National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Laboratory of Thermal Turbomachines
Parallel CFD & Optimization Unit

Implementation of the Boundary Representations of CAD Geometries in Gradient-based Optimization

PhD Thesis

Marios G. Damigos

Supervisor: Kyriakos C. Giannakoglou,
Professor NTUA

Athens, 2023

National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Laboratory of Thermal Turbomachines
Parallel CFD & Optimization Unit

Implementation of the Boundary Representations of CAD Geometries in Gradient-based Optimization

PhD Thesis

Marios G. Damigos

Examination Committee:

1. K. Giannakoglou (Supervisor)*
   Professor, NTUA, School of Mechanical Engineering
2. S. Voutsinas*
   Professor, NTUA, School of Mechanical Engineering
3. K. Mathioudakis*
   Professor, NTUA, School of Mechanical Engineering
4. J. Nikolos
   Professor, Technical University of Crete, School of Production Engineering & Management
5. G. Papadakis
   Assistant Professor, NTUA, School of Naval Architecture and Marine Engineering
6. G. Vosniakos
   Professor, NTUA, School of Mechanical Engineering
7. V. Riziotis
   Associate Professor, NTUA, School of Mechanical Engineering

* Member of the Advisory Committee.

Athens, 2023

Part of this work has been conducted within the IODA ITN on "Industrial Optimal Design using Adjoint CFD"
http://ioda.sems.qmul.ac.uk

IODA has received funding from the European Union's HORIZON 2020 Research and Innovation Programme under the Marie Skłodowska-Curie Grant Agreement No. 642959.

Abstract

This PhD thesis deals with the coupling of CAD-parameterized geometries with adjoint-based shape optimization. The mathematical formulation of methods for the inclusion of a CAD design in the optimization loop are shown and tested in applications in aerodynamics. The computation of the derivatives of various aerodynamic objective functions with respect to (w.r.t.) the CAD design variables is performed based on continuous adjoint running in the OpenFOAM environment.

CAD geometries may have two parameterizations: (a) a feature tree parameterization which is practically the definition of geometric relations in the 3D space and also the native parameterization of CAD packages that generate them and (b) a surface parameterization which is defined and transferred by the standard Boundary Representation (BRep) format. The latter is simply a collection of surface patches that define the CAD model, defined by standard mathematical forms (mainly NURBS). In this thesis, BRep is used to express the CAD geometry because its standardized open-source format allows its direct coupling with Computational Fluid Dynamics (CFD) software and, also, its differentiation which is necessary in gradient-based optimization. The feature tree parameterization is almost never accessible via open formats which makes its direct linking to optimization impossible.

The first step is the generation of a quality triangulation of the surface of the CAD model. This is because, in order to insert the CAD model into the optimization loop, it is necessary to mesh the 3D space around (or inside) it. The boundary of the domain to be meshed, in a tesselated (most commonly triangulated) form is the input to the meshing software. The triangulation process is subdivided into three main tasks: shape healing which is a process overcoming possible (but quite common) CAD model defects, size map computation which pre-computes the optimal size of the triangulation on a background grid and, finally, the surface triangulation itself. Shape healing handles the topological and geometric holes that commonly exist in a CAD model which is transferred via a standard file (STEP, IGES, etc.). The topological holes are fixed by performing vicinity tests, and the geometric holes are fixed by performing a sewing algorithm based on Plate Energy Minimization. Then, on the "healed" CAD model, the size map is computed using two dimensionless parameters, one controlling the triangle size based on local curvature, and the other controlling the triangle size gradation. Both parameters produce the size map on a coarse background grid computed via Delaunay triangulation. Finally, the triangulation is performed on each CAD patch separately, by using a version of the Advancing Front technique adapted to parametric surfaces.

The second step in this thesis is the establishment of the shape parameterization scheme which will provide a robust method to deform the shape. This is necessary because CAD designs are strongly related to their source parameterization which is defined via feature trees and cannot be accessed via external

software. CAD packages (commercial or not) have different source parameterizations and their vendors are very sensitive about them. The CAD models must, therefore, be parameterized via their surface representation which is available via the BRep format and transferred by standard files. The surfaces comprising a CAD model are trimmed parametric patches that exist as autonomous entities which makes them unfit as shape deformation tools. This is because, displacing them would create C_0 and C_1 discontinuities in the CAD model. To tackle this challenge, all parametric patches are converted to NURBS and a method that imposes desired continuity constraints on their trimming curves is developed. Based on this method, a new parameterization is defined which inherently satisfies these constraints by computing the Kernel of the Jacobian of all constraints.

Apart from continuity constraints, the imposition of various other geometric constraints can be a key factor in CAD-based optimization. Constraints can naturally be defined in a CAD model via the feature trees. However, as mentioned above, the feature trees cannot be accessed and used to deform the shape. Therefore, constraints must be defined on the surfaces of the CAD model. In this thesis, a method is developed to make possible the imposition of multi-node constraints on NURBS patches. Depending on the type of constraints and the complexity of the CAD surfaces, the number of node-wise constraints may become huge. Traditional constrained optimization techniques (i.e. SQP or Gradient projection) assume a priori that the number of constraints is less than or equal to the number of design variables. In the case of NURBS-based optimization, this can become problematic as the design variables are the control point coordinates which are vastly outnumbered by the number of boundary mesh nodes on the design surface. For this reason, all constraints are cast into a single equality constraint. Nodal constraint violations are penalized based on a quartic function that returns a positive value if the constraint is violated and zero otherwise. The penalties are then summed up to create a single constraint. The effectiveness of the single constraint is tested by constraining surface curvature, and enclosing constraints (i.e. constraints that demand that the model moves within a given space). For completeness, the inequality constraint of the volume of a given model is presented, to demonstrate a way of handling non-node wise constraints.

The developed software is applied to the design/optimization of test cases such as passenger cars, automotive cooling and intake ducts, diffusers and turbomachinery blades.

Key words: Computational Fluid Dynamics, Computer Aided Design Continuous Adjoint Methods, Boundary Representation of Surfaces, Shape Optimization, Non-Uniform Rational B-Splines, Constraints

Περίληψη

Η διδακτορική αυτή διατριβή ασχολείται με την εισαγωγή γεωμετριών, παραμετροποιημένων με CAD σχήματα, στη βελτιστοποίηση μορφής βασισμένη στη μέθοδο των συζυγών μεταβλητών. Παρουσιάζονται η μαθηματική διατύπωση και υλοποίηση μεθόδων που καθιστούν εφικτή την εισαγωγή του CAD σχεδιασμού σε βρόχο βελτιστοποίησης καθώς και ο έλεγχος των μεθόδων αυτών σε εφαρμογές της αεροδυναμικής. Ο υπολογισμός των παραγώγων ευαισθησίας διαφόρων αεροδυναμικών συναρτήσεων-στόχων ως προς τις μεταβλητές σχεδιασμού του CAD γίνεται με τη συνεχή συζυγή μέθοδο.

Οι γεωμετρίες CAD μπορεί να έχουν δύο ειδών παραμετροποιήσεις: (α) την παραμετροποίηση δέντρου στοιχείων που ορίζει γεωμετρικές σχέσεις μεταξύ στοιχείων σχεδιασμού και είναι η φυσική παραμετροποίηση των CAD πακέτων και (β) την επιφανειακή παραμετροποίηση που περιγράφεται και μεταφέρεται από το πρότυπο της Συνοριακής Περιγραφής. Η Συνοριακή Περιγραφή (Boundary Representation - BRep) αποτελείται από μία συλλογή επιφανειών που ορίζουν ένα CAD μοντέλο και ορίζονται από πρότυπες μαθηματικές περιγραφές (κυρίως NURBS). Σε αυτήν τη διατριβή, η BRep χρησιμοποιείται ως μέσο περιγραφής των CAD γεωμετριών καθώς η προτυποποιημένη, ανοιχτού κώδικα μορφή της, επιτρέπει τη σύνδεση των γεωμετριών με λογισμικά Υπολογιστικής Ρευστοδυναμικής (ΥΡΔ) καθώς και τη χωρική διαφόρισή τους, που είναι χρήσιμη κατά τη διάρκεια της βελτιστοποίησης. Η παραμετροποίηση δέντρου, είναι σπανίως διαθέσιμη σε ανοιχτή μορφή, πράγμα που κάνει τη σύνδεσή της με τη βελτιστοποίηση αδύνατη.

Το πρώτο βήμα της διατριβής είναι η γένεση ενός ποιοτικού πλέγματος με τριγωνικά στοιχεία στις επιφάνειες του CAD μοντέλου. Αυτό συμβαίνει καθώς, για να εισαχθεί το CAD στο βρόχο βελτιστοποίησης είναι απαραίτητη η πλεγματοποίηση του 3Δ χώρου γύρω (ή εντός) της γεωμετρίας του. Στα λογισμικά πλεγματοποίησης, συνήθως, παρέχεται το όριο του χωρίου που θα πλεγματοποιηθεί σε διακριτή (πιο συχνά τριγωνοποιημένη) μορφή. Η διαδικασία τριγωνοποίησης χωρίζεται σε τρία βήματα: (α) επισκευή μοντέλου, που είναι μια διαδικασία που επιλύει αρκετά συνήθη γεωμετρικά και τοπολογικά σφάλματα CAD γεωμετριών που εμπεριέχονται σε πρότυπα αρχεία (STEP, IGES κλπ.) (β) υπολογισμός μίας χαρτογράφησης του βέλτιστου μεγέθους τριγώνων σε ένα δευτερεύον πλέγμα και (γ) διαδικασία βελτιστοποίησης. Τα τοπολογικά κενά, επιλύονται με ελέγχους εγγύτητας και τα γεωμετρικά κενά με έναν αλγόριθμο ¨ραφής¨ που βασίζεται στην τεχνική Ελαχιστοποίησης Ενέργειας Επιφανειακών Πλακών. Έπειτα, στο ¨επισκευασμένο¨ μοντέλο, υπολογίζονται τα βέλτιστα μεγέθη τριγωνοποίησης με χρήση δύο αδιάστατων παραμέτρων, οι οποίες ελέγχουν το μέγεθος και τη μέγιστη επιτρεπόμενη μεταβολή μεγέθους. Ο συνδυασμός των δύο παράγει ένα χάρτη μεγέθους πλεγματικών στοιχείων πάνω σε ένα δευτερεύον (βοηθητικό) πλέγμα που κατασκευάζεται με τη μέθοδο Delaunay. Τέλος, πραγματοποιείται τριγωνοποίηση σε κάθε επιφάνεια ξεχωριστά με χρήση της μεθόδου Προελαύνοντος Μετώπου, προσαρμοσμένης σε παραμετρικές επιφάνειες.

Το δεύτερο βήμα είναι η δημιουργία ενός σχήματος παραμετροποίησης το οποίο θα παράσχει μία στιβαρή μέθοδο μορφοποίησης του μοντέλου. Το βήμα αυτό είναι αναγκαίο καθώς τα μοντέλα CAD συνδέονται ισχυρώς με τις πηγαίες παραμετροποιήσεις τους οι οποίες ορίζονται μέσω δέντρων στοιχείων και δεν είναι προσβάσιμα από εξωτερικά λογισμικά. Διαφορετικά πακέτα CAD χρησιμοποιούν διαφορετικές παραμετροποιήσεις και οι διανομείς τους δεν τις κάνουν γνωστές. Συνεπώς, τα μοντέλα CAD που θα υποστούν βελτιστοποίηση πρέπει να παραμετροποιηθούν μέσω της επιφανειακής περιγραφής τους που είναι προσβάσιμη μέσω της BRep και μεταφέρεται μεταξύ λογισμικών μέσω πρότυπων αρχείων. Οι επιφάνειες που συντελούν ένα CAD μοντέλο είναι κομμένες παραμετρικές επιφάνειες και είναι αυτόνομες ως οντότητες σε ένα λογισμικό μορφοποίησης. Αυτό τις κάνει ακατάλληλες για εργαλεία μορφοποίησης καθώς, μετατοπίζοντάς τες, θα δημιουργούνταν ασυνέχειες γεωμετρίας και ομαλότητας στο μοντέλο. Για να αντιμετωπιστεί αυτό το πρόβλημα, όλες οι παραμετρικές επιφάνειες μετατρέπονται σε NURBS και επιβάλλονται περιορισμοί συνέχειας στα σύνορα μεταξύ των επιφανειών. Στη συνέχεια, ορίζεται μία νέα παραμετροποίηση που ικανοποιεί τους ανωτέρω περιορισμούς εκ φύσεως, υπολογίζοντας τον μηδενικό χώρο του Ιακωβιανού μητρώου των περιορισμών.

Ένας παράγοντας κλειδί για τη βασισμένη-σε-CAD βελτιστοποίηση είναι η επιβολή γεωμετρικών περιορισμών. Οι περιορισμοί μπορούν με φυσικό τρόπο να ορισθούν μέσω των δέντρων στοιχείων. Ωστόσο, όπως αναφέρθηκε πιο πάνω, τα δέντρα στοιχείων δεν είναι προσβάσιμα και δεν μπορούν να χρησιμοποιηθούν ως μέσο μορφοποίηση των μοντέλων. Οι περιορισμοί πρέπει, συνεπώς, να ορισθούν στις επιφάνειες του CAD μοντέλου. Ανάλογα με τον τύπο του περιορισμού και την πολυπλοκότητα των επιφανειών του CAD, ο αριθμός των περιορισμών που πρέπει να επιβληθούν, μπορεί να είναι πολύ μεγάλος. Για αυτόν το λόγο, παρουσιάζεται μία μέθοδος για τη μείωση του αριθμού των περιορισμών. Με τη μέθοδο αυτή, η παραβίαση του περιορισμού σε κάποιο κόμβο περνά από μία συνάρτηση ποινής η οποία επιστρέφει θετική τιμή σε περίπτωση που υφίσταται παραβίαση και μηδέν σε περίπτωση που όχι. Έπειτα, οι συναρτήσεις ποινής αθροίζονται σε κάθε κόμβο της προς σχεδιασμό επιφάνειας. Η αποτελεσματικότητα της επιβολής των περιορισμών κατ' αυτόν τον τρόπο ελέγχεται επιβάλλοντας περιορισμούς καμπυλότητας και περιορισμούς εγκλεισμού. Για να καλυφθεί και η επιβολή απλούστερων περιορισμών σε NURBS, δείχνεται και διαφορίζεται ο περιορισμός του όγκου.

Το ανεπτυγμένο λογισμικό εφαρμόζεται στο σχεδιασμό / βελτιστοποίηση διαφόρων αντικειμένων όπως επιβατικά αυτοκίνητα, εισαγωγές κινητήρων, πτερύγια συμπιεστών, αγωγοί ψύξης και εισαγωγής οχημάτων και αγωγοί ψύξης πτερυγίων στροβιλομηχανών.

Λέξεις κλειδιά: Υπολογιστική Ρευστο-Δυναμική, Σχεδιασμός με τη βοήθεια Υπολογιστή, Συνεχής Συζυγής Μέθοδος, Συνοριακή Περιγραφή Επιφανειών, Βελτιστοποίηση Μορφής, NURBS, Περιορισμοί

Acronyms

AD	Automatic Differentiation
ADOL-C	A Package for Automatic Differentiation of Algorithms Written in C / C++
ALM	Augmented Lagrangian Method
BFGS	Broyden Fletcher Goldfarb Shanno
BREP	Boundary Representation
CAD	Computer Aided Design
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
CV	Complex Variables
DD	Direct Differentiation
EA	Evolutionary Algorithm
EFS	Equivalent Flow Solution
E-SI	Enhanced Severed Integral
FD	Finite Differences
FFD	Free Form Deformation
FI	Field Integral (Adjoint Method)
GDM	Grid Displacement Method
HVAC	Heat Ventilation and Air Conditioning
IGES	Initial Graphics Exchange Specification
IODA	Industrial Optimal Design using Adjoint CFD
ITN	Innovative Training Network
KKT	Karush Kuhn Tucker
MOO	Multi Objective Optimization
NTUA	National Technical University of Athens
NURBS	Non-Uniform Rational Basis Splines
OCCT	OpenCascade Technology
OPENFOAM	Open Field Operation and Manipulation
PC	Personal Computer
PCOpt	Parallel CFD & Optimization Unit
PDE	Partial Differential Equation
RANS	Reynolds Averaged Navier Stokes
RBF	Radial Basis Functions
SD	Sensitivity Derivatives
SI	Severed Integral
SIMPLE	Semi Implicit Method for Pressure Linked Equations
SOO	Single Objective Optimization
STEP	Standard for Exchange of Product Data
STL	Stereolithography
SQP	Sequential Quadratic Programming
SVD	Singular Value Decomposition

Acknowledgments

I would like to thank all those who have contributed to the completion of this PhD thesis. First of all, I would like to sincerely thank my thesis advisor, Kyriakos Giannakoglou, Professor NTUA, for entrusting to me a challenging thesis topic, for his constant investment in time and effort along the duration of the thesis, our frequent and meaningful scientific conversations and the assiduous proofreading of this text. Our collaboration during my Diploma thesis inspired me to pursue a PhD and his personal guidance led to its completion.

I would also like to express my gratitude towards my industrial advisors, Dr. Eugene de Villiers and Paolo Geremia of Engys Ltd. and Engys Srl. respectively, for the supervision of the work conducted during my time at Engys. Their constant investment in time and effort from day one was of great importance for the completion of this thesis.

I would also like to thank Dr. Spyros Voutsinas, Professor NTUA and Dr. Konstantinos Mathioudakis, Professor NTUA, for their comments on the thesis presentation and their participation in the advisory committee of my thesis.

I would like to express my sincere thanks to the entirety of the IODA ITN, Research Fellowship Programme funded by the European Commission under Marie Skłodowska-Curie Actions, for funding the first 3 years of this work and all the participants, advisors and research fellows for the collaboration during the project. The unique experience of conducting research towards aligned directions in such a scientific community will remain unforgettable to me. Special thanks to Dr. Herve Legrand, Project Manager at OpenCascade Paris and Julien Jomier, Chief Executive Officer at Kitware Lyon for kindly hosting me during my secondments at OCC and Kitware.

I am deeply grateful to the members of the research group of the Parallel CFD & Optimization Unit, NTUA (PCOpt), for their collaboration and the friendly environment during my secondment at NTUA. Especially, I would like to offer my sincerest gratitude to Dr. Evangelos M. Papoutsis-Kiachagias for the the scientific cooperation during our joint publication and the invaluable advice that was offered to me during the entirety of my stay with the group. Special thanks goes to Dr. Varvara Asouti for the administration of the computational infrastructure at PCOpt and the technical support that was provided to me when requested.

Finally, I would like to express the deepest gratitude to my family and close friends, all of whom supported me morally during the toughest times and were there for me throughout this journey.

To my family for their endless love and encouragement.
To my friends for the joyful distractions.
To Eleftheria for her advice, her patience and her faith in me.

Learn something once,
and it's yours for life.
But first you have
to be a dreamer.

Contents

Chapter 1

Introduction

In the last decades, aerodynamic simulations and design optimization have been subjects of broad research in both academia and industries. Computational Fluid Dynamics (CFD) simulations now play an immense role in aerodynamic product designs as they allow the testing of various shapes without the necessity to actually manufacture them. In the same direction, aerodynamic shape optimization is a simulation-driven design process which allows for reaching various optimality conditions (minimization, maximization of various aerodynamically-related objective functions).

CFD and aerodynamic optimization allow for a more cost-effective implementation of product design chain. However, they are not a panacea, as they require considerable amounts of time and expertise in order to properly function and produce results. Improperly setting up a simulation test case can reduce the accuracy of results and not selecting a parameterization that captures the initial design intent, can have severe consequences from the designer point of view. The design intent can include manufacturability criteria and various geometric constraints. Furthermore, it is a common requirement in industrial environments that an optimized shape is further fine-tuned by engineers. For that reason, researchers have attempted to include optimization in the design chain as much as possible, by introducing elements of Computer Aided Design (CAD) in the optimization loop. Keeping a link to the industrial design framework "alive" generates the possibilities for an optimization to deliver a geometry in CAD format.

This PhD thesis focuses on keeping aerodynamic shape optimization connected to the CAD framework and in specific (a) connecting a CAD shape to the surface grid and, in turn, to the surrounding computational mesh, (b) establishing an open-format CAD-based parameterization and (c) imposing geometric constraints while optimizing the shape.

1.1 CFD and Optimization

CFD techniques were developed over the years and were constantly refined using many validation and assessment procedures. A first major example of numerical solutions to a fluid mechanics problem was [**?**], that compiled massive tables of flows over sharp cones by numerically solving the governing differential equations [**?**]. These solutions were carried out on a primitive digital computer at the Massachusetts Institute of Technology. However, the first generation of CFD solutions appeared during the 1950s and early 1960s, spurred by the simultaneous advent of computers and the need to solve the high velocity, high-temperature re-entry body problem. Such physical phenomena generally cannot be solved analytically, even for the simplest flow geometry. Therefore, numerical solutions of the governing equations on a digital computer were an absolute necessity. Examples of these first generation computations are [**?**, **?**, **?**, **?**], all of them for inviscid flows.

More recently, CFD techniques are applied in a much broader scientific spectrum like HVAC (Heat-Ventilation and Air Conditioning) [**?**], nanofluid physics [**?**], biology-related fluid mechanics [**?**] and industrial cases and optimization [**?**, **?**]. Moreover, they portray a crucial role in astrophysics, oceanography, oil recovery, architecture, and meteorology. Various numerical algorithms and software have been developed to perform CFD analysis. Due to the recent advancements in computer technology, numerical simulation for physically and geometrically complex systems can also be evaluated using PC clusters. Large scale simulations even in multi-flow problems on meshes with millions and billions of elements can be achieved within a few hours via super-computers. However, it is completely incorrect to think that CFD describes a mature technology, as there are numerous open questions related to heat transfer, combustion modeling, turbulence, and efficient solution methods or discretization methods, etc.

Optimizing the geometry of a shape subject to some parameterization, pertains to the adjustment of said parameters (design variables) so that an objective function is minimized / maximized. Examples of objective functions are lift and/or drag (coefficients) of a geometry and the total pressure losses between the inlet(s) and outlet(s) of a flow domain. Optimization methods are classified based on the strategy used to update the design variables.

Stochastic optimization methods [**?**, **?**, **?**] mimic natural procedures (including but not limited to evolutionary techniques) in order to update the design variables. The CFD software that evaluates the objective function is treated by the optimization algorithms as a "black box", simply to assess all candidate solutions. The main examples of such algorithms are Evolutionary Algorithms (EAs) [**?**], Simulated Annealing [**?**] and Particle Swarm Optimization [**?**]. Their non-intrusive nature makes the application of stochastic methods to any problem easy and straightforward. Furthermore, constraint imposition comes naturally and these

methods can identify global minima / maxima if left to run for an adequate number of evaluations [**?**]. Their disadvantage is that they are overly time consuming as they require many objective function evaluations. Considering that each evaluation requires a full CFD solution, makes it apparent how impractical they can become. Quite a few remedies to this problem [**?**, **?**] have been proposed but their further discussion is beyond the thesis' scope.

On the other hand, deterministic (gradient-based) optimization methods [**?**, **?**, **?**, **?**] denote all algorithms that follow a rigorous mathematical approach to pursue optima. A common term to describe this field is mathematical programming and is divided into two main categories: line-search [**?**, **?**] and trust-region [**?**, **?**]. Gradient-based methods rely most commonly on the gradient of the objective function w.r.t. the design variables of the optimization (sensitivity derivatives). In cases of more sophisticated methods, second derivatives can also be required. For this reason, the exact or approximate Hessian matrix [**?**, **?**] is computed. Gradient-based methods are in general much faster than stochastic methods as they require less evaluations of the objective function. However, they can stall by becoming trapped in local optima.

In cases of aerodynamic shape optimization, major changes in baseline geometries are seldom desired. Therefore, optimization methods that compute local optima are often sufficient. That, in conjunction with the lower computational cost of gradient-based methods, makes them ideal for cases shown in this thesis.

The key to a successful implementation of a gradient-based optimization method is the computation of the gradient of an objective function J w.r.t. a design variables vector $\vec{b}$ of size N ($\nabla J = (dJ/db_1, \cdots dJ/db_N)$). In CFD-based optimization, the analytical computation of J or ∇J is not feasible and, therefore, numerical approaches must be pursued [**?**].

The first and easiest method to compute a gradient is by finite differences (FD) [**?**]. The idea is to perturb each design variable b_i by a small value ϵ and evaluate J for the initial and perturbed variables. In such a case, the derivative is given by

$$\frac{dJ}{db_i} = \frac{J(b_1, b_2, \cdots, b_i + \epsilon, \cdots, b_N) - J(b_1, b_2, \cdots, b_i, \cdots, b_N)}{\epsilon}$$

which is a first-order (forward difference) scheme. A second-order scheme (central differences)

$$\frac{dJ}{db_i} = \frac{J(b_1, b_2, \cdots, b_i + \epsilon, \cdots, b_N) - J(b_1, b_2, \cdots, b_i - \epsilon, \cdots, b_N)}{2\epsilon}$$

can be used instead, at a higher though computational cost.

An immediately obvious disadvantage of this method is that, in order to compute ∇J, one must evaluate J, N times. This is impractical as, in industrial cases, N is usually a large number. A second disadvantage of this method is that

the accuracy of dJ/db_i depends greatly on the choice of ϵ. Too large ϵ values lead to low accuracy in the derivatives whereas too small values can create round-off errors due to the division of two "almost zero" numbers. In practice, for each variable b_i a trial and error approach must be followed in order to compute a good enough value for ϵ (which obviously adds to the tally of CFD evaluations).

Another gradient computation method is the complex variables (CV) method [**?**]. The idea behind this method is to pertub each b_i in the complex plane by $i\epsilon$ (where $i = \sqrt{-1}$ is the imaginary basis). It can be shown that [**?**]

$$\frac{dJ}{db_i} = \frac{Imag\left[J(b_1, b_2, \cdots, b_i + i\epsilon, \cdots, b_N)\right]}{\epsilon}$$

provides the derivative, where the $Imag$ operator denotes the imaginary part of the expression inside the parenthesis. Computing a derivative with the CV method overcomes the ϵ-related accuracy issue of FD. However, it does not change the fact the number of CFD evaluations are still proportional to N. Furthermore, the CV method requires that the solver and its accompanying utilities must be changed to accommodate complex, rather than real, variables.

The most efficient method for computing ∇J is the method of adjoint variables [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**]. This method treats the optimization problem as constrained by the primal (or state) equations and, by doing so, is able to compute ∇J with a computational cost that is independent of the number of design variables. The continuous adjoint method is used in all cases shown in this PhD thesis. An overview of adjoint methods is presented in Chapter 2.

1.2 CAD Into the Optimization Loop

For CFD-based shape optimization problems, the selection of an appropriate shape parameterization technique is of utmost importance since it determines the quality of the optimal solution. It is responsible for translating a set of design variables into a shape, which is ultimately defined by a set of grid nodes lying on the boundary of the CFD domain. The parameterization highly affects the shapes generated throughout the optimization and, consequently, impacts the convergence rate of the whole process as well as the quality of the optimal design.

In gradient-based methods, an important aspect is the differentiation of the parameterization tool used. This allows for the computation of the $\delta x_k/\delta b_n$ terms (see Eq. 2.12, later on) along the parameterized region S_{Wp}, and (depending on the type of parameterization) even inside Ω.

A detailed survey of shape parameterization techniques for CFD-based (and also structural) optimization is presented in [**?**]. Broadly speaking, they can be classified in CAD-free and CAD-based techniques. CAD-free parameterizations,

such as node-based ones, control the CFD boundary mesh nodes and, therefore, usually lead to very rich design spaces. Their main drawback is that they result in an optimized grid, which then has to be imported back to CAD in order to obtain the optimized geometry for further analysis or manufacturing. This mesh-to-CAD step is a non-trivial task and may require extensive user intervention as well as approximations that can impair the quality of the computed shape. A brief overview of mesh-to-CAD approaches is shown in Sec. 1.3. Moreover, it is usually challenging to impose geometric constraints when using a CAD-free approach.

On the other hand, CAD-based parameterizations use either the native CAD model parameters or the control points of Non-Uniform Rational B-Splines (NURBS) [**?**] patches. The great advantage of incorporating the CAD software into the optimization loop is that, at each design cycle, the best-so-far shape is available in a CAD format. In addition, it is more straightforward to incorporate geometric constraints within a CAD framework and sometimes their imposition can a-priori be guaranteed through the parameterization of the CAD model. On the other hand, differentiating the CAD model's parameterization to obtain $\delta x_k/\delta b_n$ can even be impossible, especially within an industrial environment, where commercial (closed-source) CAD tools are often used.

This is mitigated when using NURBS, as these are a compromise that keeps the geometry in a CAD-compatible format while being a surface-based parameterization. Generally, they belong to a category of polynomial/spline parameterization techniques which are widely used in the optimization of 2D aerodynamic shapes, such as ducts, airfoils and turbomachinery blade sections. They allow for different levels of shape complexity, based on the number of control points used as design variables and/or the polynomial degree, with direct control over the shape's continuity/smoothness. Moreover, geometric sensitivities of NURBS can straightforwardly be obtained by differentiating their underlying rational polynomial expression. Due to the generic BRep format (to be covered in Sec. 1.2.2) all CAD models originating from any CAD package can be transferred via standard files, such as STEP or IGES. BRep, which is described as a set of trimmed NURBS patches, can be considered as a CAD-based parameterization. It is a common practice to generate a CAD model via a native CAD parameterization and, then, transform it to BRep. The BRep models used for the cases shown in this thesis are all generated this way.

Recent research, in the literature, has been focused on how to effectively incorporate CAD-based parameterizations into adjoint-based optimization loops, and various promising approaches have been investigated within the IODA ITN [**?**], funded by the European Commission under Marie Skłodowska-Curie Actions. For example, [**?**, **?**] computed geometric sensitivities (also referred to as design velocities) in commercial CAD packages through the application of FD between discrete representations (i.e. surface triangulations) of the geometry before and after a parameter perturbation. In [**?**], the CAD kernel developed at the von Karman

Institute for turbomachinery design [**?**] was differentiated, using AD (ADOL-C) in forward mode. ADOL-C was also applied by [**?**] for the differentiation of the Open CASCADE Technology (OCCT) CAD kernel, which was used to perform the CAD-based shape optimization of a U-bend cooling duct and a compressor stator blade [**?**].

1.2.1 Feature Tree Parameterization

In a feature-based CAD modelling system, a model part is comprised of individual features which are combined to represent an overall shape. Examples of such features could be pads, pockets, holes, fillets, chamfers etc. These are mainly defined via sketched features or dress-ups. Sketched features are created by drawing 2D profiles and creating a 3D feature by extruding, rotating, sweeping or lofting the sketch. Dress-ups are features like fillets and chamfers, which are created directly on the solid model. Parameters controlling the generation of these features are mainly real variables that control intuitively a shape (lengths, angles, etc.) When creating a model within a CAD modelling system, relations between existing features are created so that the value of one parameter is a function of the values of other parameters of the model. These relations define the design intent of the model and, once applied, the parameters cannot be controlled independently. In the process of model generation, the CAD system automatically creates a series of parameters and relationships in the background. The number of such parameters can range from hundreds to thousands, depending on the model complexity.

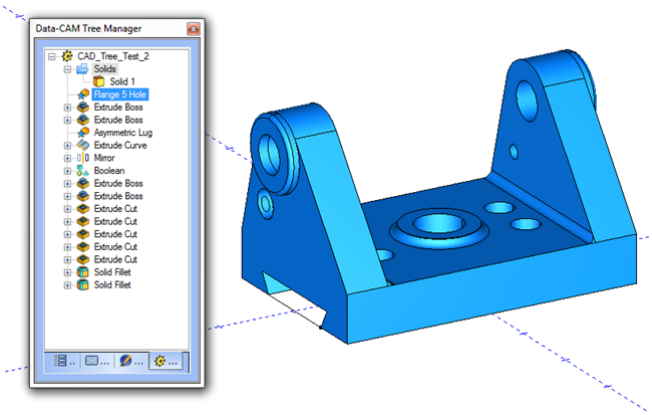


Figure 1.1: Example CAD model controlled via a feature tree. Features are generated in a top-down manner, meaning that more descriptive features of the model are generated first. [**?**].

The relationship between the top-level (user defined parameters) and the bottom level (CAD system defined parameters) is called the feature tree. Parameteriz-

ing a geometry directly via the feature tree can be advantageous w.r.t. the capture of the design intent but disadvantageous w.r.t. its coupling with the adjoint technique. This is because, the source parameterization that a CAD package uses is closed-source and hardly accessible.

1.2.2 BRep and NURBS Geometry

BRep [**?**, **?**] is a method for representing shapes in solid modeling. A solid is represented by the BRep format using surface elements defining the interface between solid and non-solid volumes. The BRep format is composed of two parts: topological data and geometry. The topology of a BRep is created using vertices, edges, faces, shells and, finally, solids. Regarding their underlying geometry:

1. Vertices: A vertex is merely a point in space.

2. Edges: Edges are represented by curves bounded by the points describing their boundary vertices. In the general case, the geometry of an edge is only a segment of its underlying curve, since the topological bounds of an edge and the geometrical bounds of a curve are not strictly identical.

3. Faces: Faces are represented by surfaces bounded by a closed loop of edges. Similarly to edges, the geometry of faces is, in general, a part of its underlying surface, since its boundary loop of edges does not coincide with the natural bounds of the surface.

4. Shells: A shell is composed of multiple faces connected to each other via their bounding edges and has no particular underlying geometry.

5. Solids: Similarly to a shell, a solid does not have an underlying geometry and is, practically, the volume bounded by a collection of shells.

The mathematical description of curve and surface elements of a BRep model could vary. Elementary curves or surfaces such as circular arcs, planes or cylinders etc., could be stored explicitly. However, more complex elements are stored in parametric form, most commonly NURBS. The conversion from elementary curves or surfaces to NURBS is trivial, thus the BRep geometry will always be handled as NURBS geometry for uniformity.

NURBS geometry is a generalization of B-spline geometry. B-splines basis functions are defined according to the recursive formula [**?**]

$$N_i^p(u) = \frac{u - u_i}{u_{i+p} - u_i} N_i^{p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1}^{p-1}$$
$$N_i^1(u) = \begin{cases} 1 & u_i \leq u \leq u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

where p is the user-defined degree of the functions, i is the basis index, u is the parameter and the indexed u variables are the knots taken from a non-decreasing knot vector of length s: $U = \begin{bmatrix} u_1 & u_2 & \cdots & u_{s-1} & u_s \end{bmatrix}^T$. The B-spline basis functions can be used to interpolate a number of control values n ($i \in [1, n]$), with $s = n + p + 1$.

In the following chapters, NURBS shapes consist of multiple faces bounded by edges. Points on the edges are expressed in terms of a parametric coordinate u as:

$$\vec{C}(u) = \frac{\sum_{i=1}^n N_i^p(u) w_i \vec{P}_i}{\sum_{k=1}^n N_k^p(u) w_k} = \sum_{i=1}^n R_i^p(u) \vec{P}_i \tag{1.2}$$

where n is the number of control points $\vec{P}_i$ of each curve and w_i the respective weights.

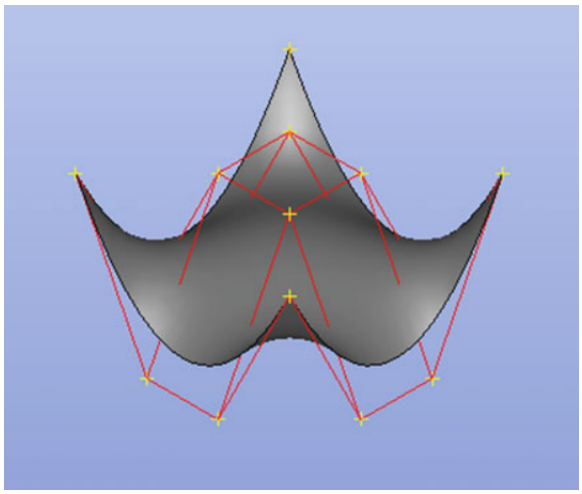


Figure 1.2: A single NURBS patch along with its 4×4 control polygon in red. The patch is created with $n_u = n_v = 4, p_u = p_v = 3$.

Similarly, points on the surfaces which represent faces are expressed in terms

of a pair of parametric coordinates (u, v) as:

$$\vec{S}(u,v) = \frac{\sum_{i=1}^{n_u}\sum_{j=1}^{n_v} N_i^{p_u}(u)N_j^{p_v}(v)w_{i,j}\vec{P}_{i,j}}{\sum_{k=1}^{n_u}\sum_{l=1}^{n_v} N_k^{p_u}(u)N_l^{p_v}(v)w_{k,l}} = \sum_{i=1}^{n_u}\sum_{j=1}^{n_v} R_{i,j}(u,v)\vec{P}_{i,j} \tag{1.3}$$

where n_u, n_v denote the number of control points $\vec{P}_{i,j}$ and p_u, p_v the bases' degrees per parametric direction. $w_{i,j}$ denote the control point weights.

For the NURBS surfaces used in Chapter 4, Eq. 1.3 will be written with a single instead of a double sum after setting $m = j + (i-1)n_v, m \in [1, n_u n_v]$.

$$\vec{S}(u,v) = \sum_{m=1}^{n} R_m(u,v)\vec{P}_m \tag{1.4}$$

where $n = n_u n_v$ (not to be confused with variable n of Eq. 1.2).

1.2.3 Trimmed Patches

Trimmed NURBS patches enable the representation of a much higher variety of shapes with less complex surfaces. The trimming procedure involves the creation of a closed wire of curves lying on the the surface.

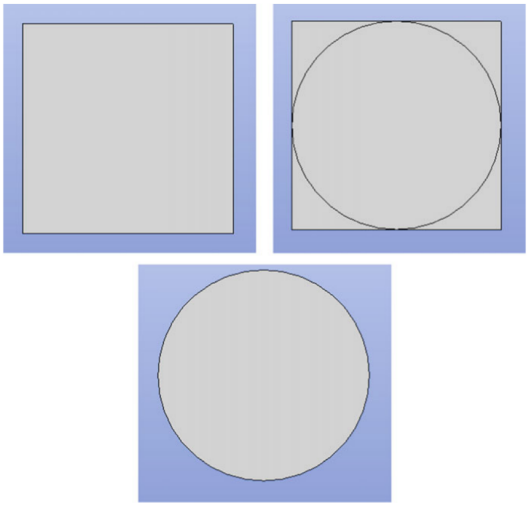


Figure 1.3: Creation of a circular disk using a rectangular planar surface and a circular curve. Top-left: The original surface defined by four co-planar control points, one at each corner. Top-right: The original surface along with the circular curve which lies on it. Bottom: The outcome of the trimming process.

The surface is then trimmed along this wire which becomes the new surface

bound. The number of curves the wire consists of is arbitrary and, thus, a multi-sided patch can be created using a single NURBS surface. A simple example of the trimming process is shown in Fig. 1.3. where the goal is to create a circular disk using a planar surface with $p_u = p_v = 1$.

1.3 Mesh-to-CAD Literature Review

In cases where a CAD-free approach is used, the geometry described by the optimal shape's boundary mesh has to be approximated with high fidelity by a high-order geometric standard that is also CAD-compatible. The mesh-to-CAD conversion deals with this challenge by approximating the boundary mesh using NURBS patches. The mesh-to-CAD conversion belongs to the family of shape reverse engineering methods. These methods mainly consist of three phases.

1. **Surface segmentation:** In general, one could try to fit a surface mesh over the whole surface. However, the continuously growing complexity of aerodynamic shapes will make a high fidelity representation impossible. Moreover, a robust global parameterization of the shape would require a 3D (x, y, z) to 2D (u, v) mapping of every point of the cloud, that would maintain the connectivity of the external surface grid (no elements should be inverted during the mapping). In order to avoid these drawbacks of global fitting, the segmentation of the surface is pursued. The resulting smaller and flatter surfaces (patches) enable faster and easier parameterization. It is obvious that the more complex an object is, the more patches are required and more continuity conditions (1^{st}, 2^{nd} derivatives etc.) must apply along the borderlines.

2. **Surface fitting:** The process of surface fitting includes two sub-steps. Firstly, each point of the patch should be mapped onto 2D, in order to acquire its corresponding parameters. Secondly, these parameters should be used as a starting point to compute the remaining NURBS requisites (knots, weights, control point coordinates).

3. **CAD model creation:** The last step involves the use of standard STEP or IGES file writers to produce the final CAD-compatible object.

1.3.1 Surface Segmentation Techniques

Segmenting a discrete surface (represented by the boundary of a mesh) can be a non-trivial task. It requires a very effective application of feature detecting, through sharp vertex or edge identification. In [**?**], a segmentation technique is applied to arbitrary triangular grids. The approach is based on two steps: a

boundary based region segmentation and a boundary rectification. During the first step, a pre-processor identifies sharp edges and vertices and the curvature tensor is calculated for each vertex. Then, vertices are grouped into clusters using K-means clustering, according to their principal curvature values κ_1 and κ_2. A region growing algorithm is then used to generate triangles into connected labelled regions according to vertex clusters. Finally, a region adjacency graph is processed and reduced in order to merge similar regions according to several criteria such as curvature similarity, size and common perimeter. During the second step, boundary edges are extracted from the previous region segmentation step. Then, for each of them, a metric is computed which notifies a degree of correctness. The angles between a boundary edge and the minimum curvature directions of its vertices represent a good boundary score. According to this score, estimated correct boundary edges are marked and used in a contour tracking algorithm to complete the final correct boundaries of the object. The contour tracking algorithm makes sure that each surface patch consists of closed edges. Resulting patches can be parameterized by projecting their points onto a base surface.

In [**?**], a hybrid approach is used. The method is applied to triangular grids and uses two algorithms in conjunction: a vertex classification method and a feature-edge identification method. The vertex-based algorithm works by assigning a scalar quantity λ to each vertex and, then, by grouping vertices with similar (with some tolerance) λ into patches. A suitable choice for this quantity could be the local curvature. The feature-edge based algorithm involves the computation of the normal vector of all triangular faces and, then, the edges between faces the normal vectors of which form angles greater than a given threshold are considered as feature edges. Those edges are, in turn, the boundaries of a patch. After defining the feature edges and vertices, the method applies the segmentation algorithm based on the computed λ values.

The segmentation method proposed in [**?**] is based on the concept that any primitive object can be constructed as a sweep of a parametric surface (cross-section) on a 3D-space curve. The method is, therefore, divided into two parts: (a) identification of primitives "hugged" by 3D surfaces and (b) identification of the 3D curves and the respective sweeping surfaces (skeletonization).

One of the most popular surface segmentation techniques is the watershed method which has best been described by [**?**]. The watershed algorithm calculates a height function for every vertex of a boundary mesh. The function could be (for a 3D case) the curvature at each vertex or the geodesic distance from some already defined feature edge. In more complex surface combinations, different geometric functions are preferred. Now, considering that every surface could be transformed into a 2D object (planar) without any violation of topology, then every vertex on that surface requires two parametric coordinates to be located: u and v. The height function h changes at each surface location, thus, $h = f(u, v)$. The

visualization of h in 3D would be similar to the one shown in Fig. 1.4. This gives some insight to the naming of the method: local minima of the height function and the regions surrounding them form catchment basins, while continuous iso$-h$ lines form plateaus. Plateaus could be thought as the waterlines that appear if the basins are flooded. Maxima of h define watershed lines or (watershed plateaus) that create segmentation boundaries. After defining the regions that surround local minima and maxima, a labelling process with adjacency tree creation is initiated. The vertices of the surface are, then, grouped based on which basin they belong to.

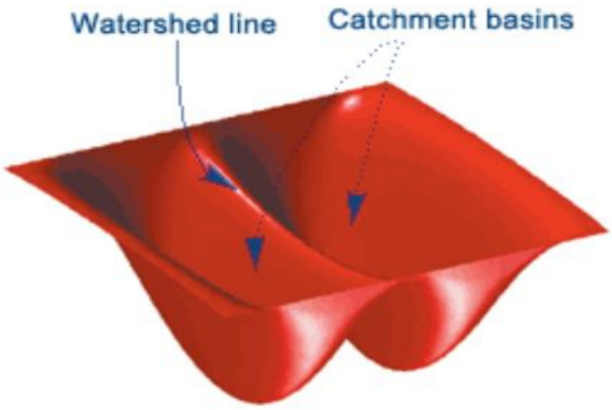


Figure 1.4: A visualized height function (red) along with a watershed line. [**?**].

Based on the watershed algorithm, many methods have been devised. A prime example is the fast marching watershed algorithm shown in [**?**]. The coupling of the classic watershed method with the Fast Marching algorithm [**?**] allows for computing the shortest geodesic paths from a point of a mesh to specified feature. There are three steps in the algorithm: (a) use of a hill-climbing algorithm to watershed segment a triangular grid, (b) definition of a height map appropriate for the minima rule using local principal curvatures and (c) application of morphological operations to improve the connectivity of the resulting segmentation.

Other algorithms introduce similarity parameters such as slippage [**?**]. Such algorithms aim to reconstruct a design feature tree (similar to a CAD package's) by trying to match regions of the grid with primitive objects and splines. Slippage is the deviation of the point cloud from these primitives.

The above mentioned methods can work well on their own. However, for more complex shapes, coupling with auxiliary methods may, sometimes, be necessary. For instance, [**?**] applied a set of tests to the points of the boundary of a mesh which defined the geometry of an object. Those tests can indicate if the points at a neighbourhood of the boundary mesh can form primitives. The tests are based on indicators which must be computed at each point at a given part of a

grid. The indicators could be geometric, similarity or error metrics. Geometric indicators could be normal vectors, curvatures, best fitting axis of revolution etc. Error indicators could be normalized errors of least-squares fitting. Similarity indicators result from the deviation of geometric and error indicators within a region. After computing indicators, some statistical tests take place to define if the measured indicators and the primitive they point to is acceptable and within a statistical trust region.

In [**?**], the target is to segment a surface and achieve the easiest possible mapping from 3D to 2D. It involves genus reduction and feature identifications through Reeb graphs.

1.3.2 Surface Fitting Techniques

Surface fitting is the main step of a shape reverse engineering algorithm. Generally speaking, the fitting process consists of two steps: (a) parameterization of the points to be fitted i.e. a 3D to 2D mapping of all the points and (b) the computation of the surface. Surface fitting can be case-specific (applied to a specific type of geometry) or more generalized. In the latter case, the fitting algorithm must almost always be coupled with a surface segmentation method.

An example of a case-specific fitting technique is shown in [**?**]. This fitting technique was used for representing wings and airfoils. It requires no segmentation of the grid as it utilizes a geometric processing of the grid points to acquire a global parameterization. Then, a NURBS surface is computed in two steps: Firstly, the weights of the control points are set equal to each other making the NURBS surface a B-splines surface. The latter must interpolate the grid points for the corresponding parameterization which gives a linear equation for calculating initial control point positions. Secondly, starting from the computed initializations, the control point positions and the weights are optimized with a chosen gradient-based method.

The method in [**?**] is another case-specific method that works best with objects that can easily be described in spherical or cylindrical coordinates (i.e. surfaces of revolution etc.). The method is best coupled with segmentation methods that produce such primitive objects. Firstly, the coordinates of the points to be fitted are transformed from cartesian to spherical or cylindrical coordinates. Then, a base sphere or cylinder is defined which is used to compute the parameters that correspond to each point. Finally, a NURBS surface is computed.

Another case-specific method is shown in [**?**] and it tackles fitting of human bones. This application is not related to CFD but for the sake of completeness it was deemed as a necessary addition. In this case, a voxel model is extracted from the base surface to generate a rectangular net of curves. This is done by projecting the exterior faces of the extracted voxels onto the base implicit surface. After generating the interior points and tangential vectors along the boundaries

in each rectangular region, a B-spline surface is reconstructed by interpolating the rectangular array of points as well as the boundary derivatives.

Another algorithm, [**?**], comes from a more general mesh to NURBS technique and it requires the segmentation of a given object. For each surface segment, an initial base surface is computed and, on that surface, each point is projected in order to acquire the parameterization. The base surface is itself a NURBS surface that is computed with some of the grid nodes as control points. After the parameterization is done, the NURBS control points and weights are computed through a least squares fitting scheme.

None of the above mentioned methods (or any other in the literature) have however been coupled with an adjoint tool for optimization. Furthermore, commercial mesh to CAD exists but is not being considered since it would be difficult to connect to the optimization framework. Coupling the display of adjoint sensitivities with the CAD parameterisation and constraint definition interface can, therefore, provide the opportunity for informed control of the optimization system. Significant efficiency can also be realised by limiting parameter adjustments to regions where significant benefits are likely to be realised. Thus, the use of a commercial package for mesh-to-NURBS conversion is not an option, since there is no connection with optimization tools.

It is obvious that, in order to pass from CAD-free to CAD-based optimization, not only the mesh-to-CAD conversion, but also a methodology to define optimization parameters (and constraints) on a CAD model, is required. Optimization parameters however need to be adjusted before the optimization cycle starts. It is thus logical to start the development of the methodology with the implementation of the CAD based systems and the constraints definition interface.

1.4 Objectives of this Thesis

The work of this thesis aims to contribute to three main CAD-related topics, all used in conjunction with adjoint-based optimization: (a) The CAD-to-Surface Grid topic where a CAD model is automatically discretized while maintaining the link of its discrete form to the CAD. (b) The CAD-based parameterization topic, where a new CAD-based parameterization scheme is created to enable the inclusion of standard CAD files into the optimization loop and, finally, (c) the constraint definition topic, where the handling of various geometric constraints is shown.

In all topics, the geometry is assumed to be available via a standard file desciption that contains the BRep described by NURBS patches. The geometry contained in such files is then handled for each topic, separately. Initially, it is discretized leading to the generation of a computational mesh around it. Then, it is differentiated in a way that ensures watertightness and smoothness during optimization. Finally, geometric constraints are imposed.

1.5 Thesis Outline

This PhD thesis consists of 9 chapters, including the current one. If necessary, each chapter includes an individual literature review that covers its relevant topics.

Chapter 2 presents a literature review on adjoint techniques (both discrete and continuous) and briefly presents the continuous adjoint formulation that was used in this PhD thesis.

Chapter 3 presents the development of a method for triangulating CAD models. A shape healing algorithm that solves geometric and topological defects is developed. Mainly, a triangulation method based on an adapted Advancing Front algorithm is described and developed. Various CAD geometries are then triangulated to show the effectiveness of the algorithm and the programmed software. The triangulated grids coming off CAD models can then be used as a basis to mesh the space around (or inside of) them.

In chapter 4, the next step of an optimization is shown. In specific, a method to parameterize a given BRep geometry is developed as the BRep itself is unsuitable for optimization purposes. This stems from the fact that the trimmed patches of a BRep must, firstly, be constrained because a potential pertubation of the NURBS control points would generate geometric holes in the geometry. For this reason, a method to constrain the control point displacement is proposed and tested in various cases.

Chapter 5 is concerned with the imposition of geometric constraints. All constraints shown are imposed using surface definitions, thus making them ideal to be coupled with the BRep format. In specific, packaging constraints are portrayed along with maximum surface curvature and shape volume.

In chapter 6, the methods presented in chapters. 3 - 5 are all tested in various cases. In specific, in sec. 6.1 the methods are tested in the optimization process of the stator case of the Technical University of Berlin. In sec. 6.2, all the methods presented are tested in the case of a concept intake manifold duct. Finally, in sec. 6.3, the methods of this thesis are tested by performing a Multi Objective Optimization (MOO) on the well studied (from a simulation point of view) ERCOFTAC UFR 4-06 diffuser case.

In the last chapter, conclusions are drawn and suggestions for future work are made.

The author has used both the HELYX Open-Source CFD for Enterprise [**?**] developed by ENGYS Srl and the OpenFOAM distribution developed by the PCOpt/LTT. In addition, the adjoint codes of both these CFD codes have been utilized while performing optimization runs. All the methods and algorithms presented in chapters. 3-5, are developed within the open-source CFD toolbox OpenFOAM, version 2.3.1, the open-source CAD Kernel OpenCascade Technology version 7 [**?**] and the SALOME platform version 8 [**?**].

Chapter 2

The Adjoint Method

There is a long history of the use of adjoint equations in optimal control theory [**?**]. In CFD, the first use of adjoint equations for design was by [**?**], but within the field of CFD in aeronautics, the adjoint was firstly used in [**?**, **?**, **?**, **?**, **?**, **?**] where the adjoint approach for potential flow, the Euler equations and the Navier-Stokes equations was developed. The complexity of the applications within these papers also progressed from 2D airfoil optimization to 3D wing design and, finally, to complete aircraft configurations. An overview of recent developments in adjoint-based design methods is provided in [**?**]. Other relevant works can be seen in [**?**, **?**, **?**, **?**] on unstructured grids using discrete adjoint, and in [**?**, **?**] in using automatic differentiation software to create the adjoint code from an original CFD code.

The pioneering works of PCOpt/NTUA have employed the constinuous adjoint method in various real-world applications, in both aeronautical [**?**] and automotive industries [**?**], [**?**], [**?**]. In [**?**] the steady continuous adjoint method was derived for both inviscid and viscous flows while [**?**] addressed the unsteady continuous adjoint method assisted by the proper generalized decomposition method. Turbulent flows are tackled by [**?**] and [**?**] for both steady and unsteady flows while [**?**] includes the cut-cell method with mesh adaptation in the adjoint code. Further development of the cut-cell method allows for the handling of cavitating flows in [**?**]. Topology optimization is handled by [**?**] while the handling of grid sensitivities is shown in [**?**].

2.1 The State Equations

The adjoint method treats the optimization problem as constrained by the primal (or state) equations. For the incompressible, turbulent steady flows studied here, these are the Reynolds-Averaged Navier-Stokes (RANS) equations. In order to account for the turbulence, the Spalart-Allmaras [**?**] turbulence model is additionally solved and differentiated in a work firstly shown by PCOpt [**?**]. This is done

in order to avoid making the "frozen turbulence" assumption [**?**] when computing the SDs as, by doing so, their accuracy could significantly be damaged [**?**, **?**]. Altogether, the state equations are

$$R^p = -\frac{\partial v_i}{\partial x_i} = 0 \tag{2.1}$$

$$R_i^v = v_j \frac{\partial v_i}{\partial x_j} + \frac{\partial p}{\partial x_i} - \frac{\partial \tau_{ij}}{\partial x_j} = 0,\ i = 1, 2, 3 \tag{2.2}$$

$$R^{\tilde{\nu}} = v_j \frac{\partial \tilde{\nu}}{x_j} - \frac{\partial}{\partial x_j}\left[\left(\frac{\nu + \tilde{\nu}}{\sigma}\right)\frac{\partial \tilde{\nu}}{\partial x_j}\right] - \frac{c_{b2}}{\sigma}\left(\frac{\partial \tilde{\nu}}{\partial x_j}\right)^2 - \tilde{\nu}\mathcal{P}(\tilde{\nu}) + \tilde{\nu}\mathcal{D}(\tilde{\nu}) = 0 \tag{2.3}$$

$$R^{\Delta} = \frac{\partial (c_j \Delta)}{\partial x_j} - \Delta \frac{\partial^2 \Delta}{\partial x_j^2} - 1 = 0,\ c_j = \frac{\partial \Delta}{\partial x_j} \tag{2.4}$$

and they correspond to the continuity, momentum, turbulence (Spalart-Allmaras) and Hamilton-Jacobi equations. The Hamilton-Jacobi equation (Eq. 2.4) is solved for the field Δ which denotes the distance of cell centres from the nearest wall and is required by the Spalart-Allmaras model. p, v_i, τ_{ij} denote the static pressure divided by the density, the Cartesian velocity components and the stress tensor components ($\tau_{ij} = (\nu + \nu_t)\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)$) respectively. Turbulent viscosity ν_t is expressed in terms of $\tilde{\nu}$ as follows

$$\nu_t = \tilde{\nu} f_{v1} \tag{2.5}$$

The production and dissipation terms are given by

$$\mathcal{P}(\tilde{\nu}) = c_{b1}\tilde{Y}\ ,\ \mathcal{D}(\tilde{\nu}) = c_{w1} f_w(\tilde{Y}) \frac{\tilde{\nu}}{\Delta^2} \tag{2.6}$$

where $\tilde{Y}$ is computed as

$$\tilde{Y} = Y f_{v3} + \frac{\tilde{\nu}}{\Delta^2 \kappa^2} f_{v2} \tag{2.7}$$

with $Y = \left| e_{ijk} \frac{\partial v_k}{\partial x_j} \right|$ being the vorticity magnitude. The turbulence model functions read

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}\ ,\ f_{v2} = \frac{1}{\left(1 + \frac{\chi}{c_{v2}}\right)^3}$$

$$f_{v3} = \frac{(1+\chi f_{v1})}{c_{v2}}\left[3\left(1+\frac{\chi}{c_{v2}}\right)+\left(\frac{\chi}{c_{v2}}\right)^2\right]\left(1+\frac{\chi}{c_{v2}}\right)^{-3}$$
$$\chi = \frac{\tilde{\nu}}{\nu}\ ,\ f_w = g\left(\frac{1+c_{w3}^6}{g^6+c_{w3}^6}\right)^{1/6}$$
$$g = r + c_{w2}(r^6 - r)\ ,\ r = \frac{\tilde{\nu}}{\tilde{Y}\kappa^2\Delta^2}$$

and the constant values are as in table 2.1 [**?**]. Finally, e_{ijk} is the Levi-Civita symbol.

Constant	c_{b1}	c_{b2}	κ	σ	c_{w1}	c_{w2}	c_{w3}	c_{v1}	c_{v2}
Value	0.1355	0.622	0.41	2/3	$\frac{c_{b1}}{\kappa^2}+\frac{1+c_{b2}}{\sigma}$	0.3	2	7.1	5

Table 2.1: Values of constants used by the Spalart-Allmaras turbulence model.

Both primal and adjoint solvers were implemented within the open-source CFD toolbox OpenFOAM. The PDEs given by Eqs. 2.1-2.4 and their adjoint counterparts (later derived in Eqs. 2.14-2.17), are discretized and solved on unstructured grids using the cell-centered, collocated, finite-volume infrastructure provided by OpenFOAM. The pressure equations for the aforementioned sets of PDEs are formulated using a SIMPLE-like algorithm [**?**]. All convection terms are discretized using second-order upwind schemes, central schemes are used for the diffusion fluxes including a correction for non-orthogonality and the Gauss divergence scheme is used for the computation of spatial gradients, with a linear interpolation of the differentiated field values from the cell-centers to the cell-faces.

2.2 The Discrete Adjoint Technique

For the sake of completeness and due to its simpler expressions the Discrete Adjoint technique is firstly shown. After discretization, a vector of state equations $\vec{R}(\vec{U}(\vec{b}),\vec{b})$, that containts the residuals of Eqs.2.1-2.4 at every cell, becomes available. $\vec{U}$ denotes the vector containing all the state variables p, v_i, ν_t and Δ at every cell.

By doing so, the following Lagrangian is formed

$$L(\vec{U}(\vec{b}),\vec{b}) = J(\vec{U}(\vec{b}),\vec{b}) + \vec{\psi}^T\vec{R}(\vec{U}(\vec{b}),\vec{b})$$

where $\vec{\psi}$ denotes Lagrange multipliers (or adjoint variables). Differentiating the Lagrangian w.r.t. $\vec{b}$ yields

$$\begin{aligned}\frac{dL}{d\vec{b}} &= \frac{\partial J}{\partial \vec{b}} + \frac{\partial J}{\partial \vec{U}}\frac{d\vec{U}}{d\vec{b}} + \vec{\psi}^T\left(\frac{\partial \vec{R}}{\partial \vec{b}} + \frac{\partial \vec{R}}{\partial \vec{U}}\frac{d\vec{U}}{d\vec{b}}\right) \\ &= \left(\frac{\partial J}{\partial \vec{b}} + \vec{\psi}^T\frac{\partial \vec{R}}{\partial \vec{b}}\right) + \left(\frac{\partial J}{\partial \vec{U}} + \vec{\psi}^T\frac{\partial \vec{R}}{\partial \vec{U}}\right)\frac{d\vec{U}}{d\vec{b}}\end{aligned}$$

In the expression above, the term $d\vec{U}/d\vec{b}$ is both computationally expensive and memory intensive regarding storage. Therefore, $\vec{\psi}$ is computed so as to nullify the multiplier of $d\vec{U}/d\vec{b}$, namely

$$\frac{\partial \vec{R}}{\partial \vec{U}}^T \vec{\psi} = -\frac{\partial J}{\partial \vec{U}}^T$$

which stand for the discrete adjoint equations. The computation of $\vec{\psi}$ is, practically, as expensive as the flow solution which makes apparent the strength of this method: The time required to compute N derivatives is independent of N. The sensitivity derivatives (SDs) are then given by

$$\frac{dL}{d\vec{b}} = \frac{dJ}{d\vec{b}} = \frac{\partial J}{\partial \vec{b}} + \vec{\psi}^T\frac{\partial \vec{R}}{\partial \vec{b}}$$

2.3 The Continuous Adjoint Technique

Contrary to the discrete adjoint, the continuous adjoint method, derives the adjoint equations in differential form prior to discretization of the state ones. The FI continuous adjoint method is derived for the state equations (2.1 - 2.4) as shown in the works of PCOpt [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**]. This abbreviation is due to the presence of Field Integrals in the final expression of the SDs. Different methods to handle the grid sensitivities in the interior of the domain lead to different adjoint formulations. The mathematical development of the FI adjoint, that is briefly presented herein, makes use of the total variations of all differentiated quantities Φ, w.r.t. to a design variable b_n

$$\frac{\delta \Phi}{\delta b_n} = \frac{\partial \Phi}{\partial b_n} + \frac{\partial \Phi}{\partial x_k}\frac{\delta x_k}{\delta b_n} \tag{2.8}$$

The first term on the right hand side (r.h.s) of Eq. 2.8 pertains to the Φ variations caused by the update of the design variables at the same location. The second

term pertains to the deformation of space, i.e. the displacement of the boundary and internal grid nodes (in the discrete sense).

Generally, J may consist of two integrals, a surface one J_S and a volume one J_Ω [**?**, **?**, **?**, **?**, **?**, **?**]:

$$J = J_S + J_\Omega = \int_S j_{S_i} n_i dS + \int_\Omega j_\Omega d\Omega \tag{2.9}$$

where j_S and j_Ω are the integrands (defined along the boundary S and the volume Ω of the computational domain). n_i denotes components of the outward looking normal vector $\vec{n}$ defined along the boundary S. or the objective functions used in this thesis, the inclusion of the sub-integral J_Ω is unnecessary and, therefore, all J_Ω related terms are omitted. Differentiating $J = J_S$ w.r.t to a design variable b_n gives [**?**, **?**, **?**, **?**, **?**, **?**]

$$\begin{aligned}\frac{\delta J}{\delta b_n} = &\int_S \left(\frac{\partial j_{S_k}}{\partial v_i} n_k + j^v_{S,i}\right)\frac{\delta v_i}{\delta b_n} dS + \int_S \left(\frac{\partial j_{S_i}}{\partial p} n_i + j^p_S\right)\frac{\delta p}{\delta b_n} dS + \int_S \frac{\partial j_{S_i}}{\partial \tau_{kj}} n_i \frac{\delta \tau_{kj}}{\delta b_n} dS \\ &+ \int_S j^{\tilde{\nu}}_S \frac{\delta \tilde{\nu}}{\delta b_n} dS + \int_{S_{Wp}} j^g_{S_{i,k}} \frac{\delta x_k}{\delta b_n} n_i dS + \int_{S_{Wp}} j_{S_i} \frac{\delta n_i}{\delta b_n} dS + \int_{S_{Wp}} j_{S_i} n_i \frac{\delta (dS)}{\delta b_n}\end{aligned} \tag{2.10}$$

where j^Φ_S accounts for terms multiplying $\delta\Phi/\delta b_n$ inside surface integrals. Term j^g_S accounts for the dependencies of j_S on the boundary geometry and S_{Wp} denotes the parameterized parts of the wall regions of S. This distinction is made as $\delta x_k/\delta b_n = \delta n_i/\delta b_n = \delta(dS)/\delta b_n = 0$, along the non-parameterized (non-displaceable) parts of S.

Next step is to define the augmented (Lagrangian) objective function. L as

$$L = J + \int_\Omega q R^p d\Omega + \int_\Omega u_i R^v_i d\Omega + \int_\Omega \tilde{\nu_\alpha} R^{\tilde{\nu}} d\Omega + \int_\Omega \Delta^\alpha R^\Delta d\Omega \tag{2.11}$$

where q, u_i, $\tilde{\nu_\alpha}$, Δ^α denote the adjoint pressure, velocity, the adjoint to the turbulence model variable and distance from the wall, respectively. Differentiating L w.r.t. b_n yields

$$\frac{\delta L}{\delta b_n} = \frac{\delta J}{\delta b_n} + \int_\Omega q \frac{\delta R^p}{\delta b_n} d\Omega + \int_\Omega u_i \frac{\delta R^v_i}{\delta b_n} d\Omega + \int_\Omega \tilde{\nu_\alpha} \frac{\delta R^{\tilde{\nu}}}{\delta b_n} d\Omega + \int_\Omega \Delta^\alpha \frac{\delta R^\Delta}{\delta b_n} d\Omega \tag{2.12}$$

Since the residuals of all the Partial Differential Equations (PDEs) are equal to zero (before and after any design variables' update), the gradient of the objective function can be given by $\delta L/\delta b_n$ instead of $\delta J/\delta b_n$. In order to compute $\delta L/\delta b_n$, the differentiated terms inside the integrals of Eq. 2.12 are developed. This yields [**?**]

$$
\begin{aligned}
\frac{\delta L}{\delta b_n} = & \int_S \left(u_i v_j n_j + \tau_{ij}^{\alpha} n_j - q n_i + \tilde{\nu}_\alpha \tilde{\nu} \frac{C_Y}{Y} e_{jql} \frac{\partial v_l}{\partial x_q} e_{jki} n_k + \frac{\partial j_{S_k}}{\partial v_i} n_k + j_{S,i}^{v} \right) \frac{\delta v_i}{\delta b_n} dS \\
& + \int_S \left(u_i n_i + \frac{\partial j_{S_i}}{\partial p} n_i + j_S^p \right) \frac{\delta p}{\delta b_n} dS + \int_S \left(-u_i n_j + \frac{\partial j_{S_k}}{\partial \tau_{ij}} n_k \right) \frac{\delta \tau_{ij}}{\delta b_n} dS \\
& + \int_S \left[\tilde{\nu}_\alpha v_j n_j - \frac{\tilde{\nu}_\alpha}{\sigma} \frac{\partial \tilde{\nu}}{\partial n} + \frac{\partial \tilde{\nu}_\alpha}{\partial n} \left(\frac{\nu + \tilde{\nu}}{\sigma} \right) - 2 \tilde{\nu}_\alpha \frac{c_{b2}}{\sigma} \frac{\partial \tilde{\nu}}{\partial n} + j_S^{\tilde{\nu}} \right] \frac{\delta \tilde{\nu}}{\delta b_n} dS \\
& - \int_S \tilde{\nu}_\alpha n_j \left(\frac{\nu + \tilde{\nu}}{\sigma} \right) \frac{\delta}{\delta b_n} \left(\frac{\partial \tilde{\nu}}{\partial x_j} \right) dS + \int_S \left(\Delta^\alpha \frac{\partial \Delta}{\partial n} + \Delta^\alpha c_j n_j \right) \frac{\delta \Delta}{\delta b_n} dS \\
& + \int_\Omega \left[u_j \frac{\partial v_j}{\partial x_i} - \frac{\partial (u_i v_j)}{\partial x_j} - \frac{\partial \tau_{ij}^{\alpha}}{\partial x_j} + \frac{\partial q}{\partial x_i} + \tilde{\nu}_\alpha \frac{\partial \tilde{\nu}}{\partial x_i} - \frac{\partial}{\partial x_l} \left(\tilde{\nu}_\alpha \tilde{\nu} \frac{C_Y}{Y} e_{mjk} \frac{\partial v_k}{\partial x_j} e_{mli} \right) \right] \frac{\delta v_i}{\delta b_n} d\Omega \\
& + \int_\Omega \left\{ - \frac{\partial (v_j \tilde{\nu}_\alpha)}{\partial x_j} - \frac{\partial}{\partial x_j} \left[\left(\frac{\nu + \tilde{\nu}}{\sigma} \right) \frac{\partial \tilde{\nu}_\alpha}{\partial x_j} \right] + \frac{1}{\sigma} \frac{\partial \tilde{\nu}_\alpha}{\partial x_j} \frac{\partial \tilde{\nu}}{\partial x_j} + 2 \frac{c_{b2}}{\sigma} \frac{\partial}{\partial x_j} \left(\tilde{\nu}_\alpha \frac{\partial \tilde{\nu}}{\partial x_j} \right) \right. \\
& \left. + \tilde{\nu}_\alpha \tilde{\nu} C_{\tilde{\nu}} - (\mathcal{P} - \mathcal{D}) \tilde{\nu}_\alpha + \frac{\partial u_i}{\partial x_j} \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) \frac{\delta \nu_t}{\delta \tilde{\nu}} \right\} \frac{\delta \tilde{\nu}}{\delta b_n} d\Omega - \int_\Omega \frac{\partial u_i}{\partial x_i} \frac{\delta p}{\delta b_n} d\Omega \\
& + \int_\Omega \left[-2 \frac{\partial}{\partial x_j} \left(\Delta^\alpha \frac{\partial \Delta}{\partial x_j} \right) + \tilde{\nu} \tilde{\nu}_\alpha C_\Delta \right] \frac{\delta \Delta}{\delta b_n} d\Omega + \int_\Omega \left(-u_i v_j \frac{\partial v_i}{\partial x_k} - u_j \frac{\partial p}{\partial x_k} - \tau_{ij}^{\alpha} \frac{\partial v_i}{\partial x_k} \right. \\
& \left. + u_i \frac{\partial \tau_{ij}}{\partial x_k} + q \frac{\partial v_j}{\partial x_k} + \Theta_{jk} \right) \frac{\partial}{\partial x_j} \left(\frac{\delta x_k}{\delta b_n} \right) d\Omega + \int_{S_{W_P}} j_{S_{i,k}^g} \frac{\delta x_k}{\delta b_n} n_i dS + \int_{S_{W_P}} j_{S,i} \frac{\delta (n_i dS)}{\delta b_n}
\end{aligned}
\tag{2.13}
$$

where Θ_{jk} is defined as

$$
\begin{aligned}
\Theta_{jk} = & - \tilde{\nu}_\alpha v_j + \tilde{\nu}_\alpha \frac{\partial}{\partial x_j} \left[\left(\frac{\nu + \tilde{\nu}}{\sigma} \right) \frac{\partial \tilde{\nu}}{\partial x_j} \right] - \left(\frac{\nu + \tilde{\nu}}{\sigma} \right) \frac{\partial \tilde{\nu}_\alpha}{\partial x_j} \frac{\partial \tilde{\nu}}{\partial x_k} \\
& - \tilde{\nu}_\alpha \tilde{\nu} \frac{C_Y}{Y} e_{iql} \frac{\partial v_l}{\partial x_q} e_{ij\lambda} \frac{\partial v_\lambda}{\partial x_k} + 2 \tilde{\nu}_\alpha \frac{c_{b2}}{\sigma} \frac{\tilde{\nu}}{\partial x_j} \frac{\partial \tilde{\nu}}{\partial x_k} - 2 \Delta^\alpha \frac{\partial \Delta}{\partial x_j} \frac{\partial \Delta}{\partial x_k}
\end{aligned}
$$

The additional degrees of freedom introduced to Eq. 2.11 in the form of the adjoint variables are used to nullify the dependence of Eq. 2.13 on the variations of the primal variables. Otherwise, the derivatives of the state variables w.r.t. the design variables would have to be computed through Direct Differentiation (DD) [**?**] of the state equations (Eq. 2.1 - 2.4) at a cost equal to N Equivalent Flow Solutions (EFS). An EFS stands for the computational cost of a CFD simulation. The elimination of the expressions multiplying the derivatives of the primal variables results to as many adjoint PDEs as the primal ones per objective function, along with the corresponding boundary conditions and the SD expression. Solving the adjoint PDEs has almost the same cost as the solution of the primal PDEs (1 EFS).

Thus, by using the adjoint method during each optimization cycle, the cost (apart from the primal solution cost) of computing the derivatives of all the objective functions is equal to M EFS, where M is the number of objective functions. An in-depth analysis of the derivation of the adjoint PDEs can be seen in PCOpt's works of [**?**, **?**, **?**]; their final expressions are:

$$R^q = -\frac{\partial u_i}{\partial x_i} = 0 \tag{2.14}$$

$$\begin{aligned} R_i^u &= u_j \frac{\partial v_j}{\partial x_i} - \frac{\partial (v_j u_i)}{\partial x_j} - \frac{\partial \tau_{ij}^{\alpha}}{\partial x_j} + \frac{\partial q}{\partial x_i} + \tilde{\nu_\alpha} \frac{\partial \tilde{\nu}}{\partial x_i} - \frac{\partial}{\partial x_l}\left(\tilde{\nu_\alpha}\tilde{\nu}\frac{C_Y}{Y} e_{mjk}\frac{\partial v_k}{\partial x_j} e_{mli}\right) \\ &= 0 \ , \ i = 1, 2, 3 \end{aligned} \tag{2.15}$$

$$\begin{aligned} R^{\tilde{\nu_\alpha}} &= -\frac{\partial (v_j \tilde{\nu_\alpha})}{\partial x_j} - \frac{\partial}{\partial x_j}\left[\left(\frac{\nu + \tilde{\nu}}{\sigma}\right)\frac{\partial \tilde{\nu_\alpha}}{\partial x_j}\right] + \frac{1}{\sigma}\frac{\partial \tilde{\nu_\alpha}}{\partial x_j}\frac{\partial \tilde{\nu}}{\partial x_j} + 2\frac{c_{b2}}{\sigma}\frac{\partial}{\partial x_j}\left(\tilde{\nu_\alpha}\frac{\partial \tilde{\nu}}{\partial x_j}\right) \\ &+ \tilde{\nu_\alpha}\nu C_{\tilde{\nu}} - (\mathcal{P} - \mathcal{D})\tilde{\nu_\alpha} + \frac{\partial u_i}{\partial x_j}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)\frac{\delta \nu_t}{\delta \tilde{\nu}} = 0 \end{aligned} \tag{2.16}$$

$$R^{\Delta_\alpha} = -2\frac{\partial}{\partial x_j}\left(\Delta^\alpha \frac{\partial \Delta}{\partial x_j}\right) + \tilde{\nu}\tilde{\nu_\alpha} C_\Delta = 0 \tag{2.17}$$

After satisfying the field adjoint equations of Eqs. 2.14-2.17 there are still a few remaining terms in Eq. 2.13 that contain the variations of the state variables along the boundaries S (inlets, outlets, symmetry planes), parameterized walls S_{Wp} and non-parameterized walls S_{Wnp}. Nullifying these expressions gives rise to the adjoint boundary conditions [**?**]. Along the inlet boundaries S_I, S_{Wp} and S_{Wnp}

$$\begin{aligned} u_j n_j = u_{\langle n \rangle} &= -\frac{\partial j_{S_I,i}}{\partial p} n_i - j_{S_i}^p \\ u_{\langle t \rangle}^I &= \frac{\partial j_{S_I,k}}{\partial \tau_{ij}} n_k t_i^I n_j + \frac{\partial j_{S_I,k}}{\partial \tau_{ij}} n_k t_j^I n_i \\ u_{\langle t \rangle}^{II} &= \frac{\partial j_{S_I,k}}{\partial \tau_{ij}} n_k t_i^{II} n_j + \frac{\partial j_{S_I,k}}{\partial \tau_{ij}} n_k t_j^{II} n_i \end{aligned}$$

where t_i^I, t_i^{II} denote the components of the Frenet tangent to the boundary unit vectors, $u_{\langle n \rangle}$ the projection of the velocity vector onto the unit nornal to the boundary and $u_{\langle t \rangle}^I$ and $u_{\langle t \rangle}^{II}$ the tangential velocity components to the boundary. Also, $\partial q / \partial n = 0$ and $\tilde{\nu}_\alpha = 0$. Regarding Δ^α, along S_I, $\Delta^\alpha = 0$ and S_{Wnp} and S_{Wp} a zero Neumann condition is imposed. Along the outlet boundaries S_O, for the normal component of the adjoint velocity the condition $\partial u_{\langle n \rangle} / \partial n = 0$ is imposed and the tangential components are given by

$$v_{\langle n\rangle}u^l_{\langle t\rangle} + (\nu+\nu_t)\left(\frac{\partial u^l_{\langle t\rangle}}{\partial n} + \frac{\partial u_{\langle n\rangle}}{\partial t^l}\right) + \tilde{\nu_\alpha}\tilde{\nu}\frac{C_Y}{Y}e_{jql}\frac{\partial v_l}{\partial x_q}e_{jki}n_k t^l_i + \frac{\partial j_{S_O,k}}{\partial v_i}n_k t^l_i + j_{S_O,i}t^l_i = 0$$

Regarding $\tilde{\nu_\alpha}$,

$$\tilde{\nu_\alpha}v_j n_j + \frac{\partial\tilde{\nu_\alpha}}{\partial n}\left(\frac{\nu+\tilde{\nu}}{\sigma}\right) + j^{\tilde{\nu}}_S = 0$$

and $q = \Delta^\alpha = 0$. On the symmetry planes S_S, $u_{\langle n\rangle} = 0$, $\partial u^l_{\langle t\rangle}/\partial n = 0$, $\Delta^\alpha = 0$ and for q and $\tilde{\nu_\alpha}$, zero Neumann conditions are imposed.

After satisfying the primal and adjoint equations along with the adjoint boundary conditions, the final expression of the SD arises

$$\begin{aligned}
\frac{\delta L}{\delta b_n} = &\int_\Omega\left(-u_i v_j\frac{\partial v_i}{\partial x_k} - u_j\frac{\partial p}{\partial x_k} - \tau^\alpha_{ij}\frac{\partial v_i}{\partial x_k} + u_i\frac{\partial\tau_{ij}}{\partial x_k} + q\frac{\partial v_j}{\partial x_k} + \Theta_{jk}\right)\frac{\partial}{\partial x_j}\left(\frac{\delta x_k}{\delta b_n}\right)d\Omega\\
&+\int_{S_{Wp}} j^g_{S_{i,k}}\frac{\delta x_k}{\delta b_n}n_i dS + \int_{S_{Wp}} j_{S,i}\frac{\delta(n_i dS)}{\delta b_n}\\
&-\int_{S_{Wp}}\left[\frac{\partial j_{S_{Wp,k}}}{\partial\tau_{lm}}n_k t^I_l t^I_m\tau_{ij}\frac{\delta(t^I_i t^I_j)}{\delta b_n}\right]dS - \int_{S_{Wp}}\left[\frac{\partial j_{S_{Wp,k}}}{\partial\tau_{lm}}n_k t^{II}_l t^{II}_m\tau_{ij}\frac{\delta(t^{II}_i t^{II}_j)}{\delta b_n}\right]dS\\
&-\int_{S_{Wp}}\left[\left(u_{\langle n\rangle} + \frac{\partial j_{S_{Wp,k}}}{\partial\partial\tau_{lm}}n_k n_l n_m\right)\tau_{ij}\frac{\delta(n_i n_j)}{\delta b_n}\right]dS\\
&-\int_{S_{Wp}}\left[\left(\frac{\partial j_{S_{Wp,k}}}{\partial\tau_{lm}}n_k(t^{II}_l t^I_m + t^I_l t^{II}_m)\right)\tau_{ij}\frac{\delta(t^{II}_i t^I_j)}{\delta b_n}\right]dS
\end{aligned} \tag{2.18}$$

Due to the presence of Field Integrals on the r.h.s. of Eq. 2.18, this adjoint method is abbreviated as the FI adjoint [**?**]. The first term on the r.h.s. of this equation requires the computation of grid sensitivities at the internal grid nodes. This is conducted after differentiating the equations of the Grid Displacement Model (GDM) w.r.t. b_n. This results in expressions which are similar to the GDM equations and have to be evaluated N times. Thus, the computation of grid sensitivities increases the cost of the FI adjoint.

To overcome the need of computing $\delta x_k/\delta b_n$ in the interior of Ω and to reduce, even further, the cost of computing the adjoint sensitivities, alternatives have been considered. In [**?**, **?**], the proposed method avoids considering internal grid displacement, giving rise to the Surface Integral (Severed SI) adjoint. In PCOpt's pioneering works of [**?**, **?**], a different approach is followed. To consider the internal

grid displacement but avoid at the same time the expensive computation of grid sensitivities, the adjoint to the GDM equations was derived giving rise to the Enhanced Surface Integral (E-SI) adjoint.

The objective functions this thesis is dealing with, are: (a) the total pressure losses between inlet(s) and outlet(s) denoted by J_{P_t} (min.), (b) the drag forces exerted on an object along a direction $\vec{r}$ (that of the far-field velocity) denoted by J_{C_D} (min.) and (c) the mean static pressure rise between inlet(s) and outlet(s) J_{C_p} (max.). These objectives are given by

$$J_{P_t} = -\int_{S_I}\left(p+\frac{1}{2}v_j^2\right)v_i n_i dS - \int_{S_O}\left(p+\frac{1}{2}v_j^2\right)v_i n_i dS \tag{2.19}$$

$$J_{C_D} = 2\frac{\int_{S_W}\left(-\tau_{ij}+p\delta_i^j\right)n_j r_i dS}{A_{ref}U_{ref}^2} \tag{2.20}$$

$$J_{C_p} = \frac{\int_{S_O} pdS}{S_O} - \frac{\int_{S_I} pdS}{S_I} \tag{2.21}$$

where A_{ref} and U_{ref} are the reference area and far field velocity magnitude (both fixed), respectively and δ_i^j is the Kronecker delta. S_O and S_I denote the surface areas of the inlet(s) and outlet(s), respectively.

Chapter 3

CAD-to-Surface Grid

In this thesis, the first step to beginning a CAD-based optimization, is to establish an 'early' link to CAD. This is done by connecting the available CAD parameters to the 3D computational mesh which will be used for the simulations (i.e. the solution to the primal and adjoint equations), which is necessary for both the computation of the sensitivity derivatives and the mesh displacement method that will be employed during the optimization. In what follows, the terms mesh and grid will be used. The former will refer to the 3D computational mesh used for the CFD solution and the latter will refer to a surface grid (generated on CAD surfaces). In case the 3D mesh exists, then it has to be mapped onto the CAD surfaces (patches) which will provide the aforementioned connection. For this reason, a surface grid of the CAD patches will be necessary. In case it does not exist, then it must be generated inside or around the CAD model and its patches. In order to do that, mesh generators use surface grids to define the boundary of the computational domain.

In both cases, there is the requirement for a surface grid and, in this chapter, after a literature survey of surface grid generation methods is presented, a new method is proposed to generate a surface grid on the CAD (BRep) patches. The input to its accompanying software will be a STEP or an IGES file containing the CAD geometry. The output will be an STL file containing the triangular grid.

3.1 Literature Survey of Surface Grid Generation Techniques

High quality surface grid generation is a subject that has been studied quite extensively and various methods have been proposed over the years. The grids can be either structured [**?**, **?**] or unstructured [**?**, **?**, **?**] depending on the requirements of the applications that they are used for and the software that they are used with.

3.1.1 Structured Surface Grids

Structured grid generation methods are mainly used to generate grids on (or around) regular geometries, which are geometries that can be mapped onto unit squares or cubes (i.e. any four-sided region of a 2D grid), and are called topological squares or cubes. Mathematical transformations are, then, used to map an orthonormal grid generated on that unit square on the simulation domain. As expected, when the simulation domain does not have this property, this task becomes highly non-trivial [**?**]. In mainstream structured grid applications [**?**, **?**, **?**, **?**], complex domains should be subdivided into simple blocks which are topological squares. From a algoritmhic perspective, the techniques used for such a division deal with similar issues as unstructured grid generation techniques. The generation of structured grids in such domains can be done either with Algebraic or Partial Differential Equation (PDE)-based Methods.

Algebraic Methods [**?**, **?**, **?**, **?**], use blending techniques to combine curves and this can be done via transfinite interpolations.

Similarly to algebraic methods, PDE-based methods generate a mapping between the parametric domain (ξ, η) and the simulation domain (x, y, z) by solving PDEs. The PDEs are solved on a reference background grid to generate the structured grid. Depending on the type of the PDEs, these methods are usually classified in (a) Elliptic [**?**, **?**, **?**] which use Laplace operators [**?**, **?**] and (b) Hyperbolic Methods [**?**, **?**, **?**].

3.1.2 Unstructured Surface Grids

Unstructured grids [**?**] are ideal for discretizing irregular domains. This, makes them very useful for the discretization of CAD surfaces which are usually arbitrarily trimmed patches. Unstructured grid generation methods can be classified as (a) Advancing Front methods [**?**, **?**, **?**] and (b) Delaunay methods [**?**, **?**] as well as some hybrid approaches [**?**, **?**].

The Advancing Front method generates a grid by progressively adding elements, starting from the boundaries. This results in the propagation of a front standing for the interface between the triangulated and the yet untriangulated subdomains. It allows for generating elements of good quality as it always aims at the generation of equilateral triangles or isosceles which are almost equilateral. The method tends to produce elements of inferior quality at regions where the sides of the front meet.

Another advantage of the Advancing Front method is that it can naturally preserve the boundary curves of a BRep patch and it is highly robust. Its disadvantage is that it requires geometric tests during the computation of every new element to ensure that the front merging is done correctly. These geometric tests can, at times, slow down the process.

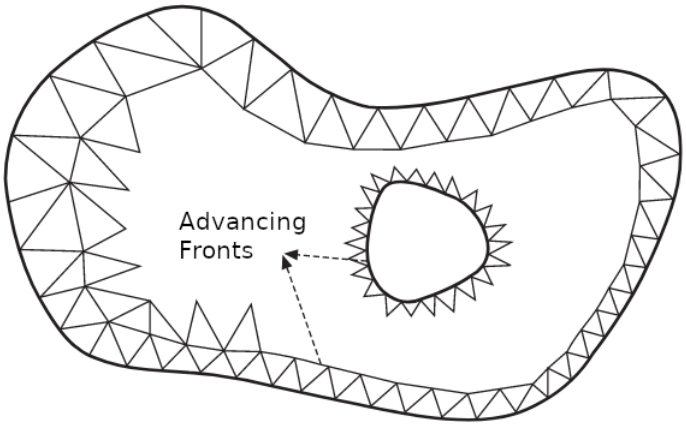


Figure 3.1: A time stamp of the propagation of two Advancing Fronts from two trimming curves – an internal and an external.

Delaunay triangulation methods consist of two main tasks. Initially, there is the so-called grid topography which includes the placement of the grid points. Then, there is the grid topology which includes the computation of a set of triangles which is unique based on the Delaunay criterion [**?**]. The Delaunay criterion states that the circumcircles of all triangles in the resulting grid should not contain any other grid points. The grid topography can be done either entirely in one go or incrementally. In applications in which a grid is generated directly from CAD surfaces [**?**], the grid points that lay on the trimming curves are initially inserted and a first triangulation is computed. Internal nodes are inserted in a second step.

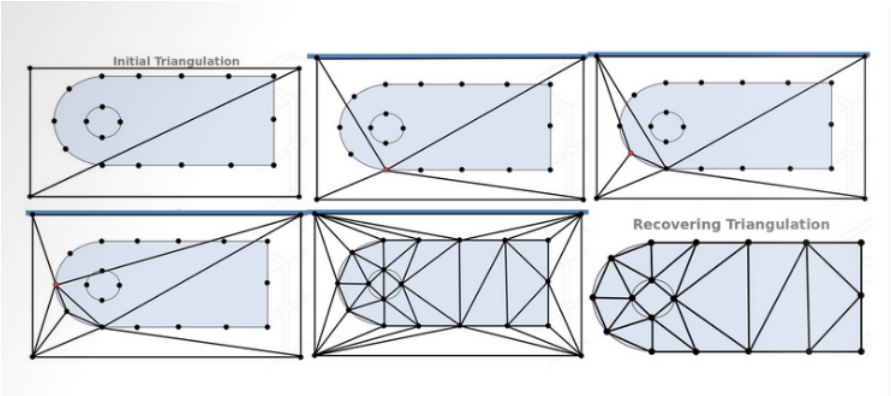


Figure 3.2: Steps of a Delaunay triangulation. Top-Left: All grid points and the domain, on which a grid is generated, are initially enclosed by two large triangles.

Generally, the process of generating a Delaunay triangulation starts off by generating one or two large triangles (as in Fig. 3.2) containing all grid points.

Then, it continues by utilizing a point insertion algorithm to insert all the grid points in the triangulation. Such an algorithm could be, for instance, the Watson-Lawson [**?**] and Bowyer-Watson [**?**] algorithms.

An advantage of Delaunay triangulation is that it is fast, it is mathematically proven that it always converges and is very inexpensive from a computational point of view. However, the quality of the Delaunay triangles is decreased if the topography phase is not done carefully and can sometimes become skinny triangles which affect the final grid quality.

3.2 The Proposed Method

A method to triangulate the surfaces of BRep models, which are contained in standard files, is shown and developed. The method uses elements of both the Advancing Front and the Delaunay algorithms at different stages. The Advancing Front is used as the main algorithm because it produces high quality triangulations and, also, respects the trimming bounds of CAD patches. Delaunay, on the other hand, is used because it is much faster and can, thus, be a reliable background grid generation technique. The aforementioned background grid can be created with a few grid points and, therefore, be a coarse, low quality grid computed on the CAD model, which is utilized for computing size metrics. The method consists of three steps:

- Shape healing is initially performed. For each solid that comprises a CAD model, defects in geometry and topology are identified and fixed.
- The background grid is generated using Delaunay triangulation and a size map is computed on that background grid.
- The Advancing Front Algorithm is used for each surface of the solid models. It makes use of the generated size map to compute each triangle. The outcome is a high quality grid that covers the entirety of a CAD model.

This method also allows for the inclusion of the CAD model into the optimization loop as it allows for a fast adaptation of the grid on the CAD model surfaces. The computational mesh that is generated by using the triangulated surfaces, is differentiated w.r.t. the CAD surface parameters which enables the shape sensitivity computation.

3.2.1 Shape Healing

When a surface grid generation algorithm is performed on a CAD model, certain assumptions about its geometric and topological integrity are made. However, the assumptions do not always hold as CAD models can originate from various

commercial (or not) packages with different rules about geometric tolerance and topological storage. These rules are not always conveyed fully in standard CAD files and this results in defects in geometry and topology. A geometric hole appears if two neighbouring patches do not touch along their boundaries. A topological hole appears if two neighbouring patches are not identified as such in the CAD topological tree. These defects are usually caused when a model is underdefined during its generation from a CAD package. This could either be due to inconcistencies of transfer data or at some rare cases on the designer. Both these defects are quite common and can lead to a bad quality surface grid.

An example of geometric defects can be seen in the Drivaer concept car model (https://www.mw.tum.de/en/aer/research-groups/automotive/drivaer/) which was designed by the Technical University of Munich, Fig. 3.3.

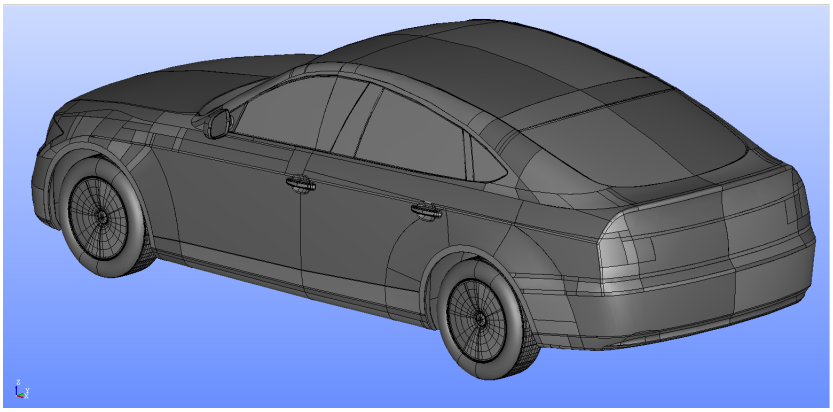


Figure 3.3: The CAD geometry of the Drivaer concept car.

Looking closer at certain areas on the car's surface, the geometric defect becomes apparent (Fig. 3.4).

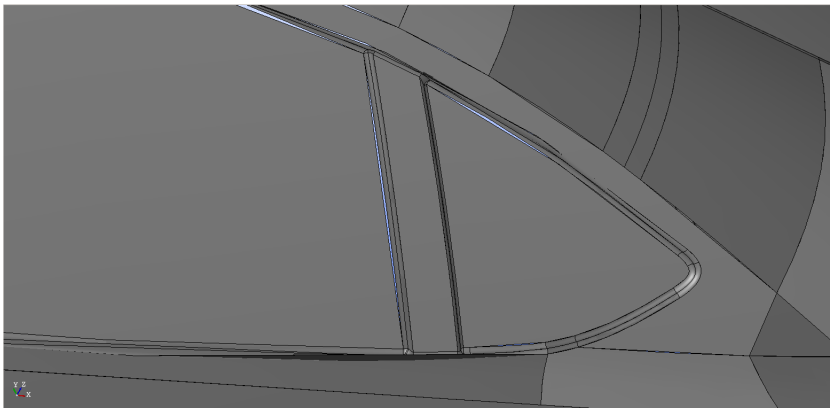


Figure 3.4: Perspective of the Drivaer's back left window. Holes between neighbouring CAD patches can be seen.

An example of a topological defect can be seen in Fig. 3.5. Topological defects may not be visible in a CAD viewer but they can seriously affect the quality of the resulting triangulation. For this reason, in the example of Fig. 3.5, the triangulations of the same two surfaces are shown, firstly topologically impaired and, then, repaired.

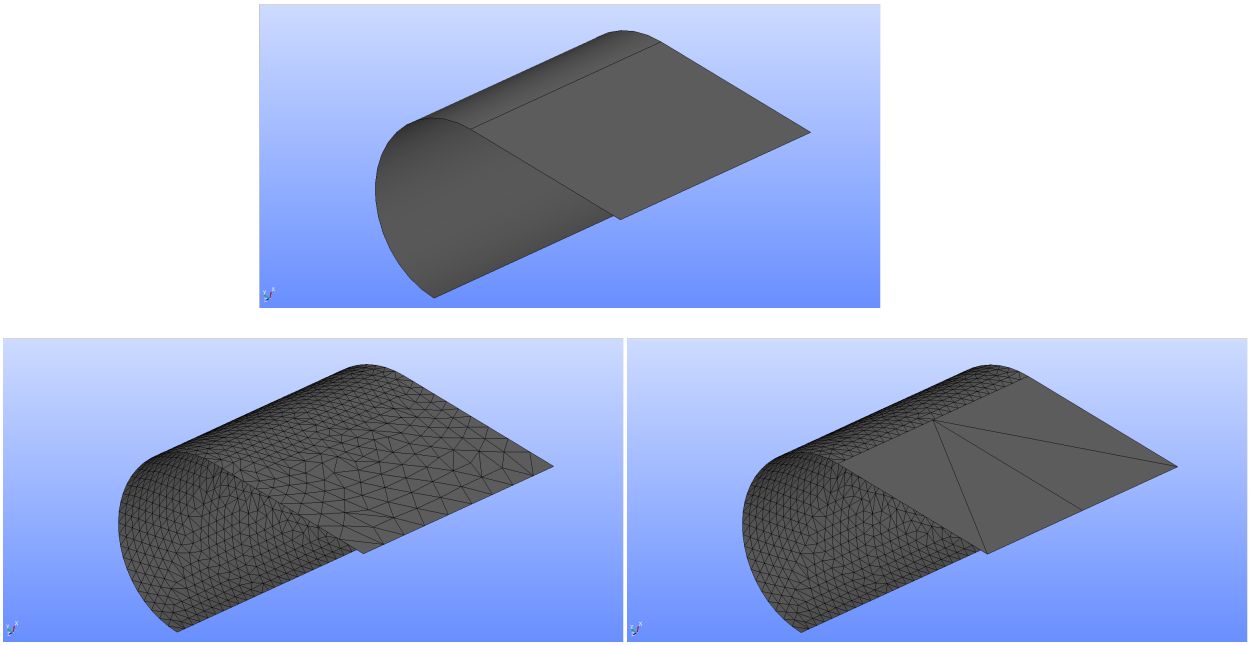


Figure 3.5: Top: Two CAD patches (a semi-cylindrical and a planar-rectangular) connected along their trimming edges. Bottom-Left: Resulting triangulation if the two patches are also connected topologically. Bottom-Right: The resulting triangulation in the opposite case.

When two CAD patches are geometrically, though not topologically connected, abrupt size jumps occur in the resulting triangulation. The reason behind this is that the triangulation of each patch is performed based exclusively on geometric criteria, in order to create the highest quality grid (quality elements that capture the details of the geometry as accurately as possible) with as few triangles as necessary. To avoid the existence of defects in the triangulation related to (any or both of) the above mentioned issues, a shape healing algorithm is necessary.

3.2.1.1 Octree Search for Topological Relations

An Octree search algorithm is developed to accommodate search routines useful for CAD models. Initially, when the Octree search starts, the entire CAD model is scanned and existing topological relations are identified. Edges that belong solely on one face (hanging) are checked and geometric tests are performed with other hanging edges in their vicinity. For example, all the edges of the Drivaer CAD

model shown in Fig. 3.6 are hanging, meaning that they belong to only one face. The tests which are mentioned compute the minimum and maximum distances between two hanging edges. Depending on the results of these tests, duplicate edges can be found which are removed and the topology of the model is updated.

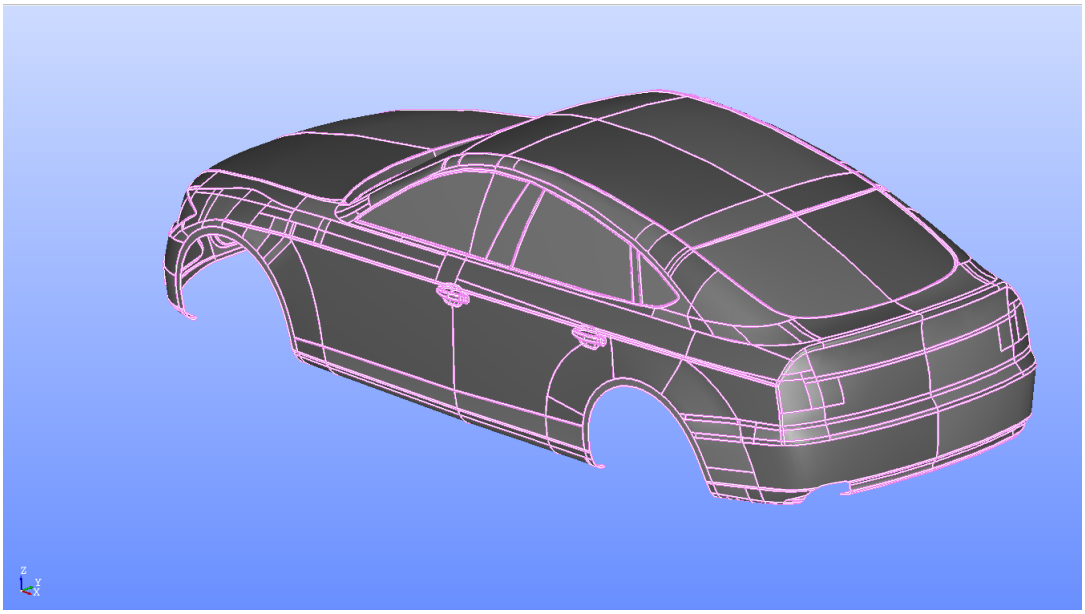


Figure 3.6: Part of the Drivaer CAD model shown in Fig. 3.3. Hanging edges are colored in pink.

The first step for this part is to generate an Octree [**?**] in the space around the CAD model. An Octree is a tree data structure in which various geometric data can be stored so that search routines become faster and less intensive from a computational point of view. The data which is classified in the Octree's nodes are the CAD trimming curves (edges). For this reason, the root octant is identical to the bounding box of the entire CAD model. A CAD edge is inserted in the Octree via its own bounding box, by recursively subdividing each branch, as long as the edge does not fit in one of the respective offspring octants. All hanging edges are classified in the Octree and tested against each other for being duplicates. The performed geometric tests check whether two edges are identical so that the extra one can be removed from the Octree and, eventually, from the CAD model. The faces that the two edges belong to are then defined as topological neighbours.

3.2.1.2 The Geometric Tests for Hanging Edges

As mentioned before, the hanging edges that are classified into the octree are tested for possible matching with others. The tests are explained below.

For each of the hanging edges of a CAD model that are classified into an Octree, a search radius equal to $0.8d$ is defined where d is the length of the diagonal of the

bounding box of the hanging edge under concern. The Octree returns the hanging edges that are within this radius and they are then sorted on a closest-first basis. The tested edge is then compared with the edges in the sorted list until a match is found. If not, the edge is considered to be the border of the shell.

Let us assume that the parametric expression of the curve of the tested edge is $\vec{h}(t)$ and the one of an edge off the sorted list is $\vec{c}(u)$ (where t and u are parametric coordinates). For $\vec{h}(t)$ and $\vec{c}(u)$ to match, multiple points on them must be identical. Therefore, on $\vec{h}(t)$, points are placed based on curvature and each of them is then checked for proximity with $\vec{c}(u)$. The curvature at each parametric coordinate t is computed by [**?**]

$$\kappa(t) = \frac{||\vec{h}'(t) \times \vec{h}''(t)||}{||\vec{h}'(t)||^3} \tag{3.1}$$

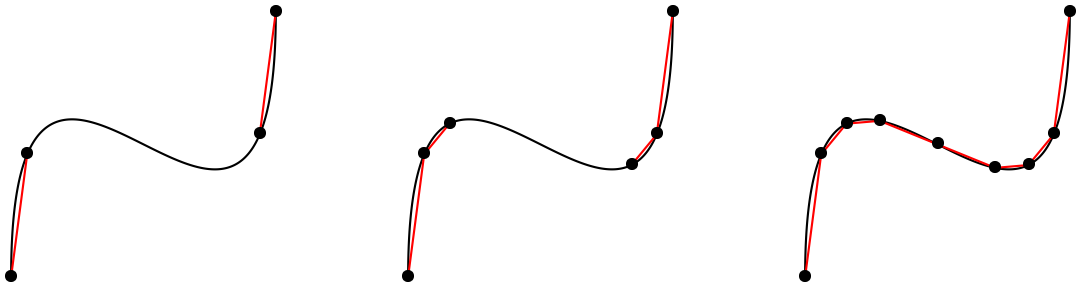


Figure 3.7: Stages of the discretization process of a 3D curve starting from left to right. Starting from both the first and the last vertex of $\vec{h}(t)$, new points are placed along its length.

The discretization of $\vec{h}(t)$ is done as follows: Assuming that $t \in [t_i, t_f]$ and that n vertices are produced, then the first vertex $\vec{P}_1 = \vec{h}(t_i)$ and the last vertex $\vec{P}_n = \vec{h}(t_f)$. Then, two more vertices are generated on $\vec{h}(t)$ such that $||\vec{P}_2 - \vec{h}(t_i)|| = s(t_i)$ and $||\vec{P}_{n-1} - \vec{h}(t_f)|| = s(t_f)$. $s(t) = min(\theta/\kappa(t), L)$ is the ideal distance between a point on the curve at t and the next to be placed where L is the length of $\vec{h}(t)$ and θ is a real, user-defined constant that controls the point density along the curve. The process is repeated for the new vertices until all of $\vec{h}(t)$ is discretized. 0 All $\vec{P}_i (i \in [1, n])$ that result from the discretization are then projected onto $\vec{c}(u)$. This projection is performed by solving

$$\frac{d\vec{c}(u)}{du} \cdot (\vec{c}(u) - \vec{P}_i) = 0 \tag{3.2}$$

for u. Eq. 3.2 is solved (using the Newton-Raphson [**?**] method) instead of a simple distance equation because it behaves much better when searching for solutions inside the parametric domain of u. If multiple solutions exist, the final solution

u^* is the one minimizing $||\vec{c}(u) - \vec{P}_i||$. The above procedure is repeated for every point $\vec{P}_i$ and if $||\vec{c}(u) - \vec{P}_i|| \leq \epsilon$ for all i, then $\vec{h}(t)$ and $\vec{c}(u)$ are considered identical. The choice of ϵ varies depending on the CAD model. Commonly, a CAD package defines a distance tolerance ϵ relative to the size of the model when generating its geometry.

3.2.1.3 Sewing Neighbouring Patches

The above mentioned process functions properly as a means to resolve topological defects. To solve the geometric defects (patches not touching at the interface between them), a slightly different approach must be followed. The same Octree search and geometric tests are repeated as above. The first difference is that the tolerance ϵ is set to a higher value so as to identify topological neighbours even with the existence of geometric gaps. Secondly, for the two surfaces around each geometric gap, a stitching algorithm is performed that requires that they touch at a number of points along their trimming edges.

This stitching is done by moving one of the two surfaces to touch the other by solving a minimization problem that takes point and normal vector constraints into account. The minimization problem is the Thin Plate Energy [**?**] algorithm Its physical analogy is that, when deforming a thin sheet of metal (stretching, bending etc), it minimizes its internal energy and, thus, it keeps its general properties due to its rigidity. In other words, if the Thin Plate Energy of a surface during the stitching process remains roughly the same, then its properties remain the same as well. This is achieved by minimizing the Thin Plate Energy of the surface perturbation while satisfying multiple point constraints and normal vector constraints.

For any parametric surface $\vec{S}(u, v)$, the Thin Plate Energy is defined as

$$E(\vec{S}) = \iint_D (\vec{S}_{uu} \cdot \vec{S}_{uu} + 2\vec{S}_{uv} \cdot \vec{S}_{uv} + \vec{S}_{vv} \cdot \vec{S}_{vv}) du dv \tag{3.3}$$

where D is the parametric domain of the surface. Indices u, v indicate partial differentiation of $\vec{S}$ along a parametric direction. Eq. 3.3 is a linear approximation of the exact Plate Energy which is equal to the integral of the squares of the principal curvatures on $\vec{S}$.

In order to modify and stitch a surface $\vec{S}$ to a target curve-on-surface and make it have a certain normal orientation, $E(\delta\vec{S})$ must be minimized while imposing a number nc of point and normal vector constraints.

Assuming that $\vec{S}$ is a NURBS surface (as is the case for BRep), its expression is given by (Eq. 1.3)

$$\vec{S}(u, v) = \sum_{i=1}^{n} \sum_{j=1}^{m} R_{i,j}(u, v) \vec{P}_{i,j} \tag{3.4}$$

and its partial derivatives w.r.t. the parametric coordinates are given by

$$\vec{S}_{uu} = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{\partial^2 R_{i,j}(u,v)}{\partial u^2}\vec{P}_{i,j}$$
$$\vec{S}_{vv} = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{\partial^2 R_{i,j}(u,v)}{\partial v^2}\vec{P}_{i,j}$$
$$\vec{S}_{uv} = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{\partial^2 R_{i,j}(u,v)}{\partial u \partial v}\vec{P}_{i,j}$$

Any NURBS surface pertubation that produces a surface $\vec{S_T}(u,v)$ from a source surface $\vec{S}(u,v)$ can be expressed as

$$\delta\vec{S}(u,v) = \vec{S_T}(u,v) - \vec{S}(u,v) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}(u,v)\delta\vec{P}_{i,j} \tag{3.5}$$

A point constraint is imposed at a certain parametric pair (u,v), by demanding that the target surface touches a target point $\vec{P}_t$. This creates an equality constraint

$$\vec{c} = \delta\vec{S} + \vec{S} - \vec{P}_t = 0 \tag{3.6}$$

Another pair of equality constraints that are imposed are normal vector constraints. This means that $\vec{S_T}$ must have a specified normal vector $\vec{n}$ at a parametric point (u,v). This can be imposed by satisfying two equations:

$$t^1 = (\vec{S}_u + \delta\vec{S}_u)\cdot\vec{n} = 0$$
$$t^2 = (\vec{S}_v + \delta\vec{S}_v)\cdot\vec{n} = 0 \tag{3.7}$$

Thus, the following constrained minimization problem is generated

$$\begin{aligned} Min\ E(\delta\vec{S}) \\ s.t.\ ||\vec{c}_i|| = 0 \\ and\ t_i^1 = 0\ ,\ t_i^2 = 0\ ,\ i \in [1, nc] \end{aligned} \tag{3.8}$$

where nc is the total number of constraints. The order of nc depends on the

complexity of the curves and the surfaces that are stitched. Further analysis of $E(\delta\vec{S})$ leads to the following expression

$$E(\delta\vec{S}) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{g=1}^{n}\sum_{h=1}^{m}\delta\vec{P}_{i,j}\cdot\delta\vec{P}_{g,h}(L_{ijgh}+2M_{ijgh}+N_{ijgh}) \tag{3.9}$$

where

$$\begin{aligned}
L_{ijgh} &= \iint_D \frac{\partial^2 R_{i,j}(u,v)}{\partial u^2}\frac{\partial^2 R_{g,h}(u,v)}{\partial u^2}dudv \\
M_{ijgh} &= \iint_D \frac{\partial^2 R_{i,j}(u,v)}{\partial u\partial v}\frac{\partial^2 R_{g,h}(u,v)}{\partial u\partial v}dudv \\
N_{ijgh} &= \iint_D \frac{\partial^2 R_{i,j}(u,v)}{\partial v^2}\frac{\partial^2 R_{g,h}(u,v)}{\partial v^2}dudv
\end{aligned}$$

Ultimately, the Thin Plate Energy problem is solved by using the Augmented Lagrangian Method [**?**] with

$$L = E(\delta\vec{S}) - \sum_{i=1}^{nc}\lambda_i||\vec{c}_i|| - \sum_{i=1}^{nc}\kappa_i^1 t_i^1 - \sum_{i=1}^{nc}\kappa_i^2 t_i^2 + \frac{\mu}{2}\sum_{i=1}^{nc}(||\vec{c}_i||^2 + {t_i^1}^2 + {t_i^2}^2) \tag{3.10}$$

where λ_i, κ_i^1, κ_i^2 are the Lagrange multipliers and μ is the penalty parameter.

The choice of the Energy quantity is made as it portrays the curvature properties of the parametric surface. Ideally, the properties of the source and the target surfaces should be almost identical and this is achieved by minimizing the Energy of the perturbation of the surface. The result is a less warped perturbed surface.

Two examples are used to demonstrate this (Figs. 3.8, 3.9). In the first example, the sewing algorithm is used on two planar rectangular surfaces that are placed apart vertically to each other. In the second (and more challenging) example two highly curved, trimmed surfaces are placed apart. In both examples, the Thin Plate Energy minimization resulted in a perfect stitching of the patches.

Applying both the topological and sewing algorithms to the CAD model of Fig. 3.6 results in a perfectly defined CAD model ready to undergo triangulation (Fig. 3.10).

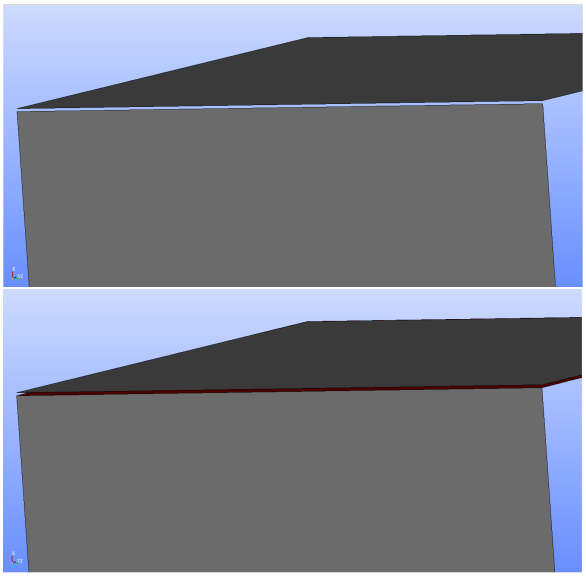


Figure 3.8: Two rectangular surfaces with a geometric gap. Top: Initial positions. Bottom: The resulting position where both the initial surfaces are visible (grey) as well as the final displaced surface (red region covering the hole).

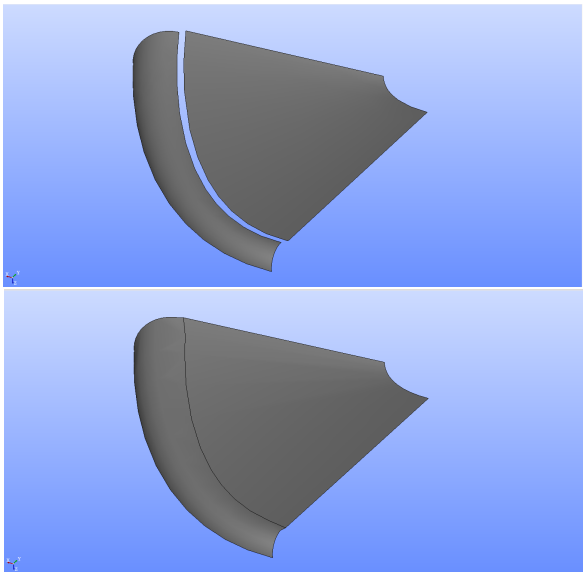


Figure 3.9: Two trimmed curved patches. Top: Initial position. Bottom: Final position.

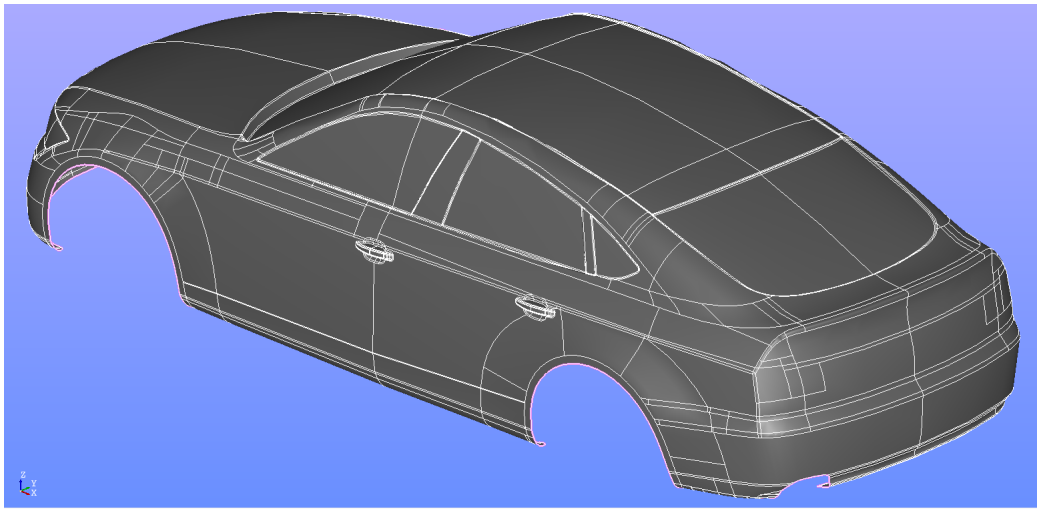


Figure 3.10: The Drivaer model of Fig. 3.6 after the shape healing process ended. The only hanging edges remaining (in pink) are located at the boundary of the shell.

3.2.2 The Background Grid and the Size Map

The background grid is required to store the size metrics at various points on the CAD model. Its generation should be fast and easy and the grid itself should be as coarse as the geometry allows. Quality is not a prerequisite here and the simple requirement is to capture the geometry with as few triangles as possible. As mentioned already, Delaunay triangulation [**?**] is used.

3.2.2.1 The Delaunay Algorithm for the Background Grid

The Delaunay algorithm always converges for a set point cloud in 2D, so the triangulation is performed in each face's parametric space. Initially, the edges of the entire CAD model are discretized in a manner similar to the one shown in section 3.2.1. Then, Delaunay triangulation is performed in the 2D parametric domains of each face by using features of both Lawson [**?**] and Watson [**?**] processes. The Watson and Lawson synergy is explained in Appendix A. During the generation of the background grid, the size of its elements is computed based on the deflection from the actual geometry. A deflection parameter that imposes a maximum allowed distance between the centres of the edges of the resulting polygons and the actual edge curve is defined. The same parameter imposes a maximum allowed distance from the barycenters of the resulting triangles to the CAD surface.

The points that are generated while discretizing the edges of a CAD surface, have parametric coordinates on the surface as well. For instance, in Figs. 3.11 and 3.12, a surface with discretized edges and the parametric coordinates of the

generated points can be seen. Then, discretization is done based on curvature and, thus, edges that are straight lines are described by just two points.

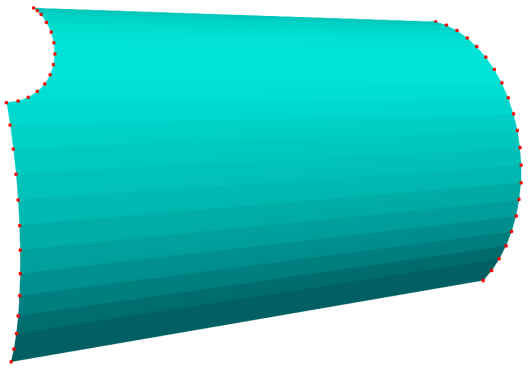


Figure 3.11: A parametric CAD surface (light blue) and the discrete points (red) that are generated on its edges.

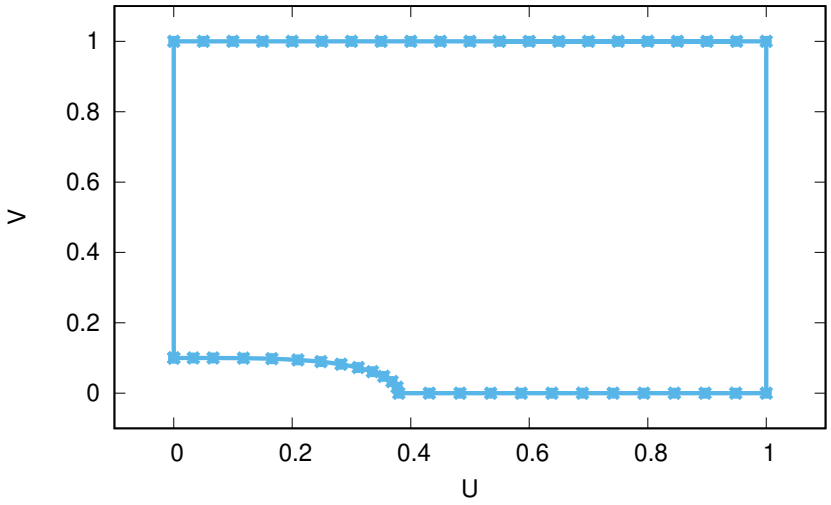


Figure 3.12: The u, v coordinates of the points shown if Fig. 3.11.

Starting from a 2D polygon as in Fig. 3.12, the first Delaunay triangles are computed by introducing one point at a time. The algorithm starts by creating a supertriangle that encloses the entire point cloud. When a new point P is inserted into the triangulation, an existing triangle that encloses P is identified and three new triangles are created by connecting P to its vertices. During this step, the original enclosing triangle is deleted, thus making the net gain of triangles equal to two. After the new point has been inserted, the triangulation is updated to Delaunay by using the Lawson [**?**] algorithm. During this process, all triangles

that are adjacent to edges opposite P are placed in a 'last in, first out' list. Then, a triangle is removed from the list and a check is made to determine if P belongs to its circumcircle and if this is the case, then this triangle and its adjacent, form a quadrilateral with the diagonal placed in the wrong direction. A swap is, therefore, made and this creates two new triangles that replace the two old ones. The two new triangles are placed in the list and the whole process is repeated until the list is empty. Once this process is finished, the initial vertices of the supertriangle are removed along with any triangles that are associated with them. Furthermore, triangles that are placed out of the closed domain defined by the trimming curves, are removed. This results in a Delaunay triangulation that is constrained to have matching boundaries with the discretized trimming edges. The computed triangles are, then, transformed to the 3D space by connecting the relevant 3D points (Fig. 3.13).

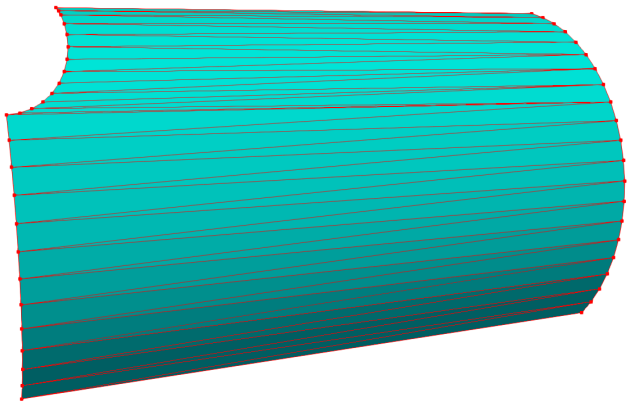


Figure 3.13: A parametric CAD surface (light blue) and the triangulation (red) that is the result of the Constrained Delaunay Method.

For all the created triangles, a check is made to determine their deflection from the actual CAD surface. For a triangle, this deflection is computed as the distance from its barycenter to its projection on the surface. If this deflection is larger than a threshold value, a new point P is inserted at the barycenter of the 2D triangle this corresponds to. When no more triangles need to be created, the algorithm terminates.

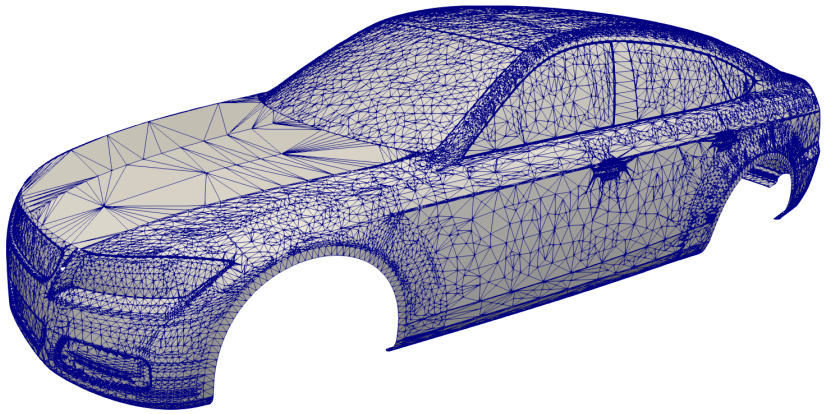


Figure 3.14: The Delaunay Triangulation of the CAD model of the Drivaer car shown in Fig. 3.10.

This algorithm avoids heavy computations and searches and is, thus, very fast. Furthermore, assuming that the shape healing algorithm described in section 3.2.1 has been completed successfuly, then the resulting triangulation of the CAD model will be watertight and with as few triangles as possible. This can lead to a smooth size map that will store the desired sizes at the vertices of the background grid.

During the execution of the Advancing Front grid generation algorithm, the size information of background grid triangles will have to be accessed multiple times. The reason behind this is that when generating a grid on a CAD face or an edge on that face, the algorithm asks for the required size at some point with face parameters (u, v). Therefore, two things are required: (a) a method to quickly identify a few background grid triangles in the vicinity of the given (u, v) and (b) a method to identify in which triangle the point with these coordinates lie. The first point is easily solved with a Quadtree structure. The Quadtree is identical to the Octree shown in section 3.2.1, with the only difference being that it is employed in 2D. The second point is solved by computing the barycentric coordinates of the parametric pair (u, v) for each triangle that is returned by the Quadtree until the correct triangle is pinpointed. Assuming a triangle with vertices with parametric coordinates (u_A, v_A), (u_B, v_B) and (u_C, v_C), the barycentric coordinates of a point on the plane in which the triangle lies, with coordinates (u, v) are computed by

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} u_A - u_C & u_B - u_C \\ v_A - v_C & v_B - v_C \end{bmatrix}^{-1} \cdot \begin{bmatrix} u - u_C \\ v - v_C \end{bmatrix}$$
$$\gamma = 1 - \alpha - \beta \tag{3.11}$$

If $0 \leq (\alpha, \beta, \gamma) \leq 1$, then and only then does (u, v) belong to the triangle.

When the correct triangle is pinpointed, the requested size value at (u, v) is interpolated as $s_{u,v} = \alpha s_A + \beta s_B + \gamma s_C$ where the indexed s variables denote the size values at the three vertices.

3.2.2.2 The Size Map

When generating a grid with triangular elements on CAD surfaces, it is very important to have a pre-computed size map on the geometry. Computing it beforehand allows for easier optimal size computation as well as gradation control. The sizes are computed on the vertices of the background grid and the gradation is controlled via the connectivity. In this method, two dimensionless input parameters control the above mentioned outputs: (a) a curvature-based size parameter d and (b) a triangle growth ratio g_R. The choice of dimensionless parameters is made so that the process does not depend on the units of measurement of the model, but rather on the geometry itself.

As mentioned before, the parameter d is used to generate an optimal grid triangle size based on the model's local curvature. At each vertex of the background grid, which belongs to surface $\vec{\sigma}$ at parameters (u, v), the size $s_1(u, v)$ is computed by

$$s_1(u, v) = K(u, v)d \tag{3.12}$$

where

$$K(u, v) = \frac{1}{max(|\kappa_1(u, v)|, |\kappa_2(u, v)|)} \tag{3.13}$$

where $\kappa_1(u, v)$, $\kappa_2(u, v)$ are the two principal curvatures [**?**] of $\vec{\sigma}(u, v)$. To compute the principal curvatures, the two fundamental form coefficients [**?**] of the surface must firstly be computed at (u, v):

$$\begin{aligned} \mathcal{F}_1 &= \begin{bmatrix} \vec{\sigma}_u \cdot \vec{\sigma}_u & \vec{\sigma}_u \cdot \vec{\sigma}_v \\ \vec{\sigma}_v \cdot \vec{\sigma}_u & \vec{\sigma}_v \cdot \vec{\sigma}_v \end{bmatrix} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} \\ \mathcal{F}_2 &= \begin{bmatrix} \vec{\sigma}_{uu} \cdot \vec{n} & \vec{\sigma}_{uv} \cdot \vec{n} \\ \vec{\sigma}_{vu} \cdot \vec{n} & \vec{\sigma}_{vv} \cdot \vec{n} \end{bmatrix} = \begin{bmatrix} L & M \\ M & N \end{bmatrix} \end{aligned} \tag{3.14}$$

where $\vec{n} = \frac{\vec{\sigma}_u \times \vec{\sigma}_v}{||\vec{\sigma}_u \times \vec{\sigma}_v||}$ is the local unit normal vector.

The two principal curvatures of a surface are computed after solving $det(\mathcal{F}_2 - \kappa \mathcal{F}_1) = 0$ for κ. In case a surface is locally planar ($\kappa_1 = \kappa_2 = 0$), K is set equal to the length of the diagonal of the surface bounding box. Such a surface must be covered with as few triangles as possible but without creating abrupt size changes and this is something that the size gradation algorithm handles (see

further below).

If a vertex lies on one of the trimming curves of the surface, then it probably lies also on a second surface $\vec{\sigma}_2(\xi, \eta)$. Then, three size values can be computed: (a) $s_1(u, v)$ based on the curvature of the first surface, (b) $s_2(\xi, \eta)$ based on the curvature of the second surface and (c) $s_3(t)$ based on the curvature of the trimming curve. If $\vec{h}(t)$ is the expression of the trimming curve, then its curvature $\kappa(t)$ is computed using Eq. 3.1 at parameter t at which $\vec{\sigma_1}(u, v) = \vec{\sigma}_2(\xi, \eta) = \vec{h}(t)$. Then

$$s_3(t) = min\left(\frac{1}{|\kappa(t)|}, L\right) \tag{3.15}$$

where L is the length of $\vec{h}(t)$ given by

$$L = \int_0^1 \sqrt{\left(\frac{dh_x(t)}{dt}\right)^2 + \left(\frac{dh_y(t)}{dt}\right)^2 + \left(\frac{dh_z(t)}{dt}\right)^2} dt \tag{3.16}$$

The introduction of L in Eq. 3.15 is done in order to capture the correct size of potentially small trimming edges that are straight or almost straight. In order to correctly capture the geometry, the smaller size requirements must be respected. Therefore, the final size s is given by

$$s = \begin{cases} s_1(u, v) & \text{, if a vertex lies simply on a surface} \\ min(s_1(u, v), s_2(\xi, \eta), s_3(t)) & \text{, if a vertex lies on a trimming edge} \end{cases} \tag{3.17}$$

After the value of s has been computed at all vertices of the background grid, a gradation algorithm must be performed to smoothen the size map on the model. It is not uncommnon that CAD patches do not maintain C_2 uniformity inside their parametric domain. Furthermore, neighbouring patches will certainly be C_0 continuous but C_1 is not guaranteed. These facts can lead to abrupt size changes on the CAD model which can in turn create convergence and quality problems at the final triangulation. In this work, the bounded H-Variation control, proposed by [**?**], is used as a gradation technique. The H-Variation between the sizes s_P, s_Q which are defined at the connected vertices $\vec{P}$ and $\vec{Q}$ of the background grid, is given by

$$v(\vec{PQ}) = \frac{s_Q - s_P}{||\vec{PQ}||} \tag{3.18}$$

which is the gradient of s on the surface of the CAD model. The input parameter g_R is used to define an upper bound to the H-Variation of all background grid

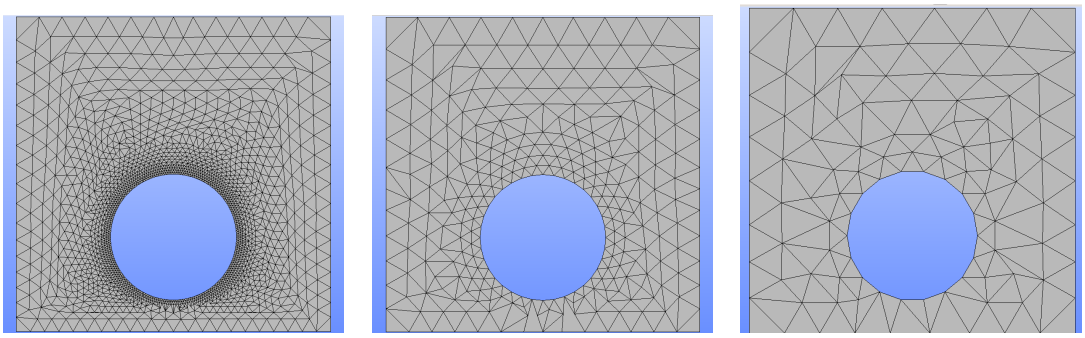
edges and if this upper bound is not met, a correction is performed. For an edge $\vec{PQ}$, the following check is made: If $|v(\vec{PQ})| \leq g_R$, then the sizes are not corrected. In case $|v(\vec{PQ})| > g_R$, then the larger of the two sizes is reduced by setting:

$$\begin{aligned} s_P &= s_Q + g_R \cdot ||\vec{PQ}|| \text{ , if } s_P > s_Q \\ s_Q &= s_P + g_R \cdot ||\vec{PQ}|| \text{ , if } s_P < s_Q \end{aligned} \tag{3.19}$$

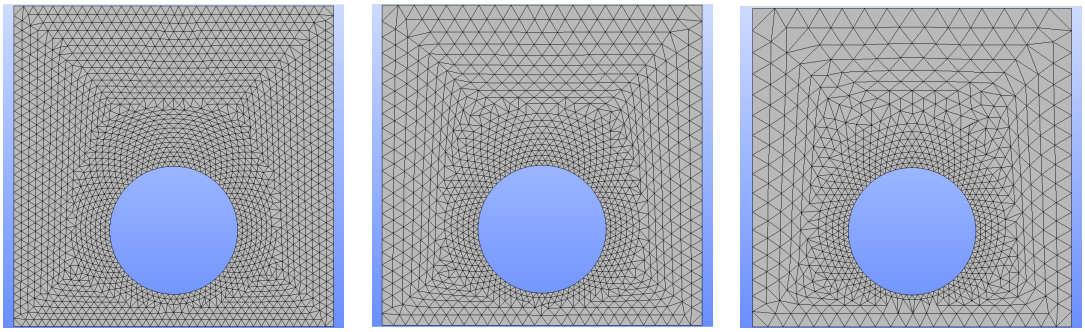
The above check is made for all the edges of the background grid. When a correction is performed to a vertex, all edges that are adjacent to that vertex are re-checked. When no more corrections are made, the algorithm terminates.

With the size computation and gradation algorithms finished, a uniform map is available for the main grid generation algorithm to use. Relations with neighbouring surfaces and local size requirements are taken into account at this point and, therefore, the grid generation can be performed at each edge and each face independently while ensuring the smoothness of the final triangulation.

An example is shown in Fig. 3.15 to demonstrate the effect of the two input parameters on the size map (and the final trianglation). Initially, the growth ratio is set to $g_R = 0.1$ and various values of d are applied. The curvature on the straight outer lines is zero and, therefore, there is no geometric requirement for a triangle size smaller than their length. However, the discretization of the inner circular curve must become finer as d becomes smaller and the gradation of the size map leads to a fine discretization of the straight curves. Then, the value of the deflection is set to $d = 5^o$ and various values for g_R are applied. The discretization of the inner circular curve is identical in all cases (as expected). The value of g_R which controls the gradation leads to a coarser and coarser discretization away from the inner curve as its values gets higher.



(a) $d = 2^o$, $g_R = 0.1$ **(b)** $d = 10^o$, $g_R = 0.1$ **(c)** $d = 20^o$, $g_R = 0.1$



(d) $d = 5^o$, $g_R = 0.02$ **(e)** $d = 5^o$, $g_R = 0.05$ **(f)** $d = 5^o$, $g_R = 0.08$

Figure 3.15: The final triangulation of the same CAD surface, with 6 different sets of input parameters.

Another example is demonstrated in Fig. 3.16 where an axial stator blade and its triangulations are shown. Six different surface grids are generated with six different pairs of input parameters. The effect of the parameters can be seen on the leading edge and the suction side. Surfaces near both the leading and the trailing edge are semi cylinders, which means that they have considerably higher curvature than the pressure and suction sides. Therefore, one can, firstly, see the effect of d on the grid of the leading edge's surface and, secondly, the effect of g_R on the grid, moving farther from high curvature areas.

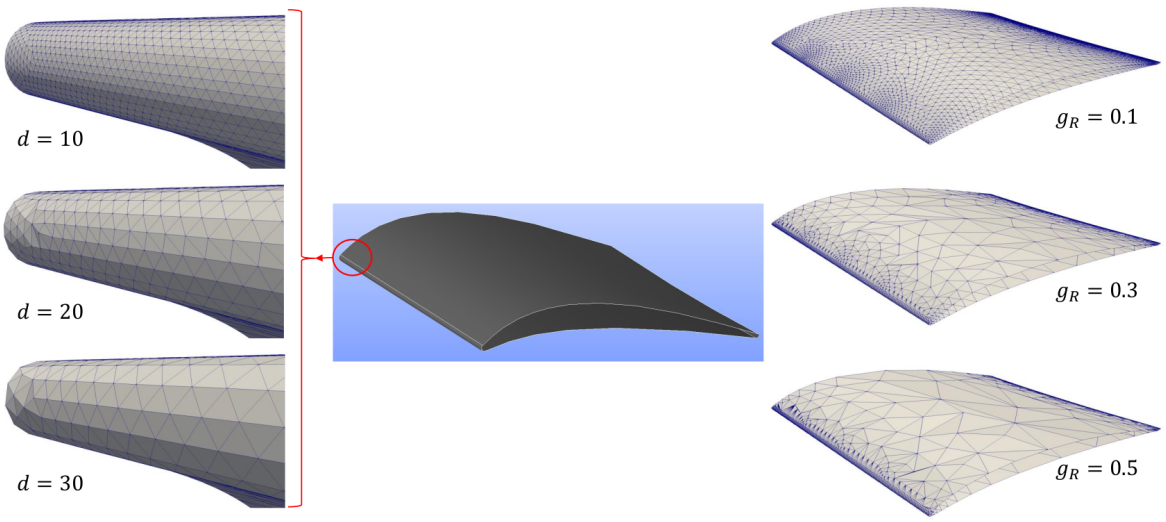


Figure 3.16: Six different triangulations of an axial stator blade. Center: The CAD model of the blade. Left: Focus on the leading edge for three different values of d. As d increases, the grid becomes coarser. Right: The suction side of the blade for three different values of g_R. As g_R increases, the gradation of the grid increases.

3.2.3 The Advancing Front Algorithm

The grid generation algorithm initiates by performing the discretization of every edge of the CAD model. An edge will belong to a primary and/or a secondary surface. On both surfaces its 3D expression $\vec{h}(t)$ will be connected one-to-one with a parametric curve $\vec{h}_p(t)$ such that

$$\vec{h}_p(t) = \begin{pmatrix} u(t) \\ v(t) \end{pmatrix}, \text{ with } \vec{h}(t) = \sigma(u(t), v(t)) \tag{3.20}$$

While the discretization of the edge takes place, the required size value at t is returned by the background grid on the primary face at $\vec{h}_p(t)$. Using these sizes, the process shown in Fig. 3.7 is performed. When the discretization of an edge is finished, the parametric coordinates on both primary and secondary surfaces are stored in different lists in order to be accessed later during the triangulation.

When the discretization of the edges is complete, the grid generation at each individual face begins by building the initial advancing front. Buidling the initial front includes sorting and orientating the grid points placed during edge discretization. Before tackling the initial front, it is important to introduce two data structures: (a) the Front Point and (b) the Front Edge. The Front Point is a structure that holds information about the parametric coordinates of the 3D point it corresponds to and, also, about the Front Edges that touch it. The Front Edge

holds information regarding its first and last Front Points and, also, the next and previous Front Edges. The notation of next and previous edges comes in handy for defining the front topology. The Advancing Front data structure holds lists of all Front Points and Front Edges as well as a Quadtree for fast querying.

Firstly, the list of the Front Points is assembled by retrieving the (u, v) parametric pairs of all the discretized CAD edges that lie on the surface. The parametric pairs are ordered to form a counterclockwise loop of points if they belong to an external wire of edges and a clockwise loop of points if they belong to an internal wire of edges (Fig. 3.17). With this ordering, a new triangle will always be placed such that its normal vector has an outward orientation w.r.t. the CAD solid. For every pair of consecutive points that creates a Front Edge, the topology of the both Front Edges and Front Points is updated.

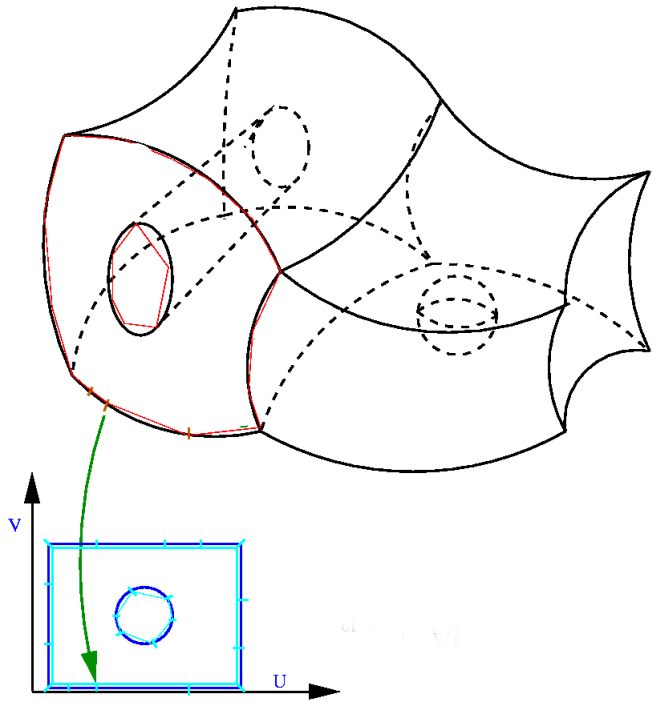


Figure 3.17: Discretization of the boundary of a CAD model's face that creates an initial front. Discrete points are visible in both 3D an 2D parametric spaces (https://www.ljll.math.upmc.fr/perronnet/mit/mit.html).

The triangulation then initiates by selecting a Front Edge from the list and calculating a so-called optimal point for the generation of a new triangle. If this was a simple 2D triangulation, the generated triangle would be isosceles with the selected Front Edge as its basis and with a height equal to the value returned from the size map. However, this is not the case as an isosceles triangle in the parametric space is not necessarily transformed to an isosceles triangle in the 3D space. Furthermore, the size returned from the size map, which corresponds

to the height of a triangle, must be adapted to the parametric domain. Let us assume a Front Edge between Front Points $\vec{A}$, $\vec{B}$, their parametric midpoint $\vec{M}$ (with $u_M = (u_A + u_B)/2$, $v_M = (v_A + v_B)/2$) and the new created point $\vec{P}$. Two values can be computed for the required size at A (s_A) and B (s_B). The optimal size of the element s is computed as $s = (s_A + s_B)/2$. The condition to be met is $s = s_{\vec{MP}}$. If $E(u,v)$, $F(u,v)$, $G(u,v)$ are the components of the first fundamental form of the face at (u,v), then

$$s_{\vec{MP}} = \int_{\vec{M}}^{\vec{P}} \sqrt{E(u,v)du^2 + 2F(u,v)dudv + G(u,v)dv^2} \tag{3.21}$$

which can be computed by substituting

$$\begin{aligned} u &= u_M + t(u_P - u_M) \\ v &= v_M + t(v_P - v_M) \\ 0 &\leq t \leq 1 \end{aligned} \tag{3.22}$$

which leads to

$$s_{\vec{MP}} = \int_0^1 \sqrt{(u_P - u_M)^2 E(t) + 2(u_P - u_M)(v_P - v_M)F(t) + (v_P - v_M)^2 G(t)}dt \tag{3.23}$$

The required condition is satisfied by approximating an initial set of u_P, v_P and recursively correcting them based on the computation of Eq. 3.23.

The first approximation to (u_P, v_P) is computed by

$$\begin{pmatrix} u_P \\ v_P \end{pmatrix} = \begin{pmatrix} u_M \\ v_M \end{pmatrix} + s_{2D} \begin{pmatrix} v_A - v_B \\ u_B - u_A \end{pmatrix} \tag{3.24}$$

where s_{2D} is an approximation to the element size in the parametric space, computed by

$$s_{2D} = \frac{s}{\sqrt{(v_A - v_B)^2 E(u_M, v_M) + 2(v_A - v_B)((u_B - u_A)F(u_M, v_M) + (u_B - u_A)^2 G(u_M, v_M)}}$$

The approximation of Eq. 3.24 creates a point $\vec{P}$ whose parametric coordinates create an isosceles triangle with the Front Edge in 2D.

$\vec{P}$ is updated by computing two new points $\vec{P}_1$ and $\vec{P}_2$, with

$$\begin{pmatrix} u_{P_1} \\ v_{P_1} \end{pmatrix} = \begin{pmatrix} u_A \\ v_A \end{pmatrix} + \frac{s}{s_{\vec{MP}}} \begin{pmatrix} u_P - u_A \\ v_P - v_A \end{pmatrix}$$
$$\begin{pmatrix} u_{P_2} \\ v_{P_2} \end{pmatrix} = \begin{pmatrix} u_B \\ v_B \end{pmatrix} + \frac{s}{s_{\vec{MP}}} \begin{pmatrix} u_P - u_B \\ v_P - v_B \end{pmatrix} \tag{3.25}$$

and, finally, the new point is updated by

$$\begin{pmatrix} u_P \\ v_P \end{pmatrix} = \begin{pmatrix} (u_{P_1} + u_{P_2})/2 \\ (v_{P_1} + v_{P_2})/2 \end{pmatrix} \tag{3.26}$$

When $s_{\vec{MP}}$ tends to become equal to s, then both $\vec{P_1}$ and $\vec{P_2}$ tend to coincide. This creates an isosceles triangle in 3D with a height equal to s.

Next step is to define a search radius in the 2D space that will be used to identify neighbouring Front Points with which triangle formation will be investigated. The radius is defined as

$$R = (1.5 \div 2.5) \cdot max(\sqrt{(u_B - u_A)^2 + (v_B - v_A)^2}, \sqrt{(u_P - u_M)^2 + (v_P - v_M)^2}) \tag{3.27}$$

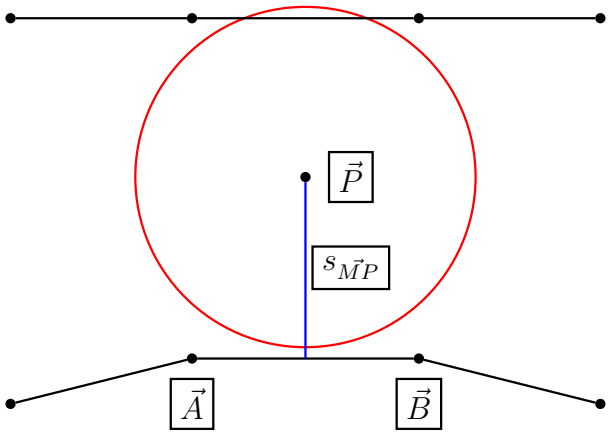


Figure 3.18: The advancing front during the placement of a new Front Point. Black lines indicate the Front Edges while the red circle marks the search area of radius R.

The Quadtree then returns a list of Front Points that have a distance from (u_P, v_P) which is less than R and, then, the triangle formation routine is initiated. The steps that are followed are:

1. Check whether the selected Front Edge $\vec{AB}$ is part of a closed triangle. Namely check if the first point of the previous edge is the same as the last point of the next edge.

2. Loop over the list of the neighbouring Front Points and assess if the 3D distance between a point of the list and $\vec{P}$ is less than $0.8s$. Once such a point is found, perform triangle feasibility tests and create the triangle if it is valid.

3. If the entire list of neighbouring points is checked and no triangle can be created, then perform a triangle feasibility test for triangle creation with the new computed Front Point $\vec{P}$.

4. If triangle formation with $\vec{P}$ is also not possible, then attempt to create a triangle with the Front Points of the neighbouring points list not checked in step 2.

5. Finally, if none of the above steps has successfully created a triangle, perform a brute force technique by investigating if a triangle can be created with any Front Point in the advancing front list. If this final step fails, the algorithm terminates without completing the triangulation.

The triangle formation tests, mentioned in the list above, check a candidate triangle for three possible failures: (a) zero area, (b) Front Point in triangle and (c) triangle with Front Edge intersection. For a triangle to be formed, all three tests must be passed. Analysis on how the tests are made can be seen in Appendix B.

During the triangle creation routine, the advancing front can be updated in two ways: (a) by introducting the new Front Point $\vec{P}$ and (b) by updating the Front with one of the existing Front Points.

When the algorithm opts for a triangle generation with a new point, then the configuration of the front is as in Fig. 3.19.

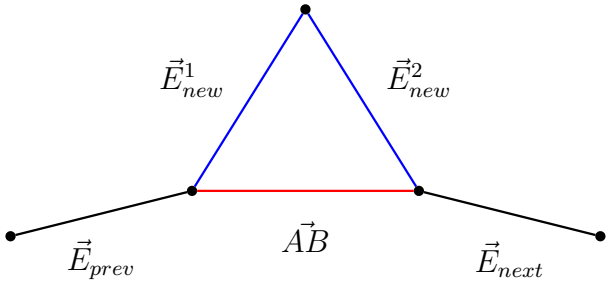


Figure 3.19: An advancing front configuration during the formation of a triangle with a new Front Point. Black lines indicate the unchanged Front Edges, blue lines the newly created Front Edges, and the red line the deleted Front Edge.

The selected Front Edge $\vec{AB}$ is deleted and two new Edges are introduced to connect to the new Front Point ($\vec{E}^1_{new}$, $\vec{E}^2_{new}$). The two new edges are placed at the end of the Front Edge list in order for them to be processed after all the already placed ones. The topology pointers are updated and the new Front Point is added to the Front Point list as well as the Quadtree.

When the algorithm opts for a triangle generation with the use of an existing point ($\vec{P}_{exist}$), then topology of all the neighbouring edges must be checked. This is because the triangle could potentially be created with more than one existing Front Edges. The four configurations that could be identified are seen in Fig. 3.20.

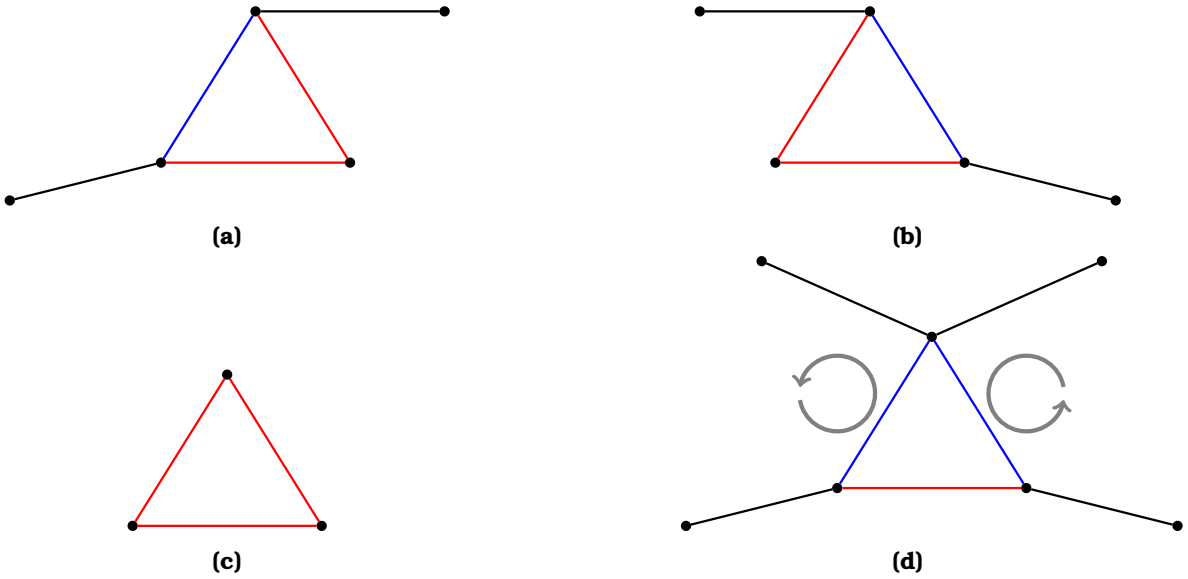


Figure 3.20: Four different advancing front configurations that could be encountered by the algorithm when forming a new triangle with an existing point. In all figures, red lines indicate the Front Edges that will eventually be deleted and blue lines the ones that will be created. Black lines are those that remain unchanged.

Firstly, the Front Edges $\vec{E}_{prev}$ and $\vec{E}_{next}$ are checked. If $\vec{P}_{exist}$ belongs to $\vec{E}_{prev}$, then both the base $\vec{AB}$ and $\vec{E}_{next}$ are removed from the front and a new Front Edge connecting $\vec{A}$ to $\vec{P}_{exist}$ is created. Front Point $\vec{B}$ is removed from both the Front Point List and the Quadtree. The topology of the neighbouring Front Edges is then updated to account for the new edge. Similarly, if $\vec{P}_{exist}$ belongs to $\vec{E}_{prev}$, this edge and the base are removed and a Front Edge connecting $\vec{B}$ to $\vec{P}_{exist}$ is created. Point $\vec{A}$ is permanently removed and topology is updated. These two configurations can be seen in Figs. 3.20a, 3.20b.

Then, a check must be made for whether $\vec{AB}$ is part of a closed loop of three Front Edges, thus creating a triangle. If this is the case, then all three edges are removed. If the associated Front Points are not connected to any more Front Edges, these are removed as well (both from the list and the Quadtree). The configuration can be seen in Fig. 3.20c.

The final case is when $\vec{P}_{exist}$ belongs to other Front Edges opposite of $\vec{AB}$ (Fig. 3.20d). In order to create this triangle, two new Front Edges are generated. This results in the division of the current advancing front structure to two sub-fronts.

The topology of the Front Edges is such that both sub-fronts create counterclockwise loops. The Front Edge $\vec{AB}$ is removed. The procedure above is repeated for all Front Edges (existing and newly created ones). Once the list of Front Edges becomes emptied, the process terminates.

3.3 Applications

In order to showcase the progress of the triangulation of the CAD surfaces, various stages of the advancing front method applied to a cylinder is shown (Figs. 3.21, 3.22).

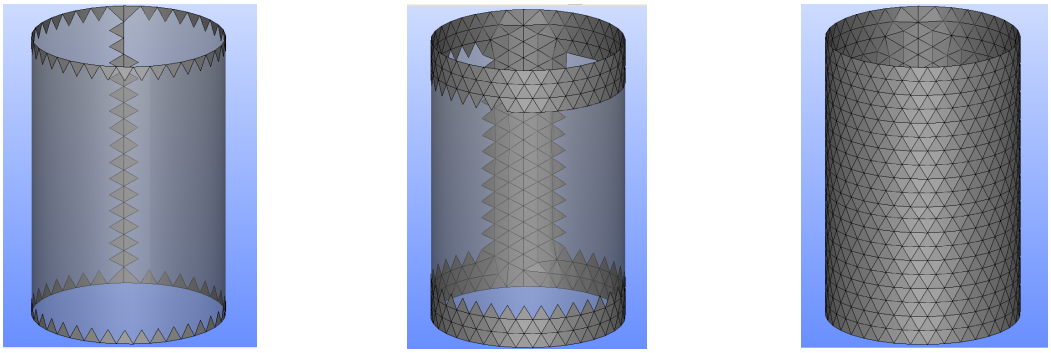


Figure 3.21: Stages of the triangulation of a cylindrical surface (transparent grey). From left to right, the first 100 triangles, 500 triangles and the full triangulation (1400 triangles) can be seen.

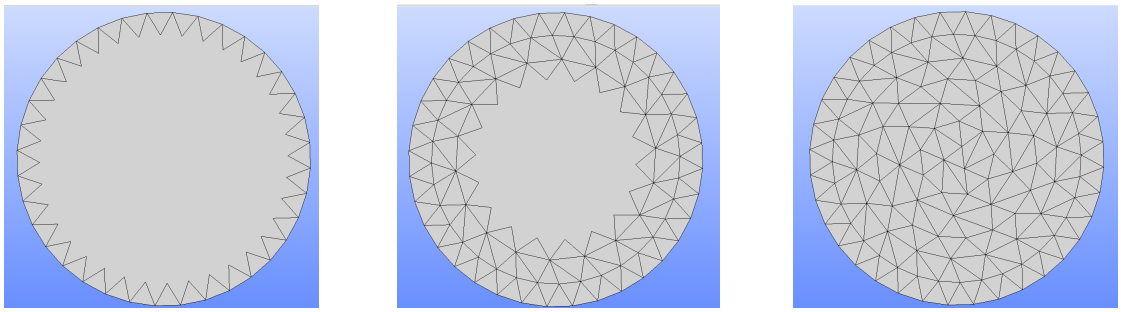


Figure 3.22: Stages of the triangulation of a circular surface (grey). From left to right, the first 36 triangles, 144 triangles and the full triangulation (220 triangles) can be seen.

Further to this simple example, a few industrial-like CAD models are triangulated.

3.3.1 The Drivaer Passenger Car

Initially, the triangulation of the CAD model of the DrivAer fastback car is shown. The model, which is provided via a STEP file, is seen in Fig. 3.3.

Then, the triangulation algorithm starts. The background grid is the one seen in Fig. 3.14 and using that background grid, the Advancing Front method is executed. Timestamps of the process of the algorithm can be seen in Fig. 3.23.

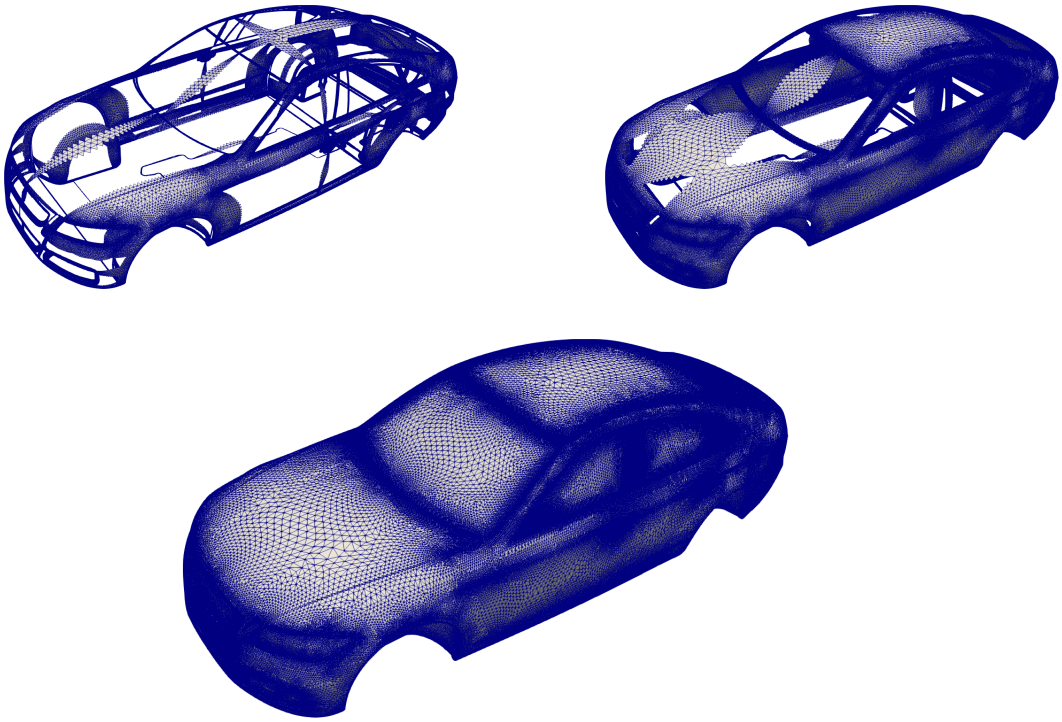


Figure 3.23: The triangulated surface of the Drivaer car model in 3 instances: Top-Left: 282K triangles. Top-Right: 462K triangles. Bottom: 663K triangles.

Zoomed images of the final triangulation are shown in Figs. 3.24 and 3.25. In the former, parts of the surface that comprises the car body are shown. In the latter, one can see the side mirror and the wheel and rim configuration.

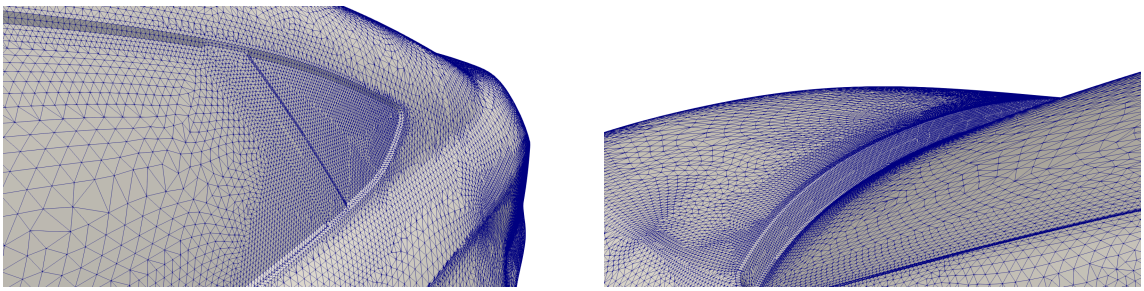


Figure 3.24: The triangulated surface of the DrivAer CAD model (bottom) along with zoomed images of the back left passenger window (top-left) and the windshield-hood interface (top-right).

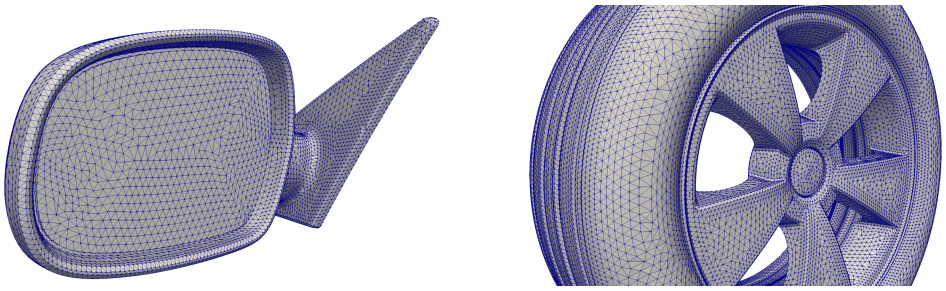
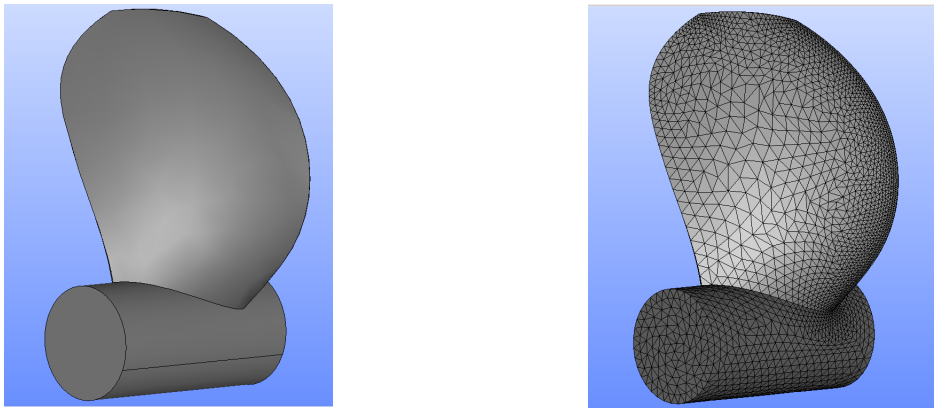


Figure 3.25: The triangulated surfaces of one of the side mirrors (left) and of one of the wheels (right) of the DrivAer CAD model.

A total of 663K triangles are generated using $d = 10^o$ and $g_R = 0.1$. The completion time is 578 seconds, averaging at a computational speed of 1147 triangles per second.

3.3.2 A Ship's Propeller Blade

Then next application of the method is for a ship's propeller (blades and shaft incuded). The CAD model (transferred via a STEP file) can be seen in Fig. 3.26a. The triangulation generated using $d = 10^o$ and $g_R = 0.1$ is seen in Fig. 3.26b.



(a) (b)

Figure 3.26: The triangulated surface of a ship propeller. The CAD model can be seen on the left while the triangulated model on the right.

Instances of the process can be seen in Fig. 3.27.

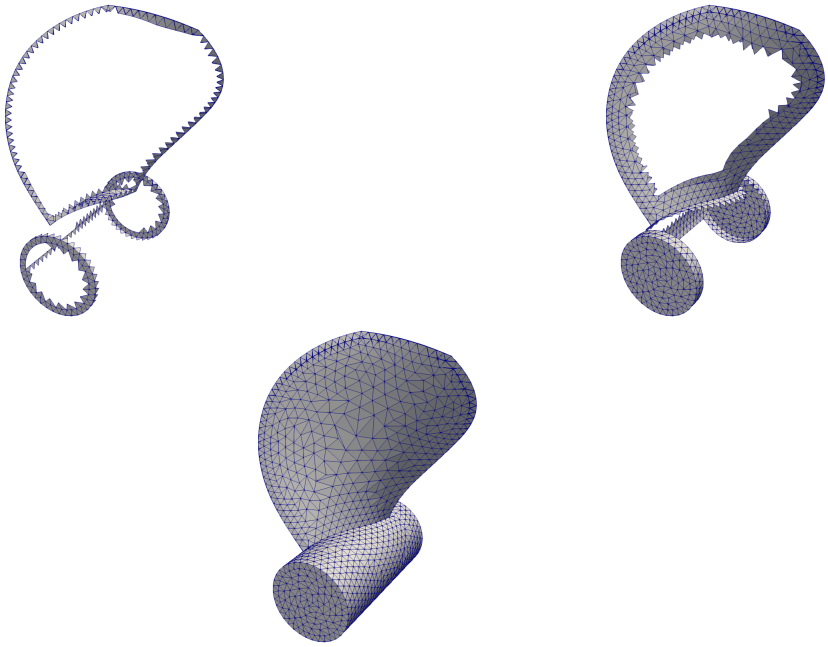


Figure 3.27: The triangulated surface of the propeller model in 3 timestamps: Top-Left: 744 triangles. Top-Right: 2184 triangles. Bottom: 3972 triangles.

A total of 3972 triangles are generated and the completion time is 3.4 seconds, averaging at a computational speed of 1200 triangles per second on 8 Intel Core i7-6700HQ processors.

3.3.3 The S-Bend Climate Duct

Finally, the full body of a climate duct used in automotive is triangulated. The duct consists of a middle S-section and two straight inlet and outlet parts. The CAD model comes via a STEP file here as well and can be seen in Fig. 3.28. The final triangulation is visible in Fig. 3.29 and a few instances can be seen in Fig. 3.30.

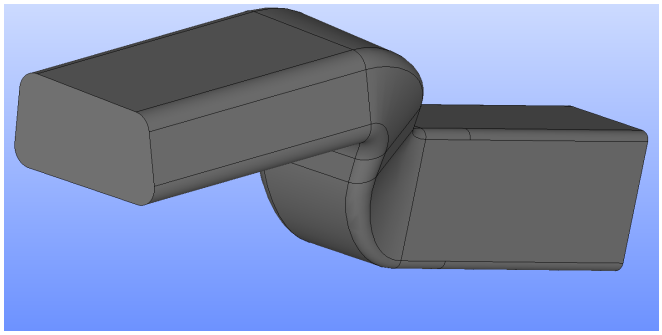


Figure 3.28: The CAD surface of an automotive climate duct.

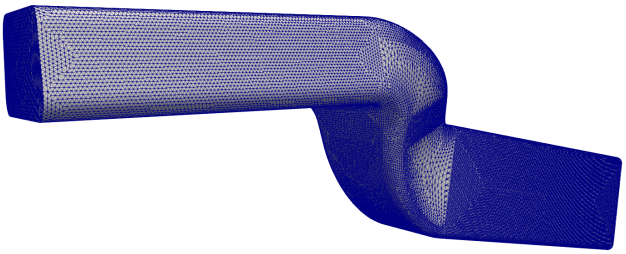


Figure 3.29: The triangulated surface of an automotive climate duct.

The total of 47.2K triangles are generated in 21.6 seconds averaging at ≈2200 triangles per second on 8 Intel Core i7-6700HQ processors.

3.4 Remarks

In order to check the quality of the produced triangular surface grids, the aspect ratio is computed for each element and then some statistics are shown. The aspect ratio in this case is defined as the quotient of the maximum side of a triangle over the minimum side. Obviously, a perfect equilateral triangle will have aspect ratio equal to 1. On the other hand, the higher the average aspect ratio, the worst the quality of the grid. The results for the cases of Sec. 3.3 are shown in table 3.1.

	Average Value	Min. Value	Max. Value	Std. Deviation
Cylinder Case	1.108	1.000	1.872	0.205
Drivaer Car	1.199	1.000	247.4	0.853
Propeller Blade	1.270	1.000	7.905	0.350
S-Bend Duct	1.048	1.000	4.411	0.149

Table 3.1: Aspect ratio for the grids produced for the applications of Sec. 3.3.

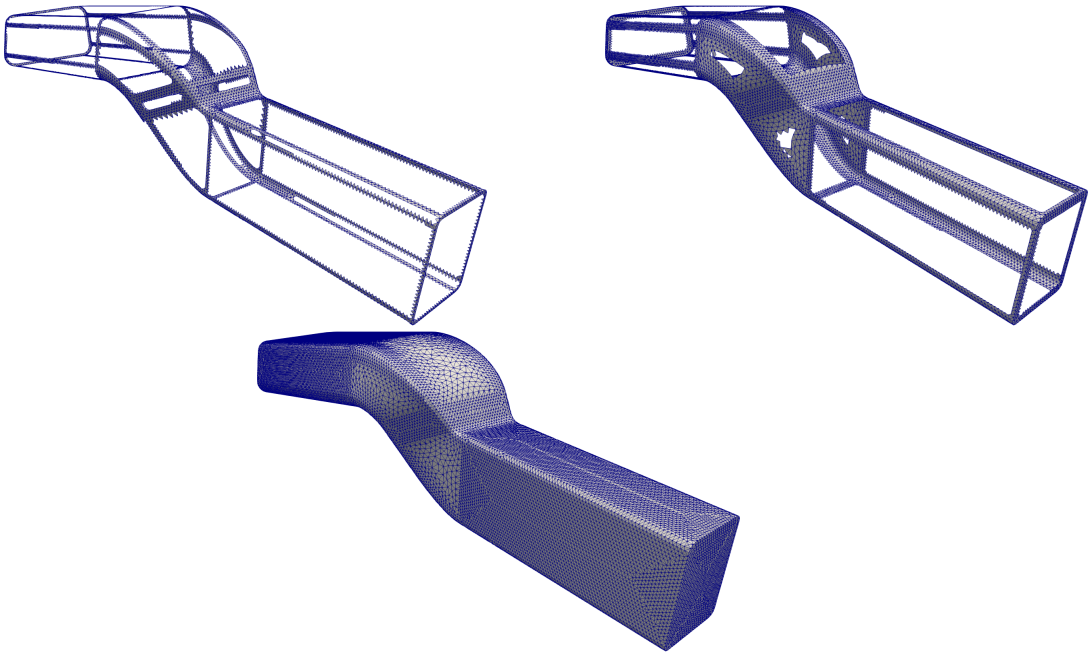


Figure 3.30: The triangulated surface of the S-bend model in 3 timestamps: Top-Left: 7.8K triangles. Top-Right: 16.1K triangles. Bottom: 47.2K triangles.

It is obvious that all of the grids are close to equilateral with average aspect ratio values very close to 1. However, some extreme values are also visible, especially for the Drivaer model. These are due to small, warped surfaces on the car geometry that do not allow the algorithm to produce elements of higher quality.

All the resulting grids can be exported in STL format, which is the main input to 3D meshers such as snappyHexMesh, and are of high quality (equilateral or almost equilateral for the most part). Furthermore, at all grid points, the associated CAD parameters were computed, which will be very useful when the optimization starts.

Chapter 4

The Geometry Morphing Technique

One of the cornerstones of performing shape optimization is the shape and/or mesh deformation technique [**?**, **?**]. The shape and mesh perturbations are closely related to the chosen parameterization, which can greatly affect the progress of the aerodynamic optimization. Depending on the nature of the chosen parameterization, the shape deformation techniques can be categorized as: (a) CAD-free [**?**] or (b) CAD-based [**?**] methods, as also mentioned in Chapter 1. The choice of the parameterization technique depends on various factors which are analyzed below. Ideally, the preferred method would have to ensure a rich-enough design space and, at the same time, maintain the link to the industrial design framework. As stated before, the native CAD parameterization is almost never accessible due to the closed-source nature of CAD packages. Therefore, after triangulating the CAD model and generating a computational mesh around it, the next step is to select a parameterization scheme for the upcoming optimization. In this chapter, a parameterization scheme based on the NURBS patches of the BRep format is proposed.

4.1 Literature Review of Parameterization Methods

The choice of shape parameterization schemes can vary depending on the design requirements for the optimization results. CAD-free methods are preferred when a rich design space is needed and when the link to CAD can be sacrificed. On the other hand, if an optimization related to the geometry of the model which would easily be manageable in an industrial design environment is required (as in this thesis), then CAD-based methods are preferred. For completeness, a review of both CAD-free and CAD-based methods follows.

4.1.1 CAD-free Methods

When performing optimization, the usage of the design parameterization can be a constraint that is, sometimes, very limiting. As a solution, the link to the CAD parameterization can be severed and parameterizations which are specific to the case can be introduced.

The primary and simplest example of CAD-free parameterizations is the node-based ones [**?**, **?**, **?**]. In this parameterization, the nodes of the boundary mesh (that lie on the wall of the optimized geometry) are used to change the shape. The sensitivity derivatives are computed w.r.t. the normal displacement of the boundary nodes [**?**]. The displacement of the boundary mesh is then propagated towards the interior of the computational mesh via various methods such as linear elasticity [**?**], linear and torsional spring analogies [**?**], Laplacian methods [**?**] and algebraic dampening (i.e inverse distance) [**?**]. It is common that, due to a noisy sensitivity map, the node displacements that emerge from such a process, create a non-smooth, wrinkly surface. Apart from the obvious design flaws this can cause (i.e. reduced manufacturability), the propagation of such displacements towards the inner domain can greatly distort the mesh which can result in high numerical errors. For this reason, a common strategy is to smooth the sensitivities before the extrapolation and their propagation towards the interior. This can be done either implicitly [**?**, **?**, **?**] or explicitly [**?**, **?**]. Implicit smoothing means that the updated field of sensitivities is computed by the solution of a PDE while explicit smoothing that the sensitivities are smoothed by taking into account the sensitivities of the neighbouring nodes. For instance, [**?**] performed a Sobolev gradient projection [**?**] in order to smooth the sensitivities (implicitly) while [**?**] performed iterative sensitivity averaging by using the immediate and second neighbours of each node (explicit smoothing).

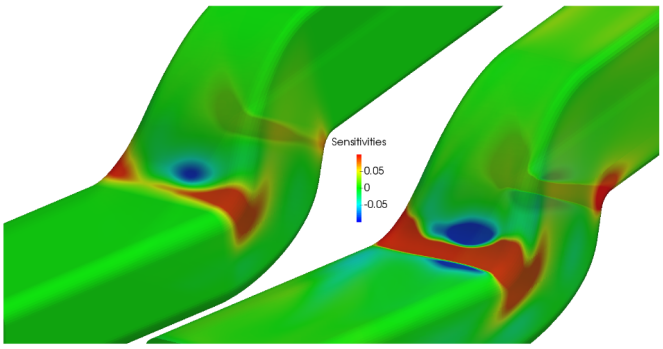


Figure 4.1: The sensitivity derivatives of total pressure losses on the surface of a cooling duct. Sensitivities, computed via the continuous adjoint technique have been smoothed implicitly via Sobolev gradient projection. Smoothed sensitivities are seen on the left while raw ones on the right. [**?**]

Apart from pure node-based parameterization, the CAD-free methods also include control point-based parameterizations such as point-based or lattice-based interpolations. A widely known representative of point-based interpolations is Radial Basis Functions (RBF) [**?**, **?**]. A standard RBF interpolation problem assumes N points $(\vec{x}_1, \ldots, \vec{x}_N)$ that are associated with N function values $(f(\vec{x}_1), \ldots, f(\vec{x}_N))$. The aim is to generate a continuous function $R(\vec{x})$ that interpolates the given values at $(\vec{x}_1, \ldots, \vec{x}_N)$. Such a function is given by [**?**, **?**]

$$R(\vec{x}) = \sum_{i=1}^{N} c_i \phi(||\vec{x} - \vec{x}_i||) \tag{4.1}$$

where $\phi(\cdot)$ are the basis functions and c_i are linear coefficients. The choice of $\phi(\cdot)$ ranges from Gaussians, to Matèrn functions and multiquadratics and c_i is computed by enforcing that $R(\vec{x}_j) = f(\vec{x}_j)$. Returning to the subject of parameterization, the RBF method becomes a shape and mesh deformation tool when for specific (user-defined) nodes of the boundary mesh $(\vec{x}_i)$, the functions $f(\vec{x}_i)$ are the prescribed displacements due to the sensitivity derivatives. After the computation of c_i, $R(\vec{x}_i)$ can provide a smooth displacement field for all nodes of the boundary mesh and the inner domain.

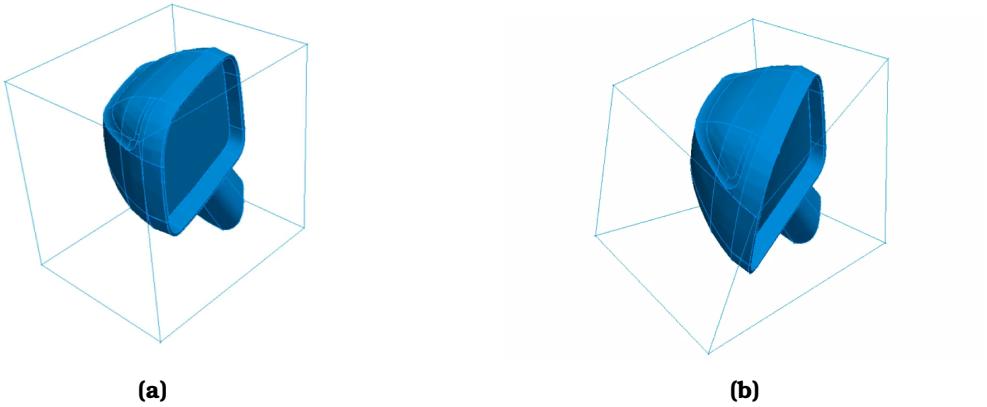
Lattice-based deformation [**?**, **?**, **?**] is the pilar of Free-Form Deformation (FFD) techniques [**?**]. FFD is based on the idea of enclosing a mesh in a hull object (a cube for instance) and deform it as the hull object deforms. The hull is based on the concept of hyper-patches which are 3D analogues of parametric curves or surfaces such as B-splines or NURBS. In this manner, it is highly convenient to select the coordinates of the lattice points as the design variables. In the case of B-splines, any node of the mesh (x_1, x_2, x_3) can be evaluated by a trivariate parametric equation

$$\vec{V}(u, v, w) = \sum_{i=1}^{n_u} \sum_{j=1}^{n_v} \sum_{k=1}^{n_w} N_i^{p_u}(u) N_j^{p_v}(v) N_k^{p_w}(w) \vec{P}_{i,j,k} = (x_1, x_2, x_3) \tag{4.2}$$

Eq. 4.2 is the 3D analogue of the B-splines equations shown in Sec. 1.2.2. u, v, w are the parametric coordinates, $N_i^{p_u}(u) N_j^{p_v}(v) N_k^{p_w}(w)$ are the b-splines basis functions and $\vec{P}_{i,j,k}$ are the position vectors of the lattice points (or control points). For any (x_1, x_2, x_3), the parameters u, v, w can be computed as shown in Appendix C, so that $\vec{V}(u, v, w) = (x_1, x_2, x_3)$. The grid sensitivities w.r.t. the coordinates of any control point can easily be computed by

$$\begin{bmatrix} \frac{\partial x_1}{\partial P^x_{i,j,k}} & \frac{\partial x_1}{\partial P^y_{i,j,k}} & \frac{\partial x_1}{\partial P^z_{i,j,k}} \\ \frac{\partial x_2}{\partial P^x_{i,j,k}} & \frac{\partial x_2}{\partial P^y_{i,j,k}} & \frac{\partial x_2}{\partial P^z_{i,j,k}} \\ \frac{\partial x_3}{\partial P^x_{i,j,k}} & \frac{\partial x_3}{\partial P^y_{i,j,k}} & \frac{\partial x_3}{\partial P^z_{i,j,k}} \end{bmatrix} = \begin{bmatrix} R_{i,j,k} & 0 & 0 \\ 0 & R_{i,j,k} & 0 \\ 0 & 0 & R_{i,j,k} \end{bmatrix} \tag{4.3}$$

where $R_{i,j,k}$ is the product of the B-spline bases shown in Eq. 4.2. Using grid sensitivities, relevant perturbations can be computed for the control points which can displace the nodes via Eq. 4.2. An example of an FFD application is shown in Fig. 4.2.



(a) (b)

Figure 4.2: The body of an automotive side mirror provided by Trabant (www.trabant-nt.de/). The initial body is seen on the left while the deformed body resulting from a 30^o rotation of the near control points around the $y-$axis on the right.

4.1.2 CAD-based Methods

In industrial applications, where gradient-based optimization and, especially, low-cost approaches based on the adjoint method are in widespread use, the necessity for maintaining a link to CAD is paramount. The optimal aerodynamic shapes generated by the optimization loop cannot be manipulated in a CAD package unless a connection between the CAD design variables and the computational mesh is established. As solutions, either the CAD software (which could be commercial) can be a part of the gradient-based optimization loop, which means that it has to be differentiated, or the optimal geometry could be approximated by a CAD-compatible parameterization (return to CAD [**?**]). The work in this section focuses on the former and overcomes the closed-source nature of most CAD packages, by introducing an open-source, transferable parameterization based on standard

CAD geometry (elemental geometry and NURBS [**?**]) which is widespread in formats like IGES, STEP, etc. The connection of this parameterization to the computational mesh is conducted via the proposed surface grid generation method (Chap. 3).

CAD-based and especially NURBS-based optimization has been a well known practice for quite some time. The field has flourished in various CFD-related [**?**, **?**] and structural related isogeometric analysis applications [**?**, **?**].

Modern CAD systems like SIEMENS NX [**?**], CATIA V5 [**?**], SOLIDWORKS [**?**] and AUTOCAD [**?**] use feature trees to model a geometry. From a design perspective, this is an optimal choice as it enables a user to define various geometric relations (constraints) which is inherently satisfied. However, the process of an aerodynamic opimization includes the change of the design variables which creates two difficulties: (a) the shape derivatives must be computed and (b) topological discontinuities must be overcome. The first difficulty stems from the fact that most CAD packages are closed-source in nature and their vendors are very sensitive regarding source-parameterizations. An example of an open source CAD package that was included in the optimization loop was given by [**?**]. Shape optimization was performed on CAD models by differentiating an entire CAD Kernel. In particular, algorithmic differentiation was applied to the open-source Open-Cascade Technology [**?**] Kernel. Unless an open-source CAD package is used, the only way to compute shape derivatives is by finite differences [**?**]. For instance, [**?**] performed shape optimization of the heat shield of a re-entry capsule by using discrete adjoint. The model geometry of the capsule was designed using a commercial CAD package and the sensitivities of the model surface nodes with respect to (w.r.t.) the CAD design variables were computed using finite difference approximations. Such methods can become computationally expensive and can hinder the speed of the optimization process. The second difficulty stems from the fact that the resulting CAD model must be coupled with the computational mesh. However, a CAD-feature tree with an updated parametric input can potentially generate different CAD faces and edges both in number and in topological relations. This means that, on the one hand, the computational mesh must be projected onto the CAD model at each optimization cycle and, on the other hand, that grid sensitivities produce discontinuities. This has a negative impact on the computation of sensitivity derivatives and the optimization itself. For these facts, strict CAD-based optimization is most commonly used with stochastic optimization methods [**?**].

NURBS have often been used as an alternate parameterization which is CAD-compatible and can overcome the above mentioned drawbacks. NURBS is the geometric standard for various CAD formats which are open source and can easily be distributed even among commercial packages. The geometry that is computed in any CAD package is parametric and is either NURBS geometry or can be transformed to a NURBS geometry in a straightforward manner. NURBS patches

that are contained in the boundary representation of a model define the surface boundary between solid and non-solid regions. Therefore, the boundary of a computational mesh can easily be projected on that representation without worrying about topology discontinuities. Furthermore, the derivatives of the NURBS control points w.r.t. any boundary mesh node can be computed analytically.

NURBS-based optimization has been performed in various applications ranging from 2D to 3D. The applications of NURBS in the optimization of airfoils [**?**, **?**], turbomachinery blades [**?**] and 3D wings [**?**, **?**] have been extensive. However, there has been an absence of literature for generalized NURBS usage in 3D design optimization. For instance, [**?**] employed NURBS surfaces with natural parametric boundaries as the parameterization tool. The differentiation of the NURBS was done through AD techniques [**?**, **?**, **?**]. Similarly, [**?**, **?**] performed NURBS-based shape optimization with continuity constraints imposed between adjacent patches. Both control points and weights were used for parameterizing the shape and the AD tool TAPENADE [**?**] was used for computing the derivatives. Both methods proved promising but did not overcome continuity issues due to the trimmed patches. Attempts to address such continuity issues and the coupling of NURBS patches have been identified in structural isogeometric analysis applications. For instance, [**?**] used trimmed NURBS surfaces to perform topology optimization of shell structures. In [**?**], a method was presented for coupling non-conforming NURBS patches in isogeometric frameworks via a master-slave configuration and examples of imposed C_0 and C_1 continutiy were shown. In [**?**], design and isogeometric analysis was performed on trimmed multi-patch BRep models of structural membranes while in [**?**], isogeometric analysis of the BRep was performed while handling trimmed NURBS patches. This was achieved by including computational solvers at the CAD design level. Finally, in [**?**], isogeometric analysis of thin shells was performed by using the STEP-format inherent parametric curves and blending functions.

4.1.3 The Proposed Method

In this section, a parameterization scheme is developed, which uses the NURBS patches contained in the BRep of standard files, to perform aerodynamic shape optimization. The main drive of this work, is to perform CAD-based adjoint optimization, in any 3D shape, irrespectively of CAD packages and without using finite differences or AD to compute shape derivatives. Taking the requirements and the pros and cons of the above mentioned methods into account, NURBS-based optimization is chosen as a go-to method if the link to CAD must be maintained. The NURBS patches have a rich design space which can be enriched even further should there be a requirement. Furthermore, NURBS can be saved in standard CAD formats which means that both the initial and final geometries can be processed in a CAD viewer. Therefore, a new NURBS-based parameter-

ization is proposed. The differentiation of the NURBS shapes is included in the proposed parameterization scheme which is, then, seamlessly coupled with the adjoint method. Furthermore, the proposed method adequately handles arbitrary trims in NURBS surfaces which (in more complex shapes) are very common while similar methods make the assumption of natural parametric boundaries. The NURBS surfaces contained in standard CAD files are firstly triangulated and, then, used to parameterize the shape. Initially, the boundary mesh nodes are projected onto the NURBS patches, thus creating a map between the boundary mesh and parametric spaces of different patches. Then, the control points are used as a means of updating the shape undergoing optimization (both its CAD geometry and its boundary mesh). The NURBS geometry contained in standard CAD files consists, mainly, of trimmed patches. The trims are generally arbitrary and independent of the patches they connect. A potential shape update can, therefore, create C_0 (geometric) and C_1 (smoothness) continuity issues, unless the control points of neighbouring patches are constrained to move in accordance. The proposed method imposes point-wise geometric and/or smoothness constraints along the trimming lines of neighbouring patches. Then, the orthonormal basis of the Null Space [**?**] of the constraints' Jacobian is computed and used to project the control points' displacements on that basis.

4.2 Formulating the new Parameterization by imposing C_0 and C_1 Constraints

The shape parameterization consists of trimmed 3D NURBS patches which must be constrained by imposing continuity and smoothness constraints between the neighbouring ones.

4.2.1 Imposing C_0 Continuity Constraints

During NURBS-based optimization, the natural choice for the design variables $\vec{b}$ is that of the control point coordinates. However, when handling trimmed patches, the control points must be constrained to move in coordination, to ensure watertightness. In what follows, two surfaces in contact along an arbitrary trimming curve (Fig. 4.3), being watertight along it, are assumed.

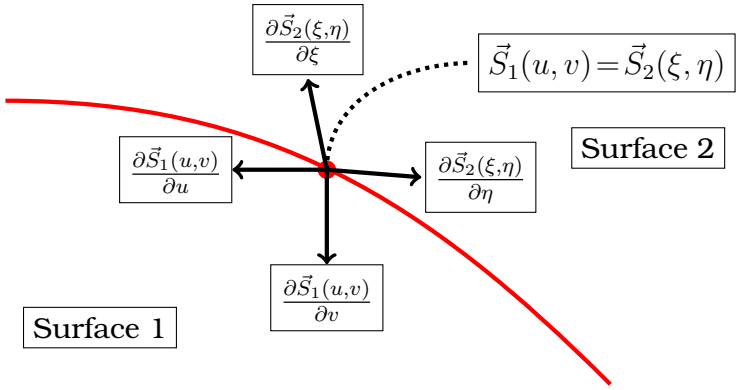


Figure 4.3: A trimming curve between two surfaces (in red) along with a node on it. Black arrows show the partial derivatives of each surface on the node, w.r.t. the parametric coordinates.

Both are NURBS surfaces the points of which are computed in 3D by equations similar to Eq. 1.4 (denoted as $\vec{S}_1(u,v)$ and $\vec{S}_2(\xi,\eta)$ respectively). This means that for any point along the trimming curve, which corresponds to parametric coordinates (u,v) and (ξ,η), the two surface equations evaluate the same 3D coordinates.

$$\vec{S}_1(u,v) - \vec{S}_2(\xi,\eta) = \vec{0} \Leftrightarrow$$
$$\sum_{m_1=1}^{n_1} R^1_{m_1}(u,v)\vec{P}^1_{m_1} - \sum_{m_2=1}^{n_2} R^2_{m_2}(\xi,\eta)\vec{P}^2_{m_2} = \vec{0} \tag{4.4}$$

The three equations (one per cartesian direction) represented by Eq. 4.4 can be written as

$$\begin{bmatrix} 0 & 0 & 0 \end{bmatrix} = \overbrace{\begin{bmatrix} R^1_1 & R^1_2 & \cdots & R^1_{n_1} \end{bmatrix}}^{\text{Basis functions of Surf. 1}} \overbrace{\begin{bmatrix} -R^2_1 & -R^2_2 & \cdots & -R^2_{n_2} \end{bmatrix}}^{\text{Basis functions of Surf. 2}} \cdot \begin{bmatrix} X^1_1 & Y^1_1 & Z^1_1 \\ X^1_2 & Y^1_2 & Z^1_2 \\ \vdots & \vdots & \vdots \\ X^1_{n_1} & Y^1_{n_1} & Z^1_{n_1} \\ X^2_1 & Y^2_1 & Z^2_1 \\ X^2_2 & Y^2_2 & Z^2_2 \\ \vdots & \vdots & \vdots \\ X^2_{n_2} & Y^2_{n_2} & Z^2_{n_2} \end{bmatrix} \tag{4.5}$$

where X_i^j, Y_i^j, Z_i^j denote the 3D coordinates of the i^{th} control point of the j^{th} surface. Eqs. 4.4 and 4.5 can be written for any number of different points along the trimming curve. The parameters (u, v) and (ξ, η) are different for each point, which means that the row matrix of Eq. 4.5 is also different as it contains the basis functions. Assuming that κ points along the trimming curve are used to generate that many equations and $B_1, B_2, \cdots, B_\kappa$ denote the different row matrices, Eq. 4.5 can be written κ times yielding

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix} = \left[\begin{array}{c} B_1 \\ \hline B_2 \\ \hline B_3 \\ \hline \vdots \\ \hline B_\kappa \end{array}\right] \cdot \begin{bmatrix} X_1^1 & Y_1^1 & Z_1^1 \\ X_2^1 & Y_2^1 & Z_2^1 \\ X_3^1 & Y_3^1 & Z_3^1 \\ \vdots & \vdots & \vdots \\ X_{n_1}^1 & Y_{n_1}^1 & Z_{n_1}^1 \\ X_1^2 & Y_1^2 & Z_1^2 \\ X_2^2 & Y_2^2 & Z_2^2 \\ X_3^2 & Y_3^2 & Z_3^2 \\ \vdots & \vdots & \vdots \\ X_{n_2}^2 & Y_{n_2}^2 & Z_{n_2}^2 \end{bmatrix} = \begin{bmatrix} \quad B \quad \end{bmatrix} \cdot \left[\begin{array}{c|c|c} X & Y & Z \end{array}\right] \tag{4.6}$$

where B is a matrix containing NURBS basis functions and X, Y, Z are column matrices containing the 3 coordinates of the control points of both sufaces. Eq. 4.6 can be re-written in block-matrix form as

$$\left[\begin{array}{c|c|c} B & 0 & 0 \\ \hline 0 & B & 0 \\ \hline 0 & 0 & B \end{array}\right] \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = C \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.7}$$

which holds for the initial control point coordinates. Since NURBS are rational polynomials, C_0 continuity can be ensured by satisfying Eq. 4.7 if κ is large enough. Eq. 4.7 can also be differentiated w.r.t. the control point coordinates yielding

$$C \cdot \begin{bmatrix} \delta X \\ \delta Y \\ \delta Z \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.8}$$

Solving Eqs. 4.7, 4.8, can lead to a homogenous solution (zero vector) which, of

course, is of no interest. Their non-zero solutions lie in the null space [1] **[?]** of matrix C. Therefore, the next step in this process is to calculate the orthonormal basis ($Kernel(C)$) of that null space and use it to express the control point positions as well as their derivatives.

The null space of matrix C is the subspace of its domain that containts all vectors $\vec{V}$ such that $C \cdot \vec{V} = \vec{0}$. The image of C is isomorphic to the quotient of $\vec{V}$ by the null space which implies the rank-nullity theorem:

$$dim(NullSpace(C)) + rank = dim(V) \tag{4.9}$$

where rank denotes the dimension of the image of C. This implies that for a null space that contains more than the zero vector, matrix C must have a rank smaller than the length of its row vectors. The product $C \cdot \vec{V}$ can be written as a dot product of its row vectors

$$C \cdot \vec{V} = \begin{bmatrix} \vec{C}_1 \cdot \vec{V} \\ \vec{C}_2 \cdot \vec{V} \\ \vdots \\ \vec{C}_{3\kappa} \cdot \vec{V} \end{bmatrix} \tag{4.10}$$

For $\vec{V}$ to belong to the null space, it must be orthogonal to each row C_i and if such a non-zero vector exists, then matrix C has non-homogenous solutions. In this case, it is clear from Eq. 4.7 that there exists such a non-zero vector for matrix C, due to the existence of the non-homogenous solutions. Therefore, the following conditions hold:

- If $\vec{V}$ belongs to the null space of C, any vector $\alpha\vec{V}$ belongs to the same null space, since $C(\alpha\vec{V}) = \alpha C\vec{V} = \alpha \cdot \vec{0} = \vec{0}$.
- If two vectors $\vec{V}_1$, $\vec{V}_2$ belong to the null space of C, then $\vec{V} = \vec{V}_1 + \vec{V}_2$ also belongs to the same null space, as $C\vec{V} = C(\vec{V}_1 + \vec{V}_2) = C\vec{V}_1 + C\vec{V}_2 = \vec{0}$

The dimension of the null space or, in other words, the reduction of the rank is indicated by the number of the right singular values of C that are zero. If for a singular value $\sigma_i = 0$, the corresponding singular vector is $\vec{U}_i$, then it holds that

$$C\vec{U}_i = \sigma_i\vec{U}_i = \vec{0} \tag{4.11}$$

The orthonormal basis of the null space of matrix C, can, therefore, be computed by using singular value / vector analysis **[?]**. Based on the conditions that stem

[1] The null space of any matrix A consists of all the vectors B such that $AB = 0$ and B is not zero. The size of the null space of A provides us with the number of linear relations among attributes.

from the non-homogenity of matrix C (listed above), any linear combination of all $\vec{U}_i$ that belong to that null space, also lies in the same null space. This conclusion provides an option to express any vector $\vec{V}$ that lies in the null space in terms of $Kernel(C)$:

$$\vec{V} = \sum_{i=1}^{L} \alpha_i \vec{U}_i = \begin{bmatrix} \vec{U}_1 & \vec{U}_2 & \cdots & \vec{U}_L \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_L \end{bmatrix} = Kernel(C) \cdot \vec{\alpha} \tag{4.12}$$

where L is the number of values $\sigma_i = 0$, $\vec{U}_i$ are the corresponding singular vectors and α_i are the null space basis coordinates. One can ensure that the vector of control point coordinates belongs to the null space of C by setting

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = Kernel(C) \cdot \vec{\alpha} \tag{4.13}$$

The initial value of $\vec{\alpha}$ can be computed via a least-square pseudo-inversion

$$\vec{\alpha} = \left(Kernel(C)^T Kernel(C)\right)^{-1} Kernel(C)^T \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{4.14}$$

By differentiating both Eqs. 4.13 and 4.14, bidirectional relations between the control point positions and the null space coordinates become available

$$\begin{bmatrix} \delta X \\ \delta Y \\ \delta Z \end{bmatrix} = Kernel(C) \cdot \delta\vec{\alpha}$$

$$\delta\vec{\alpha} = \left(Kernel(C)^T Kernel(C)\right)^{-1} Kernel(C)^T \begin{bmatrix} \delta X \\ \delta Y \\ \delta Z \end{bmatrix} \tag{4.15}$$

Eq. 4.15 will be used when performing optimization. The null space coordinates $\vec{\alpha}$ will, eventually, be used as some of the design variables and the relations with the control point coordinates will be used when applying the chain rule and when updating the shape.

4.2.2 Imposing C_1 Continuity Constraints

If the two neigbouring surfaces of Fig. 4.3 are C_1 continuous, then their normal vectors on points along the trimming curve are parallel. Therefore, (assuming that they are also C_0 continuous) their tangent planes must be the same. This means that the vectors denoted by the parametric derivatives of each surface must lie on that same plane. That can be expressed as

$$\begin{aligned} \frac{\partial \vec{S}_1(u,v)}{\partial u} - a\frac{\partial \vec{S}_2(\xi,\eta)}{\partial \xi} - b\frac{\partial \vec{S}_2(\xi,\eta)}{\partial \eta} = 0 \\ \frac{\partial \vec{S}_1(u,v)}{\partial v} - c\frac{\partial \vec{S}_2(\xi,\eta)}{\partial \xi} - d\frac{\partial \vec{S}_2(\xi,\eta)}{\partial \eta} = 0 \end{aligned} \tag{4.16}$$

Coefficients a, b, c, d are computed by solving

$$M \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \frac{\partial \vec{S}_1(u,v)}{\partial u} \cdot \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \xi} \\ \frac{\partial \vec{S}_1(u,v)}{\partial u} \cdot \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \eta} \end{bmatrix} \tag{4.17}$$

$$M \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} \frac{\partial \vec{S}_1(u,v)}{\partial v} \cdot \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \xi} \\ \frac{\partial \vec{S}_1(u,v)}{\partial v} \cdot \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \eta} \end{bmatrix} \tag{4.18}$$

where

$$M = \begin{bmatrix} \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \xi} \cdot \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \xi} & \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \xi} \cdot \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \eta} \\ \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \eta} \cdot \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \xi} & \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \eta} \cdot \frac{\partial \vec{S}_2(\xi,\eta)}{\partial \eta} \end{bmatrix}$$

Writing Eqs. 4.16 in matrix form yields

$$\begin{bmatrix} \overbrace{\frac{\partial R^1_1}{\partial u} & \cdots & \frac{\partial R^1_{n_1}}{\partial u}}^{\text{Basis functions of surf. 1}} & \overbrace{(-a\frac{\partial R^2_1}{\partial u} - b\frac{\partial R^2_1}{\partial v}) & \cdots & (-a\frac{\partial R^2_{n_2}}{\partial \xi} - b\frac{\partial R^2_{n_2}}{\partial \eta})}^{\text{Basis functions of surf. 2}} \\ \frac{\partial R^1_1}{\partial v} & \cdots & \frac{\partial R^1_{n_1}}{\partial v} & (-c\frac{\partial R^2_1}{\partial \xi} - d\frac{\partial R^2_1}{\partial \eta}) & \cdots & (-c\frac{\partial R^2_{n_2}}{\partial \xi} - d\frac{\partial R^2_{n_2}}{\partial \eta}) \end{bmatrix} \cdot \left[\begin{array}{c|c|c} X & Y & Z \end{array}\right] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.19}$$

which holds if the two surfaces are C_1 continuous. The two equations of 4.19 are written κ times for the same points on the trimming curve as described in subsection 4.2.1. Then, matrix B of Eq. 4.6 is created by taking into account both C_0 (Eq. 4.5) and C_1 constraints (Eq. 4.19). Matrix B has 3κ rows. Matrices

C and $Kernel(C)$ are generated as in Eqs. 4.7-4.13 for the larger B matrix and the update of the control points is calculated as in Eq. 4.15.

4.2.3 Constraining Multi-Patch Models

In subsections 4.2.1 and 4.2.2, a method to constrain two neighbouring trimmed patches was shown. More patches can be constrained to move in coordination if the proposed method is applied to every pair of neighbouring patches. In this respect, displaceable or non-displaceable patches can be defined as allowed or not allowed to be deformed during the optimization, respectively. Non-displaceable patches can also be defined and constrained to maintain up to C_1 continuity with their displaceable neighbours. When such a patch pair is defined, the constraints are defined only on the displaceable surface. For C_0 constraints, Eq. 4.5 is formulated so that the displacement of the displaceable surface is zero along the trimming edge. If $\vec{S}_1$ is considered displaceable and $\vec{S}_2$ non-displaceable Eq. 4.5 becomes

$$\vec{S}_2(\xi,\eta) = \begin{bmatrix} R_1^1 & R_2^1 & R_3^1 & \cdots & R_{n_1}^1 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \cdot \left[\begin{array}{c|c|c} X & Y & Z \end{array}\right] \tag{4.20}$$

and by differentiating it

$$\begin{bmatrix} 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} R_1^1 & R_2^1 & R_3^1 & \cdots & R_{n_1}^1 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \cdot \left[\begin{array}{c|c|c} \delta X & \delta Y & \delta Z \end{array}\right] \tag{4.21}$$

Similarly to the C_0 constraints, C_1 constraints between a displaceable and a non-displaceable surface are formulated as

$$\begin{bmatrix} a\frac{\partial \vec{S}_2(\xi,\eta)}{\partial \xi} + b\frac{\partial \vec{S}_2(\xi,\eta)}{\partial \eta} \\ c\frac{\partial \vec{S}_2(\xi,\eta)}{\partial \xi} + d\frac{\partial \vec{S}_2(\xi,\eta)}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial R_1^1}{\partial u} & \frac{\partial R_2^1}{\partial u} & \frac{\partial R_3^1}{\partial u} & \cdots & \frac{\partial R_{n_1}^1}{\partial u} & 0 & 0 & 0 & \cdots & 0 \\ \frac{\partial R_1^1}{\partial v} & \frac{\partial R_2^1}{\partial v} & \frac{\partial R_3^1}{\partial v} & \cdots & \frac{\partial R_{n_1}^1}{\partial v} & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \cdot \left[\begin{array}{c|c|c} X & Y & Z \end{array}\right] \tag{4.22}$$

and by differentiating it

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial R_1^1}{\partial u} & \frac{\partial R_2^1}{\partial u} & \frac{\partial R_3^1}{\partial u} & \cdots & \frac{\partial R_{n_1}^1}{\partial u} & 0 & 0 & 0 & \cdots & 0 \\ \frac{\partial R_1^1}{\partial v} & \frac{\partial R_2^1}{\partial v} & \frac{\partial R_3^1}{\partial v} & \cdots & \frac{\partial R_{n_1}^1}{\partial v} & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \cdot \left[\begin{array}{c|c|c} \delta X & \delta Y & \delta Z \end{array}\right] \tag{4.23}$$

Due to the existence of non-displaceable patches and the locality property of NURBS, matrix B is sparse. The columns of matrix B that correspond to control points of non-displaceable patches is filled with zeros. Furthermore, the same applies to control points of displaceable patches that are away from the trims and, practically, do not influence the constraints. These columns of B and the corresponding rows of X, Y, Z are removed and are not taken into account while computing $Kernel(C)$. On the one hand, the displacements of the control points on non-displaceable patches are manually set to zero. On the other hand, the displacements of the control points of displaceable patches that are away from the trims are left unconstrained.

To summarize, to fully constrain a CAD model and prepare it for optimization, the following steps are followed:

1. Define the displaceable and non-displaceable patches of the model.

2. For each pair of neighbouring patches that is either displaceable-displaceable or non displaceable-displaceable, impose C_0 and (where required) C_1 constraints on κ points along the trimming curve. The size of κ depends on the geometric complexity of the trimming curve. In all cases shown in this thesis, for trimming curves of degree p and of λ distinctive knots, $\kappa = \lambda(p+1)$.

3. Using all the formulated constraints for all trimming curves, assemble matrix B. The columns of B are as many as the total number of control points of all the constrained patch pairs minus the number of unconstrained and non-displaceable control points.

4. Formulate matrix C and compute $Kernel(C)$.

The above mentioned process is performed once in the beginning of the optimization loop. Considering that the coordinates of a node (x_1, x_2, x_3) which lies on any of the surfaces can be expressed in terms of the control point coordinates

(Eq. 1.4), they can also be expressed in terms of $\vec{\alpha}$:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = Q_{uncon} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{uncon} + Q_{con} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{con} = Q_{uncon} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{uncon} + Q_{con} Kernel(C)\vec{\alpha} \tag{4.24}$$

where the Q matrices contain the basis functions of the surface on which, node (x_1, x_2, x_3) belongs. The parametric coordinates of each node (x_1, x_2, x_3) used to compute the basis function and the matrices Q, are acquired by interpolating the (u, v) parametric coordinates of the triangulated grid of Chap. 3. Differentiating Eq. 4.24 leads to

$$\begin{bmatrix} \delta x_1/\delta X_{uncon} & \delta x_1/\delta Y_{uncon} & \delta x_1/\delta Z_{uncon} \\ \delta x_2/\delta X_{uncon} & \delta x_2/\delta Y_{uncon} & \delta x_2/\delta Z_{uncon} \\ \delta x_3/\delta X_{uncon} & \delta x_3/\delta Y_{uncon} & \delta x_3/\delta Z_{uncon} \end{bmatrix} = Q_{uncon}$$
$$\begin{bmatrix} \delta x_1/\delta \alpha_1 & \delta x_1/\delta \alpha_2 & \cdots & \delta x_1/\delta \alpha_L \\ \delta x_2/\delta \alpha_1 & \delta x_2/\delta \alpha_2 & \cdots & \delta x_2/\delta \alpha_L \\ \delta x_3/\delta \alpha_1 & \delta x_3/\delta \alpha_2 & \cdots & \delta x_3/\delta \alpha_L \end{bmatrix} = Q_{con} \cdot Kernel(C) \tag{4.25}$$

Eqs. 4.24 and 4.25 create a parameterization scheme based on NURBS which inherently satisfies C_0 and C_1 continuity (where necessary). Matrices Q and $Kernel(C)$ are constant which makes this parameterization linear w. r. t. α_i and the unconstrained control point coordinates. The design variables $\vec{b}$ are set equal to $\vec{\alpha}$ and $X_{uncon}, Y_{uncon}, Z_{uncon}$.

In Fig. 4.4, a simple example of a cylinder is portrayed. The cylinder consists of 3 patches: 2 planar circular disks identical to the one portrayed in Fig. 1.3 and a cylindrical side surface. The bottom circular disk is constrained to stay fixed while the top disk and the side surface are allowed to change. All three patches are C_0 constrained. In total, the top and side surfaces consist of $(7 \times 2 + 2 \times 2)$ control points (54 degrees of freedom). The orthonormal basis of the null space consists of 24 parameters α_i (8 controlling each cartesian direction).

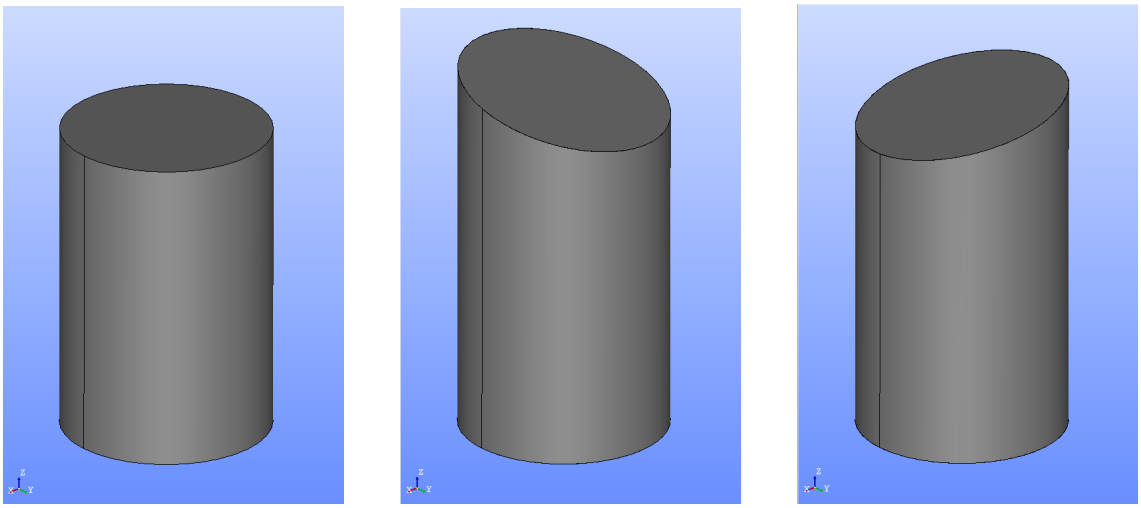


Figure 4.4: CAD model of a cylinder (left) whose patches are constrained to move in coordination. The result of the displacements of two different α_i parameters are shown in the center and right figures.

The control points of the side surface which are near the bottom disk remain fixed regardless of any $\delta\alpha_i$ applied. The control points of the top disk as well as those of the side surface near the top, move as a unit, so that both surfaces remain attached to each oher. In the example, two parameters which are controlling the z coordinates of all the control points, are used to change the shape of the cylinder. The final shape retains its watertightness.

4.2.4 Practical Computation of a Null Space

The most widely used methods for the computation of the rank or the nullity of a $M \times N$ matrix C is the Singular Value Decomposition (SVD) [**?**] and the QR factorization [**?**]. The SVD is usually considered more accurate in computing the singular values and the corresponding vectors. However, the QR factorization has always been the most efficient method with also sufficient accuracy.

In cases of large sparse matrices, both these methods (in their standard forms) can become quite inefficient. This happens, firstly, due to inability to take advantage of the sparsity pattern and, secondly, because less accuracy can compromise the rank-revealing capabilities. For this reason, it is necessary to define the concept of $\epsilon-$nullity. Since, in numerical applications, there exist roundoff errors and approximations, it is understandable that the singular values of C that produce null space vectors, are not always strictly zero. Therefore, a small constant ϵ is defined that acts as a threshold, below which, the singular values are considered numerically zero. Assuming that $M > N$ (as is the case in our applications) an

SVD would factorize C as $C = UDV^T$ where D would be an $M \times N$ diagonal matrix, and U, V would be $M \times M$ and $N \times N$ orthogonal matrices respectively. The entries of matrix D would contain (in decreasing order) the true singular values of C. $\epsilon-$nullity is defined as the number L of the true singular values $\sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_L$, such that $\sigma_{L+1} > \epsilon$. In this case, the rank of C would be equal to $N - L$ and for the right singular vectors $v_1, v_2, \cdots v_L$ it would be true that

$$\frac{||Cv_i||_2}{||vi||_2} \leq \epsilon$$

A value of ϵ that works well in practice is [**?**]

$$\epsilon = 20(N + M)10^{-16} \cdot max(||e_1^T C||, ||e_2^T C||_2, ||e_M^T C||_2)$$

where $e_1, e_2, \cdots, e_M$ are the standard orthonormal basis vectors of $\mathbf{R}^M$. To compute the singular vectors that correspond to singular values less than or equal to ϵ the process of [**?**] is followed. Typically, this process is split into two phases: Firstly, matrix C must be analyzed and re-ordered based on its non-zero pattern and, secondly, the factorization must take place.

As proposed in [**?**], a QR factorization of C is written as

$$CP = QR = Q \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} \tag{4.26}$$

where P is an $N \times N$ permutation matrix and Q, R are the approximations for the requested factors with $M \times M$ and $M \times N$ sizes respectively. Matrix R_1 is an $(N-L)\times N$ right trapezoidal matrix that can be divided into two blocks: R_{11} which is an $(N - L) \times (N - L)$ triangular matrix and R_{12} which has size $(N - L) \times L$. It is proven that all sub-matrices related to R maintain their sparsity pattern which is also related to the upper Cholesky factor of $C^T C$ [**?**]. Furthermore, matrix Q is never saved as a whole but rather as a set of elementary Householder transformations. The permutation P which is used is based on the Column Approximate Minimum Degree method proposed by [**?**]. If C is well-conditioned, then R_{11} is full rank and non-singular. In that case, the null space of C^T is computed as

$$Kernel(C^T) = Q \begin{pmatrix} 0 \\ I_{L\times L} \end{pmatrix} \tag{4.27}$$

If the matrix C is ill-conditioned, then the rank of R_{11} is equal to $(N - L - K)$, $K = 1, 2. \ldots N - L$ and has also a null space. In this case, the left singular vectors of R_{11} are estimated by subspace iteration techniques [**?**] and are stored in a matrix U. The K rightmost columns of U contain the $Kernel(R_{11}^T)$ wized

$N - L \times K$. Then, Eq. 4.27 becomes

$$Kernel(C^T) = Q \begin{pmatrix} Kernel(R_{11}^T) & 0 \\ 0 & I_{L\times L} \end{pmatrix} \tag{4.28}$$

In order to compute the $Kernel(C)$, the same process must be apllied on C^T which was the target in order to use the parameterization of this Chapter.

4.2.5 The Optimization Process

The optimization loop starts by importing the BRep of a CAD model from a standard CAD file. After performing the necessary triangulation with which, the boundary meshed is generated and/or mapped on the BRep surfaces, displaceable and non-displaceable patches are defined on the model as well as the required continuity between them. Using this information, the process shown in subsections 4.2.1-4.2.3 is followed which leads to the computation of matrices B and C. Then, the rank revealing QR decomposition described in sub-section 4.2.4 is used to compute the singular values and vectors of matrix C. $Kernel(C)$ is then computed through Eq. 4.12. The nodes of the boundary mesh are projected onto the BRep by using point inversion [**?**] which is further described in Appendix C. This projection is used to identify the surface on which the node belongs and also compute its parametric coordinates on that surface. Using Eq. 4.25, $\delta x_k/\delta b_n$ is computed for every node of the boundary mesh.

At each optimization cycle, the primal system of equations and their adjoint counterparts are solved to provide the sensitivity derivatives $\delta J/\delta b_n$. The CAD model is updated through the NURBS control points and Eq. 4.15.

The NURBS surfaces and curves were implemented within OpenFOAM along with the primal and adjoint equations (subsection [ref]). All matrices and vectors described in subsections 4.2.1-4.2.4 were handled using the Eigen [**?**] open-source library. The QR decomposition along with the null space basis computations were done using the SparseQR module within Eigen and the SuiteSparse [**?**] module.

4.3 Applications

The effectiveness of the proposed parameterization in maintaining the continuity constraints while minimizing an objective function is demonstrated in two aerodynamic shape optimization problems. The first case is a 3D duct with a middle S-section and the aim is to minimize the total pressure losses between the inlet and the outlet. The second case is the tail surface of a benchmark car model, in which the aim is to minimize the drag coefficient. The last case corresponds to Case 4

of the CFD optimization benchmark of the 11th ASMO UK/ISSMO/NOED2016: International Conference on Numerical Optimisation Methods for Engineering Design (see http://www.asmo-uk.com/11th_asmo_uk_conference/html/menu_page.html).

4.3.1 Optimization of the Mid-Section of a 3D Duct

The first case shown, is that of a duct which corresponds to a part of a cooling system used in automotive applications. The geometry consists of straight inlet and outlet ducts which have different cross-sections and different locations and orientations in the 3D space. To connect the inlet and the outlet, an S-section is designed (Fig. 4.5) with 24 NURBS patches. The optimization aims at minimizing the total pressure losses J_{Pt} (Eq. 2.19) by perturbing the shape of just the S-section. C_0 and C_1 continuity is imposed between all patches that are allowed to be displaced and all their neighbours.

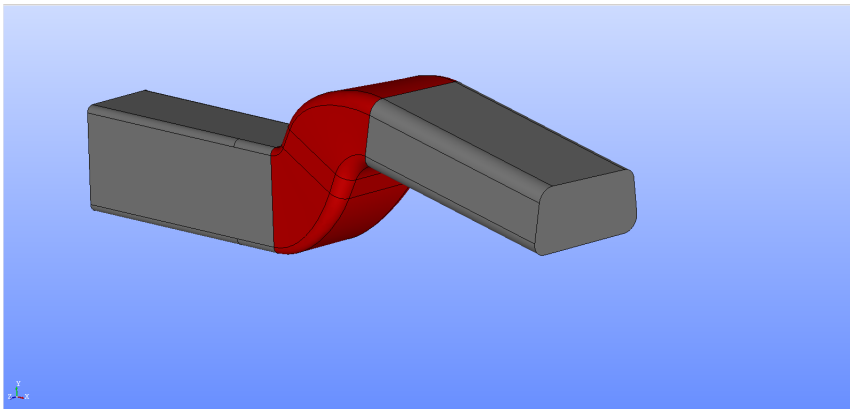


Figure 4.5: S-Section Duct: The CAD geometry of the duct (as loaded from a STEP file) with the displaceable middle S-section in red shading.

The computational mesh consists of approximately 73K hexahedral cells. The total number of control points of the displaceable patches is 1594 and these are allowed to move in all directions. The flow through the duct is laminar ($Re \approx 400$). The inlet velocity is $0.1m/s$ and the outlet has a zero pressure condition imposed on it. The flow streamlines for the initial geometry can be seen in Fig. 4.6. The sensitivity map on the initial geometry can be seen in Fig. 4.7 and the mesh displacement after the first update of $\vec{b}$ in Fig. 4.8.

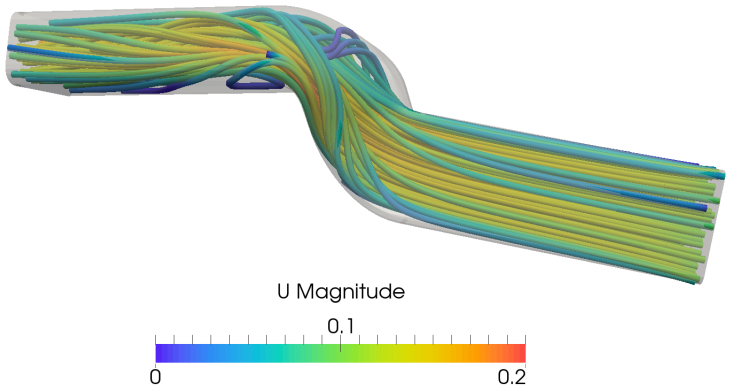


Figure 4.6: S-Section Duct: Computed flow streamlines produced in the initial geometry. Color coding indicates the velocity magnitude field.

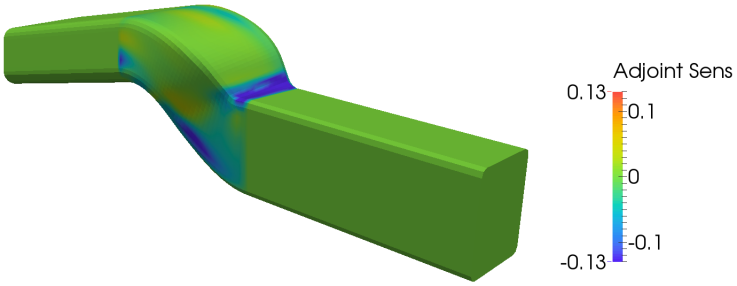


Figure 4.7: S-Section Duct: The sensitivity map plotted on the initial geometry of the duct (i.e. the derivative of J_{Pt} w.r.t. the normal displacement of the design wall nodes). Positive and negative sensitivities indicate that the geometry must be pushed in and pulled out, respectively, in order to reduce the objective function.

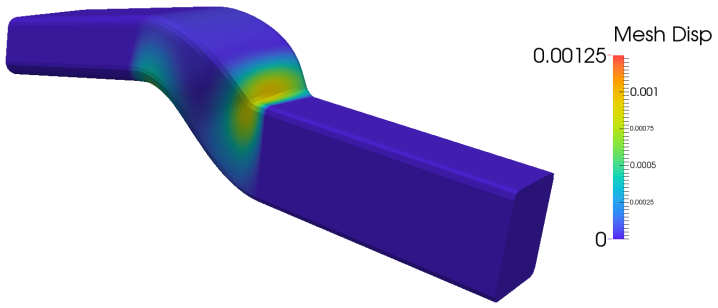


Figure 4.8: S-Section Duct: Normal mesh displacement after the first update of $\vec{b}$.

Using the proposed parameterization, with steepest descent, the objective function has reduced by 13.6% after 20 optimization cycles (Fig. 4.9).

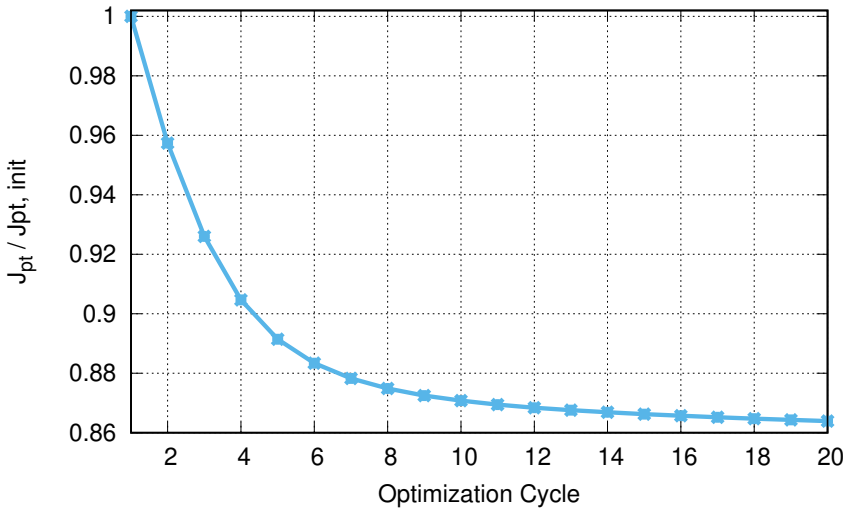


Figure 4.9: S-Section Duct: The convergence history of the optimization.

The resulting shape is easily exported in standardized STEP format which is compatible with any CAD package (commercial or not). The CAD model as well as comparisons with the initial can be seen in Fig. 4.10, 4.11. The resulting CAD shape has no holes where the trimming edges are located and is smooth along those edges. The shape can be exported as any standard CAD file and can be further manipulated in a CAD package for purposes of industrial design. In this case, the shape is exported as a STEP file. The comparisons shown below are between the CAD models of two STEP files - the one with the initial and the one with the final shape of the duct.

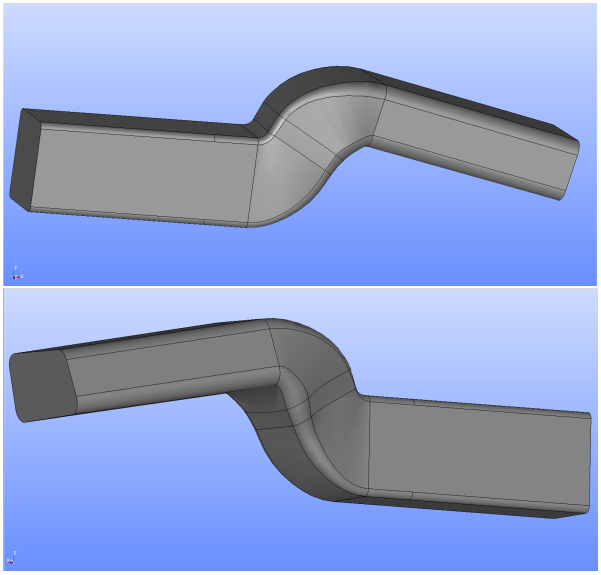


Figure 4.10: S-Section Duct: The CAD model of the final shape of the duct after 20 optimization cycles. The rendering is that of a STEP file which is loaded in an external CAD package.

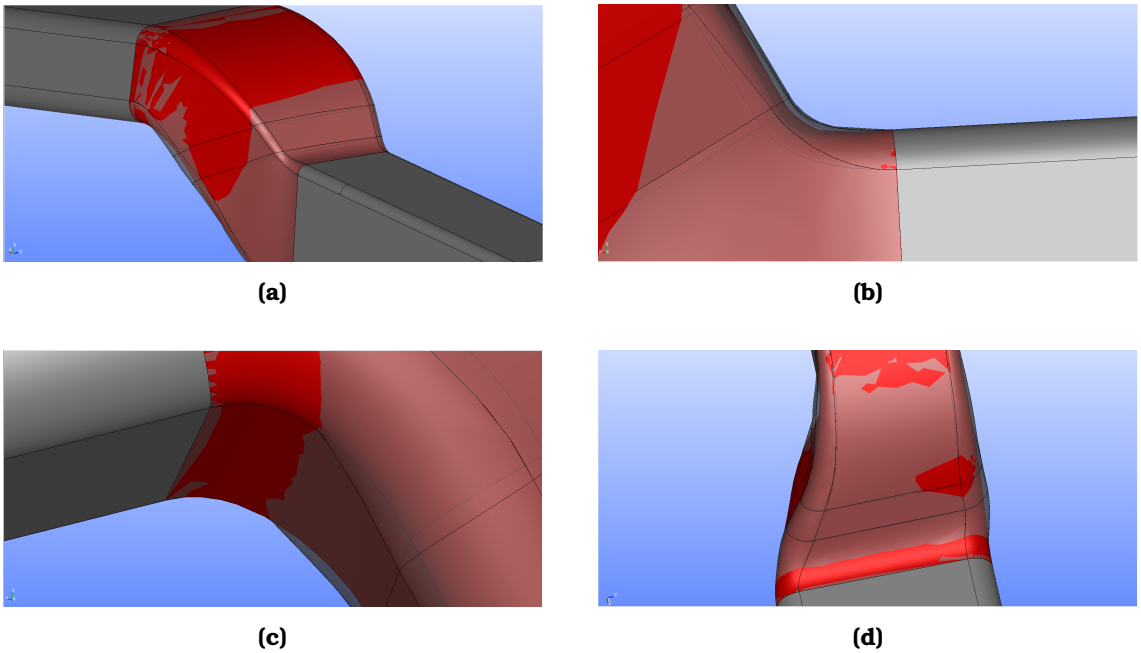


Figure 4.11: S-Section Duct: Comparison of the initial shape (red shading) and the final shape (transparent grey) of the S-section of the duct. Top-Left: Overall perspective of the comparison. Top-Right: Close-up view of the patches connecting the inlet to the S-section. Bottom-Left: Close-up view of the patches connecting the outlet to the S-section. Bottom-Right: Perspective view from under the S-section which best portrays the lateral deformations.

To further showcase the importance of the imposition of C_0 and C_1 constraints on the model, a single unconstrained shape update is performed with the shape sensitivities of the first optimization cycle. The results shown in Fig. 4.12 demonstrate two issues. Firstly, the discontinuities along the stitches between displaceable and non-displaceable patches and secondly, the multiple holes created between neighbouring trimmed patches. Both of these issues are handled effectively by the proposed method.

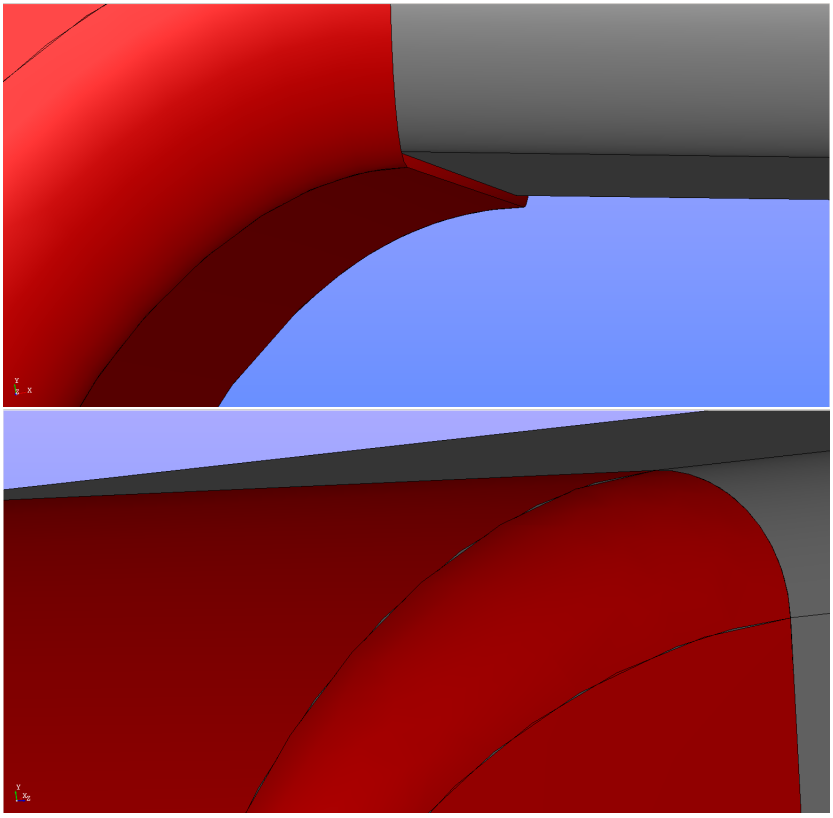


Figure 4.12: S-Section Duct: The updated shape without imposing C_0 and C_1 constraints. Top: Discontinuity between moving (red) and non-moving (grey) patches. Bottom: Holes between two moving trimmed patches.

The S-Section 3D duct is a well studied benchmark case used in several optimization runs coupled with various parameterizations. Therefore, this opportunity is taken to compare the proposed parameterization with others on the same geometry.

Firstly, a contribution that employs a CAD-based parameterization is chosen as proposed by Agarwal et al. [**?**], that includes design velocities and a study of parametric effectiveness. The case as well as the flow conditions and the Reynolds number are identical. Furthermore, the aerodynamic optimization is performed by means of the continuous adjoint technique. The parameterization is based

on feature modelling which means that the shape derivatives must be computed through finite differences. The parametric effectiveness study identifies the most effective design variables and uses them with priority. This approach is purely CAD-based and allows for an in-depth manipulation of the optimal shape. However, it is strictly tied to the originating CAD package. Furthermore, the inclusion of the feature tree inside the optimization loop can over-constrain the optimization. The optimization achieved a reduction of 10.72% in total pressure losses.

Secondly, a contribution that employs a CAD-free parameterization is chosen. Alexias et al. [**?**] optimized the S-Section duct by applying a node-based parameterization. The continuous adjoint method was employed to compute the gradient of the objective function and the same boundary conditions were employed as in this work. Implicit smoothing was used to even the transition from displaceable to non-displaceable regions of the mesh. This purely CAD-free approach allows for the richest design space which can consequently result in a greater minimization of the objective function. However, the link to CAD is severed which means that the final shape cannot be manipulated in a CAD framework. Re-establishing that link requires mesh-to-CAD procedures. The optimization achieved a reduction of 17.10% in power losses.

The proposed method achieves a middle ground solution by retaining a connection to CAD by means of the BRep while not being connected to the feature tree of any particular CAD package. This leads to a better optimization performance than a pure CAD-based method. At the same time, the optimization performance is worst than that of a pure CAD-free method but the connection to CAD allows for further manipulation of the optimal shape.

4.3.2 Optimization of the DrivAer Car Model

The second case chosen for testing the effectiveness of the parameterization is that of the DrivAer concept car (Fig. 5.19a). The configuration used in this study is the fast-back car model with a smooth underbody, with both mirrors and stationary wheels. The target is to minimize the drag coefficient (J_{c_D}) of the whole car, by modifying parts of the tail of the car. The study is performed on the half of the car by employing symmetry conditions. The CFD mesh consists of approximately 5.3M hexahedra with an average $y^+ = 75$ of the barycenters of the first cells off the wall and the Spalart-Allmaras turbulence model is used (Eq. 2.3).

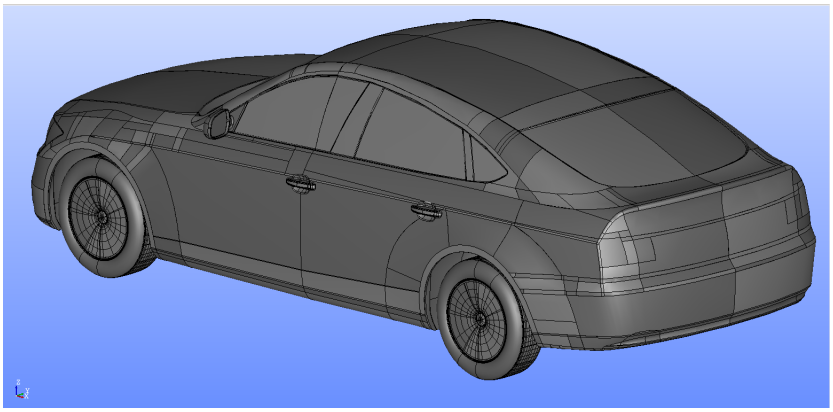


Figure 4.13: DrivAer: Computed pressure field on the surface of the car.

The model is subjected to a far-field longitudinal velocity of $38.89m/s$, with a Reynolds number of $Re \approx 2.6 \times 10^6$ (based on the car width). The proposed parameterization is applied to the rear end of the trunk lid, its sides and the back windshield (Fig. 4.14). The 62 selected trimmed NURBS surfaces consist of 5580 control points. C_0 and C_1 continuity is imposed between all displaceable-displaceable and non-displaceable-displaceable patch pairs.

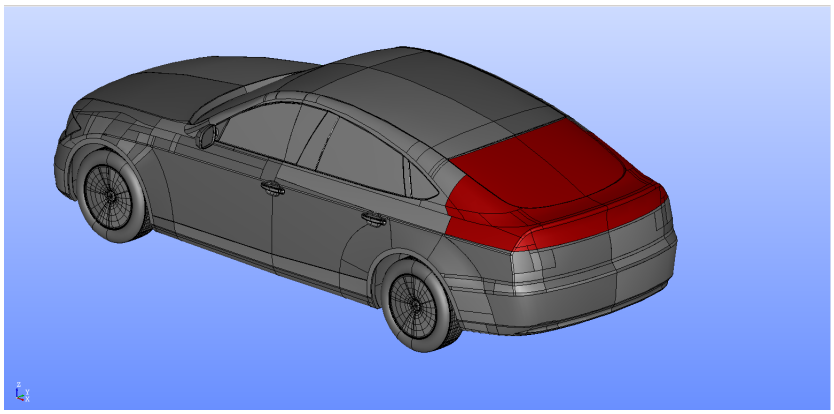


Figure 4.14: DrivAer: The CAD model of the car, as loaded directly from a STEP file. The displaceable trimmed faces are shown in red shading, while the non-displaceable ones are shown in grey.

The sensitivity map on the initial geometry can be seen in Fig. 4.15 and the mesh displacement after the first update of $\vec{b}$ along the X and Z axes (where the displacement is the most dominant) can be seen in Fig. 4.16.

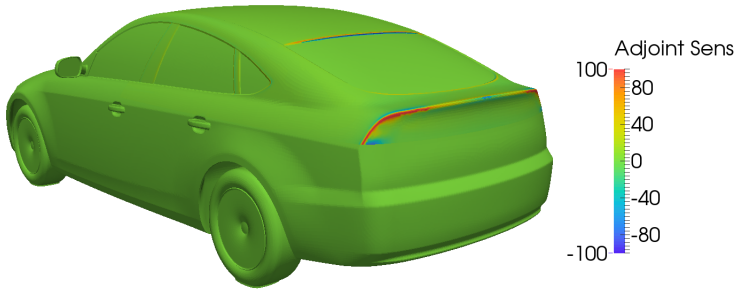


Figure 4.15: DrivAer: The sensitivity map plotted on the initial geometry of the car (i.e. the derivative of J_{C_D} w.r.t. the normal displacement of the design wall nodes). Positive and negative sensitivities indicate that the body surface must be pushed in and pulled out respectively, in order to reduce the objective function.

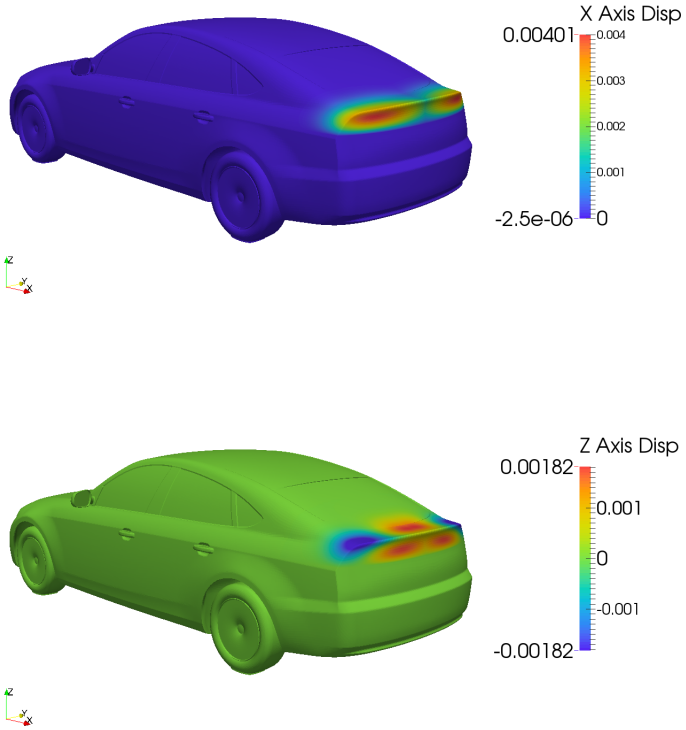


Figure 4.16: DrivAer: The mesh displacement generated after the first optimization cycle along the X axis (top) and the Z axis (bottom).

Using the proposed parameterization, with steepest descent, the objective function has reduced by almost 1.3% after 24 optimization cycles (Fig. 4.17).

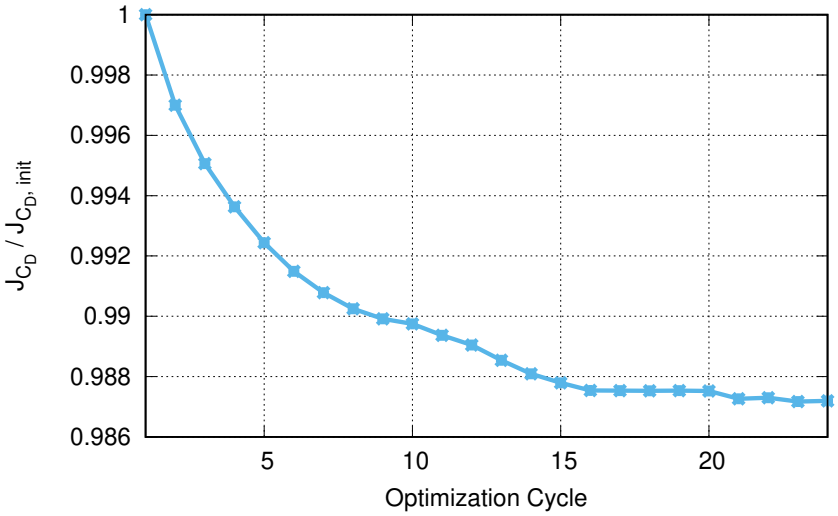


Figure 4.17: DrivAer: The convergence history of the optimization.

Similarly to the first case, the proposed parameterization allows for an easy export in CAD compatible format and further manipulation in a CAD environment. The CAD model as well as comparisons with the initial, can be seen in Fig. 4.18, 4.19. The resulting CAD shape is smooth and continuous and this makes the export as a STEP file possible.

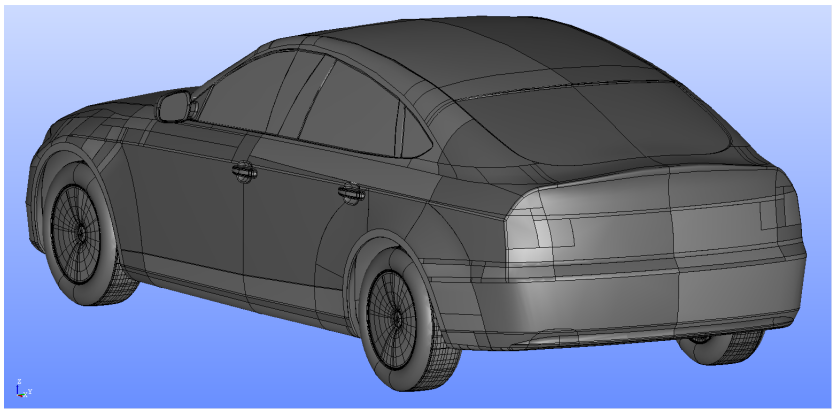


Figure 4.18: DrivAer: The CAD model of the final shape of the car after 24 optimization cycles.

Similarly to the previous case, the first update of the CAD model is shown without imposing any constraint. The resulting shape can be seen in Fig. 4.20 where holes in the geometry are visible. The proposed method overcomes this issue by producing watertight and smooth geometries.

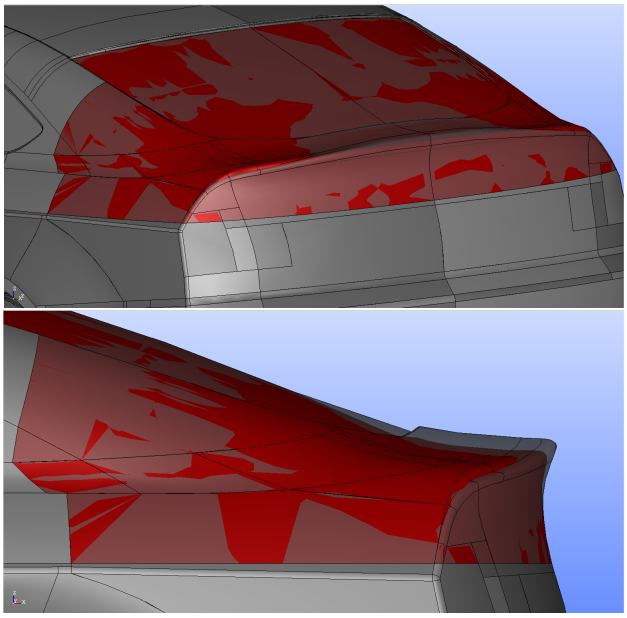


Figure 4.19: DrivAer: Comparison of the initial shape (red shading) and the final shape (transparent grey) of the displaceable part. Top: An overall perspective of the comparison. Bottom: Side view which best portrays the displacement of the trunk lid.

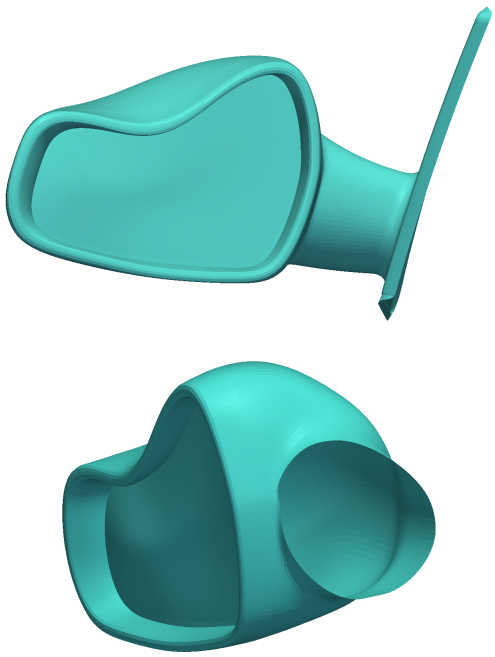


Figure 4.20: DrivAer: The updated shape without imposing C_0 and C_1 continuity constraints. The separation of adjacent patches is visible. Top: Top view of the displaceable patches (red shading). Bottom: a back view of the displaceable patches.

Similarly to the S-Section duct case, the DrivAer concept car is also a very well documented case. Thus, the proposed method can be compared with different parameterization methods.

Firstly, a lattice based approach is examined. Liatsikouras et al. [**?**] performed optimization of the rear end of the trunk lid of the car by means of a volumetric b-splines lattice. The link to CAD is not retained which means that the optimal shape cannot be manipulated in a CAD framework. The car was subjected to a flow under the same boundary conditions as in this work and the optimization achieved a 5% drop in the drag coefficient.

Finally, a pure CAD-free approach is examined. Alexias et al. [**?**] performed node-based optimization on the DrivAer model. The parameterized region of the car was the rear end of the trunk lid and the boundary conditions were the same as in this work. A 2.1% reduction was achieved in the drag coefficient.

As mentioned in the previous case, the proposed method overcomes the set-backs of CAD-free optimization by including the BRep in the optimization loop.

4.4 Remarks

In this chapter, a method to perform shape optimization using standard CAD geometries (BRep) was shown. The developed algorithm accurately imposes C_0 and C_1 continuity constraints between the multiple NURBS patches in the BRep of a model. The cases shown range from academic to industrial-like to fully test the method's capabilities. This method proved reliable in keeping the link to the CAD geometry and using that geometry for optimization. The time required for the generation of matrix C and the computation of $Kernel(C)$ through QR factorization is negligible in comparison to the time required to solve the primal and adjoint problems as it must be done once at the beginning of the optimization loop. The time required for the update of the CAD shape and the export of the STEP files is also very small.

The presented algorithm utilizes standard CAD files and by doing so, allows for the cooperation of continuous (or even discrete) adjoint with any CAD package (commercial or not). After the computation of the Null Space Basis, the shape derivatives are computed in a direct way which makes the algorithm fast and automated. Furthermore, return to CAD is not required as the CAD is updated at each optimization cycle and it can be ultimately exported in a standard CAD file.

Chapter 5

Constraining the NURBS-based Adjoint Optimization

In this chapter, algorithms for imposing geometric constraints during shape deformations in NURBS-based optimizations are presented. Firstly, the geometry is imported (triangulated and handled) as shown in Chap. 3. Secondly, the constraints are imposed in a way that any number of design variables can be handled.

There are three constraints that are developed and shown in this chapter. Curvature, bounding surface and volume constraints are investigated. The first two are node-based constraints, i.e. constraints that must be applied on many points of a NURBS surface and the last one is surface-based i.e. it is computed based on the position of the NURBS patches.

Node-based constraints would require the satisfaction of as many constraints as the nodes of the boundary mesh on the surface of the constrained NURBS patches. For instance, the requirement that a surface retains a curvature lower than a threshold must be applied at all nodes of that surface. The same applies for the bounding surface constraints which impose a distance constraint from a NURBS patch to another given surface. For such constraints, which can very likely outnumber the number of design variables, a special treatment must be introduced as conventional constrained optimization algorithms cannot handle them.

Surface-based constraints are imposed on a defined geometric shape as a whole. Thus, they do not require their imposition on multiple nodes and the relation between the constraints and the NURBS design variables must be computed and differentiated.

5.1 Literature Survey of Constraint Imposition

In industrial applications, the necessity for efficiently imposing geometric constraints is paramount. A family of geometric constraints with high practical significance are those which do not allow the shape to be designed to penetrate user-defined bounding surfaces (open or closed). These are also known as bounding constraints and play a crucial role in industrial design since most components must be designed within a given bounding space. For instance, in Heat, Ventilation and Air-Conditioning (HVAC) automotive applications, vents and cooling channels must be designed within enclosed spaces with various obstacles bounding their deformations or displacements during optimization. Another example is that of aircraft wings with enclosed fuel tanks (wet wings); minimizing the drag of the wing may reduce its thickness resulting to a wing surface that penetrates the tank. The user-defined set of bounding surfaces gives rise to a bounded or semi-bounded space of feasible solutions. Working with bounded spaces, the terms "packaging" or "enclosure" constraints can be used, in the sense that the feasible sub-space is a topological sphere. To impose these constraints, [**?**] used gradient projection [**?**] along with a vertex morphing technique [**?**] to perform shape changes. The constraints were formulated based on signed distances [**?**] computed for every surface grid node. The gradient of the objective function was projected onto the Null Space [**?**] of the Jacobian of the constraints (the matrix including the first derivatives of the constraints w.r.t. each design variable) in order to find an update that minimizes the objective function without violating the constraints. [**?**] handled bounding constraints with two metrics: (a) the length of the intersection curve(s) between the shape to be designed and the bounding surfaces and (b) the aggregated (by means of a Kreisselmeier-Steinhauser function [**?**]) minimum distances between the two shapes. [**?**] performed shape optimizations of constrained duct geometries by explicitly imposing no-displacement conditions on shape nodes that tend to violate the no-intersection criteria. [**?**] presented a morphing algorithm in the context of layout problems dealing with the placement of solid components in 3D space by avoiding to penetrate each other. The method used signed distance fields to identify constraint violation and retracted each violating node onto the surface of the constraint objects. Similar layout problems are addressed by [**?**, **?**]. [**?**] use a level set function to impose no-intersection constraints among a number of objects, in the form of a unified integral facilitating the use of gradient-based methods and allowing extension to other geometric constraints [**?**].

Geometric constraints, other than bounding ones, have been addressed in a similar node-wise manner. For instance, [**?**] presented a method to control the curvature of the boundaries of 2D domains during the optimization loop. Node-based parameterization of the boundaries was used and curvature was controlled at certain user-defined nodes. The Augmented Lagrangian Method (ALM) was

used to satisfy the curvature constraints, the values of which were computed after locally parameterizing the boundaries using three approaches: quadratic polynomials, NURBS curves and generalized arc-length. Thickness has also been controlled in a similar manner; [**?**] proposed a method to constrain shape thickness extrema by using the level set [**?**] method to compute them and the ALM to impose them. Finally, using the notion of signed distance, which will be addressed in this chapter, [**?**] optimized thickness by constraining stress during shell fabrication.

5.2 The Proposed Method

In this chapter, the mentioned constraints are imposed during aerodynamic shape optimization. Flow problems governed by the Navier–Stokes equations for incompressible fluids are tackled and the gradient of the objective function J w.r.t. the design variables controlling the shape to be designed is computed using the continuous adjoint method.

For the bounding constraints, the bounding surfaces are defined in discrete (triangulated) form. The constraint values from all boundary mesh nodes are aggregated into a single constraint function which is used to avoid intersection of the surface to be designed with the bounding one(s). This method's efficiency must not be jeopardized by the possibly high number of constraints, especially when the parameterization is not node-based (which could lead to a much smaller number of design variables). The constraint function identifies whether the design and bounding surfaces intersect; this is done in a node-wise manner. For each node on the shape, a signed distance to the bounding surface is computed. This distance is negative for nodes lying in the feasible space and positive otherwise. An inequality constraint is, therefore, formulated for each node, requiring that the corresponding signed distance remains non-positive. Hence, the number of geometric constraints is equal to the number of nodes on the shape and this may become problematic, depending on the number of design variables and the method used to update them.

For the curvature constraints, the same method is applied. During NURBS-based optimization, the geometry of the patches can sometimes be warped (meaning that the control point grid is not uniformly distributed along the surface). This can lead to a finer control point grid at certain areas and a coarser elsewhere. In areas with a high concentration of control points, the perturbed surface can become wrinkly (with high curvature regions). For this reason, for each and every node of the boundary mesh, a maximum curvature constraint must be applied. Similar to the bounding surface constraints, this can lead to numerical difficulties. For instance, working with the Sequential Quadratic Programming (SQP) [**?**] and the active set method [**?**], the constraints seen by the method are only the active (violated) ones. Depending on the case, the number of active constraints can

become larger than the number of design variables, especially when the shape is parameterized with a relatively small number of design variables. This is a degenerate problem [**?**], as the systems in which the standard SQP and gradient projection methods result, become singular. In such a case, Augmented Lagrangian or penalty methods must preferably be used. On the other hand, these methods often misbehave for a high number of constraints, since a good enough initialization of the penalty factors or the Lagrange multipliers can hardly be obtained. Thus, it is desirable to avoid handling such a large number of constraints and a method to reduce them is proposed.

The main drive here is to efficiently solve optimization problems with bounding constraints, even if the number of the latter is large and the design variables number is relatively small (i.e. the parameterization is not node-based), which, as stated before, can become numerically unstable. The proposed method transforms a number of inequality constraints to a single equality constraint by summing all nodal constraint values (filtered by a penalty function) for the surface to be designed. This step allows handling a potentially very large number of constraints with a single Lagrange multiplier and/or penalty value. This single constraint is, then, differentiated w.r.t. the design variables. In order to showcase the benefits of the proposed constraint aggregation, a study is performed in one of the cases. Initially, constrained optimization is performed using the ALM method and node-wise constraints with slack variables. The optimization is repeated using the ALM and the proposed single constraint and, finally, using SQP. The design variables are the control points of both NURBS and Volumetric B-splines, though any other parameterization scheme could have been used.

Apart from these two types of constraints, a minimum allowed volume constraint is studied. The volume is computed and differentiated w.r.t. the NURBS control points.

5.3 Constraint Imposition

The target is to solve an aerodynamic shape optimization problem, the solution of which minimizes a flow related objective function and satisfied a number of constraints. The optimization is gradient-based and the derivatives of the objective function are computed using continuous adjoint.

5.3.1 The Bounding Surface Constraint

In what follows, two shapes are available in each optimization cycle, i.e. the current solution to the optimization problem and the bounding surface that defines the boundary between two spaces: that in which the shape to be designed is allowed to reside (feasible space) and the infeasible space (Fig. 5.1).

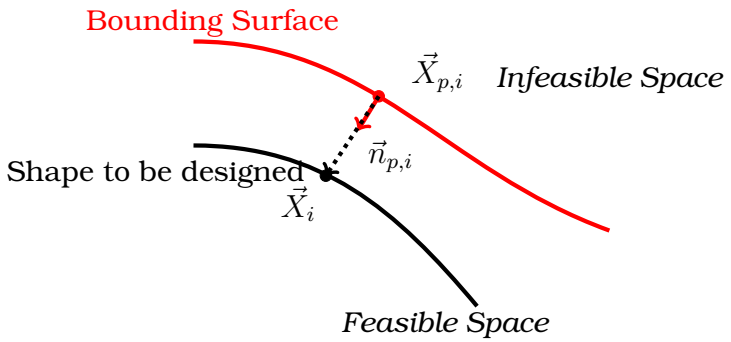


Figure 5.1: Bounding surface and the shape to be designed, sketched as 2D curves. $\vec{X}_i$ denotes the coordinates of the $i-th$ node on the shape to be designed. $\vec{X}_{p,i}$ and $\vec{n}_{p,i}$ denote the projected point of $\vec{X}_i$ to the bounding surface and the bounding surface's normal at that point, respectively. The normal vector is pointing toward the feasible space.

In order to impose the bounding surface constraints, the first course of action is to compute the closest projection of every node of the current solution (shape) onto the bounding surface. Since this projection is repeated in each optimization cycle, it should be fast. Initially, the bounding surface is triangulated and a dynamic octree structure is built around it, in order to facilitate fast and reliable point-to-triangle search routines. Therefore, the discrete geometry is stored in STL format and each triangle is classified in an octree leaf. At the beginning of each optimization cycle, an octree search is made for each design surface node $\vec{X}_i = [x_{i,1}, x_{i,2}, x_{i,3}]^T$. This returns a list of triangles from the STL lying within a user-defined fixed radius from $\vec{X}_i$. For each of these triangles, a projection algorithm is performed. This algorithm [**?**] projects $\vec{X}_i$ onto the plane that each triangle belongs to and, then, computes the barycentric coordinates of the projection (denoted as $\vec{X}_i^*$). In case $\vec{X}_i^*$ lies inside the triangle, this is considered to be a valid projection. Let $\vec{T}_1, \vec{T}_2$ and $\vec{T}_3$ be the position vectors of the three vertices of such a triangle, then the barycentric coordinates (α, β, γ) of the projection are

$$\gamma = \frac{(\vec{u} \times \vec{w}) \cdot \vec{n}}{|\vec{n}|^2}, \quad \beta = \frac{(\vec{w} \times \vec{v}) \cdot \vec{n}}{|\vec{n}|^2}, \quad \alpha = 1 - \beta - \gamma \tag{5.1}$$

where $\vec{u} = \vec{T}_2 - \vec{T}_1, \vec{v} = \vec{T}_3 - \vec{T}_1, \vec{w} = \vec{X}_i - \vec{T}_1, \vec{n} = \vec{u} \times \vec{v}$. The projection is valid if $0 \leq \alpha, \beta, \gamma \leq 1$ and, in such a case, $\vec{X}_i^* = \alpha\vec{T}_1 + \beta\vec{T}_2 + \gamma\vec{T}_3$. The above procedure is repeated for all the triangles returned from the octree list. From all the valid projection points $\vec{X}_i^*$, the nearest to $\vec{X}_i$ is its projection $(\vec{X}_{p,i})$ to the bounding surface. The normal vector at $\vec{X}_{p,i}$ pointing towards the feasible space is denoted as $\vec{n}_{p,i}$. In cases with sharp corners and $n_{p,i}$ is chosen to be the one that forms the

smallest angle with the normal to the boundary mesh at $\vec{X}_i$. The signed distance

$$g_i(\vec{b}) = -\vec{n}_{p,i} \cdot (\vec{X}_i(\vec{b}) - \vec{X}_{p,i}) \tag{5.2}$$

is then computed and this determines if $\vec{X}_i(\vec{b})$ remains in the feasible space w.r.t. the bounding surface. $\vec{b}$ at this stage could be arbitrary. If $g_i(\vec{b}) \leq 0$, then $\vec{X}_i(\vec{b})$ lies inside the feasible space (Fig. 5.1). If this condition holds for all nodes, then (assuming that the discretization of the design surface is much finer than that of the bounding surface) the design surface will reside entirely in the acceptable space. The derivative of the signed distance function w.r.t. b_n is

$$\frac{\delta g_i(\vec{b})}{\delta b_n} = -\vec{n}_{p,i} \cdot \frac{\delta \vec{X}_i(\vec{b})}{\delta b_n} \tag{5.3}$$

and can be computed analytically based on the parameterization.

Let M be the number of nodes on the shape to be designed. Then, the constrained optimization problem is defined as follows:

$$\begin{gathered} \min\ J(\vec{b}),\ \vec{b} = [b_1, b_2, ..., b_N] \\ \text{subject to } g_i(\vec{b}) \leq 0,\ i = 1, 2, ..., M \\ c_j \leq 0,\ j = 1, 2, ..., K \end{gathered} \tag{5.4}$$

where c_j represents other geometric or flow constraints which could potentially exist in the same problem.

5.3.2 The Curvature Constraint

In the attempt to constrain the curvature field on a surface, a prerequisite is the selection of the curvature metric (among principal curvatures, mean or Gaussian curvatures or functions of all the above), which the constraint should be imposed to. Based on this, an inequality constraint must be imposed at all points on that surface. The curvature at each point of a surface can be computed using the coefficients of its first and second fundamental forms, defining tensors $\mathcal{F}_1$ and $\mathcal{F}_2$ respectively. At each point, the principal curvatures κ_1, κ_2 are computed after solving $det(\mathcal{F}_2 - \kappa\mathcal{F}_1) = 0$ for κ (Chap. 3). Through them, the mean and Gaussian curvatures

$$\kappa_{mean} = \frac{\kappa_1 + \kappa_2}{2}$$

$$\kappa_{Gauss} = \kappa_1 \kappa_2 \tag{5.5}$$

can be defined. As stated in Sec. 3.2.2.2, the principal curvatures represent the curvature values on perpendicular parametric directions and, therefore, by constraining them to have an absolute value less than a threshold, local "flatness" of the surface can be achieved. Therefore, the expression chosen to be constrained is

$$\kappa = \frac{1}{2}(\kappa_1^2 + \kappa_2^2) \leq \kappa_{thres} \tag{5.6}$$

where κ_{thres} is an upper bound value for κ. In order to compute and, then, differentiate Eq. 5.6 w.r.t. the design variables $\vec{b}$, its dependence upon the surface expression must firstly be computed. According to [**?**], Eq. 5.6 can further be rewritten as

$$\begin{aligned}\kappa &= \frac{1}{2}(\kappa_1^2 + \kappa_2^2) = \frac{1}{2}\left((\kappa_{mean} + C)^2 + (\kappa_{mean} - C)^2\right) = \kappa_{mean}^2 + C^2 \\ &= \left(\frac{GL - 2FM + EN}{2(EG - F^2)}\right)^2 + \left(\frac{L(EG - 2F^2) + 2EFM - E^2N}{2E(EG - F^2)}\right)^2 + \left(\frac{EM - FL}{E\sqrt{EG - F^2}}\right)^2\end{aligned} \tag{5.7}$$

where E, F, G and L, M, N are the coefficients of the first and second fundamental forms respectively. Using the chain rule, the derivative of κ w.r.t. a design variable b_n can be computed. $\delta\kappa/\delta b_n$ depends on the following quantities:

$$\begin{aligned}\frac{\delta E}{\delta b_n} &= 2\frac{\delta \vec{\sigma}_u}{\delta b_n} \cdot \vec{\sigma}_u \\ \frac{\delta G}{\delta b_n} &= 2\frac{\delta \vec{\sigma}_v}{\delta b_n} \cdot \vec{\sigma}_v \\ \frac{\delta F}{\delta b_n} &= \frac{\delta \vec{\sigma}_u}{\delta b_n} \cdot \vec{\sigma}_v + \vec{\sigma}_u \cdot \frac{\delta \vec{\sigma}_v}{\delta b_n} \\ \frac{\delta L}{\delta b_n} &= \frac{\delta \vec{\sigma}_{uu}}{\delta b_n} \cdot \vec{n} + \vec{\sigma}_{uu} \cdot \frac{\delta \vec{n}}{\delta b_n} \\ \frac{\delta M}{\delta b_n} &= \frac{\delta \vec{\sigma}_{uv}}{\delta b_n} \cdot \vec{n} + \vec{\sigma}_{uv} \cdot \frac{\delta \vec{n}}{\delta b_n} \\ \frac{\delta N}{\delta b_n} &= \frac{\delta \vec{\sigma}_{vv}}{\delta b_n} \cdot \vec{n} + \vec{\sigma}_{vv} \cdot \frac{\delta \vec{n}}{\delta b_n}\end{aligned}$$

which directly depend on the surface expressions. Again, if M is the number of nodes on the shape to be designed, then the constrained optimization problem can be defined as follows:

$$\begin{aligned} &\min\ J(\vec{b}),\ \vec{b} = [b_1, b_2, ..., b_N] \\ &\text{subject to } \kappa_i(\vec{b}) \leq \kappa_{thres},\ i = 1, 2, ..., M \\ &c_j \leq 0,\ j = 1, 2, ..., K \end{aligned} \tag{5.8}$$

where c_j represents other geometric or flow constraints which could potentially exist in the same problem.

5.3.3 Transforming Node-wise Constraints to a Single Equality Constraint

The large number of constraints $g_i(\vec{b})$ or $\kappa_i(\vec{b})$ in such a formulation can cause numerical problems if certain optimization methods are used. Constrained optimization algorithms, such as the SQP or gradient projection, assume a priori that the number of constraints is less than the number of design variables. Both need to compute and factor or invert matrices containing the Jacobian of the constraints. These matrices become rank-deficient if there are more equality and active inequality constraints than design variables and, therefore, the computation of a suitable update for $\vec{b}$ poses numerical difficulties. As a result, ALM methods are preferably used. However, it is not desirable to solve constrained optimization problems with a great number of constraints as this can cause severe convergence issues (see section 5.4.1). Thus, an alternative approach is pursued which can accommodate most optimization methods and lead to a faster convergence of the optimization problem. The constraint function values are summed up after passing through a penalty function which penalizes nodal constraint violations. The penalty function has the following form:

$$F_p(c_i) = \begin{cases} 0 & , c_i < c_m \\ \alpha(c_i - c_m)^4 + \beta(c_i - c_m)^3 & , c_m \leq c_i < c_s \\ c_i + (y_s - c_s) & , c_i \geq c_s \end{cases} \tag{5.9}$$

Terms involved in Eq. 5.9 can be seen in Table 5.1 and are chosen to achieve first and second-order continuity. c_m is the threshold value and c_s is the value above which the penalty function increases linearly. c_s is mainly related to the size of the case and the units of the constraint functions.

It is understandable that the values of c_s and c_m change depending on the upper threshold defined by the problem. For instance, for the bounding surface constraints, $c_m = 0$ and $F_p(g_i)$ is plotted in Fig. 5.2 for various c_s values.

Table 5.1: Quantities in eq. 5.9.

Variable	Value
c_m	Constraint upper bound
c_s	A value greater than c_m over which $F_p(c_i)$ increases linearly
α	$-\frac{1}{2(c_s-c_m)^3}$
β	$-2\alpha(c_s-c_m)$
y_s	$\alpha(c_s-c_m)^4+\beta(c_s-c_m)^3$

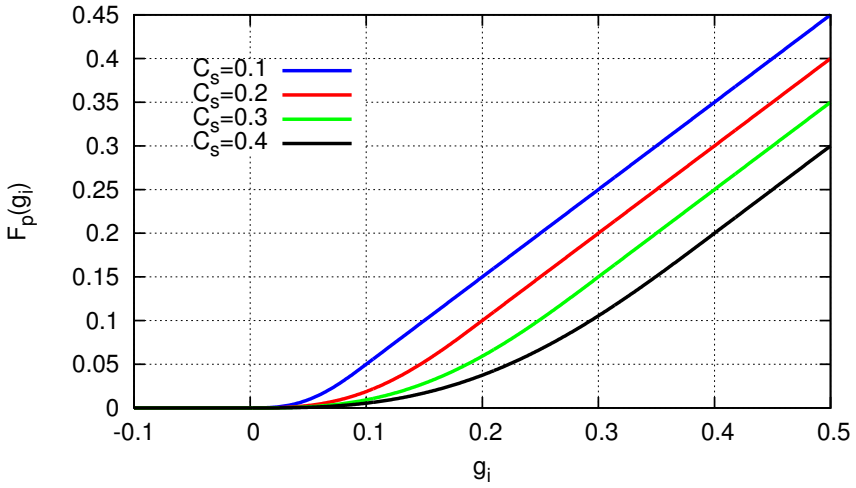


Figure 5.2: The penalty function for $c_m = 0$ and various values of c_s.

The penalty function returns positive values only for the violated constraints. The returned penalties are then weighted with the boundary face areas S_i at the respective nodes and are normalized with the total design surface area, in order to make the constraint value resolution–independent. The summation of all the penalty values over the design surface yields:

$$C_f(\vec{b}) = \frac{\sum_i^M F_p(c_i(\vec{b}))S_i}{\sum_i^M S_i} = \sum_i^M F_p(c_i(\vec{b}))w_i \tag{5.10}$$

The value of C_f for various boundary mesh resolutions for the case of section

5.4.1 is depicted in figure 5.3;

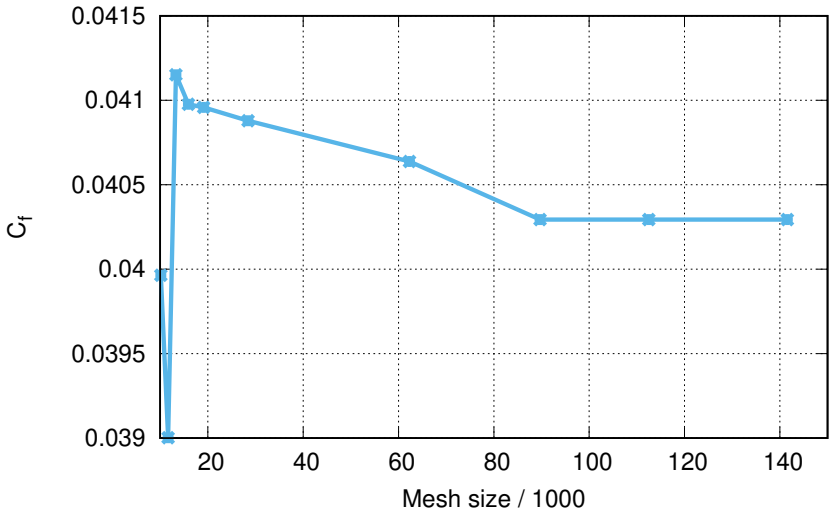


Figure 5.3: Double-Outlet Duct: C_f values plotted for various mesh sizes.

Nullifying Eq. 5.10 means that none of the nodes on the design shape violate its nodal constraints. To facilitate the convergence of the optimization problem, this condition can be transformed into an inequality constraint by relaxing it with a small positive value ϵ, i.e. imposing $C_f \leq \epsilon$. The derivative of C_f w.r.t. b_n is

$$\frac{\delta C_f}{\delta b_n} = \sum_i^M \left[\frac{\partial F_p(c_i(\vec{b}))}{\partial c_i(\vec{b})} \frac{\delta c_i(\vec{b})}{\delta b_n} w_i + F_p(c_i(\vec{b})) \frac{\delta w_i}{\delta b_n} \right] \tag{5.11}$$

The constrained optimization problem defined in Eqs. 5.4, 5.8, becomes

$$\begin{aligned} &\min J(\vec{b}),\ \vec{b} = [b_1, b_2, ..., b_N] \\ &\text{subject to } C_f = 0, \\ &\quad c_j \leq 0,\ j = 1, 2, ..., K \end{aligned} \tag{5.12}$$

A simple example is portrayed below, using the proposed formulation for the curvature constraints on a NURBS semi-cylinder. The radius of the semi-cylinder is $r = 0.5$ which leads to a uniform $\kappa_i = 2$ at all of its surface nodes. Eq. 5.12 is formulated for three different values of c_m and equation $C_f = 0$ is solved iteratively by selecting the control point positions. For all three cases, $c_s = c_m + 0.1$ has been set for consistency. Solving $C_f = 0$, leads the curvature metric of each surface node to become less than c_m. The resulting shapes as well as the convergence history of all three cases, are shown in Fig 5.4.

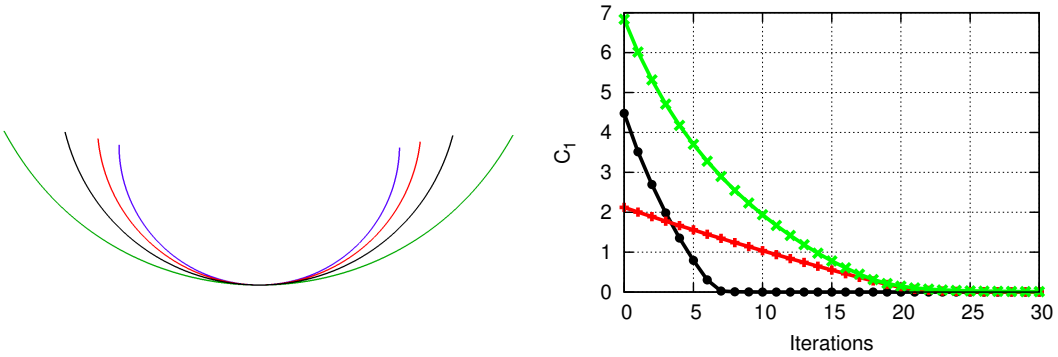


Figure 5.4: Left: The initial cross–section of a semi–cylindrical surface (blue) and the cross–sections resulting from the solution of $C_f = 0$ for $c_m = 1.5$ (red), $c_m = 1.0$ (black) and $c_m = 0.5$ (green). Right: The convergence history of the three iterative procedures leading to the results presented on the left. Curves for $c_m = 1.5$ (red), $c_m = 1.0$ (black) and $c_m = 0.5$ (green) are shown.

5.3.4 The Volume Constraint

For a watertight solid model, its volume can be computed using only the nodal coordinates of its boundary patches. Assume such a model and let V be its volume and S its total surface area. From the Gauss divergence theorem, it is known that the volume of a watertight shell of surface S can be computed using its boundary surfaces. It is obvious that the total surface of a solid is a union of all its trimmed patches $\vec{\sigma}_i$ and, therefore, the volume is given by

$$V = \frac{1}{3} \sum_i \int_u \int_v \vec{\sigma}_i(u,v) \cdot \vec{n}(u,v) \left\| \vec{\sigma}_u(u,v) \times \vec{\sigma}_v(u,v) \right\| dudv \tag{5.13}$$

The volume constraint is an inequality that enforces the volume to be lower (in case a model must fit inside an assembly) or greater (in case an assembly must fit inside a model) than a threshold. For the latter case, the constraint is expressed as

$$C_v = V - V_{min} \geq 0 \tag{5.14}$$

This constraint is handled using slack variables within the Augmented Lagrangian Method (ALM) algorithm [**?**].

Similarly to the flow and adjoint solvers, all the constraints shown here were implemented within OpenFOAM.

5.4 Applications

The imposition of the shown constraints is demonstrated in five problems. All of them stand for aerodynamic shape optimization problems, constrained either by the presence of bounding surfaces or with the surface curvature or the volume. The first case is a 2D duct and the aim is to minimize the total pressure losses between the inlet and the two outlets. The second case is a 3D U-Bend duct and, similarly to the first case, the aim is to minimize the total pressure losses. The third case is the shape optimization of the side mirror of a benchmark car model for minimum drag coefficient. The fourth case is a 3D automotive cooling duct and the aim is to minimize the total pressure losses between its inlet and outlet. The final case is an extruded NACA0012 airfoil and the aim is to minimize the drag coefficient. In the first three cases, the bounding surface constraints are imposed while, in the fourth and fifth, the curvature and volume constraints are tested respectively.

Cases 2 and 3 in this Chapter, correspond to Cases 1 and 4 of the CFD optimization benchmark conducted during the 11th ASMO UK/ISSMO/NOED2016 International Conference on Numerical Optimisation Methods for Engineering Design (see http://www.asmo-uk.com/11th_asmo_uk_conference/html/menu_page.html).

5.4.1 Optimization of a Double-Outlet Duct

This case was initially shown by Koch et al. (2017) [**?**], in an article dealing with topology optimization in fluid mechanics, without involving the bounding box constraints. The duct geometry is described by B-spline control points. The initial duct is the one shown in Fig. 5.5. The optimization aims at minimizing total pressure losses (Eq. 2.19) and the bounding box happens to cross the initial duct geometry. This bounding box consists of two narrow regions which restrict the flow passage and the duct must be redesigned in order to make it lie inside them. The constrained optimization starts from an infeasible solution w.r.t. the bounding box constraint. The three straight ducts which are linked directly to the inlet and the outlets cannot be deformed while the rest of the duct can.

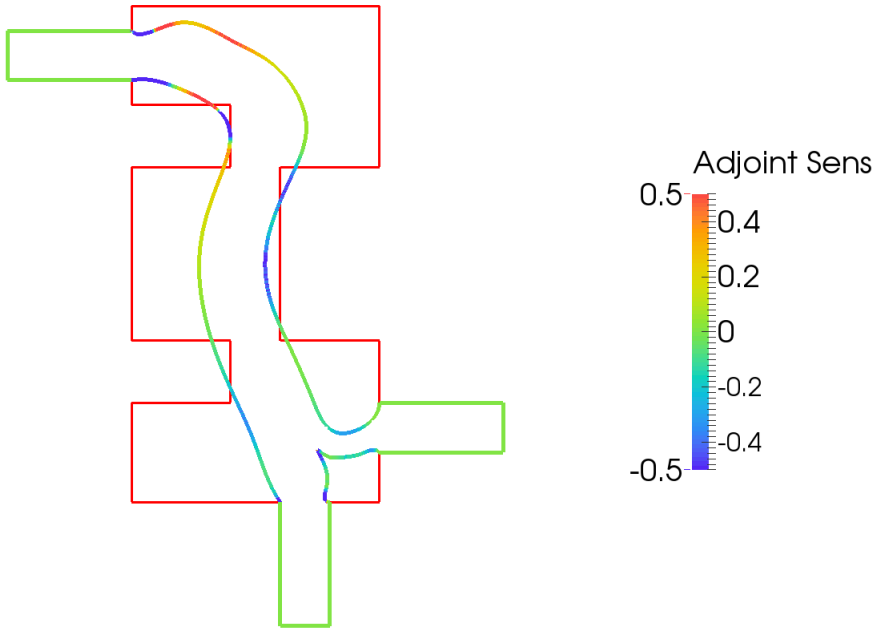


Figure 5.5: Double-Outlet Duct: The initial duct geometry colored based on the sensitivity map, i.e. the derivative of J_{Pt} w.r.t. the normal displacement of the design wall nodes, and the bounding surface in red. For the parts of the surface with negative / positive sensitivities, the duct walls must be pulled out / in to reduce the objective function value.

The computational domain is 2D (pseudo-3D with one cell depth) and consists of approximately 85K hexahedral cells. To parameterize the duct walls, multiple 2D curves extruded along the depth direction are used.

A total of 35 control points are used to parameterize the duct walls. All of them are allowed to move in both directions with the exception of the two first control points after each inlet and outlet wall. Therefore, the total number of free control points is 23, giving rise to $N = 46$ design variables. The flow is laminar ($Re = 200$).

Initially, a mesh independence study is performed in order to justify the resolution of the mesh. The primal equations are solved for various mesh sizes and the corresponding objective function (J_{pt}) values are plotted in Fig. 5.6. Based on this study, a mesh of approximately 85K cells appears to be adequate and this is used hereafter. On this mesh, the adjoint equations are solved to compute the sensitivity map of Fig. 5.5. It can be observed that the thickness of the duct cross-section should be increased along the largest part of the left design wall.

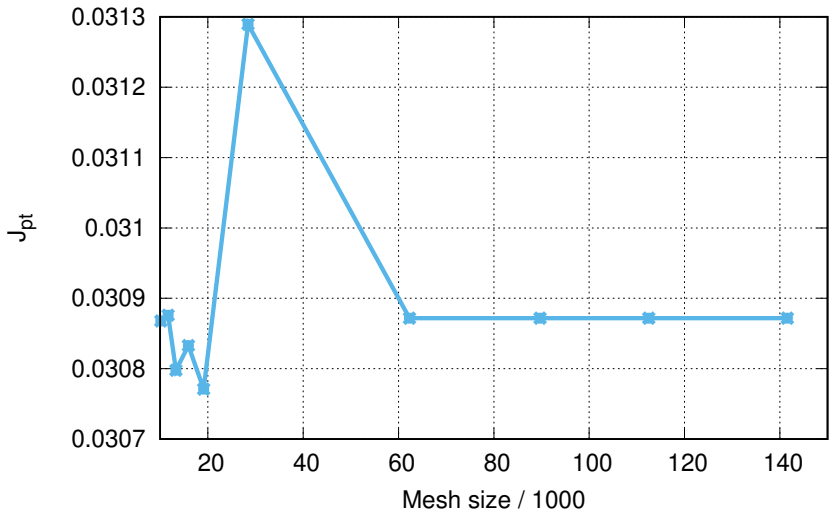


Figure 5.6: Double-Outlet Duct: J_{pt} values for different mesh sizes expressed by the number of cells.

The solution of the constrained optimization problem of the double-outlet duct is tried using three approaches. Firstly, the no-penetration constraints are imposed in a node-wise manner using the ALM method. The design patches of the duct consist of a total of 2618 boundary nodes, making it difficult to initialize λ_i and μ_i for such a high number of constraints. Many different initializations were tried, leading however to an impasse. At the start of the optimization, the ALM method tends to ignore the constraints, until the penalty values μ_i become sufficiently large. However, increasing μ_i makes the solution highly unstable, as λ_i values do not converge. Thus, computing the optimal set of Lagrange multipliers proved very challenging and this problem was turned into the development of an extremely sophisticated ALM method. However, even that would require a good enough initialization to work. Secondly, the same problem is solved by using the same ALM method as before, but with the single equality constraint formed for $c_m = 0$, $c_s = 0.001$. The constraint makes the duct shrink to move into the feasible space. The optimized solution gives an objective value which is by 21.5% greater than the initial. The optimization converges within 8 optimization cycles.

Finally, the same problem is solved using the SQP and the single constraint. The SQP computes a different local optimum which leads to a 9.6% increase in the objective function. The convergence of the objective function and the residual of the KKT equations are plotted in Fig. 5.7. This study showcased that the combination of the single, aggregated constraint and the SQP is the most efficient technique, in this case at least. Hence, this combination will also be utilized for the cases that follow.

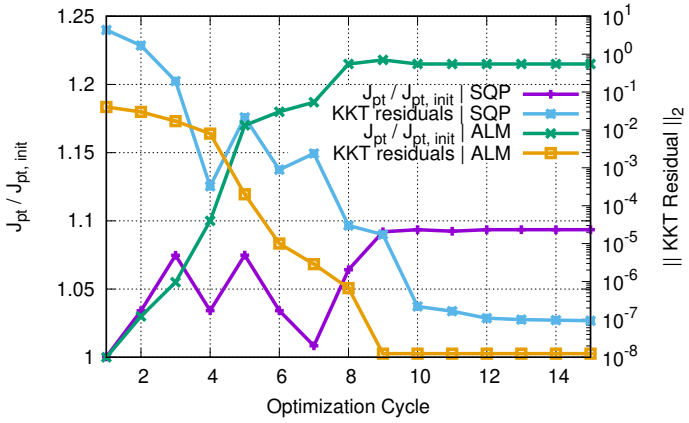


Figure 5.7: Double-Outlet Duct: Convergence history of the objective function and the residuals of the KKT equations for the constrained optimization using the ALM and the SQP. The objective function values are read on the left whereas the KKT residuals on the right. The quantities to which the graphs values refer are shown in the legend.

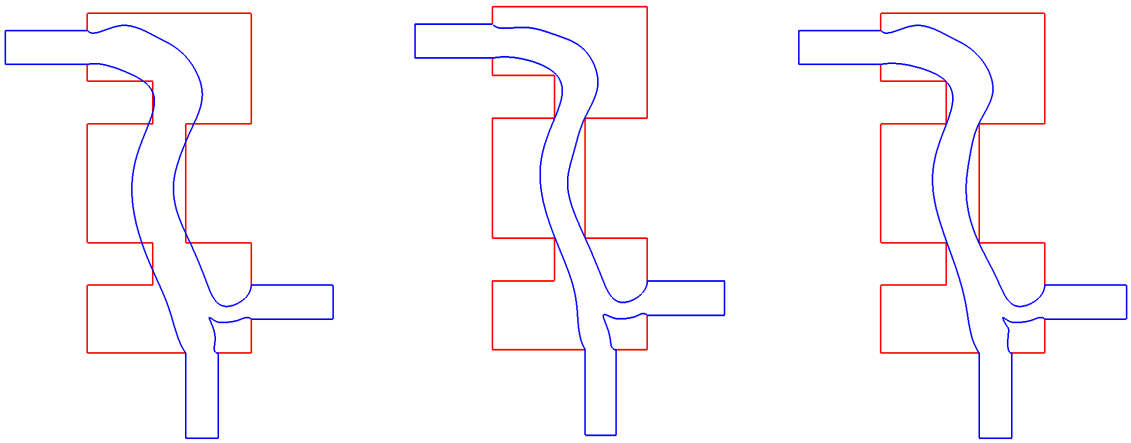


Figure 5.8: Double-Outlet Duct: Initial (left) and optimized shapes resulting from the constrained optimization with the ALM (center) and the SQP (right) methods.

Shrinking parts of the duct makes the fluid traverse the narrow passages at a higher speed compared to that of the initial geometry (Fig. 5.9). Thus, the optimized geometry has higher friction forces and higher total pressure losses.

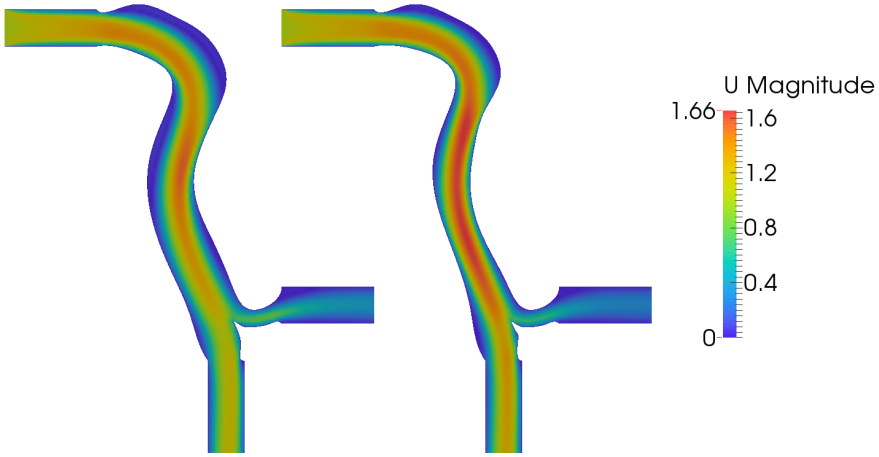


Figure 5.9: Double-Outlet Duct: Velocity magnitude iso-areas plotted on the initial (left) and the optimized via the SQP (right) geometries of the duct.

5.4.2 Optimization of a 3D U-Bend Duct

The second case is the optimization of a 3D U-Bend duct [**?**, **?**]. This geometry mimics part of the serpentine type internal cooling channel of turbine blades (Fig. 5.10a). The aim is to minimize the total pressure losses by deforming the U-turn; pure 3D deformations are allowed. The computational mesh has approximately 870K cells with an average $y^+ = 1.8$ of the first cell barycenters off the wall.

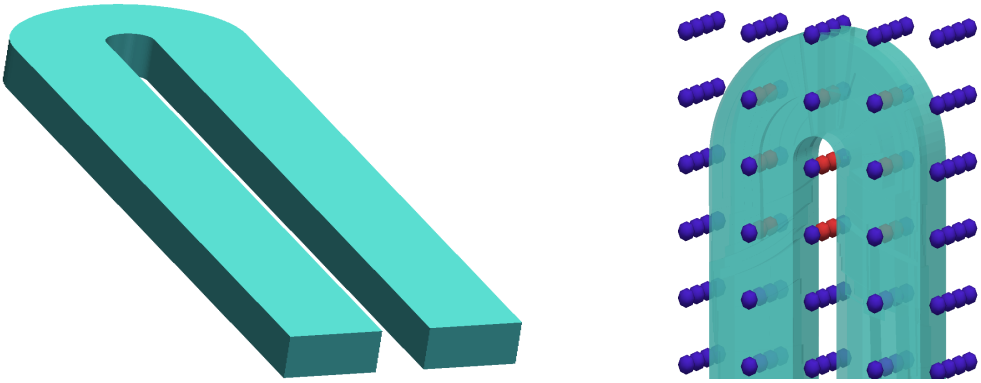


Figure 5.10: U-Bend Duct: Initial shape. Left: perspective view of the whole duct. Right: parameterized part of the duct along with the control points of the volumetric splines control box. Red control points are allowed to move while the blue are fixed.

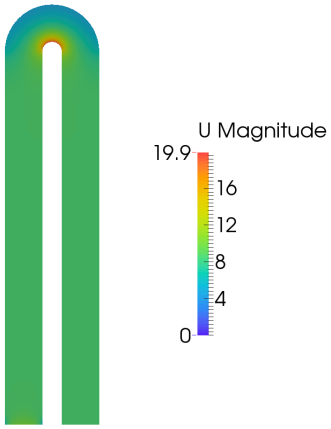


Figure 5.11: U-Bend Duct: Velocity field plotted on the initial duct geometry.

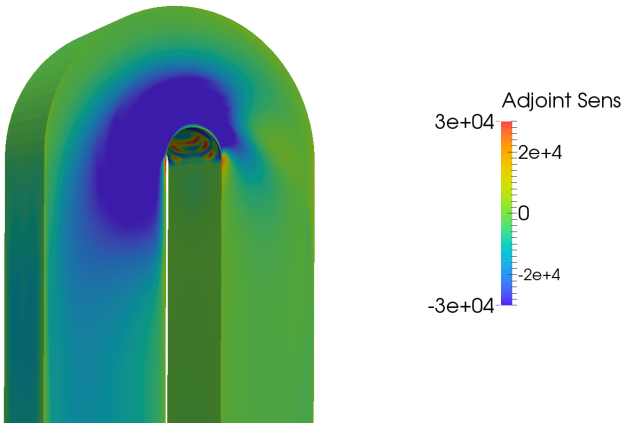


Figure 5.12: U-Bend Duct: Total pressure losses sensitivity map. Color map as in Fig. 5.5.

The inlet normal velocity is $8.4m/s$, the hydraulic diameter is $0.075m$ and the flow Reynolds number based on the above quantities is $Re \approx 43,800$. The initial velocity field is plotted in Fig. 5.11. The parameterization of the U-turn of the duct is performed with volumetric B-splines. The control box used can be seen in Fig. 5.10b. The total number of moveable control points is 18 and, since they are allowed to move in all directions, this leads to $N = 54$. The initial direction of the deformations can be seen by the computed surface sensitivities, Fig. 5.12.

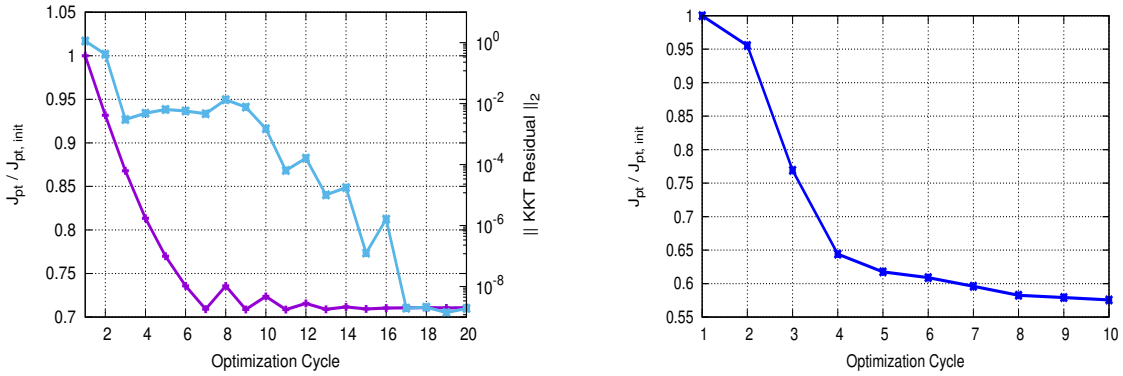
Using this parameterization, an unconstrained optimization with the BFGS method leads to a 42.5% drop in J_{pt} (Fig. 5.14b). The final shape (Fig. 5.13a) is inflated w.r.t. the initial one in certain areas which can lead to violations of manufacturability criteria. For instance, the shape could tend to form a zero-thickness geometry with the surface of a neighbouring part and, therefore, containment of

the optimized geometry within certain boundaries should be imposed. Therefore, a bounding box constraint is imposed as in Fig. 5.13b.

(a) (b)

Figure 5.13: U-Bend Duct: Final shape resulting from the unconstrained optimization (left) and the same shape along with the bounding box (transparent red) used in the constrained optimization (right).

The constrained optimization starts with the initial geometry and aims at preventing the shape from intersecting with the bounding box. A value of $c_s = 0.001$ is used to set the penalty function. The constrained optimization leads to a 29% reduction in total pressure losses. The convergence history of both the objective function and the residual of the KKT equations can be seen in Fig. 5.14a and the final shape which lies completely inside the bounding box in Fig. 5.15a. In Fig. 5.16, the cumulative normal displacements, resulting from the constrained and unconstrained optimizations, are compared. In the unconstrained case, the total pressure losses are minimized by deforming ('inflating') the outer walls of the U-turn. In the constrained case, since the outer walls are not given much space to deform, the objective function is minimized by reshaping mainly the inner walls. Because of this, the large recirculation bubble which appears in the initial duct geometry (along the inner walls and towards the outlet), Fig. 5.17a, is not entirely suppressed. As seen in Fig. 5.18, for the duct which has been optimized under constraints, the difference between the total pressure and an average inlet total pressure is much smaller towards the outlet compared to the initial geometry.



(a) **(b)**

Figure 5.14: U-Bend Duct: Convergence history of the constrained (left) and unconstrained (right) optimization. For the former, the convergence history of the objective (purple) and the residual of the KKT equations (blue) are plotted.

(a) **(b)**

Figure 5.15: U-Bend Duct: Two perspective views of the final shape resulting from the constrained optimization. The duct is in blue while the bounding box in transparent red.

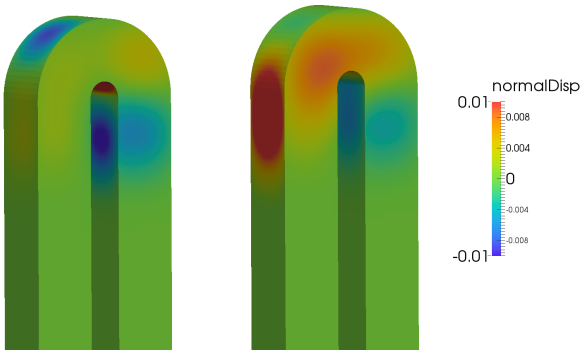


Figure 5.16: U-Bend Duct: Comparison of the cumulative normal displacements resulting from the constrained (left) and the unconstrained (right) optimizations, plotted on the initial shape. Areas with positive values correspond to outward displacements and areas with negative values to inward.

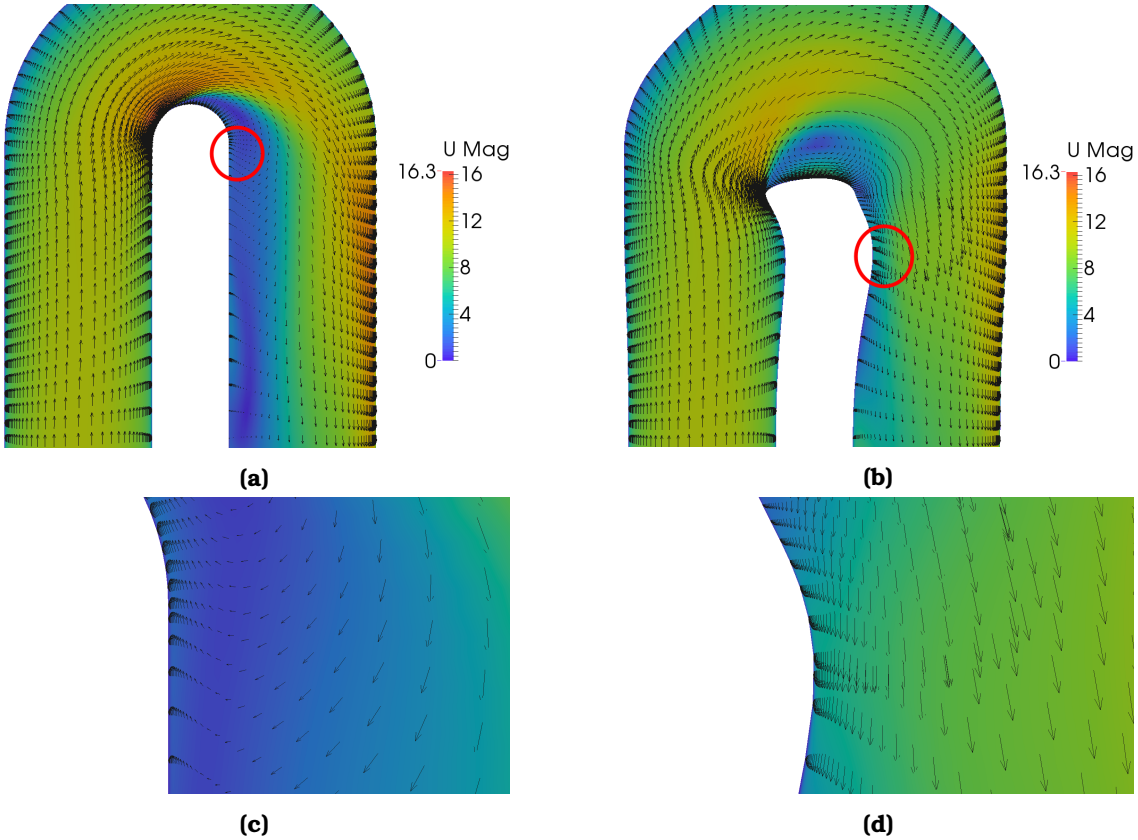


Figure 5.17: U-Bend Duct: Flow velocity field plotted on the half-span slices of the initial and optimized (under constraints) geometries. Arrows stand for the velocity vectors. Top-Left: Initial geometry. Top-Right: Optimized geometry (under constraints). Bottom-Left: Close-up view of the right leg at the inner walls of the initial geometry (circled area in (a)). Bottom-Right: Same view of the right leg at the inner walls of the optimized (under constraints) geometry (circled area in (b)).

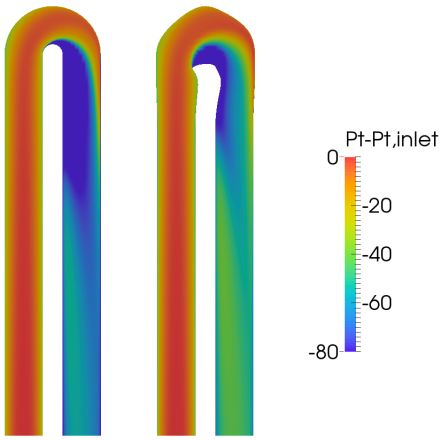
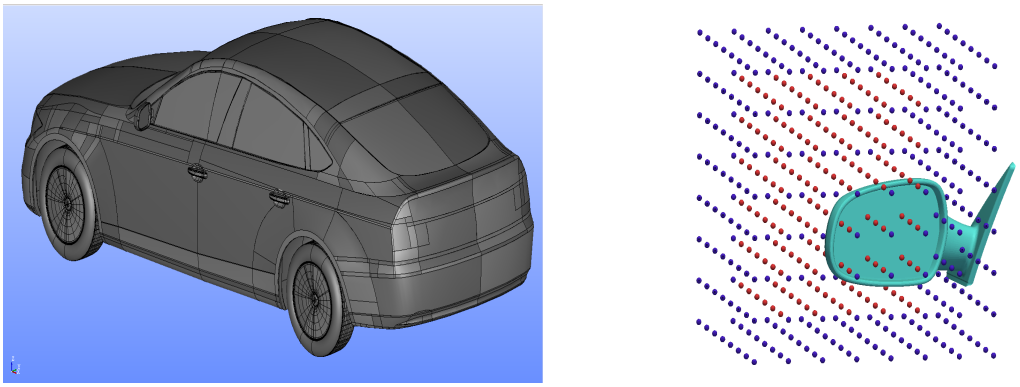


Figure 5.18: U-Bend Duct: Difference between the total pressure field and an average total pressure at the inlet, plotted on half-span slices of the initial (left) and the optimized (right) geometries.

5.4.3 Optimization of the Side Mirror of the DrivAer Car Model

The third case chosen for testing the effectiveness of the method is that of the DrivAer concept car (Fig. 5.19a), designed by the Technical University of Munich (TUM). The configuration used in this study is the fast-back car model with a smooth underbody, with both mirrors and stationary wheels. The target is to minimize the drag coefficient (c_D) of the whole car, by exclusively modifying the shape of the side mirrors. The study is performed on half of the car by employing symmetry conditions. The computational mesh consists of approximately 5.3M hexahedra with an average $y^+ = 75$ of the barycenters of the first cells off the wall.

The model is subjected to a far-field longitudinal velocity of $38.89m/s$, with a Reynolds number of $Re \approx 2,6 \times 10^6$ (based on the car width). The side mirror is parameterized in two ways: (a) with volumetric splines and a control box with $8 \times 8 \times 8$ control points (Fig. 5.19b) and (b) using the BRep model of the side mirror. With the volumetric spline configuration, C_0 and C_1 continuity between the free-to-move regions of the mesh and those which are fixed, is ensured by freezing certain control points, thus leaving $6 \times 5 \times 6$ free-to-move control points ($N = 540$). With BRep configuration, C_0 and C_1 constraints are imposed via the method of Chap. 4 on all patches for which the constraints are already true.



(a) (b)

Figure 5.19: DrivAer Side Mirror: The full CAD model of the car (left) and the control box used for the surface and mesh parameterization (right). Red control points are allowed to move while blue points are fixed.

The first step of this study is to perform an unconstrained optimization and observe the optimized side mirror's shape. Design variables of the splines volume are updated using BFGS and the initial sensitivity map can be seen if Fig. 5.20. As expected, the mirror tends to shrink so as to reduce its frontal area. However, parts of the mirror become distorted and unusable. For instance, the surface of the mirror cover folds and the reflector glass twists (Fig. 5.21). Because of this, two bounding surfaces are placed as in Figs. 5.22a, 5.22b to block the distortion of the mirror's shape. These are placed in front of and behind the reflector, in close proximity to the glass, in order to block potential twisting and are shaped after the protruding parts of the mirror so as to block potential over-shrinking. The distance of each bounding surface to the mirror's reflector glass is $0.0025m$ and the constraint is setup for $C_s = 0.001$.

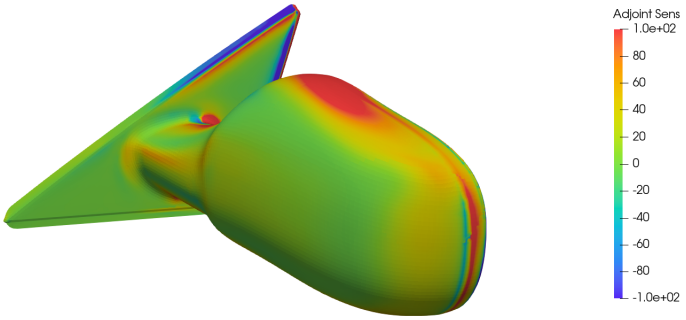
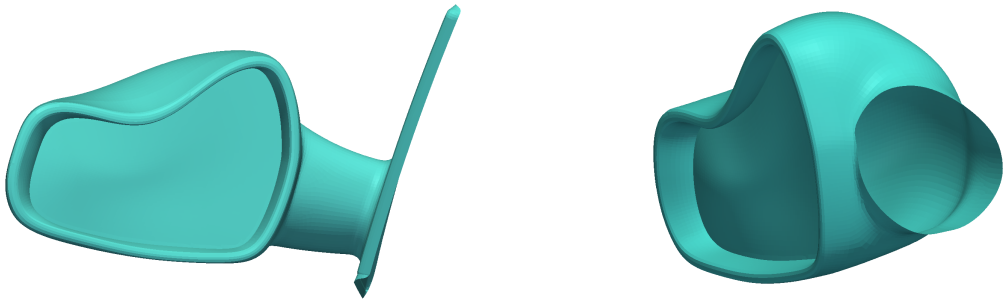
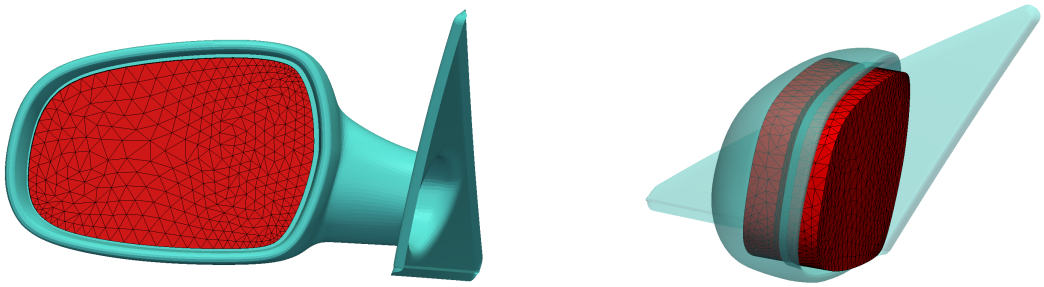


Figure 5.20: DrivAer Side Mirror: Drag sensitivity map. Color map as in Fig. 5.5.



(a) **(b)**

Figure 5.21: DrivAer Side Mirror: Front (left) and side view (right) of the mirror resulting from the unconstrained optimization for the selected parameterization.



(a) **(b)**

Figure 5.22: DrivAer Side Mirror: Left: Front view of the mirror. Right: Side view (with a transparent mirror body). The triangulated bounding surfaces are shown in red.

The constrained optimization is, then, performed and the drag coefficient is reduced by 0.8% (Fig. 5.23). The optimized shape is shown in Fig. 5.24. The reflector glass of the mirror remains between the bounding surfaces and at the same time, the cover of the mirror shrinks slightly without high distortions. Finally, the comparison of the cumulative normal displacements resulting from the constrained and unconstrained optimizations are shown in Fig. 5.25 .

According to this figure, for the constrained optimization, the displacement of the reflector glass in the normal direction is less than $0.0025m$. The parts with the greatest displacement are located near the protruding parts of the mirror, where the shape almost touches the bounding surfaces, and on the back of the mirror's shell. While that distance is smaller than the offset of the bounding surfaces defined, the curvature of the reflector glass must be checked as it indicates local flatness. The curvature is, therefore, evaluated according to the algorithm in [**?**] for three shapes: (a) the initial mirror shape, (b) the shape resulting from the constrained optimization and (c) that resulting from the unconstrained optimization.

The curvature values, plotted on each shape, can be seen in Fig. 5.26. The reflector glass is practically flat as the curvature values on it (for the shape resulted from the constrained optimization) range between 0 and 0.4 which is practically zero in this scale.

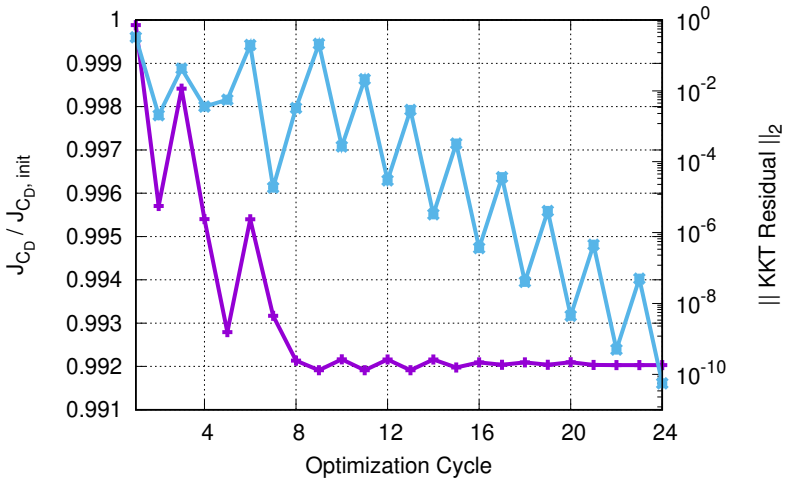
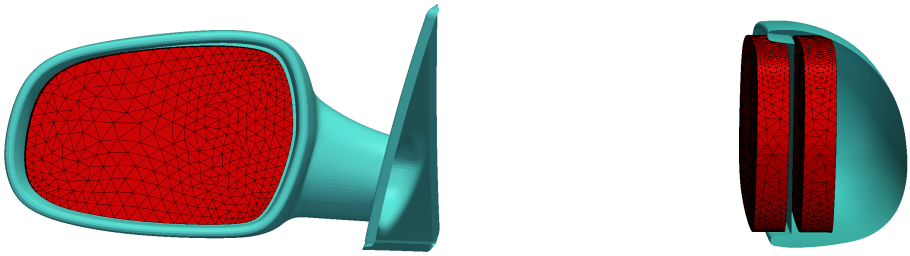


Figure 5.23: DrivAer Side Mirror: Convergence histories of the objective function (purple) and the residual of the KKT equations (blue).



(a) **(b)**

Figure 5.24: DrivAer Side Mirror: Optimized shape (light blue) resulting from the constrained optimization along with the two bounding surfaces in triangulated form (red). Left: Front view. Right: Lateral cross section of the mirror.

Finally, for the constrained optimization case, it is interesting to see how such small displacements affect the force exerted on the surface of the car by the flow. Thus, the local forces (i.e. integrand of Eq. 2.20) are computed on the entire car for two geometries; the initial and the optimized under constraints. The difference between these paired forces is plotted in Fig. 5.27.

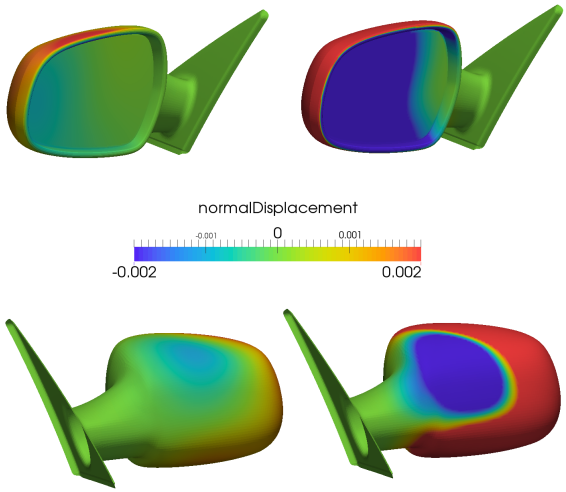


Figure 5.25: DrivAer Side Mirror: Comparison of the cumulative normal displacements resulting from the constrained and the unconstrained optimizations plotted on the initial shape. Positive / negative displacements indicate that the surface is 'pushed in / out'. Top-Left: Front view of the mirror for the constrained optimization. Top-Right: Front view of the mirror optimized without imposing constraints. Bottom-Left: Back view of the mirror resulting from the constrained optimization. Bottom-Right: Back view of the mirror resulting from the unconstrained optimization.

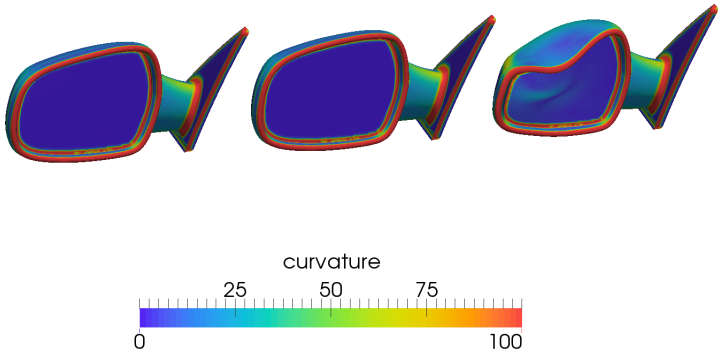
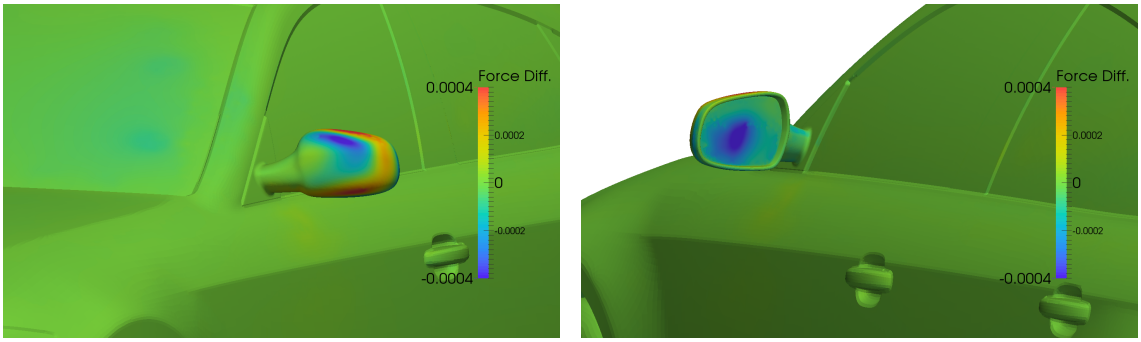


Figure 5.26: DrivAer Side Mirror: Curvature values plotted on the initial mirror (left), the shape resulted from the constrained (middle) and unconstrained (right) optimization.



(a) (b)

Figure 5.27: DrivAer Side Mirror: Difference between the local forces exerted by the flow on the surface of the car for the constrained optimized and the initial geometries. Negative (positive) force difference means positive (negative) effect on the drag coefficient reduction.

The constrained optimization is then repeated by using the BRep configuration (Fig. 5.28), with the same two bounding surfaces.

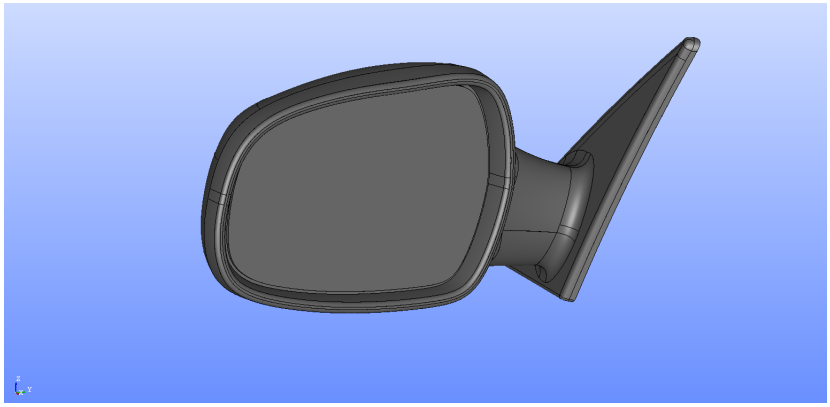
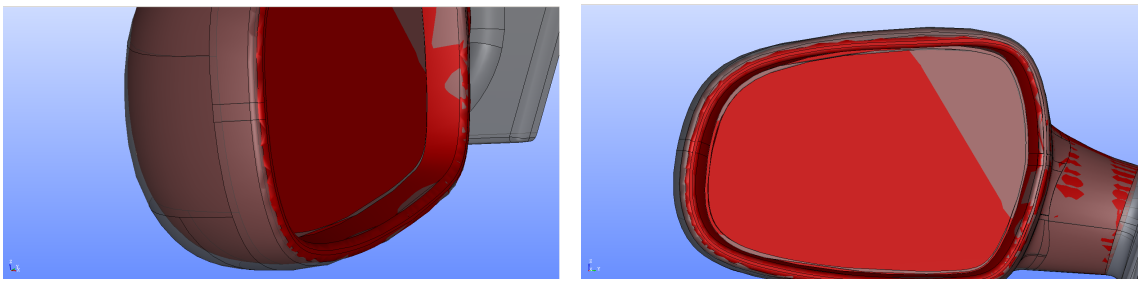


Figure 5.28: DrivAer Side Mirror: BRep model of the side mirror.

SQP is also used and the constrained optimization converges after 10 cycles producing a drop in the objective function equal to 0.82%. The initial along with the updated shapes can be seen in Fig. 5.29.

It can be noted that with this configuration, apart from the top of the mirror's shell, its side is also shrinked towards the side of the car, thus resulting in a slightly better performance.



(a) **(b)**

Figure 5.29: DrivAer Side Mirror: The initial (transparent grey) and the final (red) shapes of the side mirror's BRep. The side (left) and the front (right) views of the mirror are visible.

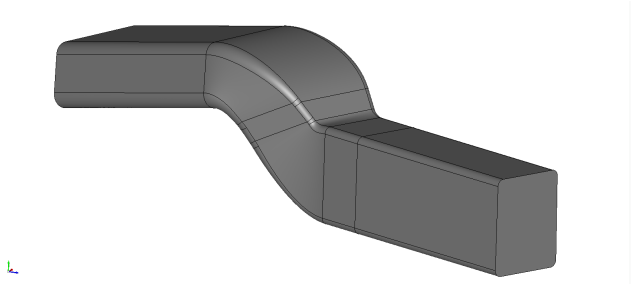


Figure 5.30: The S–bend climate duct.

5.4.4 Optimization of the S–bend Duct

The S–bend climate duct is a test case provided by VolksWagen AG. The goal is to minimize the pressure losses J_{pt} (Eq. 2.19) of the flow subject to the deformation of the S–section of the duct. The objective function has the following form.

The geometry consists of 46 trimmed NURBS patches, among which 28 belong to the S–section. These 28 patches are high–degree surfaces and this results to a total number of 5830 control points (17490 degrees of freedom) on the S-section. The flow analysis is done using a mesh of 700,000 cells. The flow is laminar with $Re \approx 400$. The parameterization method includes CAD into the loop as shown in Chap. 4.

Initially, an unconstrained optimization is performed on the duct that leads to a total pressure losses drop of 9.1 % after 40 cycles. However, the resulting shape, due to the high number of control points, becomes wavy with areas of very high curvature (fig. 5.31). Then, a constrained optimization with the curvature constraint is performed with $c_m = 10^6$. Using the constrained optimization, a drop of 3.9 % is achieved.

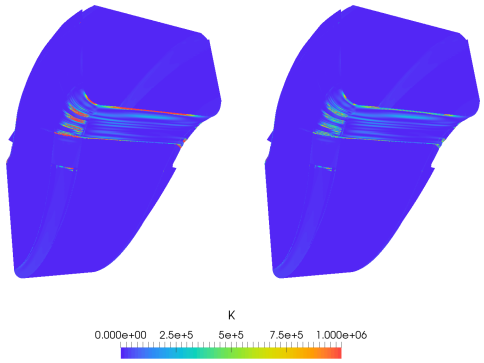


Figure 5.31: Curvature iso-areas on the optimal shape resulting from the unconstrained (left) and the constrained optimization (right) of the S-bend duct.

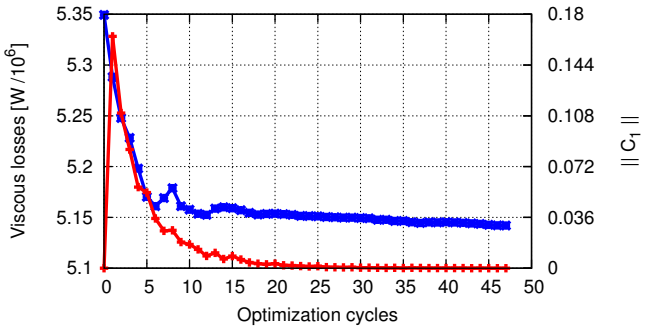


Figure 5.32: The convergence history of the viscous losses objective function (blue) and the curvature constraint (red) for the S-bend duct.

Figure 5.33: Comparison of the optimal shape resulting from the constrained (right) and the unconstrained (left) optimization of the S-bend duct.

The curvature map on the optimal solution of the constrained and unconstrained optimization are shown in figure 5.31 and the final CAD surfaces are shown in fig. 5.33. From the curvature map, one may see that the shape attempts to form the wavy surface but the constraint does not allow this to happen. Finally, the convergence history of the objective function and constraint can be seen in fig. 5.32.

5.4.5 Optimization of the Extruded NACA0012 Airfoil

A NACA0012 airfoil which lies on the $X - Y$ plane is fitted by a NURBS curve. This NURBS curve is then extruded along the Z axis to create a wing (Fig. 5.34).

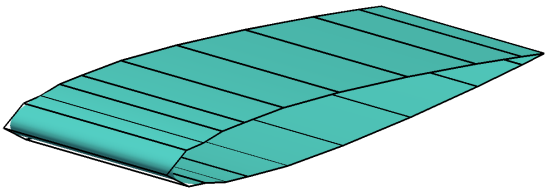


Figure 5.34: The NACA0012 extruded wing (light blue) and its NURBS control grid (black wireframe).

The volume of the wing spanning between the planes at $Z = 0m$ and $Z = 0.5m$ is $V_{init} = 0.0412m^3$. The flow analysis around the wing is performed using a 2D mesh of approximately $250,000$ cells. The flow is turbulent $(Re = 10^6)$ with a free stream velocity $U = 60m/s$ and an angle of attack at 0^o. The target of the constrained optimization is to reduce the drag force on the wing while making sure that $V \geq V_{min} = 0.8V_{init} = 0.033$. As shown in fig. 5.35, the optimization procedure starts with the objective function dominating the constraint but, as the penalty factor of the constraint increases, the optimization converges to a 4.3% drop in drag.

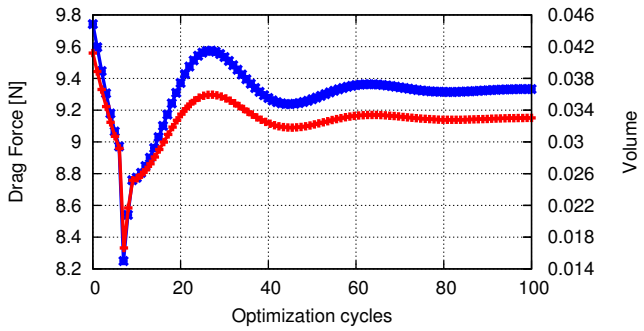


Figure 5.35: The convergence history of the drag objective function (blue) and the volume constraint function (red) for the NACA0012 wing.

5.5 Remarks

In this chapter, the imposition of constraints during NURBS-based shape optimization is presented. Algorithms have been developed for accurately imposing constraints on multiple mesh nodes on a shape to be designed. The cases shown, range from academic to industrial-like ones to fully test the algorithms' capabilities. The methods proved reliable and the time required for the quantification of the constraints (in all cases, including evaluation and differentiation) is negligible in comparison to the time required to solve the primal and adjoint problems.

The proposed algorithm addresses spline-based parameterizations but, definitely, does not depend on the parameterization scheme.

For the bounding constraints, two types of bounding surfaces were used to constrain the shape's deformation: surfaces that lie inside the shape to be designed (surface behind the mirror of the car) and surfaces that lie outside of the shape to be designed and / or enclose it. The proposed method was able to handle both cases reliably. For the curvature constraints, smoothness was imposed on a wrinkled NURBS shape that restored manufacturability.

This study can be the motivation to address many interesting topics related to point-wise and NURBS-based constraints in general. One example could be drag minimization for airfoils which tends to make them thinner. Optimizing a ship's hull can be another example since in certain areas the hull must (to an extent) retain its shape.

Lastly, the implicit integration of inequality constraints, by means of a penalty function, to generate a single equality constraint has proven to be successful and can be used in various other geometric constraints such as thickness after formulating them as point-wise inequalities.

Chapter 6

Applications

6.1 The Compressor Stator of Technical University of Berlin

The "TurboLab Stator" was measured rig at the Technical University of Berlin [**?**] in the TurboLab at the Chair for Aero Engines. A photo-realistic CAD model of the stator assembly with 15 blades is shown in Fig. 6.1. In this chapter, this will be used as a test case in which the developments shown in Chap. 3-5 are applied.

Figure 6.1: The TurboLab stator in full assembly with 15 blades and mounting holes.

The case has been designed based on a representative stator geometry as used in modern jet engine compressors. This initial geometry (Fig. 6.2) has to be optimized to reduce the total pressure losses J_{Pt} over the incidence range given in table 6.1. The BRep model of the CFD domain with just one blade included is provided in STEP format. Its trimmed and untrimmed states can be seen in Figs. 6.3 and 6.4, respectively.

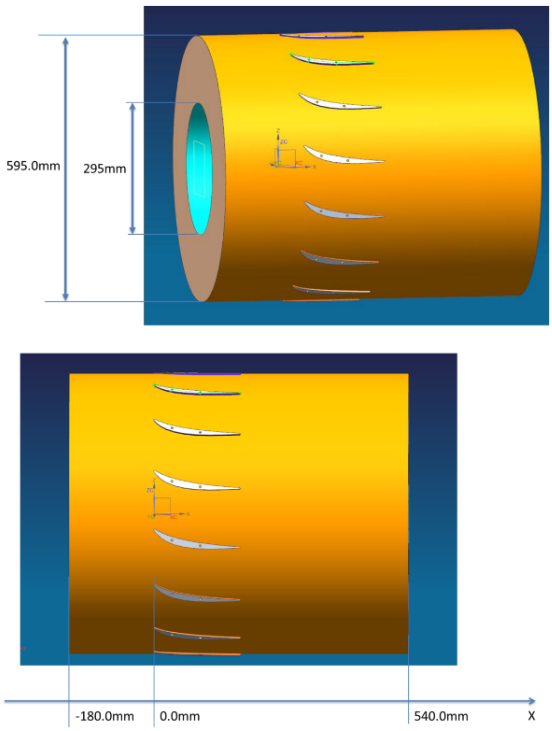


Figure 6.2: CAD geometry of the TurboLab stator with dimensions. The computational domain is portrayed as a hollow cylinder of length equal to 720 mm.

Description	Value
Inner Radius	147.5 mm
Outer Radius	297.5 mm
Inlet Axial Position	-180.0 mm
Outlet Axial Position	540.0 mm

Table 6.1: Geometric characteristics of the computational domain for the TurboLab stator.

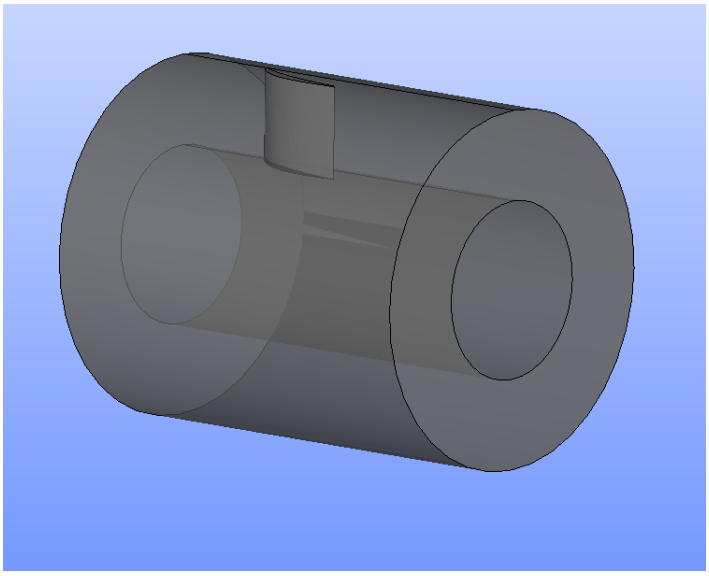


Figure 6.3: The BRep geometry of the TurboLab stator in transparent grey.

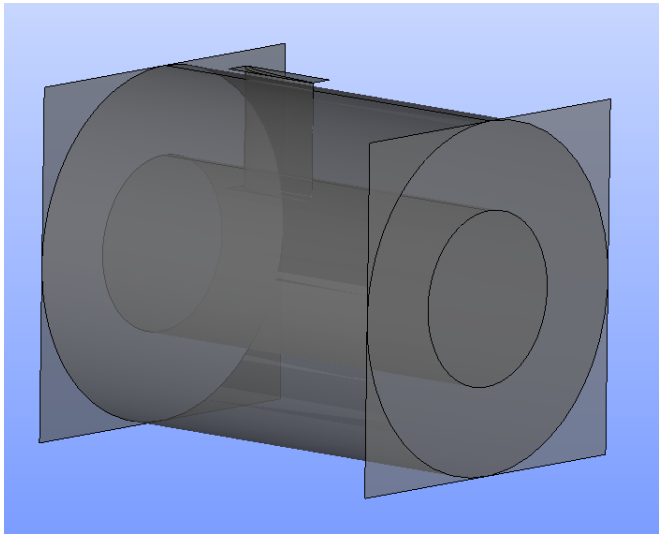


Figure 6.4: The BRep geometry of the TurboLab stator in its untrimmed form.

The entire model (blade, inlet, outlet, hub and shroud), consists of a total of 10 NURBS patches and 1716 NURBS control points. The two cylindrical surfaces consist of 7×2 control points each, while the inlet and the outlet are planar surfaces consisting of 2×2 control points each. The surface of the stator blade consists of 6 patches in total: 2 patches with 25×23 control points, two patches with 8×23 control points and two patches with 9×9. These patches along with the control points can be seen in Fig. 6.5.

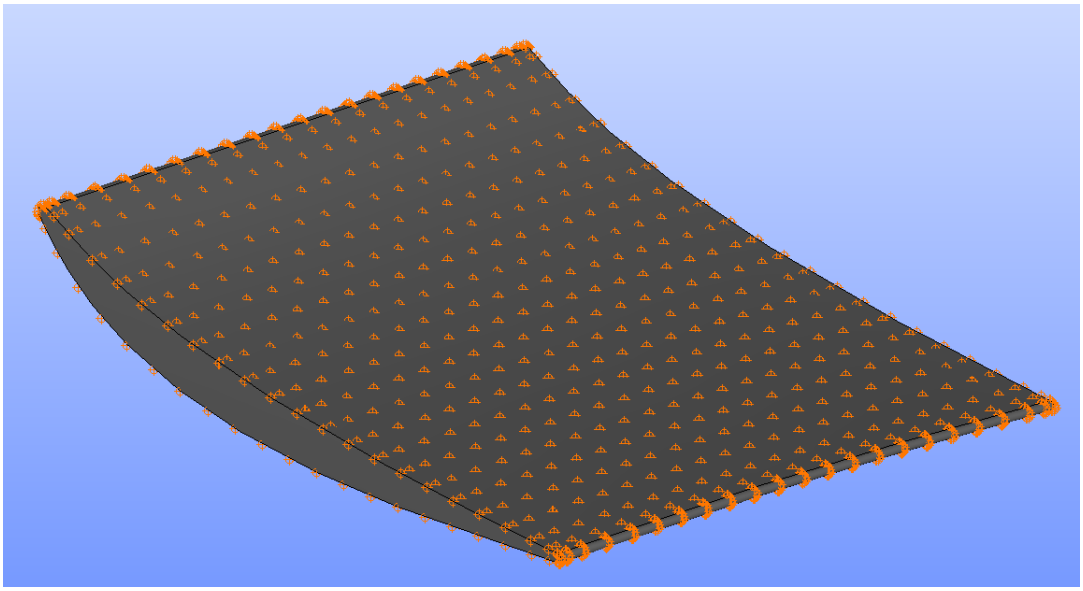


Figure 6.5: The BRep geometry of the TurboLab stator blade. The control points of the NURBS patches that correspond to the pressure and suction sides can be seen in orange.

6.1.1 Generating the CFD Mesh

The mesh of the CFD domain, bounded by the BRep of the stator and the casing, must, firstly, be generated. For that to happen, the BRep model is triangulated (Fig. 6.7). As seen in Fig. 6.4, the radial span of the blade must be reduced as it exceeds the outer casing. After this, the Shape Healing algorithm is used to snap the newly created blade on the surfaces of the inner and outer casing (cylinders). The background mesh (Fig. 6.6) is, then, generated and the size map on it, for parameters $d = 10$ and $g_R = 0.05$, is computed. The Advancing Front method is, then, used for the above mentioned parameters to create the surface grid of Fig. 6.7. These parameters were chosen as they produced the highest quality surface grid.

The entire process required $\approx 90\ secs$ on 8 Intel Core i7-6700HQ (2.60 GHz) CPUs and produced $\approx 120K$ triangles averaging at approximately $1350\ tris/sec$. This was run in parallel using the OpenMP [**?**] shared memory parallel programming. Job partitioning among processors was done patch-wise, meaning that the total number of patches to be triangulated was equally divided over the available processors.

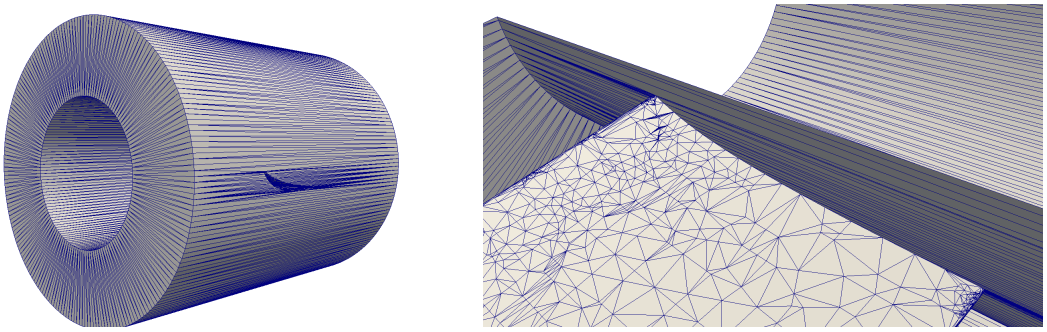


Figure 6.6: The background grid of the BRep model. Left: The entire domain. Right: The domain cut to make the blade visible.

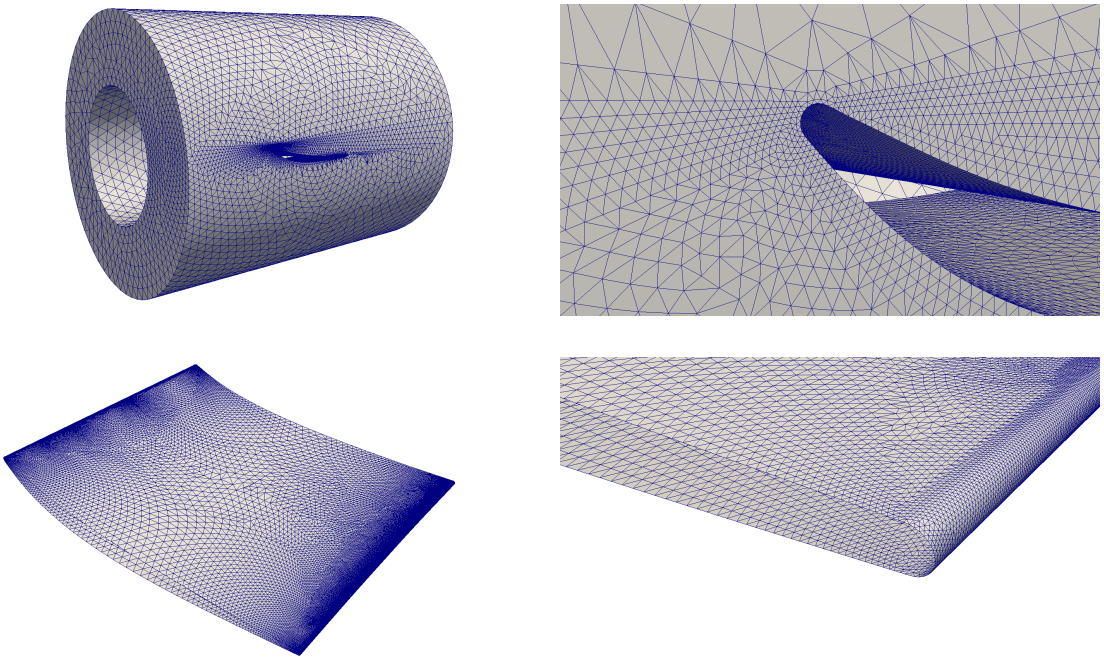


Figure 6.7: The triangulated BRep model using the method proposed in Chapter 3. Top-Left: The entire domain. Top-Right: Close-up view of the leading edge region of the outer cylindrical casing. Bottom-Left: The stator blade. Bottom-Right: Close-up view of the trailing edge region.

The CFD simulation is carried out using a single blade passage and periodic boundary conditions. The cylindrical domain is a 24^o sector of the domain. The interfaces are also triangulated surfaces generated based on the blade geometry. The mesh was, then, generated based on the defined triangulated boundary surfaces (Fig. 6.8). The mesh consists of two periodic boundaries, one patch for the shroud, and one for the hub, two patches on the blade (pressure and suction sides), an inlet and an outlet.

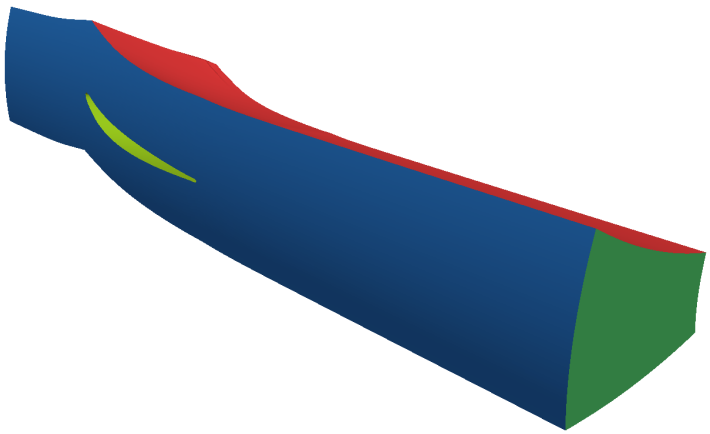


Figure 6.8: The CFD domain for the test case with each CFD patch colored differently. In this figure, the outer cylindrical patch is seen in blue, the outlet in dark green, one of the periodic boundaries in red and the blade surface in light green.

6.1.2 Setting up the Parameterization

In order to determine the characteristics of the parameterization, a few manufacturing constraints have to be taken into account. In order not to jeopardize the capacity of the cylindrical domain for 15 blades, a constraint is imposed that the blade geometry does not penetrate the 24^o interfaces. Secondly, the axial chord of the blade must remain unchanged. For that reason, the control points of the NURBS patches near the leading and trailing edges remain fixed. Thirdly, the thickness of the blade must be such that the leading and trailing edge circles' radii are at least $1\ mm$. Furthermore, the thickness must be enough that two mounting holes of diameter equal to $5\ mm$ each must fit into the blade profile both at shroud and hub (Fig. 6.9). The length of the holes must be equal to $20\ mm$. The thickness constraints are all imposed using the packaging constraints that are developed in Chapter 5. Finally, the blade must completely touch the hub and the shround surfaces (which are not part of the deformable domain).

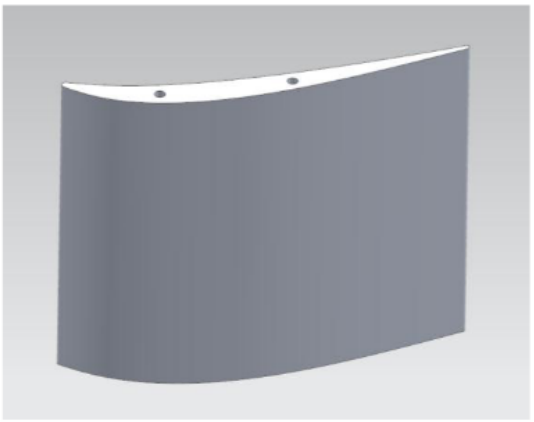


Figure 6.9: The blade geometry with two mounting holes visible. Two mirroring holes are also required on the opposite side.

Taking the manufacturing requirements stated above into account, only the NURBS patches that correspond to the blade can be displaced (Fig. 6.10). From these patches, the control points shown in Fig. 6.10 are fixed in place to impose the axial chord constraint. Using the method proposed and developed in Chapter 4, continuity constraints are imposed between all the surfaces belonging to the stator blade as well as the surfaces of the casing.

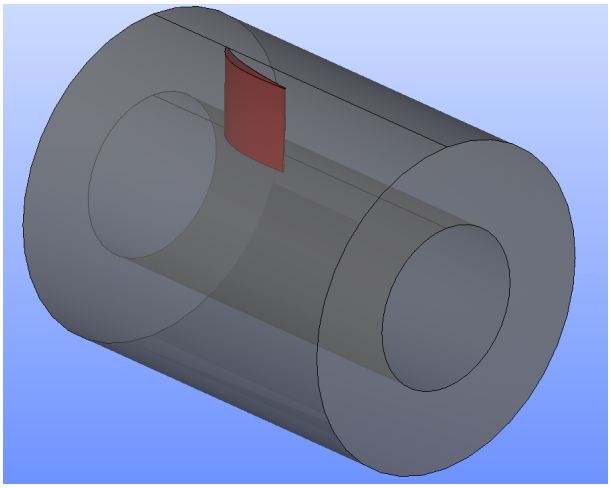


Figure 6.10: The BRep geometry of the stator. The deformable patches are seen in red, while the non-deformable patches are seen in transparent grey.

For all pairs of interconnected deformable NURBS patches, up to C_1 continuity is imposed. For pairs with a non-deformable patch, the condition that the deformable patch keeps touching the common edge, without a change in the orientation of its normal vector, is imposed. Taking all these into account, the

number of free-to-move control points is reduced to 1266, while the number of constrained control points via the Geometry Morphing method is 252. In Fig. 6.11, the control points of the blade are shown with different colors denoting the status of the design variables related to each control point.

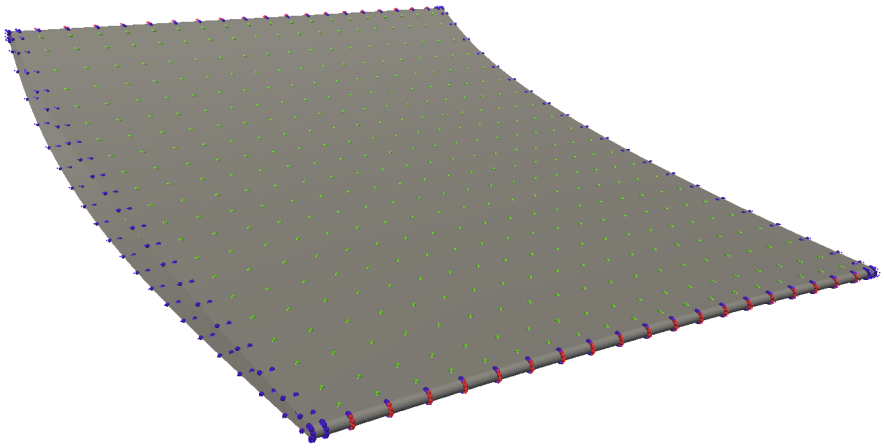


Figure 6.11: The control points of the 4 patches that correspond to the blade geometry. Unconstrained control points are seen in green while fixed-in-place control points in red. Control points constrained by the Geometry Morphing method are seen in blue.

6.1.3 Flow Conditions and Optimization Targets

The boundary conditions of the flow simulation are constant for the entire radial span. For the pressure field, at the outlet, a zero Dirichlet condition is imposed while at the inlet and all wall patches zero Neumann conditions are imposed. At the inlet, the whirl angle is 42^o and the pitch angle is 0^o. The inlet velocity magnitude is computed so that a massflow of $9.0\ kg/sec$ is achieved. At the outlet, a zero Neumann condition is imposed on the velocity components and at the wall patches, no-slip conditions are imposed. For the turbulence, the Spalart-Allmaras model along with the boundary conditions shown in Sec. 2.3 and [**?**], is used.

Based on these flow conditions, a CFD simulation on the baseline geometry was run. The target of this test case is to minimize pressure losses J_{Pt} between the inlet and the outlet (Eq. 2.19). J_{Pt} is differentiated for the parameters of the Geometry Morphing method. The surface sensitivities (i.e. the derivative of J_{Pt} w.r.t. the normal displacement of the design wall nodes) that is generated is seen in Fig. 6.12.

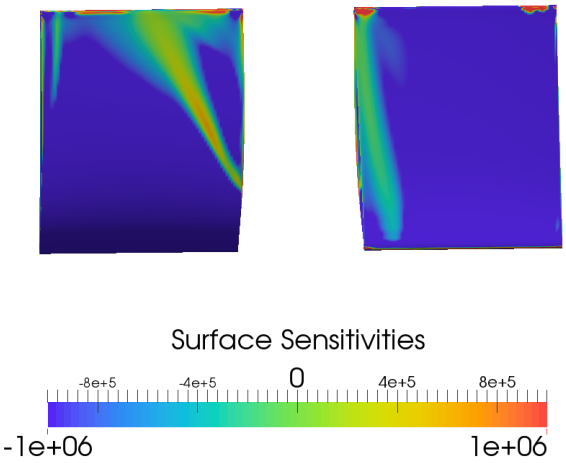


Figure 6.12: The surface sensitivities generated after the differentiation of the objective function with adjoint method, on the suction (left) and pressure side (right). Positive (negative) sensitivities indicate that the blade must be pulled out (pushed in) to reduce the objective function. In both figures, the trailing edge is at the top and leading edge at the bottom.

Apart from the objective function, there are also manufacturing constraints which must be respected during the optimization. These concern the mounting holes to the hub and shround (4 in total) and the minimum allowed radius of the trailing and leading edge circles (not less that $1\ mm$). Both these constraints are imposed via the no-intersection constraints shown in Chapter 5.

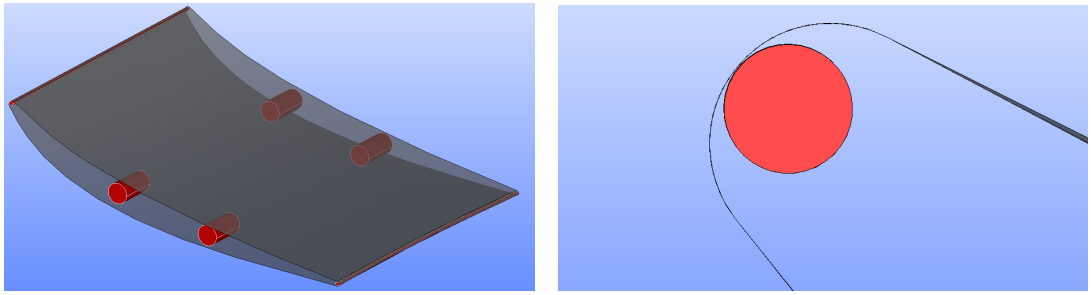


Figure 6.13: The stator blade (in transparent grey) along with the constrained surfaces (red). Left: Isometric view of the blade along with all the surfaces. Right: A close-up view of the leading edge of the blade. A surface created from a $2\ mm$ radius circle, swept along the profile of the blade, is created.

In these cases, the bounding surfaces are cylinders of $5\ mm$ diameter and

$20\ mm$ height for the mounting holes and $2\ mm$ diameter and height equal to the radial span of the blade for the leading and trailing edges. The formation of these constrained surfaces are shown in Fig. 6.13.

6.1.4 Unconstrained Optimization

Initially, an optimization run is performed without taking the bounding surface constraints into account. This is a necessary step in order to check how the shape is perturbed and verify that the bounding surface constraints will affect the result. To update the design variables as formed by the Geometry Morphing method, steepest descent is used [**?**]. After 10 optimization cycles, steepest descent converges to a solution with total pressure losses reduced by $6.6\ \%$ (Fig. 6.14).

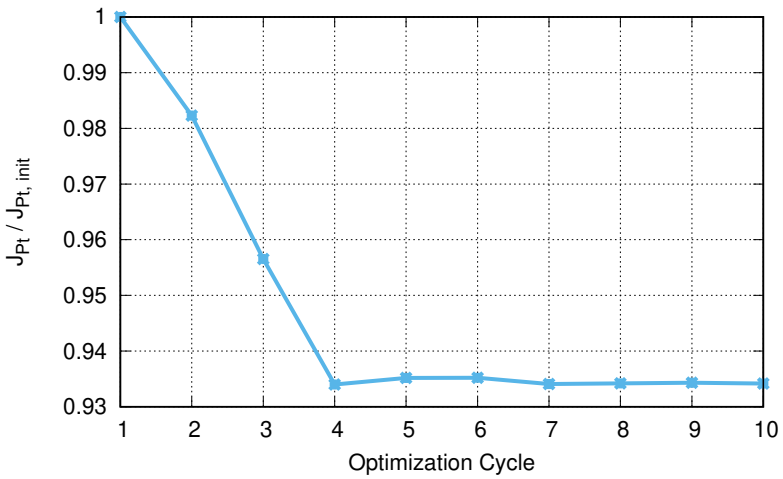


Figure 6.14: Convergence history of the unconstrained run for the TurboLab stator.

The final shape produced by the unconstrained optimization can be seen in Figs. 6.15, 6.16. It is clear from the resulting shape that the bounding surfaces that correspond to the mounting holes are penetrated by the optimal design. Therefore, a constrained optimization follows in order to take the STL surfaces corresponding to these holes into account.

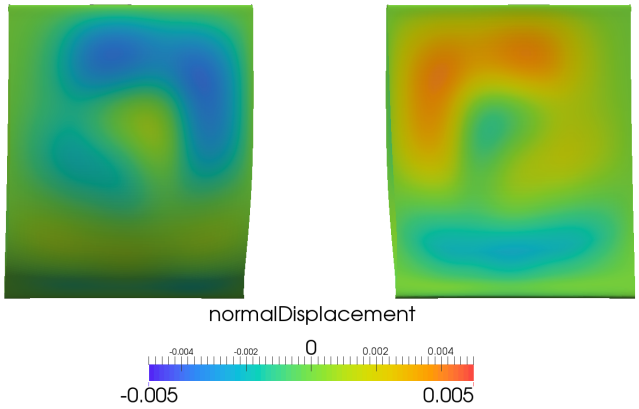


Figure 6.15: The cumulative normal displacement of the boundary mesh on the blade, resulting from the unconstrained optimization after 10 cycles. The normal displacement is the dot product of the displacement and boundary mesh face normals vector fields. Positive (negative) normal displacement means that the blade is pulled out (pushed in). The orientation is as in Fig. 6.12.

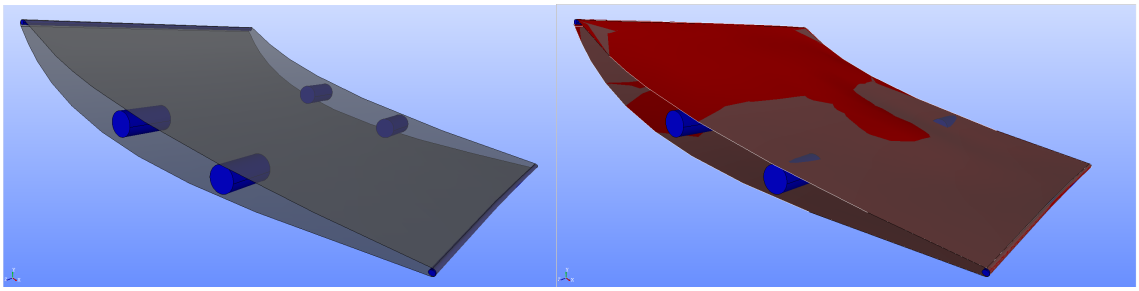


Figure 6.16: The final CAD model of the TurboLab stator resulted from the unconstrained optimization. Left: The baseline model in transparent grey with the bounding surfaces in blue. Right: The baseline model and the bounding surfaces as on the left, along with the optimized shape of the CAD model in red.

6.1.5 Constrained Optimization

The distance of the bounding surfaces to the initial geometry is of the order of millimiters, therefore, $c_s = 0.001$ is chosen. The constrained optimization method chosen is the ALM [**?**]. This is because of the high number of design variables which ruled out the possibility to use an SQP here. The constrained optimization run achieved a $\approx 3.5\ \%$ drop in the objective function (Fig. 6.17).

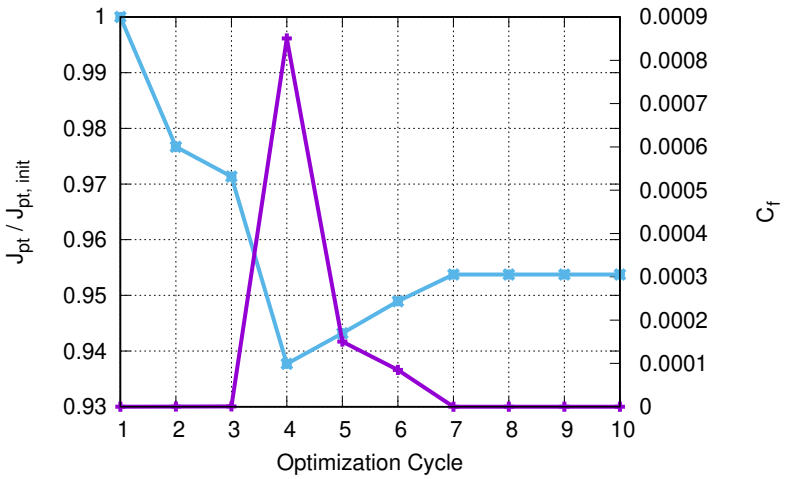


Figure 6.17: Convergence history of the objective (blue) and constraint (purple) functions.

Similarly to the unconstrained case, the signed cumulative displacement projected on the normal to each baseline face vector is portrayed in Fig. 6.18.

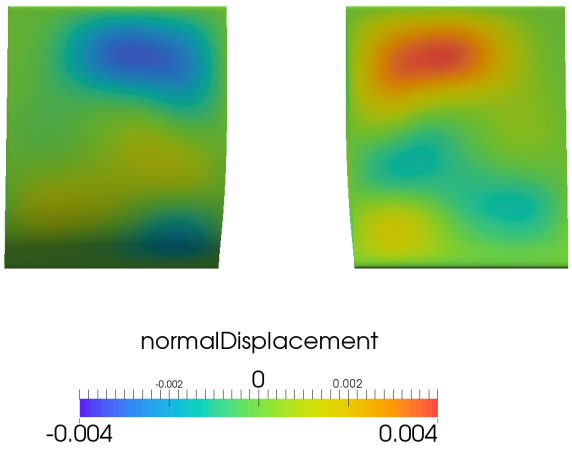


Figure 6.18: The cumulative normal displacement of the boundary mesh on the blade, resulting from the constrained optimization after 10 cycles. Positive (negative) normal displacement means that the blade is pulled out (pushed in).

The final CAD model as exported from the constrained optimization (along with the bounding surfaces is seen in Fig. 6.19.

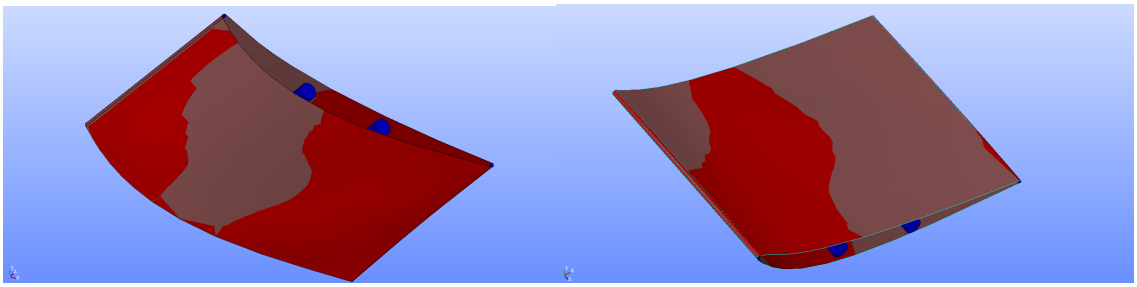


Figure 6.19: The final CAD model of the TurboLab stator resulted from the constrained optimization. The baseline model is seen in transparent grey with the bounding surfaces in blue and the optimized model in red. Isometric views of the pressure side (right) and the suction side (left) are shown.

6.1.6 Remarks

In this chapter, the developments of Chap. 3-5 are combined to perform a full-scale CAD-based optimization. The geometry of the TurboLab stator is provided in the standardized Step format and all algorithms are run based on that geometry. The final optimized geometry is then exported in Step format as well. The blade can now be inserted in any CAD package for post-processing or in a CAM package for manufacturing. It can also be exported in STL format for rapid prototyping which would make for a huge advantage in a supply chain.

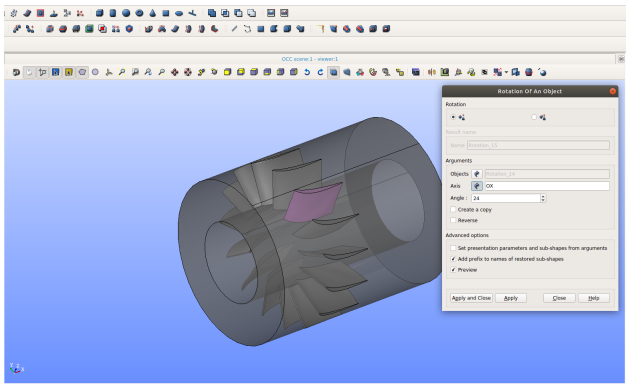


Figure 6.20: The stator optimized geometry along with all 15 blades. The entire CAD Graphical User Interface is shown in order to portray the process of revolving the optimized blade (pink).

The optimization of the TUB stator geometry subject to a number of flow objectives has been tried in various PhD theses accomplished in PCOpt/NTUA.

Vasilopoulos [**?**] performed both CAD-free and CAD-based optimizations of the same stator. In the CAD-free one, multi-objective optimization (MOO) was per-

formed that aimed to decrease both the total pressure losses and the exit flow angle. MOO was turned into a single objective optimization (SOO) by using the weighted sum of the objectives. The parameterization was node-based and the sensitivities were smoothed implicitly as shown in [**?**]. The resulting optimized geometry produced slightly increased J_{Pt} by 0.15 $\%$ due to the fact that the sensitivities for the second objective were dominant. In the CAD-based case, MOO was also performed as well as multi-point optimization. The chosen CAD parameterization is that of the in-house Rolls-Royce Deutschland (RRD) tool Parablading [**?**]. Since, it is in-house, its differentiation was possible. The CAD-based optimization, resulted in a decrease in J_{Pt} by 0.2 $\%$.

Gagliardi [**?**] performed CAD-based optimization using RBF-morphing, a method that was developed in [**?**]. This method uses RBF parameterization as a means to displace the NURBS control points of a model. Then, it follows a continuity recovery step that ensures watertightness and/or smoothness of the model were required. Here, too, an decrease in J_{Pt} by 0.2 $\%$ was the result.

In this thesis the objective function in the constrained run was improved by $\approx 3.5\%$. The reason behind the better performance is due to the factors that this is a SOO and that the nature of the parameterization made the design space richer than the other methods.

6.2 The Concept Intake Manifold

For the further testing of the methods and algorithms developed in this thesis, a test case which resembles an automotive intake manifold is designed. The manifold consists of one circular inlet and four circular outlets of same diameters (Fig. 6.21).

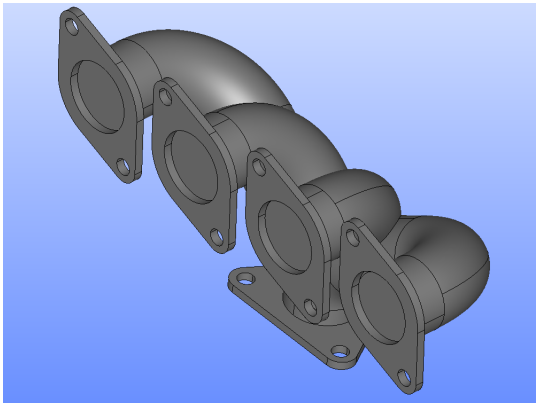


Figure 6.21: The CAD model of the intake manifold designed along with mounting flanges.

6.2.1 Generating the CAD Geometry & the CFD Mesh

The geometry of the manifold is designed in the open-source CAD software SALOME which is based on the OpenCascade Technology CAD Kernel. The desing is done by generating the inlet circular disk sketch and extruding it along four different 3D paths that lead to the outlets. The diameter of the circular disks is equal to $70mm$. The normal to the inlet is paraller to the $Y-$axis while the normals to the outlets are parallel to the $Z-$axis. The CAD model (without the mounting flanges) consists of 34 patches with a total of 356 control points and can be seen in Fig. 6.22.

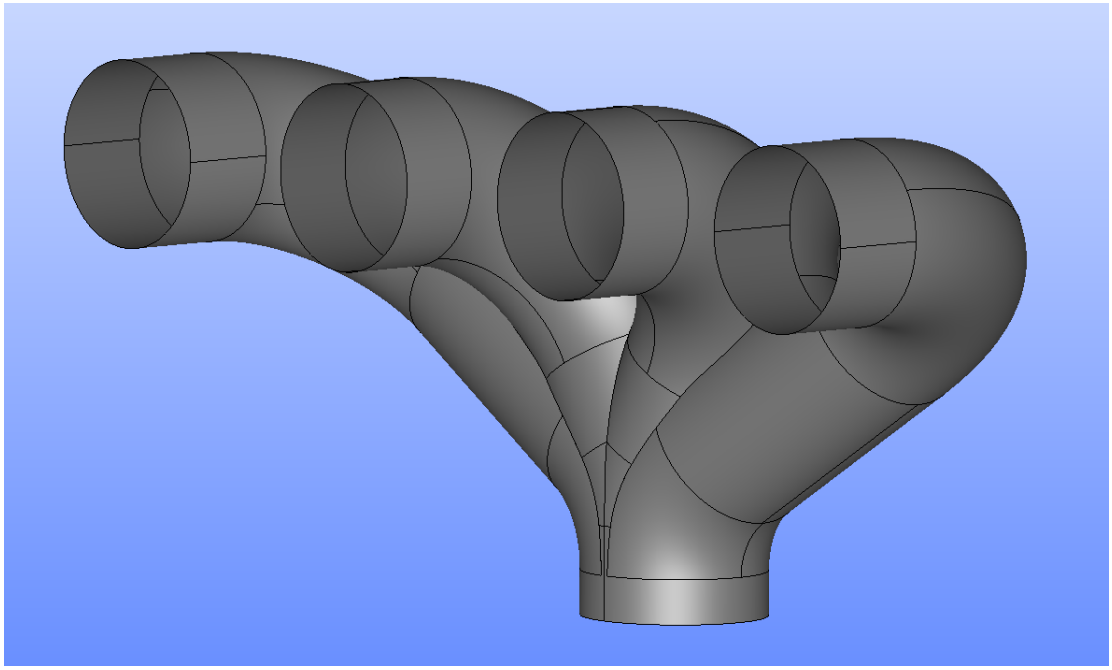


Figure 6.22: The CAD model of the intake manifold without the mounting flanges.

By using the method proposed in Chap. 3, and input parameters $d = 10$ and $g_R = 0.05$ the surface mesh of Fig. 6.23 is obtained.

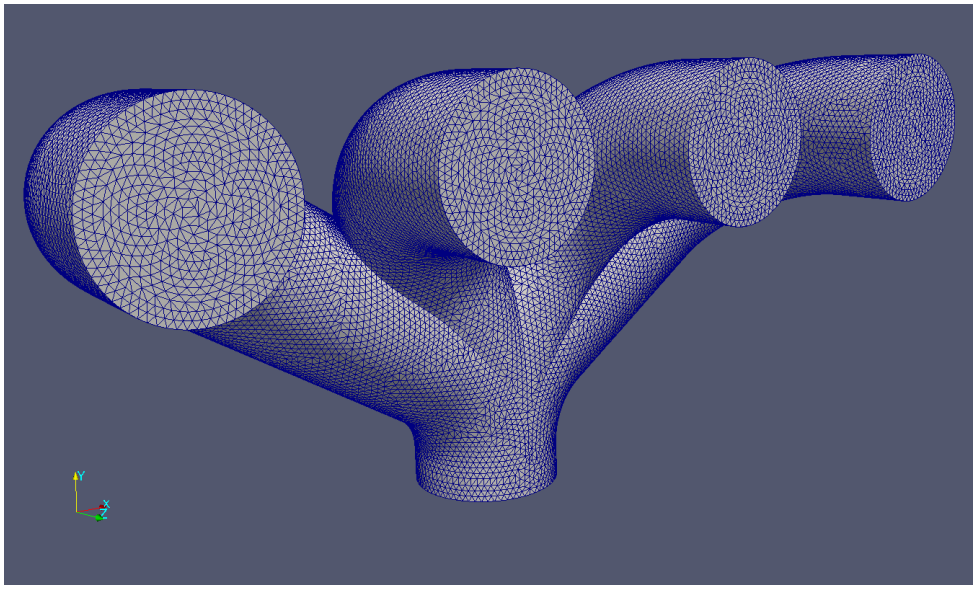


Figure 6.23: The surface mesh of the intake manifold.

The surface grid is then divided into three parts: The inlet grid, the outlet grid and the walls grid. These three different (but coherent) grids are then given to the snappyHexMesh tool of OpenFOAM to produce the computational mesh shown in Fig. 6.24.

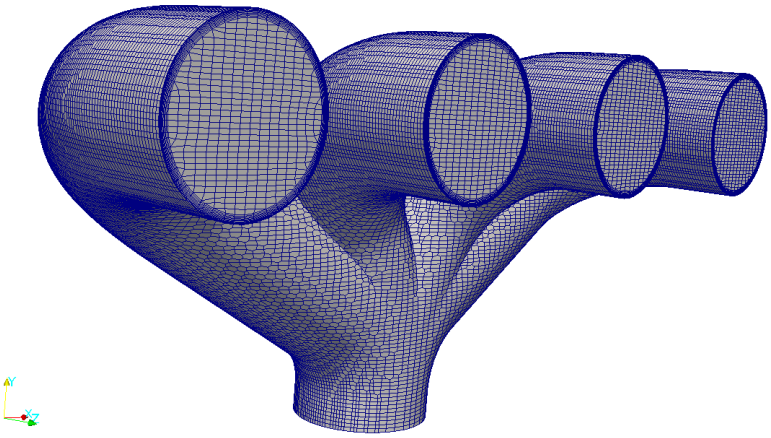


Figure 6.24: The mesh of the intake manifold.

6.2.2 Setting Up the Optimization

The boundary condition of the pressure field at the inlet and all wall patches is a zero Neumann condition. At the outlet, a zero Dirichlet condition is imposed. For the velocity, zero Neumann condition is imposed at the outlet, while at the wall patches, no-slip conditions are imposed. At the inlet, the velocity distribution is computed in order to have enough massflow to accommodate a four-cylinder internal combustion engine of $1.4L$ of volume running at $2400RPM$. This results in a velocity (normal to the inlet) with magnitude equal to approximately $15m/s$ and a Reynolds number of $70K$. For the turbulence, the Spalart-Allmaras model (as described in Sec. 2.3 and [**?**]) is used.

The target of this test case is to minimize the total pressure losses J_{Pt} between the inlet and the outlets (Eq. 2.19). Therefore, the relevant adjoint equations are solved.

6.2.3 The Optimization Run

The Geometry Morphing method is setup so that all adjacent patches retain C_1 continuity at their trimming curves with the exception of the curves at the inlet and the outlets. There, C_0 continuity is imposed and, moreover, the condition that their adjacent patches remain perpendicular to the circular disks should be met.

By doing so, an optimization run of 20 cycles is performed which produces a drop of 1.9% in the objective function (Fig. 6.25).

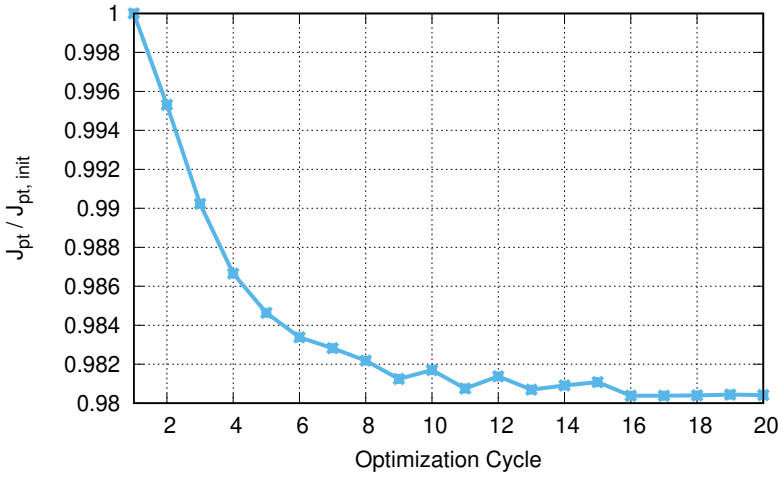


Figure 6.25: Convergence history of the optimization run.

The comparison of the initial and optimized CAD shapes can be seen in Fig. 6.26. The developments of Chap. 3-5 of this thesis were combined to perform

a full-scale CAD-based optimization. The input geometry was designed by the author and was provided in STEP format while, the optimal geometry was also made available in STEP format. Due to being self-designed, there are no other works done on this case with which one can compare the results. However, this case can serve as an excellent benchmark to show that the coupling of CAD to CFD and optimization can be performed automatically.

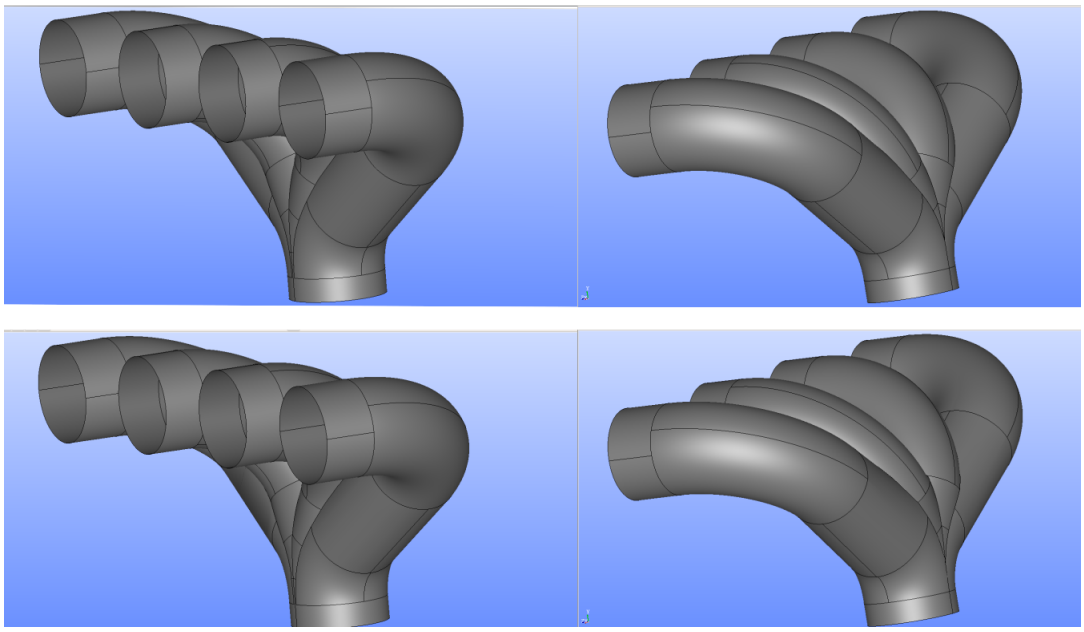


Figure 6.26: The initial (top) and the final (bottom) CAD shape after 20 optimization cycles. The front (left) and the rear (right) view of the manifold can be seen. Mainly, the displacement of the model is located at the two middle ducts leading to the corresponding outlets.

6.3 The ERCOFTAC UFR 4-06 Diffuser

The ERCOFTAC Conical Diffuser test case is a swirling boundary layer circular pipe, developing in a conical diffuser. A description of the measurements made by Clausen, Koh and Wood is available in [**?**]. The experimental set-up is such that the inlet swirl prevents boundary layer separation, without though recirculation in the core of the flow. Experimental results are available in the ERCOFTAC Classic database [**?**]. The test case was included in the ERCOFTAC Workshop on Data Bases and Testing of Calculation Methods for Turbulent Flows held in Karlsruhe in 1995 [**?**].

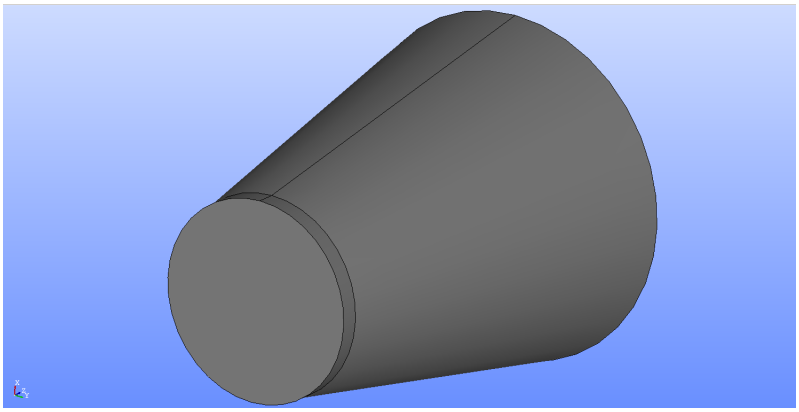


Figure 6.27: The CAD geometry of ERCOFTAC Conical Diffuser in STEP format.

Fig. 6.27 shows a 3D view of the CAD geometry. The inlet is located at the beginning of the small cylindrical section. In the experiments, an extended version of the inlet pipe was used to generate a plug flow with a solid body rotation/swirl. Several 2D and 3D flow computations were submitted to the ERCOFTAC Workshop from 1995 to 1997 [**?**, **?**, **?**]. More recently, results were presented using OpenFOAM [**?**]. Computations reflecting more the experimental set-up were presented in 2006 [**?**]. Remarks on the work of this case study were presented by [**?**, **?**]. However, no optimization cases have been reported (to the author's knowledge) thus far.

6.3.1 The CFD Mesh

Despite the simplicity of the domain of the ERCOFTAC Diffuser, it is important to have fine-enough boundary layers due to the swirl of the inlet velocity. The triangulation algorithm of Chap. 3 is run on the BRep geometry of Fig. 6.27. In this case, there is no necessity for a Shape Healing algorithm. The background grid is generated on the geometry using the Delaunay method and the result is shown in Fig. 6.28.

The size map is then computed for parameters $d = 10$ and $g_R = 0.1$. The Advancing Front method generates the surface grid (Fig. 6.29). The process requires ≈ 1.5 secs on 8 Intel Core i7-6700HQ (2.60 GHz) CPUs and produced $1,870$ triangles averaging at ≈ 1300 tris/sec. Similarly to Chap. 6.1, the process was run in parallel using the OpenMP shared memory parallel programming protocol. The computational mesh is, then, defined based on the triangulated boundary.

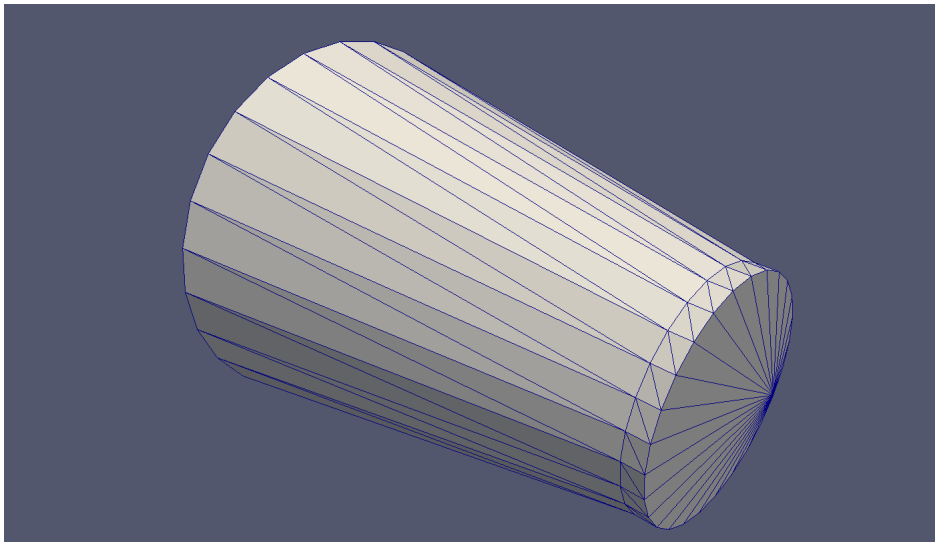


Figure 6.28: The coarse background grid generated on the ERCOFTAC Diffuser CAD geometry (Fig. 6.27).

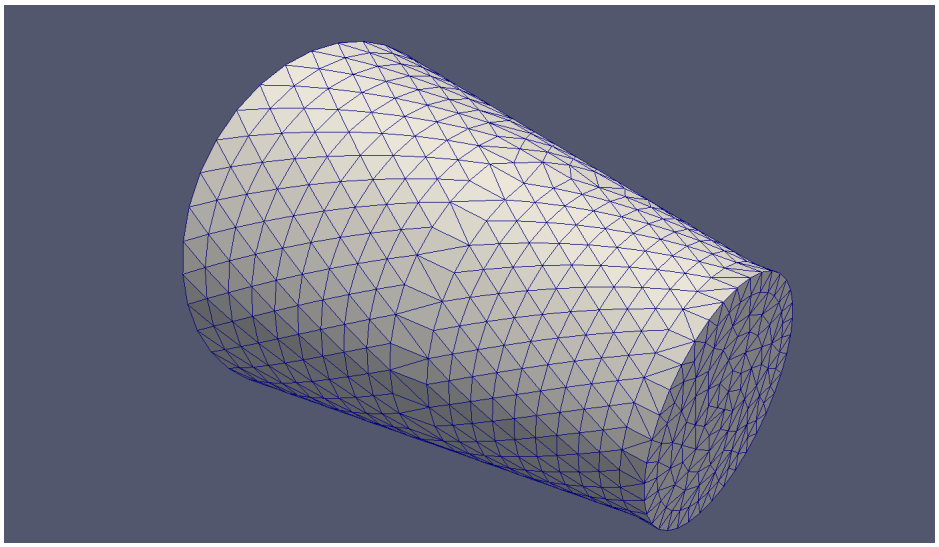


Figure 6.29: The triangulated ERCOFTAC Diffuser.

6.3.2 Setting up the Parameterization

In order to determine the parameterization, a few constraints must be taken into account. Firstly, in the exprerimental setup, the swirl is generated via a rotating honeycomb formation placed prior to the diffuser inlet. Thus, the inlet diameter must not be changed. Moreover, the assumption is made that the diffuser outlet is attached on a certain layout which would benefit from the gained static pressure.

Therefore, the outlet's diameter must not be changed either. Taking this into consideration, a choice is made to parameterize only the part of the wall that corresponds to the conical surface. The relevant NURBS patch along with its control points can be seen in Fig. 6.30 together with the NURBS basis functions corresponding to control point $(4, 2)$, computed for boundary mesh nodes in its vicinity.

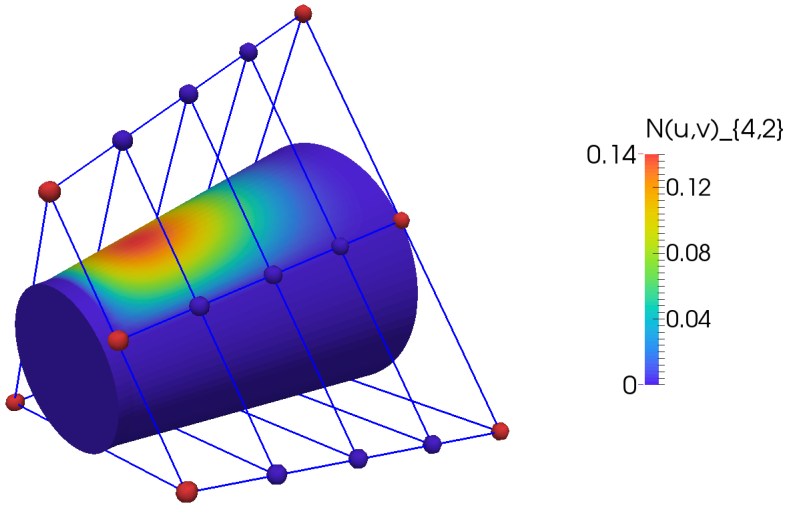


Figure 6.30: The ERCOFTAC Diffuser's along with the control point grid. The color coding is done for the NURBS basis functions of boundary mesh nodes for control point $(4, 2)$.

6.3.3 Flow Conditions and Optimization Targets

At the inlet, the average axial velocity is $U_z \approx 11.6m/s$ leading to a Reynolds number of $Re \approx 202,000$. According to the experimental data, the inlet is placed at coordinate $x = -25mm$, which is $75mm$ downstream of the swirl generator and $25mm$ upstream of the diffuser entrance. At this location the swirl is close to solid-body rotation with a nearly uniform axial velocity in the core region outside the boundary layers. The swirl number is $W_{max}/U_0 = 0.59$ where W_{max} is the maximal circumferential velocity. The velocity profile at the inlet is shown in Fig. 6.31.

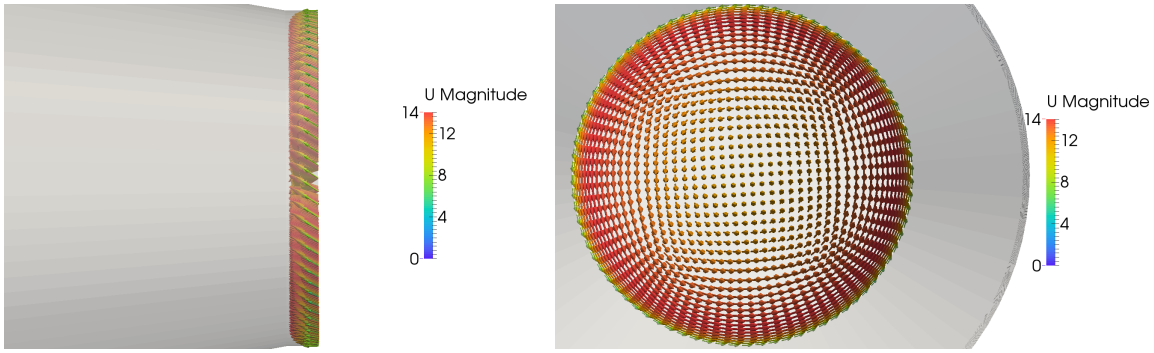


Figure 6.31: The velocity profile at the inlet from two perspectives: a side view (left) and normal to the inlet (right).

For the other boundary conditions, a zero Neumann condition is imposed for the velocity at the outlet, while for the pressure, there is a zero Neumann inlet condition and zero Dirichlet outlet condition.

Initially, a run is made with the target to minimize the total pressure losses J_{pt} (Eq. 2.19). Then, a second run is made with the target to maximize the static pressure gain J_{C_p} between the inlet and the outlet (Eq. 2.21). Finally, a weighted (SOO) is run where the target is to maximize J_{C_p} without allowing an increase in J_{pt} greater than 10%.

6.3.4 Minimizing Total Pressure Losses

Initially, an optimization run with the target to minimize the total pressure losses is made. This is done in order to assess the displacement of the shape and compare it to the one resulting from the pressure recovery minimization. To update the design variables as formed by the Geometry Morphing method (Chap. 4), the steepest descent method is used. After 20 optimization cycles, steepest descent converges to a solution with total pressure losses reduced by $8.6\ \%$ (Fig. 6.32). The updated CAD surfaces resulting from this optimization can be seen in Fig. 6.33.

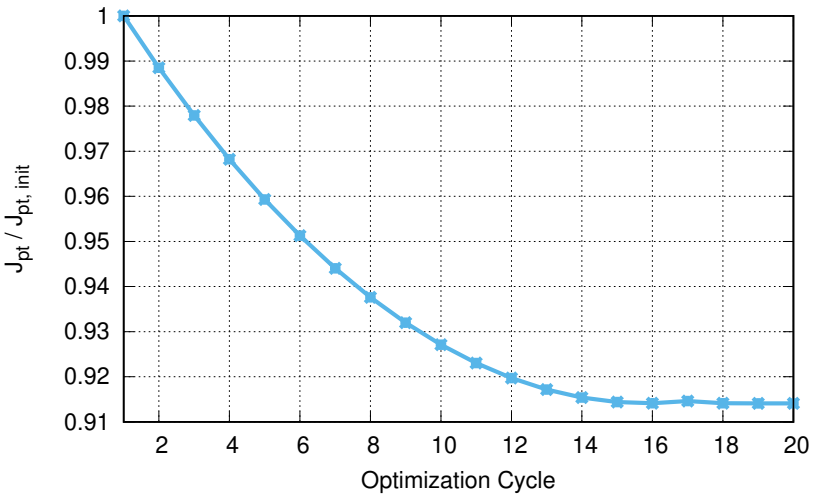


Figure 6.32: The convergence history of the total pressure losses minimization.

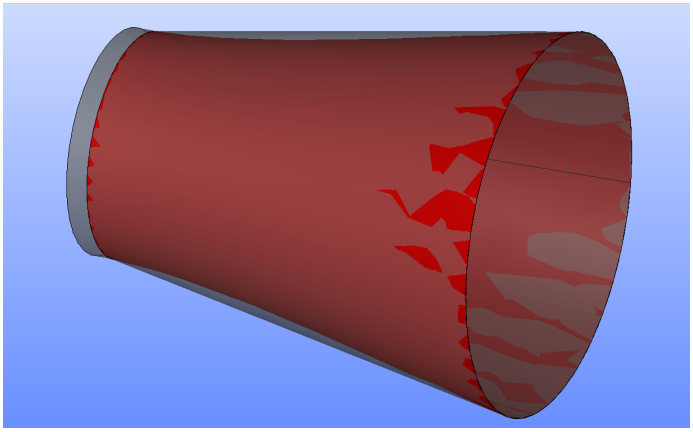


Figure 6.33: Isometric view of the diffuser CAD model. The original CAD model along with the non-displaceable surfaces can be seen in transparent grey and those resulting from the optimization can be seen in red.

6.3.5 Maximizing the Static Pressure Gain

The main objective of a diffuser is to make the flow gain in static pressure between inlet and outlet. An optimization is, therefore, run with the target to maximize that static pressure gain. Similarly to before, to update the design variables, steepest descent is used. After 20 optimization cycles, steepest descent converges to a solution with a static pressure gain approximately equal to $1.2\ \%$ (Fig. 6.34).

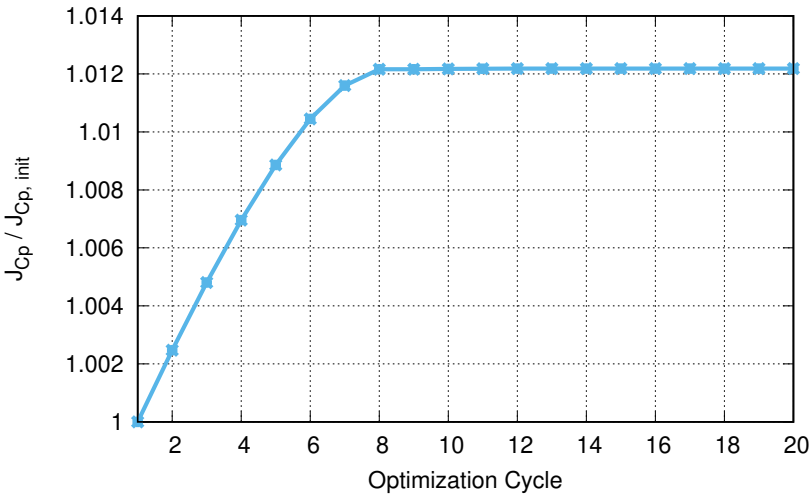


Figure 6.34: The convergence history of the static pressure gain maximization.

The updated CAD surfaces resulting from this optimization can be seen in Fig. 6.35. It is obvious that, for both optimization runs, the design update acts on the geometry near the inlet. In the case of total pressure losses minimization, the conical surface tends to shrink, while in the case of static pressure maximization, the conical surface tends to inflate.

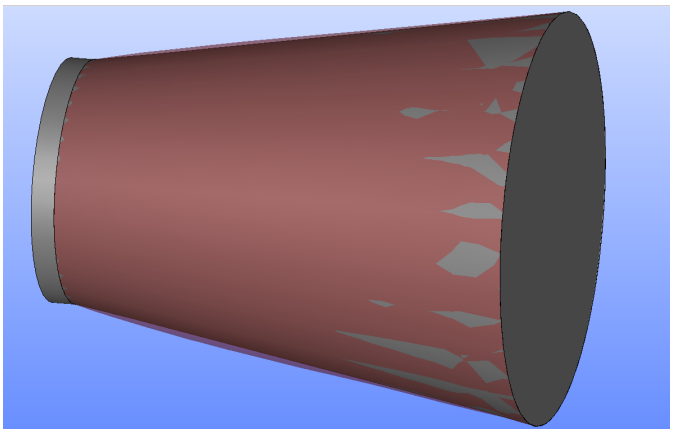


Figure 6.35: Isometric view of the diffuser CAD model. THe original CAD model along with the non-displaceable surfaces can be seen in grey and the surfaces resulting from the optimization in transparent red.

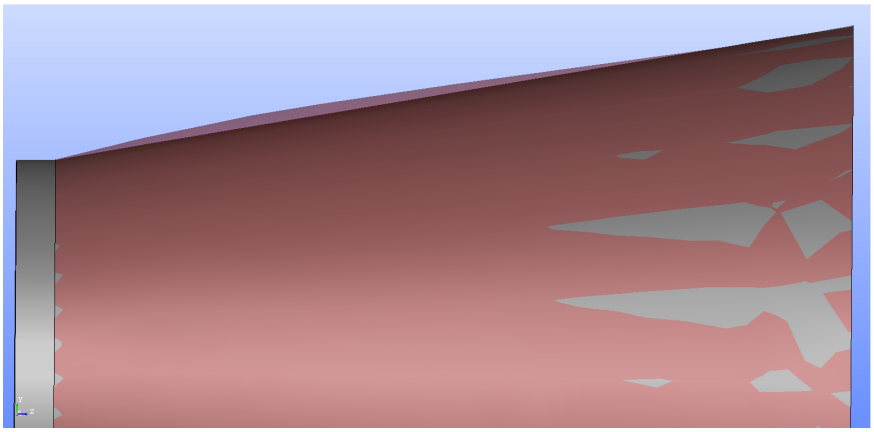


Figure 6.36: Side view of the diffuser CAD model. Coloring as in Fig. 6.35.

6.3.6 Weighted Single Objective Optimization

In this section, a run with both objectives is made. As shown in the previous sections, the design updates that result from the optimization of each objective, are of opposite directions. Therefore, a weighted SOO is used. The target here is to maximize the static pressure gain without allowing a drop in the total pressure losses which is greater than $10\ \%$. In order to achieve that, the weights are regulated accordingly. The weighted SOO becomes:

$$J_{SOO} = w_1 J_{C_p} + w_2 J_{Pt} \tag{6.1}$$

At each optimization cycle, two adjoint equations are solved to provide two sensitivity derivatives $\frac{\delta J_{C_p}}{\delta b_n}$ and $\frac{\delta J_{Pt}}{\delta b_n}$. Then, the SOO sensitivity derivatives are

$$\frac{\delta J_{SOO}}{\delta b_n} = w_1 \frac{\delta J_{C_p}}{\delta b_n} + w_2 \frac{\delta J_{Pt}}{\delta b_n} \tag{6.2}$$

where the weights chosen are $w_1 = -5$ and $w_2 = 1$.

The pressure gain increased by $0.87\ \%$ while the increase in the total pressure losses was $9.86\ \%$. The decrease in the w-SOO objective was equal to 0.83%. The convergence history can be seen in 6.37.

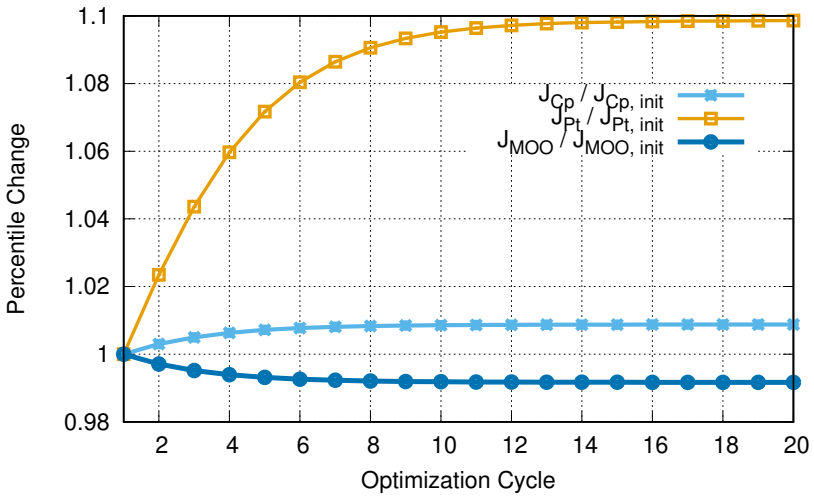


Figure 6.37: The convergence history of the MOO.

As expected, the optimal geometry tends to approach the one resulting from the run in Sec. 6.3.5, even though the resulting shape is much closer to the initial geometry (Fig. 6.38). All the geometries of the ERCOFTAC diffuser are in the standardized Step format (input geometry and the results of the runs). The CFD simulations performed are in agreement with the experimental data provided.

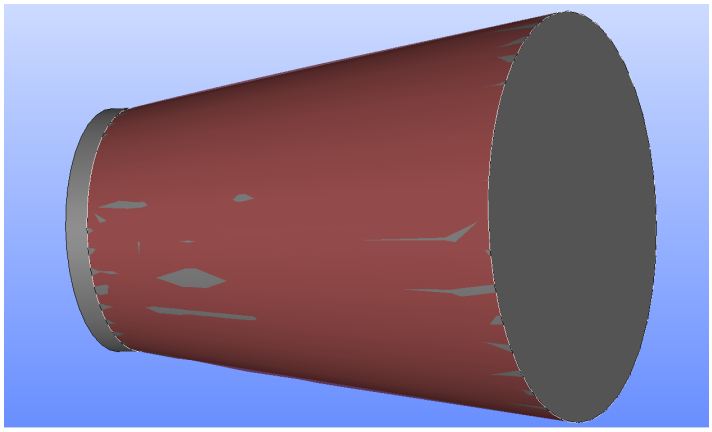


Figure 6.38: Isometric view of the diffuser CAD model. The baseline CAD model along with the non-moveable surfaces can be seen in grey and the surfaces resulting from the optimization can be seen in transparent red.

Chapter 7

Conclusions

This PhD thesis aimed at the development of methods and tools to integrate CAD within aerodynamic shape optimization workflows through their Boundary Representation format. With the methods and algorithms developed in this thesis, the CAD geometry of a shape to be designed can completely remain inside the optimization loop. To this end, a discretization method was developed to generate a grid with triangular elements on the surfaces of CAD models, a method to parameterize said models by using their BRep and removing any dependency on any CAD package and, finally, methods to constrain CAD models via their BRep surfaces. Henceforth, the main remarks and conclusions drawn during this development are discussed below.

The triangulation tool was developed from scratch by the author in an attempt to make it completely independent on any CAD package. The prerequisite is that the geometry of a CAD model becomes available through standard CAD files. The BRep in these files is treated with caution and a Shape Healing algorithm is performed to ensure that the model is geometrically and topologically intact. After Shape Healing, a fast Delaunay triangulation algorithm is used to compute a coarse (and low quality) background grid and, on it, optimal triangle size metrics are computed based on the model's curvature. Finally, an Advancing Front algorithm is proposed and programmed to be used in the parametric domain of each surface to compute the final triangulation which is ultmately differentiated w.r.t. the BRep design parameters (mostly NURBS control points). Most CAD-to-surface triangulation algorithms are implemented within commercial CAD packages which enables them to exploit the underlying parameterization and produce high quality triangulations. However, the possibility to connect these algorithms to the adjoint optimization is severed by the fact that the internal parameterizations of CAD packages are closed-source and, therefore, non-differentiable. The proposed method overcomes these setbacks and generates high quality triangulations. The triangulation speed is generally high averaging at 1000 to 1500 triangles per second for the entire process at Intel Core i7-6700HQ CPUs. This includes

both the background grid / size map computation and the Shape Healing. Furthermore, it also includes the parsing of the standard CAD files. The background grid / size map computation is done analytically and is, therefore, fast. However, the Shape Healing can become very demanding from a computational point of view as a low quality CAD model may require the application of the Plate Energy Minimization algorithm multiple times. For this reason, the average speeds mentioned above are highly satisfactory. Ultimately, a CAD-to-surface grid tool of adequate quality can become the cornerstone of the entire optimization process by serving as the foundation on which the computational mesh and the relation of its boundary to the parameterization are built.

The Geometry Morphing method was developed in order to serve as a reliable parameterization scheme adapted for gradient-based optimization. CAD-based parameterizations (commercial or not) are not built to inherently serve shape perturbations. Internal CAD parameterizations re-compute the boundary surfaces of the CAD model when a parameter is changed which created topological discontinuities at a CAD model. BRep on the other hand is comprised by NURBS patches with trimmed boundaries. These patches are not "aware" of one another (at a geometric level) and their control points must be constrained to move in concert. The Geometry Morphing method tackles this challenge by imposing C_0 and C_1 continuity constraints between the NURBS patches of a model. The Jacobian matrix of all the imposed constraints is then analyzed in order to compute its Null Space basis which would produce vectors of design variables perturbations that would satisfy all the constraints. The analysis is done via Singular Vector / Value analysis. The QR decomposition technique is used for that analysis and is adapted to this process by including rank-revealing properties to it. The analysis reveals a number of control points which are constrained by the Null Space basis and some that are not. Ultimately, the parameters that are used for the shape perturbation are the unconstrained control points positions and the the Null Space basis parameters together. This proposed method requires a negligible amount of time to be employed. In all cases shown in this thesis, the time required to compute the parameterization along with the exports of the final STEP files ranged anywhere between $0.3\,\%$ to $0.5\,\%$ of the runtime for a single optimization cycle (solution of both primal $\&$ adjoint systems). With Geometry Morphing, the CAD can enter the optimization loop through the BRep format. In that sense, the Geometry Morphing overcomes two major setbacks: (a) The closed-source nature of CAD packages and (b) the surface continuity issues that would become apparent if the optimization was standard NURBS.

Both the triangulation method and the Geometry Morphing method can be used, in a broader spectrum, to accommodate different optimization-related necessities. For instance, the triangulation tool can serve strictly as a CFD mesh generation auxilliary tool. The Geometry Morphing method and the usage of the Null Space projection, can be used in the family of gradient projection methods

as a more general way to impose constraints. It suffices to say that even if, in this thesis, the method is implemented for linear constraints, it can also be generalized for non-linear ones. Furthermore, the developed software provides a way to re-define the number of NURBS patches that can move (either decrease them by defining non-moveable patches or increase them by breaking down a patch into multiple ones).

A key element of CAD-based parameterization is the imposition of constraints. Internal CAD parameterizations can accommodate constraints naturally. However, with BRep, the constraints must be imposed using the surface representations. Global constraints such as minimum (maximum) volume constraints are tested and perform adequately. However, point-wise constraints pose problems as the number of points to which the constraints are imposed become smaller than the number of design variables. For this reason, a method is proposed to deduce the number of constraints.

The developed algorithm accurately imposes no-penetration and curvature constraints for each point on a shape to be designed. The method's capabilities are fully tested in cases that range from academic to industrial. This method proved reliable even in cases where the initial solution does not satisfy the constraints (which is a requirement for constraint optimization algorithms such as SQP). The time required for the quantification of the single resulting constraint (evaluation and differentiation) is negligible in comparison to the time required to solve the primal and adjoint problems. The presented algorithm can be used with any parameterization regardless of the number of design variables. Spline-based parameterizations have been tested in the form of Volumetric NURBS control boxes as well as the Geometry Morphing method. In the context of bounding surface constraints, two types of bounding surfaces were used as constraints: surfaces that lie inside the shape to be designed and surfaces that lie outside of the shape to be designed and / or enclose it. The proposed method was able to handle both cases reliably. This study can be the motivation to address many interesting topics related to bounding surface, curvature or point-wise constraints in general. One example of using this method is drag minimization for airfoils which tends to make them thinner. Optimizing a ship's hull can be another example since in certain areas the hull must (to an extent) retain its shape. Lastly, the implicit integration of inequality constraints, by means of a penalty function, to generate a single equality constraint has proven to be successful and can be used in various other geometric constraint formulations such as curvature and thickness after formulating them as point-wise inequalities.

7.1 Novel Contributions

The novelties presented in this thesis are summarized below:

- The development of a surface triangulation tool that can be connected directly and seamlessly to gradient-based optimization is novel. Most triangulation tools either exploit internal CAD parameterizations or do not keep a link to CAD parameters for optimization purposes.
- A solution to the challenging task of incorporating CAD-based parameterizations within gradient-based optimization workflows is given in this work. The Geometry Morphing method is a novel approach to handling the BRep of the models. It is fast and reliable and can be applied to virtually any geometry.
- The imposition of point-wise constraints during CAD-based optimization is another novelty of this thesis. The proposed method manages, by means of a penalty function, to solve constrained optimization problems with more constraints than design variables. Traditional constrained optimization methods assume a priori that the number of design variables are less than the number of constraints. This would limit the possibility to use control boxes or NURBS as the number of control points would be much smaller than the number of imposed constraints.

7.2 Suggestions for Future Work

Some suggestions are made for the extension of the, thus far, presented developments.

- The triangulation tool can become even faster by improving the parallelization scheme. The shared memory scheme that is used means that each processor is charged with the triangulation of a number of patches, without taking the complexity of each patch into account. This can result in situations where some processors finish their jobs faster and then remain idle while the rest of the processors keep on computing.
- In terms of the Geometry Morphing method, the method for computing the rank revealing QR decomposition can be improved. In its current state, it is reliable but can sometimes suffer from numerical inaccuracies.
- Optimization under more point-wise constraints can be done by using the presented method. Examples can be the thickness of a CAD model at certain regions and other geometric constraints such the imposition of cylindricality or parallelism w.r.t. given objects.
- Investigate the imposition of more intuitive constraints like flatness, cylindricity etc. Such constraints are very closely related to a CAD package's parameterization. In that sense, it would be very interesting to attempt to impose such constraints.

Publications

The publications that resulted from the research conducted in this thesis are listed below:

Journal Papers:

- M. G. Damigos, E. M. Papoutsis-Kiachagias, and K. C. Giannakoglou. Adjoint variable-based shape optimization with bounding surface constraints. International Journal for Numerical Methods in Fluids, 93(3):590-609, 2021
- M. G. Damigos and E. De Villiers. Imposing C_0 and C_1 continuity constraints during cad-based adjoint optimization. International Journal for Numerical Methods in Fluids, 93(8):2468-2485, 2021

Conference Papers:

- M. G. Damigos, K. C. Giannakoglou, and E. De Villiers. Geometric constraint imposition on trimmed nurbs patches for adjoint optimization. In Proceedings of the 2018 6th European Conference on Computational Mechanics (Solids, Structures and Coupled Problems)(ECCM 6), 7th European Conference on Computational Fluid Dynamics (ECFD 7), 2018.

Chapters in Books:

- M. G. Damigos and E. De Villiers. Adjoint shape optimisation using model boundary representation. In Evolutionary and Deterministic Methods for Design Optimization and Control With Applications to Industrial and Societal Problems, pages 19-33. Springer, 2019.
- M. G. Damigos and E. De Villiers. CAD-based parameterization for adjoint optimization. In OpenFOAM. Selected Papers of the 11th Workshop, pages 23-38. Springer, 2019

Conference Talks:

- M. G. Damigos, E. De Villiers, P. Geremia. CAD based parameterization and constraints for Adjoint optimization. VII European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS), Crete, Greece. Zenodo: https://doi.org/10.5281/zenodo.322869.

Appendix A

The Watson-Lawson Algorithm

In this appendix, the algorithm of computing a Delaunay triangulation is presented. The algorithm combines features of both Watson [**?**] and Lawson [**?**] procedures. The Watson procedure introduced the removal of triangles and retriangulating the regions where nodes were inserted. On the other hand, the Lawson procedure introduced edge-flipping. The Delaunay triangulation is created by introducing each point (one at a time) into an existing triangulation which is, then, updated.

The process starts by creating a "supertriangle" i.e. triangle that encloses all the data points $\vec{P}_i$ which are assumed to be available prior to the triangulation. In the case of parametric domain triangulation, the "supertriangle" must enclose the entire parametric domain. When a new point $\vec{P}_i$ is inserted into the triangulation, the triangle in which it belongs to is identified and three new triangles are created by connecting its vertices to $\vec{P}_i$. The original triangle is, then, deleted from the triangle list which means that the net gain of triangles is two. After the insertion of $\vec{P}$ and the creation of the new triangles, the triangulation is updated to Delaunay by using a swapping algorithm. In this, all the triangles which are adjacent to the edges opposite $\vec{P}$ are placed in a last-in, first-out stack. Each triangle is then unstacked, one at a time, and a check is made to determine if $\vec{P}$ lies inside its circumcircle. Should this be the case, then the triangle containing P as a vertex and the adjacent triangle form a convex quadrilateral with the diagonal drawn in the wrong direction, which must be replaced by the alternate diagonal to preserve the structure of the Delaunay triangulation. The swapping procedure replaces two old triangles with two new triangles with no net gain in the triangle list. Once the swap is completed, any triangles which are now opposite $\vec{P}$ are added to the stack. The next triangle is then unstacked and the whole process is repeated until the stack is empty and this results in a new Delaunay triangulation containing the point $\vec{P}$. An illustration of the swapping procedure is shown in Fig. A.1.

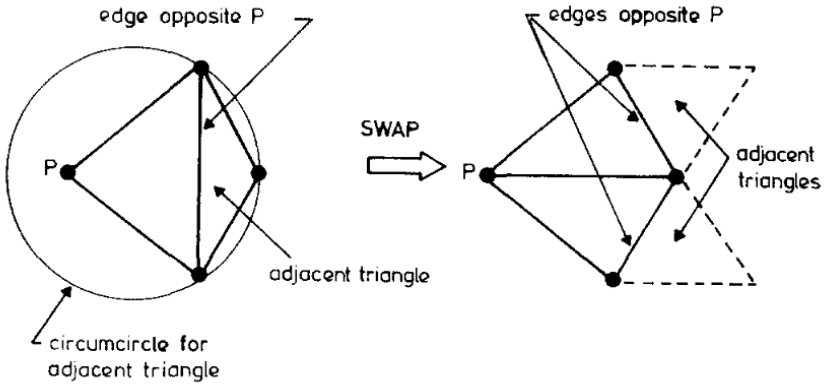


Figure A.1: The swapping procedure

Note that if $\vec{P}$ lies outside (or on) the circumcircle of a stacked triangle, then no action is taken and the triangle is skipped. It has been shown by Lawson that this iterative algorithm must result in a Delaunay triangulation and will always terminate after a finite number of swaps. Typically, only a few levels of swaps are necessary for each edge which is initially opposite $\vec{P}$ and the process is thus efficient. After all necessary points have been added to the triangulation, the final Delaunay triangulation is obtained by removing all of the triangles that contain one or more of the "supertriangle" vertices. Any vertex which appears in these deleted triangles, but is not a supertriangle vertex, must lie on the boundary of the triangulation. Since the insertion of each new point into the triangulation creates two new triangles, the final number of triangles, including those formed with the vertices of the supertriangle, is $2N + 1$, where N is the total number of vertices.

Appendix B

Advancing Front Validity Tests

During the Advancing Front algorithm and the creation of a new triangle, in order to ensure the quality and the convergence of the triangulation, the new triangle must pass certain geometric tests. In this appendix, the three tests to which a new triangle is subjected, are examined.

Assume a Front Edge e with first Front Point $\vec{A}$ and final Front Point $\vec{B}$. Furthermore, assume that a triangle creation is attempted using e and a third Front Point $\vec{P}$ .The first test that is made, is the zero area test. This test ensures that the triangle is not too skinny or inverted which in turn ensures that the Advancing Front orientation will be proper. For this test, the area of the potential triangle $\widehat{ABP}$ is computed in the parametric space as

$$a = \begin{vmatrix} u_B - u_A & u_P - u_A \\ v_B - v_A & v_P - v_A \end{vmatrix} \tag{B.1}$$

If $a \leq 10^{-12}s$ (where s is the optimal size of the triangulation at the midpoint of $\vec{AB}$), then either a is almost zero or a is negative. In the first case, the triangle will be degenerate and in the second it will be inverted. Therefore, the first test is $a > 10^{-12}s$.

The second test is to check if neighbouring points of triangle $\widehat{ABP}$ are enclosed by it or lie on it (Fig. B.1). To perform this test, the list of neighbouring points acquired during the main triangulation algorithm of Chap. 3 is used. If none of these points are in the triangle or close to its bounds, then test is passed. The check is done by computing the barycentric coordinates (α, β, γ) of a point $\vec{C}$ of the said list w.r.t. $\widehat{ABP}$.

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ u_A & u_B & u_P \\ v_A & v_B & v_P \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ u_C \\ v_C \end{bmatrix} \tag{B.2}$$

The strict form of the test is $0 \leq (\alpha, \beta, \gamma) \leq 1$. In our case it is relaxed for quality purposes by making it $-\epsilon \leq (\alpha, \beta, \gamma) \leq 1 + \epsilon$ for some small value of ϵ. In all the cases shown in this thesis, $\epsilon = 10^{-5}$.

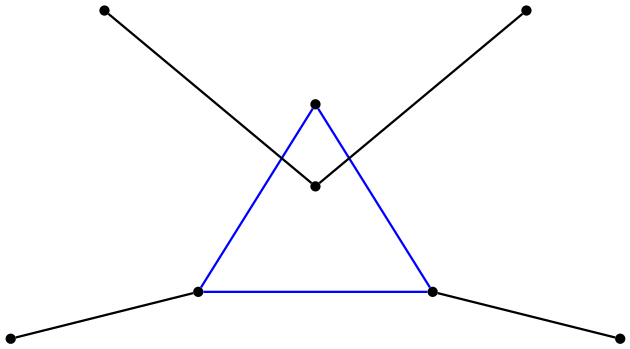


Figure B.1: A situation where the second test would fail: The new potential triangle (blue) encloses at least one Front Point. The triangle is therefore invalid.

The third test is performed in order to identify potential intersections of the new triangle with neighbouring Front Edges (Fig. B.2). This is necessary to identify invalidities during which, a Front Edge intersects with two sides of the new triangle (meaning that the second test will not identify them). The edges touching the Front Points of the list mentioned above are tested by performing an edge-to-edge intersection check between them and $\vec{MP}$ ($\vec{M}$ is the midpoint between $\vec{A}$ and $\vec{B}$). Assuming that the tested edge is $\vec{CD}$, the following quantities are computed

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} u_P - u_M & u_C - u_D \\ v_P - v_M & v_C - v_D \end{bmatrix}^{-1} \begin{bmatrix} u_C - u_A \\ v_C - v_A \end{bmatrix} \tag{B.3}$$

If $0 \leq (\lambda_1, \lambda_2) \leq 1$, then the two segments intersect. Therefore, for every tested edge $\vec{CD}$, it must be true that $\lambda_i \geq 1$ and $\lambda_i \leq 0$. Similarly to the second test, these two conditions are relaxed by a small value $\epsilon = 10^{-5}$ and they become $\lambda_i \geq 1 - \epsilon$ and $\lambda_i \leq \epsilon$.

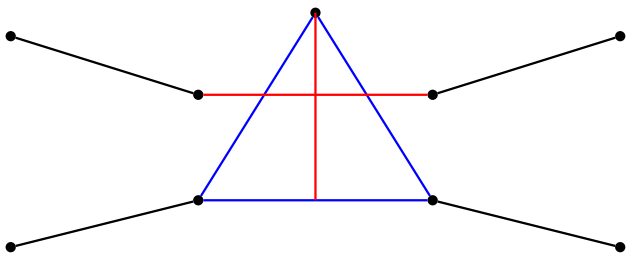


Figure B.2: A situation where the third test would fail: The new potential triangle (blue) is not valid because its median line intersects with a Front Edge (both in red).

Appendix C

Point Inversion in Curve, Surface and Volume NURBS

In various instances throughout this thesis, a 3D point $\vec{P}$ must be projected to the parametric domain of a piece of geometry. This can be a surface, a curve or a 3D hull object during an FFD process. This process is called point inversion and is basically the minimization of a distance function from the specific piece of geometry to $\vec{P}$.

Assuming that the parametric equations of curves, surfaces and volumes are given by $\vec{C}(u)$, $\vec{S}(u,v)$ and $\vec{V}(u,v,w)$ respectively, then the equations to be solved are

$$(\vec{C}(u) - \vec{P}) \cdot \vec{C}_u(u) = 0 \tag{C.1}$$

for curves,

$$(\vec{S}(u,v) - \vec{P}) \cdot \vec{S}_u(u,v) = 0 \tag{C.2}$$
$$(\vec{S}(u,v) - \vec{P}) \cdot \vec{S}_v(u,v) = 0 \tag{C.3}$$

for surfaces and

$$\vec{V}(u,v,w) - \vec{P} = \vec{0} \tag{C.4}$$

for volumes. In all three cases, u and/or v and/or w denote the parametric coordinates. Similarly to throughout this thesis, u, v, w indexing, denotes parametric differentiation.

For the cases of curves and surfaces, the dot products with tangent vectors exist so as to enable easier solution of the equations when $\vec{P}$ is not directly on

the geometry. For the case of volumes, this is not necessary because the points projected are usually part of the hull object.

All three equations are solved using the Newton-Raphson method **[?]** after appropriate initialization u_0, v_0, w_0. For Eq. C.1:

$$u_{n+1} = u_n - \eta_n \frac{(\vec{C}(u_n) - \vec{P}) \cdot \vec{C}_u(u_n)}{(\vec{C}(u_n) - \vec{P}) \cdot \vec{C}_{uu}(u_n) + \vec{C}_u(u_n) \cdot \vec{C}_u(u_n)} \tag{C.5}$$

where η_n denotes a line search step and n is the previous iteration number. Similarly, for surfaces:

$$\begin{bmatrix} u \\ v \end{bmatrix}_{n+1} = \begin{bmatrix} u \\ v \end{bmatrix}_{n} - \eta_n M(u_n, v_n)^{-1} \begin{bmatrix} (\vec{S}(u_n, v_n) - \vec{P}) \cdot \vec{S}_u(u_n, v_n) \\ (\vec{S}(u_n, v_n) - \vec{P}) \cdot \vec{S}_v(u_n, v_n) \end{bmatrix} \tag{C.6}$$

where

$$M(u_n, v_n) = \begin{bmatrix} (\vec{S}(u_n, v_n) - \vec{P}) \cdot \vec{S}_{uu}(u_n, v_n) + E_n & (\vec{S}(u_n, v_n) - \vec{P}) \cdot \vec{S}_{uv}(u_n, v_n) + F_n \\ (\vec{S}(u_n, v_n) - \vec{P}) \cdot \vec{S}_{uv}(u_n, v_n) + F_n & (\vec{S}(u_n, v_n) - \vec{P}) \cdot \vec{S}_{vv}(u_n, v_n) + G_n \end{bmatrix}$$

E_n, F_n, G_n are the coefficients of the first fundamental form of the surface at the parametric coordinates (u_n, v_n). Finally, for volumes:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix}_{n+1} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}_{n} - \eta_n \begin{bmatrix} V_u^x(u_n, v_n, w_n) & V_u^x(u_n, v_n, w_n) & V_u^z(u_n, v_n, w_n) \\ V_v^x(u_n, v_n, w_n) & V_v^y(u_n, v_n, w_n) & V_v^z(u_n, v_n, w_n) \\ V_w^x(u_n, v_n, w_n) & V_w^y(u_n, v_n, w_n) & V_w^z(u_n, v_n, w_n) \end{bmatrix}^{-1} \begin{bmatrix} V^x(u_n, v_n, w_n) - P^x \\ V^y(u_n, v_n, w_n) - P^y \\ V^z(u_n, v_n, w_n) - P^z \end{bmatrix} \tag{C.7}$$

where the exponents x, y, z denote the cartesian directions.