary perturbations as a way of speeding up the simulation process in an optimisation context. This final report also describes existing algorithms for quality mesh generation in two-dimensional domains with curved boundaries, as well as suitable curve-modelling approaches.

Keywords

Airfoil parametrisation, Delaunay Triangulations, Quality-mesh generation, Mesh reconstruction

Resumo

A simulação numérica de fenómenos físicos de interesse é hoje uma prática incontornável no processo de desenvolvimento de novos produtos e sistemas, nomeadamente nas indústrias aeroespacial e automóvel, entre muitas outras. No entanto, o aumento da fidelidade da simulação traduz-se inevitavelmente no aumento do peso computacional dos simuladores, o que condiciona fortemente a optimização numérica de sistemas físicos baseada em simulação. Nesta dissertação é proposto um novo método para reconstrução de malhas na sequência de perturbações às fronteiras do domínio do problema como forma de acelerar o processo de simulação num contexto de optimização. Este relatório final descreve também algoritmos já existentes para a geração de malhas de qualidade em domínios bidimensionais limitados por fronteiras curvas, bem como formas de modelar essas fronteiras.

Palavras-chave

Parametrização de perfis aerodinâmicos, Triangulações de Delaunay, Geração de malhas de qualidade, Reconstrução de malhas

Contents

1	Introduction	1
2	Curve Modelling and Airfoil Parametrisation	3
2.1	Non-Uniform Rational B-Splines	3
2.1.1	Control Points	4
2.1.2	Knot Vector	4
2.1.3	Basis functions	5
2.1.4	NURBS curve	5
2.2	Class Shape Transformation	5
2.2.1	Class function	6
2.2.2	Shape function	6
2.2.3	Trailing edge thickness	8
2.2.4	Airfoil function	8
2.3	Boundary Interface	9
2.3.1	Critical points	9
2.3.2	Initial discretisation	11
2.3.3	Segment splitting	13
2.3.4	Point finding	13
2.3.5	Curvature	15
3	Two-Dimensional Delaunay Triangulation and Refinement	17
3.1	Geometry Predicates and Functions	17
3.2	Data Structures	20
3.3	Properties of a Delaunay Triangulation	21
3.4	Building a Delaunay Triangulation	22
3.4.1	Inserting a Vertex	22
3.4.2	Point Location by Walking	24
3.5	Building a Constrained Delaunay Triangulation	26
3.5.1	Inserting a Vertex	26
3.5.2	Inserting a Segment	26
3.5.3	The Flip Algorithm	27
3.5.4	The Gift-Wrapping Algorithm	28
3.6	Delaunay Refinement Algorithms	29
3.6.1	Ruppert's Delaunay Refinement Algorithm	30
3.6.2	Chew's Second Delaunay Refinement Algorithm	31
3.6.3	Shewchuk's Diametral Lenses	32
3.7	Application to Curved Boundaries	33
3.8	Cell Size and Grading Control	33
3.8.1	Ollivier-Gooch and Boivin's Length Scale	33
3.8.2	Gosselin's Modifications for Curved Boundaries	34

4	Mesh Reconstruction for Two Dimensions	37
4.1	Mesh deformation	37
4.1.1	Tension spring method	37
4.1.2	Torsional spring method	38
4.1.3	Observations	38
4.2	New reconstruction process	39
4.2.1	Boundary adjustment	39
4.2.2	Interior building	39
4.2.3	Removal	39
4.2.4	Rebuilding	41
4.3	Additional Observations	41
5	Experimental Setup	43
5.1	Methods and Algorithms	43
5.1.1	Airfoil Parametrisation	43
5.1.2	Mesh Generation	43
5.1.3	Summary	44
5.2	Shape Optimisation	44
5.2.1	Direct-search methods	44
5.2.2	Covariance Matrix Adaptation - Evolution Strategy (CMA-ES)	45
5.3	Airfoils	45
5.3.1	Standard airfoil	46
5.3.2	Medium thickness	46
5.3.3	Small thickness	47
5.3.4	Large thickness	47
5.4	Methodology	48
6	Experimental Results and Discussion	49
6.1	Shape optimisation	49
6.2	Impact of standard deviation	50
6.3	Scalability tests	51
6.4	Improvements	53
6.4.1	Shape coefficients	53
6.4.2	Number of triangles	54
6.4.3	Evaluation	54
7	Conclusions and Future Work	57
A	Results' Figures	61

List of Figures

2.1	Basic shapes examples produced by the Class function	6
2.2	Airfoil and its Critical points	9
2.3	Golden section search	10
2.4	Airfoil Uniform discretisation (30 points)	12
2.5	Airfoil Variation discretisation (30 points, $TV(\theta)_{max} \approx \frac{\pi}{25}$)	12
2.6	Airfoil Cosine discretisation (30 points)	13
2.7	Bisection method	15
3.1	Doubly Connected Edge List and its Half-edges	20
3.2	Locally Delaunay property	22
3.3	Bowyer-Watson algorithm	23
3.4	Point location by Walking	24
3.5	Point location by Walking (special case)	25
3.6	Flipping step	27
3.7	Segment insertion prior to the gift-wrapping algorithm	28
3.8	Gift-Wrapping algorithm	28
3.9	Steps of the Ruppert's refinement algorithm and segment splitting	31
3.10	Step of the Chew's refinement algorithm and segment splitting	32
3.11	Comparison between a diametral circle and Shewchuk's diametral lenses	32
4.1	Illustration of tension/linear springs	38
4.2	Combined tension/torsional springs	38
4.3	Descending lexicographic triangulation at the interior of an airfoil	39
4.4	Trapezoids creation	40
4.5	Steps of the mesh reconstruction method	42
5.1	NACA 0012 airfoil	46
5.2	NACA 2412 airfoil	46
5.3	Clark-Y airfoil	47
5.4	Boeing 737 airfoil	47
5.5	RAE 2822 airfoil	47
5.6	NACA 63206 airfoil	47
5.7	Eppler 376 airfoil	47
5.8	Eppler 545 airfoil	48
5.9	Gottingen 702 airfoil	48
6.1	Number of individuals evaluated by the CMA-ES algorithm	49
6.2	Illustration of triangle surplus	51
6.3	Illustration of bad gradation	53
6.4	Comparison between generation, reconstruction and its improvements, $\sigma = 0.15$, $I = 450$, $G = 7$	55

A.1	Comparison between mesh generation and mesh reconstruction while varying σ for the NACA 63206 airfoil (thin), $I = 250$, $G = 4$	61
A.2	Comparison between mesh generation and mesh reconstruction while varying σ for the Clark-Y airfoil (medium), $I = 250$, $G = 4$	62
A.3	Comparison between mesh generation and mesh reconstruction while varying σ for the Gottingen 702 airfoil (thick), $I = 250$, $G = 4$	63
A.4	Comparison between mesh generation and mesh reconstruction while varying I and G for the Eppler 376 airfoil (thin), $\sigma = 0.05$	64
A.5	Comparison between mesh generation and mesh reconstruction while varying I and G for the RAE 2822 airfoil (medium), $\sigma = 0.05$	65
A.6	Comparison between mesh generation and mesh reconstruction while varying I and G for the Eppler airfoil (medium), $\sigma = 0.05$	66
A.7	Comparison between mesh generation and mesh reconstruction and its improvements for the Boeing 737 airfoil, $\sigma = 0.05$, $I = 250$, $G = 4$	67
A.8	Comparison between mesh generation and mesh reconstruction and its improvements for the Boeing 737 airfoil, $\sigma = 0.15$, $I = 250$, $G = 4$	67
A.9	Comparison between mesh generation and mesh reconstruction and its improvements for the Boeing 737 airfoil, $\sigma = 0.05$, $I = 450$, $G = 7$	68
A.10	Comparison between mesh generation and mesh reconstruction and its improvements for the Boeing 737 airfoil, $\sigma = 0.15$, $I = 450$, $G = 7$	68

List of Tables

2.1	Convergence study results (Required order of Bernstein polynomial)	7
3.1	Data Structure interface	21
5.1	Methods and algorithms for the experimentation	44
5.2	Machine and implementation specifications	48
6.1	Comparison between mesh generation and reconstruction under variation of σ	50
6.2	Additional information on the reconstruction results	51
6.3	Comparison between mesh generation and reconstruction under variation of I and G	52
6.4	Additional information on the reconstruction results	52
6.5	Comparison between mesh generation and reconstruction for the Boeing 737 airfoil	53
6.6	Comparison between mesh generation and improved reconstruction based on coefficient variation for the Boeing 737 airfoil	54
6.7	Comparison between mesh generation and improved reconstruction based on triangle surplus for the Boeing 737 airfoil	54
6.8	Comparison between mesh reconstruction and its extensions for the Boeing 737 airfoil	55

List of Algorithms

3.1	Bowyer-Watson algorithm	24
3.2	Walking algorithm	25
3.3	Flip algorithm	27
3.4	Gift-Wrapping algorithm	29

Acronyms

CDT	Constrained Delaunay Triangulation
CMA-ES	Covariance Matrix Adaptation - Evolution Strategy
CST	Class Shape Transformation
DCEL	Doubly Connected Edge List
DT	Delaunay Triangulation
NURBS	Non-Uniform Rational B-Splines
PSLG	Planar Straigh-Line Graph

Notation

Boundary Interface

I	Number of points in the initial discretisation
P	Vector of critical points
$TV(\theta)$	Total variation of the tangent angle

Class Shape Transformation

ψ	CST variable, $\frac{x}{c}$
$\Delta\eta$	Trailing edge thickness
η	CST function, $\frac{y}{c}$
A	Shape function coefficients
a	Airfoil function
B	Bernstein basis polynomial
C	Class function
c	Airfoil's chord length
e	Class function coefficients, $\{e_1, e_2\}$
K	Binomial coefficient
S	Shape function
T	Trailing edge thickness function

Delaunay Triangulation

B	Minimum angle bound
G	Length scale grading parameter
H	Length scale bound
lfs	Local feature size
lfs_c	Local feature size for vertices on curves

LS	Length scale
R	Length scale resolution parameter

Non-Uniform Rational B-Splines

c	Curve function
N	Basis function

Shape Optimisation

σ	CMA-ES initial standard deviation
----------	-----------------------------------

Chapter 1

Introduction

Designing the shape of aerodynamic components can be a demanding and costly task, and as it is difficult and expensive to perform tests on real models, it is common practice to perform the design process on computer simulators. In many engineering domains, and especially in aeronautics, fluid dynamic simulators are commonly used to optimise various system designs, e.g. airfoils of aircraft wings. However, the complexity of such simulators is increasing rapidly and high-fidelity simulators are computationally heavy, leading to simulations that may take intolerable amounts of time to finish. Usually, the problem is overcome by using more and more powerful hardware and using parallel processing to accelerate the process.

In engineering design optimisation, computer simulations are often driven by an optimisation process. The optimisation method, given a set of parameters concerning the shape of the model, repeatedly perturbs these parameters until a solution close to optimal is reached. The simulator has the task of, given a certain model as input, returning useful information for its evaluation, e.g. the resulting lift and drag in the case of aircraft wings. Fluid dynamic simulators with a partitioning of the space around the design, such as an airfoil, in the form of meshes, also known as grids, to discretise the problem.

The purpose of this dissertation is not only to implement algorithms to generate good-quality unstructured triangular meshes, i.e. meshes with irregular connectivity, but also to develop methods to adapt previous meshes to new designs, thereby preserving useful information between iterations. This should allow the simulator to perform more efficiently in an optimisation context, and thus decrease the overall computational cost of the optimisation process.

It is known that the quality of input meshes have a considerable influence over the quality and precision of the results of fluid dynamic simulators. Therefore, when it comes to be able to generating guaranteed good-quality meshes, using of Delaunay Triangulations is a natural choice. Among the optimal properties that they exhibit, the one that stands out the most is that the Delaunay triangulation maximizes the minimum angle among all possible triangulations of a fixed set of points. Moreover, Delaunay triangulations have been object of extensive study over the past years and good algorithms are widely available for their construction and refinement.

Incremental approaches to re-meshing based on mesh deformation have been studied in the past [4], [11]. However, with these methods it is not possible to guarantee that the new mesh is still Delaunay, and thus that it retains its optimal properties. Furthermore, the number of elements in the mesh is not altered when using mesh deformation methods, which may not be suitable for large perturbations.

In this dissertation we propose a new incremental approach to re-meshing based on mesh reconstruction. Unlike mesh deformation, we have the possibility of removing and adding triangles to the mesh, and thus the number of triangles may be varied as needed. Moreover, we are capable of maintaining a Delaunay Triangulation, thus preserving its properties. Also, we can apply refinement algorithms to guarantee high mesh quality in every iteration.

This document is organised as follows. Chapter 2 reviews two methods for curve modelling and the importance of this step in airfoil design. The mathematical models are presented and the correlation between their design parameters and the corresponding shapes discussed, as well as how these models can be used in an optimisation context. The description of a software interface for handling domains with curved boundaries in a object-oriented way completes the chapter. In Chapter 3 we review Delaunay Triangulations, along with their properties and quality guarantees. Appropriate data structures, and algorithms for DTs construction and refinement with quality guarantees and good-grading properties are described.

In Chapter 4, existing alternatives to iterative re-meshing using mesh deformation methods are reviewed before our new mesh-reconstruction approach is proposed and discussed, including its advantages, disadvantages, and possible extensions and adjustments. In Chapter 5 we recover the methods described in previous chapters, and select the most appropriate for the experimental study. In addition, algorithms to simulate a real optimisation scenario are discussed. Finally, Chapter 6 is dedicated to the results and their analysis.

Chapter 2

Curve Modelling and Airfoil Parametrisation

An important aspect of any the optimisation process is the parametrisation that is used to represent the designs to be optimised. A parametrisation that does not provide an intuitive and easy control over the design shape may not be a good choice. According to B. Kulfan [19], it is desirable for an airfoil parametrisation to have the following features:

- Well behaved, and produce smooth and realistic shapes
- Mathematically efficient and numerically stable process that is fast, accurate and consistent
- Require relatively few variables to represent a large enough design space to contain optimum aerodynamic shapes for a variety of design conditions and constraints
- Allows specification of key design parameters such as leading edge radius, boat-tail angle, airfoil closure
- Provide easy control for designing and editing the shape of a curve
- Intuitive: Geometry algorithm should have an intuitive and geometric interpretation
- Systematic and Consistent: The way of representing, creating and editing different types of geometries must be the same
- Robust: The represented curve does not change its geometry under geometric transformations such as translation, rotation and affine transformations

This chapter reviews two distinct ways to model curves. One is more generic and capable of supporting any kind of shapes, while the other was specifically created for the practical case of this dissertation, airfoils. The latter is also capable of representing other components of an aircraft, such as nacelles and fuselage, among others. Further on, an interface for boundaries that serves as a bridge between mesh generation methods and the boundary itself is introduced.

2.1 Non-Uniform Rational B-Splines

One way to easily model surfaces is through the use of Non-Uniform Rational B-Splines (NURBS). They are commonly used in the field of Computer Graphics, and the main reason for this is that they offer great flexibility when it comes to representing both analytic and modelled surfaces. Although they can be used in three dimensions, in this review these are only considered in a two-dimensional space.

NURBS are governed by two types of parameters, *control points* and a *knot vector*. Briefly, control points determine the shape of the curve, whereas the knot vector determines where and how these control points affect the curve. Together, they produce a set of *basis functions*, one for each control point. Basis functions are polynomial functions that determine where and how much a single control point can influence the shape of the curve.

2.1.1 Control Points

Unlike other types of splines, such as *interpolating splines*, NURBS's control points are not interpolated. Instead, they are approximated. This means that there is no obligation for the resulting curve to pass through these points. Control points are defined as coordinates in the Cartesian coordinate system.

Each of these control points has an associated *weight*, which represents how much influence that control point has on the shape of the curve. If the value is high, the curve passes through closer to the point, i.e. the point is better approximated. For a large enough value, this weight can make the curve to pass almost through the point, as if it was interpolated. However, this weight is relative to those of the other control points. For example, setting all weights to 1 would produce the same shape as setting them all to 5.

The term *rational* in NURBS refers to these weights.

2.1.2 Knot Vector

The knot vector is a sequence of values that determines where each control point affects the curve. The knot vector divides the parametric space into several intervals, called *knot spans*. Whenever the parameter value enters a new knot span, a new control point becomes active, i.e. it starts affecting the shape of the curve. At the same time, an old control point becomes inactive, no longer affecting the shape.

The number of knots, or knot vector length, is always equal to the number of control points, plus the degree of the curve, plus one. Thus, if the curve is cubic, and 10 control points are defined, the length of the knot vector must be 14. These values must also be in a non-decreasing order. It is possible to have consecutive knots with the same value, meaning that the knot span is null. This is sometimes referred to as a knot with a certain *multiplicity*. Multiple knots cause two or more consecutive control points to become active at the same time. The domain of the knot vector is not bounded, thus it can assume any range of values.

To obtain the same degree along the whole curve, the knot vector usually starts with a knot that has multiplicity equal to its degree plus one. This causes the activation of the right number of control points right at the start. Similarly, the knot vector ends with a knot of the same multiplicity. An example for a knot vector for a NURBS of degree 3 defined by 8 control points is as follows:

$$[0, 0, 0, 0, 0.2, 0.3, 0.5, 0.9, 1, 1, 1, 1]$$

The term *non-uniform* in NURBS refers to the fact that the length of knot spans may vary along the curve.

2.1.3 Basis functions

NURBS curves are constructed using basis functions. These are denoted as $N_{i,n}(t)$, where i refers to the i^{th} control point, n is the degree of the basis function, and t is the parameter value, defined within the domain of the knot vector. These functions are recursive in n , with $N_{i,n}$ being the linear combination of $N_{i,n-1}$ and $N_{i+1,n-1}$. Basis functions of degree 0, $N_{i,0}$, are piecewise constant functions, taking the value 1 on the corresponding (i^{th}) knot span and 0 everywhere else. $N_{i,n}$ is defined as:

$$N_{i,n}(t) = f_{i,n}(t)N_{i,n-1}(t) + g_{i+1,n}(t)N_{i+1,n-1}(t) \quad (2.1)$$

where $f_{i,n}$ increases linearly from 0 to 1 where $N_{i,n-1}$ is non-zero, and $g_{i+1,n}$ decreases from 1 to 0 where $N_{i+1,n-1}$ is non-zero. They can be defined as:

$$f_{i,n}(t) = \frac{t - k_i}{k_{i+n} - k_i} \quad g_{i,n}(t) = \frac{k_{i+n} - t}{k_{i+n} - k_i} \quad (2.2)$$

where k_i represents the i^{th} knot.

2.1.4 NURBS curve

Using the definition of basis function (Equation 2.1), a NURBS curve is defined as follows:

$$c(t) = \frac{\sum_{i=1}^m N_{i,n}(t) w_i p_i}{\sum_{i=1}^m N_{i,n}(t) w_i} \quad (2.3)$$

where m is the number of control points, p_i the i^{th} control point and w_i the corresponding weight.

2.2 Class Shape Transformation

In 2006, Brenda Kulfan proposed a new method to represent parametric airfoil geometries, called Class Shape Transformation (CST) [19]. This representation method, although capable of representing several shapes beyond airfoil shapes (as it is shown further on), was specifically created for them, thereby easing their modelling and modification. Compared to NURBS, the CST method is less generic, but more appropriate to the practical case addressed in this dissertation. As such, it was the chosen method for the subsequent work.

The CST method involves three main functions: *class function*, *shape function* and *trailing edge function*. The first function defines the class of the airfoil, giving it a basic shape. The second function adjusts the shape given by the first function. Finally, the third function controls the thickness of the airfoil at the trailing edge. These functions are further detailed in the following sections.

An airfoil represented by the CST method is defined as

$$\eta(\psi) = C(\psi) \cdot S(\psi) + T(\psi) \quad (2.4)$$

where η and ψ are defined as

$$\eta = \frac{y}{c} \quad \psi = \frac{x}{c}$$

where c denotes the length of the airfoil chord, i.e. the distance between the leading and trailing edges of the airfoil.

Leading and Trailing edges Regardless of what is being represented by the CST method, the direction of its motion is always from the right to the left. Therefore, the leading edge is the leftmost point of the model and the trailing edge is the rightmost point.

2.2.1 Class function

As mentioned previously, the class function is what defines the class of a model, giving it its basic shape. It is defined as

$$C(\psi) = \psi^{e_1} \cdot (1 - \psi)^{e_2} \quad (2.5)$$

where e_1 and e_2 are two coefficients that control the shape of the model at its leading and trailing edges, respectively.

As it can be seen in Figure 2.1, the class function can be used for many shapes, not necessarily all airfoils. Usually, the values assigned to e_1 and e_2 range between 0 and 1, although it is also possible to have values greater than 1. In this dissertation, since the practical case concerns models similar to those represented by the class of NACA Airfoils [17] (Figure 2.1A), the chosen values for these coefficients are 0.5 and 1, respectively.

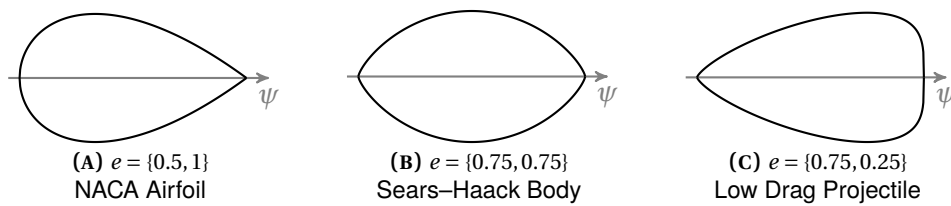


FIGURE 2.1. Basic shapes examples produced by the Class function

2.2.2 Shape function

The purpose of the shape function is to adjust the shape given by the class function until the desired representation of the model is obtained. The shape function is defined as

$$S(\psi) = \sum_{i=0}^n A_i \cdot B_{i,n}(\psi) \quad (2.6)$$

where A_i is the i^{th} shape coefficient and $B_{i,n}$ the i^{th} Bernstein basis polynomial of degree n .

Shape coefficients

The shape coefficients are represented by the vector A , which is of size $n + 1$. The purpose of these coefficients, together with the Bernstein polynomials, is to adjust the shape of the model at a certain region without affecting the other regions of the curve too much. These are the object of study when we talk about shape optimisation.

In an unpublished note [18], Kulfan showed results of convergence studies for several definitions of exactness using a well-known airfoil, the RAE2822. Two of them are the following:

- **Manufacturing exactness:** The differences, ϵ , between the actual airfoil coordinates and the approximated ones are less than wind tunnel or full scale manufacturing tolerances:
 - for $\psi = 0$ to 0.2 : $\epsilon < 0.0762$ millimetres
 - for $\psi > 0.2$: $\epsilon < 0.1524$ millimetres
- **Measurement exactness:** The differences between the actual airfoil coordinates and the approximated ones are less than typical wind tunnel measurement capability:
 - for any value of ψ : $\epsilon < 0.0254$ millimetres

The results of the convergence studies (Table 2.1) indicate the minimum order of Bernstein polynomial required to achieve the desired values of exactness. Given the results presented, the adopted Bernstein polynomial order for this dissertation was chosen to be 8, meaning that $n = 7$.

	Manufacturing	Measurement
Upper Surface	4	7
Lower Surface	5	8

TABLE 2.1. Convergence study results (Required order of Bernstein polynomial)

Bernstein polynomial

A Bernstein polynomial is a linear combination of Bernstein basis polynomials. Bernstein polynomials are used in the shape function (Equation 2.6), and, together with A , allow the modification of the design shape. The higher the degree of a Bernstein polynomial, the higher the control one can have. However, a higher degree also means higher complexity, rendering the representation method slow.

A Bernstein basis polynomial is defined as

$$B_{i,n}(\psi) = K_{i,n} \cdot \psi^i (1 - \psi)^{n-i} \quad (2.7)$$

where $K_{i,n}$ is the i^{th} binomial coefficient of degree n , defined as

$$K_{i,n} = \binom{n}{i} = \frac{n!}{i!(n-i)!} \quad (2.8)$$

One useful property of Bernstein basis polynomials is the fact that their derivative can be written as a combination of two Bernstein basis polynomials of lower degree:

$$B'_{i,n}(\psi) = \begin{cases} n \cdot B_{i-1,n-1}(\psi) & , i = n \\ n \cdot -B_{i,n-1}(\psi) & , i = 0 \\ n \cdot (B_{i-1,n-1}(\psi) - B_{i,n-1}(\psi)) & , \text{otherwise} \end{cases} \quad (2.9)$$

2.2.3 Trailing edge thickness

To control the trailing edge there is yet another component of the CST. In the original scheme, proposed by Kulfan, this function was presented as part of Equation 2.4. However, we decided to divide the parametrisation into three functions to better explain its components. The function to control the trailing edge is defined as

$$T(\psi) = \psi \cdot \Delta\eta \quad (2.10)$$

where $\Delta\eta$ is the desired thickness at the trailing edge. Using this function not only can one vary the thickness of the trailing edge of the airfoil, but also translate it in the η axis.

2.2.4 Airfoil function

The CST as it is presented in Equation 2.4 only gives us one half of the airfoil. The use of two CST functions is thus required to represent both the upper and lower surfaces of the model. To achieve this, another function was created, the *airfoil function*, that calls the adequate CST function based on the parameter t . To keep things standardized, t always varies between 0 and 1. The airfoil function is defined as

$$a(t) = \begin{cases} \eta_u(1 - 2t) & , 0 \leq t < \frac{1}{2} \\ \eta_l(2t - 1) & , \frac{1}{2} \leq t \leq 1 \end{cases} \quad (2.11)$$

Note that functions with subscript u correspond to the upper surface, while functions with l correspond to the lower surface. The remaining functions are defined as follows

$$\begin{aligned} \eta_u(\psi) &= C(\psi) \cdot S_u(\psi) + T_u(\psi) & \eta_l(\psi) &= C(\psi) \cdot S_l(\psi) + T_l(\psi) \\ S_u(\psi) &= \sum_{i=0}^n A_{ui} \cdot B_{i,n}(\psi) & S_l(\psi) &= \sum_{i=0}^n A_{li} \cdot B_{i,n}(\psi) \\ T_u(\psi) &= \psi \cdot \Delta\eta_u & T_l(\psi) &= \psi \cdot \Delta\eta_l \end{aligned} \quad (2.12)$$

It can also be noticed that the Class function is the only one that does not depend on which surface is considered, meaning that the upper and lower surfaces must belong to the same class. Also note, from Equation 2.11, that the upper surface is mapped backwards (from the right to the left), while the lower surface is mapped from the left to the right. This makes the function more intuitive, starting at the trailing edge and traversing the airfoil counter-clockwise until the trailing edge is reached again.

Such a modification however, changes the first derivative of the airfoil function. Since the upper surface is mapped backwards in t , its derivative must be the negative of what it normally is (Equation 2.13). The second derivative does not depend on direction, and so it remains unchanged.

$$a'(t) = \begin{cases} -\eta'_u(1-2t) & , 0 \leq t < \frac{1}{2} \\ \eta'_l(2t-1) & , \frac{1}{2} \leq t \leq 1 \end{cases} \quad (2.13)$$

Since the design has two surfaces, the length of A is twice the order of the Bernstein polynomial. However, concerning airfoils, it is desired that the curvature be continuous at the leading edge. One can enforce this property by defining the first coefficient of the lower surface as the negative of the first coefficient of the upper surface [19], as follows

$$A_{u0} = -A_{l0}$$

2.3 Boundary Interface

Regardless of the method one might choose to represent the model, whether NURBS, CST, or any other, something that is always useful is a Boundary Interface, allowing the mesh generation program to interact with its boundaries and retrieve any necessary information related to them.

Any boundary interface must be able to provide at least two things: an initial discretisation, i.e. a set of points that accurately represents the boundary, and the midpoint of a segment defined by two points of the boundary (*segment splitting*). These two points may not necessarily be part of the initial discretisation. The interface may also provide other information that the user might need, such as the value of curvature at any given point of the boundary. However, they are not mandatory.

In the following, we shall discuss and explain the essential components of such an interface, how they are obtained, and where they can be used. To make things less vague, these are explored in the context of the CST parametrisation and the practical case of this dissertation, airfoils.

2.3.1 Critical points

The first thing that must be done at initialisation is determining the critical points of the given airfoil. These are useful in obtaining a possible initial discretisation, as well as performing segment splitting.

Critical points, or stationary points, are by definition the points in the domain of a function where its derivative is 0. In our case, the second derivative is also considered. From the first derivative we can determine the minima and maxima of the function, and from the second its inflexion points. In addition to these, since an airfoil is defined by two functions of the type $y = f(x)$, both leading and trailing edges are also considered (minimum and maximum in ψ). Figure 2.2 shows an airfoil and its critical points.

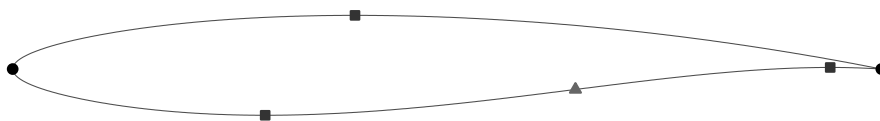


FIGURE 2.2. Airfoil and its Critical points
Leading and trailing edges are represented by circles, minima and maxima by squares and inflexion points by triangles.

To determine the critical points of an airfoil two methods were tested. The first one consisted on running the Gradient descent algorithm on both the function and its first derivative in order to find the minima and maxima, and the inflexion points, respectively. However, this method may prove too slow in regions where the derivatives are too small, as it is often the case in a typical airfoil. For the second method, a different approach was taken. Instead of performing a search on the entire domain, it is divided into small intervals. These intervals are then checked and whenever a change of sign is found in the first or second derivatives, the Golden section search is used instead to determine the critical point within.

Gradient descent

Gradient descent is an optimisation algorithm used to find local extrema of a function using its first derivative. It is an iterative method that checks the value of the derivative at each step and moves along the function accordingly (ideally closer to the solution). The iterative process can be defined as

$$t_{n+1} = t_n - \gamma_n \cdot a'(t_n) \quad (2.14)$$

where γ_n is the step size for iteration n . This parameter γ is useful for controlling the algorithm and preventing it from oscillating too much. The algorithm stops when the difference between t_n and t_{n+1} is smaller than a predefined value. As it is presented in Equation 2.14 the algorithm finds local minima. To make it find local maxima, the sign of the step must be changed.

With this, one can find all critical points of an airfoil by finding its maxima and minima, alternating between the maximisation and minimisation versions of the gradient descent at each critical point found, until the value of $t = 1$ is exceeded.

Golden section search

The golden section search is a method for finding the extremum of a function in a pre-specified range of values. Its called golden search due to the fact that the method uses triples of points whose distances form a golden ratio. It is an iterative method that, at each step, narrows the range of values inside which the extremum is known to exist. An illustration of the method can be seen in Figure 2.3.

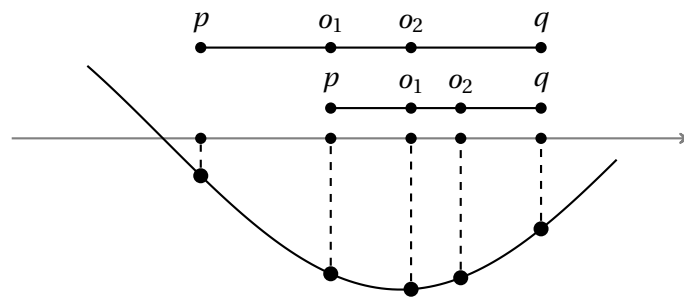


FIGURE 2.3. Golden section search
The change of orientation occurs in o_2 . Points p and o_1 are translated into o_1 and o_2 , respectively, and a new point o_2 is determined.

The algorithm starts with two points p and q , and determines two auxiliary points, o_1 and o_2 . The ratio between $o_2 - p$ and $q - o_2$ must be the golden ratio. The same happens with o_1 . These can be calculated as

$$o_1 = q - (\phi - 1) \cdot (q - p) \quad o_2 = p + (\phi - 1) \cdot (q - p)$$

where ϕ is the golden ratio. Then it is checked whether a change in orientation occurs in o_1 or o_2 , with the points being updated accordingly, narrowing the search space down. The algorithm stops when the distance between p and q is smaller than a predefined value.

Total variation of the tangent angle

The total variation of the tangent angle is a concept introduced by Boivin and Ollivier-Gooch [5], that is helpful to compute an initial discretisation based on variation and to perform the segment splitting. Its value can be calculated simply by the difference of tangent angle between each pair of consecutive critical points, determined above. This is only possible due to the fact that the maximum variation of tangent angle between consecutive critical points is $\frac{\pi}{2}$ (removing the need to determine if the curve changed orientation by a value of α or $2\pi - \alpha$ [5]) and no variation is lost due to the use of inflexion points as critical points.

The value of total variation of the whole airfoil can be computed as

$$TV(\theta)_{total} = \sum_{i=2}^n |\arctan(a'(P_i)) - \arctan(a'(P_{i-1}))| \quad (2.15)$$

where n is the number of critical points and P_i the i^{th} critical point. Note that to achieve this, the trailing edge must have two critical points (one for each surface), taking the values $t = 0$ and $t = 1$. In this way, it is possible to compute the total variation of the whole airfoil and to deal with the discontinuity at the trailing edge.

2.3.2 Initial discretisation

The mesh generation algorithm starts with an initial discretisation of its boundaries, if any, and many choices can be made at this point. If a coarse discretisation or one with better resolution; where to place the points along the boundary so it can represent the airfoil as good as possible; etc..

Due to certain properties of the algorithms that construct and control the mesh, explained in Sections 3.6, 3.7 and 3.8, it is desired for the discretisation to have good resolution all over the airfoil, as well as to accurately represent the airfoil in regions where the curvature is higher, allowing at most a $TV(\theta)$ value of $\frac{\pi}{6}$ or $\frac{\pi}{4}$ between consecutive points.

In the next sections three possible discretisations for airfoils are presented.

Uniform

The uniform discretisation is the simplest of the three presented here. Here, the points are placed at uniform intervals in the ψ axis. The value of t for these points can be calculated using

$$t_i = \left(\frac{i}{n} \right), \quad i = 0, \dots, n-1 \quad (2.16)$$

where n is the number of points in the discretisation. As can be seen in Figure 2.4, the points are well placed along the airfoil, allowing for a good resolution all over it. However, it does not accurately represent the leading edge of the airfoil where a higher value of curvature is expected. To solve this, one can increase the number of points in the discretisation, but at the cost of adding more points in regions where they are not needed.

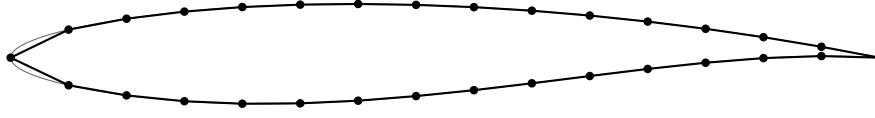


FIGURE 2.4. Airfoil Uniform discretisation (30 points)

Variation

Another possible discretisation strategy makes use of the values of $TV(\theta)$, calculated when determining the critical points, and the requirement for consecutive points to have a maximum $TV(\theta)$ value of $\frac{\pi}{6}$ or $\frac{\pi}{4}$ between consecutive points. For this discretisation, the points are determined not by their value of t , but of $TV(\theta)$, requiring a point finding method (explained later in Section 2.3.4) to obtain t . The values of $TV(\theta)$ can be calculated using

$$TV(\theta)_i = \frac{i}{n} \cdot TV(\theta)_{total} \quad , i = 0, \dots, n-1 \quad (2.17)$$

where n is the number of points in the discretisation. n may be a predefined value, but it can also be computed using the maximum value of $TV(\theta)$ allowed between consecutive points, with

$$n = \left\lceil \frac{TV(\theta)_{total}}{TV(\theta)_{max}} \right\rceil \quad (2.18)$$

In Figure 2.5, it can be seen that most points are located at the leading edge, where the curvature is higher, whereas fewer points are needed for the rest of the airfoil. Although it accurately represents the airfoil, the variation discretisation does not guarantee good resolution over the entire space. Just like with the uniform discretisation, this problem can be overcome by adding more points. However, if too many points are added, the resolution on the leading edge can become extremely high, which can cause the creation of unnecessary elements in the subsequent triangulation, and problems such as over-refinement (Section 3.6) and precision errors.

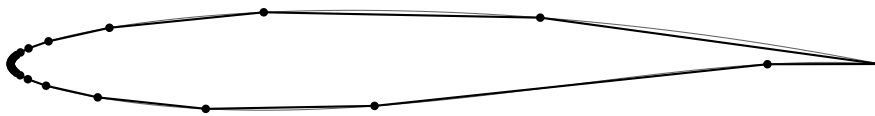


FIGURE 2.5. Airfoil Variation discretisation (30 points, $TV(\theta)_{max} \approx \frac{\pi}{25}$)

Cosine

The third and final discretisation scheme presented here is widely used in practice and is based on cosines. Here, the unit circle is divided into uniform sections of arc length, and their cosine values are

used as t . These can be calculated as

$$t_i = \begin{cases} \frac{\cos\left(\frac{2\pi i}{n}\right) - 1}{-4} & , 0 \leq \frac{i}{n} < \frac{1}{2} \\ \frac{\cos\left(\frac{2\pi i}{n}\right) + 3}{4} & , \frac{1}{2} \leq \frac{i}{n} < 1 \end{cases} , i = 0, \dots, n-1 \quad (2.19)$$

where n is the number of points in the discretisation. In Figure 2.6 is possible to observe that more points are present at the leading and trailing edges than in the middle, as expected. Also, the cosine discretisation does not only guarantee good resolution over the entire airfoil but also represents accurately its shape, being the only one of the three discretisations presented here that does so.

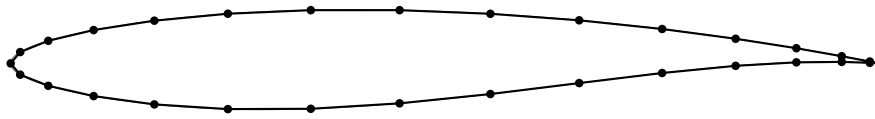


FIGURE 2.6. Airfoil Cosine discretisation (30 points)

Note that, even not representing the leading edge as well as the variation discretisation, it represents it sufficiently well.

2.3.3 Segment splitting

Besides the initial discretisation, the other mandatory procedure that a boundary interface must provide is segment splitting. This procedure is required for mesh refinement algorithms (Section 3.6) and consists of finding the midpoint of a segment.

For straight boundaries, the midpoint of a segment is just the point halfway from both ends. However, for curved boundaries, this notions are not so simple. One possibility is to locate the midpoint at the arc length. Other alternative uses the total variation of the tangent angle, where the midpoint is such that its value of $TV(\theta)$ is the average of the segment endpoints $TV(\theta)$ value. This second approach has the advantage of taking into account the curvature of the airfoil, and therefore inserting points where the curvature is higher, as desired.

For cases where the difference of $TV(\theta)$ between the endpoints is too small, using the arc length is preferable [5]. This ensures that straight boundaries are split as they normally would be, and prevents precision errors that might otherwise occur where the curved boundary is almost flat, possibly leading to over-refinement.

2.3.4 Point finding

For both variation discretisation and segment splitting, there is the need to find points with a specific value of $TV(\theta)$, hence the designation *point finding*.

Using only $TV(\theta)$ this task would be more time consuming since we have no knowledge of how $TV(\theta)$ varies along the airfoil. However, we know that two of the properties of critical points are that the variation of the tangent angle between consecutive critical points is at most $\frac{\pi}{2}$, and that between two

consecutive critical points there cannot exist two points with the same tangent angle. Therefore, once the pair of critical points between which the desired point is located is known, its tangent angle is also known, and one can formulate this search as finding the zero of a function (distance to the tangent angle of the desired point).

To perform point finding using only the information given by $TV(\theta)$, algorithms such as the Golden section search (Section 2.3.1) or the Bisection method can be used. To take advantage of the information that critical points give, and to use it in a function, Newton's method is also a possibility.

Newton's method

Newton's method is an algorithm used to find the roots (zeros) of a function, given its first derivative. It is an iterative method that evaluates the function and its derivative and moves along the function domain, to achieve a better approximation to the root at each step. This iterative process is defined as

$$t_{n+1} = t_n - \frac{h(t_n)}{h'(t_n)} \quad (2.20)$$

where $h(t_n)$ is the function for the zeros of which we want to compute. The algorithm stops when the difference between t_n and t_{n+1} is smaller than a predefined value. Applying this method to the process of point finding, described above, the function $h(t_n)$ and its derivative can be defined as follows

$$h(t) = \arctan(a'(t)) - \nu_{obj} \quad h'(t) = \frac{a''(t)}{a'(t)^2 + 1} \quad (2.21)$$

where $a(t)$ is the airfoil function and ν_{obj} the desired value of tangent angle.

Since the rate of convergence of Newton's method is quadratic, this would be a very efficient solution to point finding. However, the method presents some difficulties in cases where the derivative of the function is not well behaved. An example of a function whose derivative is not well behaved is $f(x) = \sqrt{x}$, as we happen to have in the Class function (Section 2.2.1). Thus, using Newton's method close to the leading edge can cause problems, and may not even give the correct results.

Bisection method

The bisection method is a method for finding the roots of a function. It is an iterative method that, at each step, divides an interval in half and selects of the two halves, the one that still contains the root of the function. An illustration of the method can be seen in Figure 2.7.

The method starts with two points p and q inside which the root of the function is known to exist, and determines its midpoint. Then, it is checked whether the signal of the function at the midpoint is the same as at p or at q , choosing the subinterval that contains the root and updating the points accordingly. The algorithm stops when the distance between p and q is smaller than a predefined value.

When compared to Newton's method for point finding this method is considerably slower. However, its termination is guaranteed, and the correct solution is always found.

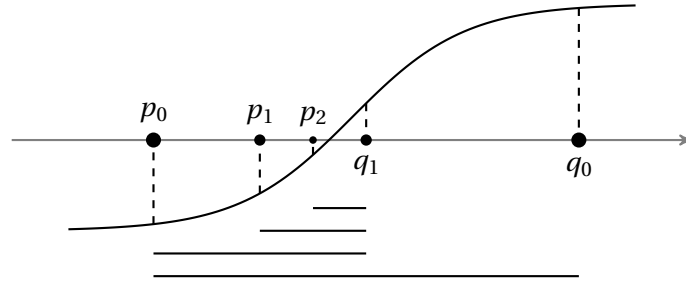


FIGURE 2.7. Bisection method

The midpoint q_1 has the same sign as q_0 , being the chosen subinterval the left one. Next iteration, the midpoint p_1 has the same sign as p_0 , and the chosen subinterval is the right one. Etc..

2.3.5 Curvature

Knowing the value of the curvature at a specific point in the airfoil can be another useful information to have on a boundary interface, since it allows the mesh refinement and control algorithms (Section 3.8.1, Equation 3.11) to place more points in regions where the curvature is higher, resulting in a better representation of the airfoil.

For a curve that is parametrically defined as $f(t) = (x(t), y(t))$ (e.g. NURBS), the curvature is given by

$$\kappa(t) = \frac{|x'(t)y''(t) - y'(t)x''(t)|}{(x'(t)^2 + y'(t)^2)^{3/2}} \quad (2.22)$$

On the other hand, when the curve is defined as $y = f(x)$ (e.g. CST),

$$\kappa(x) = \frac{|y''(x)|}{(1 + y'(x)^2)^{3/2}} \quad (2.23)$$

Chapter 3

Two-Dimensional Delaunay Triangulation and Refinement

Delaunay Triangulations have been extensively used in several fields of engineering and, in the specific case of shape optimisation they are of key importance. The main reason for this is that Delaunay Triangulations lead to meshes that optimise several geometric criteria. The concept of a Delaunay Triangulation was introduced by Boris Delaunay in 1934, and it corresponds to the dual graph of the Voronoi diagram, introduced by his Doctoral advisor, Georgy Voronoy.

This chapter reviews Delaunay Triangulations (DTs) and Constrained Delaunay Triangulations (CDTs) as well as their properties. It also explores the available methods and algorithms to construct them in a two dimensional space. Further on, methods designed to improve the quality of a DT and to allow additional control over its elements are also addressed.

3.1 Geometry Predicates and Functions

The following predicates and functions are useful to understand DTs and the algorithms for constructing them.

Orientation

Let a , b , and c be three vertices on the plane. The orientation predicate returns a positive value if vertices a , b , and c are arranged in a counter-clockwise order, zero if they are collinear, and a negative value otherwise.

$$\begin{aligned}\text{ORIENTATION}(a, b, c) &= \begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix} \\ &= (a_x - c_x)(b_y - c_y) - (a_y - c_y)(b_x - c_x)\end{aligned}\tag{3.1}$$

Vertex in the circumcircle of a triangle

Let a , b , c , and v be four vertices on the plane, such that a, b, c are counter-clockwise oriented, i.e. $\text{ORIENTATION}(a, b, c) > 0$. This test returns a positive value if v lies inside the circumscribed circle defined by a , b and c , a negative value if it lies outside, and zero if all four vertices lie on a common circle.

$$\text{INCIRCLE}(a, b, c, v) = \begin{vmatrix} a_x - v_x & a_y - v_y & (a_x - v_x)^2 + (a_y - v_y)^2 \\ b_x - v_x & b_y - v_y & (b_x - v_x)^2 + (b_y - v_y)^2 \\ c_x - v_x & c_y - v_y & (c_x - v_x)^2 + (c_y - v_y)^2 \end{vmatrix} \quad (3.2)$$

Segment crossing

This test consists of verifying whether two segments defined by two pairs of vertices intersect each other. Let a , b , c , and d be four vertices on the plane, and consider the two segments defined by \overline{ab} and \overline{cd} . The segment-crossing predicate returns TRUE if \overline{ab} and \overline{cd} intersect each other and FALSE otherwise.

To say that two segments intersect is the same as saying that, for each of the two segments, the two vertices of that segment lie on opposite sides of the other segment. For example, \overline{ab} and \overline{cd} intersect each other if c and d lie on opposite sides of \overline{ab} and a and b lie on opposite sides of \overline{cd} . With this in mind, we can use the Orientation predicate (Equation 3.1), since we know that if two vertices lie on opposite sides of a segment s , the multiplication of their orientations with s is ≤ 0 .

Note that even if a vertex of a segment lies on the other segment, the test returns TRUE. This also implies that if both segments share a vertex the return value is also TRUE. This test however, does not cover the case when all four vertices are collinear. In that case the test comes down to verifying whether a vertex of a segment lies on the other segment. Let

$$\begin{aligned} o_1 &= \text{ORIENTATION}(a, b, c) & o_2 &= \text{ORIENTATION}(a, b, d) \\ o_3 &= \text{ORIENTATION}(c, d, a) & o_4 &= \text{ORIENTATION}(c, d, b) \end{aligned}$$

then,

$$\text{CROSSING}(a, b, c, d) = \begin{cases} \text{TRUE} & \text{if } o_1 \cdot o_2 \leq 0 \text{ and } o_3 \cdot o_4 \leq 0 \\ \text{FALSE} & \text{otherwise} \end{cases} \quad (3.3)$$

Angle between segments

Let a , b and c be three vertices on the plane. This function returns the angle formed between \overline{ab} and \overline{bc} , i.e. the angle at b , or \overline{abc} . To calculate this angle we use the *dot product*, or *scalar product*, between \overrightarrow{ba} and \overrightarrow{bc} as well as their magnitudes.

$$\text{ANGLE}(a, b, c) = \arccos\left(\frac{\overrightarrow{ba} \cdot \overrightarrow{bc}}{\|\overrightarrow{ba}\| \cdot \|\overrightarrow{bc}\|}\right) \quad (3.4)$$

Circumcentre of a triangle

Let a , b and c be three vertices defining a triangle t on the plane. This function returns the coordinates of the circumcentre of t . This calculation can be simplified by translating the vertex a to the origin of the coordinate system. Let

$$\begin{aligned}
b' &= b - a \\
c' &= c - a \\
d' &= 2(b'_x c'_y - b'_y c'_x)
\end{aligned}$$

then,

$$\text{CIRCUMCENTRE}(a, b, c) = \left[\left(\frac{c'_y(b'^2_x + b'^2_y) - b'_y(c'^2_x + c'^2_y)}{d'} \right), \left(\frac{-c'_x(b'^2_x + b'^2_y) + b'_x(c'^2_x + c'^2_y)}{d'} \right) \right] \quad (3.5)$$

Circumradius of a triangle

Let a , b and c be three vertices defining a triangle t on the plane. The radius of the circumcircle of t may be computed either from the length of all sides of t , or from any of the three angles and the length of the opposite side. Let

$$\begin{aligned}
d &= \|a - b\| \\
e &= \|b - c\| \\
f &= \|c - a\|
\end{aligned}$$

then,

$$\begin{aligned}
\text{CIRCUMRADIUS}(a, b, c) &= \left(\frac{def}{\sqrt{(d+e+f)(-d+e+f)(d-e+f)(d+e-f)}} \right) \\
&= \left(\frac{d}{2 \sin(\text{ANGLE}(b, c, a))} \right)
\end{aligned} \quad (3.6)$$

Distance from a vertex to a segment

Let s be a segment on the plane, defined by two vertices a and b , and v be a vertex. This function returns the minimum distance between v and s . To compute the distance between a vertex v and a line it suffices to find the projection of v on that line. However, in this case we have not a line, but a line segment s , and thus, the projection of v may not lie on s . To address this problem we must verify where the projection of v lies. If it does not lie on the segment, then the distance to a or b is used, accordingly. Otherwise, the distance between a vertex and a line can be used. Let

$$t = \frac{(v_x - a_x)(b_x - a_x) + (v_y - a_y)(b_y - a_y)}{\|a - b\|^2}$$

then,

$$\text{DISTANCEVTOS}(a, b, v) = \begin{cases} \|a - v\| & \text{if } t < 0 \\ \|b - v\| & \text{if } t > 0 \\ \frac{|v_x(b_y - a_y) - v_y(b_x - a_x) + b_x a_y - b_y a_x|}{\|a - b\|} & \text{otherwise} \end{cases} \quad (3.7)$$

3.2 Data Structures

In many areas of Computational Geometry, and particularly in Mesh Generation, there are two main ways of storing mesh topologies: *edge-based* and *cell-based*. Cell-based structures usually store a list with all the cells present in the mesh (triangles in our case), and within each one, a number of pointers (three) that point to each cell's neighbours. Edge-based structures achieve the same in practice, but by different means.

One of the best known edge-based structures is the Doubly Connected Edge List (DCEL), first proposed by Muller and Preparata [22]. In this structure, an edge is treated as two *half-edges*, where each half-edge has a pointer to the next and previous half-edges of the same triangle. Note that these half-edges are positively oriented accordingly to Equation 3.1. This allows us to know all the vertices of a triangle given any of its half-edges, as well as the neighbouring triangles by using their twins. An illustration of this structure can be seen in Figure 3.1.

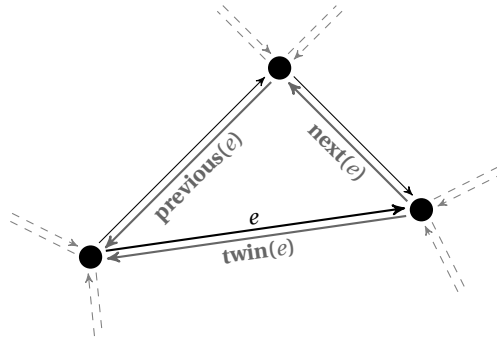


FIGURE 3.1. Doubly Connected Edge List and its Half-edges

In this dissertation the same idea and principles of the DCEL were used, although in a slightly different way. The structure that was created was a *Hash Table* (or Hash Map), where the half-edges represent the keys and the associated triangles represent the values [27]. The main reason to use a Hash table is the expected average time complexity of $\mathcal{O}(1)$. With this structure we can easily map the functions described for the DCEL. To access the next and previous half-edges we just need the triangles' third vertex, easily obtained once we know the triangle. To know the twin half-edge, and consequently the triangle's neighbours, we just need to search the Hash table with the inverted vertices.

Given this structure, an interface (proposed by Shewchuk [27]) composed by three functions, `ADDTRIANGLE`, `DELETETRIANGLE` and `ADJACENT`, comes in handy. The first two functions create and remove triangles specified by three positively oriented vertices. These operations are done by adding or removing groups of three entries, each representing one half-edge of the triangle, from the Hash table. The third function returns the third vertex of a triangle, requiring one half-edge as input. More information can be seen in Table 3.1.

ADDTRIANGLE(a, b, c)	Adds a positively oriented triangle $\triangle abc$
REMOVETRIANGLE(a, b, c)	Deletes a positively oriented triangle $\triangle abc$
ADJACENT(a, b)	Returns a vertex c such that $\triangle abc$ is a positively oriented triangle

TABLE 3.1. Data Structure interface

3.3 Properties of a Delaunay Triangulation

For a triangulation to be considered Delaunay its elements have to follow several rules and contain certain properties. In this section, these properties are considered for both triangles and edges, since constructing a DT is possible using either. The first and most obvious property that all elements of a DT must have is to be Delaunay.

DEFINITION 3.1. Delaunay [27]

- A triangle t is said to be *Delaunay* if its circumcircle encloses no vertex in the triangulation. Note that vertices can lie on the circumcircle and t still be Delaunay
- An edge e is said to be *Delaunay* if it has at least one circumcircle that encloses no vertex in the triangulation

A slightly stronger variation of the Delaunay property is “strongly Delaunay”.

DEFINITION 3.2. Strongly Delaunay [27]

- A triangle t is said to be *strongly Delaunay* if no vertex in the triangulation lies inside or on its circumcircle, except the vertices that compose t
- An edge e is said to be *strongly Delaunay* if it has at least one circumcircle such that no vertex in the triangulation lies inside it or on it, except the endpoints of e

Uniqueness As can be noticed from the previous definitions, the only difference between them is that for strongly Delaunay elements, it is not allowed for vertices to lie on the circumcircles. This has to do with the uniqueness property of DTs. For a certain set of vertices there is only one possible DT if this set has no four cocircular vertices. In that case, the triangulation is considered strongly Delaunay. If that is not the case and cocircular vertices are present, the set has multiple DTs. However they differ only in the elements that are merely Delaunay.

Although the previous properties can be applied to both triangles and edges, in practice they are usually applied to triangles only. However, there is one property specifically for edges that, depending on the chosen algorithm, can prove more useful than the previous ones when constructing a DT. Figure 3.2 illustrates this property.

DEFINITION 3.3. Locally Delaunay [27]

- An edge e is said to be *locally Delaunay* if
 - it is an edge of a single triangle
 - it is an edge of exactly two triangles, t_1 and t_2 , and has a circumcircle enclosing no vertex of t_1 nor t_2 . Equivalently, the circumcircle of t_1 encloses no vertex of t_2 . Equivalently, the circumcircle of t_2 encloses no vertex of t_1

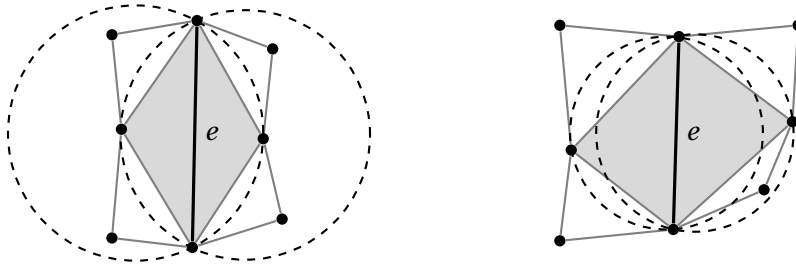


FIGURE 3.2. Locally Delaunay property

At left, edge e is not locally Delaunay, since the circumcircles of both its triangles enclose the opposite vertices. At right, edge e is locally Delaunay since that does not happen. Note that one of the circumcircles enclose a vertex of the triangulation. That makes e not Delaunay, but still locally Delaunay,

If every edge of the triangulation is locally Delaunay, then the triangulation is also Delaunay.

Although the properties presented above have their own significance and utility, they are merely different ways of expressing the same overall property, and can be unified by the Delaunay lemma, introduced by Boris Delaunay himself [27].

Delaunay Lemma The following three statements are equivalent [27]:

- Every triangle in the triangulation is Delaunay
- Every edge in the triangulation is Delaunay
- Every edge in the triangulation is locally Delaunay

Optimality The main reason why DTs are widely used to build quality meshes resides in their optimality guarantees. Among all the possible triangulations for a set of vertices, the Delaunay Triangulation maximises the smallest angle and minimises the largest circumcircle and min.-containment circle [27], leaving badly-shaped triangles out of the triangulation. The *min.-containment circle* of a triangle is the smallest circle that encloses its vertices. For a triangle with no obtuse angles, the circumcircle and the min.-containment circle are the same, but for an obtuse triangle, the min.-containment circle is smaller.

3.4 Building a Delaunay Triangulation

There are several methods to build a Delaunay Triangulation. Some take all the necessary input vertices at the beginning, while others take one vertex at a time. Methods of the first type begin with some initial valid triangulation (not yet Delaunay), and then apply an algorithm to make the necessary corrections until a fully DT is obtained. One of the best known algorithms to perform those corrections is the Flip Algorithm, detailed in Section 3.5.3. The second type of methods, referred to as incremental, always maintain a triangulation that is Delaunay, preserving its properties as vertices are inserted.

3.4.1 Inserting a Vertex

A vertex may be inserted into an existing DT using the *Bowyer-Watson* algorithm. This algorithm was created independently by Adrian Bowyer [6] and David Watson [28], so it is also mentioned as Bowyer algorithm or Watson algorithm.

To recall, for a triangulation to be considered Delaunay, no triangle circumcircle must enclose another vertex of the triangulation. If we insert a vertex inside the triangulation, we then know that it is within one triangle, and therefore inside the corresponding circumcircle. This means that simply inserting a new vertex into the triangulation violates its Delaunay properties, which need to be restored, e.g. using the Bowyer-Watson algorithm. Let v be a new vertex within the limits of the triangulation. The algorithm follows:

1. Locate one triangle whose circumcircle encloses v
2. Find other such triangles by a depth-first search
3. Delete all such triangles, creating a cavity inside the triangulation
4. For each cavity edge, create a new triangle with v

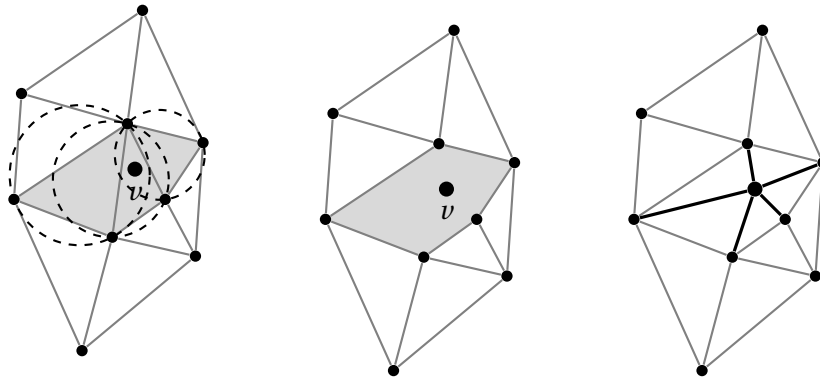


FIGURE 3.3. Bowyer-Watson algorithm

Triangles whose circumcircle encloses v are found using a depth-first search. Those triangles are deleted, forming a cavity. New edges from v to every point of the cavity are created.

The first step of the algorithm is what is called *point location*, discussed in Section 3.4.2. The second step performs a depth-first search through the triangulation searching for other triangles whose circumcircle also encloses v . The search is started at the triangle found in the first step. The third step deletes these triangles, leaving the triangulation with a cavity. The cavity is then filled with triangles created from the edges of the cavity and the new vertex v . Figure 3.3 illustrates the process, while Algorithm 3.1, described by Shewchuk [27], gives pseudocode for it.

Time complexity Although the Bowyer-Watson algorithm can take, in some special degenerate cases, $\mathcal{O}(n^2)$ time to perform, in the average case scenario its time complexity is $\mathcal{O}(n \log n)$ [27].

Vertex outside the triangulation When inserting a vertex there is also the possibility for it to lie outside the triangulation. One way to deal with this situation is by using *ghost points* [27]. Ghost points are said to be at infinity, and together with every edge of the triangulation boundary, form several *ghosts triangles*. That way we can ensure that every location has at least one triangle to work with. However, these ghost triangles need to be treated differently from the other ones, adding complexity to the algorithm. An alternative to this method is to use a *bounding box* that encloses all the vertices of the triangulation. Usually the shapes of this bounding box are triangular or rectangular.

ALGORITHM 3.1. Bowyer-Watson algorithm

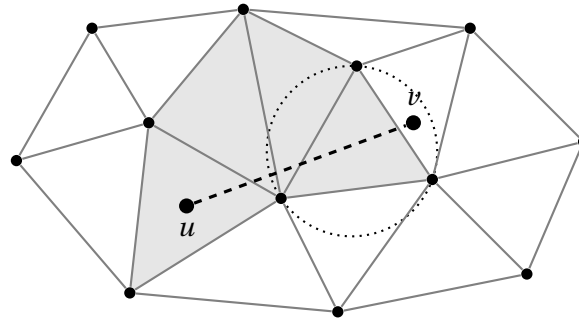
```

1: function INSERTVERTEX( $v, a, b, c$ )
2:   REMOVETRIANGLE( $a, b, c$ )
3:   DIGCAVITY( $v, b, a$ )
4:   DIGCAVITY( $v, c, b$ )
5:   DIGCAVITY( $v, a, c$ )
6: function DIGCAVITY( $v, a, b$ )
7:   if ISBOUNDARY( $a, b$ ) then
8:     ADDTRIANGLE( $v, b, a$ )
9:   return
10:   $c \leftarrow$  ADJACENT( $a, b$ )
11:  if  $c \neq \emptyset$  then
12:    if INCIRCLE( $a, b, c, v$ )  $> 0$  then
13:      REMOVETRIANGLE( $a, b, c$ )
14:      DIGCAVITY( $v, a, c$ )
15:      DIGCAVITY( $v, c, b$ )
16:    else
17:      ADDTRIANGLE( $v, b, a$ )

```

3.4.2 Point Location by Walking

Point Location can be the most important part of vertex insertion, since it can be the most complex and time consuming step [27]. To perform it one can use a method called *Walking* [3]. As can be seen in Figure 3.4, this method traces a line segment in the triangulation from one triangle to the new vertex. Then, the triangles are traversed along the line segment until one whose circumcircle encloses the new vertex is reached. Pseudocode for this process can be found in Algorithm 3.2.

**FIGURE 3.4.** Point location by Walking

A line segment is traced from the centroid, u , of the last modified triangle to the vertex, v , to be inserted. The triangles are traversed along that line segment.

In practice this algorithm has good time results if the vertices are inserted in the triangulation in a way to minimise the sum of the distance between consecutive vertices. If so, the algorithm can start at the last created triangle, making the traversal of the triangulation as short as possible. In this dissertation, since the vertices are only located along the airfoil, they are inserted as we traverse it.

Time complexity This algorithm may take linear time, $\mathcal{O}(n)$, for each vertex processed. However, if the order in which the vertices are inserted is carefully chosen, as mentioned above, it has better results in practice.

ALGORITHM 3.2. Walking algorithm

```

1: function WALKING( $v, a, b, c, u$ )
2:   first  $\leftarrow$  TRUE
3:   while INCIRCLE( $a, b, c, v$ )  $\leq 0$  do
4:     if first and CROSSING( $a, b, u, v$ ) then
5:        $a, b \leftarrow b, a$ 
6:     else if CROSSING( $b, c, u, v$ ) then
7:        $a \leftarrow c$ 
8:     else
9:        $b \leftarrow c$ 
10:    first  $\leftarrow$  FALSE
11:  return  $a, b, c$ 

```

To verify the correctness of the algorithm we need to make sure that it terminates, i.e. that it reaches a triangle whose circumcircle encloses the new vertex. The following sentences demonstrate that.

- **The algorithm always has a possible ending:** If the vertex is inserted inside the triangulation there is at least one triangle whose circumcircle encloses the new vertex;
- **There is always a valid path from the starting triangle to a final triangle:** If the triangulation is convex and it does not contain cavities in its interior, every line segment defined by whatever two vertices located inside the triangulation always has triangles on its full path;
- **The path can always be traversed from its start to finish:** In this final proof a special case arises when the path intersects one or more of the vertices already present in the triangulation (Figure 3.5), resulting in the possibility of a triangle being intersected by the path on its three sides, instead of the usual two. These cases are treated separately:
 - **Default case:** If the triangle is intersected twice it means that it has only one entry and one exit. This is guaranteed by never considering the entry edge whenever a triangle is visited. Since the path is a line segment, a triangle is visited only once, meaning that for a certain path, with a finite number of triangles, the algorithm always terminates;
 - **Special case:** If this occurs, there is the possibility for the algorithm to enter a cycle within the triangles that share the vertex intersected by the path. This is avoided by always checking the edges of a triangle in the same oriented order (Equation 3.1), which guarantees that at some point in the set of triangles, an edge that breaks the cycle is checked.

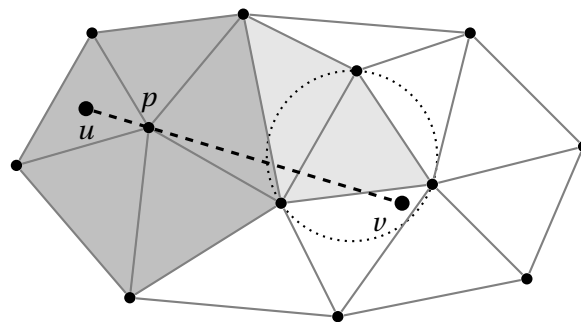


FIGURE 3.5. Point location by Walking (special case)
Once the traced line segment passes through p , there exists the possibility for the algorithm to get stuck in a cycle (dark gray triangles)

3.5 Building a Constrained Delaunay Triangulation

A Constrained Delaunay Triangulation (CDT) adds the possibility of having segments and/or enforced vertices inside the triangulation. A set of such elements is called a Planar Straight-Line Graph (PSLG), and is the input to a CDT. One requirement for the segments present in the PSLG is that they do not intersect one another.

In a CDT, triangles only need to have vertex-free circumcircles if there exists no input segment between them. This implies a visibility test that needs to be made each time a vertex is inserted. However, this test can be avoided with a simple modification to the depth-first search shown in Algorithm 3.1 [27]. Previously, the search was stopped once an edge of the bounding box is reached, preventing the algorithm to go out of the boundaries. Taking advantage of that, we just need to add to the same check the possibility for the edge to be a segment of the PSLG, thereby removing the need for a visibility test.

3.5.1 Inserting a Vertex

The insertion of a vertex in a CDT is similar to the unconstrained case. The only difference is at the end of the Point Location algorithm. While in the original algorithm the return object is just one triangle whose circumcircle encloses the new vertex, in the constrained version it may be necessary to return two triangles if the vertex lies on top of a segment of the PSLG. Some implementations require that the PSLG does not contain such cases (using two segments with the vertex in common, instead of a segment with the vertex on it), but the solution is easy to implement.

The problem in this case is that the depth-first search only checks half of the triangles due to the visibility modification, explained above. To solve this, two searches are made, starting at both triangles that share the segment [27].

3.5.2 Inserting a Segment

To convert an unconstrained DT into a constrained one, the segment insertion is usually preferred. This method requires an existing DT, that can be built using vertex insertion in an unconstrained triangulation, as it was shown in Section 3.4.1.

To insert a segment is basically the same as forcing certain edges to be present. If we take as input an unconstrained triangulation, it can happen that those edges are already present, and no modifications need to be done. However, if those edges are not present they need to be somehow inserted.

To insert a segment into the triangulation there are two main alternatives. One of them uses the *Flip* algorithm, while the other uses the *Gift-Wrapping* algorithm. The Flip algorithm is also used to build unconstrained DTs (the first type of algorithms discussed at the beginning of Section 3.4), and needs an initial triangulation to work with. The Gift-Wrapping algorithm can construct triangulations from scratch.

If we choose to use the Gift-Wrapping algorithm, segments are forced into the triangulation, which causes several triangles to be deleted, namely those that intersect the inserted segment. Once again, a cavity is created inside the triangulation, but unlike vertex insertion, in segment insertion there is no vertex in its interior, and therefore the triangles need to be created in some other way.

3.5.3 The Flip Algorithm

The purpose of the Flip algorithm is to convert a regular triangulation into a DT or CDT by flipping edges [27]. It starts with a regular triangulation (not necessarily Delaunay) and maintains a list of edges that may not be locally Delaunay. Initially, this list contains all edges of the triangulation.

The flip algorithm repeats a procedure called *flipping* until the edge list is empty. At each flipping step (Figure 3.6), an edge is removed from the list and checked. If it still exists, and is not locally Delaunay, then it is flipped, and the four edges of the enclosing quadrilateral are added to the list. Once this algorithm terminates, the triangulation is guaranteed to be Delaunay. The pseudocode for the algorithm is given in Algorithm 3.3.

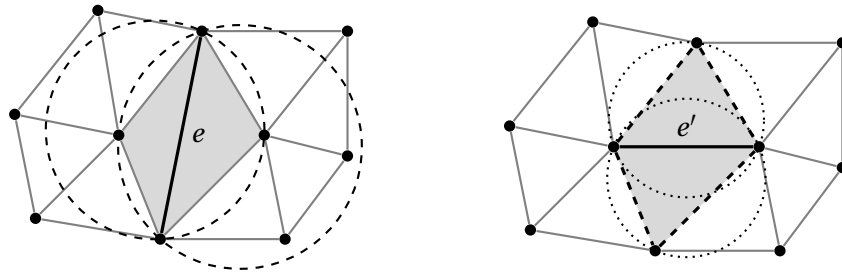


FIGURE 3.6. Flipping step

Edge e is not locally Delaunay, since the circumcircle of both its triangles enclose the third vertex of the respective opposite triangles. Edge e is flipped, originating e' . Dashed edges are then marked for verification.

ALGORITHM 3.3. Flip algorithm

```

1: function FLIP(list)
2:   while not list.ISEMPTY() do
3:      $e = \text{list.POP}()$ 
4:      $v_3 \leftarrow \text{ADJACENT}(e_1, e_2)$ 
5:     if ISBOUNDARY( $e$ ) or ISSEGMENT( $e$ ) or  $v_3 = \emptyset$  then
6:       continue
7:      $v_4 \leftarrow \text{ADJACENT}(e_2, e_1)$ 
8:     if INCIRCLE( $e_1, e_2, v_3, v_4$ )  $> 0$  then
9:       REMOVETRIANGLE( $e_1, e_2, v_3$ )
10:      REMOVETRIANGLE( $e_2, e_1, v_4$ )
11:      ADDTRIANGLE( $e_1, v_4, v_3$ )
12:      ADDTRIANGLE( $e_2, v_3, v_4$ )
13:      list.PUSH( $e_1, v_4$ )
14:      list.PUSH( $v_4, e_2$ )
15:      list.PUSH( $e_2, v_3$ )
16:      list.PUSH( $v_3, e_1$ )

```

Time complexity The Flip algorithm usually runs in $\mathcal{O}(n + k)$ time, where n is the number of edges in the triangulation and k is the number of flips performed. In a worst case scenario, this algorithm can take $\mathcal{O}(n^2)$ running time [27].

As it stands, the flip algorithm only creates a DT. However, it can be adapted to build a CDT. During the process previously described, at the verification step, edges that belong to the PSLG do not need

to be checked, since they are forced into the triangulation. This modification, by itself, keeps some of the segments of the PSLG in the triangulation. The remaining segments are inserted by flipping the edges that are intersected by them. These edges can be easily found, since they are the opposite edges to the vertices shared by their triangles and the segments. Once this process terminates, the final triangulation is guaranteed to be Constrained Delaunay.

3.5.4 The Gift-Wrapping Algorithm

The Gift-Wrapping algorithm consists of creating triangles at the front of unfinished edges, i.e. edges that still have one side left to be filled with triangles. Although this algorithm might fail when building a triangulation from scratch (if there are four cocircular vertices in the input), it does not fail when applied to segment insertion [27]. The first step of the segment insertion, the creation of a cavity, can be seen in Figure 3.7. The purpose of the gift-wrapping algorithm is to fill the cavity with triangles.

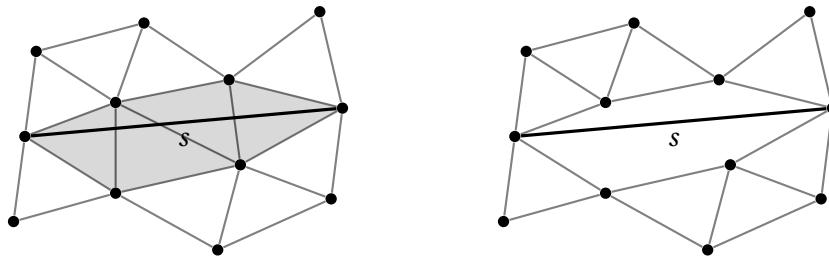


FIGURE 3.7. Segment insertion prior to the gift-wrapping algorithm

All the triangles that are intersected by s are removed from the triangulation, originating a cavity, later to be filled.

For each side of the inserted segment, the algorithm operates as follows. Let s be the inserted segment, s_1 and s_2 its endpoints, and V the set of vertices of the cavity hull on one side of s (excluding s_1 and s_2). Vertices in V are ordered as we traverse the cavity hull. For each vertex v in V is computed the signed distance from the circumcenter of the triangle composed by s_1 , s_2 and v to s . This distance is positive if the circumcenter lies on the side of s currently considered, and negative otherwise. The vertex that leads to the lowest value is chosen to form a triangle with s . The procedure can also be seen as a circumcircle formed only by s_1 and s_2 that starts expanding in front of s , stopping when it reaches a vertex.

The algorithm is then called recursively for both edges $\overline{s_1 v}$ and $\overline{v s_2}$ (if unfinished). The algorithm ends when there are no more unfinished edges, and the resulting triangulation is guaranteed to be Constrained Delaunay. A representation of this procedure can be found in Figure 3.8 while the pseudocode is given by Algorithm 3.4.

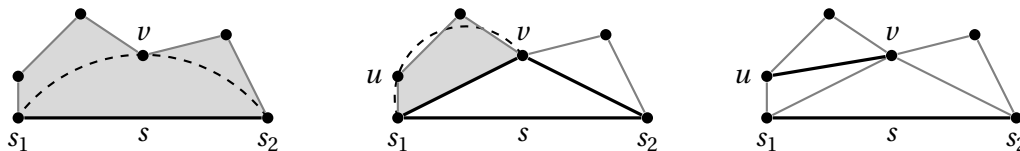


FIGURE 3.8. Gift-Wrapping algorithm

v is chosen to form a triangle with s . Procedure is recursively called for both $\overline{s_1 v}$ and $\overline{v s_2}$. $\overline{v s_2}$ only has one vertex available and creates a triangle with it. $\overline{s_1 v}$ has two vertices to choose and forms the triangle with u . Procedure is only recursively called for $\overline{u v}$, since $\overline{s_1 u}$ is not an unfinished edge.

ALGORITHM 3.4. Gift-Wrapping algorithm

```

1: function GIFT-WRAPPING( $s_1, s_2, V, i_1, i_2$ )
2:    $d_{min} \leftarrow +\infty$ 
3:   for  $i = i_1 \rightarrow i_2$  do
4:      $c \leftarrow \text{CIRCUMCENTER}(s_1, s_2, V[i])$ 
5:      $d \leftarrow \text{DISTANCEVTOS}(V[i], s_1, s_2)$ 
6:     if  $\text{ORIENTATION}(s_1, s_2, c) < 0$  then
7:        $d \leftarrow -d$ 
8:     if  $d < d_{min}$  then
9:        $d_{min} \leftarrow d$ 
10:       $i_{best} \leftarrow i$ 
11:   $\text{ADDTriangle}(s_1, s_2, V[i_{best}])$ 
12:  if  $i_{best} \neq i_1$  then
13:    GIFT-WRAPPING( $s_1, V[i_{best}], V, i_1, i_{best} - 1$ )
14:  if  $i_{best} \neq i_2$  then
15:    GIFT-WRAPPING( $V[i_{best}], s_2, V, i_{best} + 1, i_2$ )

```

Time complexity As long as the recursive calls tend to divide the set V into two subsets of roughly the same size, the Gift-Wrapping algorithm completes in $\mathcal{O}(n \log n)$ time. In the worst case scenario, where the set V is never divided into two subsets at each recursive call, the algorithm takes $\mathcal{O}(n^2)$ time to complete [27].

3.6 Delaunay Refinement Algorithms

When the expected result is a guaranteed-quality triangulation, a simple DT is not enough. To improve it, we use Delaunay refinement algorithms, which operate by inserting vertices in the triangulation while maintaining its Delaunay or Constrained Delaunay properties, until some constraints are satisfied.

Quality constraints Usually these constraints are related to the minimum angle of all triangles in a triangulation. A smaller minimum angle causes the presence of *poor-quality* elements, such as skinny triangles. The main purpose of refinement is to remove those, producing a better quality triangulation. Another way to measure the quality of a triangle is through its *radius-edge* ratio. This method is perhaps the most important one, since it is related to how Delaunay refinement algorithms work [21] and it helps in proving their effectiveness. The radius-edge ratio of a triangle is defined as its circumradius divided by its shortest edge. For two dimensions it is also directly related to the triangle's minimum angle by the formula

$$\frac{r}{e_{min}} = \frac{1}{2 \sin(\theta_{min})} \quad (3.8)$$

Over-refinement Although using quality constraints usually improves the triangulation, if these are too restrictive, some undesired effects may occur. One outcome of setting the minimum angle constraint, B , too high is that at some point the refinement algorithm worsens the triangulation more than it improves, which often results in the appearance of clusters composed by very small triangles. Since each triangulation is unique, the limit for this constraint might vary, although it usually takes values around and a little above 30° .

Something that almost all Delaunay refinement algorithms have in common is the method used to improve the triangulation. When a skinny triangle t appears, it needs to be removed somehow. To achieve this, a vertex v is inserted at its circumcenter. This process is commonly referred as *triangle splitting* [27]. Once v is inserted, t is no longer Delaunay, and thus it is removed by the insertion process. With the help from the radius-edge ratio it can also be proved that, at each step of the algorithm, the triangles resulting from splitting t are no worse than t [27].

In this section we review two algorithms, developed by Ruppert and Chew, as well as an adaptation made by Shewchuck.

3.6.1 Ruppert's Delaunay Refinement Algorithm

In 1992, Jim Ruppert published an algorithm [25] for producing quality triangulations that are both *well-graded* and *size-optimal*.

Good-grading Good-grading relates to the gradation of triangle size when moving away from boundaries. Unlike previous algorithms that produced mostly uniform triangulations, Ruppert's algorithm has this property, making it one of the first algorithms to be satisfactory in practice.

Size-optimality To say that a Delaunay refinement algorithm is size-optimal is to say that, for a given bound B on the minimum angle, the algorithm produces a triangulation whose number of triangles is at most a constant factor larger than the size of the smallest possible triangulation that meets the same angle bound.

Ruppert's algorithm starts with a DT or a CDT. New vertices are inserted at the circumcenter of badly-shaped/poor-quality triangles. Insertion is done using the Bowyer-Watson algorithm, which maintains the Delaunay properties of the triangulation. This is repeated until all triangles satisfy the constraints initially set. Like previous algorithms, Ruppert's has the possibility of splitting each segment into subsegments. To fully understand the vertex insertion process, some notions need to be introduced.

DEFINITION 3.4. A *diametral circle* is the smallest (unique) circle that encloses a segment s . In other words, it is the circle that has s as its diameter.

DEFINITION 3.5. A segment is said to be *encroached* if a vertex, other than its endpoints, lies on or inside its diametral circle and is visible from the interior of the segment.

Encroached segments are given priority over poor-quality triangles. If in the process of splitting a triangle, its circumcenter would encroach upon any segment, then the vertex is not inserted. Instead, all the segments it would encroach are split. If during the segment splitting process the triangle is removed from the triangulation, the vertex is not inserted afterwards. A representation of the process can be seen in Figure 3.9. This algorithm guarantees that no circumcenter of any triangle can lie outside the triangulation.

The most interesting aspect about Ruppert's algorithm is that, even if it starts with a CDT, the final triangulation is Delaunay, not just Constrained Delaunay.

Jim Ruppert proved that his algorithm produces well-graded and size-optimal triangulations with no angles smaller than 20.7° [25]. However, the algorithm usually terminates for angle bounds higher than this guaranteed value.

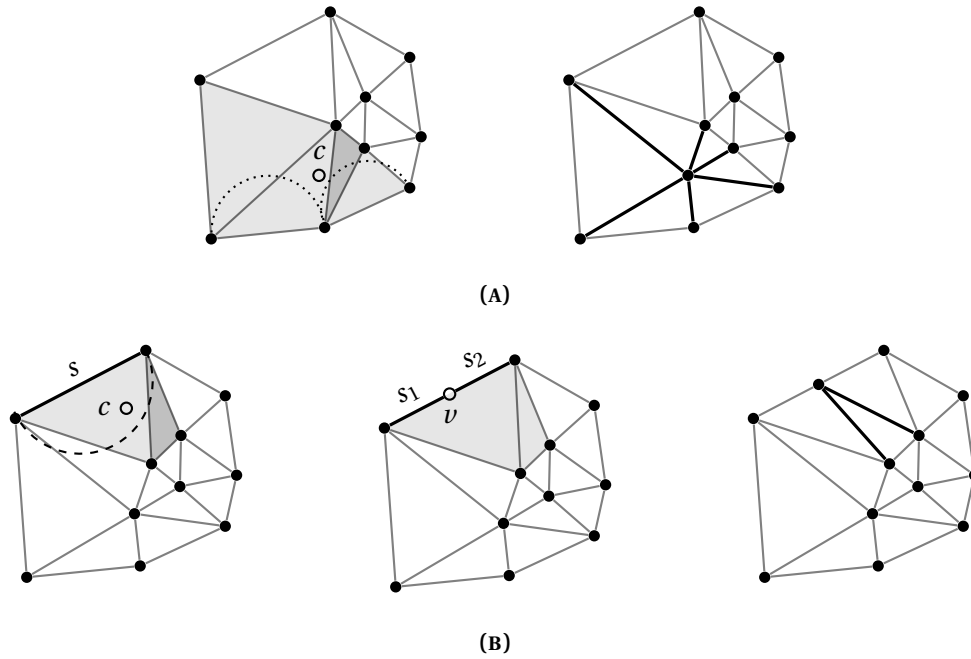


FIGURE 3.9. Steps of the Ruppert's refinement algorithm and segment splitting
 At (A), a poor-quality triangle is found. Since its circumcenter c does not encroach any segment, it is inserted. At (B), another poor-quality triangle is found, but now its circumcenter c encroaches a segment s , forcing it to be split into two smaller subsegments, s_1 and s_2 . Splitting vertex v is then inserted. Since the poor-quality triangle has been removed, there is no need to insert c anymore.

3.6.2 Chew's Second Delaunay Refinement Algorithm

Paul Chew's first refinement algorithm [8] predates the one from Ruppert. However it was not reviewed here since it produces uniform triangulations. The second one [9] was published slightly after Ruppert's, and yields better results. When compared to Ruppert's, it improves the guarantee of good-grading in theory, and splits fewer segments in practice.

Chew's second Delaunay refinement algorithm starts with a CDT, and eliminates poor-quality triangles by inserting a vertex at its circumcenter. However, Chew's algorithm does not make use of diametral circles to check the encroachment of a segment. Instead, vertices are added regardless of encroachment unless it happens that, when splitting a poor-quality triangle t , its circumcenter c lies on the opposite side of a segment s .

When this happens, the insertion of c would not cause t to be removed, due to visibility issues. Thus, c is rejected for insertion, and all vertices enclosed by the diametral circle of s are deleted. This alone causes t to be removed. Then, s is divided into two new subsegments. Figure 3.10 illustrates the process. If more than one segment lie between t and its circumcenter c , only the segment closest to t is split.

Unlike Ruppert's refinement algorithm, Chew's final triangulation is just Constrained Delaunay. However, this has its advantages. Since it allows the insertion of vertices closer to the boundaries, it avoids several segment splits that would otherwise occur, producing fewer triangles. It also produces well-graded and size-optimal triangulations for minimum angle bounds up to 26.56° [27], when compared to Ruppert's 20.7° .

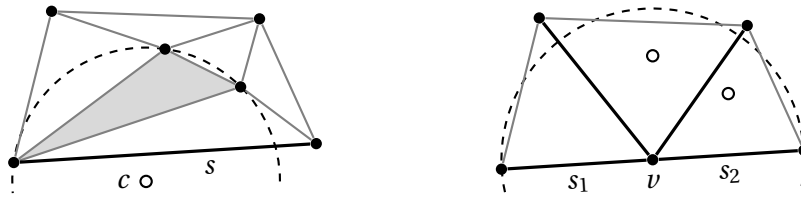


FIGURE 3.10. Step of the Chew's refinement algorithm and segment splitting
A triangle and its circumcenter c lie on opposite sides of a segment s . The vertices enclosed in the circumcircle of s are removed and s is split into s_1 and s_2 .

3.6.3 Shewchuk's Diametral Lenses

Jonathan Shewchuk claimed [26] that Chew's second Delaunay refinement algorithm is equivalent to a variant of Ruppert's algorithm in which diametral circles are replaced with *diametral lenses*. These lenses are depicted in Figure 3.11.

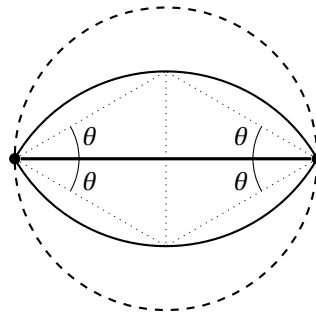


FIGURE 3.11. Comparison between a diametral circle and Shewchuk's diametral lenses

In this Ruppert/Chew hybrid algorithm, if the circumcenter c of a poor-quality triangle t lies on or inside the diametral lens of a segment s and c is visible from the interior of s , all vertices located inside the diametral circle of s are removed, and s is split. With lenses where θ is at least 30° , either a vertex of t or its circumcenter c lies inside the lens [26]. However, with diametral lenses with an angle smaller than 30° it might happen that neither a vertex of t or its circumcenter lie inside the diametral lens, despite being on opposite sides of s . In this case, one must use Chew's verification for segment encroaching.

Compared to Ruppert's algorithm, the use of diametral lenses has the disadvantage that it does not guarantee that the final triangulation is Delaunay, but only Constrained Delaunay. However, since diametral lenses are smaller than diametral circles, it has the advantage of avoiding segment splits that otherwise would occur.

The advantage of using diametral lenses in Chew's algorithm is that any vertex that would normally be inserted inside the diametral lens of some segment would produce poor-quality triangles. Therefore, by rejecting these vertices, the creation and subsequent removal of these triangles can be avoided, speeding up the refinement algorithm. Diametral lenses are usually defined with an angle θ equal to the minimum angle bound B . θ can also be defined with angles larger than B , yet it makes little sense in making θ smaller than B , since it would allow the insertion of vertices that would be deleted later on.

Shewchuck showed that using diametral lenses it is possible to achieve a triangulation with guaranteed good-grading and size-optimality for minimum angles up to 25.65° [27].

3.7 Application to Curved Boundaries

Until now the methods described in this chapter to produce triangulations only contemplate straight lines as boundaries. When considering curved boundaries, as it is the case of this dissertation, some precautions need to be taken. As may be recalled from Chapter 2 and more specifically from Section 2.3, methods to obtain an initial discretisation and perform segment splitting concerning curved boundaries were presented, as well as the restrictions they must comply with. Now that we have the knowledge of how Delaunay refinement algorithms work, the reasons for these restrictions can be explained and more easily understood.

Since the curves are represented by straight line segments, a problem that can arise is that of a vertex being inserted inside the triangulation (inside the segments) but outside the actual curve. In 2002, Boivin and Ollivier-Gooch [5] addressed this problem, introducing at the same time the concept of total variation of the tangent angle, $TV(\theta)$, already explained in Section 2.3. In their work they have considered the use of Shewchuk's lenses with $\theta = 30^\circ$.

The solution they came up with is to use Shewchuk's lenses to protect the curve, i.e. contain the curve represented by a segment in the interior of its lens. In that way, since no vertex can be inserted inside the segment's lens, no vertex can lie outside the curved boundary. From their analysis they have concluded that for a curve with uniform curvature (e.g. arc circle), the maximum allowed value of $TV(\theta)$ between consecutive vertices on the boundary must be $\frac{\pi}{3}$, while for a curve with non-uniform curvature this value must be $\frac{\pi}{6}$. These values are used as $TV(\theta)_{max}$ in Equation 2.18 for the initial discretisation.

The segment splitting process for curved boundaries does not use the midpoint of a segment. Instead, the process is governed by $TV(\theta)$, as described in Section 2.3.3.

3.8 Cell Size and Grading Control

To control the triangulation, whether the size of its elements or their gradation, refinement algorithms usually rely only on the restriction of minimum angle. However, it is often necessary in real world applications for the elements of the triangulation to be smaller and achieve a better gradation than the refinement algorithms alone provide, so as to obtain a better representation of the mapped space. Further mechanisms to improve control over the triangulation are explored next.

3.8.1 Ollivier-Gooch and Boivin's Length Scale

In 2001, Ollivier-Gooch and Boivin proposed a method to control the elements of the triangulation without sacrificing guarantees of quality, good-grading and size-optimality [24]. They introduced a definition of *length scale* based on a previous concept introduced by Ruppert called *local feature size*.

Local feature size The local feature size of a vertex v , $lfs(v)$, can be defined as the radius of the smallest disk centred at v that touches two disjoint parts of the domain boundary, or PSLG [27].

The length scale of an input vertex, i.e. a vertex present in the PSLG is defined as

$$LS(v) = \frac{lfs(v)}{R} \quad (3.9)$$

where R is a value that gives control over the resolution of input features. To achieve uniform resolution over the entire the domain, R must be defined as a constant. It can be dynamic, otherwise. To have control over the triangles gradation, i.e. how fast triangles can grow when moving away from boundaries, they proposed a new definition of length scale for vertices inserted during the refinement algorithm:

$$LS(v) = \min\left(\frac{lfs(v)}{R}, \min_{\text{neighbours } u_i} \left(LS(u_i) + \frac{\|u_i - v\|}{G}\right)\right) \quad (3.10)$$

In this new formulation, G is a constant. With this definition, the length scale of v cannot exceed that of any of its neighbour u_i by more than the distance between v and u_i divided by G .

In a later work [5], Boivin and Ollivier-Gooch have modified the definition of local feature size to support curved boundaries as part of the PSLG. They have defined the local feature size for vertices located at curved boundaries to be:

$$lfs_c(v) = \min(\rho(v), lfs(v)) \quad (3.11)$$

where ρ is the radius of the curvature of the boundary at v . The formulation is applicable to both input and new vertices. This allows a smaller local feature size value at vertices where the curvature is higher, causing the insertion of more vertices on those regions, as desired, and leading to a better representation of the boundary.

Following this scheme, a triangle is marked for removal if its circumradius divided by the average length scale of its vertices is larger than a certain bound H . The adopted value for H was $\frac{\sqrt{2}}{2}$.

3.8.2 Gosselin's Modifications for Curved Boundaries

Subsequently to Ollivier-Gooch and Boivin's work on length scale, Serge Gosselin proposed a modification to the way this metric was used in the case of curved boundaries [14]. The main problem with the previous work is that the computation of the length scale of a new vertex v had to consider the local feature size at v (first half of Equation 3.10). Since $lfs(v)$ is computed with respect to the PSLG given at the beginning, this value can become deprecated for new vertices, since the input segments given in a PSLG may no longer exist as such due to segment splitting.

Gosselin proposed the use of *subcurves* to represent the boundaries of the triangulation. In that case the local feature size of a vertex v is not computed using the segments of the PSLG. Instead, it uses the distance from v to the actual subcurves, rather than their discretisation.

Gosselin also referred to segment splitting as *subcurve splitting*. This process is governed by the total variation of the tangent angle, $TV(\theta)$, as proposed by Boivin and Ollivier-Gooch. However, as opposed to a maximum total variation of $\frac{\pi}{3}$ or $\frac{\pi}{6}$, Gosselin suggests a maximum of $\frac{\pi}{2}$ or $\frac{\pi}{4}$, respectively. This allows the curve to lie outside the lens, no longer being protected by it. Nevertheless, it lies inside the diametral circle of a segment, causing the vertices inserted in that region in the meantime to be deleted whenever the segment is split [14].

To compute the length scale of input vertices, Gosselin does not take into account the curvature of the boundary at that point (Equation 3.11). Instead, only Equation 3.9 is used. As opposed to the

original scheme, when a new vertex is inserted during refinement, its length scale is computed using only the second half of Equation 3.10, i.e.

$$LS(v) = \min_{\text{neighbours } u_i} \left(LS(u_i) + \frac{\|u_i - v\|}{G} \right) \quad (3.12)$$

Whenever a subcurve s is split, the length scale of the splitting vertex v is computed using a separate method. A ceiling value is first set by linear interpolation. Using the length scale of the endpoints of s it is possible to define the maximum possible length scale for v by the position of v along the arc length. The length scale is then taken to be the minimum between the interpolated value and the one obtained by Equation 3.12.

Chapter 4

Mesh Reconstruction for Two Dimensions

In the context of shape optimisation, methods for adapting an existing mesh to a new shape play a key role. The objective is mainly to reduce time needed to evaluate modified shapes, allowing the user to obtain a mesh for the new model based on the previous mesh, instead of building a new mesh from scratch. Other factors, such as the preservation of certain features, or the similarity between meshes can be important as well, although they always depend on the subsequent processing to be performed in order to fully evaluate the design.

In this chapter we propose a new method for mesh reconstruction in two dimensions. It is expected that the method be capable not only to reduce processing time (when compared to building a completely new mesh) but also to preserve as many elements (triangles) as possible across model modifications and not that the total number of triangles does not increase too much in comparison to a completely new mesh. It is also intended for the method to maintain the quality of the mesh as well as its gradation.

First, we review several alternative methods based on mesh deformation (not mesh reconstruction!) that have been studied in the past few years, their advantages and disadvantages. Next, we explain the proposed method for mesh reconstruction. At the end, we explore the difficulties and downsides of the method, as well as the improvements that can be made to prevent them.

4.1 Mesh deformation

Mesh deformation algorithms have been extensively studied for the past twenty years, and most of them are applied in the context of fluid-dynamics optimisation. They can be roughly divided into two categories [16]: Laplacian smoothing methods and mechanical analogy methods, spring-analogy methods being the most well-known. In this section, we explore two spring-analogy methods, the tension spring method and its successor, the torsional spring method.

4.1.1 Tension spring method

In 1990, Batina proposed a mesh deformation method [4] for two-dimensional unstructured meshes where each edge is replaced by tension springs whose stiffness is inversely proportional to its length. These tension springs, illustrated in Figure 4.1, are also referred to as linear springs, and the method as linear spring method.

After computing the stiffness for every edge present in the mesh, and given the translation applied to the boundary vertices, a set of equations is solved iteratively until all the forces applied on these tension springs are in equilibrium.

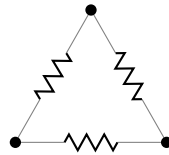


FIGURE 4.1. Illustration of tension/linear springs

Despite preventing vertices from colliding with each other due to the stiffness that is present between them, this method does not prevent a vertex from crossing its opposite edges, giving rise to what is called negative areas, i.e. triangles with negative orientation (orientation should normally be positive). Upon large modifications on the boundary, or for complicated geometries, this method often fails [7].

4.1.2 Torsional spring method

In order to provide the spring analogy with a more robust method in two dimensions, Farhat proposed in 1998 [11] the use of additional torsional springs around vertices to prevent them from crossing edges. An illustration of these torsional springs can be seen in Figure 4.2.

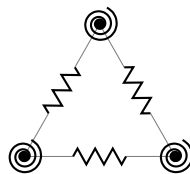


FIGURE 4.2. Combined tension/torsional springs

The stiffness of these torsional springs is inversely proportional to the sine of the angle for a specific triangle, causing each spring to have multiple values of stiffness. When a vertex moves towards an edge, the angle goes towards zero, increasing the force applied on the torsional spring, and thus guaranteeing a final mesh without negative area elements [7]. Just like the previous method, a set of equations is then solved until the forces applied in both tension and torsional springs are in equilibrium.

Farhat demonstrated that, by using an additional set of springs, the method is much more robust than Batina's, being capable of handling very large deformations.

4.1.3 Observations

Unlike mesh reconstruction methods, mesh deformation methods (also known as mesh morphing in the field of Computer Graphics) do not delete elements from the mesh. Instead, all elements are maintained between designs, and their position is adjusted in accordance with changes made to the design, which can propagate throughout the whole mesh. Therefore, elements are in general not preserved.

Relatively to the mesh reconstruction method that we propose, these methods also have the disadvantage of not necessarily maintaining a Delaunay Triangulation after the modifications, possibly causing the process of improving the mesh to be much more difficult.

4.2 New reconstruction process

The new method for mesh reconstruction is presented here. It can be divided into four distinct stages: Boundary adjustment, Interior building, Removal and Rebuilding. The execution of the last three stages depends on the result of the first stage. If the Boundary adjustment is concluded successfully, the other three stages can be skipped, further accelerating the process of reconstruction. Independently of what is actually performed during the reconstruction process, an additional Delaunay refinement step is always required to guarantee the quality of the mesh.

4.2.1 Boundary adjustment

If the modifications made to the airfoil are sufficiently small, there should be no need to remove any triangles from the mesh. What the boundary adjustment algorithm does is to traverse the boundary of the mesh, checking whether boundary triangles still preserve their quality properties when modified. Here, the only modification considered is the new value of η of boundary vertices for the same value of ψ .

For a modified triangle to pass the test, it has to be Constrained Delaunay and respect the minimum angle bound. The reason to consider the minimum angle bound is because, otherwise, it could give rise to bad quality triangles close to the boundary, possibly leading to over-refinement. To verify if it is Constrained Delaunay, one can use the local Delaunay test on the edges that are not segments. For the algorithm to complete successfully, all triangles must maintain these properties. If that is the case, the triangles are modified and the resulting triangulation is guaranteed to be Constrained Delaunay. Otherwise, if a triangle that does not maintain the properties is detected, the algorithm stops without checking the remaining triangles. In this case, removal of triangles is needed.

4.2.2 Interior building

From one iteration to another, it can happen that vertices of the new airfoil come to lie inside the old one. Therefore, and since the removal process is based on local search (as in vertex insertion, Section 3.4.1), it is necessary to fill the interior of the old airfoil with triangles. These triangles do not need to be good-quality ones or even Constrained Delaunay, since they are marked for later deletion immediately after creation. They just need to be present and provide the removal algorithm with a valid triangulation.

To create such a triangulation, one can use a lexicographic technique. To build a lexicographic triangulation, the available vertices are sorted by their coordinates and the triangles are formed while traversing the sorted vertices. An example of a lexicographic triangulation can be seen in Figure 4.3.

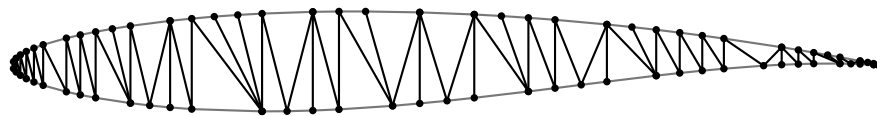


FIGURE 4.3. Descending lexicographic triangulation at the interior of an airfoil

4.2.3 Removal

The removal process can be divided into two components. The first one computes a cavity based on the geometry of the two airfoils. The second component, the local search, uses the cavity previously defined along with other criteria to determine which triangles need to be deleted.

Cavity creation

When the thickness of the new airfoil decreases, we expect to find very small triangles in the regions where the old airfoil boundary was. Therefore, if they were maintained, the mesh would no longer be well-graded, since we would have an increase in triangle size, then a decrease, and yet another increase. Problems arise the other way around as well. If the thickness of the new airfoil increases, we would encounter triangles that are larger than expected close to the boundary, where smaller triangles would be created. This would cause an abrupt increase in triangle size, making the mesh no longer well-graded. If the geometry stays roughly the same, the triangles already have the required size, and only slight modifications are necessary.

To compute the vertices that define the cavity we take into account the distance between the airfoils at the points given by the initial discretisation. This distance is then mirrored out, resulting in something like Figure 4.4. The mirrored distance does not necessarily need to be the same. It can be either smaller or bigger, depending on what is desired. The cavity can be conceptually divided into several trapezoids, which is useful for the next step, the local search.

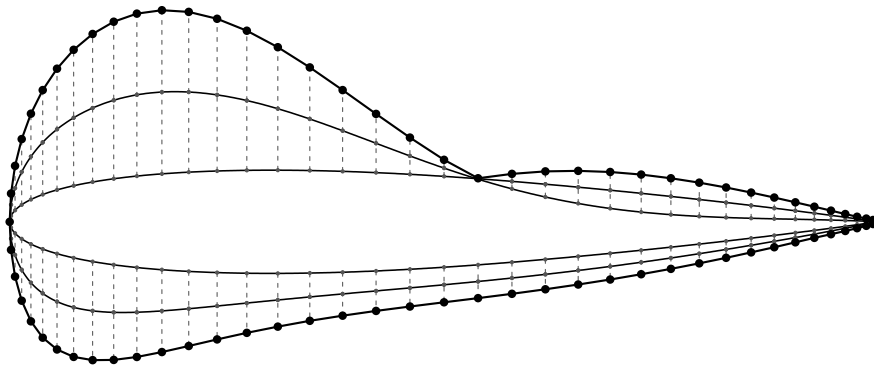


FIGURE 4.4. Trapezoids creation

Local search

In the local search step, cavity triangles are marked for deletion. The reason for the marking and not immediate deletion is that deletion could possibly cut access to other triangles that must be deleted as well. Here, we use a similar method to vertex insertion (Section 3.4.1), but instead of actually inserting them, we just simulate their insertion. The point location algorithm (Section 3.4.2) is applied to obtain a triangle that contains a vertex v from the new airfoil to be inserted. Then a local search starting at that triangle is performed, traversing all the triangles in the corresponding trapezoid that need to be marked for deletion. For each local search, the considered trapezoid is defined by v and its successor.

Removal conditions Since the local search process is performed for every boundary vertex, we have to be careful not to traverse many more triangles than needed, which would result in a slower algorithm. Let v be the vertex considered for the local search, s be the segment defined by v and its successor, and t the triangle to verify. The removal conditions considered are the following (in this order):

1. At least one vertex of t is inside the trapezoid
2. At least one edge of t crosses the right edge of the trapezoid

3. At least one vertex of t is within a pre-defined distance from s
4. At least one vertex of t lies inside the diametral lens of s
5. The circumcircle of t encloses v

In most of the local searches, the last three checks are not needed, since the trapezoids would cover that space. However, for regions where the distance between airfoils is small, as well as for the leading and trailing edges of the airfoil, where the trapezoids are small or non-existing, these verifications are crucial to ensure the correctness of the algorithm. Also note that, if the pre-defined distance in the third condition encloses the diametral lens s , the fourth verification can be ignored. The distance that we recommend for the verification is the length of s . This causes the algorithm to take into account the triangle size in the region, deleting neither more nor less triangles than necessary.

Although the second verification may seem to make little sense, since when a triangle crosses an edge of a trapezoid, one of its vertices might be expected to be inside as well. However, a triangle may cross both edges of a trapezoid without having any vertex in its interior, which it would cause the local search to stop, preventing other triangles beyond that from being reached.

After local search, the marked triangles are deleted from the mesh, originating a cavity in its interior. An example of a mesh after such a deletion can be seen in Figure 4.5B.

4.2.4 Rebuilding

The final process of mesh reconstruction is rebuilding, where the vertices from the new airfoil are inserted in the mesh. However, these are located inside the cavity, and a normal vertex insertion using the Bowyer-Watson algorithm is not possible. To solve this, we use something similar to the gift-wrapping algorithm (Section 3.5.4), but this time with two sets of vertices: the airfoil's and the cavity's. Both sets are ordered counter-clockwise, as we traverse their edges.

First, we select the trailing edge vertex from the airfoil and its closest vertex from the boundary. The closest vertex can be found during the deletion process, when determining the cavity. Hereafter, we follow the same principles as the gift-wrapping algorithm. Candidate vertices from both sets are evaluated and a triangle is formed. The evaluation of both sets stops when a vertex that is no longer visible due to the airfoil's shape is reached. In cases where the endpoints of the unfinished edge belong to the same set, only the vertices from that set are evaluated.

After applying this method along the airfoil, the resulting triangles are guaranteed to be Constrained Delaunay, and the refinement process is run once again to guarantee their quality and good gradation. An illustration of the results of the rebuilding process is provided in Figure 4.5C.

4.3 Additional Observations

One limitation of the method arises in cavity creation, which only occurs within the limits of the leading and trailing edges. It can happen for an airfoil to have a very large leading section, which can cause the formation of small triangles beyond the value of ψ of the leading edge (to the left of it). If it also happens that the next airfoil has a very small leading section, despite the resulting cavity having a large coverage, this is only in the vertical direction, not covering some of the small triangles that should be deleted. When very close to the leading edge, the last three removal conditions (Section 4.2.3) can help reach these triangles, although not for great distances, as it would be the case.

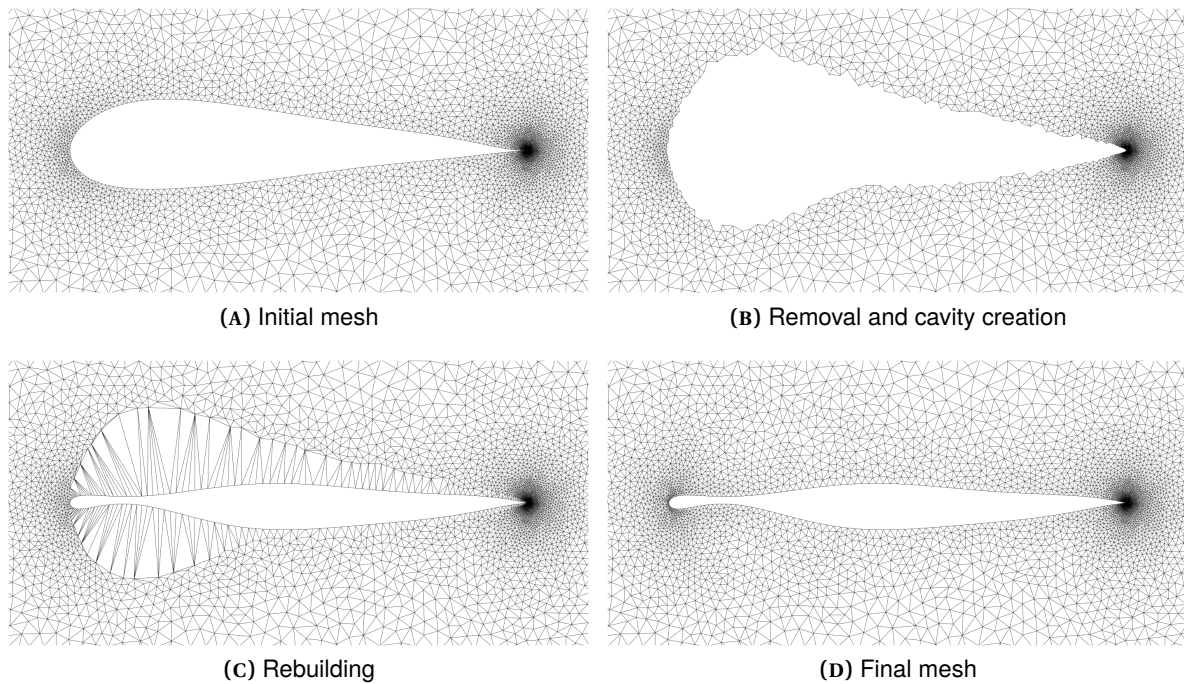


FIGURE 4.5. Steps of the mesh reconstruction method

One alternative that was explored during the dissertation consisted in determining the cavity not by the vertical distances between the airfoils, but by the shortest distance. When close to the leading and trailing edges, this solution would create a cavity that would enclose those small triangles. The removal algorithm would then consist of traversing the triangles that would be crossed by the cavity's border, creating an island of triangles to be deleted. However, since we are allowing a fair amount of change between airfoils, the produced cavity would be highly irregular, possibly even crossing itself and/or preventing the removal and rebuilding algorithms from functioning properly. Not only that, but we would also lose the local search checks, represented by the last three removal conditions, without which the Delaunay properties of the triangulation are not guaranteed.

One way of avoiding or at least attenuating this problem is to limit the magnitude of the changes between airfoils. We could use the function shape coefficients, A , or even the value of the airfoils at carefully chosen points, and impose a limit to the differences between them. If said limit is surpassed, the mesh reconstruction is not considered acceptable and the mesh would be rebuilt from scratch. This would cause the optimisation process to be slower. However, we could ensure that all meshes are well-graded.

Another alternative may also be used. Since having a few more small triangles than needed is not critical in principle, provided that they are of good quality, we could impose a limit not on the differences between airfoils, but on the number of triangles of the resulting mesh. If, since the last time the mesh was built from scratch, the number of triangles has increased by a certain percentage, a new mesh would be built. This way, we should have to build meshes from scratch fewer times, speeding-up the algorithm.

Both alternatives would also prevent larger changes to be expected at the beginning of the optimisation process, to propagate to the final meshes, where only small adjustments should occur.

Chapter 5

Experimental Setup

In this chapter we will start by recovering the available algorithms for model parametrisation (Chapter 2) and mesh generation (Chapter 3), and explain which were selected for experimentation and why. The subject of shape optimisation and how it was addressed in the context of this dissertation will also be explained. Further on, several models of airfoils and their characteristics are presented.

5.1 Methods and Algorithms

5.1.1 Airfoil Parametrisation

For the airfoil parametrisation, two options were presented, NURBS (Section 2.1) and CST (Section 2.2). The first one is more general and is capable of approximating any type of curve, and therefore any airfoil. However, the second method was created specifically to model aircraft sections, including airfoils, and therefore it is also capable of representing airfoils of any kind. Plus, its not as complex as NURBS, and the optimisation parameters are well-defined, making CST a better parametrisation for this case.

As for the parameters in the CST, the chosen order of the Bernstein polynomial is 8, meaning that $n = 7$ (Equation 2.6). The object of optimisation is the shape function coefficient vector A .

5.1.2 Mesh Generation

Initial generation For the initial generation, which involves creating a DT with vertices provided by the initial discretisation of the boundaries, the Bowyer-Watson algorithm (Section 3.4.1) seems the most appropriate, not only because it is a simple algorithm, but also due to the fact that refinement algorithms make use of it, simplifying the implementation. To adjust the DT into a CDT and to that make sure the segments of the boundary are present, the Gift-wrapping algorithm (Section 3.5.4) was chosen.

Refinement With respect to mesh refinement, the choice was between the Ruppert's algorithm (Section 3.6.1) and the Chew's algorithm with Shewchuk's lenses (Sections 3.6.2 and 3.6.3). We chose to use the latter for two reasons. Firstly, Ruppert's algorithm is too restrictive on vertex insertion close to the boundaries, imposing a vertex-free diametral circle, while Shewchuk's lenses are more flexible and can be adjusted according to the minimum angle bound. Secondly, both mesh control methods explored in this dissertation make use of Shewchuk's lenses, therefore removing the need for an adaptation to Ruppert's scheme. The adopted value for the minimum angle bound as well as for Shewchuk's lenses was 30° , or $B = 1$ in Equation 3.8.

Mesh control Finally, the chosen method for mesh control was Gosselin's (Section 3.8.2). The correction that was made to the computation of the values of length scale, LS , of the initial vertices for curved boundaries was the main factor for the decision. Besides that, this method is far simpler than of Ollivier-Gooch and Boivin (Section 3.8.1), as it does not use the curvature of the boundary nor the control it provides over the resolution of input features. Although having less control over the mesh could be seen as an disadvantage, since it is expected that the initial discretisation provides the necessary vertices for a good representation of the boundary, these additional constraints are not needed. The adopted value for the length scale bound was $H = \frac{\sqrt{2}}{2}$.

5.1.3 Summary

In Table 5.1 a summary of the chosen methods and values adopted for experimentation is presented.

Airfoil Parametrisation		Class Shape Transformation
		• Bernstein polynomial of order 8, $n = 7$
Mesh Generation	Vertex insertion	Bowyer-Watson algorithm
	Segment insertion	Gift-Wrapping algorithm
	Mesh refinement	Chew's algorithm with Shewchuk's lenses
	Mesh control	Gosselin's method
		• Length scale bound $H = \sqrt{2}/2$

TABLE 5.1. Methods and algorithms for the experimentation

5.2 Shape Optimisation

Mesh reconstruction methods and the optimisation process function closely to one another. However, we do not have access to a real fluid dynamics optimisation scenario on which in order to test the methods proposed in this dissertation. As such, the objective of the experimentation is not to perform a real aerodynamics optimisation process but to simulate what could possibly be the successive modifications that are made to the airfoil. In this case we decided to perform an interpolation of a target airfoil by some simple optimisation task. Starting with an initial airfoil, its shape is modified varying its parameters until it successfully interpolates several representative points of the target airfoil. The objective function consists of the sum of squared errors at these points.

Next, we describe two types of optimisation methods and how they can affect the mesh reconstruction algorithms.

5.2.1 Direct-search methods

Direct-search methods are derivative-free methods that evaluate the objective function at a finite number of points at each iteration and act based only on those function values [10].

One method of this type is called the coordinate-search method. At each iteration, this method performs what is called a poll step, where it evaluates a certain number of candidates that differ from the current solution in one dimension only. In our case, that is the equivalent of changing only one of the function shape coefficients, A , at a time. If none of the candidates is better than the current solution,

the step size is reduced and the process is repeated until a good solution is reached or the step size decreases beyond a predefined limit.

Another method is the Directional direct-search method. It is identical to the coordinate-search method, with the addition of a search step before the pool step. In the search step, a finite number of candidates is also evaluated. However, these candidates may differ from the current solution in more than one dimension. Usually they are the result of combinations of modifications from the pool step.

The use of direct-search methods along with the proposed mesh reconstruction algorithms should originate good results, since at each iteration the modifications to the shape of the airfoil are small, therefore removing the need for significant changes to the mesh, preserving its overall design and most of its elements. Despite that, these methods are not typically used in real case scenarios due to the fact that they are point-based, i.e. they use only one current solution that is improved through the iterations, instead of population-based, where several solutions are maintained at each iteration.

5.2.2 Covariance Matrix Adaptation - Evolution Strategy (CMA-ES)

Unlike direct-search methods, the Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) is a population-based algorithm. This method is explored here mainly because it is a very well known and extensively studied optimisation method, being widely used over the last years [15].

The CMA-ES method starts with a single individual, from which an initial population is generated. The creation of new individuals uses a value of standard deviation, σ , that determines the magnitude of the modifications applied. At each iteration, new individuals are created based on the previous population and according to σ , that tends to decrease over time. The process is repeated until some performance criteria is met, or σ becomes too small.

Since the shape coefficients can be modified all at once in the same iteration, this method may produce larger changes than those from direct-search methods. The magnitude of these changes always depend on the initial value of σ , which it is expected to be sufficiently high at the beginning. Since new individuals are created from older ones, one can use multiple meshes to store the entire population, and perform the reconstruction method for new individuals based on the respective parent.

The above implies additional overhead for maintaining several copies of the mesh, although it should also reduce the time spent on mesh reconstruction and preserve more of the mesh elements. For simplicity, only one mesh is used in the experimentation for every individual created during the process, which may give rise to bigger changes to the mesh and subsequently, more demanding reconstruction tasks. Nevertheless, it is expected for the method to be capable of handling such situations, while allowing us to assess its performance also in more challenging scenarios.

5.3 Airfoils

The main purpose for the chosen set of airfoils is to be as representative as possible. To achieve that, the idea is to diversify their characteristics, the main ones being thickness and camber (the maximum value and its position on the chord). In this section, nine distinct airfoils are presented and divided into several categories.

The data files for these airfoils were obtained from the University of Illinois Urbana-Champaign's Airfoil Coordinates Database [1].

5.3.1 Standard airfoil

In this context, a standard airfoil is an airfoil without any particular characteristics and whose shape and values of thickness are somewhat average. The purpose of having a standard airfoil in the set is to use it as the initial airfoil of the optimisation process, and since it has no singular characteristics, the limitations that may occur during the optimisation process should be at some point reduced, hopefully.

The NACA 0012 airfoil, from the symmetric four-digit NACA series was chosen as the standard airfoil. As it can be seen from Figure 5.1, the airfoil has zero camber and its thickness is an average one. The camber line is represented by the grey line, computed as the average of the upper and lower surfaces.



FIGURE 5.1. NACA 0012 airfoil

Saying that the maximum thickness is “12% at 30% of the chord” means that the distance between the upper and lower surfaces has a maximum value of 12% the length of the chord, $0.12c$, and it is located at $\psi = 0.3$. The last two digits of the symmetric four-digit NACA series come from the maximum thickness value.

5.3.2 Medium thickness

The choice of airfoils with medium thickness was not so much for the purpose of diversifying their characteristics, but instead consider well-known airfoils that are widely used not only in optimisation studies but also in actual aircrafts.

The first one, presented in Figure 5.2, is another airfoil from the four-digit NACA series, but this time an asymmetric one, with camber.

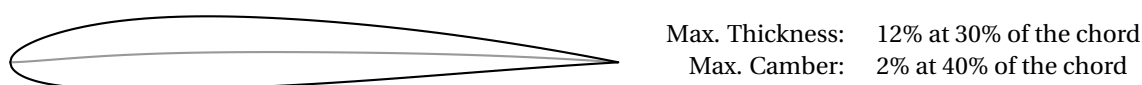


FIGURE 5.2. NACA 2412 airfoil

To say that the maximum camber is “2% at 40% of the chord” means that the maximum distance between the camber (grey line) and the ψ axis (horizontal) has a value of 2% the length of the chord, $0.02c$, and it is located at $\psi = 0.4$. The first digit of the series represents the maximum value of camber and the second digit indicates its position on the chord multiplied by 10. The last two digits are the same as in the symmetric series.

The next two airfoils (Figures 5.3 and 5.4) have been widely used in real aircrafts, that being the main reason for their presence in this set. Recently, the Clark-Y airfoil is more used in aerodynamic studies, while the Boeing 737 airfoil is still present in modern aircrafts.



Max. Thickness: 11.7% at 28% of the chord
 Max. Camber: 3.4% at 42% of the chord

FIGURE 5.3. Clark-Y airfoil



Max. Thickness: 10% at 39.9% of the chord
 Max. Camber: 1.5% at 20.4% of the chord

FIGURE 5.4. Boeing 737 airfoil

Finally, in Figure 5.5 a well-known transonic airfoil that has been used in many studies over the years is shown.

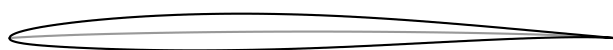


Max. Thickness: 12.1% at 37.9% of the chord
 Max. Camber: 1.3% at 75.7% of the chord

FIGURE 5.5. RAE 2822 airfoil

5.3.3 Small thickness

In the small thickness category two airfoils are presented. With this category, the objective was to diversify the airfoils as much as possible, which can be seen from the values and positions of maximum thickness and camber. In Figure 5.6 an airfoil from the six-digit NACA series is presented and in Figure 5.7 an Eppler airfoil that is supposed to represent an extreme case is shown, allowing us to assess the quality of the proposed mesh reconstruction algorithms in such situations.



Max. Thickness: 6% at 35% of the chord
 Max. Camber: 1.1% at 50% of the chord

FIGURE 5.6. NACA 63206 airfoil



Max. Thickness: 2.5% at 4.3% of the chord
 Max. Camber: 9% at 32.7% of the chord

FIGURE 5.7. Eppler 376 airfoil

5.3.4 Large thickness

In the last category two more airfoils are presented, this time with large thickness. As can be seen from Figures 5.8 and 5.9, they both have distinct values of maximum camber and its position, as well as maximum thickness position.



FIGURE 5.8. Eppler 545 airfoil

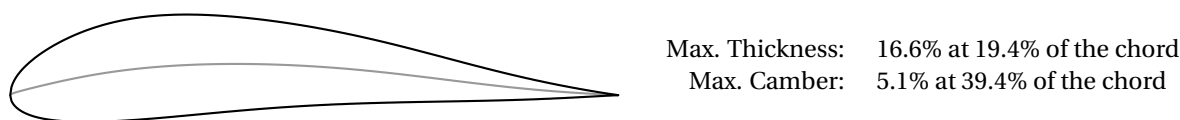


FIGURE 5.9. Gottingen 702 airfoil

5.4 Methodology

Since the optimisation processes of interest use fluid dynamics simulations to obtain the necessary information for the objective function, the mesh has to represent the outside of the airfoil, which makes the airfoil the inner boundary of the mesh. As for the outer boundary, usually referred as *far field*, we used a circular shape with twenty times the chord length as radius (similar to those used in practice and in optimisation studies [23]), so as to adequately represent the space outside the airfoil. The airfoil is located at the centre.

The tests performed in Chapter 6, were run on a laptop running a Linux Operating System, equipped with 4 GB of RAM and an Intel i7 CPU running at 1600 MHz. The source code for the implementation was written in C++ and compiled with GCC 5.1.0. C++ was chosen over C for two reasons. Firstly, its object-oriented features were particularly useful for the boundary interface (Section 2.3). Secondly, it became possible to use the *unordered_map* (hash-table) and *set* containers present in the Standard Template Library (STL). These were used to store the topology of the mesh (Section 3.2) and the queue of poor-quality triangles that have been marked for refinement (Section 3.6), respectively. Besides these three elements, the rest of the source code is written purely in C.

More information on the machine and the compiler can be seen in Table 5.2

OS	Arch Linux 4.0.4, 64 bits
Storage	Hard-Disk Drive, 5400 rpm, 75 MB/s (read)
Memory	4 GB, 1333 MHz
CPU	Intel i7 Quad-Core, 1.6 GHz
Compiler	GCC 5.1.0
	-Wall -ansi -lm -O3 -std=c++11

TABLE 5.2. Machine and implementation specifications

Chapter 6

Experimental Results and Discussion

In this chapter we present the results and respective analysis of the proposed mesh reconstruction method. First, we evaluate the results of shape optimisation using the CMA-ES algorithm under several values of initial standard deviation, σ , in order to make our informed choice of this parameter for subsequent experiments. Next, we compare the proposed method to mesh generation for different values of σ , so as to check its response to different modification magnitudes. The method is also tested for multiple numbers of points in the initial discretisation, I , and gradation control coefficient, G , thus assessing its scalability. Finally, two improvements are tested and compared with the simple reconstruction method.

6.1 Shape optimisation

One of the parameters that can influence significantly the performance of the proposed method for mesh reconstruction is the CMA-ES's value of initial standard deviation, σ . Ideally, for mesh reconstruction the smaller the value of σ the better. However, a very small value could cause the optimisation algorithm to be inefficient. In order to obtain a suitable value for σ , we executed the CMA-ES algorithm using three well-known airfoils for several values of σ . In Figure 6.1 we present how many solutions were evaluated by the algorithm.

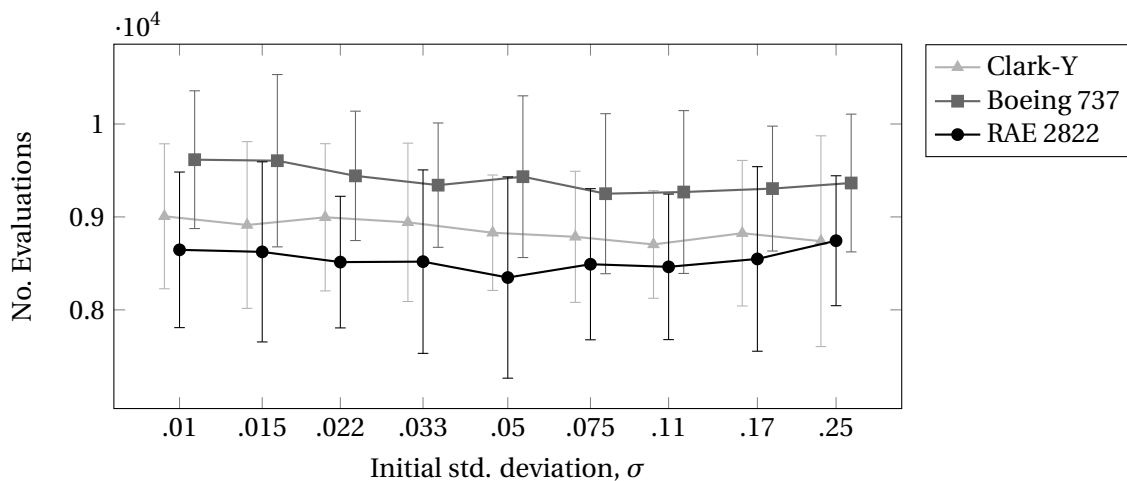


FIGURE 6.1. Number of individuals evaluated by the CMA-ES algorithm
The results present the averages of 50 executions with 99.7% confidence intervals.
The graphics were slightly shifted for visibility reasons

From the results we conclude that there is no significant difference between the tested values. However, it can be noticed that the number of solutions evaluated tends to increase slightly for smaller values of σ , especially below 0.05. For the airfoils in our set, the values of A usually range between 0.1 and 0.3. Also, the authors of the CMA-ES method suggest σ to be roughly one fourth of the search interval. For this reason, we decided to use $\sigma = 0.05$ as default.

6.2 Impact of standard deviation

In this section, we compare the results of mesh generation and mesh reconstruction under variations on the value of σ . The comparison focuses on two aspects: the potential speed-up afforded by using reconstruction, and the surplus of triangles present in the final reconstructed mesh when compared to a newly generated one. Also, we evaluate the preservation of triangles between iterations. Ideally we would have a large speed-up, a low triangle surplus and high element preservation. As the value of σ gets higher, it is expected that the surplus of triangles increases and the speed-up decreases.

In the following tests, we used three airfoils, each of which with a different thickness, so as to have a perspective on how airfoil geometries may affect the method. The values used for the points in the initial discretisation, I , and gradation parameter, G , are 250 and 4, respectively, since these are the ones that produce meshes with an amount of elements similar to those used in the literature ([23], [2] and [20]). In order to account for slight fluctuations, both in the optimisation algorithm and running time, we chose to perform five repetitions for each instance. Results are presented in Tables 6.1 and 6.2. The resulting meshes can be found in Figures A.1 to A.3.

Airfoil	σ	Generation		Reconstruction		Comparison	
		# Tri.	Avg. time	# Tri.	Avg. time	+ Tri.	Speed-up
NACA 63206 (thin)	.01	5405.4	32.61 ms	6389.2	1.88 ms	18.20%	17.35
	.05	5402.2	32.66 ms	6498.0	1.97 ms	20.28%	16.58
	.25	5402.0	31.17 ms	7182.2	2.61 ms	32.95%	11.94
Clark-Y (medium)	.01	5328.0	31.87 ms	5904.4	2.13 ms	10.82%	14.96
	.05	5328.0	30.60 ms	6087.8	1.97 ms	14.26%	15.53
	.25	5328.0	31.61 ms	6720.6	2.46 ms	26.14%	12.85
Gottingen 702 (thick)	.01	5218.0	32.84 ms	6091.0	2.40 ms	16.73%	13.68
	.05	5216.6	33.15 ms	6124.8	2.57 ms	17.41%	12.90
	.25	5208.4	32.16 ms	6738.2	2.65 ms	29.37%	12.14

TABLE 6.1. Comparison between mesh generation and reconstruction under variation of σ
Average of 5 executions, $I = 250$, $G = 4$

As expected, the reconstruction method performs much faster than simple generation, reaching speed-ups from 11.94 up to 17.35, although at the expense of a considerably high triangle surplus. The impact of σ on the results also confirms the initial expectations. The average time per iteration increases for higher values of σ , as does the surplus of triangles. This is due to the larger modifications that a higher value of σ imposes, forcing reconstruction to delete and create triangles farther away from the airfoil.

In Table 6.2 we present additional information on the reconstruction method. The first noticeable thing is that the boundary adjustment process proves to be much faster than full reconstruction process, also preserving more mesh elements. As expected, the number of times it is performed generally

Airfoil	σ	Full Reconstruction			Boundary Adjustment		
		# Iter.	Avg. time	Preserv.	# Iter.	Avg. time	Preserv.
NACA 63206 (thin)	.01	1651.4	4.74 ms	84.20%	7319.8	1.23 ms	91.66%
	.05	1650.6	5.03 ms	83.91%	7258.2	1.26 ms	91.75%
	.25	2912.6	5.30 ms	84.61%	5974.6	1.30 ms	92.53%
Clark-Y (medium)	.01	2155.2	4.71 ms	83.17%	6816.0	1.30 ms	91.00%
	.05	1696.6	4.82 ms	83.19%	7063.4	1.28 ms	91.24%
	.25	2566.2	5.18 ms	83.98%	6016.2	1.30 ms	92.10%
Gottingen 702 (thick)	.01	2783.2	4.80 ms	83.13%	6524.0	1.37 ms	91.23%
	.05	3153.2	4.84 ms	83.41%	5916.4	1.36 ms	91.35%
	.25	2995.8	5.22 ms	83.82%	6071.4	1.39 ms	92.11%

TABLE 6.2. Additional information on the reconstruction results
Average of 5 executions

tends do decrease as σ increases.

An example of the triangle surplus can be observed in Figure 6.2, especially close to the leading and trailing edges.

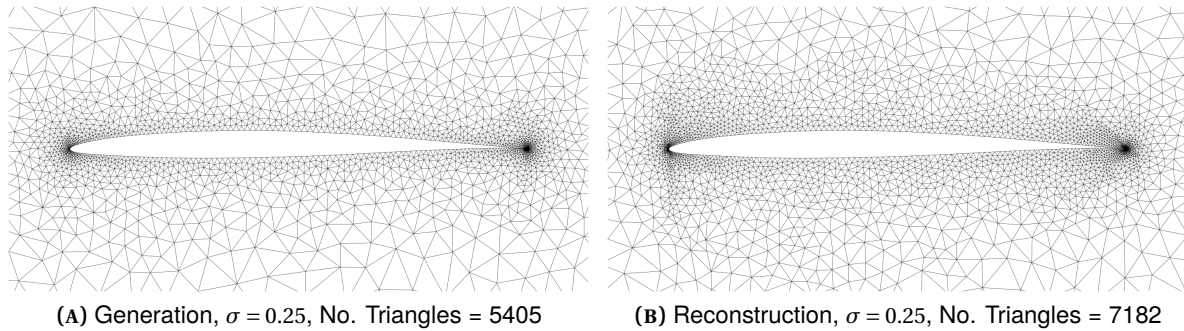


FIGURE 6.2. Illustration of triangle surplus

6.3 Scalability tests

In this section we again compare the generation and reconstruction methods, but this time for different values of I and G with the objective of assessing the scalability of reconstruction. Three of the four tests use values of I and G that allow us to achieve meshes in the same range as those found in the literature [23]. The fourth test uses even higher values, serving almost as an extreme case. Once again, three airfoils with different levels of thickness are used. For the value of σ we chose the default one, 0.05. Results are presented in Tables 6.3 and 6.4 and the resulting meshes are shown in Figures A.4 to A.6.

From Table 6.3 we can see that as I and G increase, and consequently the number of triangles in the mesh increases, so does reconstruction speed-up. This can be explained mostly by the value of G , which by imposing a greater gradation reduces the percentage of triangles that are located close to the airfoil. However, despite these improvements, triangle surplus is still an important issue for smaller

Airfoil	I	G	Generation		Reconstruction		Comparison	
			# Tri.	Avg. time	# Tri.	Avg. time	+ Tri.	Speed-up
Eppler 376 (thin)	150	1	2087.6	12.49 ms	3340.4	1.93 ms	60.01%	6.47
	250	4	5378.2	32.61 ms	7148.2	3.34 ms	32.91%	9.76
	450	7	15716.0	92.38 ms	19760.6	8.70 ms	25.73%	10.62
	700	15	61305.2	420.35 ms	71524.6	25.27 ms	16.67%	16.63
RAE 2822 (medium)	150	1	2244.8	13.66 ms	2638.0	1.20 ms	17.52%	11.38
	250	4	5266.4	34.78 ms	5923.4	1.73 ms	12.48%	20.10
	450	7	15506.8	102.20 ms	16910.6	4.56 ms	9.05%	22.41
	700	15	59955.6	422.90 ms	63544.8	18.53 ms	5.99%	22.82
Eppler 545 (thick)	150	1	2039.6	13.44 ms	2725.4	1.44 ms	33.62%	9.33
	250	4	5186.0	34.70 ms	5992.4	2.40 ms	15.55%	14.46
	450	7	15270.4	101.52 ms	16671.6	7.32 ms	9.18%	13.87
	700	15	59042.4	417.23 ms	62167.6	21.50 ms	5.29%	19.41

TABLE 6.3. Comparison between mesh generation and reconstruction under variation of I and G
Average of 5 executions, $\sigma = 0.05$

Airfoil	I	G	Full Reconstruction			Boundary Adjustment		
			# Iter.	Avg. time	Preserv.	# Iter.	Avg. time	Preserv.
Eppler 376 (thin)	150	1	4686.4	2.72 ms	85.02%	4256.0	1.05 ms	90.81%
	250	4	4132.2	5.46 ms	85.14%	4810.2	1.50 ms	92.53%
	450	7	5915.8	11.71 ms	89.62%	3026.6	2.78 ms	95.01%
	700	15	7407.4	29.06 ms	94.49%	1616.6	7.70 ms	97.62%
RAE 2822 (medium)	150	1	2761.8	2.20 ms	82.26%	5885.4	0.76 ms	88.30%
	250	4	1396.0	4.64 ms	83.08%	7251.2	1.16 ms	91.03%
	450	7	2960.6	10.43 ms	88.36%	5686.6	2.35 ms	94.16%
	700	15	5367.0	25.71 ms	94.12%	3280.2	6.71 ms	97.35%
Eppler 545 (thick)	150	1	3704.8	2.31 ms	82.17%	5201.6	0.83 ms	88.65%
	250	4	2922.6	4.77 ms	83.13%	5983.8	1.24 ms	91.12%
	450	7	5608.0	10.20 ms	88.30%	3298.4	2.36 ms	94.13%
	700	15	7032.6	25.33 ms	94.06%	1873.8	6.76 ms	97.30%

TABLE 6.4. Additional information on the reconstruction results
Average of 5 executions

values of I and G , since the more triangles the mesh has, the slower the subsequent simulation stages are likely to be.

The additional results presented in table 6.4 reinforce the conclusions made in the previous section. This time, however, the number of boundary adjustments performed has decreased. This can be explained by the smaller triangles located on the airfoil boundary, requiring even smaller modifications made to the shape coefficients, A , than before.

When we increase the number of triangles in the mesh, we start to observe another problem (addressed to a certain extent in Section 4.2.3) that also has to do with triangle surplus: bad gradation.

As can be seen in Figure 6.3, several excess triangles were left behind by the reconstruction process, causing the mesh to alternate between increasing and decreasing gradation in those areas.

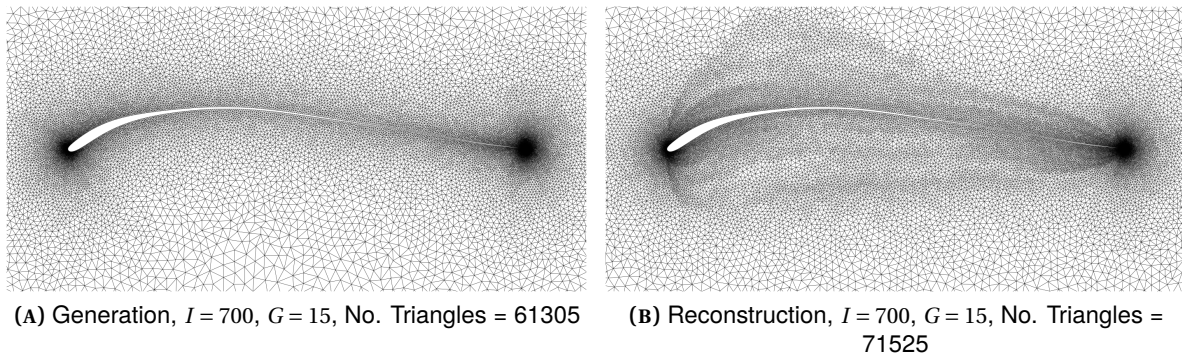


FIGURE 6.3. Illustration of bad gradation

6.4 Improvements

In the previous tests, the main issue that we encounter is the surplus of triangles in the final mesh obtained using reconstruction when compared to a newly generated one. As the number of elements in the mesh increases, this also tends to produce badly-graded meshes. In this section we propose two improvements to mesh reconstruction, in order to solve or at least reduce the magnitude of these problems.

For the following tests, we chose a medium thickness airfoil, the Boeing 737. Since it is the thinnest of its category, it should be able to recreate the problems mentioned above. Initially, we start by performing the original mesh reconstruction method, so as to compare with the two proposed improvements. The results are presented in Table 6.5.

σ	I	G	Generation		Reconstruction			Comparison	
			# Tri.	Avg. time	# Tri.	Avg. time	Preserv.	+ Tri.	Speed-up
.05	250	4	5352.2	30.98 ms	6056.4	2.06 ms	89.59%	13.16%	15.04
	450	7	15498.0	98.17 ms	17248.2	7.03 ms	91.01%	11.29%	13.96
.15	250	4	5352.2	31.17 ms	6581.6	2.38 ms	89.63%	22.97%	13.10
	450	7	15498.0	96.61 ms	18021.4	7.46 ms	91.03%	16.28%	12.95

TABLE 6.5. Comparison between mesh generation and reconstruction for the Boeing 737 airfoil
Average of 5 executions

6.4.1 Shape coefficients

The first improvement to mesh reconstruction is based on the observed variation of shape coefficients, A . Since the airfoil shape is directly related to its coefficients, a large variation of these coefficients should produce considerably different shapes. To use that to our advantage, we check the variation of the coefficients before each iteration and decide between the generation and reconstruction methods. We choose the generation over reconstruction when one of the following applies:

- One of the coefficients changes by more than 0.1
- The average variation is higher than 0.05

With this simple method, we should expect better results in terms of triangle surplus, although at the cost of a small loss of speed-up and preservation, since in some iterations no triangles are preserved at all. Results are presented in Table 6.6 and the resulting meshes are depicted in Figures A.7 to A.10.

σ	I	G	Extension (A)					vs Gen.	
			Gen.	Rec.	# Tri.	Avg. time	Preserv.	+ Tri.	Speed-up
.05	250	4	191.8	9257.0	5399.6	2.38 ms	87.17 %	0.89%	13.02
	450	7			15684.2	7.67 ms	89.44%	1.20%	12.80
.15	250	4	476.0	8857.6	5557.0	3.50 ms	84.09 %	3.82%	8.91
	450	7			16006.6	10.90 ms	86.30%	3.28%	8.86

TABLE 6.6. Comparison between mesh generation and improved reconstruction based on coefficient variation for the Boeing 737 airfoil
Average of 5 executions

6.4.2 Number of triangles

The second alternative improvement uses the variation in the number of triangles instead of shape coefficients. When the number of triangles in the mesh at a certain iteration differs by more than 5% from the last newly generated one, we consider that the modifications are higher than normal, and a new mesh is generated. As with the previous approach we expect the method to achieve better results of triangle surplus. However, the generation method is expected to perform less often, and thus the values of both speed-up and preservation should not deteriorate as much. The results can be seen in Table 6.7 and in Figures A.7 to A.10.

σ	I	G	Extension (Tri.)					vs Gen.	
			Gen.	Rec.	# Tri.	Avg. time	Preserv.	+ Tri.	Speed-up
.05	250	4	35.2	9413.6	5556.6	2.11 ms	88.53%	3.82%	14.68
	450	7	23.0	9425.8	15898.8	6.40 ms	90.63%	2.59%	15.34
.15	250	4	78.4	9226.4	5515.6	2.38 ms	87.64%	3.05%	13.10
	450	7	62.0	9242.8	15983.0	7.68 ms	89.52%	3.13%	12.58

TABLE 6.7. Comparison between mesh generation and improved reconstruction based on triangle surplus for the Boeing 737 airfoil
Average of 5 executions

6.4.3 Evaluation

In Table 6.8 we gather the results from the mesh reconstruction method and its improvements when compared to the generation method. Both improvements achieved good results, reducing very considerably the value of triangle surplus in the final mesh when compared to the original reconstruction method. In terms of speed-up, the first extension lead to considerable degradation. Also, as the number of triangles in a mesh grows larger, the method becomes less efficient. However, for the second improvement the speed-up is essentially the same, and in some cases, even better. This happens

due to the triangle surplus that is created by the pure reconstruction method in the first iterations, causing more triangles to be deleted in later iterations, thus becoming slower. As for average triangle preservation, both improvements present good results, considering that some of the iterations have 0 preservation.

Overall, it can be said that the second improvement is the best alternative between the three approaches, achieving good values of triangle surplus while maintaining the original speed-up and not compromising on the values of preservation. Moreover, it performs well not only for different values of σ , being resistant to the magnitude of the variations, but also for meshes with different number of elements.

σ	I	G	Reconstruction		Extension (A)		Extension (Tri.)	
			+ Tri.	Speed-up	+ Tri.	Speed-up	+ Tri.	Speed-up
.05	250	4	13.16%	15.04	0.89%	13.02	3.82%	14.68
	450	7	11.29%	13.96	1.20%	12.80	2.59%	15.34
.15	250	4	22.97%	13.10	3.82%	8.91	3.05%	13.10
	450	7	16.28%	12.95	3.28%	8.86	3.13%	12.58

TABLE 6.8. Comparison between mesh reconstruction and its extensions for the Boeing 737 airfoil
Average of 5 executions

To conclude, some meshes produced by these methods are depicted in Figure 6.4. It is clearly noticeable that the bad gradation problems, caused by triangle surplus, have been essentially suppressed by both improvements, with no visible differences between the meshes produced by these approaches and mesh generation.

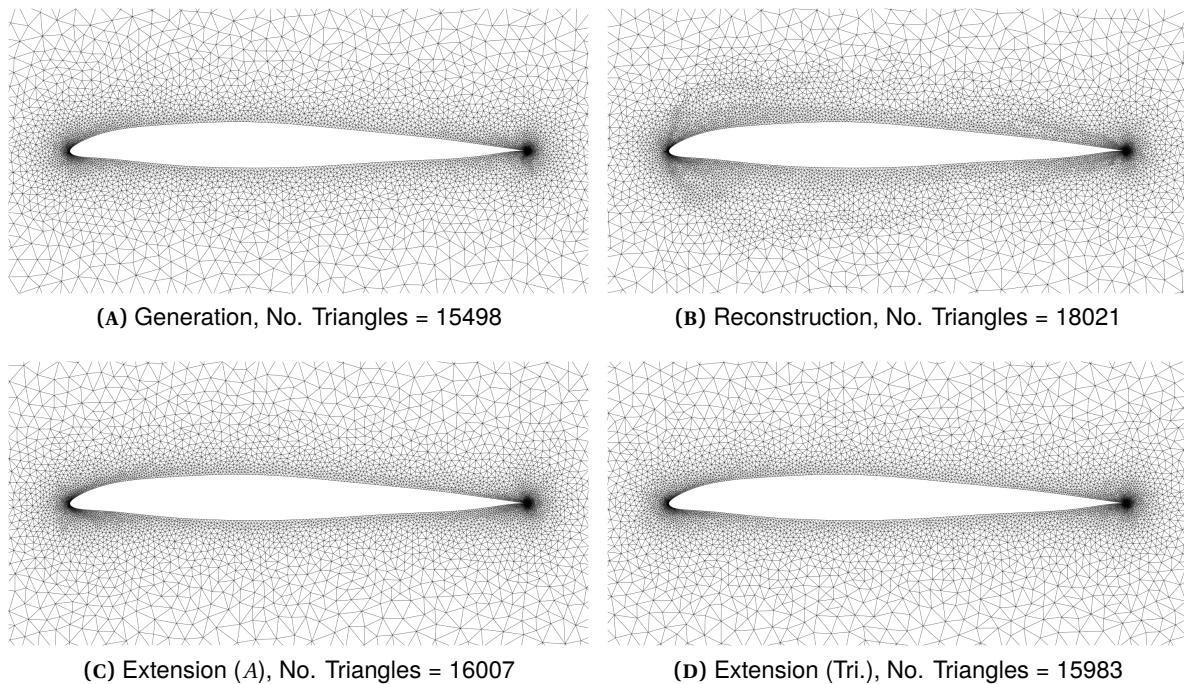


FIGURE 6.4. Comparison between generation, reconstruction and its improvements, $\sigma = 0.15$, $I = 450$, $G = 7$

Chapter 7

Conclusions and Future Work

The goal of this dissertation was to develop a new approach to incremental re-meshing in the context of engineering design optimisation. With this new approach, we intended to increase the efficiency of mesh generation across optimisation iterations, and to potentiate further improvements in the remaining simulation stages.

To achieve this, we proposed a new mesh reconstruction method that, unlike mesh deformation methods, is capable of removing and adding elements to the mesh as needed (changing the mesh topology), instead of just modifying their position and geometry. Additionally, as we make use of Delaunay Triangulations for the mesh, we can guarantee its high-quality and optimal properties at every stage of the process, making it a very reliable and robust method.

The results obtained with this method were very satisfactory, although not without some shortcomings. As we change the topology of the mesh, it is impossible to ensure that for the same final model, reconstructed meshes do not contain more elements than newly built meshes. Since the quality of these additional elements is still guaranteed, the results given by the simulator should be as reliable as before if not better, but the larger mesh size may also make the simulator slower. Improvements to the simpler mesh reconstruction method were then proposed, which allowed the number of additional elements to be limited without compromising on the other advantages that mesh reconstruction provides.

In terms of speed-up, the proposed algorithm for mesh reconstruction achieves relevant results, reaching values up to 22 for finer meshes. However, the time it takes for a fluid-dynamics simulator to perform a single evaluation may be much larger than mesh generation, reaching several minutes [20]. As such, it is not expected that just performing mesh reconstruction instead of generation will reduce the duration of the optimisation process in any significant way. Nonetheless, it is a considerable improvement in itself.

Much more importantly, mesh reconstruction should allow subsequent stages of the simulator to be implemented more efficiently as well. In several methods used in fluid dynamics simulation, such as finite-element and finite-volume methods (FEM and FVM, respectively), the first step consists of generating a set of equations based on the topology of the mesh. Since our mesh reconstruction method preserves most of the elements between iterations (reaching values around 90%) most of these equations will remain the same, which should offer further opportunities to speeding-up the equation solving stage. For example, it is known that updating the solution of a system of linear equations under column or row exchange can be performed much faster than resolving the system from scratch, [13] and [12].

Despite the fairly good results that the method already provides, there is still room for improvement, especially in the local search component (4.2.3). As mentioned in Section 4.3, we explored an alternative that in practice, performed twice as fast as the current method. However, since it is not guaranteed to always work, it could not be used.

From the good results that both improvements to the original method provide, especially the second one, it is noticeable that minor tweaks to the original method can make a great difference. Further work on these improvements includes using threshold values of varying them during the process. Additionally, there is the opportunity to devise other criteria to decide when it is better to generate a new mesh rather than adapting it.

In the medium to long term, the natural follow-up would be to extend mesh reconstruction to three-dimensional space. Scalability test results show that the proposed method performs better on larger and finer meshes, which suggests that it may be possible to achieve considerable performance gains on three dimensions. Provided that such gains of at least one order of magnitude can be realised also on the other solver stages, this could potentially change engineering design optimisation practice quite radically.

References

- [1] Airfoil Coordinates Database, University of Illinois Urbana-Champaign, USA. http://m-selig.ae.illinois.edu/ads/coord_database.html.
- [2] Accuracy preserving limiter for the high-order accurate solution of the Euler equations, author=Michalak, Christopher and Ollivier-Gooch, Carl. *Journal of Computational Physics*, 228(23):8693–8711, 2009.
- [3] N. Amenta, S. Choi, and G. Rote. Incremental constructions con BRIO. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 211–219. ACM, 2003.
- [4] J. T. Batina. Unsteady Euler airfoil solutions using unstructured dynamic meshes. *AIAA journal*, 28(8):1381–1388, 1990.
- [5] C. Boivin and C. Ollivier-Gooch. Guaranteed-quality triangular mesh generation for domains with curved boundaries. *International Journal for Numerical Methods in Engineering*, 55(10):1185–1213, 2002.
- [6] A. Bowyer. Computing dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.
- [7] C. O. Burg. A robust unstructured grid movement strategy using three-dimensional torsional springs. *AIAA paper*, 2529:2004, 2004.
- [8] L. P. Chew. Guaranteed-quality triangular meshes. Technical report, Cornell University, 1989.
- [9] L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 274–280. ACM, 1993.
- [10] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization*, volume 8. Siam, 2009.
- [11] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Computer methods in applied mechanics and engineering*, 163(1):231–245, 1998.
- [12] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Maintaining LU factors of a general sparse matrix. *Linear Algebra and its Applications*, 88:239–270, 1987.
- [13] J. Gondzio. Stable algorithm for updating dense LU factorization after row or column exchange and row and column addition or deletion. *Optimization*, 23(1):7–26, 1992.
- [14] S. Gosselin. Delaunay refinement mesh generation of curve-bounded domains. 2009.
- [15] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages=312–317, year=1996, organization=IEEE.

- [16] B. T. Helenbrook. Mesh deformation using the biharmonic operator. *International journal for numerical methods in engineering*, 56(7):1007–1021, 2003.
- [17] E. N. Jacobs and K. E. Ward. Interference of wing and fuselage from tests of 209 combinations in the NACA variable-density tunnel. Technical report, DTIC Document, 1935.
- [18] B. M. Kulfan. Modification of CST Airfoil Representation Methodology. <http://brendakulfan.com/docs/CST8.pdf>.
- [19] B. M. Kulfan and J. E. Bussoletti. Fundamental parametric geometry representations for aircraft component shapes. In *11th AIAA/ISSMO multidisciplinary analysis and optimization conference*, pages 1–42. sn, 2006.
- [20] C. Michalak and C. Ollivier-Gooch. Globalized matrix-explicit Newton-GMRES for the high-order accurate solution of the Euler equations. *Computers & Fluids*, 39(7):1156–1167, 2010.
- [21] G. L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. A Delaunay based numerical method for three dimensions: generation, formulation, and partition. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages=683–692. ACM, 1995.
- [22] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217–236, 1978.
- [23] A. Nejat and C. Ollivier-Gooch. A high-order accurate unstructured finite volume Newton–Krylov algorithm for inviscid compressible flows. *Journal of Computational Physics*, 227(4):2582–2609, 2008.
- [24] C. Ollivier-Gooch and C. Boivin. Guaranteed-quality simplicial mesh generation with cell size and grading control. *Engineering with Computers*, 17(3):269–286, 2001.
- [25] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *SODA*, volume 93, pages 83–92, 1993.
- [26] J. R. Shewchuk. Delaunay refinement mesh generation. Technical report, DTIC Document, 1997.
- [27] J. R. Shewchuk. *Lecture notes on Delaunay mesh generation*. Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1999.
- [28] D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The computer journal*, 24(2):167–172, 1981.

Appendix A

Results' Figures

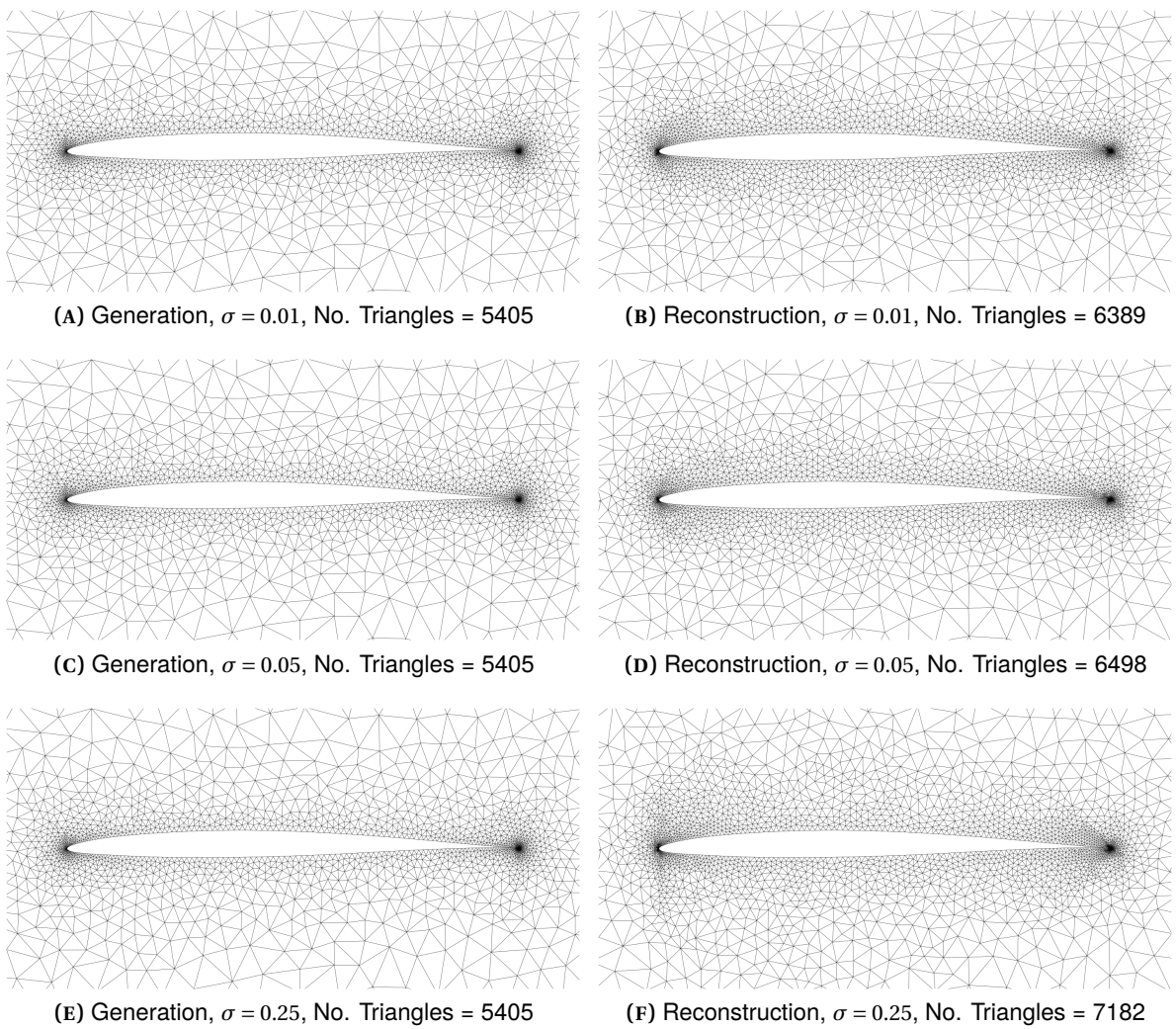


FIGURE A.1. Comparison between mesh generation and mesh reconstruction while varying σ for the NACA 63206 airfoil (thin), $I = 250$, $G = 4$

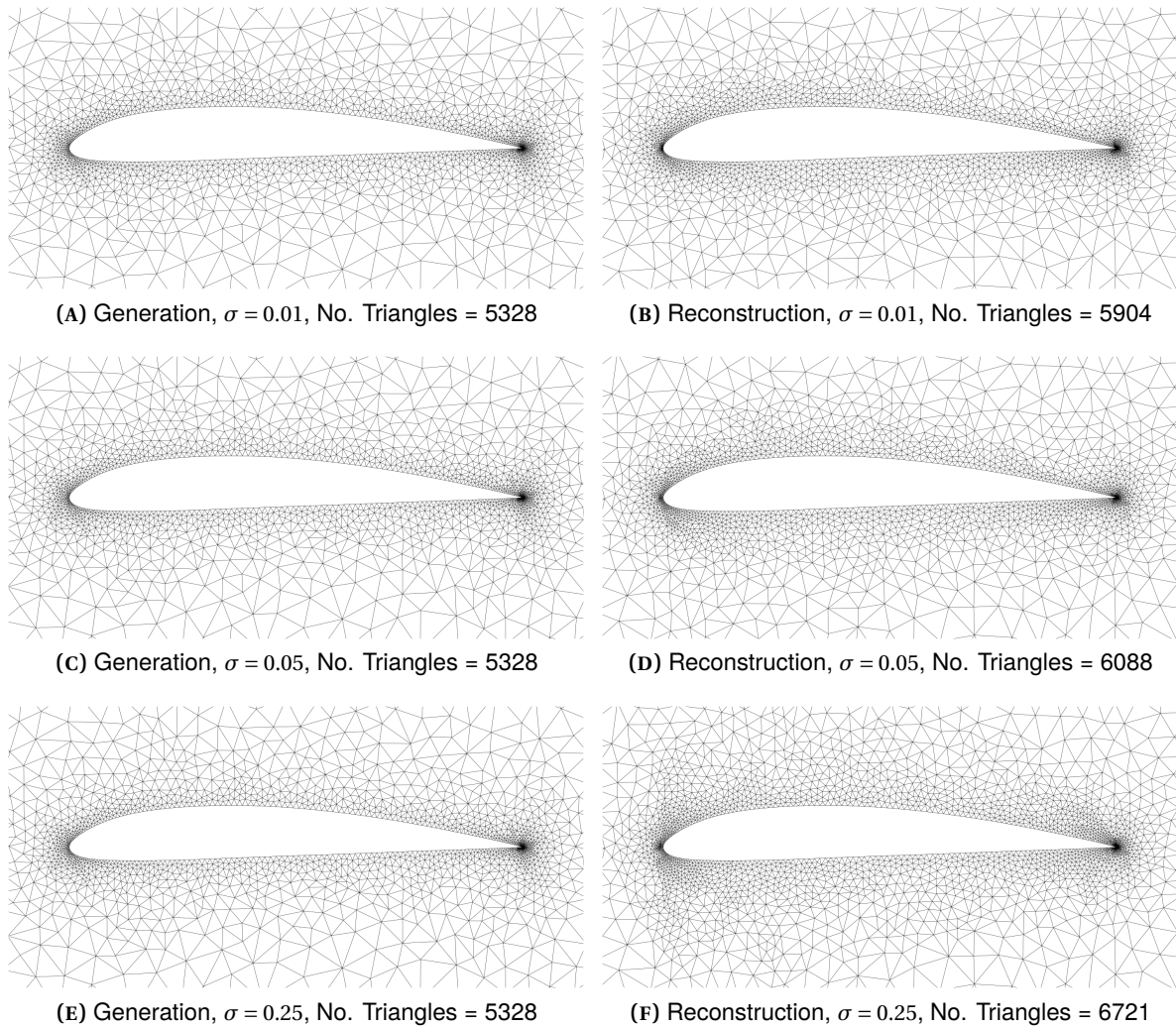


FIGURE A.2. Comparison between mesh generation and mesh reconstruction while varying σ for the Clark-Y airfoil (medium), $I = 250$, $G = 4$

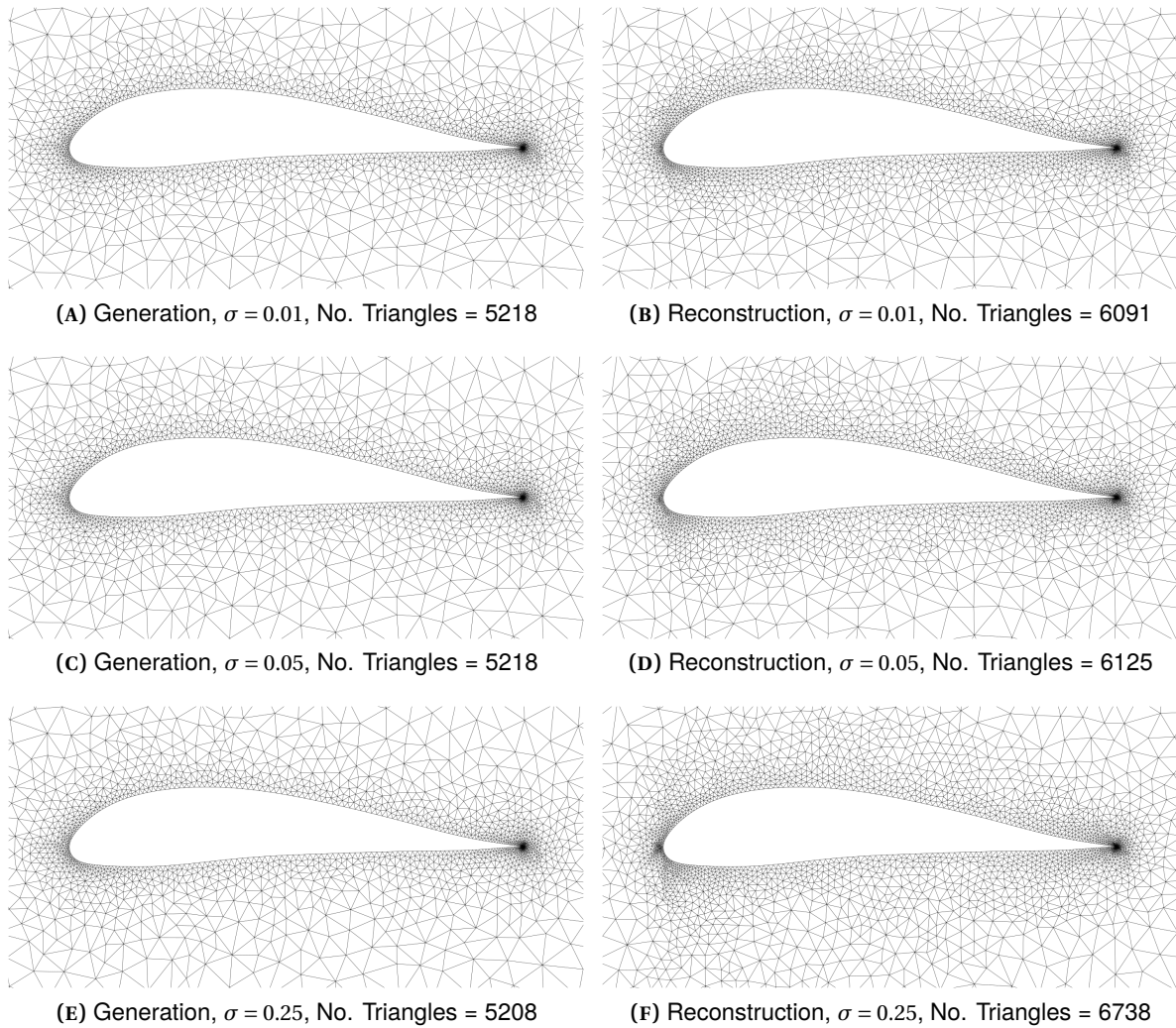


FIGURE A.3. Comparison between mesh generation and mesh reconstruction while varying σ for the Gottingen 702 airfoil (thick), $I = 250$, $G = 4$

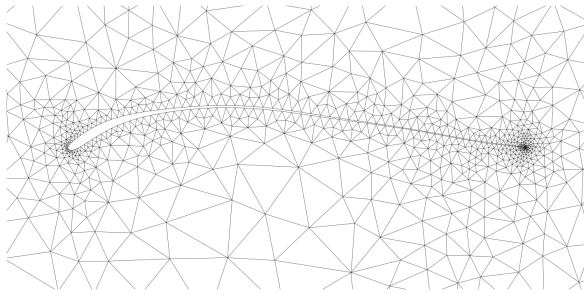
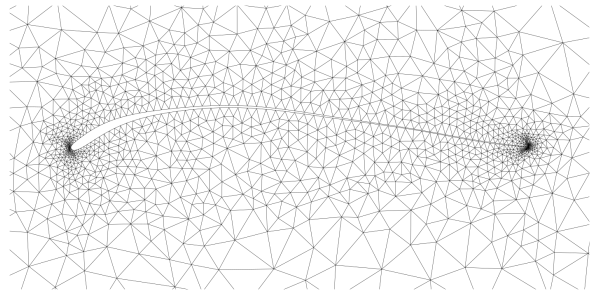
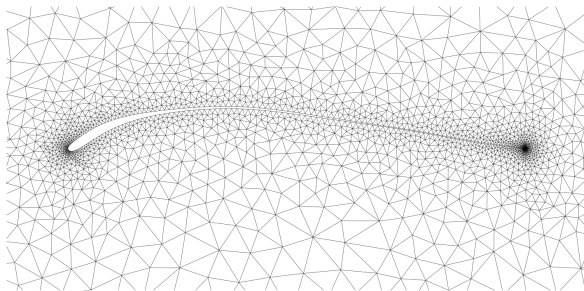
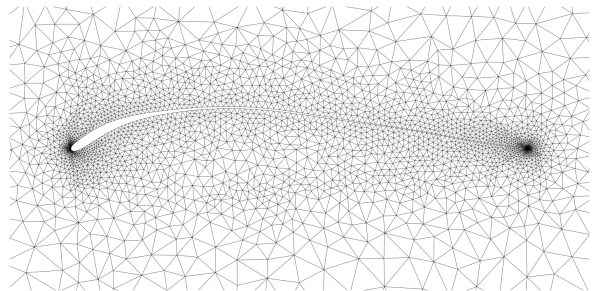
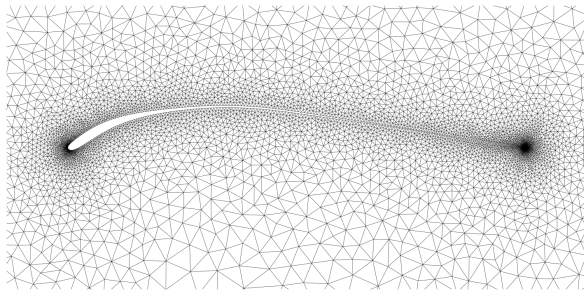
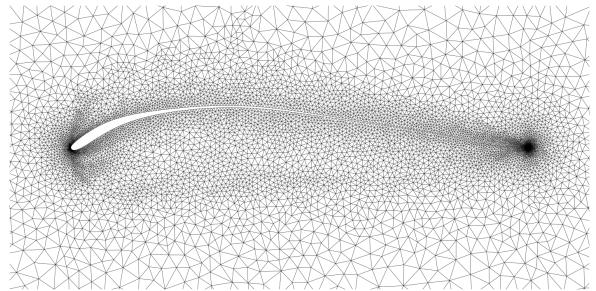
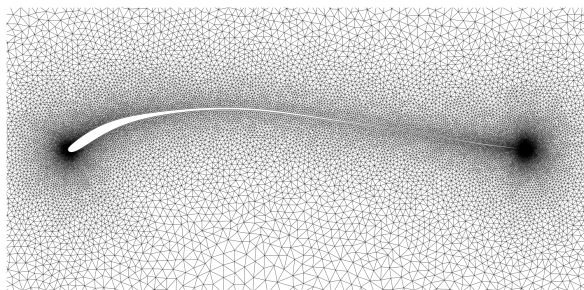
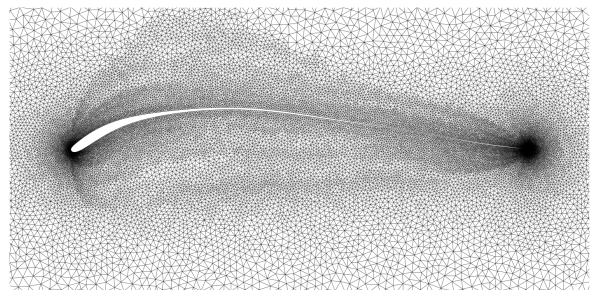
(A) Generation, $I = 150$, $G = 1$, No. Triangles = 2088(B) Reconstruction, $I = 150$, $G = 1$, No. Triangles = 3340(C) Generation, $I = 250$, $G = 4$, No. Triangles = 5378(D) Reconstruction, $I = 250$, $G = 4$, No. Triangles = 7148(E) Generation, $I = 450$, $G = 7$, No. Triangles = 15716(F) Reconstruction, $I = 450$, $G = 7$, No. Triangles = 19760(G) Generation, $I = 700$, $G = 15$, No. Triangles = 61305(H) Reconstruction, $I = 700$, $G = 15$, No. Triangles = 71525

FIGURE A.4. Comparison between mesh generation and mesh reconstruction while varying I and G for the Eppler 376 airfoil (thin), $\sigma = 0.05$

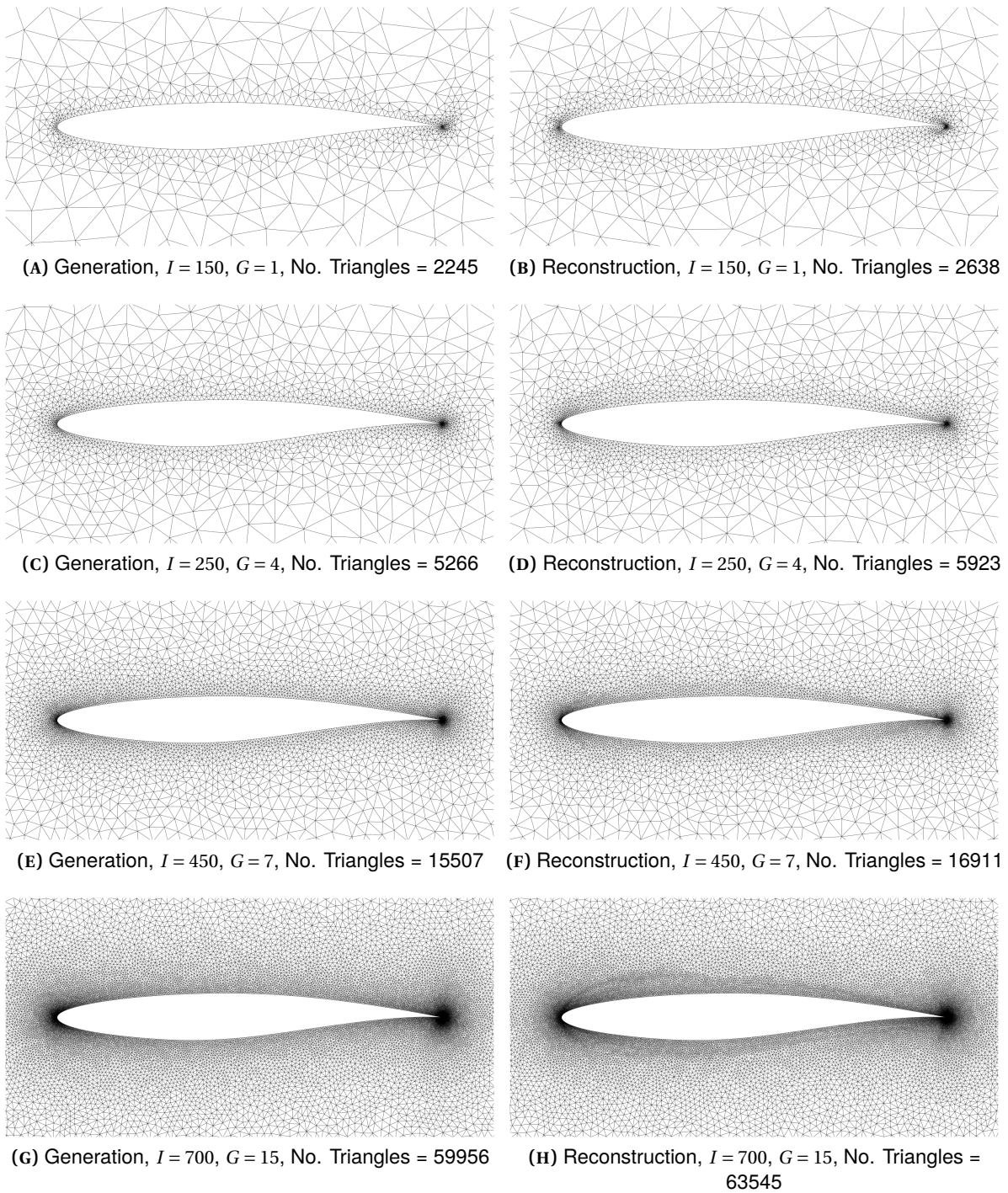


FIGURE A.5. Comparison between mesh generation and mesh reconstruction while varying I and G for the RAE 2822 airfoil (medium), $\sigma = 0.05$

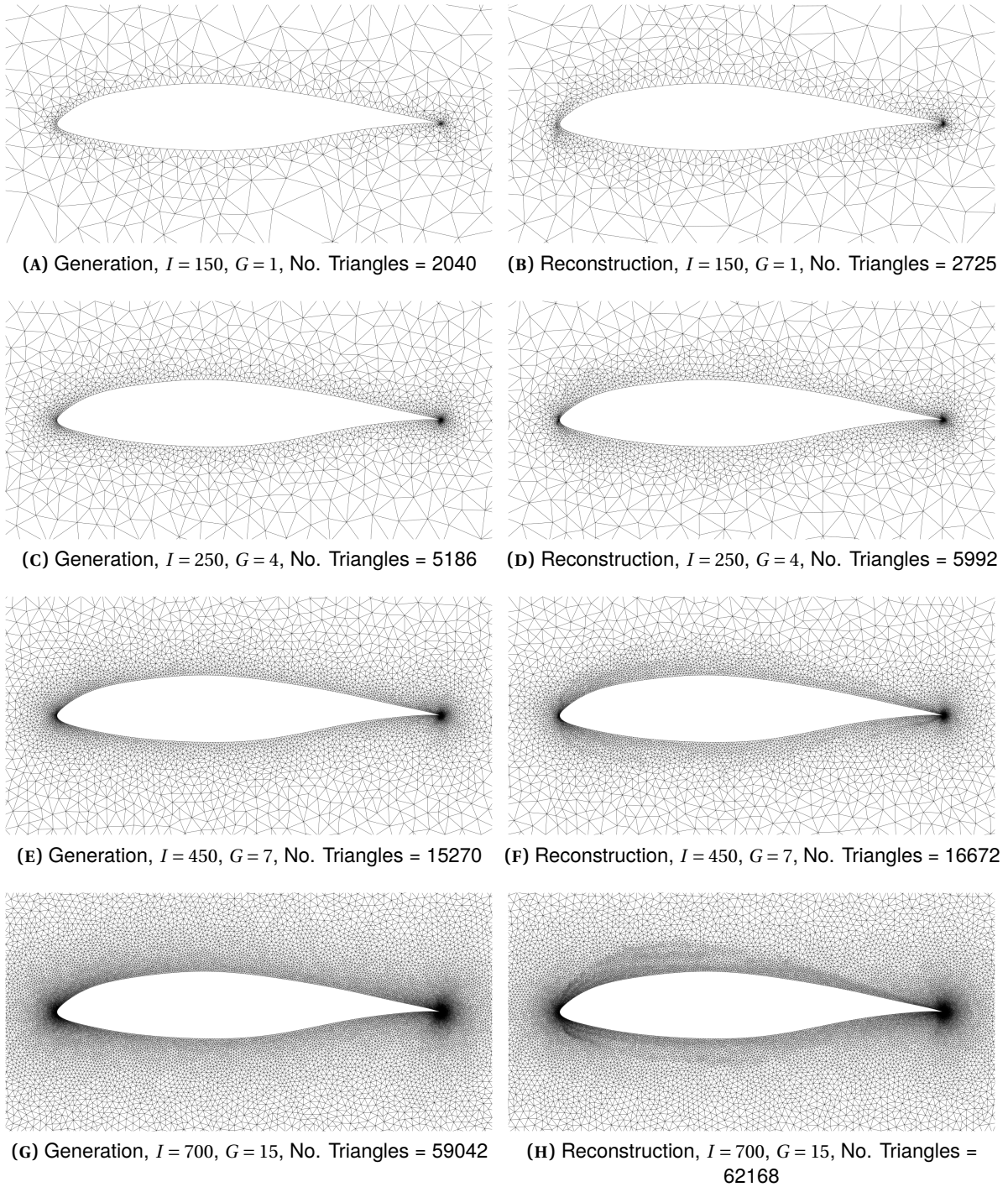


FIGURE A.6. Comparison between mesh generation and mesh reconstruction while varying I and G for the Eppler airfoil (medium), $\sigma = 0.05$

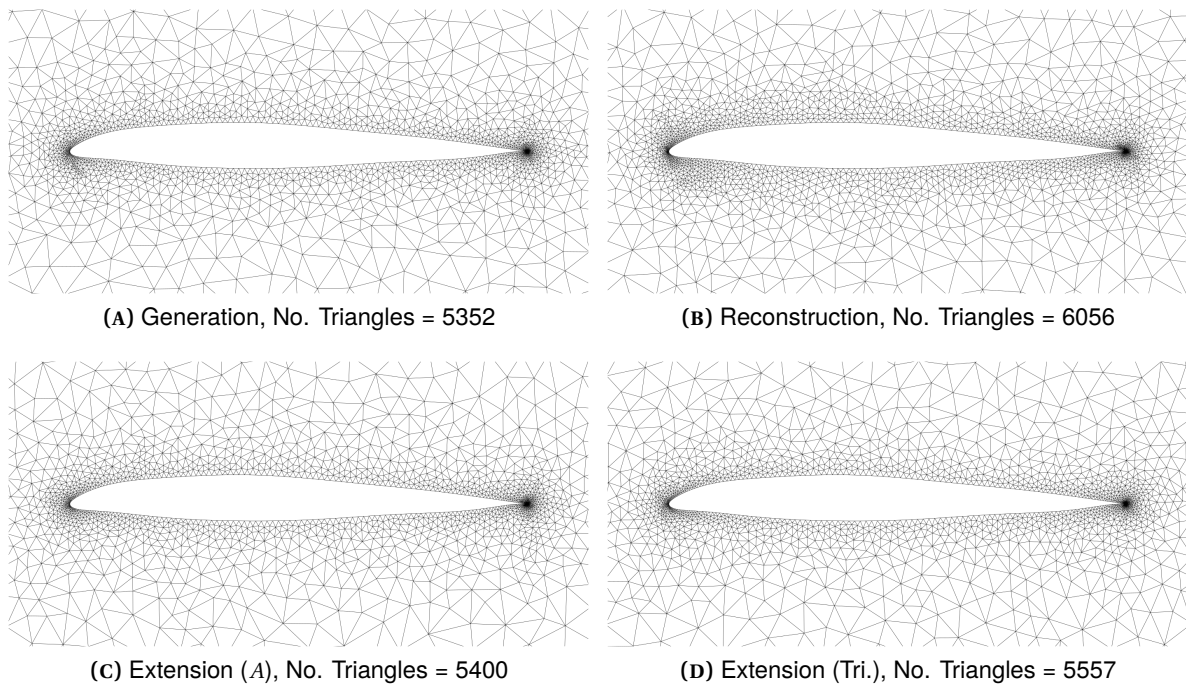


FIGURE A.7. Comparison between mesh generation and mesh reconstruction and its improvements for the Boeing 737 airfoil, $\sigma = 0.05$, $I = 250$, $G = 4$

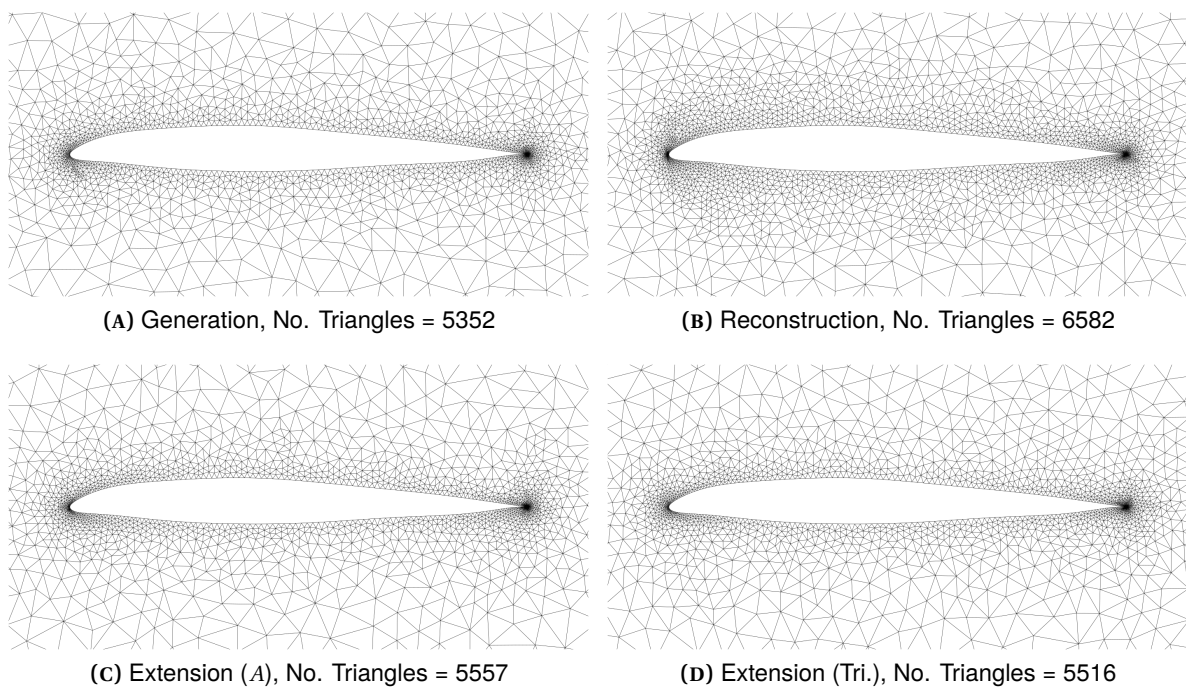


FIGURE A.8. Comparison between mesh generation and mesh reconstruction and its improvements for the Boeing 737 airfoil, $\sigma = 0.15$, $I = 250$, $G = 4$

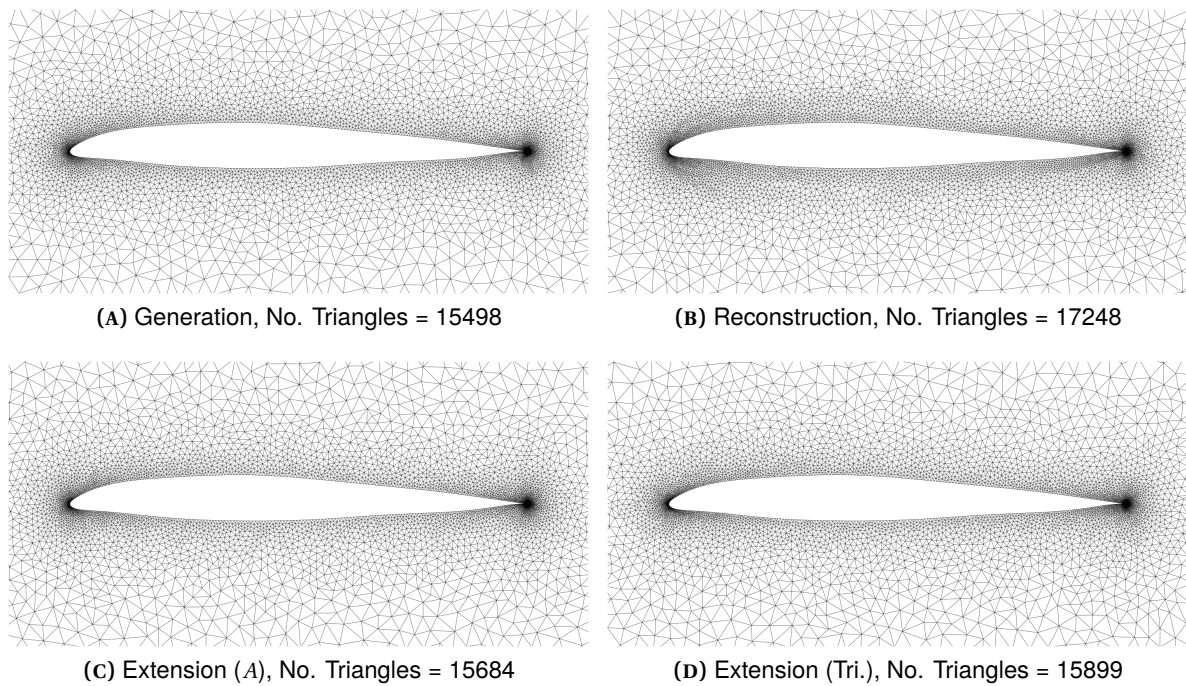


FIGURE A.9. Comparison between mesh generation and mesh reconstruction and its improvements for the Boeing 737 airfoil, $\sigma = 0.05$, $I = 450$, $G = 7$

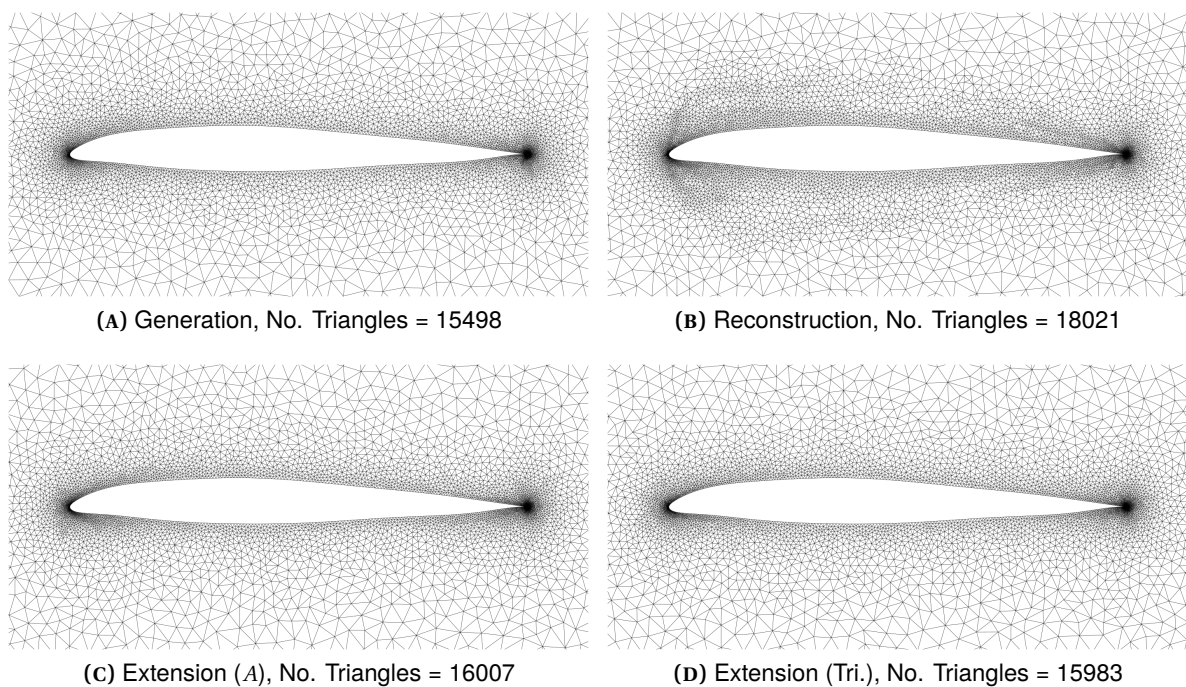


FIGURE A.10. Comparison between mesh generation and mesh reconstruction and its improvements for the Boeing 737 airfoil, $\sigma = 0.15$, $I = 450$, $G = 7$