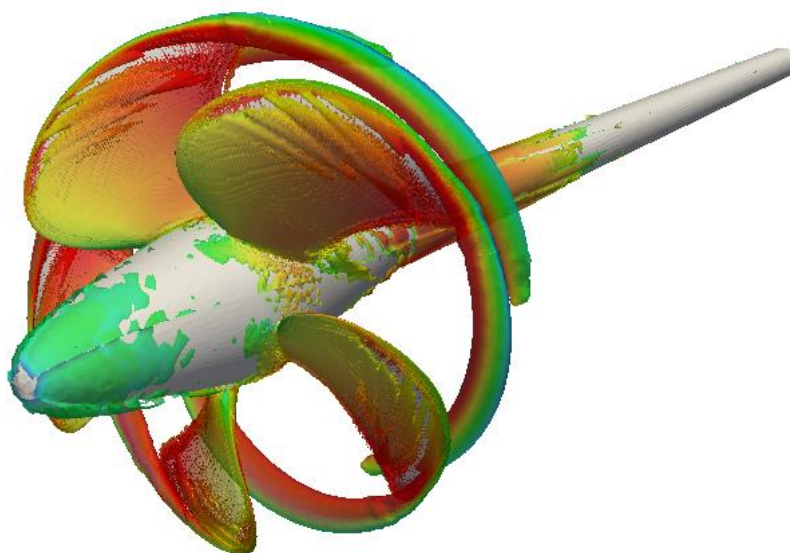




**NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF NAVAL ARCHITECTURE AND MARINE ENGINEERING
DIVISION OF SHIP HYDRODYNAMICS**

DIPLOMA THESIS MARINE PROPELLER OPTIMIZATION USING OPEN SOURCE CFD



Theofanis Papakonstantinou

Thesis Supervisor: Prof. G. Grigoropoulos

Committee Member: Prof. G. Politis

Committee Member: Prof. G. Papadakis

Athens, February 2019

Abstract

Due to the continuous increase in computing power, Reynolds-averaged Navier Stokes CFD simulations have become an integral part of propeller design. Moreover, advances in computer technology led to the development of modern CAD/CAE systems like CAESES that allow for parametric design and shape optimization by tightly coupling CAD to CFD.

The goal of this project is to investigate and validate open-source CFD tools that can be used to evaluate the open water characteristics of a propeller and can automated in order to support an optimization process. Then, develop a computationally cheap methodology inside CAESES that will eventually be used to optimize a propeller with the limited computational resources available. The efforts are focused on the difficulties of the automation the grid generation process, that still remains a bottleneck in optimization studies with CFD. The search leads to the selection of OpenFOAM to simulate the flow and cfMesh to generate the computational grid. The steady state Moving Reference Frame approximation is used to model the rotation and the flow is solved for one of the blades, using periodic boundaries. The turbulence is modelled with the $k-\omega$ SST turbulence model.

The method used is validated using the Potsdam Propeller Test Case (PPTC) by comparing simulation results to experimental data. A parametric model for the PPTC is created, CFD automation is achieved and the propeller is optimized, by modifying the radial distributions that define the blade shape. The optimization is performed using the Sobol sequence and the Tangent Search Method that are available in CAESES.

Finally, basic geometrical operations that are essential for the connection of a 3D propeller model to the CFD setup are scripted and automated inside CAESES, minimizing the manual effort needed to get a fast and rough calculation of any imported propeller blade. The scripts developed are validated with two powerboat propellers, by comparing the acquired CFD results to Vortex Lattice results.

Keywords:

CFD, OpenFOAM, MRF, Open Water Characteristics, Propeller design and optimization, Grid Generation, CAESES, $k-\omega$ SST, CFD automation, CAD/CAE systems

Περίληψη

Λόγω της συνεχώς αυξανόμενης υπολογιστικής ισχύος, οι Reynolds-averaged Navier Stokes CFD προσομοιώσεις έχουν γίνει αναπόσπαστο κομμάτι της σχεδίασης ελίκων. Επιπλέον, οι εξελίξεις στην τεχνολογία υπολογιστών οδήγησαν στην ανάπτυξη μοντέρνων συστημάτων CAD/CAE όπως το CAESES που παρέχει δυνατότητες παραμετρικής σχεδίασης και βελτιστοποίησης, η οποία επιτυγχάνεται με την σύζευξη CAD με CFD.

Σκοπός της εργασίας αυτής είναι η αναζήτηση και επαλήθευση ελεύθερου λογισμικού που μπορεί να χρησιμοποιηθεί για τον υπολογισμό των χαρακτηριστικών έλικας σε ελεύθερη ροή αλλά και να αυτοματοποιηθεί για να υποστηρίξει μια διεργασία βελτιστοποίησης. Έπειτα, να αναπτυχθεί μια υπολογιστικά φθηνή μεθοδολογία μέσα στο CAESES που τελικά θα χρησιμοποιηθεί για την βελτιστοποίηση μιας έλικας, λαμβάνοντας υπόψη την περιορισμένη υπολογιστική ισχύ που διατίθεται. Οι προσπάθειες εστιάζονται στις δυσκολίες της αυτοματοποίησης της γένεσης του υπολογιστικού πλέγματος, διεργασία που παραμένει εμπόδιο στις μελέτες βελτιστοποίησης με CFD. Η αναζήτηση οδηγεί στην επιλογή του λογισμικού OpenFOAM για την προσομοίωση της ροής και του εργαλείου cfMesh για την γένεση του υπολογιστικού πλέγματος. Η μέθοδος μόνιμης ροής Moving Reference Frame χρησιμοποιείται για να μοντελοποιήσει την περιστροφή και η ροή λύνεται για ένα από τα πτερύγια χρησιμοποιώντας περιοδικά σύνορα. Η τύρβη μοντελοποιείται με το μοντέλο τύρβης k- ω SST.

Για την επαλήθευση της μεθόδου χρησιμοποιείται η έλικα του Potsdam (PPTC) συγκρίνοντας τα αποτελέσματα των προσομοιώσεων με πειραματικά δεδομένα. Έπειτα αναπτύσσεται παραμετρικό μοντέλο για την έλικα αυτή, η αυτοματοποίηση των CFD εργαλείων επιτυγχάνεται και η έλικα βελτιστοποιείται, μεταβάλλοντας τις ακτινικές κατανομές που ορίζουν το σχήμα του πτερυγίου. Η βελτιστοποίηση γίνεται χρησιμοποιώντας την ακολουθία Sobol και την μέθοδο Tangent Search που παρέχονται στο περιβάλλον CAESES.

Τελικά, βασικές γεωμετρικές διεργασίες που απαιτούνται για την σύνδεση ενός τρισδιάστατου μοντέλου έλικας με τον επιλύτη μέσω του CAESES προγραμματίζονται και αυτοματοποιούνται, ώστε να ελαχιστοποιηθεί η χειροκίνητη προσπάθεια που απαιτείται για τον άμεσο υπολογισμό των χαρακτηριστικών ελεύθερης ροής οποιασδήποτε έλικας. Οι αλγόριθμοι που αναπτύσσονται εφαρμόζονται και επαληθεύονται σε δύο έλικες ταχύπλοων, συγκρίνοντας τα CFD αποτελέσματα με Vortex Lattice αποτελέσματα που διατίθενται για τις έλικες αυτές.

Λέξεις κλειδιά:

CFD, OpenFOAM, MRF, Χαρακτηριστικά ελεύθερης ροής, Σχεδίαση και βελτιστοποίηση έλικας, Γένεση πλέγματος, CAESES, k- ω SST, Αυτοματοποίηση CFD διεργασιών, Συστήματα CAD/CAE

Acknowledgements

This thesis is submitted in partial fulfillment of the requirements for the degree of Naval Architecture and Marine Engineering at the National Technical University of Athens. It was undertaken at Friendship Systems AG, Potsdam, Germany, developer of the software CAESES.

Firstly, I would like to thank my supervisor at NTUA, Professor Grigoris Grigoropoulos, who gave me the chance to work on such an interesting and challenging topic. I would also like to thank Professor Gerasimos Politis for his guidance and the valuable input he provided throughout the project as well as Professor George Papadakis for his support. The founders of the company, Dr. Stefan Harries and Claus Abt, for their support, the pleasant environment they provided and for introducing me to the fantastic world of simulation-driven design. Special thanks also go to the whole team of Friendship Systems especially to Heinrich von Zadow, Carsten Futterer and Paulo Macedo, this thesis would not have been possible without their help and guidance.

I express my deepest gratitude to my family, for their endless support throughout all my years of studies. My close friends and colleagues at NTUA, with whom I spent my studies and share countless memories. Finally, I would like to thank my beloved girlfriend Evangelia for all her love, understanding and support.

Theofanis Papakonstantinou, February 2019

Contents

Abstract	2
Περίληψη	3
Acknowledgements	5
List of Figures	8
List of Tables	10
Nomenclature	11
Abbreviations	11
Symbols	12
Operators	13
1 Introduction	14
1.1 Method Outline	16
1.2 Thesis Outline	17
2 Theory	18
2.1 Propeller Performance	18
2.2 Governing Equations	20
2.3 Turbulence and RANS	22
2.4 Turbulence Modelling: SST $k - \omega$ Model	23
2.5 Moving Reference Frame	25
2.6 Near Wall Treatment	27
2.7 Discretization	29
2.8 SIMPLE Algorithm	32
2.9 CAD - Parametric Modelling	34
2.10 Grid Generation	36
2.11 Propeller Optimization	39
3 Software	45
3.1 Geometry - Friendship Framework – CAESES	45
3.2 Solver - OpenFOAM	47
3.2.1 Introduction to OpenFOAM	47
3.2.2 Structure of an OpenFOAM Case	49
3.2.3 Compiling OpenFOAM Applications and Libraries	50

3.3 Grid Generation - CfMesh.....	52
4 Method.....	53
4.1 The Test Case - PPTC	53
4.2 Pre Processing - Geometry Cleanup	54
4.3 Domain Construction	56
4.3 Grid Generation.....	59
4.4 CFD Setup.....	65
4.4.1 Flow conditions	66
4.4.2 Boundary Conditions.....	66
4.4.3 Wall Treatment	69
4.5 Building the Parametric Model	70
4.6 CAESES - OpenFOAM coupling	76
4.7 Running the case in parallel	79
5 Validation and Results.....	80
5.1.1 Mesh quality metrics	80
5.1.2. Grid Independence Study	85
5.1.4 Convergence.....	87
5.1.5 Comparison with experimental data.....	90
5.2 Optimization Phase and results	100
5.3 Process Automation and Validation.....	106
6.1 Conclusion	112
6.2 Suggestions for future work.....	113
Bibliography	114
Appendix A.....	118
Appendix B	122

List of Figures

Figure 2.1 Open water test setup	18
Figure 2.2 Turbulent boundary layer	28
Figure 2.3 OpenFOAM cell definition [6] (left), face definition [32] (right)	29
Figure 2.4 The SIMPLE algorithm flow chart [5]	33
Figure 2.5 Qualitative assessment of geometric modelling techniques [13]	35
Figure 2.6 Cell types used in CFD codes	36
Figure 2.7 Unstructured mesh using Pointwise for the PPTC	38
Figure 2.8 Fidelity of numerical methods for propeller simulation [17]	40
Figure 3.1 Phases of product development [11]	45
Figure 3.2 Meta Surface creation process in CAESES [29]	46
Figure 3.3 Overview of OpenFOAM structure [32]	48
Figure 3.4 Case directory structure [32]	49
Figure 3.5 C++ Class mechanism [32]	51
Figure 3.6 Directory structure for an application	51
Figure 3.7 Meshing process stages	52
Figure 4.1 CAD Geometry of the PPTC propeller	54
Figure 4.2 Detail of the trailing edge near the fillet	55
Figure 4.3 Complete watertight geometry including the shaft	55
Figure 4.4 First step of the domain construction	56
Figure 4.5 Domain construction	57
Figure 4.6 Domain construction	58
Figure 4.7 Final STL to be exported	59
Figure 4.8 Geometry preparation	60
Figure 4.9 Surface mesh on the propeller blade and shaft	61
Figure 4.10 Monitoring the mesh quality in Paraview	62
Figure 4.11 Prism layers at the wall	63
Figure 4.12 Refinements near the tip	63
Figure 4.13 Volume mesh around the blade without cylinder volume refinements	64
Figure 4.14 Volume mesh around the propeller with cylinder volume refinements	65
Figure 4.15 Boundary Conditions	66
Figure 4.16 Profile definition	70
Figure 4.17 Radial distributions (left), Parametric blade with tip (right)	71
Figure 4.18 Trailing edge fillet (left), Hydrofoil Section comparison (right)	72
Figure 4.19 Camber distribution	73
Figure 4.20 Thickness distribution	74
Figure 4.21 Pitch Distribution	75
Figure 4.22 Watertight parametric blade	75
Figure 4.23 Software connector	76
Figure 5.1 Aspect Ratio for a quadrilateral element [47]	80
Figure 5.2 Definition of non-orthogonality [48]	81

Figure 5.3 Skewness between neighbor cells [49].....	81
Figure 5.4 Concave faces (left), Concave cells (right).....	83
Figure 5.5 Low quality faces at the tip.....	84
Figure 5.6 Low quality faces at the fillet.....	84
Figure 5.7 Grid independence study	85
Figure 5.8 Y^+ values on the propeller blade corresponding to the medium mesh	86
Figure 5.9 Residuals of the simulation for $J=1.2$	87
Figure 5.10 Calculated forces for $J=1.2$	88
Figure 5.11 Calculated moments for $J=1.2$	88
Figure 5.12 Thrust coefficient for $J=1.2$	89
Figure 5.13 Torque coefficient for $J=1.2$	89
Figure 5.14 Comparison with experimental data.....	90
Figure 5.15 Pressure distribution on the suction side for advance coefficients from $J=0.2$ to $J=0.8$	92
Figure 5.16 Pressure distribution on the suction side for advance coefficients from $J=1$ to $J=1.4$	93
Figure 5.17 Pressure distribution on the pressure side for advance coefficients from $J=0.2$ to $J=0.8$	94
Figure 5.18 Pressure distribution on the pressure side for advance coefficients from $J=1$ to $J=1.4$	95
Figure 5.19 Streamline's behavior on the leading edge.....	96
Figure 5.20 Iso-surfaces of $Q = 20000$ colored with the velocity magnitude for $J=0.6$	97
Figure 5.21 Mesh plane cut	98
Figure 5.22 Axial velocity distribution for $J=0.6$	98
Figure 5.23 Pressure field around the blade on $y=\text{constant}$ for $J=0.6$	99
Figure 5.24 Vorticity Magnitude plot near the leading edge and tip for $J=0.6$	99
Figure 5.25 27 Sobol designs for the first DoE	101
Figure 5.26 Influence of parameter pitchMax on K_T	102
Figure 5.27 Influence of parameter ThicknessStart on open water efficiency	102
Figure 5.28 Camber1 values for every run	103
Figure 5.29 PitchMaxPos values for every run	103
Figure 5.30 Views of the Wageningen B-series and of the custom design [55].....	107
Figure 5.31 Wageningen B-Series geometry preparation (left), domain construction (right)	108
Figure 5.32 Open water chart for Wageningen B-series	108
Figure 5.33 Pressure distributions on the suction side (left) and the pressure side (right) for $J=0.6487$	109
Figure 5.34 Custom made geometry preparation (left), domain construction (right).....	110
Figure 5.35 Open water chart for Wageningen B-series	110

List of Tables

Table 4.1 Potsdam Propeller Test Case Geometry	53
Table 4.2 Flow conditions.....	66
Table 4.3 Boundary Conditions for pressure and velocity.....	67
Table 5.1 Number of cells of each type	82
Table 5.2 Comparison of with experimental data.....	91
Table 5.3 Design variables and bounds.....	100
Table 5.4 New design variables.....	104
Table 5.5 Open water characteristics obtained with OpenFOAM and Vortex lattice for the Wageningen B-Series propeller	109
Table 5.6 Open water characteristics obtained with OpenFOAM and Vortex lattice for the Wageningen B-Series propeller	111

Nomenclature

Abbreviations

Symbol	Definition
AMI	Arbitrary Mesh Interface
BREP	Boundary Representation
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CFD	Computational Fluid Dynamics
DOE	Design of Experiment
DNS	Direct Numerical Simulation
GPL	General Public License
ITTC	International Towing Tank Conference
LES	Large Eddy Simulation
MRF	Moving Reference Frame
NURBS	Non-uniform Rational Basis Spline
PFF	Propeller Free Format
PPTC	Potsdam Propeller Test Case
RANS	Reynolds Averaged Navier-Stokes
STL	Stereolithography
SST	Shear Stress Transport

Symbols

Symbol	Definition
α	Vector
β^*	Turbulence model coefficient
β_1	Turbulence model coefficient
δ_{ij}	Kronecker delta
η	Open water efficiency
γ	Blending factor
κ	Von-Karman constant
λ	Lamda viscosity
μ	Dynamic viscosity
ν	Kinematic viscosity
ω	Rotation vector, specific dissipation
φ	General variable
ρ	Density
σ	Cavitation number
τ	Stress tensor
D	Propeller diameter
J	Advance Coefficient
k	Turbulence kinetic energy
K_Q	Torque coefficient
K_T	Thrust coefficient
m	Mass
n	Number of propeller revolutions per second
p	Pressure
p_0	Ambient pressure
\mathbf{q}	Source term
Q	Torque
\mathbf{r}	Position vector
R	Reynolds stress tensor

Re	Reynolds number
S	Strain stress tensor
\mathbf{Sf}	Face area vector
T	Thrust
u^+	Dimensionless velocity
u_τ	Friction velocity
V	Volume
\mathbf{U}	Velocity
w	Weight factor
y^+	Dimensionless wall distance

Operators

$\nabla \mathbf{A}$	Gradient of vector \mathbf{A}
$\nabla \cdot \mathbf{A}$	Divergence of vector \mathbf{A}
$\nabla \times \mathbf{A}$	Curl of vector \mathbf{A}
$\mathbf{A} \cdot \mathbf{B}$	Inner product of vectors \mathbf{A} and \mathbf{B}
$\mathbf{A} \mathbf{B}$	Outer product of vector \mathbf{A} and \mathbf{B}
$\frac{D}{Dt}$	Total derivative in time
$\frac{\partial}{\partial t}$	Partial derivative in time

1 Introduction

For many years, the design and selection of a vessel's propeller has been a matter of investment cost. Nowadays, the increase of bunker price along with environmental concerns and regulatory requirements necessitate ship owners to reduce fuel consumption, leading propeller manufacturers in the search for more efficient propeller designs.

Propeller performance is expressed through the open water characteristics that are evaluated with the open water test, a model scale experiment in a towing tank. While these experiments are accurate and trustworthy, they are time consuming and expensive. Moreover, the flow behavior and performance prediction may differ between model scale and full scale due to the difference in the Reynolds number. Apart from the experimental methods, the hydrodynamic performance of the propeller can also be evaluated with numerical tools such as the lifting line method, lifting surface, boundary element methods (BEM) and CFD methods. Each method considers different levels of neglected effects that define its computational cost and accuracy. The selection of the appropriate method depends on the task, the phase in the propeller design process and the computational power available. Focusing on propeller optimization, BEM methods are used widely in the initial phases of propeller design. They are well tested, reliable and many design variants can be generated and evaluated automatically and fast. At the final stages of propeller design, more expensive viscous CFD computations are used to validate the BEM analysis and fine-tune the design.

CFD methods are being used more and more for propeller simulations, due to the rapid development of computational power. Reynold Averaged Navier-Stokes (RANS) methods have evolved to the point where they can be applied in industrial applications to predict the propeller's performance, include the propeller-hull interaction and run in full scale to investigate scale effects. RANS methods offer several advantages over the potential-flow methods. They can model the tip-vortex roll up and flow separation phenomena and predict propeller performance accurately at off-design conditions. However, even if computational power is not much of a problem, applying these methods requires a lot of effort and time to model and prepare the geometry for the simulation.

A number of CFD packages and techniques are available that can be used for propeller simulations. Commercial packages like STAR-CCM+ or ANSYS are widely used since they offer friendly graphical user interfaces, advanced meshing algorithms and tutorials. However, the cost for an annual license may be prohibitive. The CFD package used in this thesis is OpenFOAM, an open-source CFD toolkit based on C++. Due to its open-source nature, OpenFOAM is used extensively by both the academia and the industry for complex propeller simulations, usually along with commercial grid generators. There are several techniques that can be used to simulate rotating machinery with CFD. Three popular approaches are the sliding grid, mixing plane and the frozen rotor approach. The frozen rotor approach is used in this work, also called the Moving Reference Frame (MRF) method. It is the most computationally cheap method of the three but is expected to yield reasonably accurate results [1]. The flow around the operating propeller is unsteady. The MRF approach is a steady state approximation where different zones of the domain are given rotational speed without actually moving the mesh. To model the rotation, the governing equations

are written as observed from a relative coordinate system that follows the propeller movement. While this steady state approximation cannot capture the transient phenomena accurately, it can provide reasonable estimates of the flow behavior and accurate prediction of the propeller performance.

Recent developments in CAD/CAE systems allowed the use of parametric models. The CAE system used in this work is CAESES. It is capable of generating flexible parametric models that are CFD-ready, minimizing the time needed for the preparation of the geometry for simulation. Within CAESES, the parametric geometry can be coupled to an external CFD package that evaluates the performance of the model. By tightly coupling the parametric geometry to CFD, optimization algorithms can be employed and hundreds of designs can be generated and evaluated automatically. The potential gains of the optimization are limited only by the computational resources available. The main difficulties that are typically faced are related to the automation of the CFD processes. Regarding propeller optimization problems with CFD, an issue that persists is the automation of the grid generation process.

Marine propeller design and optimization is a very complex procedure involving numerous factors. Apart from the open water characteristics, strength requirements and cavitation need to be considered as well. The propeller optimization problem can be posed as multi-objective or single-objective, depending on the number of the factors that are considered. In the present work, no considerations regarding strength or cavitation are made and the problem is set as single-objective, to increase open water efficiency.

In this project, open-source tools are investigated and connected to the propeller geometry inside CAESES. The automation of the grid generation process is achieved with the use of cfMesh, an open-source unstructured grid generator that is compatible with openFOAM. Due to the limitations of cfMesh, generating meshes of acceptable quality has been a demanding process. Since a short computation time is sought, the flow is solved for one of the blades using periodic boundaries. The turbulence model used is the $k-\omega$ SST.

1.1 Method Outline

The Potsdam Propeller Test Case (PPTC) is used for the validation of the OpenFOAM – cfMesh configuration and the optimization study. The method initiates with the pre-processing of the PPTC geometry that is available online. The geometry contains open edges and gaps and requires treatment before the meshing process. In order to run the computations with the one blade passage approach, a flexible periodic domain is constructed using the advanced design capabilities of CAESES. The periodic domain, the propeller blade and shaft are combined for the mesh generation process. OpenFOAM and cfMesh are configured carefully to achieve optimal mesh quality, robustness and stability of the computations. Simulations are carried out for a wide range of advance coefficients and are compared to experimental data.

As a next step, a parametric model of the PPTC is constructed within CAESES to fully match the initial geometry. The execution of the grid generator and the solver is automated within a script, resulting in closed loop that can be used to optimize the propeller. Because of the limited computational resources, only a small number of parameters can be activated for the optimization. Therefore, the selected parameters should have maximum impact on the propeller blade. To make sure that the simulation results are independent from the mesh resolution a grid dependency study is carried out. To perform the optimization, the Sobol sequence is used to provide insight into the systems behavior and identify promising designs that are eventually fine-tuned with the Tangent Search Method. Both of the algorithms are available in CAESES.

In the end, using the programming language inside CAESES, processes that are needed to connect a 3D propeller blade to cfMesh and solver, like the shaft, hub and domain construction are automated within scripts. The scripts are tested with two powerboat propellers, where the CFD results are compared to Vortex lattice results.

1.2 Thesis Outline

This thesis report is divided into 6 chapters. Chapter 1 includes the introduction, goal of the project and method outline. Chapter 2 describes the theoretical background, including the propeller performance, CFD basics, CAD and grid generation techniques, as well as the concept of optimization. In chapter 3, the software used are presented. Chapter 4 describes the implementation, from pre-processing to grid generation and then from parametric modelling to software connection. In chapter 5, the results from the simulations carried out for the validation are presented, as well as the optimization phase. Finally, chapter 6 ends with the conclusions and suggestions for future work.

2 Theory

2.1 Propeller Performance

Propeller performance is evaluated by measuring the thrust (T) produced by the propeller and the torque (Q) needed to drive the propeller. Thrust and torque are measured in an open water test, which is carried out in a towing tank or a cavitation tunnel with the model scale propeller operating in uniform flow. The experiment is typically done by moving the propeller forward through the towing tank using a towing carriage as shown in Figure 2.1. The measurements are performed for a number of operating points depending on the loading of the propeller which is usually controlled by adjusting the speed of advance (U) and keeping the propeller revolutions constant.

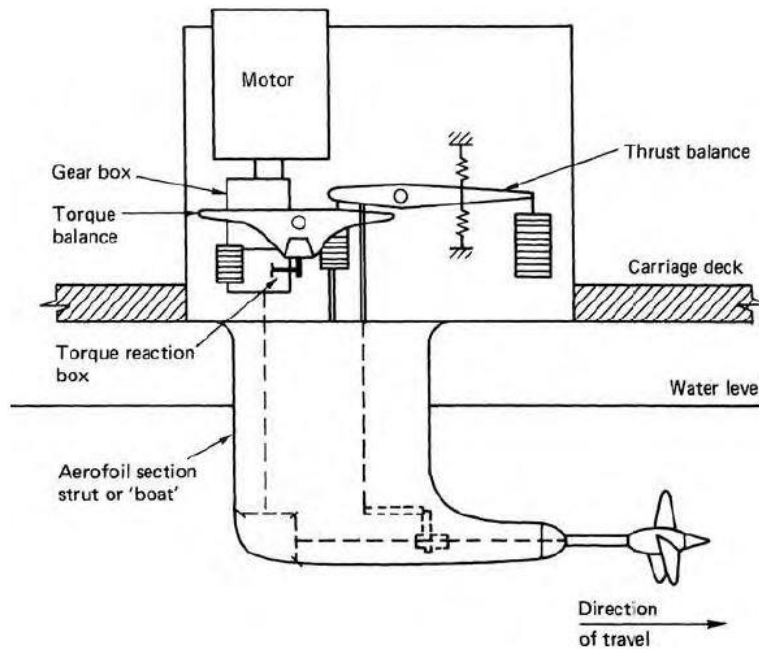


Figure 2.1 Open water test setup

By applying dimensional analysis and assuming that the free surface has no effect on the propeller performance, the measured thrust and torque are expressed as non-dimensional coefficients K_T and K_Q [2]. The coefficients K_T and K_Q are given as a function of a non-dimensional speed called advance coefficient (J).

$$K_T = \frac{T}{\rho n^2 D^4} \quad (2.1)$$

$$K_Q = \frac{Q}{\rho n^2 D^5} \quad (2.2)$$

$$J = \frac{U}{nD} \quad (2.3)$$

Above, the density of the water in the towing tank is denoted as ρ , n is the rotational speed of the propeller, D is the diameter of the propeller, T is the thrust, U the speed of advance, and Q the torque.

Furthermore, the open water efficiency (η) is introduced as the ratio of the thrust horsepower (THP) to the delivered horsepower (DHP) as seen below.

$$\eta = \frac{THP}{DHP} = \frac{T U}{2\pi n Q} = \frac{K_T}{K_Q} \frac{J}{2\pi} \quad (2.4)$$

The series of obtained characteristics K_T , K_Q and η are presented in an open water diagram where they are plotted on the ordinate, while J is plotted on the abscissa. The open water diagram contains all the information necessary to define the performance of a propeller at a particular operating condition and theoretically, it is applicable to any propeller having the same geometric form. This however, is not entirely true, since the open water characteristics are, to some extent, influenced by scale effects and cavitation.

Cavitation is a complex two-phase flow phenomenon. The operating propeller produces thrust by forming a variable pressure field around the blade surface, suction on the back of the blades and pressure on the faces of the blade. Depending on the loading of the propeller, the suction can be so great, that the absolute pressure approaches the vapor pressure of the water at a specific temperature and small cavities containing water vapor form. Once cavities enter regions of higher pressure, they will collapse violently producing noise, vibration and material erosion. Apart from the aforementioned problems, moderate levels of cavitation are not expected to cause significant impact on the propulsion performance [2]. Cavitation is expressed with the non-dimensional quantity, cavitation number, which is defined as

$$\sigma = \frac{p_0 - p_s}{\frac{1}{2}\rho U^2} \quad (2.5)$$

where p is the absolute static pressure at the shaft center line, and p_s is the vapor pressure at the ambient temperature.

As mentioned above, scale effects affect the propeller performance as well. Whilst open water tests are performed on model scale propellers with low Reynolds numbers (Re), full scale propellers operate at much higher Reynolds numbers. The Reynolds number is defined as

$$Re = \frac{U D}{\nu} \quad (2.6)$$

where ν is the kinematic viscosity of water. This difference on the Reynolds number corresponds to different viscous boundary layers. Turbulent flow over the blade surface is expected for full scale propellers however that is not the case for model scale propellers where laminar flow can prevail over significant parts of the blade. An analytical procedure is suggested by the 1978 ITTC committee, in order to quantify and take into account the viscous scale effects. [2] Nowadays, full scale CFD simulations are carried out as well, but the computational cost still remains high.

2.2 Governing Equations

As any mechanical system, the dynamics of fluids are governed by the conservation laws, specifically, conservation of mass, conservation of momentum and conservation of energy. In the marine CFD, the water temperature is assumed to be fixed and the energy equation is not included.

Conservation of mass

The conservation of mass law states that the quantity of mass is conserved over time. The continuity equation can be written as [3]:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = \frac{\partial \rho}{\partial t} + \frac{\partial \rho U_i}{\partial x_i} = 0 \quad (2.7)$$

Where ρ is the density and \mathbf{U} is the velocity vector. For the majority of marine CFD simulations, the flow is considered to be incompressible. Setting $\rho = \text{constant}$ in the above equation simplifies the equation to:

$$\nabla \cdot \mathbf{U} = \frac{\partial U_j}{\partial x_j} = 0 \quad (2.8)$$

Conservation of Momentum

The conservation of momentum states the fundamental second law of Newton. According to the law, the net force applied on the fluid element equals its mass times the acceleration of the element.

$$\mathbf{F} = m \frac{D\mathbf{U}}{Dt} \quad (2.9)$$

Where \mathbf{F} is a force vector and m is the mass and $\frac{D}{Dt}$ is the total time derivative expressed as:

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{U} \cdot \nabla = 0 \quad (2.10)$$

The conservation of momentum per unit volume can be written as:

$$\rho \frac{\partial (\mathbf{U})}{\partial t} + \rho \nabla \cdot (\mathbf{U} \mathbf{U}) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \rho \mathbf{q} \quad (2.11)$$

$$\rho \frac{\partial (U_i)}{\partial t} + \rho \frac{\partial (U_i U_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho q_i \quad (2.12)$$

Where the term $\rho \nabla \cdot (\mathbf{UU})$ is called convection term. For Newtonian fluids, the shear stress tensor can be expressed through the velocity gradients and viscosity. [3]

$$\tau = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) + \delta_{ij} \lambda \overbrace{(\nabla \cdot \mathbf{U})}^{=0} \quad (2.13)$$

where μ is the dynamic viscosity and λ the λ -viscosity that vanishes since the divergence of velocity is zero for incompressible flows. Kinematic viscosity ν is defined as:

$$\nu = \frac{\mu}{\rho} \quad (2.14)$$

Dividing the equation 2.13 by density gives:

$$\frac{\tau}{\rho} = \nu e \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (2.15)$$

The stress tensor can be written in a shorter form using the strain rate tensor S .

$$\tau/\rho = 2\nu S \quad (2.16)$$

Where the strain rate tensor S is defined as:

$$S = \frac{1}{2} (\nabla \mathbf{U} + (\nabla \mathbf{U})^T) \quad (2.17)$$

$$S_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (2.18)$$

Assuming that the density ρ is constant we can divide the equation 2.11 by ρ . Introducing the shear stress, we get the momentum equation in differential form for an incompressible Newtonian fluid, stating the conservation of momentum for every point in space:

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{UU}) = -\nabla \left(\frac{p}{\rho} \right) + \nabla \cdot (2\nu S) + \mathbf{q} \quad (2.19)$$

$$\frac{\partial (U_i)}{\partial t} + \frac{\partial (U_i U_j)}{\partial x_j} = -\frac{\partial \left(\frac{p}{\rho} \right)}{\partial x_j} + \frac{\partial (2\nu S_{ij})}{\partial x_j} + q_i \quad (2.20)$$

2.3 Turbulence and RANS

The phenomenon of turbulence was first identified by Osborne Reynolds. With experiments he performed in 1883, injecting a painted jet in the middle of pipe he observed that there is a laminar-turbulent transition between laminar and turbulent flow. This transition is affected by the Reynolds number. Fluid flows in industrial applications are high-Reynolds-number flows and turbulent. Turbulent flow is unsteady, three dimensional and chaotic. Turbulent flow is fully described by the Navier-Stokes equations that have been introduced above. There are three approaches to turbulence. The Direct Numerical Simulation (DNS), Large Eddy simulation (LES) and the Reynolds Averaged Navier-Stokes (RANS).

The DNS methods solve the Navier-Stokes equations for all the scales of turbulence. Turbulent eddies vary in scale. The small eddies require very fine grids, and the large eddies require large computational domains. Therefore, even if the DNS methods are valid and accurate, the simulation cost is too expensive for engineering applications. With LES methods, only the large-scale eddies are computed directly. The effect of the smaller eddies is taken into account using additional stresses obtained from the turbulence theory. Even though LES methods are not as expensive as the DNS methods, they still require great computational resources.

The RANS are the most commonly used in industrial applications and the present work. With the RANS approach, only the vortices of largest scale according to the size of the domain are resolved, and the rest of the turbulence is modelled with a turbulence model. With the RANS method, the governing equations are averaged in time. Velocity and pressure are decomposed into the mean (time-averaged) and fluctuating components.

$$\mathbf{U} = \bar{\mathbf{U}} + \mathbf{u}' \quad (2.21)$$

$$p = \bar{p} + p' \quad (2.22)$$

Where the time average of velocity \mathbf{U} is defined as:

$$\overline{\mathbf{U}(\mathbf{x})} = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T \mathbf{U}(\mathbf{x}, t) dt \quad (2.23)$$

And the time interval T has to be larger than the time scale of turbulence. Now, the time averaged velocity \mathbf{U} is introduced to the continuity and momentum equation. The continuity equation remains the same, since it is linear in velocity. However, the momentum equation contains the non-linear convection term. Therefore, the time averaged momentum equation becomes

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) = -\nabla \left(\frac{p}{\rho} \right) + \nabla \cdot (2\nu \mathbf{S}) + \mathbf{q} + \nabla \cdot \mathbf{R} \quad (2.24)$$

Including the new term \mathbf{R} is called Reynolds stress tensor, a symmetrical second order tensor. The Reynolds stress tensor can be written in index notation as $R_{ij} = -\overline{u'_i u'_j}$. This new term is problematic, since it introduces six unknowns in the equations. Now, the number of unknowns exceeds the number of equations resulting in the well-known closure problem. In order to close the system, new relations have to be defined.

According to the Boussinesq assumption, the Reynolds stresses can be expressed as:

$$R_{ij} = -\overline{u'_i u'_j} = 2\nu_T S_{ij} - \frac{2}{3}k\delta_{ij} \quad (2.25)$$

Thus, turbulence is modelled with two terms. The turbulent viscosity ν_T , and the turbulent kinetic energy, k . δ_{ij} is the Kronecker delta, ν_T is a scalar object, dependent of the flow field and the kinetic energy k is defined as:

$$k = \frac{1}{2}\overline{u'_k u'_k} \quad (2.26)$$

2.4 Turbulence Modelling: SST k – ω Model

The SST k- ω model was first introduced in 1994 by F.R. Menter. This turbulence model combines the advantages of both the k- ω and k- ϵ models. It includes two equations to model the effect of turbulence, one is for the turbulence kinetic energy, k , and the second one for specific dissipation rate, ω . The turbulent viscosity can be defined as:

$$\nu_T = \frac{k}{\omega} \quad (2.27)$$

The model equation for the turbulence kinetic energy k :

$$\frac{\partial k}{\partial t} + \nabla \cdot (Uk) - (\nabla \cdot U)k - \nabla \cdot (D_{k,eff} \nabla k) = \min\{G, c_1 \beta^* k \omega\} - \beta^* \omega k \quad (2.28)$$

The model equation for the turbulence kinetic energy ω :

$$\frac{\partial \omega}{\partial t} + \nabla \cdot (U\omega) - (\nabla \cdot U)\omega - \nabla \cdot (D_{\omega,eff} \nabla \omega) = \frac{\gamma(F_1)G}{\nu_T} - \beta(F_1)\omega^2 - (F_1 - 1)CD_{k,\omega} \quad (2.29)$$

In equation 2.28, the term G is called turbulent kinetic energy production and is given as:

$$G = 2\nu_T |S|^2 \quad (2.30)$$

In equations both 2.28 and 2.29 the coefficients $D_{k,eff}$ and $D_{\omega,eff}$ are defined as below:

$$\begin{aligned} D_{k,eff} &= a_k(F_1)v_T + \nu \\ D_{\omega,eff} &= a_\omega(F_1)v_T + \nu \end{aligned} \quad (2.31)$$

The coefficients $\gamma(F_1)$, $a_k(F_1)$, $a_\omega(F_1)v_T$, $\beta(F_1)$ are filtered between the model coefficients by the function F_1 according to equation 2.33.

$$(\gamma \ \alpha_k \ \alpha_\omega \ \beta)^T = F_1(\gamma \ \alpha_k \ \alpha_\omega \ \beta)_1^T + (1 - F_1)(\gamma \ \alpha_k \ \alpha_\omega \ \beta)_2^T \quad (2.32)$$

With the following values:

$$\begin{aligned} \gamma_1 &= 0.5532 & a_{k1} &= 0.85034 & a_{\omega1} &= 0.5 & \beta_1 &= 0.075 \\ \gamma_2 &= 0.4403 & a_{k2} &= 1.0 & a_{\omega2} &= 0.85034 & \beta_2 &= 0.0828 \end{aligned}$$

The coefficient β^* present in equation 2.28 is constant with a value $\beta^* = 0.09$ and $c_1 = 10$. The switching between the ω - and ε -equations is

$$F_1 = \tanh(\Gamma^4) \quad (2.33)$$

Where

$$\Gamma = \min \left\{ \min \left[\max \left(\frac{\sqrt{k}}{\beta^* \omega y}; \frac{500\nu}{\omega y^2} \right); \frac{4a_{\omega2}k}{CD_{k\omega}y^2} \right]; 10 \right\} \quad (2.34)$$

$$CD_{k\omega} = \frac{2a_{\omega2}}{\omega} (\nabla k \cdot \nabla \omega) \quad (2.35)$$

Where $CD_{k\omega} \geq 1 \cdot 10^{-10}$. The turbulent viscosity ν_T is calculated with the Bradshaw modification:

$$\nu_T = \frac{a_1 k}{\max(\alpha_1 \omega; \beta_1 F_2 \frac{1}{\sqrt{2}} |S|)} \quad (2.36)$$

Where $a_1 = 0.31$. F_2 is a switching function that depends on the wall distance defined as

$$F_2 = \tanh(\Gamma_2^2) \quad (2.37)$$

And

$$\Gamma_2 = \min \left[\max \left(\frac{2\sqrt{k}}{\beta^* \omega y}; \frac{500\nu}{\omega y^2} \right); 100 \right] \quad (2.38)$$

2.5 Moving Reference Frame

As discussed in the introduction, the time-dependent problem of rotating geometries can be simplified into a steady state problem by modifying the reference frame according to the rotational movement. In the OpenFOAM implementation that is used in this thesis and analyzed below, the absolute velocities are solved with the equations expressed in the relative coordinate system. [4]

For any vector \mathbf{a} the time derivative is experienced in a different way by an observer in the inertial coordinate system and an observer in the rotating coordinate system. Their relation is expressed as:

$$\left. \frac{D\mathbf{a}}{Dt} \right|_I = \left. \frac{D\mathbf{a}}{Dt} \right|_R + \boldsymbol{\omega} \times \mathbf{a} \quad (2.38)$$

Where $\boldsymbol{\omega}$ is the rotation vector. Defining as \mathbf{r} the position vector indicating the displacement from the origin and using it in equation (2.38), the relationship between the relative and absolute velocities is derived.

$$\mathbf{U}_I = \left. \frac{D\mathbf{r}}{Dt} \right|_I = \left. \frac{D\mathbf{r}}{Dt} \right|_R + \boldsymbol{\omega} \times \mathbf{r} = \mathbf{U}_R + \boldsymbol{\omega} \times \mathbf{r} \quad (2.39)$$

The equation 2.38 is now used again for the inertial velocity.

$$\begin{aligned} \left. \frac{D\mathbf{U}_I}{Dt} \right|_I &= \left. \frac{D\mathbf{U}_I}{Dt} \right|_R + \boldsymbol{\omega} \times \mathbf{U}_I = \\ &= \left. \frac{D(\mathbf{U}_R + \boldsymbol{\omega} \times \mathbf{r})}{Dt} \right|_R + \boldsymbol{\omega} \times (\mathbf{U}_R + \boldsymbol{\omega} \times \mathbf{r}) = \\ &= \frac{D(\mathbf{U}_R)}{Dt} + \frac{D\boldsymbol{\omega}}{Dt} \times \mathbf{r} + 2\boldsymbol{\omega} \times \mathbf{U}_R + \boldsymbol{\omega} \times \boldsymbol{\omega} \times \mathbf{r} \end{aligned} \quad (2.40)$$

The momentum equation 2.19 is rewritten below. As is, the equation is expressed in the inertial coordinate system using the absolute velocity. Using the equations described above, the momentum equation can be rewritten and expressed in the relative coordinate system using the relative velocity. The pressure is not influenced by the change of the frame of reference since it is a scalar quantity.

$$\frac{D(\mathbf{U}_I)}{Dt} = -\nabla \left(\frac{p}{\rho} \right) + \nabla \cdot (2\nu S) + \mathbf{q}$$

The diffusion term $\nabla \cdot (2\nu S)$ requires some special treatment.

$$\begin{aligned}
 \nabla \cdot (2\nu S) &= \nabla \cdot [\nu(\nabla \mathbf{U}_I + (\nabla \mathbf{U}_I)^T)] = \\
 &\nabla \cdot [\nu(\nabla(\mathbf{U}_R + \omega \times \mathbf{r}) + (\nabla(\mathbf{U}_R + \omega \times \mathbf{r}))^T)] = \\
 &\nabla \cdot [\nu(\nabla \mathbf{U}_R + \nabla(\omega \times \mathbf{r}) + (\nabla \mathbf{U}_R + \nabla(\omega \times \mathbf{r}))^T)] = \\
 &\nabla \cdot [\nu(\nabla \mathbf{U}_R + \nabla(\omega \times \mathbf{r}) + (\nabla \mathbf{U}_R)^T + (\nabla(\omega \times \mathbf{r}))^T)] = \\
 &\nabla \cdot \left[\nu(\nabla \mathbf{U}_R + (\nabla \mathbf{U}_R)^T) + \nu(\underbrace{\nabla(\omega \times \mathbf{r}) + \nabla(\omega \times \mathbf{r})^T}_{=0, \nabla(\omega \times \mathbf{r}) \text{ is antisymmetric}}) \right] = \\
 &\nabla \cdot [\nu(\nabla \mathbf{U}_R + (\nabla \mathbf{U}_R)^T)]
 \end{aligned} \tag{2.41}$$

It can be seen that the diffusion term can be expressed using both the absolute and relative velocity.

If we also consider that the source term in the momentum equation \mathbf{q} independent of the frame of reference, the momentum equation can be expressed in the relative coordinate system.

$$\begin{aligned}
 \frac{D\mathbf{U}_R}{Dt} &= -\nabla \left(\frac{p}{\rho} \right) + \nabla \cdot [\nu(\nabla \mathbf{U}_R + (\nabla \mathbf{U}_R)^T)] + \mathbf{q} \\
 &\quad - \frac{D\omega}{Dt} \times \mathbf{r} - 2\omega \times \mathbf{U}_R - \omega \times \omega \times \mathbf{r}
 \end{aligned} \tag{2.42}$$

Now, $\frac{D\mathbf{U}_R}{Dt}$ can be expanded further to include the absolute velocities.

$$\frac{D\mathbf{U}_R}{Dt} = \frac{\partial \mathbf{U}_R}{\partial t} + \nabla(\mathbf{U}_R \mathbf{U}_R) = \frac{\partial \mathbf{U}_R}{\partial t} + \nabla \cdot (\mathbf{U}_R \mathbf{U}_I - \mathbf{U}_R \omega \times \mathbf{r})$$

$$\frac{\partial \mathbf{U}_R}{\partial t} + \nabla \cdot (\mathbf{U}_R \mathbf{U}_I) - \overbrace{\nabla \cdot \mathbf{U}_R}^{=0} (\omega \times \mathbf{r}) - \mathbf{U}_R \cdot \nabla(\omega \times \mathbf{r})$$

$$\frac{\partial \mathbf{U}_I}{\partial t} + \frac{\partial(\omega \times \mathbf{r})}{\partial t} + \nabla \cdot (\mathbf{U}_R \mathbf{U}_I) - \omega \times \mathbf{U}_R \tag{2.43}$$

We can now substitute equation 2.43 to equation 2.42 and if we assume that ω is constant in time, equation 2.42 takes the following form:

$$\begin{aligned}
\frac{\partial \mathbf{U}_I}{\partial t} + \nabla \cdot (\mathbf{U}_R \mathbf{U}_I) &= -\nabla \left(\frac{p}{\rho} \right) + \nabla \cdot [\nu (\nabla \mathbf{U}_R + (\nabla \mathbf{U}_R)^T)] + \mathbf{q} + \\
&\omega \times \mathbf{U}_R - \omega \times \omega \times \mathbf{r} \\
\\
\frac{\partial \mathbf{U}_I}{\partial t} + \nabla \cdot (\mathbf{U}_R \mathbf{U}_I) &= -\nabla \left(\frac{p}{\rho} \right) + \nabla \cdot [\nu (\nabla \mathbf{U}_I + (\nabla \mathbf{U}_I)^T)] + \mathbf{q} - \\
&\omega \times \mathbf{U}_I
\end{aligned} \tag{2.44}$$

The above equation expresses the momentum equation in the moving reference frame. Only two changes are observed in comparison to the original momentum equation 2.11. The first one is on the convection term, and the second one is an additional source term $\omega \times \mathbf{U}_I$.

2.6 Near Wall Treatment

The presence of the wall influences turbulent flows. The region near the wall that is affected by viscosity is called boundary layer. Turbulent boundary layers have been the subject of both experimental and computational research over the years. Experiments indicate that the boundary layer is made up of three layers, the viscous sublayer, the buffer layer, and the log-law or fully turbulent layer. The division depends on the non-dimensional quantities: dimensionless wall distance y^+ and dimensionless velocity u^+ defined as:

$$u^+ = \frac{U}{u_\tau} \tag{2.45}$$

$$y^+ = \frac{y u_\tau}{\nu} \tag{2.46}$$

Where u_τ is the friction wall velocity defined as:

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \tag{2.47}$$

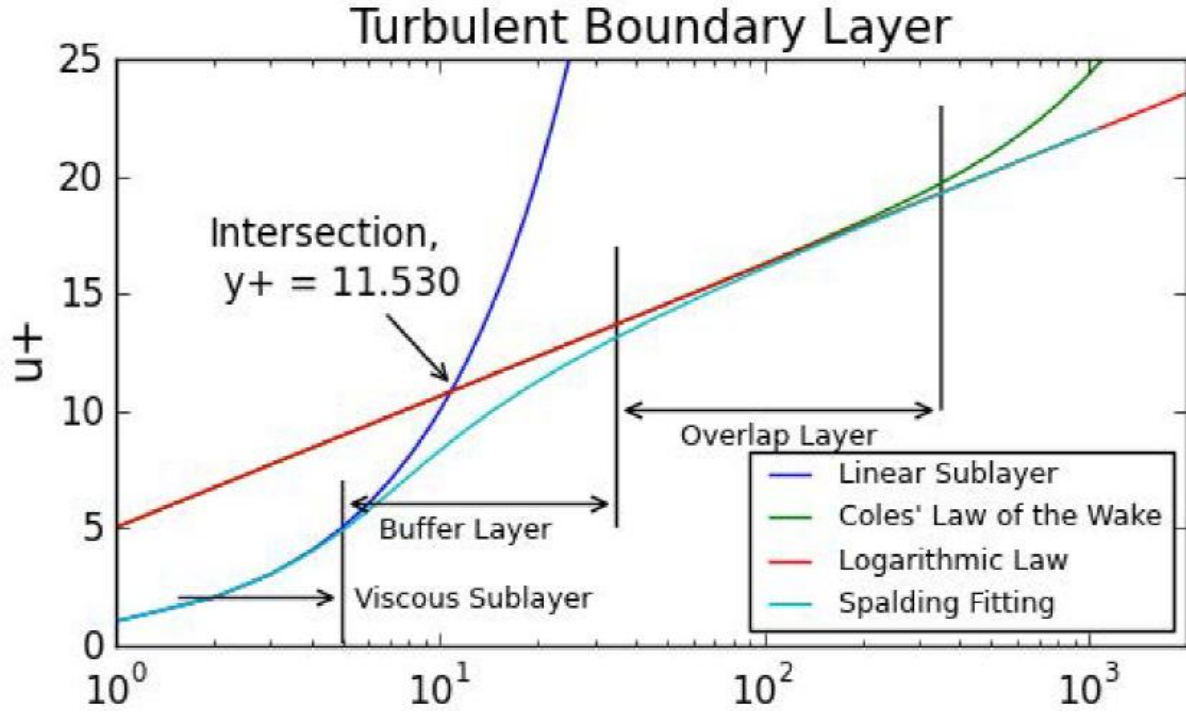


Figure 2.2 Turbulent boundary layer

The viscous sublayer corresponds to values $y^+ \leq 5$. Within this region, the flow is almost laminar and is dominated by viscosity related effects, u^+ is directly proportional to y^+ .

$$u^+ = y^+ \quad (2.48)$$

The log-layer extends from $y^+ = 35$ to $y^+ = 350$. Within this area, the flow is mainly affected by turbulence. The profile of the non-dimensional velocity obeys a logarithmic relationship as seen below:

$$u^+ = \frac{1}{\kappa} \ln(E y^+) \quad (2.49)$$

Where both κ and E are experimental constants with $\kappa = 0.41$ and $E = 9.8$.

The area in-between the aforementioned regions is called buffer layer corresponding to values $5 \leq y^+ \leq 35$. Within this layer, the effects of both viscosity and turbulence are important.

Two main approaches can be distinguished for the treatment of the near-wall region. With the first approach, the viscous sub-layer is not resolved. Semi-empirical formulas called wall functions are used to calculate the region that is affected by viscosity and bridge it to the with the fully-turbulent sub-layer. This approach is widely accepted and used because it is significantly cheaper than the second approach and reasonably accurate. With the second approach, the whole boundary is simulated using small cells close to the wall. This approach yields more accurate results but is in general not preferred due to its simulation cost.

2.7 Discretization

The Navier-Stokes equations described above are non-linear partial differential equations. Due to the non-linear terms, the equations cannot be solved directly, instead numerical methods need to be applied. In order to solve the equations, they need to be transformed into a set of algebraic linear equations that can be solved iteratively. There are several methods to achieve this, the finite volume method, the finite element method and the finite difference method. The most common approach in CFD that is also used in OpenFOAM is the finite volume approach. In the finite volume method, the computational domain is subdivided into a number of discrete finite volumes called cells. Then, the governing equations are integrated over each cell, and using the Gauss divergence theorem the integrals containing the divergence term are converted to surface integrals. [5] The equations are then reformulated on each cell, as a set of linear algebraic equations. Before we introduce the basic principles for the derivation of the algebraic equations, some basic OpenFOAM terminology is introduced.

A typical cell in OpenFOAM is depicted in Figure 2.3. The group of cells that define the computational domain should not overlap and fill the domain completely. The equations are solved in the cell centers (P and N). Every cell is bounded by a set of flat faces (f in the Figure 2.3) and there is no limitation on the number of the faces or their alignment. A mesh of this type is called ‘arbitrarily unstructured’. [6]

Every face is owned by one adjacent cell called neighboring cell. The definition of a face is presented in Figure 2.3(right). Each face is an ordered list of points and the ordering is such that two neighboring points are connected by an edge. The face normal vector \mathbf{S}_f can be seen in Figure 2.3. Its direction is defined by the right-hand rule, according to the point numbering on each face. The volume of a cell is defined as V_P .

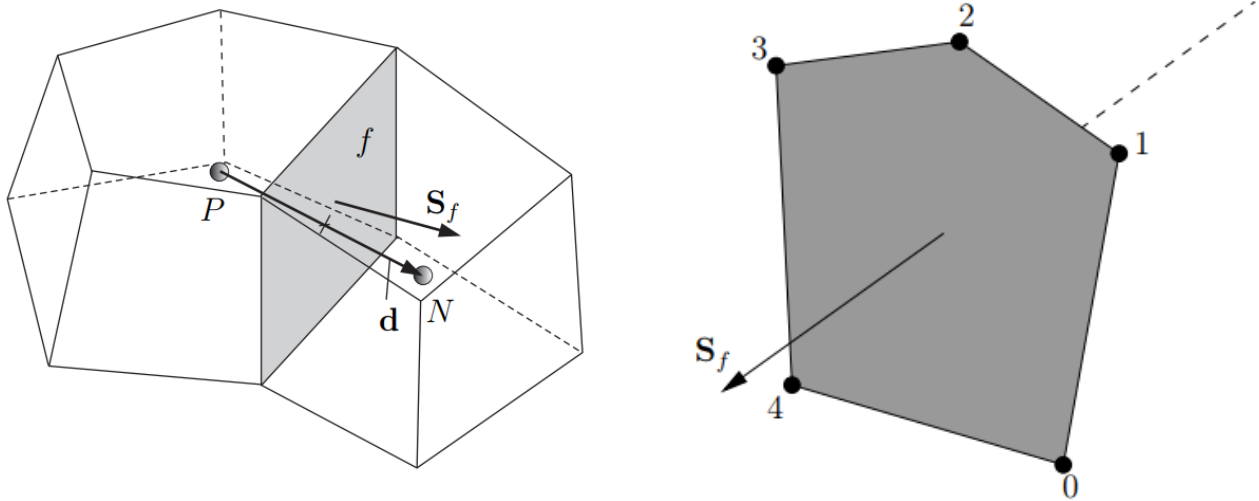


Figure 2.3 OpenFOAM cell definition [6] (left), face definition [32] (right)

The Navier-Stokes equations are in the form of a general transport equation. In order to present how the terms of the equations are discretized, a general form of a transport equation is used. For a fluid property φ :

$$\frac{\partial \varphi}{\partial t} + \nabla \cdot (\mathbf{U} \varphi) = \nabla (a \nabla \varphi) + \mathbf{q}_\varphi \quad (2.50)$$

Integrating the equation over a control volume V we get:

$$\overbrace{\int_V \frac{\partial \varphi}{\partial t} dV}^{\text{time derivative}} + \overbrace{\int_V \nabla \cdot (\mathbf{U} \varphi) dV}^{\text{convection term}} = \overbrace{\int_V \nabla (a \nabla \varphi) dV}^{\text{diffusion term}} + \overbrace{\int_V \mathbf{q}_\varphi dV}^{\text{source term}} \quad (2.51)$$

The integration of both the time derivative and the source term is simple, the terms are multiplied by the cell volume V_P . The time derivative can be approximated using the implicit Euler method, or the more accurate Backward method. [6]

To integrate the convection and diffusion terms, the Gauss divergence theorem is applied. In a general form it can be written as:

$$\int_V \nabla \star \varphi dV = \int_S \varphi \star d\mathbf{S} \quad (2.52)$$

Surface \mathbf{S} is enclosed the volume V_P , φ is any tensor field and the $\nabla \star$ symbol any tensor product. For example, $\nabla \times \text{curl}$ or just ∇ (gradient).

$$\int_V \nabla \cdot (\mathbf{U} \varphi) dV = \int_S \varphi \mathbf{U} \cdot d\mathbf{S} \quad (2.53)$$

We can then approximate the surface integral on the right part of the equation 2.53 as the sum of discrete surfaces \mathbf{S}_f .

$$\int_V \nabla \cdot (\mathbf{U} \varphi) dV = \int_S \varphi \mathbf{U} \cdot d\mathbf{S} = \sum_f \varphi \mathbf{U} \cdot \mathbf{S}_f \quad (2.54)$$

Performing the same steps for the diffusion term, the discrete form of the transport equation is obtained:

$$V_P \frac{\partial \varphi}{\partial t} + \sum_f \varphi \mathbf{U} \cdot \mathbf{S}_f = \sum_f (a \nabla \varphi) \cdot \mathbf{S}_f + V_P \mathbf{q}_\varphi \quad (2.55)$$

The values of the variables $(\nabla\varphi, \alpha, \varphi, \mathbf{U})$ by default are expressed in the cell centers (P, N according to the notation used in Figure 2.3). In order to evaluate the surface integrals in equation 2.55 above, the variables need to be expressed on the surfaces. To achieve this, differencing schemes are used. The differencing scheme is important for the convergence of the method. The efficient differencing scheme should have the following attributes:

- Conservativeness
- Boundedness
- Transportiveness

for a detailed description of them, the reader can refer to [5]. The most common schemes are presented briefly below [6]:

Central differencing (CD) or Linear Interpolation is a second-order accurate but unbounded scheme. The value of a variable on a face φ_f is defined as:

$$\varphi_f = w\varphi_P + (1 - w)\varphi_N \quad (2.56)$$

Where w is a weight factor defined as $w = \frac{\overline{fN}}{\overline{PN}}$, \overline{fN} is the distance between f and the cell center, and \overline{PN} is the distance between the cell centers P and N .

Upwind Differencing (UD) determines φ_f depending on the direction of the flow. This scheme is bounded at the expense of accuracy. The upwind interpolation of φ_f is written as:

$$\varphi_f = \begin{cases} \varphi_P, & \mathbf{U}_f \cdot \mathbf{S}_f \geq 0 \\ \varphi_N, & \mathbf{U}_f \cdot \mathbf{S}_f < 0 \end{cases} \quad (2.57)$$

Blended Differencing (BD) schemes combine the aforementioned UD and CD schemes, aiming to combine the boundedness of the UD scheme with the accuracy of the CD scheme. The value of a variable on a face φ_f is defined as:

$$\varphi_f = (1 - \gamma)(\varphi_f)_{UD} + \gamma(\varphi_f)_{CD} \quad (2.58)$$

Where γ is a blending coefficient.

2.8 SIMPLE Algorithm

Within the Navier-Stokes equations 2.19, there are four unknown quantities, the pressure and three velocity components \mathbf{U} . The extra equation that we can use is the conservation of mass 2.8, however this equation does not include the pressure. Therefore, special treatment is needed for this problem, that is often called the pressure-momentum coupling problem. [7]

In order to derive the pressure equation, a semi-discretized form of the momentum equation is used [8]:

$$a_P \mathbf{U}_P = \mathbf{H}(\mathbf{U}) - \nabla p \quad (2.59)$$

Where

$$\mathbf{H}(\mathbf{U}) = \sum_N \alpha_N \mathbf{U}_N \quad (2.60)$$

Using equation 2.59 we can express \mathbf{U} :

$$\mathbf{U}_P = \frac{\mathbf{H}(\mathbf{U})}{a_P} - \frac{1}{a_P} \nabla p \quad (2.61)$$

The discretized form of the continuity equation is:

$$\nabla \mathbf{U} = \sum_f \mathbf{S} \cdot \mathbf{U}_f = \mathbf{0} \quad (2.61)$$

Velocities on the face of the cell are expressed as:

$$\mathbf{U}_f = \left(\frac{\mathbf{H}(\mathbf{U})}{a_P} \right)_f - \left(\frac{1}{a_P} \right)_f (\nabla p)_f \quad (2.62)$$

Substituting equation 2.62 to equation 2.61, the following form of pressure equation is derived:

$$\nabla \cdot \left(\frac{1}{a_P} \nabla p \right) = \nabla \cdot \left(\frac{\mathbf{H}(\mathbf{U})}{a_P} \right) = \sum_f \mathbf{S} \cdot \left(\frac{\mathbf{H}(\mathbf{U})}{a_P} \right)_f \quad (2.63)$$

The discretized incompressible Navier-Stokes system is:

$$a_P \mathbf{U}_P = \mathbf{H}(\mathbf{U}) - \sum_f \mathbf{S} \cdot \mathbf{p}_f \quad (2.64)$$

$$\sum_f \mathbf{S} \cdot \left[\left(\frac{1}{a_P} \right)_f (\nabla p)_f \right] = \sum_f \mathbf{S} \cdot \left(\frac{\mathbf{H}(\mathbf{U})}{a_P} \right)_f \quad (2.65)$$

The form of these equations shows linear dependence of velocity on pressure. This inter-equation coupling needs special treatment. Two approaches are distinguished. The simultaneous algorithms, that solve the complete systems of equations at the same time over the whole domain, and the segregated approach, where the equations are solved in sequence. The second approach is used in this thesis with the SIMPLE algorithm.

The SIMPLE algorithm which stands for Semi-Implicit Method for Pressure Linked Equations, has been originally proposed by Ratnakar and Spalding [9], it is widely adopted in marine CFD and it is primarily used for steady state problems. In order to achieve stability and fast convergence rate, it is important to estimate the relaxation factors for the fields and equations. The basic steps of the algorithm are presented in the flow chart below.

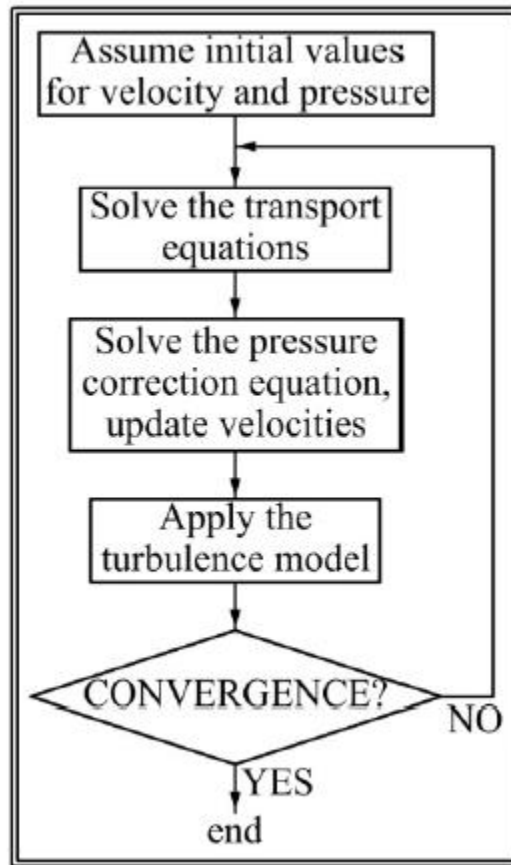


Figure 2.4 The SIMPLE algorithm flow chart [5]

2.9 CAD - Parametric Modelling

Computer aided design has evolved rapidly in the past decades. The beginnings of CAD are traced back to the year 1957, when Dr. Patrick J. Hanratty developed PRONTO, the first commercial numerical-control programming system [10]. A few years later, Ivan Sutherland created SKETCHPAD, the first program ever to use a complete graphical user interface. The invention of the 3D CAD is attributed to the French engineer Pierre Bezier, that developed UNISURF serving as a design tool for the automotive industry. In 1989, T-FLEX and Pro/Engineer introduced software based on parametric engines.

CAD tools today are used extensively in many different industries. They are used in aerospace, automotive, architecture, computer animation and ship design. CAD systems in shipbuilding increased the ease of design, speed of construction and reuse of information. Today, naval architecture design software is used in the fairing of 2D curves like sections and waterlines, in 3D curve and surface design, but also for hydrostatic, resistance and strength calculations. The need for more efficient and optimal designs opened the way to parametric modelling.

Parametric modelling is the definition of a product by means of important descriptors. [11] The objects and features created are modifiable by the use of parameters. This allows for many designs to be generated easily by modifying the parameters that define the model. Constraints can be set to the parameters, as well as relationships between them, resulting in complex geometric models. These models can be used in optimization studies by coupling the parametric geometry to an external software or code that evaluates the hydrodynamic or structural performance of the model.

Three main categories of geometric modelling can be distinguished in computer-aided design [12], conventional, partially parametric and fully parametric modelling.

- Conventional modelling uses a low-level definition of geometry. Shapes are defined by completely independent objects that do not carry any problem specific information. Applying fundamental changes to the model requires for the model to be rebuilt from scratch.
- Partially parametric modelling allows for changes to an existing shape. These changes are defined and controlled by the means of parameters. Some representatives of partial parametric modelling techniques are morphing, free-form deformation and shift transformation. Partially parametric models are typically easy to set up when compared to fully parametric model.
- Fully parametric modelling features shapes that are defined entirely by the means of parameters. A fully parametric model can be looked as a system that takes parameters as input and produces a shape as an output [11]. These parameters define the purpose and essence of the product. Building an efficient parametric model whose parameters will have maximum influence on the performance (e.g. hydrodynamic) requires experience and understanding of the attribute that is evaluated.

According to [13] and the aforementioned classifications, the techniques can be plotted in 2D diagrams and be ranked for their effectiveness, efficiency, cost per variant, flexibility as well as about the required know-how as seen in Figure 2.5 below.

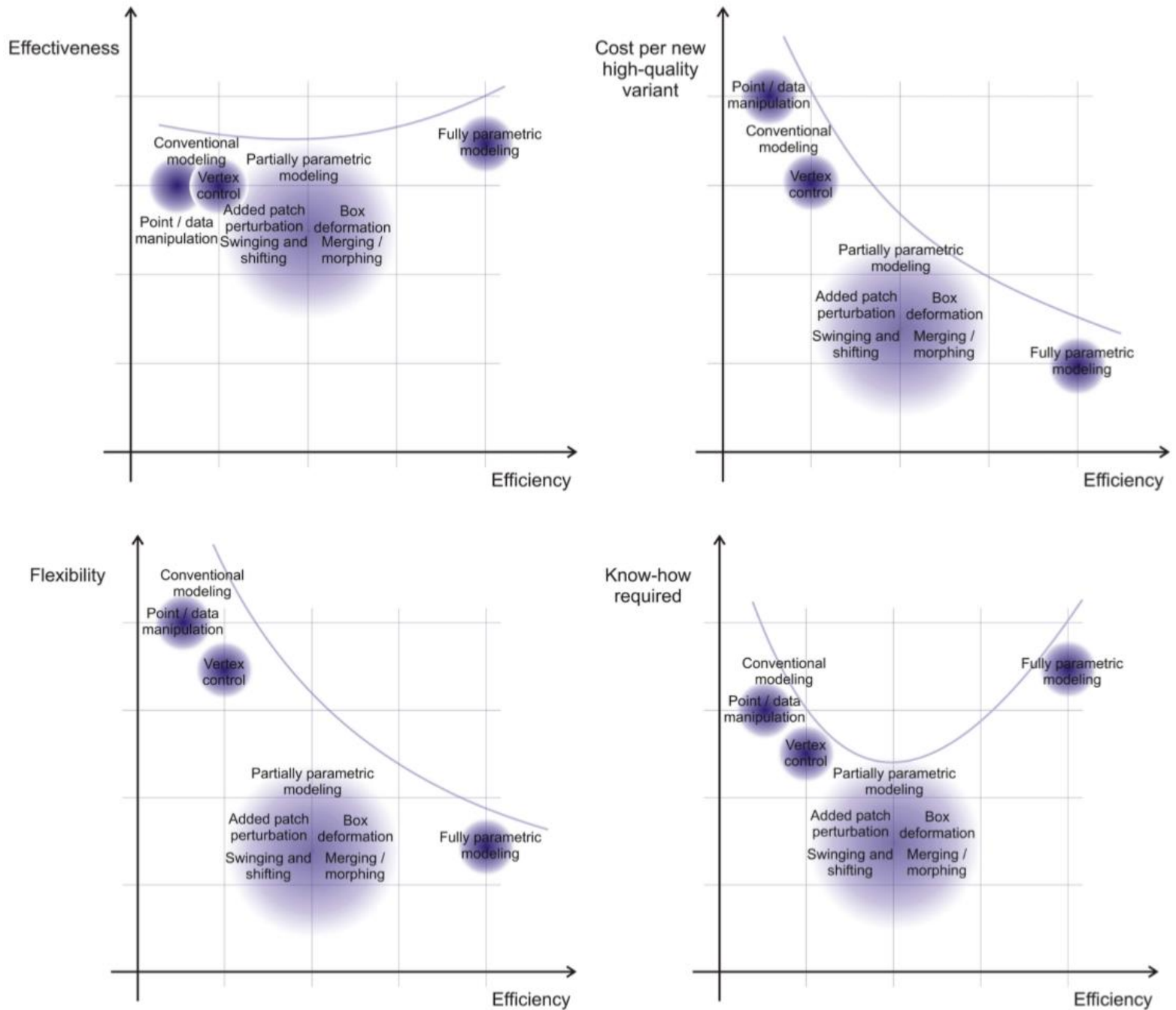


Figure 2.5 Qualitative assessment of geometric modelling techniques [13]

2.10 Grid Generation

In the finite volume method, a finite number of control volumes (CV) are required for the numerical computations. These control volumes are called cells and a group of these cells constitutes a mesh or grid. Even though modern mesh generation techniques, algorithms and software have evolved significantly, grid generation remains a significant challenge for CFD, especially for complex geometries like the propeller blade. Moreover, for the optimization of flow exposed shapes using CFD, automatic and robust grid generation is a prerequisite. This need for complete automation of the mesh generation process is a common bottleneck in optimization processes of complex geometries using CFD.

Three very basic categories of grids can be distinguished, structured, unstructured and hybrid. The basic difference between structured and unstructured lies in the form of the data structure that defines the grid. A structured grid of quadrilaterals consists of a set of coordinates and connectivities that naturally map into elements of a matrix. Thus, neighboring points in a mesh in the physical space are the neighboring elements in the mesh matrix. [14] In this way, the elements are restricted to quadrilaterals in 2D and hexahedra in 3D. With an unstructured mesh, the points cannot be represented in computer memory just as two- or three-dimensional matrices. Additional information needs to be provided. For any point, the connection with other points must be defined explicitly. These meshes usually employ triangles in 2D and tetrahedra in 3D, however hexahedral, pyramids or polyhedral cells can also be present.

Hybrid meshes, as the name suggests, contain both structured and unstructured portions. Typically, parts of the domain that are easy to mesh have structured grids, while unstructured grids are applied in the complex parts. The interface of the two meshes however can be difficult to handle.

Basic characteristics of the aforementioned categories and their comparison is discussed briefly below in the basis of four important metrics.

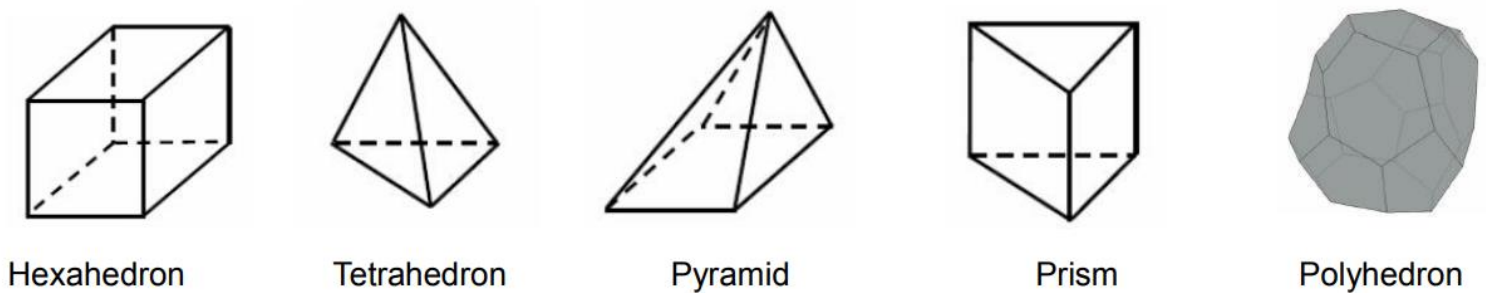


Figure 2.6 Cell types used in CFD codes

- Degree of quality and control

This is an area where structured grids are supreme. Unstructured and hybrid meshing algorithms are highly automated, sacrificing control. Structured grid generation requires careful node placement inside the domain around the geometry. This provides maximum control on the generated mesh as well as high quality. Generating a high-quality structured mesh around a complex geometry can be very time consuming, however the result will most likely worth it. Also, the wall-near cells can be controlled easily making grid refinement studies easier to carry out.

- Flow alignment - Convergence

Structured grids are aligned in the flow direction leading to faster and better convergence in CFD solvers. In unstructured grids, the flow alignment is not guaranteed.

- Computational memory required

As mentioned above, in unstructured grids the connection between the points must be defined. For this reason, they require large computational memory for storing elements and a connectivity table to link them. Structured meshes, on the other hand, do not require any connectivity table. As a result, CFD computation time is typically less in structured meshes.

- Automation

Despite the superior numerical properties of the structured grids, unstructured grids have somehow prevailed in industrial applications, mostly because of their ease of use and automation. Especially for optimization studies, with a structured mesh, special care and thought needs to be spent in advance so that the node placement (topology) will allow the generation of high quality meshes for every variant. On the other hand, unstructured meshes require minimum user effort. Given a geometry and some settings, the grid generator can provide meshes of sufficient quality even for very difficult geometries.

For the purpose of this thesis, the suitable grid generator should be most of all, able to be integrated in an automated closed loop, meaning that it has to be triggered and successfully executed with a few command lines, but also be capable of meshing a complex shape like the propeller blade with sharp edges.

There are several open source solutions compatible with OpenFOAM that meet these requirements. [15]

- blockMesh – Generates block structured grids for simple geometries
- snappyHexMesh – This is an automated hex-core poly mesher with some inflation layer capabilities
 - cfMesh – This is an automated hex-core poly mesher that allows for retaining edges and can subdivide first layer of cells to create pseudo boundary layer mesh
- SALOME – It is capable of generating tetrahedral, hexahedral and prism meshes and exporting them in OpenFOAM file format. Mesh generation routines in SALOME can be automated using python scripts. This software also provides a GUI and can be used as a pre-processor. However, no previous work is found to indicate that it will be suitable for the propeller case.

BlockMesh combined with snappyHexMesh is the most common choice in OpenFOAM applications but literature review indicates that it will most likely not be the best choice for the propeller case, mostly because of the inability of snappyHexMesh to generate prism layer cells of sufficient quality on sharp edges. For this reason, the efforts in this thesis are focused on cfMesh.

These open source tools use some sort of cut-cell approach instead of a bottom-up approach (e.g. extrusion - starting from the geometry wall and building cells in the wall normal direction) that is commonly used in commercial software (Pointwise, ANSA) and they do show some important advantages. They are automatic, stable and robust, they can handle complex arbitrary geometries and they can run in parallel using distributed memory. However, all of them sacrifice control over the surface mesh and boundary layer cells in favor of automation. [16] SnappyHexMesh and cfMesh provide boundary layer tools, but they both suffer from severe limitations including overall boundary layer thickness and lack of control over initial cell height that are important for the control of the y^+ values. Also, the quality of the boundary layer cells that can be generated is not guaranteed. The difficulties faced and workarounds used in this thesis are described in detail in the Method – Grid generation section. On the other hand, the meshing algorithms in commercial packages, structured or unstructured have advanced to provide maximum control on the boundary layer cells, reduce overall cell count, as well as a user-friendly GUIs. Automation is achievable using scripts to replace the manual procedures that take place inside the grid generator. A plane cut of an unstructured mesh built with Pointwise for the Potsdam Propeller Test Case is depicted in Figure 2.7 below. One can see how the high-quality prismatic layers are stretched nicely in the wall normal direction, the rest of the domain is filled with tetrahedra cells.

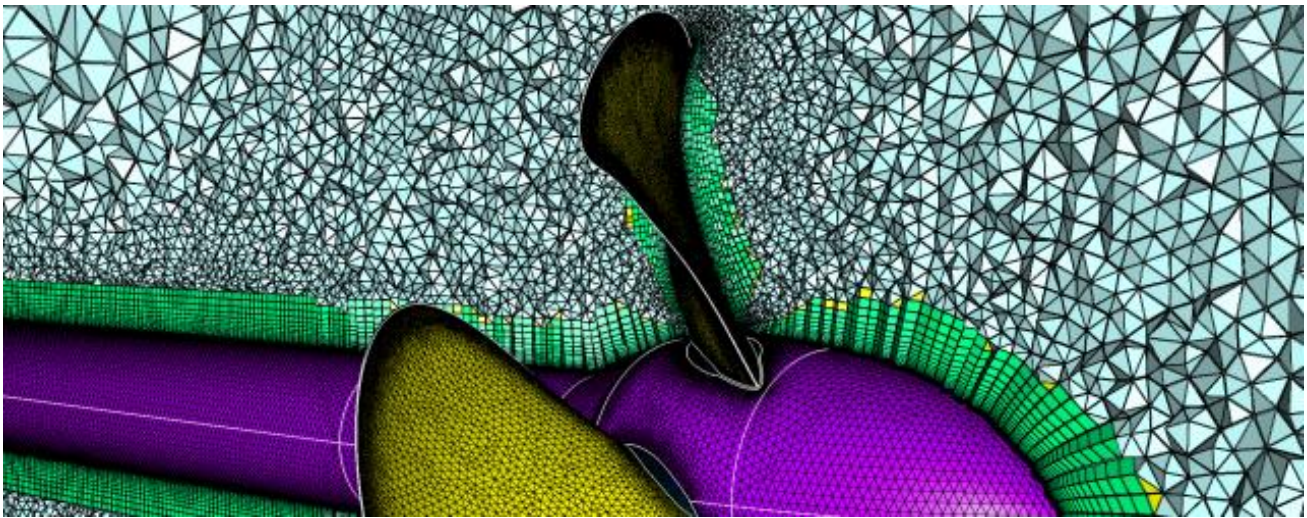


Figure 2.7 Unstructured mesh using Pointwise for the PPTC

2.11 Propeller Optimization

Optimization as a concept is used everywhere, in scheduling everyday life, finance, decision making and, of course, engineering. Optimization can be defined as a continuous manipulation of a system to find better solutions with limited resources. [17] Mathematically, it can be expressed as the problem of finding a vector $\mathbf{x}^* \in \mathbb{R}$ for which,

$$f(\mathbf{x}^*) = \min f(x) \quad (2.66)$$

Subject to

$$g_i(\mathbf{x}^*) \leq 0 \quad i = 1, \dots, I \quad \text{inequality constraints, (2.67)}$$

$$h_j(\mathbf{x}^*) = 0 \quad j = 1, \dots, J \quad \text{equality constraints, (2.68)}$$

$$x_k^l \leq x_k \leq x_k^u \quad k = 1, \dots, K \quad \text{box constraints. (2.69)}$$

In practical shape optimization studies, the goal is to maximize or minimize some attribute of the design, called objective. To achieve this, changes are applied to some aspects of the design. These are called variables, and the objective is a function of these variables. In most applications, many attributes are evaluated and they are related and often conflicting, (i.e. low cost and high performance), defining multi-objective optimization problems. For a multi-objective optimization problem, no single solution exists that optimizes each objective at the same time, on the contrary, a number of Pareto optimal solutions exist. A solution is called, Pareto optimal, or non-dominated if none of the objective functions can be improved in value without degrading some of the other objective values [18].

A wide number of techniques and algorithms are available in order to solve optimization problems [17]. Researchers may use algorithms that terminate in a finite number of steps, iterative procedures that converge to a solution, or heuristics that can lead to approximate solutions [19].

During the past decades, numerous optimization algorithms have been developed. For example, genetic algorithms (GA), successive quadratic programming (SQP), particle swarm optimization (PSO), simulated annealing (SA), infeasibility driven evolutionary algorithms (IDEA) and the quasi-Newton method, among others [20]. These algorithms can be categorized into two major categories, gradient-based and gradient-free methods [21]. For the gradient-based optimization algorithms (e.g., SQP and quasi-Newton approaches), the advantage is the fast convergence; however, the disadvantage is that they may get stuck at locally optimal solutions. For the gradient-free optimization algorithms (e.g., SA, GA, IDEA, and PSO), the convergence is not guaranteed, but they can be used to perform global optimization.

There is no general approach or guidelines regarding optimal optimization. Each engineering problem is unique and the selection of the methods utilized depends on the computational power available and the task at hand.

Marine propeller design and optimization is a very complex process involving numerous factors. The optimum design should not only be more efficient, it also has to produce less noise and vibrations. Cavitation and strength requirements need to be considered as well. For example, thin and narrow blades are expected to show better hydrodynamic properties but there are limitations regarding the blade strength, according to classification requirements [22]. Moreover, during its lifetime the propeller operates in varying inflow conditions, different sea states and revolutions. Thus, propeller optimization is multi-disciplinary and multi-objective procedure where successive compromises need to be made between performance and requirements.

There is a wide variety of numerical simulation tools that can be used to evaluate the performance of a propeller. Depending on the method used, different optimization techniques are utilized. Typically, especially in the early phases of the propeller design, boundary element methods are used for optimization, because many variants can be generated and evaluated automatically and fast. At the end of the design process, more demanding CFD computations are employed that are able to predict the viscous flow effects. [22] In the figure below, the methods available are ranked according to the time they require and the physical phenomena they neglect. One can notice that fully unsteady RANS or large eddy simulations cannot be employed for an optimization task. With these methods, depending on the computational power and the mesh density, evaluating one variant may take even weeks.

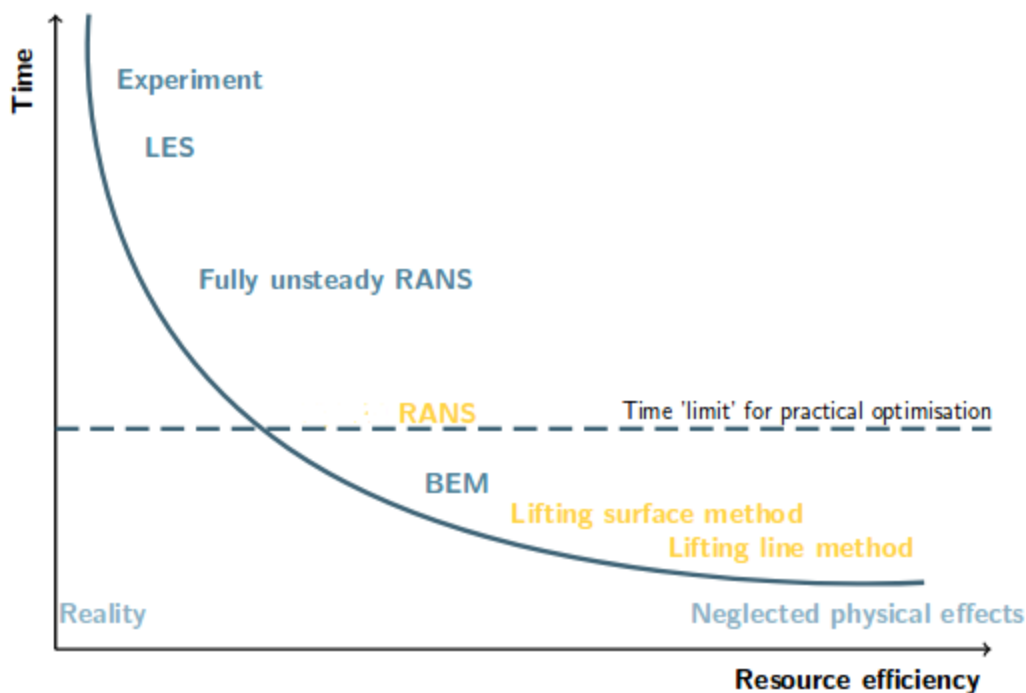


Figure 2.8 Fidelity of numerical methods for propeller simulation [17]

Optimization processes are often seen as a black box, that given some input design variables the algorithms used will automatically provide the best outcomes. Global optimization techniques like genetic algorithms are not far from that. Genetic algorithms can handle the whole optimization process, from variant creation to assessment, crossover and mutation and finally giving pareto fronts. [23] Genetic algorithms have been used widely to deal with multi-objective propeller optimization problems. More specifically, the NSGA-II algorithm has been used extensively to perform global optimization in naval architecture and specifically in multi-objective propeller optimization considering even cavitation constraints [24] . According to [25], the success of the NSGA-II algorithm depends on the population number, probability of crossover and probability of mutation. Literature review indicates that an initial population of 100 variants are needed for the algorithm to converge and reach satisfactory results, that means 100 propeller performance evaluations only for the initial population. Keeping in mind that a number of generations will be generated and evaluated as well, the total number of evaluations required is large. In the work referenced, the authors relied on B-series polynomials to evaluate the propeller performance, thus computational time has not been an important factor. In optimization studies using CFD, simulation time and computational resources can be very restrictive.

In the present work, steady state RANS computations are used to evaluate the propeller performance and the computational resources available are not sufficient to perform global optimization using for example the NSGA-II algorithm.

For this reason, the search for the optimal design is an interactive process. The success of the process depends heavily on the robustness of the grid generator, the quality of the parametric model and the impact of the design variables on the blade shape.

According to [11], two major approaches can be distinguished for optimization in simulation driven design. These are parameter-based optimization and parameter-free optimization. The first approach relies on parametric models whereas the second one uses topology optimization and adjoint simulation. The parameter-based optimization approach is used in this thesis and for this reason the rest of the section refers to it.

First of all, the standard format of an optimization problem can be summarized using the following five elements. [11]

- Objective(s): They define the attribute(s) of the design that is to be improved
- Free variables: These are the aspects of the designs that can be modified
- Constraints: These are restrictions applied to the free variables or attributes of the design
- Fixed parameters: They are aspects of the design that influence its behavior but are kept constant on purpose
- Noise: Other factors that influence the system but are beyond our control

In order to perform shape parameter-based optimization, the following components are required.

- Variable geometry: A parametric model that is controlled by the means of a number of variables
- Pre-processing: Generation of watertight models that can be meshed
- Simulation:
 1. Discretizing the fluid domain into a finite number of cells
 2. Solving the governing flow equations
- Post-processing: Evaluating the results by visualizing the flow data
- Optimization & Assessment: Generation and assessment of a number of variants.

Harries [11] suggests a two-phase approach for shape optimization. These two phases are exploration and exploitation. This approach is also followed in this thesis, and its main attributes is discussed briefly below.

Once the parametric model is constructed, a selection is made on which parameters will be activated as design variables. The number of these design variables defines the size of the design space. With the exploration phase, the designer actually searches through the design space trying to understand the influence of the design variables on the attributes that are evaluated and detect areas of interest.

During the exploitation phase, the designer searches locally in the promising regions identified at the exploration phase. This can be done by removing the parameters of less importance, or by modifying the bounds of the parameters.

An efficient way to explore the design space is to perform Design-of-Experiments (DoE). Design of experiments can be defined as a series of statistical methods used to understand and evaluate a systems behavior and more specifically, how the input factors influence the output variables. A successful design of experiment will provide maximum understanding of the system's behavior with the minimum cost. A popular algorithm used to perform DoEs that is also used in the present work is the Sobol sequence introduced by the Russian mathematician Sobol I.M in 1967.

The Sobol sequence is a quasi-random or low discrepancy sequence. Quasi-random generators are deterministic in contrast to pseudo random generators. The Sobol sequence can fill up a space with points that are distributed more uniformly than a pseudorandom number generator. [26] An example of how the Sobol sequence fills up a two-dimensional space is depicted in Figures 2.9 below. The Sobol sequence proved to be very helpful in the present work. It generates combinations of the design variables that fill up the multi-dimensional design space evenly, providing insight about their influence and selection.

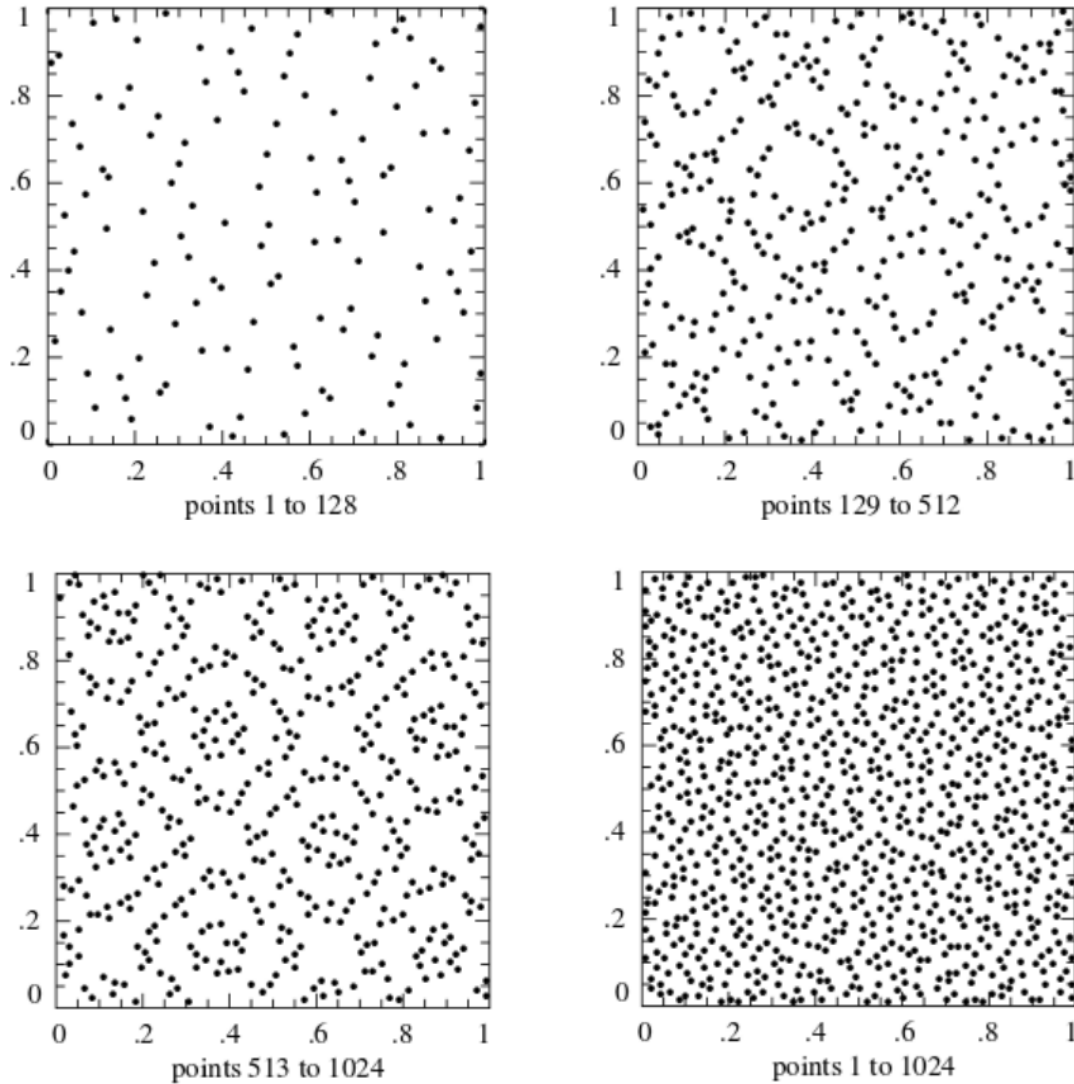


Figure 2.9 The first 1024 points of a two-dimensional Sobol sequence

Once the exploration phase is completed and promising design candidates are found, they can be fine-tuned to produce the best possible outcome. This phase is called exploitation or formal-optimization.

Typically, the designer selects a group of the best designs discovered during the exploration phase and uses an optimization algorithm for every selected design in order to advance towards local optima. Ideally, the exploitation phase will bring out a global optimum; however computational resources usually limit the search iterations and the number of the designs selected from the exploration. In most cases, the designer doesn't know if the optimized design is actually a global optimum, but this not the goal after all, since any improvement on the objectives is welcome.

In the present work, the exploitation is performed using the Tangent Search Method, originally proposed by proposed by Hilleary [27]. The T-Search is a gradient-free method that can be very efficient for local optimization problems (Figure 2.10) with a single objective goal; inequality constraints can be set as well. It detects a descent search direction in the design space towards the objective. According to [28], T-Search is characterized by efficient operation within the admissible region with satisfactory performance along the boundary of that region. The algorithm starts by exploratory moves (modifying one variable at a time) to detect promising directions and global moves (modifying many variables at a time) making steps along the identified directions towards the improvement of the objective. [29]

During this project, no considerations regarding cavitation or strength requirements are made. Introducing these requirements in the overall process and implementing them into a closed automated loop to include them in the optimization would be very demanding exceeding the limits of a thesis project. For example, to evaluate the propeller blade strength, finite element analysis would have to be carried out using advanced commercial CAE software (Abaqus, ANSYS). Thus, the optimization problem is set as single objective; to improve the open water efficiency. During the exploitation phase, the objective set to the T-Search algorithm is to minimize K_Q .



Figure 2.10 Optimization algorithm searching for optimum designs [30]

3 Software

3.1 Geometry - Friendship Framework – CAESES

CAESES® is a parametric CAD platform for fast and comprehensive design studies using external simulation tools, developed by Friendship Systems AG. It is an evolved version of the Friendship-Modeller, presented by Harries in 1998 and a result of extensive research at Technical University of Berlin. CAESES is capable of creating flexible and smooth 3D geometries using design parameters, setting constraints, importing and exporting geometries of various types, coupling geometry to external CFD packages and performing optimization.

CAESES helps and aims to move CFD analysis and optimization studies at earlier phases of the product development of flow-exposed shapes (hulls, propellers, turbines). These studies are typically carried out in the final stages of the development of a product where the potential gains of an optimization study are restricted [11]. Moreover, 3D models at this phase usually contain information and detail (e.g. when they are intended for production) that are irrelevant to the evaluation of the hydrodynamic performance. Thus, time and effort needs to be spent on defeating the 3D models to open the way for the CFD analysis (cleaning, meshing, solving, post-processing). With CAESES, the resulting models are clean, robust, CFD-ready and able to be contained in closed automated optimization loops.

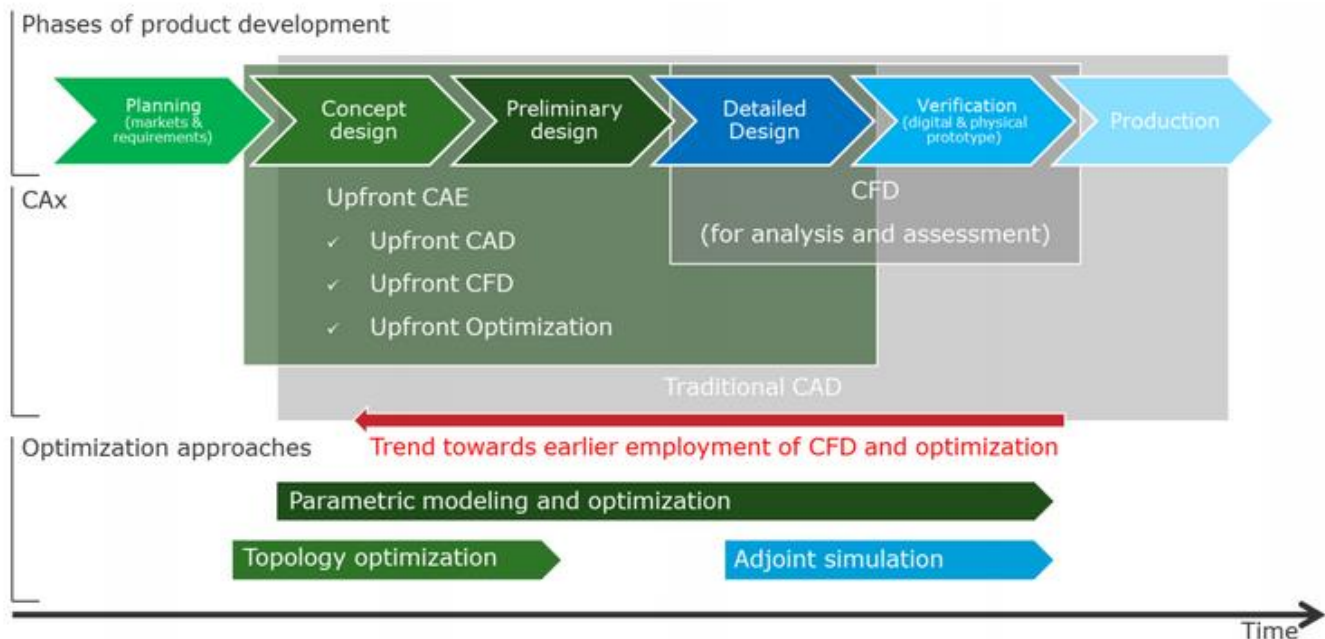


Figure 3.1 Phases of product development [11]

The software offers a powerful object-oriented scripting language that is easy to learn and manipulate, with auto-completion capabilities, syntax highlighting and even a debugger. Anything that a user can do with the graphical user interface (e.g. imports-exports, geometry generation, geometry manipulation) can be coded and automated inside scripts called Feature Definitions. Feature Definitions are also used to create the most smooth and flexible surfaces available in the software, called Meta Surfaces. In order to construct a Meta Surface, the user has to define a cross-sectional curve in a Feature Definition with parameters defining its geometrical aspects. This Feature Definition serves as an input to a Curve Engine along with 2D curves that control the distribution of the predefined parameters. Now, the Curve Engine carries all the information that defines the surface. Finally, the surface is materialized with the Meta Surface object. The process is illustrated in Figure 3.2 below.

These flexible surface objects along with the F-splines (fairness-optimized splines), allow the designer to create complex parametric surfaces with maximum control. Moreover, imported geometry can be modified using partially parametric techniques like free-form deformations, morphing and shift transformation.

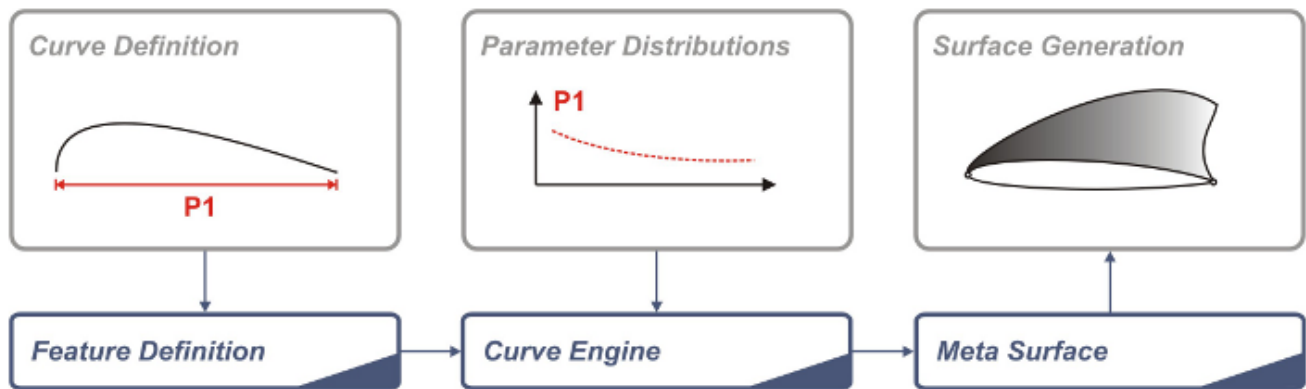


Figure 3.2 Meta Surface creation process in CAESES [29]

Finally, CAESES provides a number of optimization tools and algorithms suitable for various tasks and optimization problems (e.g. Sobol, Brent, TSearch, NSGA-II, Dakota), as well as post-processing tools to visualize the flow data and utilities to visualize the optimization results, for example the influence of the parameters, pareto fronts and so on.

3.2 Solver - OpenFOAM

3.2.1 Introduction to OpenFOAM

OpenFOAM, which stands for Open Source Field Operation and Manipulation, is the leading free open source CFD software, owned by the OpenFOAM Foundation and distributed exclusively under the General Public License (GPL) [31]. OpenFOAM was created by Henry Weller in the late 1980s aiming to provide a more flexible and powerful simulation platform than FORTRAN. The main idea was to develop a code based on the *Finite Volume Method* to solve systems of Partial Differential Equations using C++ and object-oriented programming.

It was developed on the basis of the C++ programming language, due to its object-oriented features and flexibility. Object oriented languages, like C++, offer the mechanism of classes that can be used to declare types and associated operations for each mathematical object. OpenFOAM makes use of this functionality, by defining the scalars, vectors and tensors that appear in the equations which describe the fluid flow, as objects. For example, the velocity field \mathbf{U} is an object that belongs to the class *vectorField*, which derives from the classes *vector* and *field*, and inherits their properties. This object-oriented approach does not only make the codes clearer and more compact, but also much easier to develop and maintain, since the code development is contained in specific regions of the code, i.e. the classes themselves, and inheritance helps reduce duplication of the code.

Moreover, the models are implemented in such way, so that the Partial Differential Equations (PDEs) are expressed in their natural language (equation mimicking). Consequently, the syntax of the code that involves tensor operations, closely resembles the equations that are being solved. For example, the following differential equation

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot \phi \mathbf{U} - \nabla \cdot \mu \nabla \mathbf{U} = -\nabla p$$

can be written in an OpenFOAM file, as seen below

```
solve
(
    fvm::ddt(rho, U)
    + fvm::div(phi, U)
    - fvm::laplacian(mu, U)
    ==
    - fvc::grad(p)
);
```

In this way, the source code is comprehensible and given the open source nature of OpenFOAM, it is easier for users and developers to modify existing solvers or create new ones. Deep knowledge of C++ programming is not necessarily needed to write a new solver, however basic understanding of C++ code syntax and the principles of object-orientation and classes are essential.

To this day, OpenFOAM has received high customization and extension of its functionalities, and is widely used both in the academia and the industry. New versions are released to the public every year, containing a great collection of precompiled applications and features able to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to solid mechanics and electromagnetics. Apart from being totally free of charge and customizable, OpenFOAM has many more benefits to offer like parallel computing and powerful pre- and post-processing utilities. However, the lack of a Graphical User Interface (GUI) makes the learning curve for new users quite steep.

OpenFOAM's pre-built applications fall into two categories: *solvers*, that are designed to solve a specific problem; and *utilities* that are designed to perform tasks that involve pre- and post-processing, meshing and data manipulation. [32]

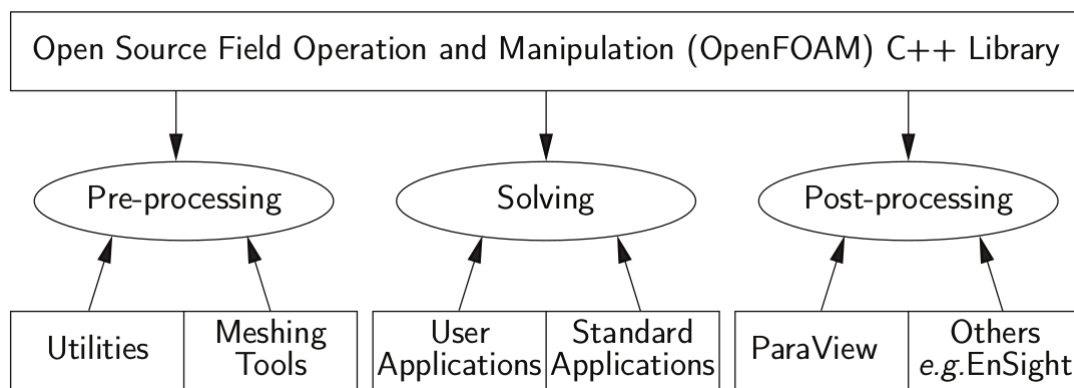


Figure 3.3 Overview of OpenFOAM structure [32]

There is a great variety of different solvers, so they are divided in the following categories:

- Basic CFD codes
- Incompressible flow
- Compressible flow
- Multiphase flow
- Direct numerical simulation (DNS)
- Combustion
- Heat transfer and buoyancy-driven flows
- Particle-tracking flows
- Discrete methods
- Electromagnetics
- Stress analysis of solids
- Finance

The utilities can be divided in the following categories:

- Pre-processing
- Mesh generation
- Mesh conversion
- Mesh manipulation
- Post-processing
- Post-processing data converters .
- Surface mesh tools
- Parallel processing

3.2.2 Structure of an OpenFOAM Case

Executing an OpenFOAM application requires proper setting of the CASE directory that will contain files (dictionaries) where all the parameters involved in the simulation are specified. The basic CASE directory structure can be seen in Figure 3.4. The main directories are **constant** and **system** and a number of time directories that are created during the simulation containing the simulation data. The functionalities of these directories are described below.

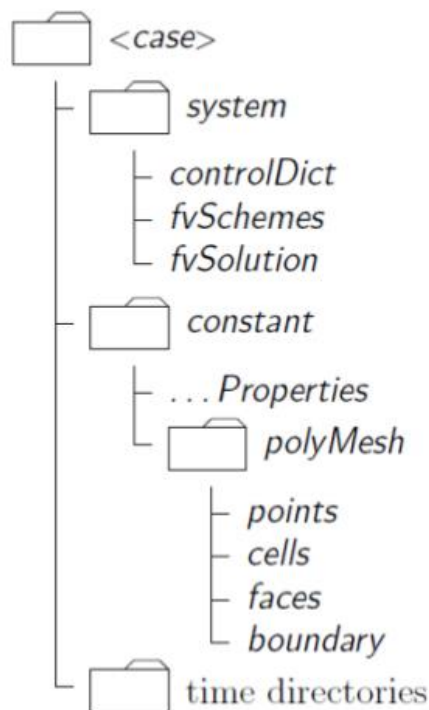


Figure 3.4 Case directory structure [32]

- The **constant** directory:

This directory contains dictionaries where the physical properties for the application are specified, such as the turbulence model or the value of the viscosity. A subfolder called **polyMesh** is also included, containing a set of files that describe the mesh used in the simulation

- The **system** directory:

In this directory, all the parameters regarding the solution procedure are selected. The three most important files are: **controlDict**, **fvSchemes**, **fvSolution**. In the **controlDict**, run control parameters are specified, including the start and end time of the simulation, the time step as well as the time interval between saving data. The **fvSchemes** dictionary includes the discretization schemes used in the solution and the **fvSolution** contains the equation solvers, tolerances and other algorithm controls (*i.e.* Relaxation factors).

- The **time** directories:

During the simulation, time directories are written depending on the settings specified in the **controlDict**. These time directories contain files of data for particular fields, *i.e.* pressure. These data are initial values and boundary conditions that are required to start the simulation, usually specified in time directory '0', or result data that are saved in the selected time step (*i.e.* 900).

When all the necessary directories are set, the user can start the simulation by executing commands in the terminal. These commands trigger utilities or solvers that are responsible for the generation of the computational grid, running the computations and post-process the data. If the user desires to run the computations on multiple processors, dedicated utilities are available that will decompose the domain and run the solver in parallel. In a post-processing stage, the user can recompose the domain and visualize the flow data. The commands used can be written in a single executable, and the whole process can be easily automated. At the post-processing stage, the data contained in the time directories can be visualized in the open source software **Paraview**, that comes built with OpenFOAM or can be downloaded separately. Paraview can be used to visualize and inspect the mesh, create colored plots with the flow data, visualize streamlines and so on.

3.2.3 Compiling OpenFOAM Applications and Libraries

As stated above, users are free to customize the codes available in OpenFOAM or develop their own. Compilation is an integral part of this process and for this reason, some basic information about compiling OpenFOAM applications will be described below.

Since OpenFOAM is a C++ library, the compilation process is eventually the same as the one used to compile C++ applications. However, OpenFOAM uses its own **wmake** compilation script that is based on the standard **UNIXmake** utility, commonly used in UNIX/Linux systems. Wmake though, is more versatile, easier to use and can be actually used to any code, not only the OpenFOAM library. Before describing how wmake is used, some basic aspects of C++ need to be introduced. In C++, a class is defined in a file that contains its attributes such as object construction and data storage and takes a .C extension. This file can be compiled independently of other code into an executable library file called shared object library with the .so extension. That being said, when a new piece of code is to be compiled *i.e.* newObject.C that uses another class *i.e.* oldObject,

oldObject.C doesn't need to be recompiled, and newObject.C will call the oldObject.so library that was created when oldClass.C was compiled in the first place. This is known as dynamic linking.

Another important aspect is that every piece of code contained in a .C file has an associated .H file. This piece of code, will most likely require properties and functions from other classes, so every used class needs to be declared through a **class declaration** that is contained in a file with a .H extension. In order to link the .C and .H files, the command `#include 'otherObject.H'` is stated at the beginning of the new code contained in a .C file. The .H files are called **dependencies**. The procedure described above is illustrated in Figure 3.5.

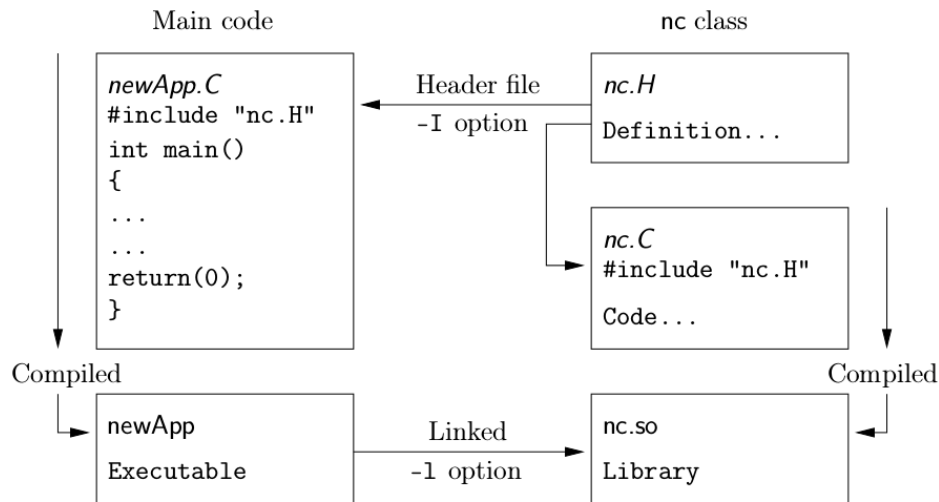


Figure 3.5 C++ Class mechanism [32]

OpenFOAM applications are typically organized using the convention that the source code of each application is placed in a directory with the same name as the application. They are made up of two files a .C and a .H. By navigating to the folder containing these two files and executing the command **wmake** the application is compiled. To perform the compilation successfully, **wmake** requires two extra files located in a subfolder named **Make**, these are called **options** and **files**. **Options** contains the paths to the .H and .so files needed for the compilation, while **files** contains a list of the .c source files that have to be compiled, as well as the path and name of the executable to be created.

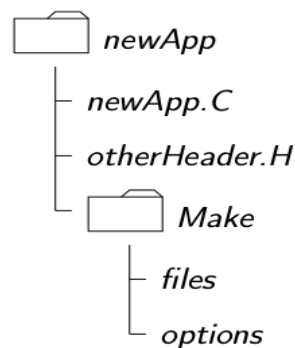


Figure 3.6 Directory structure for an application

3.3 Grid Generation - CfMesh

CfMesh is a library for unstructured automatic volume mesh generation that is built on top of OpenFOAM. It is developed by Creative Fields Limited, headed by Dr. Franjo Juretic, principal developer of the tool and managing director and founder of Creative Fields. The purpose behind it was to develop a grid generator that is robust and simple to use, but also easy to learn and extend. CfMesh is an open-source tool, licensed under the GPL license and compatible with all the recent versions of OpenFOAM.

The library supports various 2D and 3D workflows to generate meshes of arbitrary cell types. The current workflows generate cartesian, tetrahedra and polyhedral meshes. All workflows are parallelized for shared memory machines and use all available computer cores while running, accelerating the meshing process. [33]

CfMesh requires the input geometry in a form of surface triangulation and a set of user defined settings that control the cell sizes and refinements. Some stages of the meshing process are depicted in Figure 3.7 below with the trailing edge of the propeller blade as an example. The process starts by creating a template based on the input geometry, then the template is adjusted to match the input geometry. Finally, boundary layers are generated according to the user's settings. The process is designed to be able to handle poor quality input geometries that may contain gaps.

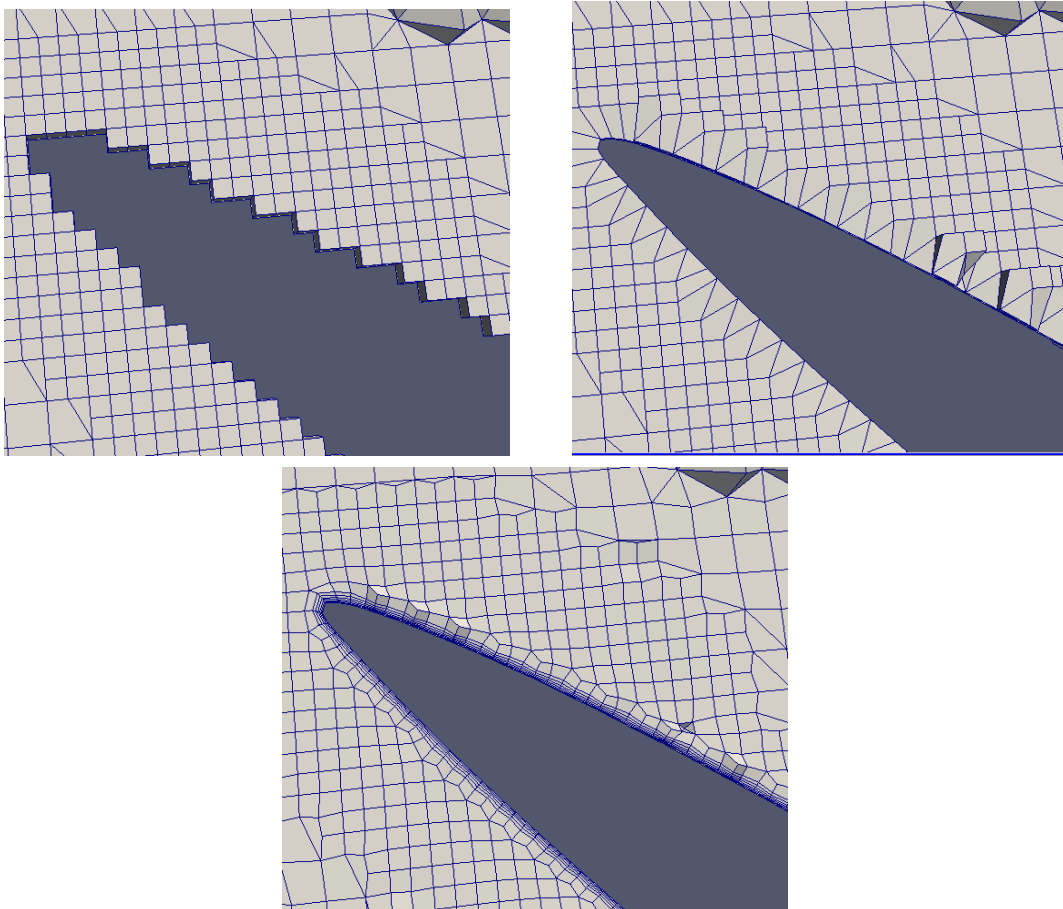


Figure 3.7 Meshing process stages

4 Method

4.1 The Test Case - PPTC

The Potsdam Propeller Test Case is selected as the basis of the computations and the optimization study. Detailed description of the geometry and experimental data including open water characteristics, velocity field measurements as well as cavitation patterns are provided by the SVA Potsdam (Schiffbau-Versuchsanstalt) and are available online [34]. The Potsdam Propeller Test Case was part of the SMP'11 Workshop on Cavitation and Performance and the data is intended to offer research groups the possibility to test and validate their calculation methods. The basic characteristics of the propeller can be seen in the Table 4.1 below.

The geometry is available in many file formats. Apart from the traditional propeller drawings, including the radial distributions and the projected and developed outlines, the geometry is available as a 3D IGES/STEP/3dm file including the hub, fillet and shaft but also as a PFF file (Propeller Free Format), a common format used by design companies and classification societies that proved to be particularly helpful for the creation of the parametric model later on.

Table 4.1 Potsdam Propeller Test Case Geometry

Propeller Diameter	D	[mm]	250.0000
Pitch at $r/R=0.7$	$P_{0.7}$	[mm]	408.7500
Pitch at $r/R=0.75$	$P_{0.75}$	[mm]	407.3804
Mean pitch	P_{mean}	[mm]	391.8812
Chord length at $r/R=0.7$	$C_{0.70}$	[mm]	104.1670
Chord length at $r/R=0.75$	$C_{0.75}$	[mm]	106.3476
Thickness at $r/R=0.75$	$t_{0.75}$	[mm]	3.7916
Pitch ratio	$P_{0.7}/D$	[-]	1.6350
Mean pitch ratio	P_{mean}/D	[-]	1.5675
Area ratio	A_E/A_0	[-]	0.7790
Skew	θ	[°]	18.8000
Hub diameter ratio	d_h/D	[-]	0.1500
Number of blades	Z	[-]	5
Direction of rotation	right-handed		

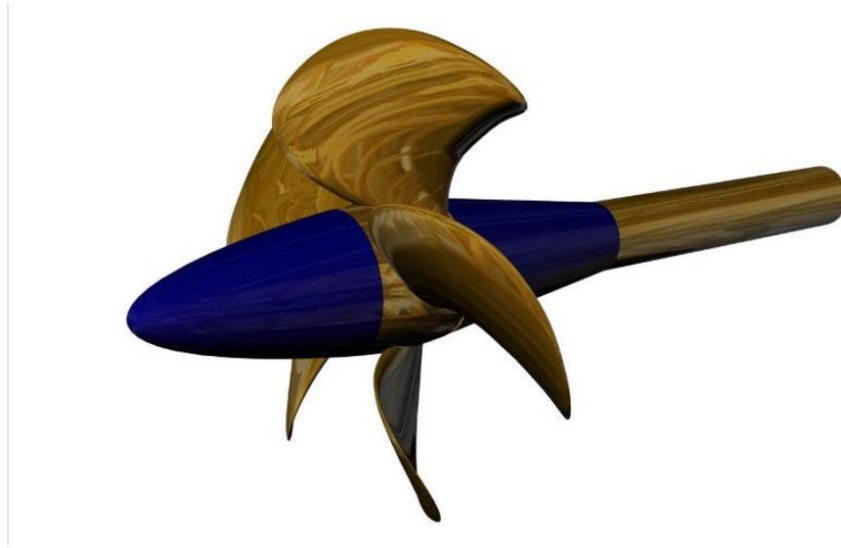


Figure 4.1 CAD Geometry of the PPTC propeller

4.2 Pre Processing - Geometry Cleanup

As a first step for the validation of the OpenFOAM-cfMesh configuration, the original geometry provided online has to be prepared for the mesh generation process. The PPTC is a controllable pitch propeller, and this results in a 0.3mm gap between the hub and the propeller blade near the leading and trailing edge. At the pre-processing stage, this small gap has to be removed since a watertight STL¹ is required by the grid generator. The tip area needs special treatment as well.

As imported into a CAD system, in this case that is CAESES, the geometry comes as a set of different Brep² Parts. These individual parts have to be combined into a single Brep, that will finally be exported as an STL file and will be given to the grid generator. CAESES provides the functionality of Boolean Operations between Breps and this functionality is extensively used throughout this thesis. Boolean Operations between Breps in CAESES are quite powerful and robust, leading to significant reduction of the time spent on pre-processing and clean ready for meshing models. A specific Boolean operation is available that creates a smooth fillet between the hub and the propeller blade, this operation is used to replace original fillet and remove the gap. In the picture below, on the left one can see the small gap between the fillet and the blade as well as the red curves in their connection, indicating that they are different Brep Parts. On the right part, the new fillet is visible, and the Brep Parts are connected into a single Brep.

¹ STL: Abbreviation of “stereolithography”. A common file format in 3D CAD systems

² Brep: Abbreviation of “Boundary representation”. Method of representing geometry in CAD systems using the limits

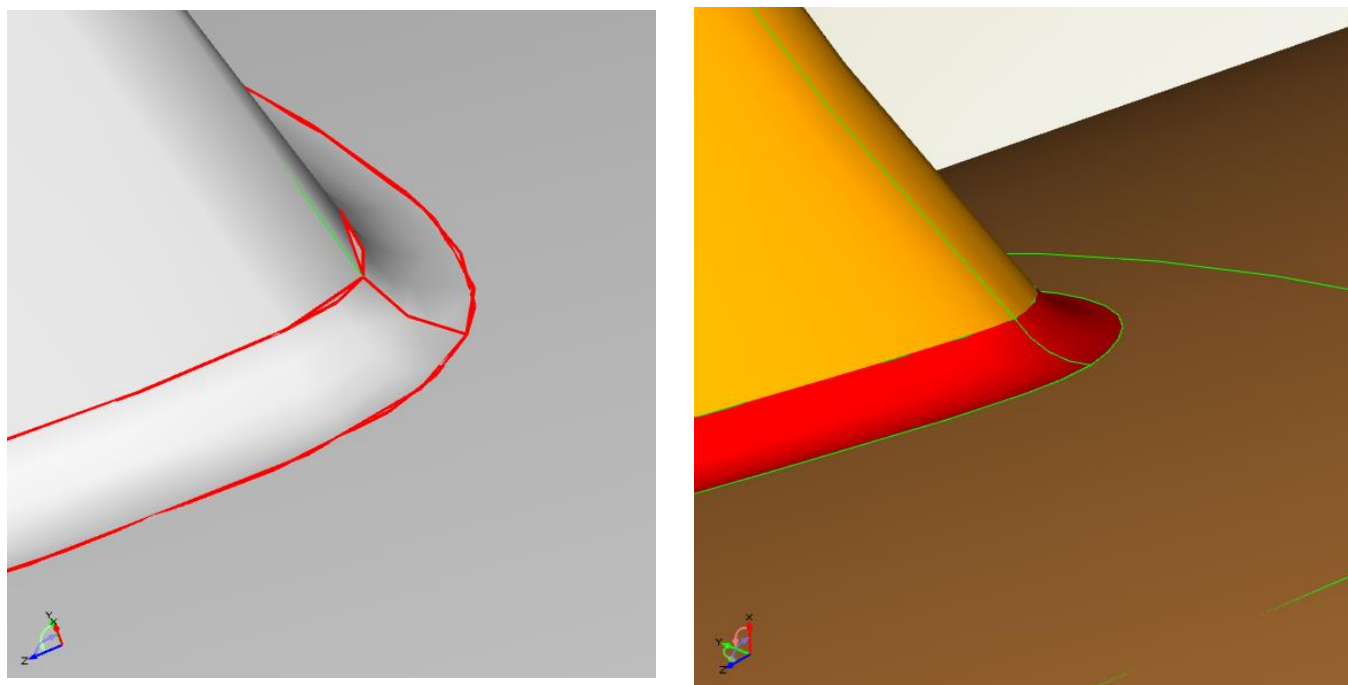


Figure 4.2 Detail of the trailing edge near the fillet

Another deficiency of the original CAD geometry is a gap at the tip region. A new surface is created to fill the gap and all the different Brep parts are combined in a watertight solid as seen in Figure 4.3.

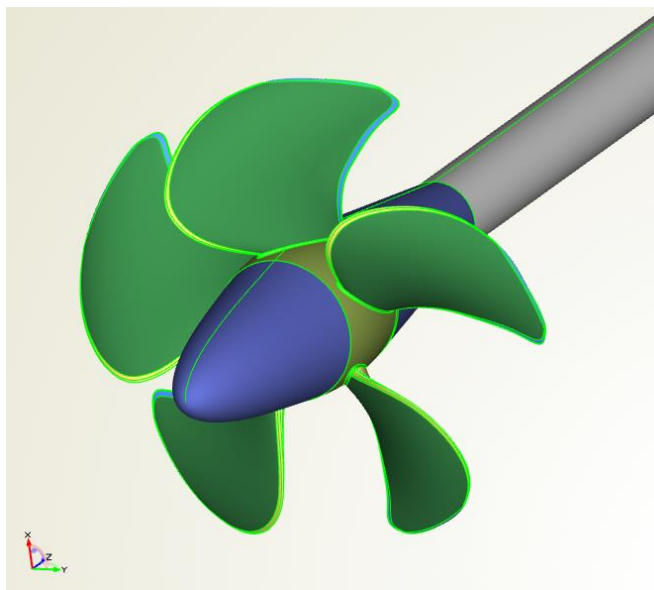


Figure 4.3 Complete watertight geometry including the shaft

4.3 Domain Construction

A computational domain that will be discretized into a finite number of cells is needed for the CFD computations. In the case of the propeller, that is usually a cylinder. However, when simulating the open water test in straight flow, one can use the one blade passage approach, take advantage of the axial flow symmetry and solve the flow for one of the blades using periodic boundaries. In this way, the volume of the computational domain is reduced significantly and consequently the number of cells needed for the computations is reduced as well, leading to shorter simulation time. The time needed for the computations is a very important factor in an optimization study, especially when the computational power is limited, since it will determine the size of the design space and the number of variants to be explored.

The periodic domain is constructed so that the periodic sides are kept as far from the blade as possible. For the construction of the domain, the Meta-Surface technology is utilized, that was briefly presented in the Software Section. Firstly, two points are extracted from the leading and the trailing edge, at the radial position corresponding to the maximum chord length of the blade. The two points are connected with a line that is extended in both directions. The edges of this line and two more points controlling the start and the end of the domain, will be used as the weighted control points of a NURBS curve. The weights of the control points can be modified in order to fit the propeller sufficiently. This NURBS curve is the core of the MetaSurface that will be created in the next step.

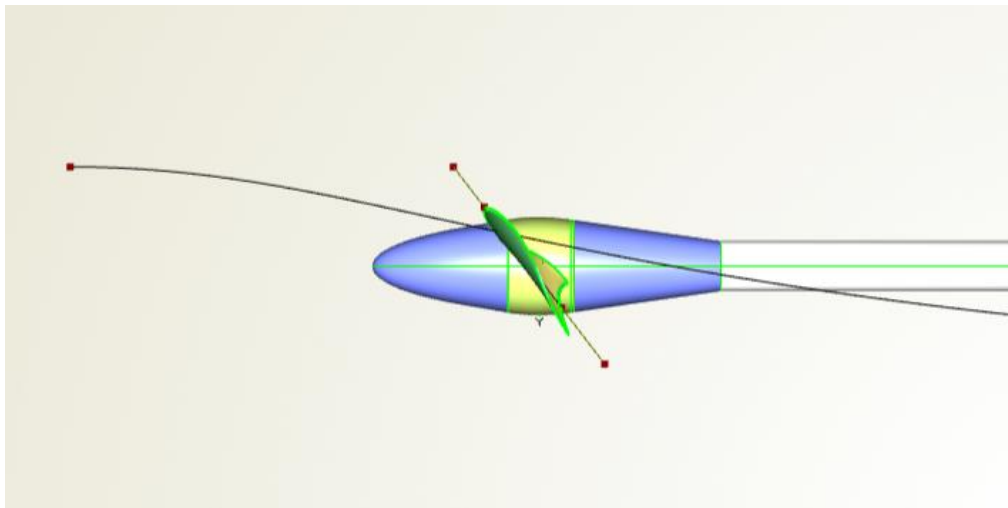


Figure 4.4 First step of the domain construction

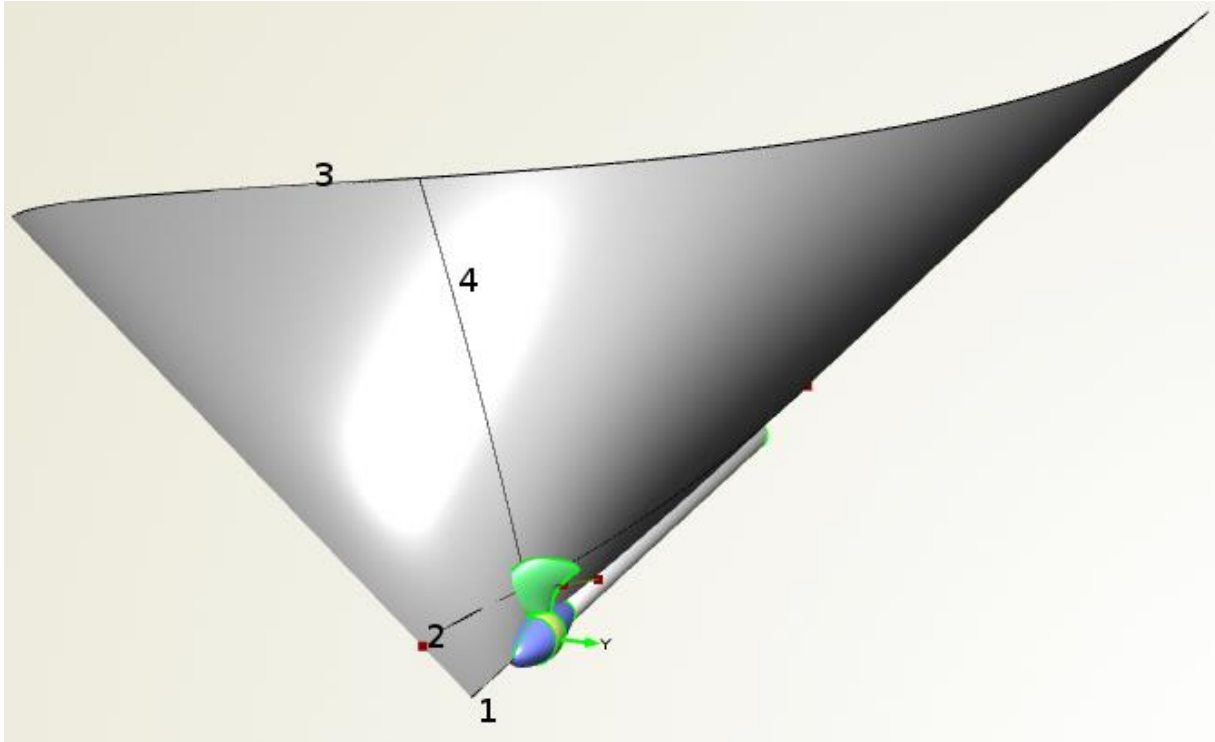


Figure 4.5 Domain construction

Two more curves are essential, one straight line passing through the center of the shaft and hub (curve 1 in Figure 4.5), and a curve defining the upper bound of the domain (curve 3 in Figure 4.5), constructed in a similar way as the curve 2. Intersecting the curves 1,2,3 at a specific position along the Z axis gives the curve 4. However, getting intersections at specific Z locations, generating the in-between surface patches and then merging them would not provide a smooth 3D surface. Also, preserving the continuity between the parts would be very difficult. That can only be accomplished with a FMetaSurface. Firstly, a Feature Definition needs to be created that given three curves, it will export an FIntersectionCurve, this can be done with a single line command inside the Feature Definition. This Feature Definition is then given to a FCurveEngine where it is connected with these 3 curves and a parameter that ‘runs’ from 0 to 1. Now, all the information needed to construct a surface is contained in this FCurveEngine.

The FCurveEngine is then given to a FMetaSurface that creates a fair and smooth surface by sweeping along these curves given some user defined boundaries, in this case, the location on the Z axis. By default, the generated meta surface information is used for the creation of a NURBS surface that can be visualized, used for more operations or get exported. With this FmetaSurface, two new surfaces are generated using the object FImageSurface, that transforms the initial FmetaSurface by rotating it around the Z axis, creating two identical surfaces that enclose between them one fifth of the whole domain. The FimageSurface object is a very useful tool, since it allows for all kinds of transformations. It can also be used to isolate parts of the surface that serves as a source, by specifying bounds for the U and V parameters.

With the two periodic surfaces that have been created, two Breps are generated illustrated with blue and red color in Figure 4.6. As a next step, a cylinder is created defining the boundaries of the computational domain. All the surfaces including the cylinder are contained into Breps and are distinguished with the use of different colors. Using colors to distinguish the different Breps is essential in the overall process, since they will be identified as different patches by cfMesh and OpenFOAM. This is quite helpful for the configuration of OpenFOAM for example for the setting of the boundary conditions.

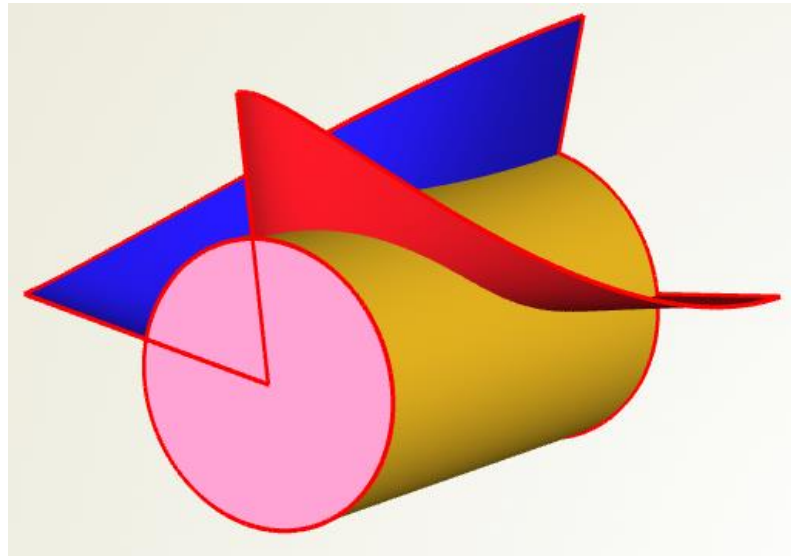


Figure 4.6 Domain construction

The inlet can be seen in Figure 4.6 colored with pink and it is placed $3R$ upstream, where R is the radius of the propeller. The outlet is placed $10R$ downstream and the cylinder radius is $7R$. The distances are selected so that the disturbance of the flow caused by the geometry does not influence the flow on the boundaries. They are based on experience and relevant work found at [35], [36] and [37].

At this point, all the geometry needed for the computations is created and the last part is combining them all together in a single watertight solid that will be exported as a Multicolor STL file. The Brep tessellation can be seen in Figure 4.7, one can notice the difference of tessellations between the red and the pink Brep. CAESSES offers a number of ways to control the tessellations of a Brep. The user can, for example, limit the value of the angle between two adjacent triangles or control the maximum edge length. By increasing the density of the tessellations, the quality of the final exported STL file is improved and performing Boolean operations between the different Breps is easier, however the final STL file becomes heavier and the visualization slower. Higher density is used at the blade edges, to capture the geometry sufficiently, and lower density is used at the inlet and outlet. The final solid illustrated in Figure 4.7 is exported as a Multicolor STL file and is ready for meshing with cfMesh.

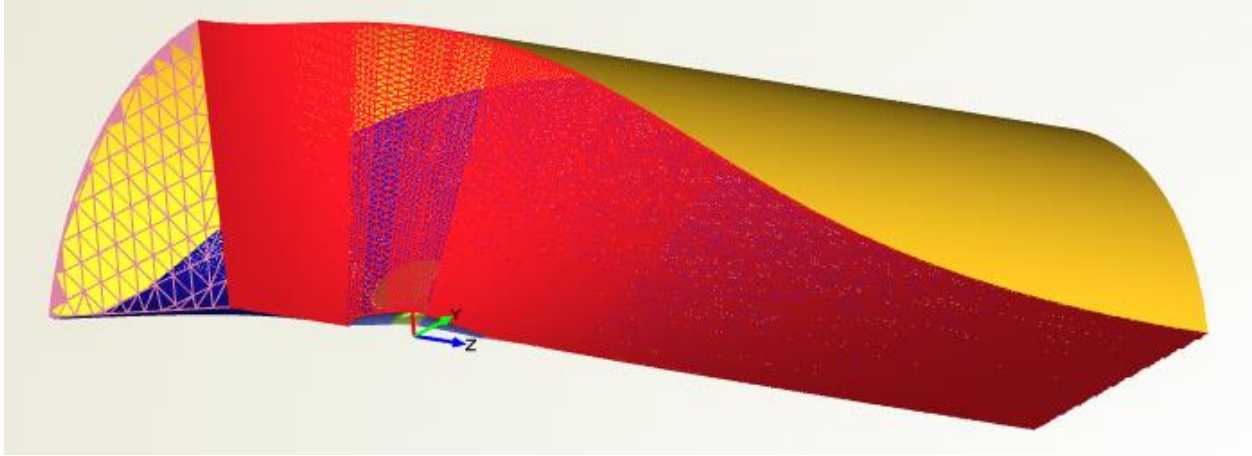


Figure 4.7 Final STL to be exported

4.3 Grid Generation

According to [38], grid generation is estimated to take up to 80% of the whole analysis time in industrial applications. Moreover, the mesh quality has a great impact on CFD results and special attention should be paid in this part of the process. In the present work, a significant amount of time had to be spent to configure cfMesh properly to achieve the desirable grid quality.

Meshing a geometry like the propeller blade with sharp edges is quite demanding, especially with an open-source software like cfMesh. Most commercial unstructured grid generators can detect automatically the surface curvature and adjust the density of the surface mesh to capture the edges and maintain the mesh quality. However, cfMesh does not support this functionality and the control of the cell size in specific regions has to be assisted, utilizing the capabilities of CAESSES.

Using cfMesh is quite straightforward and simple geometries can be meshed easily. CfMesh is executed with a terminal command in a folder with a proper OpenFOAM folder structure, containing the STL file that is going to be meshed along with a C++ dictionary file called **meshDict**, contained in the **system** folder. The open-source version of cfMesh comes with four available meshing workflows [33], these are Cartesian, 2D Cartesian, Tetrahedral and Polyhedral. Excluding 2D Cartesian, all the other meshes have been tried and tested for the task at hand. As the names suggest, the tetrahedral workflow creates meshes consisting of tetrahedral cells and Polyhedral creates meshes consisting of arbitrary polyhedral cells. The Cartesian workflow proved to be the most efficient. The generated mesh consists predominately of hexahedral cells with polyhedral in the transition regions between cells of different size.

The grid generation process is controlled with inputs in the **meshDict** file. Two mandatory settings need to be set, the name of the file that is going to be meshed (**surfaceFile**) and the default size used for the meshing job (**maxCellSize**). With these basic settings, **cartesianMesh** can be executed and ran, however the mesh quality will most likely suffer. A number of options are available to control the cell size locally and improve the mesh quality.

As stated above, the meshing process is assisted by CAESES. In CAESES, the surfaces are parametric, with U and V parameters running in the domain $[0,1] \times [0,1]$. Using the FimageSurface object with specified U-V bounds, the propeller blade is split, with the leading edge, trailing edge, fillet and tip separated and contained into different Breps, that will be eventually recombined.

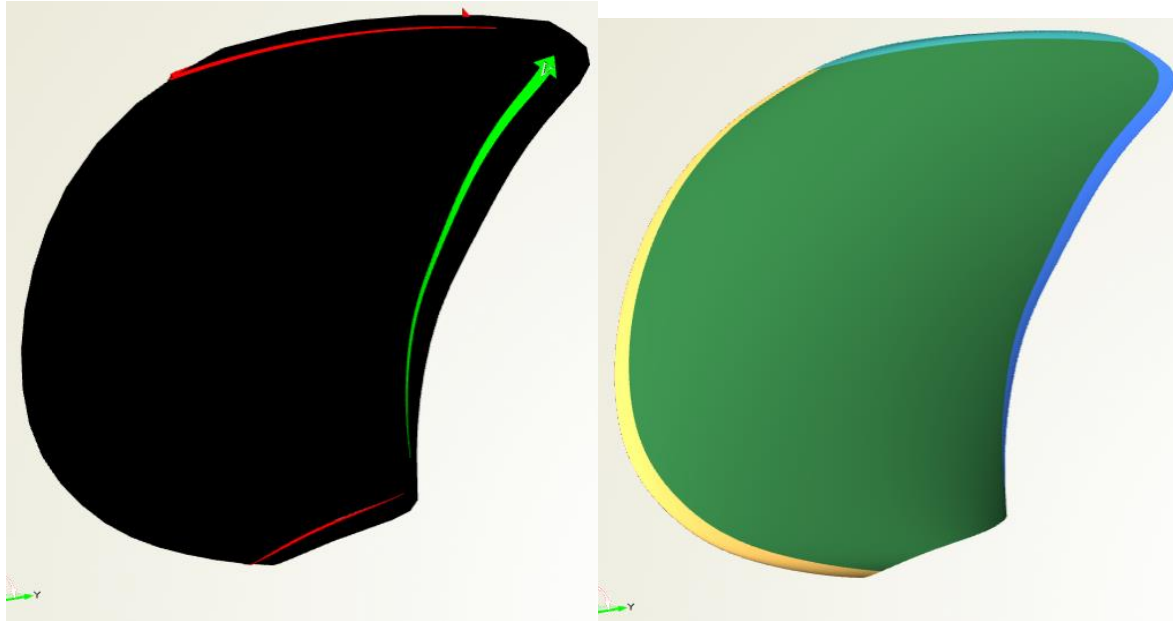


Figure 4.8 Geometry preparation

In Figure 4.9, one can see how the cell size is controlled. The parts distinguished with colors are identified as patches by cfMesh and the cell size can be controlled on each patch with entries in the **meshDict** file. The difference in the cell size is visible between the patches, small quadrilateral faces are generated on the leading edge and tip colored with purple and white, while larger ones on the central area of the blade, and the shaft.

Fluid flows close to walls are characterized by high gradients [39]. With unstructured grid generators, prism layers are usually used to resolve the boundary layer, and they are very important for the accuracy of the computations. The thickness and number of prism layers is chosen depending on the turbulence model, wall functions and Reynolds number. But this topic is described in detail in the Wall Treatment section. CfMesh is able to generate prism layers at the walls, using the appropriate commands in the **meshDict** file, where a number of options are selected. However, once the generation of prism layers is activated, the quality of the generated mesh deteriorates significantly, and extra care needs to be taken. An iterative procedure is needed, where almost all the possible combinations of parameters are tested.

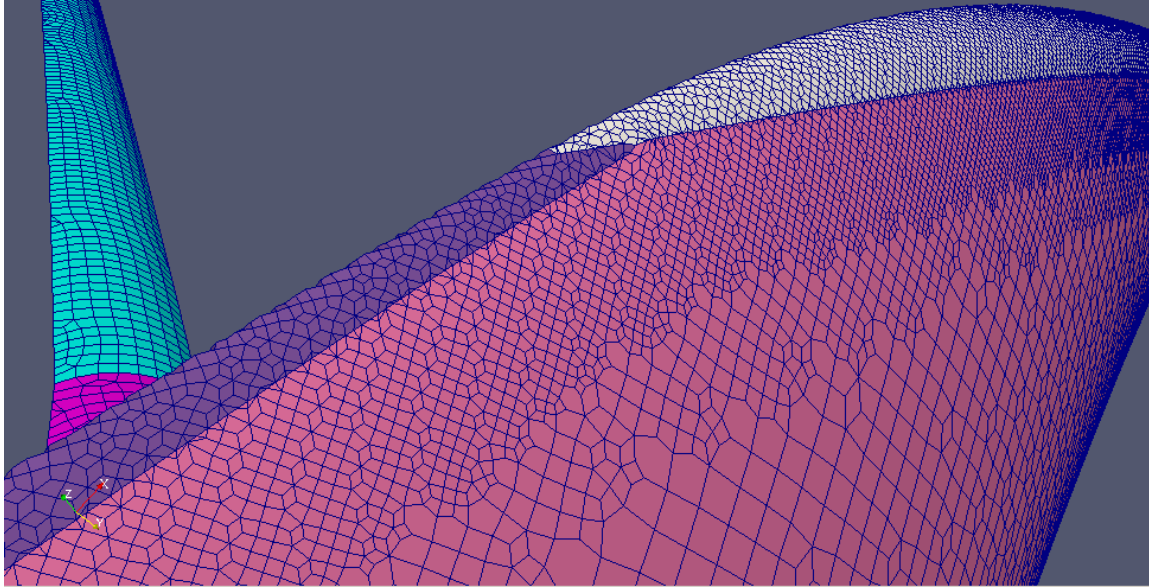


Figure 4.9 Surface mesh on the propeller blade and shaft

While cfMesh is running, it provides information about its behavior and its internal optimization algorithms. For example, the number of skewed cells it identifies and the optimization loops used to get rid of them. With the proper commands, the mesh generation procedure can be stopped, and the mesh can be visualized at a specific time of the process. In this way, the user can identify the root of the low-quality cells and make the appropriate changes in the settings.

The lack of GUI makes the overall process quite demanding. It is a trial and error process, carried out by applying changes to the **meshDict** and monitoring the quality of the generated mesh and the convergence of the simulations. The mesh quality is evaluated using the command **checkMesh -allTopology -allGeometry**. The output is a detailed quality report, including all kinds of statistics, like the number of cells of each type (hexahedra, polyhedral, prism) and the quality constraints that are violated, non-orthogonality, aspect ratio, skewness and so on. The results of the report of the mesh used are presented in the Validation section.

The mesh is also inspected visually, using the OpenFOAM's post-processing software Paraview. Most low-quality cells are detected at the sharp edges of the geometry, these are the leading edge, trailing edge, fillet and tip. In the Figure 4.10 below, one can see the low-quality cells generated at the leading edge, colored with red and blue.

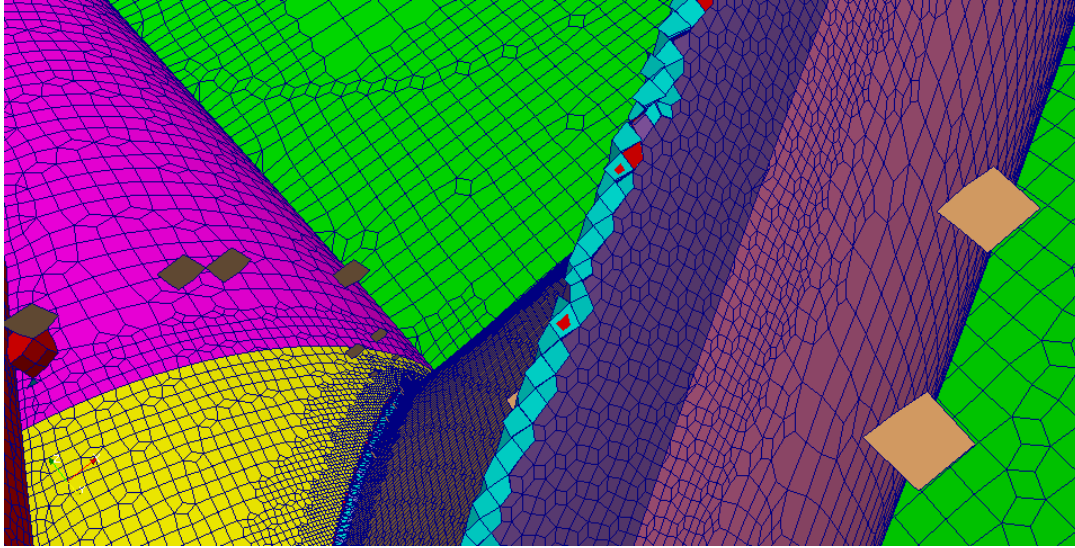


Figure 4.10 Monitoring the mesh quality in Paraview

Ideally, ten to twenty prism layers are desired to resolve the boundary layer, and achieve accurate results [40], [41]. In this case, increasing the number of prism layers, reduces the mesh quality significantly. After many iterations, a maximum number of five prism layers could be generated with the mesh quality kept at acceptable levels. The prism layers are illustrated in Figure 4.11, where a view of the leading edge and a section of the 3D grid is depicted. One can notice the difference in height from the last prism layer to its neighbor cell. A smoother, more gradual transition is desired.

CfMesh offers a number of utilities [33], i.e. mesh manipulation and conversion routines. The **improveMeshQuality** utility is thoroughly tested for this application. The smoother that is applied actually ensured a gradual transition from the last prism layer to its neighbor cell, but required about one hour to execute and had insignificant impact on the calculated forces, it was therefore not used.

A smooth transition in cell size is also needed within the domain, from the very small cells of the edges to the much larger cells at the inlet and outlet. To achieve this, refinements are specified by activating the **refinementThickness** option that is applied on specific patches. The effect of these refinements is depicted in Figures 4.11 and 4.12. This option controls the distance of the wall that the specified on the patch cell size, is kept constant. These options not only ensure a smooth transition in cell size, but also make sure that a sufficient amount of high-quality hexahedral cells will be placed at the vicinity of the propeller blade, keeping the transition areas that are filled with polyhedral cells, away from the wall. These polyhedral filled transition regions can be seen in Figures 4.11 and 4.12.

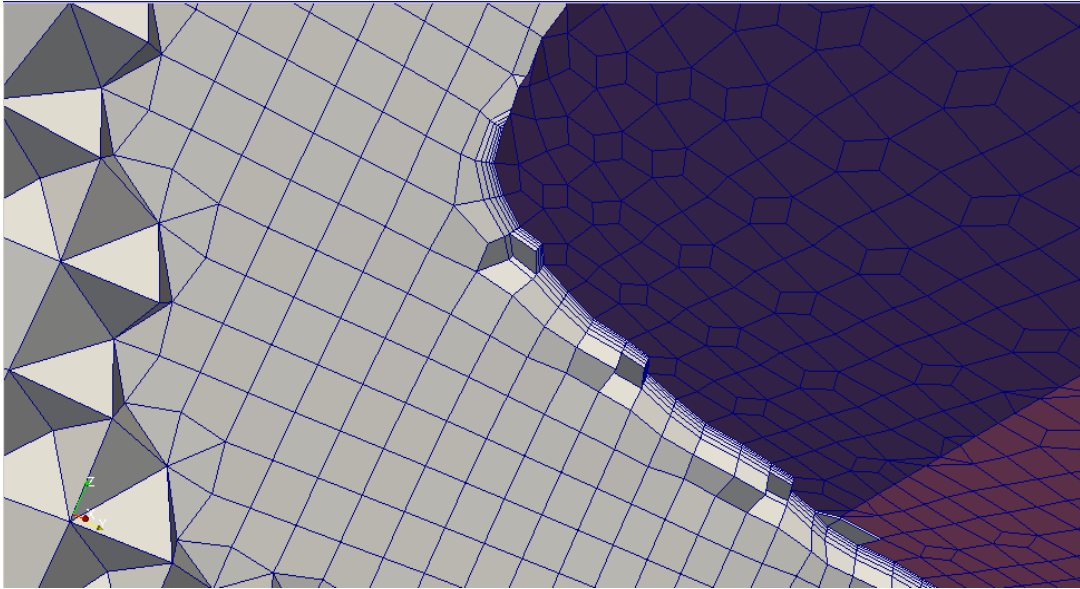


Figure 4.11 Prism layers at the wall

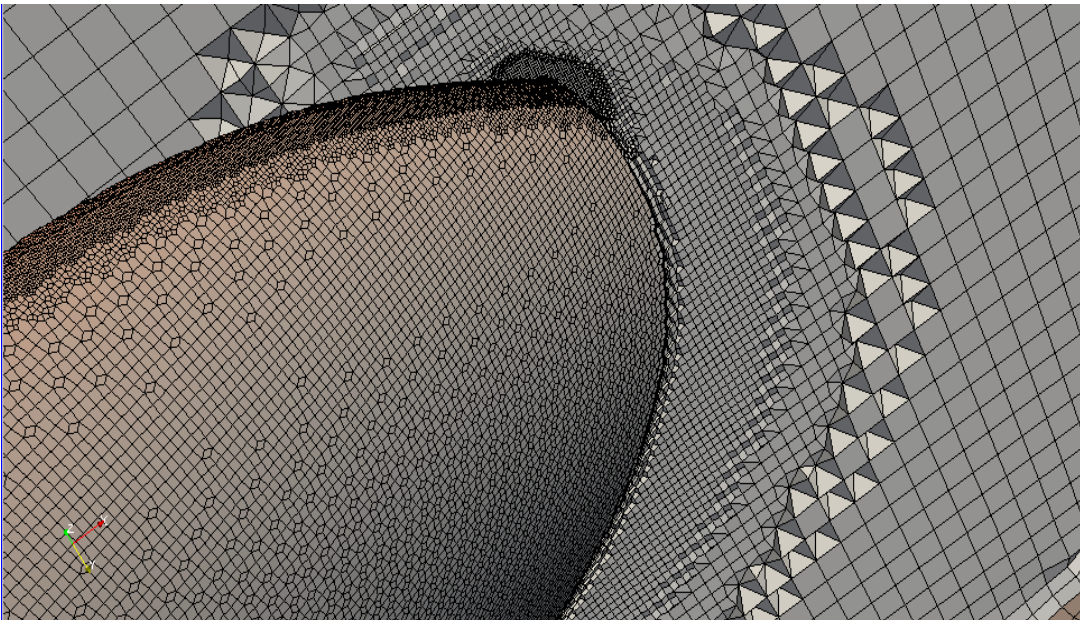


Figure 4.12 Refinements near the tip

In Figure 4.12, the refinements near the tip region are visible, they are needed not only to improve the mesh quality, but also to capture the trailing vortices.

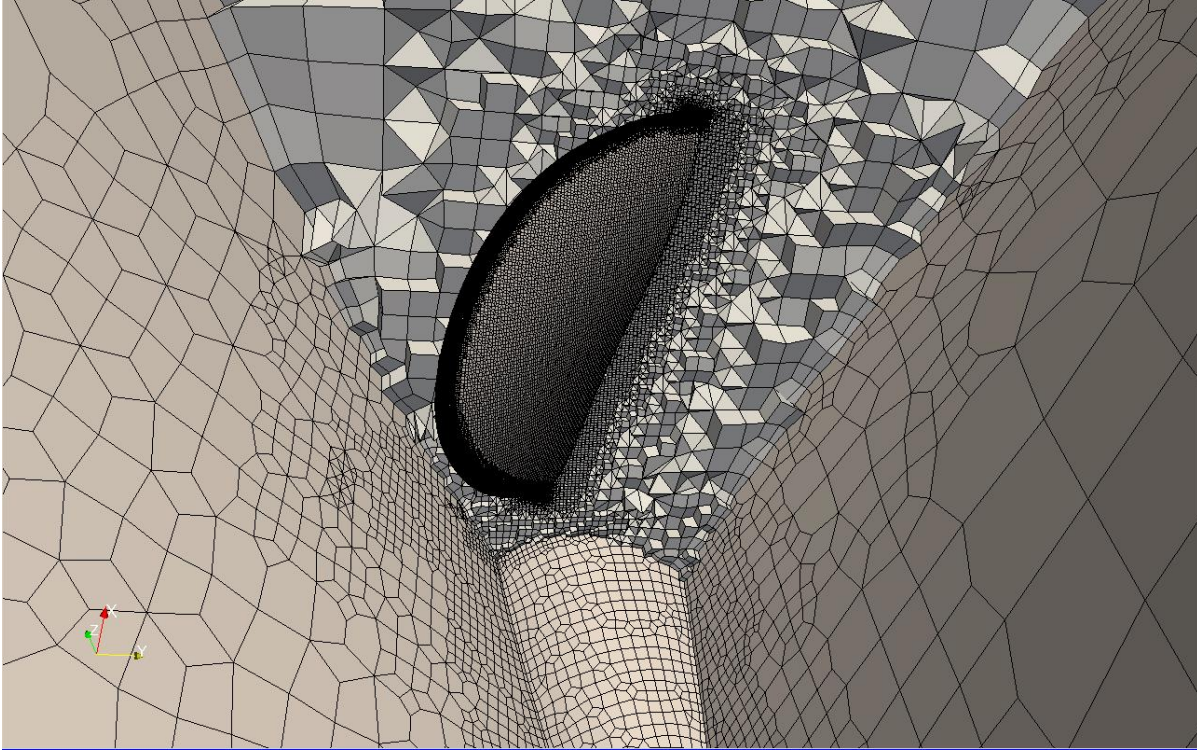


Figure 4.13 Volume mesh around the blade without cylinder volume refinements

Periodic boundary conditions are applied to the sides of the domain. The boundary condition typically used in OpenFOAM to couple periodic sides is called **cyclic**, and the faces on each coupled patch must have the same topology. In our case, due to the nature of the grid generator and the geometric form of the domain, the faces of the cells at the periodic boundaries do not match completely. The Potsdam Propeller has five blades, and even with the flexible domain that is used the leading and trailing edge are quite close to the periodic sides. As stated above, small cells are used at the leading and trailing edge of the blade and this results in small cells on the left periodic side near the leading edge, and small cells on the right periodic side near the trailing edge. This mismatch is depicted in Figure 4.13 above. Consequently, the resulting surface meshes on the periodic sides are quite different and the **cyclic** boundary condition cannot be applied.

The specific boundary condition that is applied is called **cyclicAMI**, where AMI stands for arbitrary mesh interface. This boundary condition was introduced in OpenFOAM v2.1.0. to enable simulation across non-matching adjacent interfaces. With the AMI procedure, each face accepts contributions from the partially overlapping faces of the neighbor patch, with weights defining this contribution as a fraction of the intersecting areas. For each face, a sum of weights close to 1 is needed to ensure the convergence of the simulations. [42]

However, due to the mismatch of the periodic sides, the sum of weights deviates from 1 causing the cyclicAMI condition to fail and the simulations to diverge. To overcome this difficulty, a cylinder volume refinement is specified near the propeller blade illustrated in Figure 4.14. This refinement constraints the cell size inside a specified cylinder and the matching of the periodic sides is achieved. Moreover, it helps to capture the complex flow phenomena around the blade more accurately and ensures a smooth transition in cell size.

All these refinements used to increase the mesh quality, also increase the cell count and the simulation time. A reasonable compromise needs to be made between the mesh quality and the computational cost. The number of cells that is finally used for the validation study is selected with a grid dependency study that is presented in the Validation section. Another difficulty is that the mesh settings selected have to ensure flexibility and stability during the optimization phase, where a number of different variants will be created and cfMesh will have to generate a grid of adequate quality for each and every one of them.

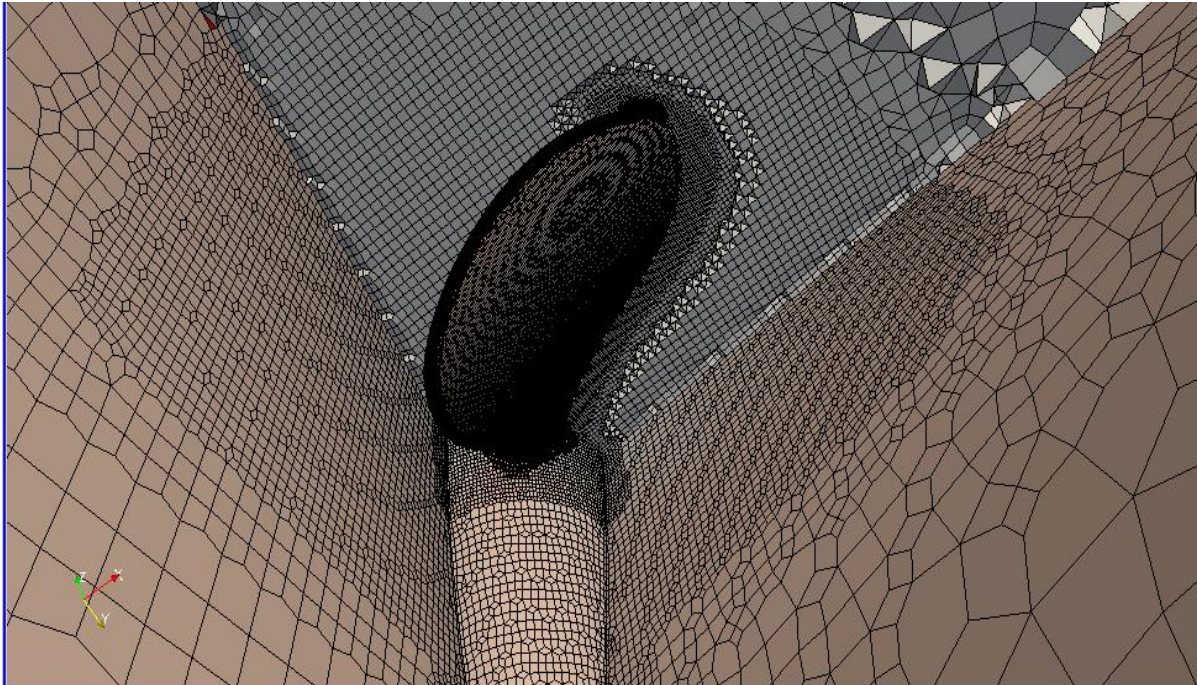


Figure 4.14 Volume mesh around the propeller with cylinder volume refinements

4.4 CFD Setup

From the solvers available in OpenFOAM, simpleFOAM is selected, a steady-state solver for incompressible flows with turbulence modelling, using the SIMPLE algorithm to couple pressure to velocity [43]. This section contains the settings of the simpleFOAM solver.

4.4.1 Flow conditions

The flow conditions for the open water experiment are chosen to match the ones provided by SVA for the open water test in the towing tank [34]. These conditions can be seen in Table 4.2 below.

Table 4.2 Flow conditions

Water density (for T=17.5°)	ρ	[kg/m ³]	998.67
Kinematic viscosity of water (for T=17.5°)	ν	[m ² /s]	1.07E-06
Rate of revolutions	n	[1/s]	15
Advance Velocity	V_A	[m/s]	0.75-5.25

4.4.2 Boundary Conditions

The equations of fluid motion will be solved for a certain computation domain that is restricted by boundaries. Boundary and initial conditions need to be set on them. The equations will be solved under the specified conditions.

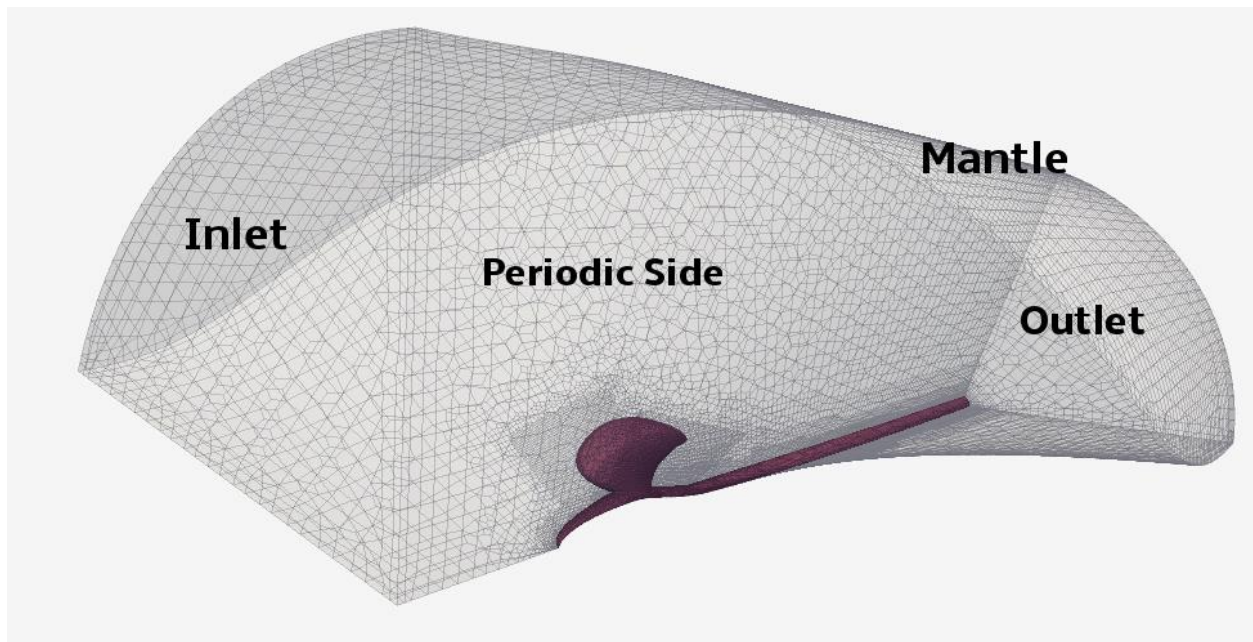


Figure 4.15 Boundary Conditions

The Dirichlet boundary condition is applied on the inlet for velocity, defining a constant value to match the advance coefficient that is simulated. The simulations are executed for a wide range of advance coefficients with the revolutions kept constant while the inlet velocity varies.

The no slip boundary condition is applied on the propeller, hub and shaft. It ensures that the fluid sticks to the wall and moves with the same velocity as the wall. As a common practice for incompressible solvers, the pressure on the outlet is set to a fixed value of 0. The cyclicAMI boundary condition is applied on the periodic boundaries. In OpenFOAM, the Dirichlet boundary condition is expressed as **fixedValue**, whereas the Neumann condition is expressed as **zeroGradient**. The slip condition is applied on the mantle. It defines the velocity component normal to the wall as zero, preventing the flow to pass through it, and the tangential to the wall component of velocity as zeroGradient. The pressure and velocity boundary conditions for all boundaries can be seen in Table 4.3 below.

Table 4.3 Boundary Conditions for pressure and velocity

Boundary	Velocity boundary condition	Pressure boundary condition
Inlet	Dirichlet	Neumann
Outlet	Neumann	Dirichlet
Propeller	No Slip	Neumann
Left Periodic Side	CyclicAMI	CyclicAMI
Right Periodic Side	CyclicAMI	CyclicAMI
Mantle	Slip	Neumann
Shaft	No Slip	Neumann

Furthermore, initial values for the turbulent quantities need to be set on the boundaries. They can be estimated by restricting the turbulence intensity between 0.5% and 1%. Also, the turbulence viscosity ratio approximately 10 as seen below. The values depend on the fluid velocity, so they are recalculated for every advance coefficient.

$$0.5\% < I = \frac{\sqrt{\frac{2}{3}k}}{|U_{inlet}|} < 1\% \quad (4.1)$$

$$\frac{\nu_T}{\nu} = \frac{k}{\nu\omega} \approx 10 \quad (4.2)$$

To measure the thrust force and torque acting on the blades and obtain the open water characteristics K_T and K_Q , a function needs to be defined in the **controlDict**.

As discussed in the Theory Section, the MRF approach is used to model the rotation, also called the “Frozen Rotor Approach”. For propeller simulations with the MRF approach (Multiple Reference Frames), the mesh is usually divided into two different cylindrical domains, one moving and one stationary. However, the interface between the rotating and stationary regions needs special treatment. To avoid this, the entire domain is referred to as a single moving reference frame. Depending on the solver, there are different ways to set up the MRF case [44]. In the present work, the rotation is included by configuring a dictionary called **MRFProperties** located in the **constant** folder. Inside this dictionary, rotational movement can be applied on a specified **cellZone**, by defining the rotational velocity (**omega**), the position of the axis of rotation (**origin**), and the directional vector of the axis of rotation (**axis**). Since the entire domain is to be included to a single moving reference frame, the entire domain needs to be contained in a **cellZone**. To achieve this, the **topoSet** utility is used. It can manipulate **cellSets**, **faceSets** and **pointSets**, with proper settings in a dictionary called **topoSetDict** that is placed in the system folder.

Part of the **MRFProperties** file can be seen below, omega is the specified rotation in radians per second, here corresponding to 15 revolutions per second as the PPTC case. Non-rotating patches are also specified as seen below.

Code 1. MRF Dictionary

```
MRF1
{
    cellZone    rotating;
    active      yes;

    nonRotatingPatches (inlet mantle outlet left right);

    origin      (0 0 0);
    axis         (0 0 1);
    omega        -94.25;
}
```

4.4.3 Wall Treatment

The flow near the wall is the most interesting part of fluid simulations, but the most difficult to predict. As stated in the Theory Section, there are two approaches to model the near-wall region. Either use wall functions, semi-empirical formulas that are able to handle the region between the wall and the fully-turbulent region at low cost, or place a sufficient number of short -in the wall normal direction- cells to resolve the boundary layer up to the wall.

In optimization studies, the wall function approach is usually selected, since the shorter computation time allows for more designs to be explored. Depending on the approach followed, a suitable wall mesh resolution is selected by controlling and monitoring the dimensionless quantity y^+ that expresses the distance of the center of the first cell to the wall. Values of y^+ values over 30 are desirable with the wall function approach. With automatic grid generators, controlling the height of the boundary layer cells for curved geometries can be quite challenging and achieving the proper y^+ is difficult but important for the accuracy of the results.

The flow field needs to be calculated first in order to calculate the dimensionless quantity y^+ , so a number of simulations are executed to properly set the y^+ values on the propeller blade. In a first approach, y^+ values over 30 are desired. These y^+ values can be acquired without the use of prism layers easily, and simulations converge quickly. The results however deviate from the experimental measurements, since prismatic layers are essential even with the use of wall functions. Ideally 10 prism layers would be sufficient for a wall function mesh with y^+ values kept over 20-30 for the main area of the blade.

Trying to generate prism layers and control the y^+ on the blade surface introduced a number of problems that have been analyzed in the Grid Generation section. When the prism layers are activated, very small cells are needed at the edges of the blade to maintain the mesh quality, and the prismatic cells generated are very short corresponding to low y^+ values. CfMesh offers three main options to control the generation of prism layers. The number of the layers, the thickness ratio and the maximum thickness of the first layer. The number of layers is restricted to 5 to keep the mesh quality to acceptable levels. The thickness ratio controls the thickness between two successive layers. This option does not offer much flexibility either, a value of 1.3 was selected as the maximum possible. Controlling the maximum thickness of the first layer does not help at all in the specific case. On the other hand, controlling the minimum thickness could help increase the wall distance and y^+ values, but this option does not exist, and the distance of the first cell from the wall can only be controlled with the cell size. Taking the above under consideration, and given the Reynolds number of the simulation, y^+ values over 20-30 are impossible to achieve with prism layers.

That being said, the second approach is followed. According to [45], placing the first cell in the buffer layer should be avoided. To achieve this, small cells are used and the five prism layers that can be generated are placed in the viscous layer, keeping the y^+ values below 2.

4.5 Building the Parametric Model

The first set of computations for the validation study was executed using the original geometry provided by the SVA. For the optimization study that will follow, a fully parametric propeller blade needs to be developed that will match the original geometry. To make sure that the baseline design of the parametric model matches the original propeller, simulations are executed for all advance coefficients again. In this section, the construction of the parametric blade is discussed.

CAESES offers a dedicated entity for blade design called FGenericBlade. With this object, arbitrary user-defined radial functions for rake, skew and pitch can be combined with arbitrary profile definitions. In this way, a fully parametric propeller blade can be created in a few minutes. The software also provides a selection of commonly used profiles like NACA profiles or NACA Four-Digit-Series. For the purpose of this thesis, the profile used is custom made.

Blade design in CAESES initiates with the definition of the profile which is encapsulated in a Feature Definition, in a small piece of code. Firstly, both the camber line and the thickness distribution are defined with B-Spline curves. As a next step, given the camber line, two offset curves are created using the thickness distribution. The offsets are then scaled on the X axis according to the chord. The parameterization lies in the fact that the coordinates of the control points of the curves as well as the chord are arguments in a code contained in the Feature Definition. The coordinates of the control points are multiplied with coefficients. Later on, the coefficients will be adjusted to make sure that the resulting blade matches the original geometry as close as possible.

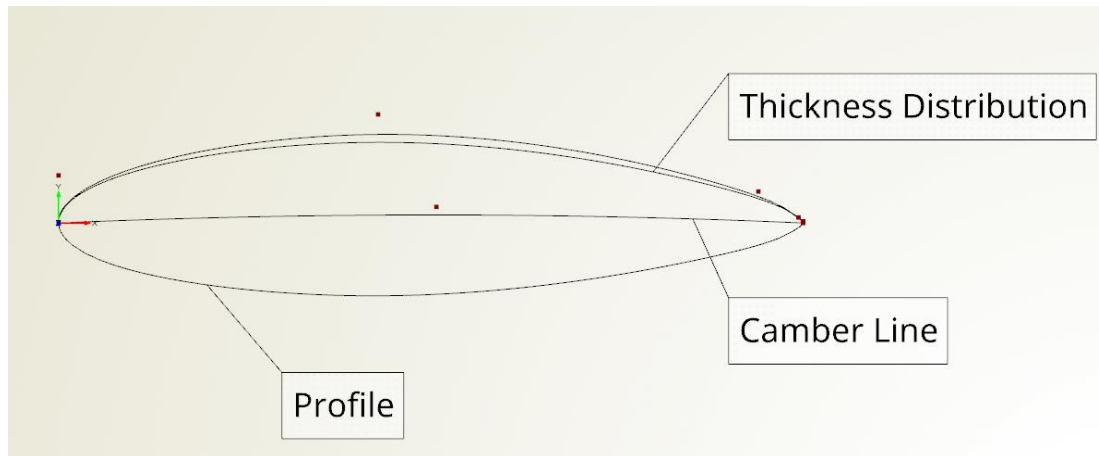


Figure 4.16 Profile definition

The next step is to set up a FCurveEngine by means of the recently created Feature Definition that contains the profile definition. The profile is selected as the base curve of the FCurveEngine. In this engine the profile's scalar parameters are linked to radial distributions i.e. camber, camber position, maximum thickness and so on. These radial functions will determine the coordinates of the control points contained in the Feature Definition, thus defining how the sections will form in the 3D space.

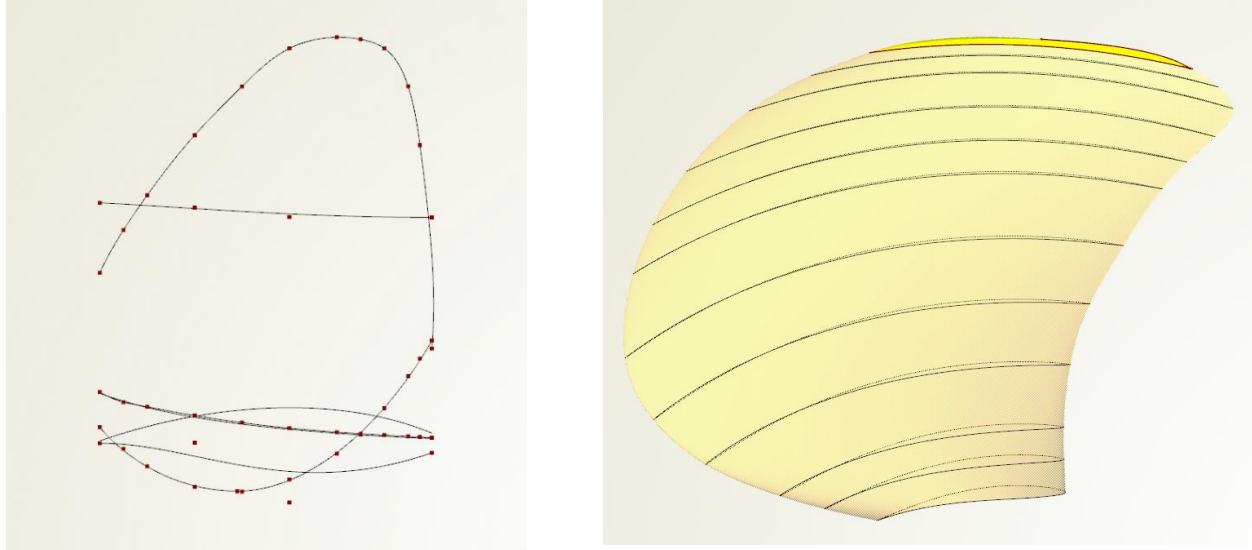


Figure 4.17 Radial distributions (left), Parametric blade with tip (right)

Any curve type available in CAESES can be used as radial function. The functions have to be designed in the normalized space "[rhub,1.0]" where "rhub" is the normalized hub radius. They are defined in the xy-plane in the range $[0,1] \times [0,1]$, where the x-coordinate corresponds to the normalized radius. The rake function is expected to be normalized with respect to the real radius. The skew radial function can be defined either by the angle ' θs ', or the distance ' s ' of the mid-chord position to the projected flow axis. The radial function for the pitch value can be defined either as a function of the pitch angle ' Φ ' using degree or as the common value ' P ', i.e. the distance moved forward by the helical line during one revolution. The pitch value ' P ' is normalized with respect to the radius. [29]

The FGenericBlade object can now be used. Within this object, the radial distributions of pitch, skew and rake will be linked to the FCurveEngine, that carries the hydrofoil related information. The parametric blade object is depicted in Figure 4.17 on the right. It spans from the "rhub" to 0.99 times the radius leaving the tip area empty. A special feature is available that creates a smooth tip, preserving the continuity at the edges and making sure that no gaps exist. The smooth tip is also easier to mesh, it can be seen in Figure 4.17 marked with yellow.

As mentioned above, the geometry of the PPTC is available online and it includes the radial functions that are needed at this point. Some pre-processing is necessary to adjust them in the format that CAESES expects. As extracted for example from the propeller drawing, the radial functions are actually just a number of points. These points are interpolated with the object FInterpolationCurve, and these curves are then linked to the FCurveEngine and FGenericBlade.

In our case, the radial functions are known in advance. If this wasn't the case, CAESES offers another powerful tool called Blade Analysis Tool. With this feature, provided a list of surfaces that define a blade, CAESES automatically constructs a fully-parametric blade including the radial distributions.



Figure 4.18 Trailing edge fillet (left), Hydrofoil Section comparison (right)

So far, a fully parametric single blade is created with U and V parameters running from zero to one, U in the chordwise direction and V in the radial direction. The blade can now be scaled according to the radius. The blade is compared with the original in two ways, by comparing the 3D models in the 3D view but also by comparing the 2D profiles using the PFF propeller format. The 3D comparison helps to identify geometry deviations related to the pitch, skew and rake distributions. The radial distributions for these geometric features are finetuned to increase the matching. Propeller Free Format (PFF) is a common way to exchange propeller geometry data. CAESES supports the functionality of importing and exporting PFF files. The PFF file that is provided by the SVA is imported into CAESES and the 2D profiles can be visualized in 2D. At the same time, the profiles of the parametric model can be visualized in 2D and be plotted against the PFF ones as seen in Figure 4.18 on the right, with the colorful hydrofoils representing the original PFF sections. The coefficients defined inside the Feature Definition are adjusted to match the original hydrofoil sections.

At Figure 4.18 on the left, a view of the trailing edge is depicted. The original PPTC propeller features a very thin trailing edge. To properly mesh this trailing edge and successfully generate prism layers, the cell size along this edge had to be reduced significantly, increasing the total number of cells. In the parametric model, the trailing edge is designed to be blunter and more curved, making the prism layer generation easier for cfMesh. To investigate the influence of this change on the calculated forces and torque, the computations are executed again. Comparing the results shows that the influence is insignificant and the new trailing edge can be used for the optimization study. This change helped to decrease the computational cost significantly, bringing the cell count down from approximately 2.700.000 cells to 1.800.000 saving up to three hours of simulation time for each design.

At this point, the radial distributions are Intersection curves passing through the data points. This however is not particularly helpful for the optimization study that will follow, where a number of parameters will be activated and a number of designs will be explored. Usually, there are eleven to twelve points defining each radial distribution, corresponding to values of 0.2,0.25,0.3...0.9,0.95,1. Introducing parameters to control the abscissas or ordinates of these points would give at least 10 parameters for each distribution, that means up to 50-60 parameters for the whole blade. This number of parameters defines a design space impossible to handle, even with substantial computational power. That being said, the intersection curves need to be approximated by curves that can be controlled with fewer parameters. We need as few parameters as possible with as much control on the curve as possible. At the same time, a selection needs to be made on which radial distributions will be activated for the optimization. Even if only 2 parameters are activated for each distribution, there are seven distributions, these are functions controlling the camber of the sections, the position along the chord where the specified camber is applied, the chord length, the pitch-ratio, the rake, the skew and the thickness. Experience at Friendship Systems suggests three functions to be enabled, these are the pitch, camber and thickness.

The camber distribution along the blade is approximated with a 3rd degree B-spline with 4 control points as seen in Figure 4.19 below. Two parameters are introduced, the first one controls the abscissa of the 3rd control point, and it is named **camber1**. The movement is indicated with red arrows in Figure 4.19 below. The second parameter controls the ordinate of the 2nd control point (**camber2**), the movement is indicated with the green arrows below.

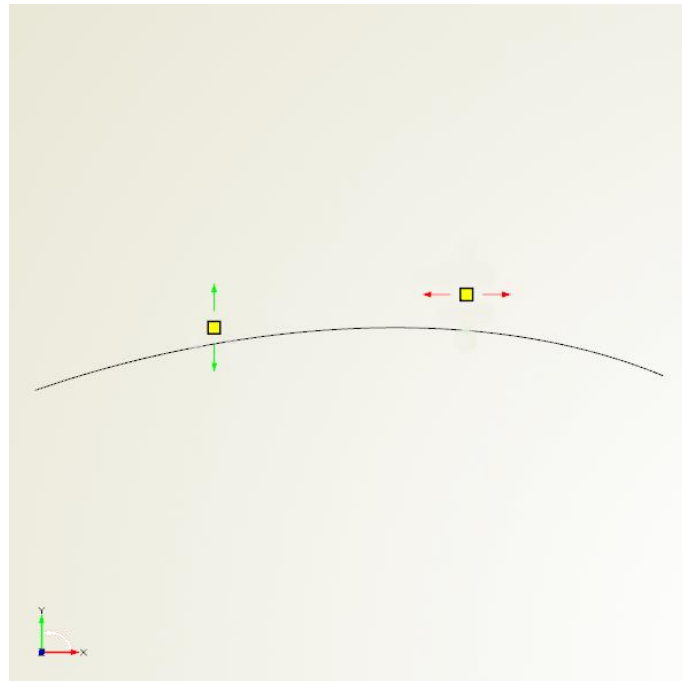


Figure 4.19 Camber distribution

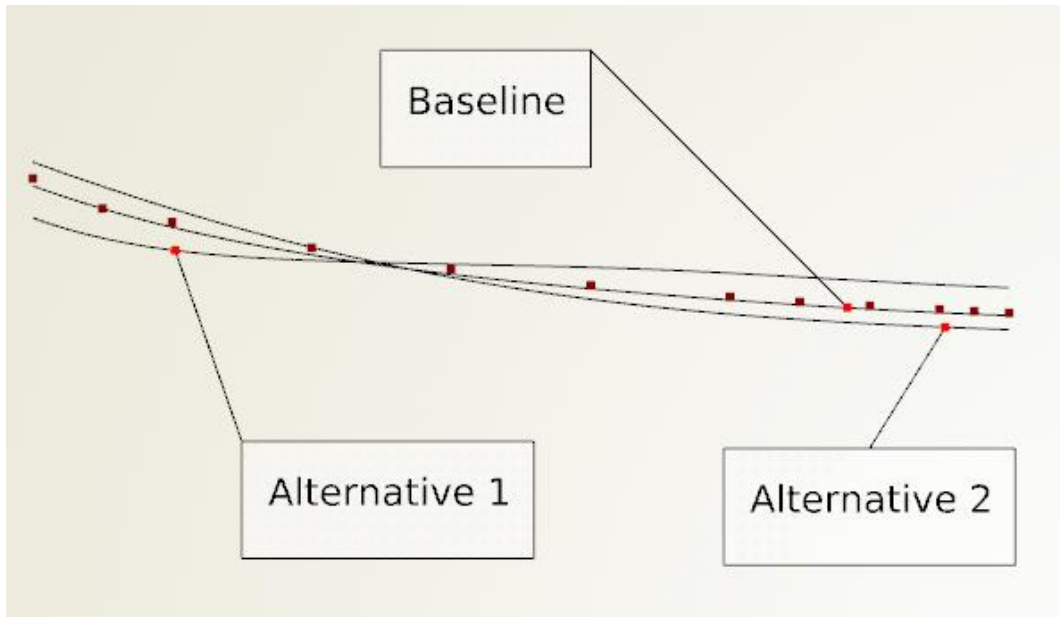


Figure 4.20 Thickness distribution

A feature called FCurveGenericBasic is used to approximate the pitch ratio and the thickness distribution. The feature creates a multi-segmented planar curve where the plane and elevation, number of intermediate points, abscissa and ordinate values for every point can be controlled. Tangents at every point and fullness values for the single curve segments can be set as well. The feature helps to achieve maximum control on the curve with as few parameters as possible. At first, the parameters of the feature are adjusted to fit the baseline curve to approximate the original data points, as seen in Figure 4.20. Two parameters are then activated as design variables, these are the ordinates at the start (**thicknessStart**) and end (**thicknessEnd**) of the curve. In order to visualize how these parameters affect the 2D curve, two random alternatives are presented in Figure 4.20. In the first alternative, the ordinate at the start of the curve is reduced, whereas the ordinate at the end is increased. The exact opposite is illustrated with alternative 2. One can notice how reducing the ordinate at the start of the curve affects the whole curve up to the intersection with the Baseline curve. Same goes for the parameter controlling the ordinate at the end.

A similar strategy is employed for the pitch ratio distribution. In this curve however, an intermediate point needs to be defined. The baseline design is then fitted to approximate the original points, by adjusting the ordinates at the start and the end, the tangents at the start and the end as well as the fullness of the two parts. The intermediate point is depicted with a yellow mark in Figure 4.21. In this case, two parameters are activated, they control the abscissa (**pitchMaxPos**) and ordinate (**pitchMax**) of the intermediate (yellow) point. Two alternatives are illustrated in Figure 4.21. Alternative 1 is generated with a decrease in abscissa and an increase in the ordinate. The exact opposite is carried out to generate the Alternative 2.

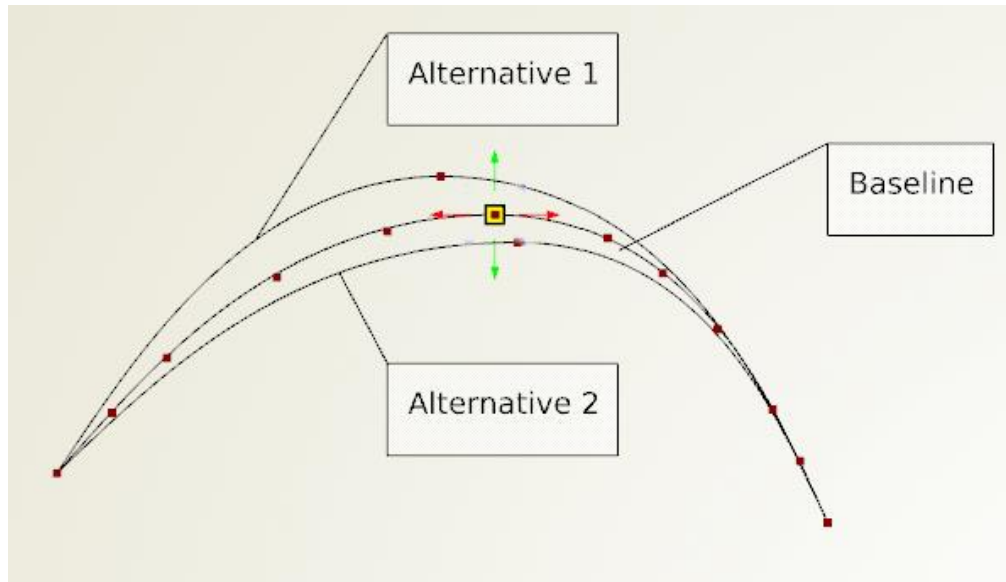


Figure 4.21 Pitch Distribution

At this point, the parametric blade has been created. New parametric surfaces for the hub, cap and shaft are created as well. No changes will be applied on them during the optimization phase, they were created just to make the assembling into a single Brep easier. For their creation, the FCurveGenericBasic feature is used. The parameters are adjusted to match the profile of the original hub and shaft. Surfaces of revolution are then created by rotating the 2D curve around the Z axis. In order to prepare the geometry for cfMesh, the blade is treated exactly as the original geometry. The blade is split, the edges are separated using the FImageSurface object and then they are all recombined into a single watertight Brep as seen in Figure 4.22 on the right, with a fillet operation between the blade and the hub. The blade is then extracted from the periodic domain and the domain including the blade is exported as a colorful STL file.

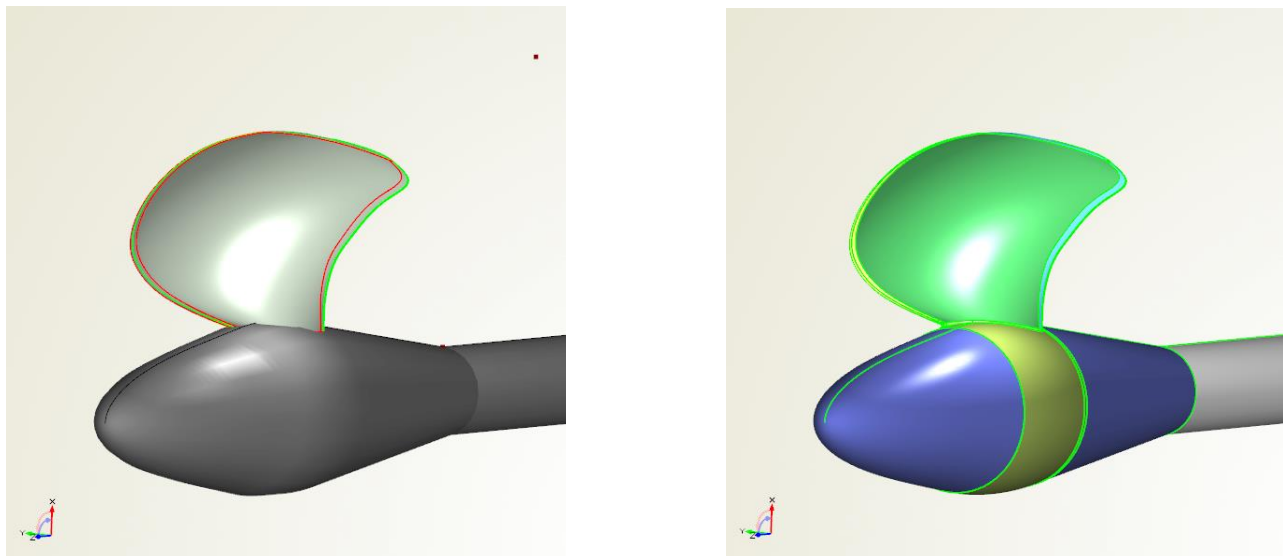


Figure 4.22 Watertight parametric blade

4.6 CAESES - OpenFOAM coupling

In order to run an any optimization algorithm, the whole process of creating the geometry, setting up the simulation case, creating the mesh, running the simulation and finally extracting the simulation results has to be completely automated. The parametric model has already been created and upon request, any design variant can be generated and be exported in any file format. Now, in order to contain the whole process into a closed loop, the Software Connector available in CAESES is used. The Software Connector can be seen in Figure 4.23 where the parametric geometry shown as the 'STL' box, is coupled to the input dictionaries that OpenFOAM requires in order to successfully execute and run. Result files and values are connected as well.

The simulation has to be executed manually once, and then any simulation output data can be connected as result values, for example the calculated forces included in the forces.dat file shown on Figure 4.23. In the same Figure, one can notice how the calculated forces are extracted from the output files. Two parameters, eval_Tpress and eval_Tvis are used to extract the pressure and viscous forces calculated by OpenFOAM with the command `Runner01.getResults().getTable("postProcessing/forces/0/forces.dat").getElementAt("Tpress")`. At the top left side of the same Figure, these parameters are used to calculate K_T .

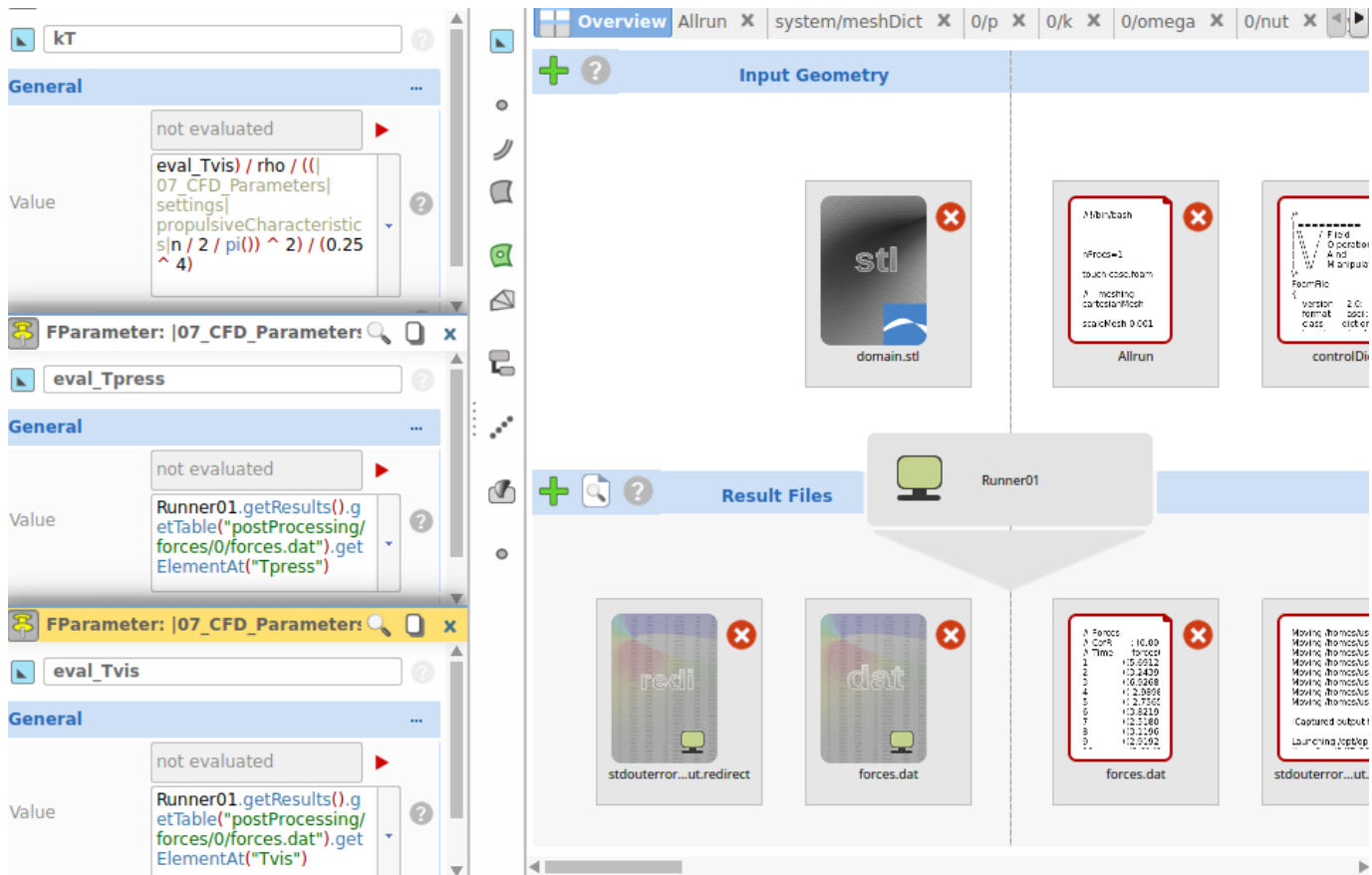


Figure 4.23 Software connector

As discussed in Chapter 3, OpenFOAM requires a proper directory structure to execute, containing the files that are necessary to trigger the OpenFOAM utilities. With all the connections properly set, the Runner shown in the middle of Figure 4.23 reproduces this folder structure for every variant. For every design we ask for, the Runner copies the exported STL geometry that includes the blade and overall domain, to a proper OpenFOAM folder structure along with the necessary dictionaries. One final step is left. Somehow, for every geometry, the OpenFOAM utilities and solver need to be executed. The Runner is able to execute local applications in every folder generated so we need to contain all the essential OpenFOAM utilities into a single executable. This is a common situation in OpenFOAM applications. The workaround is to include all the terminal commands that execute the utilities in an Allrun script. Executing this script in the terminal makes sure that the utilities will be ran, one by one. The Allrun script is shown and analyzed below.

Code 2. Allrun script

```
nProcs=1
touch case.foam
cartesianMesh
scaleMesh 0.001
topoSet
setsToZones
createPatch -overwrite
checkMesh -allTopology -allGeometry
setsToZones -noFlipMap
simpleFoam
gnuplot Residuals
gnuplot Forces
simpleFoam -postProcess -func yPlus
foamToVTK
```

s

- **nProcs=1:** This command declares that only one processor will be used for the simulation. During this thesis, some computations ran in parallel and some serial. OpenFOAM can be configured easily to run in parallel significantly reducing computational time. These configurations will be discussed later on.

- **touch case.foam:** This command creates a dummy file called case.foam. The file is actually empty, it is needed to open the simulation results in the post-processor Paraview.
- **cartesianMesh:** It triggers the grid generator cfMesh that creates the computational grid according to the settings specified in the **meshDict** dictionary.
- **scaleMesh:** This command scales the entire mesh.
- **topoSet:** This utility was briefly discussed in the CFD setup section. It operates according to the settings set in the **topoSetDict**. In our case, two operations are specified in the **topoSetDict**. **CylinderToCell** and **setToCellZone**, the first one selects all cells with cell center within a bounding cylinder whose dimensions are selected to include the whole computational grid and puts them into a **cellSet**. The second operation transfers the created **cellSet** to a **cellZone**, because the **MRF** can be applied in a **cellZone** but not a **cellSet**. There is no significant difference between a **cellZone** and **cellSet**. Both can be used to isolate groups of cells, to enable the use of utilities on them.
- **createPatch -overwrite:** This utility handles the periodic sides. New patches are constructed from the left and right periodic sides. They are coupled as neighbors and the **cyclicAMI** boundary conditions is applied. The tolerance of the matching can be increased if the simulations diverge due to mismatch.
- **checkMesh -allTopology -allGeometry:** As discussed in the Grid Generation section, this utility is of great importance. It provides a detailed quality report but also creates **cellSets** for all the low-quality cells. CheckMesh actually detects the cells whose quality metrics exceed the threshold values, examples are skewed cells, concave cells and so on. The quality metrics will be analyzed in detail in the Validation section.
- **setsToZones -noFlipMap:** Creates a **cellZone** out of all existing **cellSets**. It is required in order to visualize the low-quality cells that **checkMesh** detected in Paraview and perform a visual quality check.
- **simpleFOAM:** This command triggers the solver. The solver operates according to the settings set in the dictionaries **fvSchemes**, **fvSolution** and **controlDict**. The turbulence model is specified in the **turbulenceProperties** dictionary located in the **constant** folder. The solver can be configured to stop after a selected timestep or when the residuals fall under a specified value. In the **controlDict** the user specifies with the **writeInterval** option at what timesteps to save the calculated flow data. Only the last time step is selected to save computer memory. The solver generates a large amount of data and not all of it is needed.

- **Gnuplot:** The gnuplot graphing utility is used to retrieve graphs for the residuals and forces. Pictures of the graphs are generated so they can be monitored afterwards for every design. In this way, after a number of designs are explored, the user can navigate through the designs in CAESES and check the convergence curves and residuals.
- **simpleFOAM -postProcess -func yPlus:** This command calculates the y^+ values on every patch (shaft, blade and so on). It can only be triggered at a post processing stage when the flow field has been calculated.
- **foamToVTK:** This post-process utility converts the flow data from the OpenFOAM format to VTK format. In this way, any VTK-based graphics tool can be used to process the case.

4.7 Running the case in parallel

With minimal effort, the setup can be configured to run in parallel. The method of parallel computing that OpenFOAM uses is known as domain decomposition. [46] With this method, the geometry and associated fields are split into pieces and allocated to separate processors for solution. The process involves: decomposition of the mesh and fields, running the application in parallel, and finally, post-processing the decomposed case. The parallel running uses the public domain **openMPI** implementation. [31]

The modifications are based on the existing described Allrun script shown in Code 2. At first, the **nProcs** is changed to specify the number of processors available for use. Using all the processors of the computer may cause problems to CAESES, since it also needs some computational power to handle the whole process. Then, before simpleFOAM is executed, the computational grid needs to be split up. The **decomposePar** utility is used to handle this operation, with settings specified in the **decomposeParDict** located in the **constant** folder. Three options are set. The first one is the **numberOfSubdomains**, that is set to match the **nProcs**. The second one is the method of decomposition. There are four available options [31], in our case, the **scotch** method is used. Lastly, the neighbor patches specified in the **createPatch** utility need to be kept in the same processor, this can be done with the **preservePatches** constraint.

Now, the solver can run in parallel with the command **mpirun simpleFOAM -parallel**. The mesh and field data need to be reconstructed after the simulation has finished in order to post-process the case. This can be done with the commands **reconstructParMesh -constant -mergeTol 1e-6** and **reconstructPar -latertTime**. With the mesh reconstructed the y^+ utility or any other post-processing utility can be triggered.

5 Validation and Results

5.1.1 Mesh quality metrics

With automatic grid generators high quality meshes are difficult to accomplish. On the other hand, mesh quality has a great impact on CFD results. In the Grid Generation section, the efforts made and methods used in this thesis have been described thoroughly. This section deals with the quality of the generated mesh in terms of mesh quality metrics and criteria. In OpenFOAM, the mesh quality is assessed with the utility **checkMesh**. This utility provides a detailed report that includes the violation of some predefined quality criteria. These reports are created for every design generated in the optimization phase and some meshes violate these predefined criteria. Some of these violations are more critical than others and they need to be discussed. The problematic cells are mainly related to the sharp edges of the blade and most of them could be resolved by applying refinements. Some errors however still persist. The main parameters that control the mesh quality in OpenFOAM are the **aspect ratio**, the **non-orthogonality**, and the **skewness** of the cells.

The **aspect ratio** for triangular and tetrahedral cells, is described as the ratio between the maximum cell length l to the minimum height of the cell, h , as seen in equation 5.1 below:

$$\text{Aspect Ratio} = \frac{\max(l_i)}{\max(h_i)} \quad (5.1)$$

For quadrilateral and hexahedral cells, the idea is the same, however the formulation changes in order to take into account all the edges of the cells in every coordinate direction:

$$\text{Aspect Ratio} = \frac{\max(e_1, e_1, e_1, \dots, e_n)}{\min(e_1, e_1, e_1, \dots, e_n)} \quad (5.2)$$

As depicted in Figure 5.1, e_i is the average edges length in the coordinate direction i , while n represents the all possible coordinate directions (2 for 2D elements such as quadrilaterals, 3 for 3D like hexahedral cells).

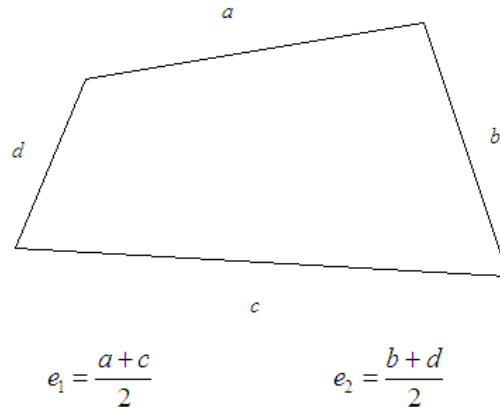


Figure 5.1 Aspect Ratio for a quadrilateral element [47]

Non-orthogonality is defined as the angle θ formed by the face vector \mathbf{S}_f and the vector \mathbf{d}_{PN} , that connects the cell centers \mathbf{P} and \mathbf{N} of two neighboring cells with the same face, and face center \mathbf{f} . The angle θ can be seen in Figure 5.2 below. According to [8], non-orthogonality affects the accuracy of the discretization of the diffusive terms related to the transport equations and it can deteriorate the accuracy of the solution, increasing the computational time if its value is higher than 70° . However, for a mesh to be considered invalid the non-orthogonality needs to exceed the 90° mark.

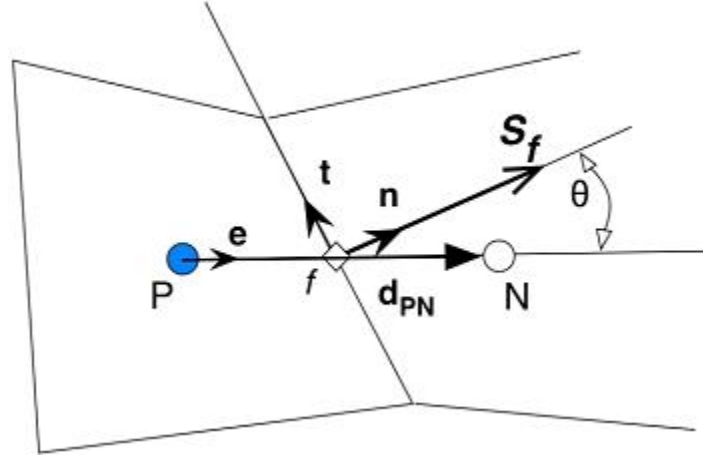


Figure 5.2 Definition of non-orthogonality [48]

Skewness occurs when there is a deviation between the location of the face center, \mathbf{f} , and where the center-to-center vector meets the face, \mathbf{f}' , as seen in Figure 5.3. Skewness, ε can be calculated from the expression 5.3 below.

$$\varepsilon = \frac{|\mathbf{f} - \mathbf{f}'|}{|\mathbf{d}_{PN}|} \quad (5.3)$$

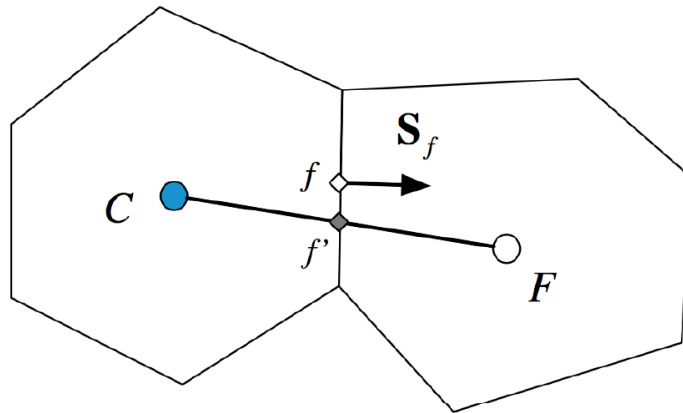


Figure 5.3 Skewness between neighbor cells [49]

Part of the checkMesh output for the baseline design (PPTC) can be seen in the Code 3 below. The aspect ratio should be as close to 1 as possible and the standard threshold in OpenFOAM is set to 1000. In our case, the maximum aspect ratio is 46.94. As stated above, non-orthogonality should be less than 90° . Industry reasonable numbers are typically 80° - 85° and values less than 70° are rather unlikely. The predefined threshold in OpenFOAM is set to 70° . In this case the maximum non-orthogonality encountered is 84.18 with 92 faces exceeding the threshold. The maximum skewness detected is 5.87 with 136 faces exceeding the limit that is set to 4. The aspect ratio and non-orthogonality ratings are quite satisfactory. Lower levels of skewness would be desired but they could not be achieved. The skewed cells are located at near the trailing edge of the blade.

The total number of cells of each type is shown in Table 5.1 below. The mesh is dominated by hexahedral cells with prism cells at the propeller blade. The difficult regions are filled with pyramids, tetrahedra and polyhedra.

Code 3. CheckMesh output for baseline design

```
Overall domain bounding box (-0.171135 -0.499958 -0.375) (0.499994 0.397107 1.25)
Mesh has 3 geometric (non-empty/wedge) directions (1 1 1)
Mesh has 3 solution (non-empty) directions (1 1 1)
Boundary openness (-3.79792e-15 -1.32471e-15 1.39821e-16) OK.
Max cell openness = 1.14047e-15 OK.
Max aspect ratio = 46.9349 OK.
Minimum face area = 4.43744e-11. Maximum face area = 0.000913717. Face area magnitudes OK.
Min volume = 2.91823e-16. Max volume = 1.97263e-05. Total volume = 0.254678. Cell volumes
OK.
Mesh non-orthogonality Max: 84.1793 average: 8.0599
*Number of severely non-orthogonal (> 70 degrees) faces: 92.
Non-orthogonality check OK.
<<Writing 92 non-orthogonal faces to set nonOrthoFaces
Face pyramids OK.
***Max skewness = 5.87447, 136 highly skew faces detected which may impair the quality of the
results
<<Writing 136 skew faces to set skewFaces
```

Table 5.1 Number of cells of each type

hexahedra	1621428
prisms	36634
pyramids	17709
tetrahedra	23877
polyhedra	90901

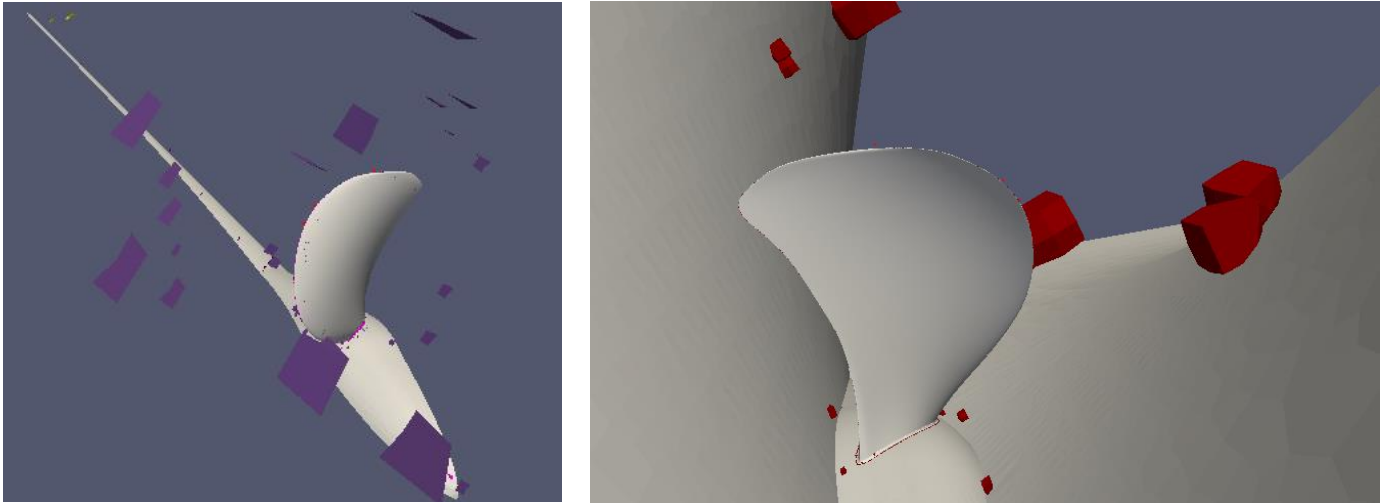


Figure 5.4 Concave faces (left), Concave cells (right)

The three basic quality measures described above, are evaluated using the utility **checkMesh**. Including the **-allTopology -allGeometry** gives a more detailed report with more groups of cells of low quality. No official guidelines are found to evaluate the importance and influence of these groups of cells. At the 7th OpenFOAM Workshop, Engys [50] presented a guide that deals with these groups ranking them according to their importance. This guide, along with the efforts made during the course of the thesis sum up the following.

Low Volume Ratio Faces. These faces can cause problems during simulations. A great number of these faces were generated close to the edges. However, most of these faces were eliminated after applying the refinements that control the cell size at a distance from the wall.

Concave Cells/Faces. These cells and faces are not as bad as the Low volume ratio faces. A great number of these cells are contained in the mesh. They can be seen in Figure 5.4. It was observed that a great number of these cells and faces was generated even without the propeller included in the domain. Most of these cells are located in the transition areas where there is a sudden reduction in cell size. Their existence did not influence the convergence of the simulations.

Low Quality Tet Faces. Similar to the concave, their severity is ranked low. It was observed that these cells occur when trying to generate prism layers. They are formed along the edges of the blade, the leading and trailing edge, fillet and tip.

In the end, the meshes used for the validation and optimization study are not perfect. Even though a significant amount of time has been spent on fine tuning the settings to improve the mesh quality, the thorough check still detects cells of low quality. However, the basic quality criteria are kept under acceptable levels and the simulations converge to accurate results.

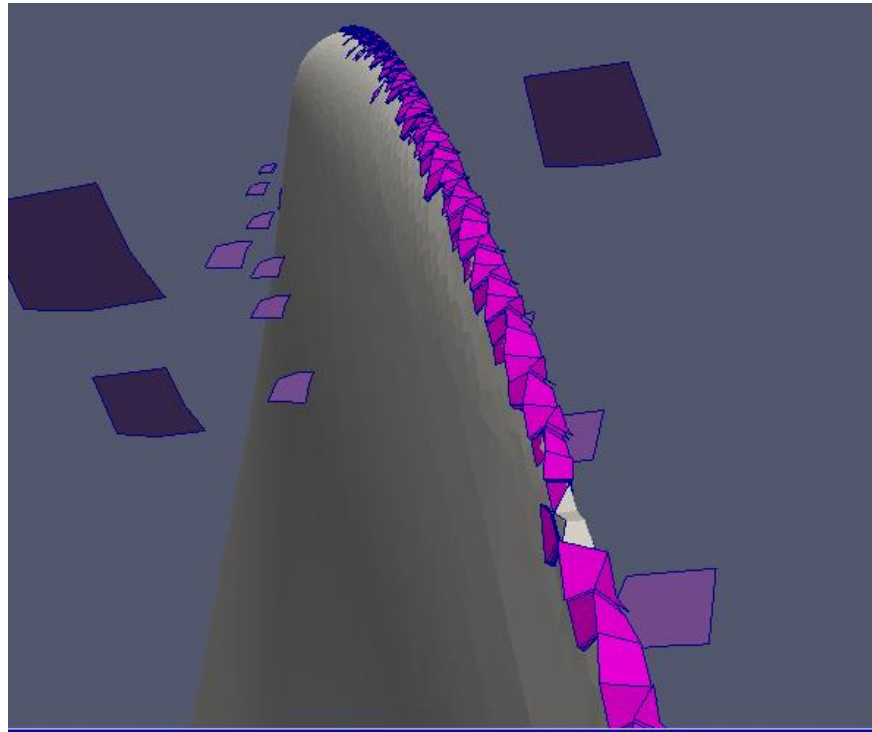


Figure 5.5 Low quality faces at the tip

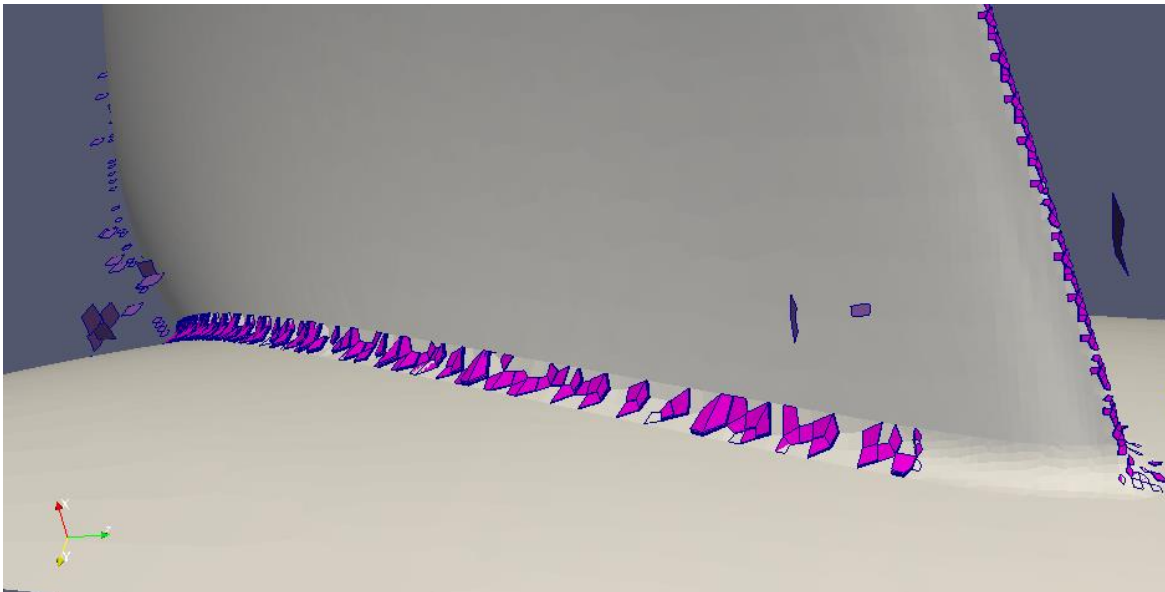


Figure 5.6 Low quality faces at the fillet

5.1.2. Grid Independence Study

In CFD simulations, we need to validate that the solution is independent of the mesh resolution. For this reason, a grid independence study is conducted and the simulations are performed for a sequence of three mesh resolutions, coarse, medium and fine. It is noted that the results presented here are referring to the baseline design of the parametric model, that features the curved trailing edge, for advance coefficient $J=1.2$ that is also used for the optimization.

The first simulation is performed with a coarse mesh of 1049482 cells. This initial simulation helps to make sure that the boundary conditions are correct and to get a rough estimation of the calculated forces and the y^+ values on the blade. For the medium mesh, smaller cell size is selected on the surface patches, on the refinement thickness and the cylinder refinement and the cell count increases to 1806552. A difference of 1.7% is observed for $10K_Q$ and about 2% for K_T . For the finest grid used, the cell size selected is even smaller leading to 2994760 cells. A deviation of only 0.6% is observed and mesh convergence is achieved. For all the meshes, five prism layers are generated on the propeller blade. For the rest of the work, the medium mesh resolution is used. The y^+ values corresponding to the medium mesh are presented in Figure 5.8, they are kept below 2 for the main area of the blade.

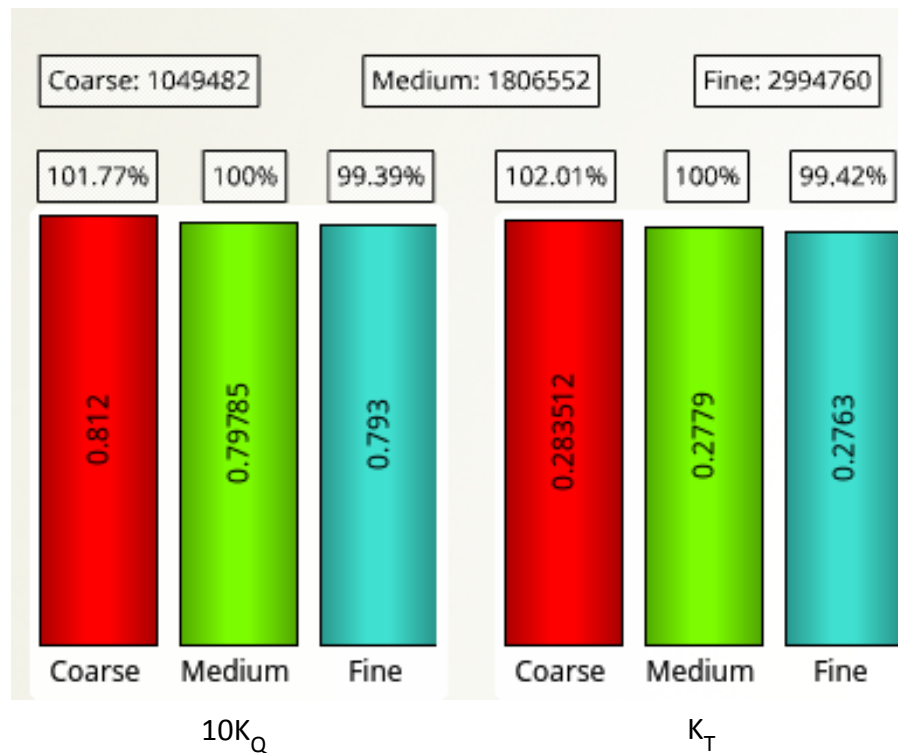


Figure 5.7 Grid independence study

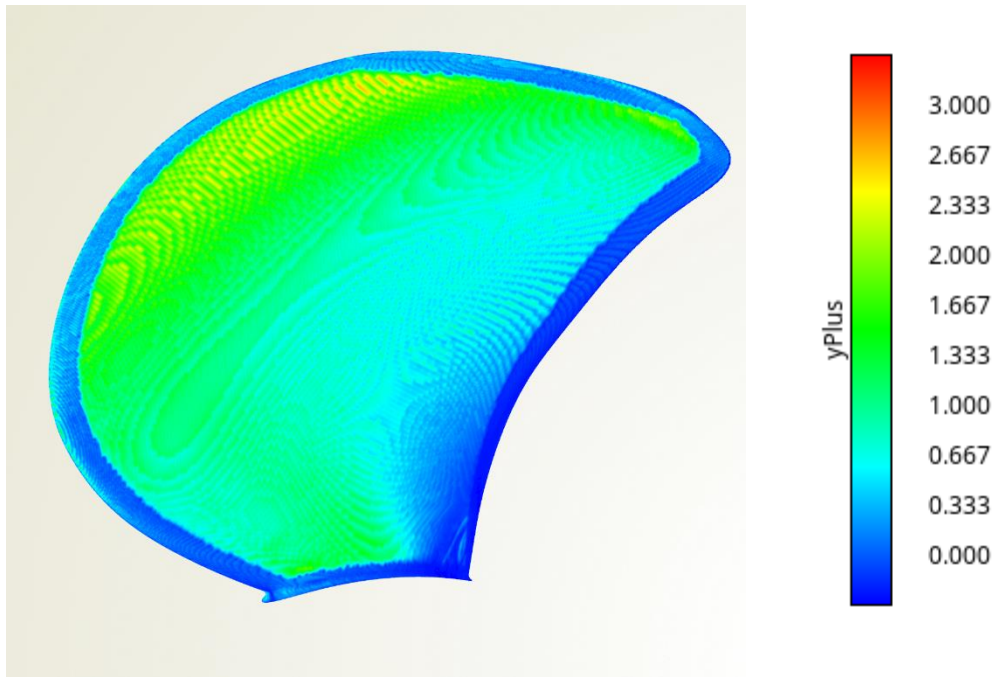


Figure 5.8 Y^+ values on the propeller blade corresponding to the medium mesh

One of the difficulties faced in this thesis is the limited computational power that is restricted to a notebook with an i5 processor of 2.5GHz. Convergence is achieved in about 1.5, 4 and 8 hours for the coarse, medium and fine mesh respectively. That being said, exploring 30 designs requires 120 hours of computational time.

5.1.4 Convergence

The convergence of the simulations is monitored closely. The forces, torque and residuals values for the pressure, velocity and turbulent quantities are written in a log file for each iteration. Gnuplot commands are included in the Allrun script that is executed for every design generated. The measured forces and torque are measured for one blade and they are transformed into the thrust and torque coefficient. The convergence curves are presented from figures 5.9 to figure 5.13. The residuals are stable, K_T and $10K_Q$ have changed very little in the last 500 iterations (approximately 0.01%).

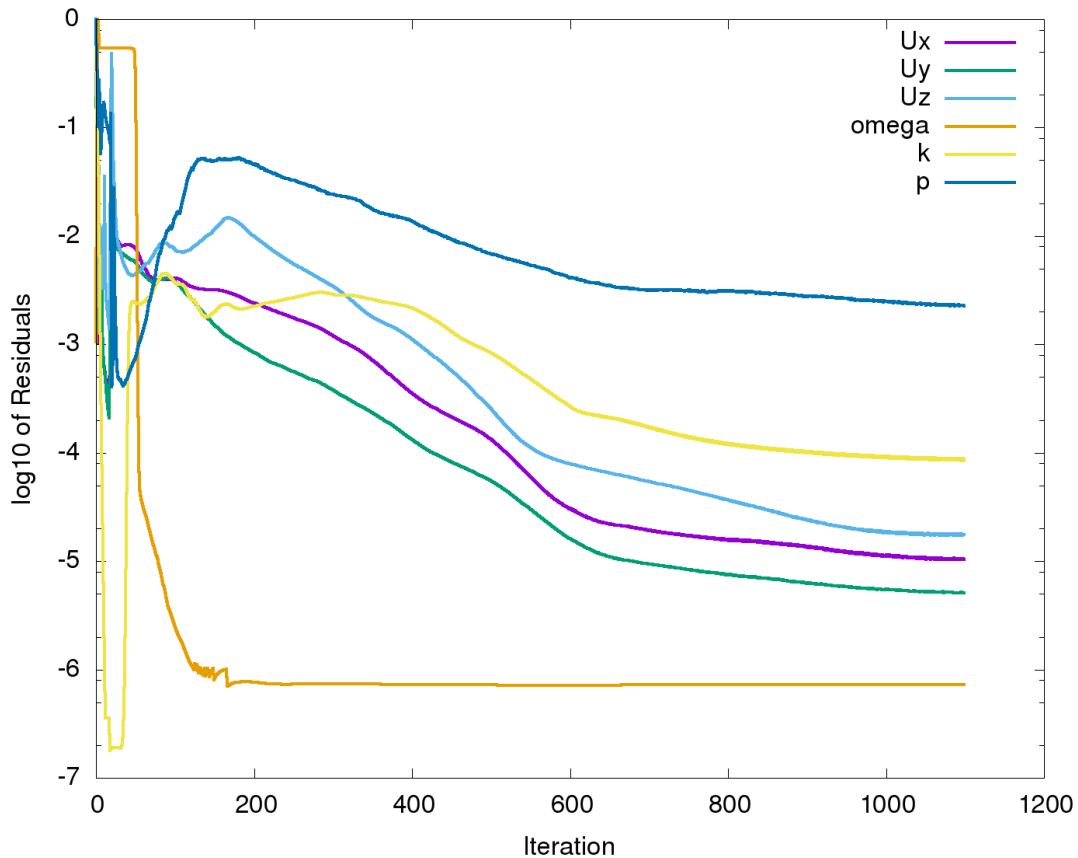


Figure 5.9 Residuals of the simulation for $J=1.2$

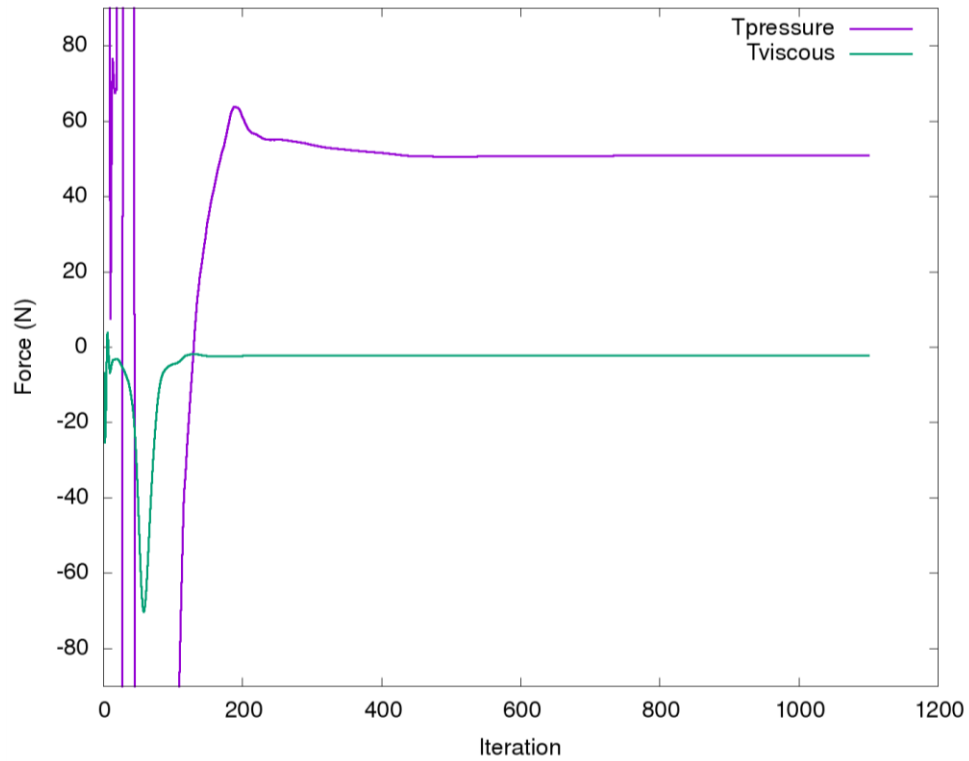


Figure 5.10 Calculated forces for J=1.2

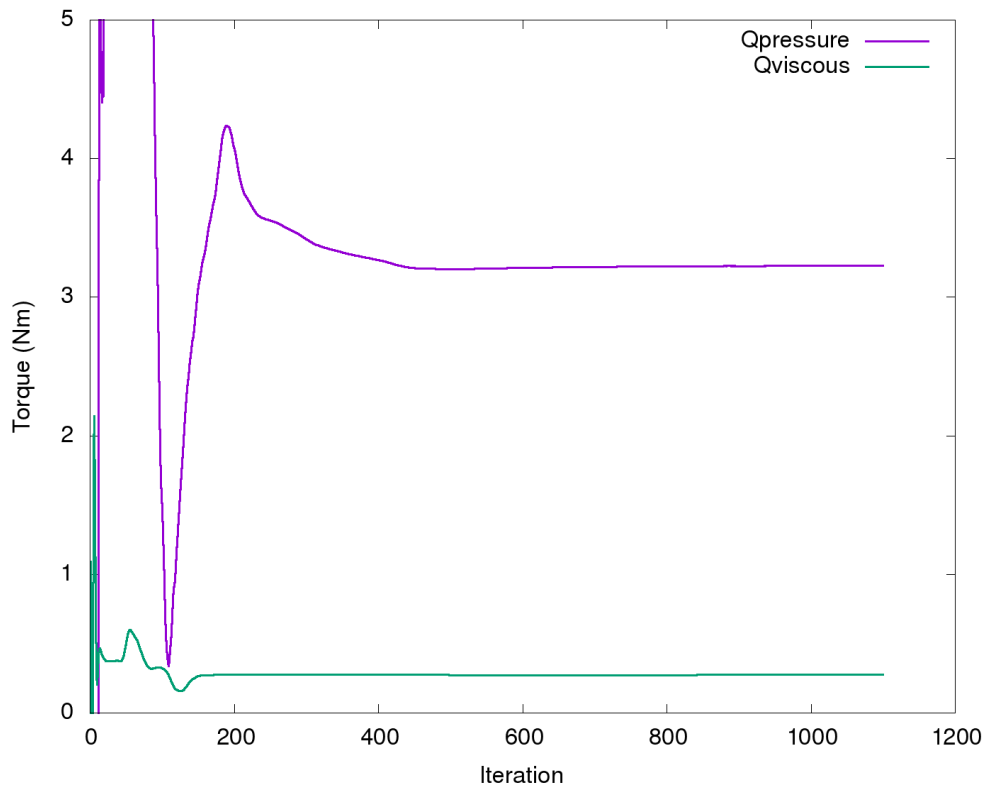


Figure 5.11 Calculated moments for J=1.2

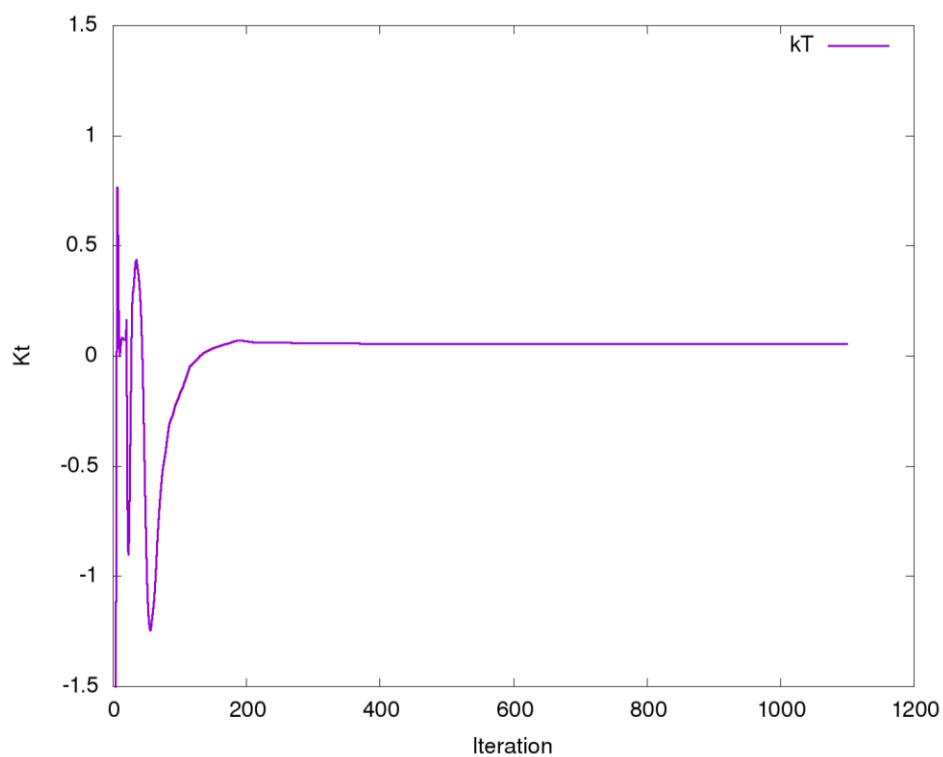


Figure 5.12 Thrust coefficient for $J=1.2$

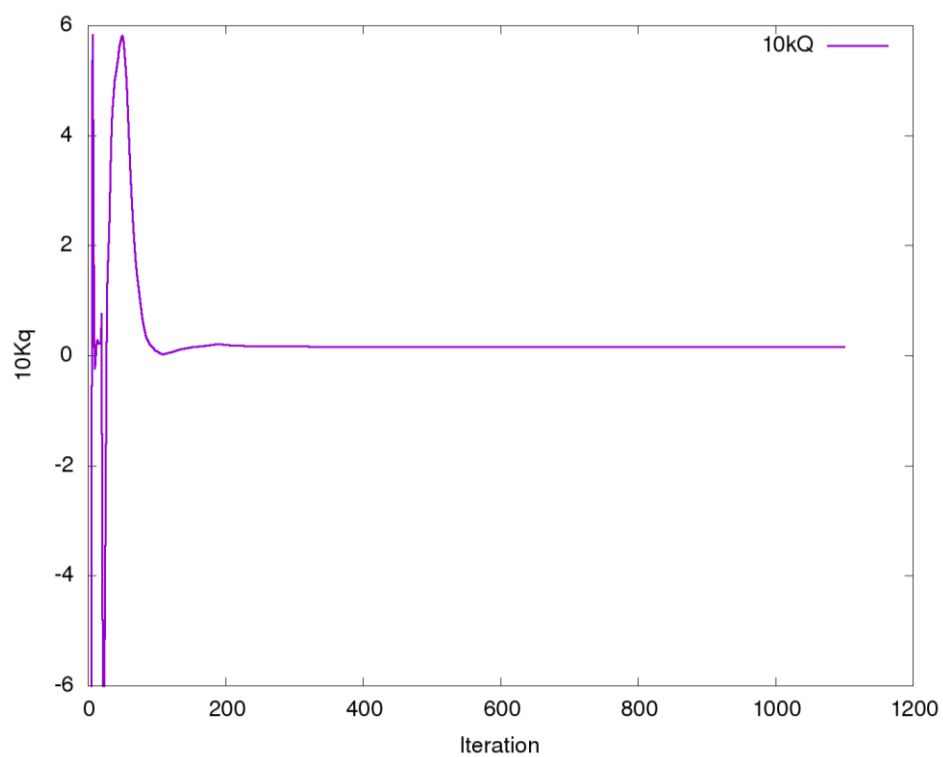


Figure 5.13 Torque coefficient for $J=1.2$

5.1.5 Comparison with experimental data

The simulations are performed for a wide range of advance coefficients from $J=0.2$ to $J=1.4$ and the calculated open water characteristics are compared with the experimental data presented in the open water chart presented in the Figure 5.14 below. The results from the viscous openFOAM analysis that are presented below correspond to the mesh classified as medium. The experimental data are found in the report provided online by the SVA [34]. Specifically, “Potsdam Propeller Test Case (PPTC) – Open Water Tests with the Model Propeller VP1304 – Report 3752”. In order to compare the results at the same advance coefficients, the open water characteristics are extracted from the polynomials provided in the report 3752 page 2.11 as seen below.

Table 5.2 Coefficients of polynomials

p	a₀	a₁	a₂	a₃	a₄
K_T	0.955438	-0.346932	-0.629537	0.586304	-0.175174
10 K_Q	2.076022	-0.949651	-0.719299	0.873861	-0.306054

Valid in the area $0 \leq J \leq 1.677$, $p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$

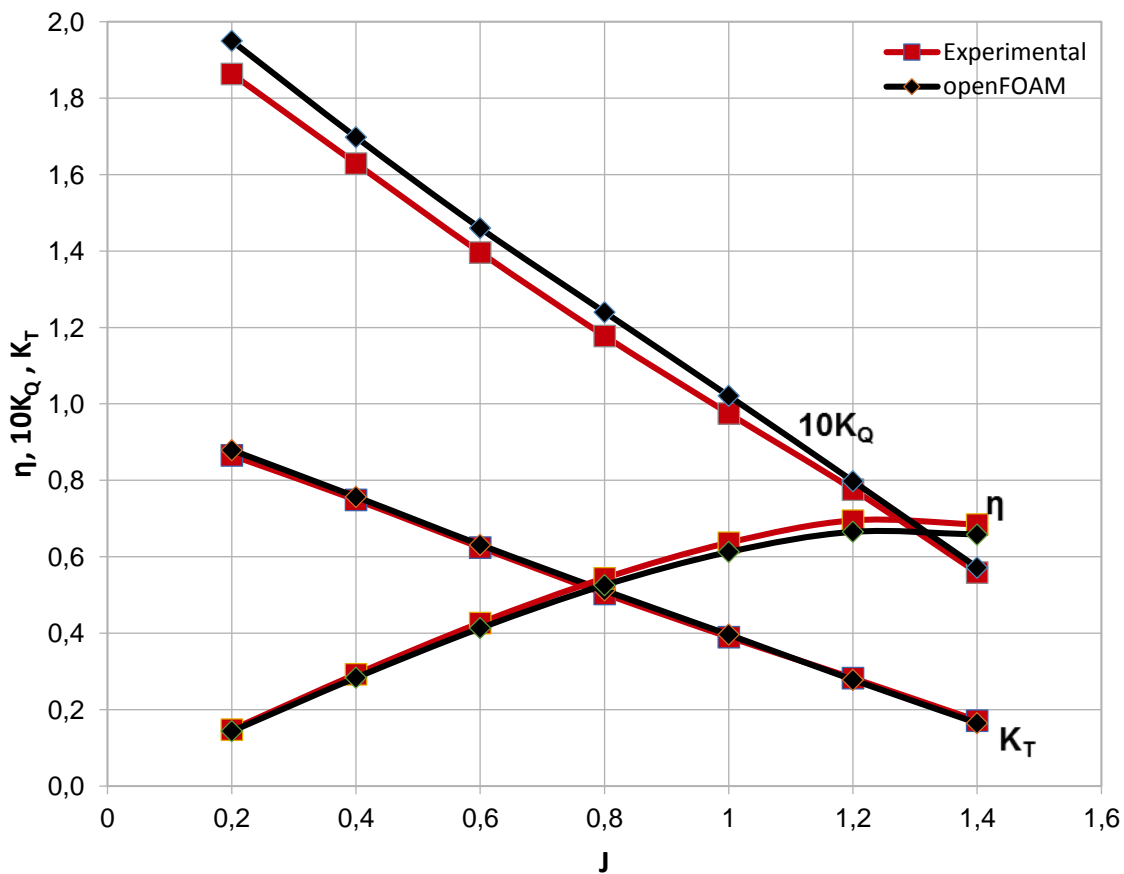


Figure 5.14 Comparison with experimental data

Table 5.2 Comparison of with experimental data

J	K_T(CFD)	K_T(EFD)	ΔK_T[%]
0.2	0.8792	0.8653	1.5810
0.4	0.7516	0.7490	0.3459
0.6	0.6315	0.6246	1.0926
0.8	0.5119	0.5034	1.6605
1	0.3964	0.3901	1.5893
1.2	0.2779	0.2825	-1.6553
1.4	0.1690	0.1717	-1.5976
J	K_Q(CFD)	K_Q(EFD)	ΔK_Q[%]
0.2	1.9500	1.8638	4.4205
0.4	1.6873	1.6292	3.4434
0.6	1.4597	1.3964	4.3365
0.8	1.2402	1.1780	5.0153
1	1.0300	0.9749	5.3495
1.2	0.7979	0.7760	2.7447
1.4	0.5720	0.5588	2.3077
J	η₀(CFD)	η₀(EFD)	Δη₀[%]
0.2	0.1435	0.1478	-2.9965
0.4	0.2836	0.2927	-3.2087
0.6	0.4131	0.4271	-3.3890
0.8	0.5256	0.5441	-3.5198
1	0.6125	0.6369	-3.9837
1.2	0.6652	0.6952	-4.5099
1.4	0.6583	0.6847	-4.0103

The obtained results are quite satisfactory especially when compared to simulation results from various research groups from the academia (University of Genua, TUHH) and the industry (Berg, HSVA, VOITH) presented at the Second International Symposium on Marine Propulsors (smp'11) that took place in 2011. [34] In the published results, deviations up to 10% and 20% corresponding to advance coefficients $J=1.2$ and $J=1.4$ are observed, while most of the research groups used commercial solvers and grid generators.

In the obtained results, a maximum percentage difference of 1.66% is observed for K_T , a 5.35% for K_Q and a maximum of 4.5% difference on the open water efficiency. The prediction of K_T is quite accurate, however the CFD calculations seem to overpredict the torque coefficient. This overprediction of K_Q is also observed in the results presented at the smp'11. The results are also in agreement with Da-Qing's [51] work, where a 3% and 5% error difference is observed for K_T and K_Q respectively. The deviation observed on K_Q could be caused by the weakness of the boundary layer mesh as discussed in the grid generation section.

In the Figures below, the pressure distributions on the suction and pressure side of the blade are presented for advance coefficients from $J=0.2$ to $J=1.4$.

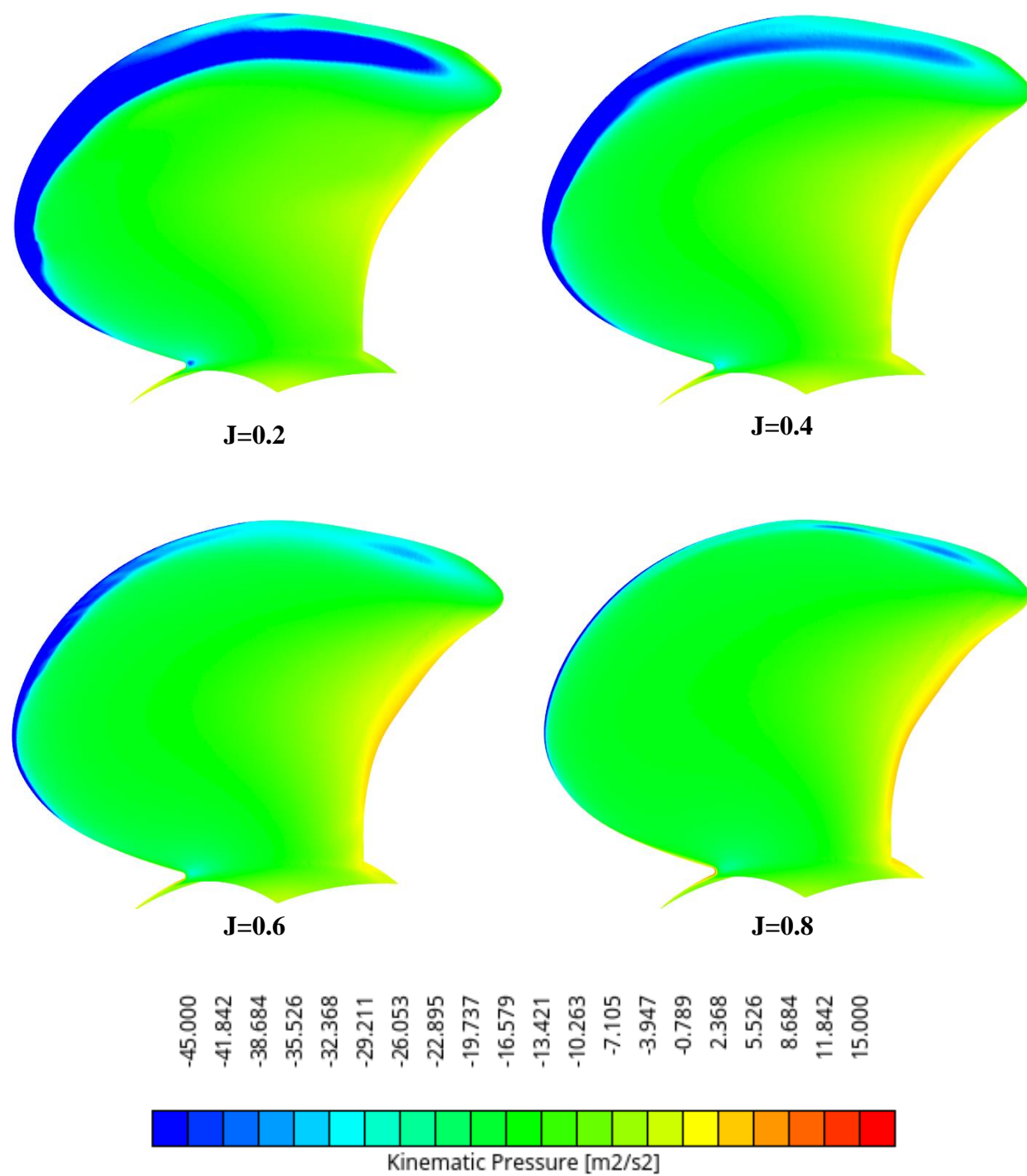


Figure 5.15 Pressure distribution on the suction side for advance coefficients from J=0.2 to J=0.8

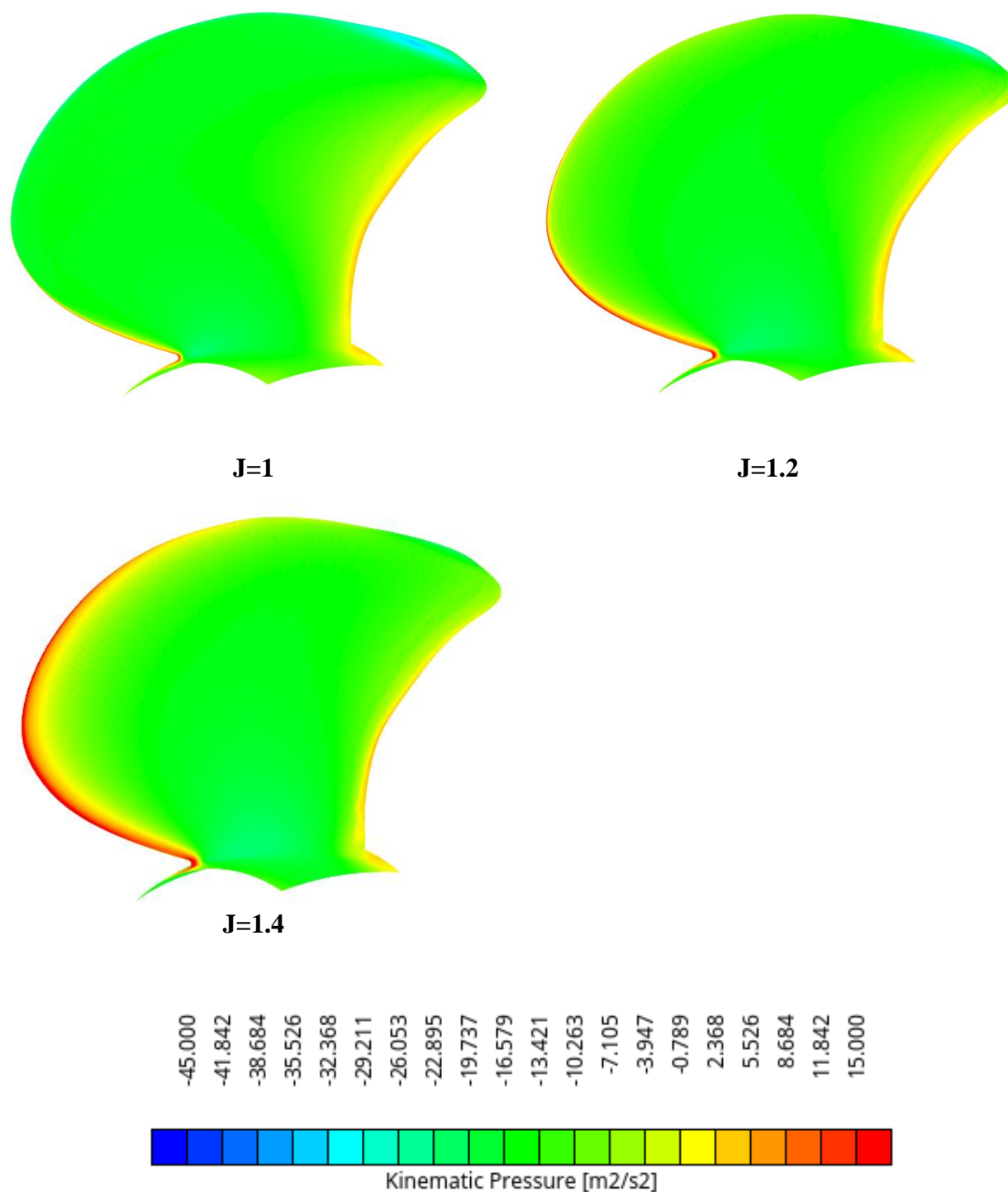


Figure 5.16 Pressure distribution on the suction side for advance coefficients from $J=1$ to $J=1.4$

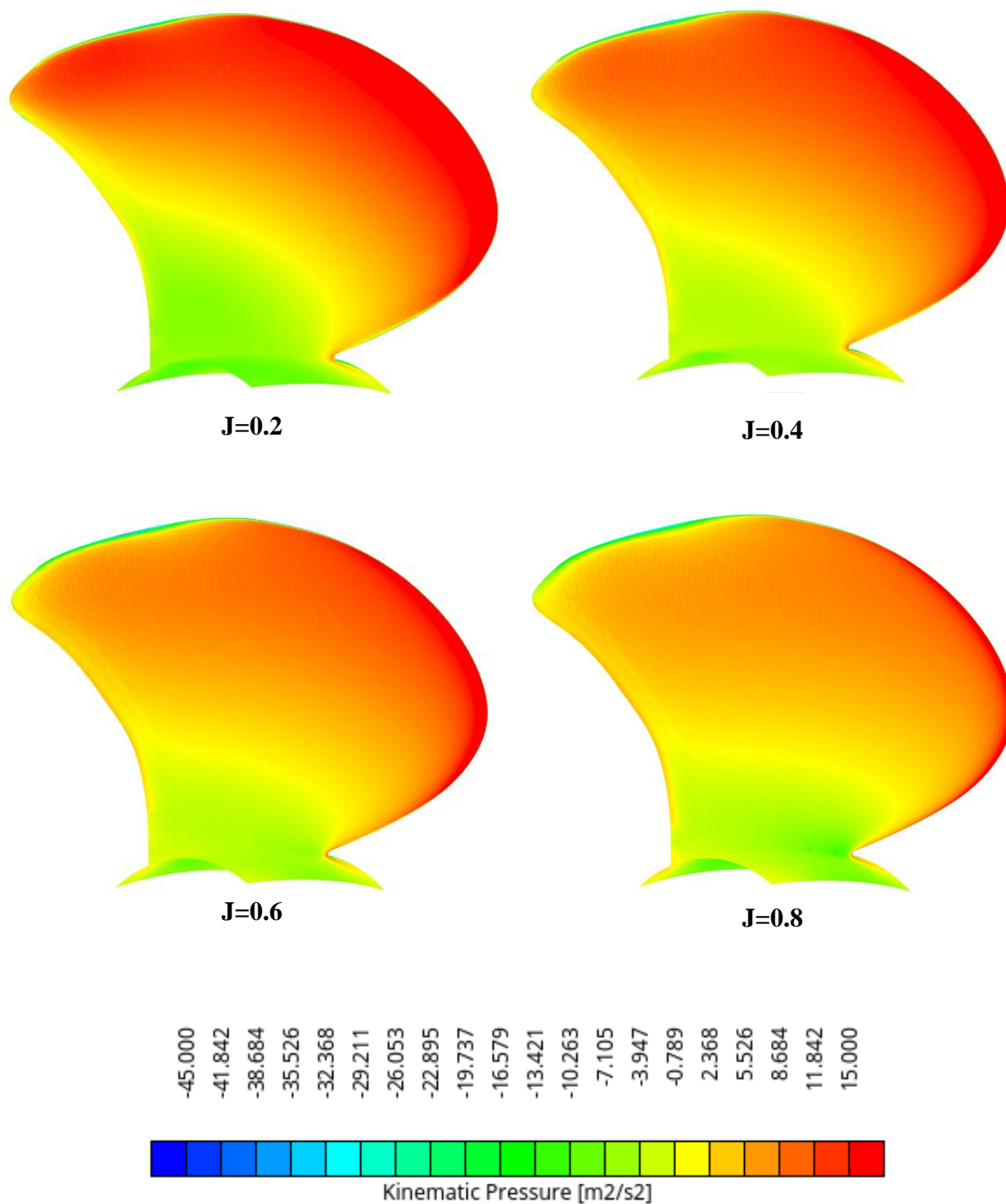


Figure 5.17 Pressure distribution on the pressure side for advance coefficients from J=0.2 to J=0.8

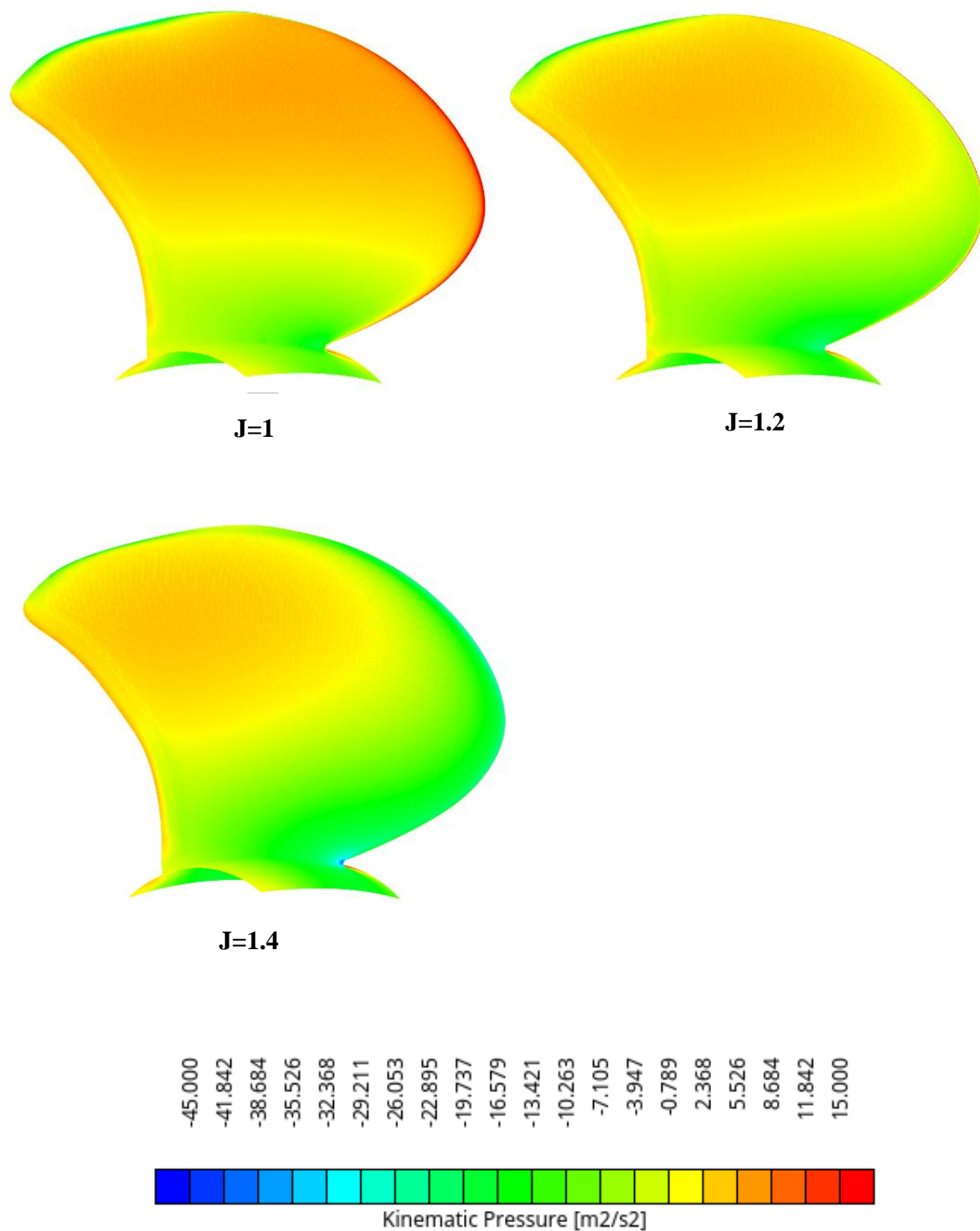


Figure 5.18 Pressure distribution on the pressure side for advance coefficients from J=1 to J=1.4

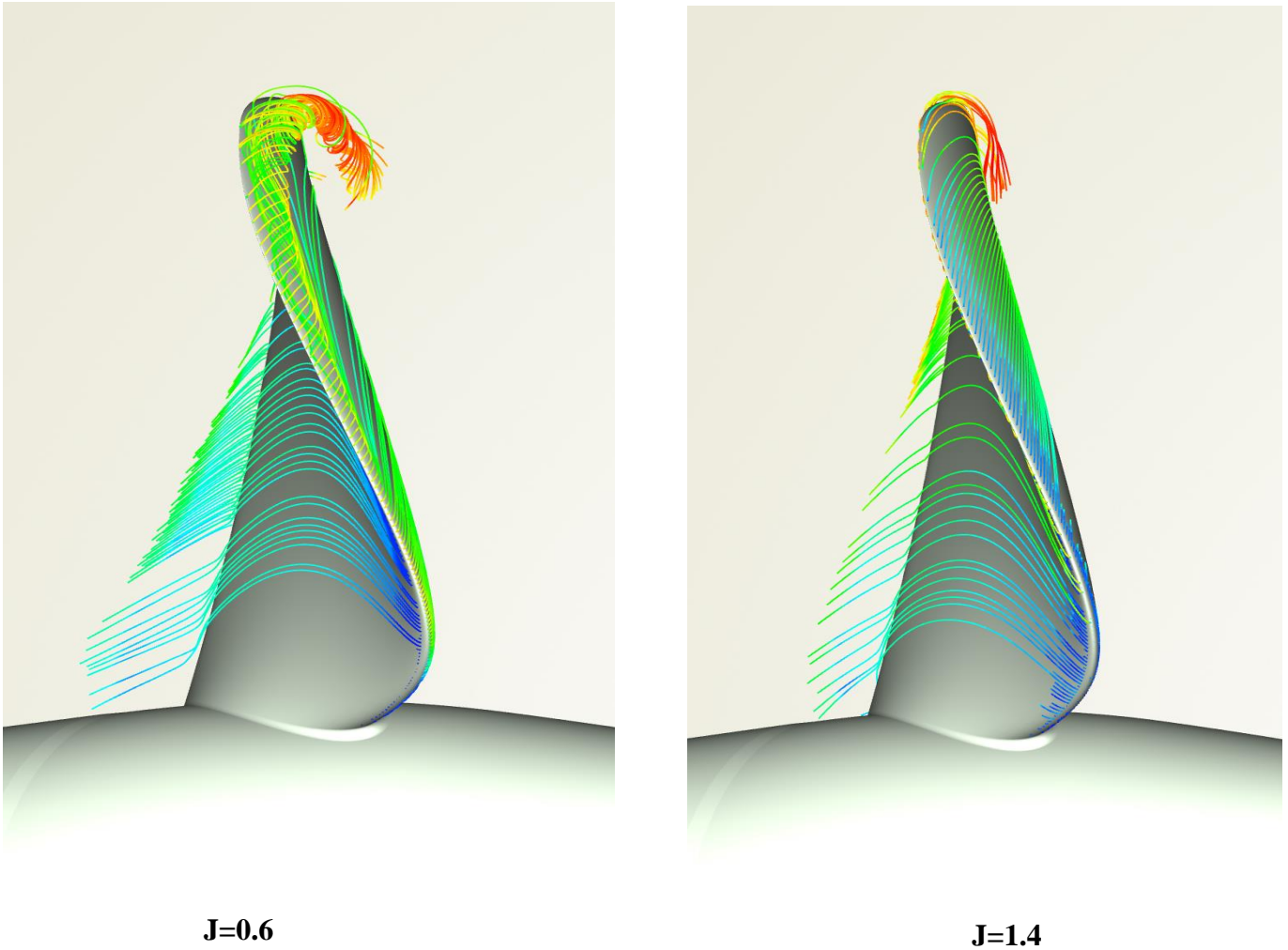


Figure 5.19 Streamline's behavior on the leading edge

High overall pressure is observed on the pressure side and lower pressure on the suction side. As expected, the loading is larger at lower J values indicated by the red areas on the pressure side of the blade for $J=0.2$ to $J=0.8$. On the suction side, flow separation is observed along the leading edge that is more intense for lower J values.

Observing the pressure distributions one can also notice how the advance coefficient determines the location of the stagnation point. For low advance coefficients, the stagnation point is located on the pressure side and moves towards the suction side as the J increases. The same behavior is depicted in Figure 5.19 where the streamlines passing through the leading edge are visualized. For $J=0.6$, the flow hits on the pressure side of the blade decelerating, and the streamlines are colored with blue. At the rest of the pressure side, the flow follows the blade accelerating gradually. The streamlines find their way around the leading edge when they separate from the blade and accelerate, forming a vortex. On the right side, for $J=1.4$, the stagnation point moves towards the suction side and the streamlines follow the blade smoothly.

The Potsdam propeller is designed to generate a tip vortex [34] and visualizing the vortex is important for the validation of the computations. One way to visualize the vortical structures around the blade is the Q criterion [52]. The Q criterion defines a vortex as a spatial region where

$$Q = \frac{1}{2} [|\boldsymbol{\Omega}|^2 - |\boldsymbol{S}|^2] > 0,$$

that is, when the Euclidean norm of the vorticity tensor dominates over the Euclidean norm of the rate of strain tensor. Where the vorticity tensor is defined as $\boldsymbol{\Omega} = \frac{1}{2} [\nabla \boldsymbol{v} - (\nabla \boldsymbol{v})^T]$, and the rate-of-strain tensor as $\boldsymbol{S} = \frac{1}{2} [\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T]$. For a greater value of Q, stronger vortices will appear. In OpenFOAM, there is an implemented function object that calculates Q at a post-processing stage. Iso-surfaces can then be generated in Paraview. Iso-surfaces of $Q=20000$ are illustrated in Figure 5.20 below and colored with the velocity magnitude. Fine grids are needed to capture the vortices. They stop where the refinements end downstream as seen in the Figure 5.21 where a plane cut of the 3D mesh is depicted as well. The axial velocity distribution is depicted in Figure 5.22 below.

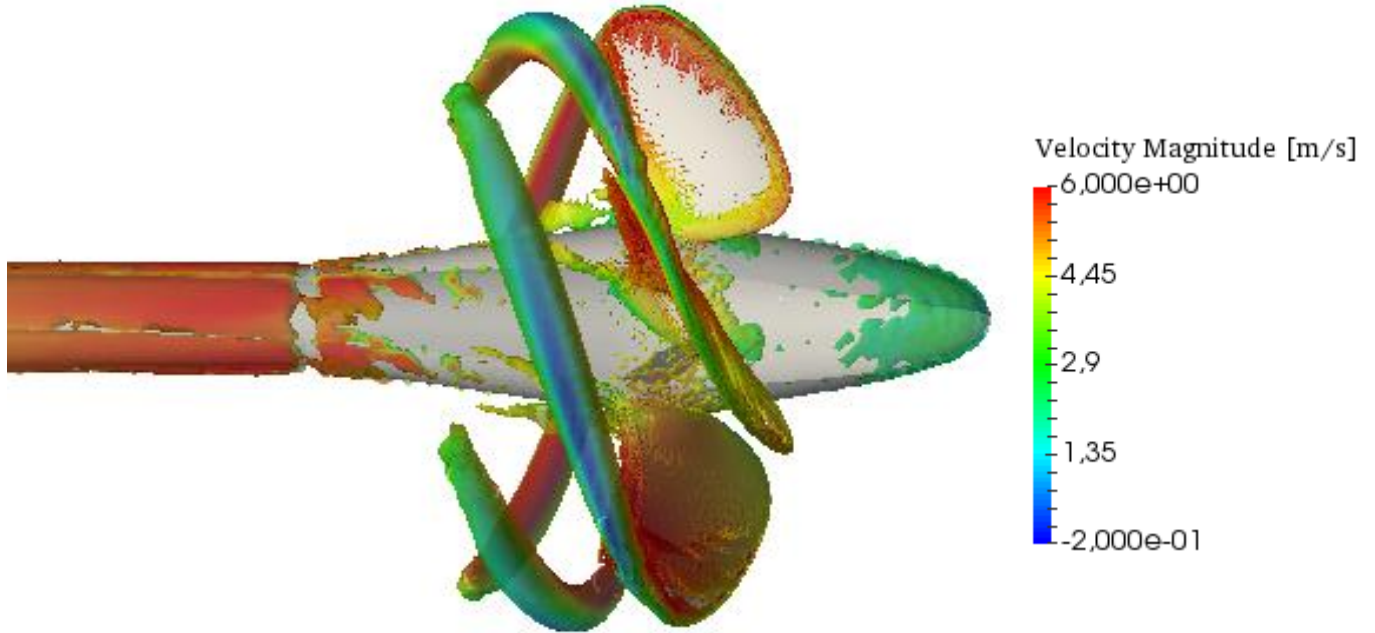


Figure 5.20 Iso-surfaces of $Q = 20000$ colored with the velocity magnitude for $J=0.6$

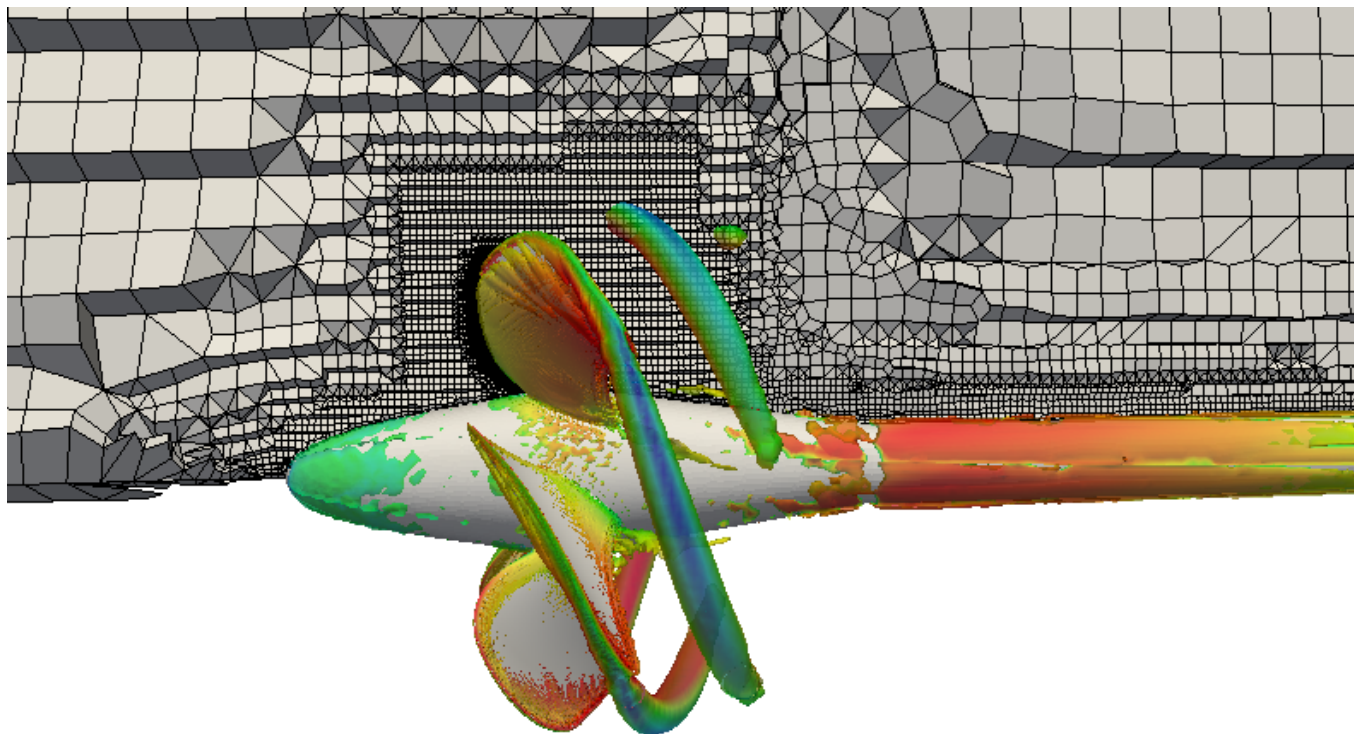


Figure 5.21 Mesh plane cut

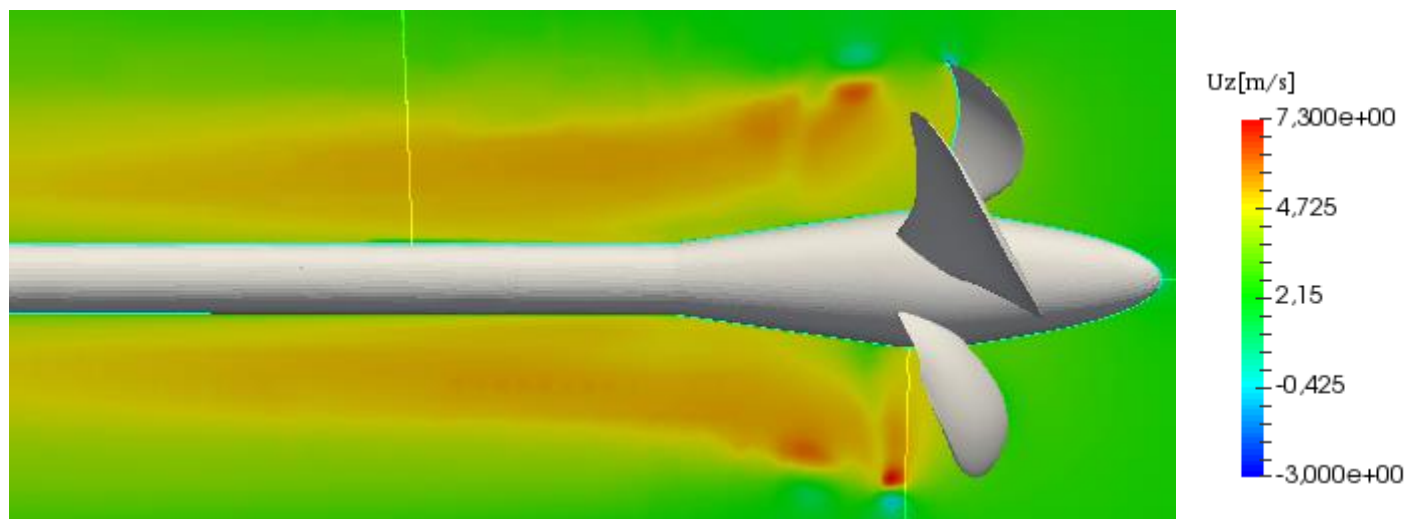


Figure 5.22 Axial velocity distribution for $J=0.6$

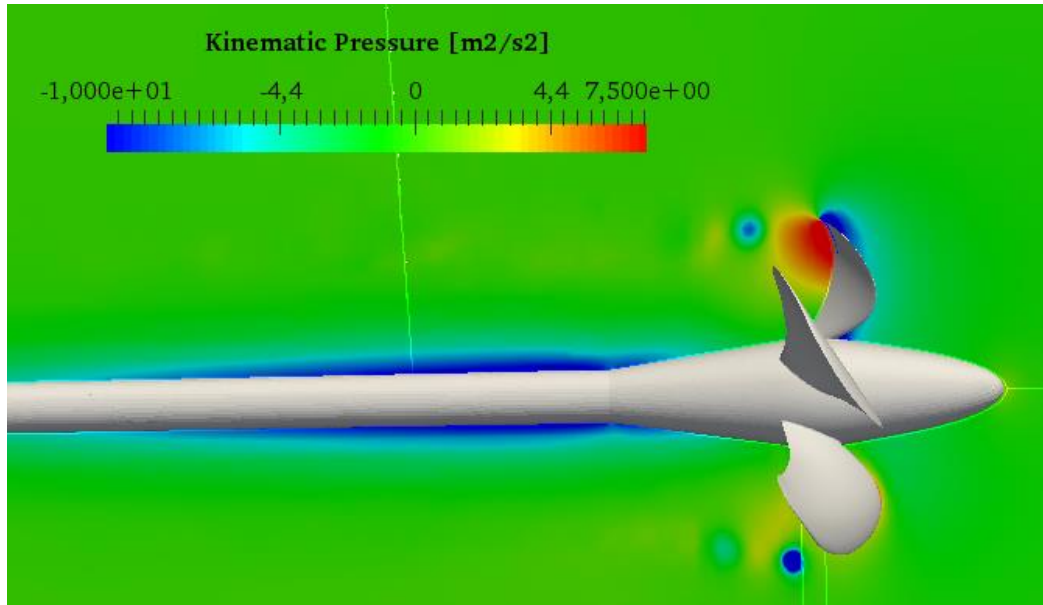


Figure 5.23 Pressure field around the blade on $y=\text{constant}$ for $J=0.6$

In the Figure 5.23 above, the pressure field around the blades is illustrated. As expected, suction is observed upstream and high pressure downstream behind the blade. Also, the circle colored with blue indicates that a vortex passes through.

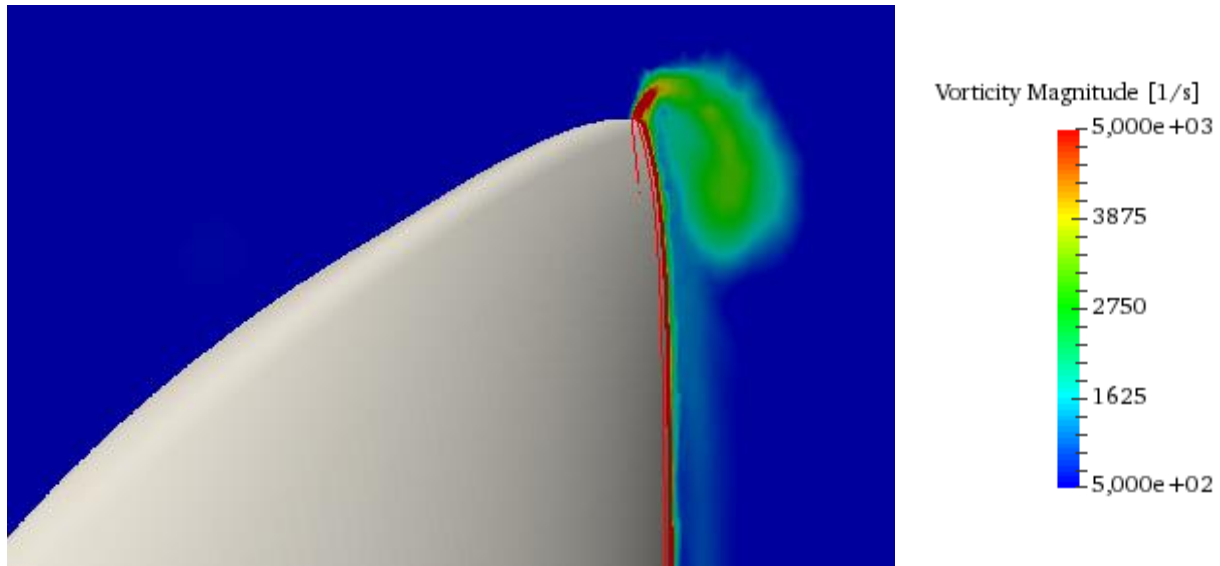


Figure 5.24 Vorticity Magnitude plot near the leading edge and tip for $J=0.6$

5.2 Optimization Phase and results

As discussed in the Theory section, the optimization process is conducted in two phases. The exploration phase that is performed using the Sobol algorithm and exploitation phase, performed with the Tangent Search Method. Both of the methods are implemented in CAESES and can be triggered, provided that the overall setup is robust and reliable.

Starting with the exploration phase, a design of experiment is carried out in order to gain insight into the design space. The Sobol algorithm produces sequences of the parameters aiming to fill the multi-dimensional space evenly. The Sobol sequence offers two significant advantages. The first one is that the generation of variants can be stopped and be restarted at any time, since the parameter combinations generated do not rely on any previous simulation. The designer can for example explore a first set of 10 Sobol designs in one run, and then, if no useful information is acquired, decide to run the next 10 designs. The second advantage is that, provided that the parameter bounds are the same, a Sobol sequence can be reproduced, since the parameter selection is not random. This attribute can be useful if the designer wants to make sure that the mesh resolution does not influence the ranking of the variants.

In order to get full understanding of the design space defined by n number of design variables, 3^n variants need to be explored. [30] For the first DoE five parameters are activated, meaning that 243 designs have to be evaluated. Using the mesh classified as medium, the simulation time is 4 hours, that means that 40 days of computations are needed. Since this simulation cost cannot be afforded, a more humble approach is followed. According to [11], n times n variants can provide a reasonable appreciation of the system's behavior and in the first DoE 27 variants are explored. The design variables and their bounds are presented in the Table 5.3 below. The parameters and their selection are discussed in detail above in the Building the parametric model (4.5) section. The simulation results are presented in the Figure 5.25 where they are ranked according to the open water efficiency.

Table 5.3 Design variables and bounds

Parameters	Lower Bound	Baseline	Upper Bound
pitchMax	1.62	1.635	1.65
pitchMaxPos	0.65	0.68	0.71
thicknessEnd	0.008	0.01	0.012
thicknessStart	0.088	0.108	0.118
camber1	0.05	0.19	0.15

pitchMax	pitchMaxPos	thicknessEnd	thicknessStart	camber1	kT	10kQ	etaOpenWater
1.6209375	0.681875	0.009625	0.0945625	0.096875	0.27478784	0.78269317	0.67051321
1.6415625	0.670625	0.010875	0.0889375	0.090625	0.28477457	0.81204148	0.66976797
1.63875	0.6575	0.0095	0.09925	0.0625	0.27934243	0.79672944	0.66961847
1.6396875	0.689375	0.008125	0.0983125	0.084375	0.2868168	0.81819221	0.66950008
1.6453125	0.663125	0.008375	0.0926875	0.103125	0.28892462	0.82496507	0.66888333
1.621875	0.70625	0.01075	0.097375	0.06875	0.27692687	0.79092116	0.668703
1.63125	0.6725	0.0105	0.09175	0.1375	0.27940638	0.79836352	0.66840089
1.6289062	0.6809375	0.0114375	0.09784375	0.1453125	0.27650819	0.79096402	0.66765582
1.6275	0.695	0.009	0.1105	0.075	0.27843382	0.79676184	0.66741327
1.6378125	0.708125	0.009375	0.1151875	0.128125	0.27963704	0.80026372	0.66736427
1.636875	0.67625	0.00875	0.112375	0.11875	0.27677227	0.79216959	0.66727644
1.6476562	0.6884375	0.0109375	0.09409375	0.1328125	0.28918566	0.82785792	0.6671482
1.6246875	0.659375	0.010125	0.1133125	0.134375	0.27481938	0.78699261	0.66692667
1.6228125	0.678125	0.011375	0.1001875	0.078125	0.27120523	0.77674741	0.66683689
1.640625	0.69875	0.01125	0.093625	0.05625	0.27979023	0.80143632	0.66675289
1.6214063	0.6659375	0.0104375	0.10534375	0.1203125	0.27318726	0.7825411	0.66673716
1.635	0.68	0.01	0.103	0.1	0.27747292	0.7950211	0.66656625
1.6265625	0.700625	0.008875	0.1039375	0.140625	0.27714053	0.79409513	0.66654409
1.6471875	0.674375	0.009125	0.1058125	0.059375	0.28398814	0.8139739	0.66633266
1.625625	0.66875	0.00925	0.108625	0.10625	0.2753981	0.78946107	0.66624139
1.6303125	0.693125	0.010375	0.1076875	0.053125	0.27650819	0.79268429	0.66620688
1.6401562	0.6734375	0.0119375	0.11659375	0.1078125	0.2773071	0.79525319	0.6659735
1.64625	0.7025	0.0085	0.10675	0.0875	0.2861368	0.82073902	0.66584021
1.6434375	0.696875	0.010625	0.1020625	0.121875	0.28450457	0.81675879	0.66526826
1.6439062	0.6509375	0.0094375	0.11284375	0.0953125	0.27633243	0.79343018	0.66515755
1.6490625	0.685625	0.011875	0.1114375	0.065625	0.2816508	0.80910752	0.66482313
1.6359375	0.651875	0.011625	0.1095625	0.146875	0.27580752	0.79313545	0.66414074

Figure 5.25 27 Sobol designs for the first DoE

From the above results we can see that K_T values are within a range of $\pm 3\%$ with the minimum $K_{TMin} = 97,51\% K_{TBaseline}$, and the maximum $K_{TMax} = 103,9\% K_{TBaseline}$. About the same fluctuations are observed for $10K_Q$, with $10K_{QMin} = 97,3\% 10K_{QBaseline}$ and $10K_{QMax} = 103,6\% 10K_{QBaseline}$. The design corresponding to the highest efficiency shows a 0.8% improvement in open water efficiency that comes from a 1.12% decrease for K_T and a more significant 1.89% decrease for $10K_Q$. Even if some improvement has been made, this design cannot be accepted as optimum since K_T is less than the baseline design. The optimum design should produce at least the same thrust as the baseline design. On the Figure 5.26 the influence of the parameter pitchMax on K_T is depicted. Increasing the parameter pitchMax, increases K_T and vice versa. On the Figure 5.27, one can notice the influence of the parameter ThicknessStart on the open water efficiency. Decreasing the thickness of the blades, increases the open water efficiency. Figures 5.28 and 5.29 show how the Sobol designs are spread evenly, with two parameters Camber1 and PitchMaxPos as an example.

Figure 5.26 Influence of parameter pitchMax on K_T

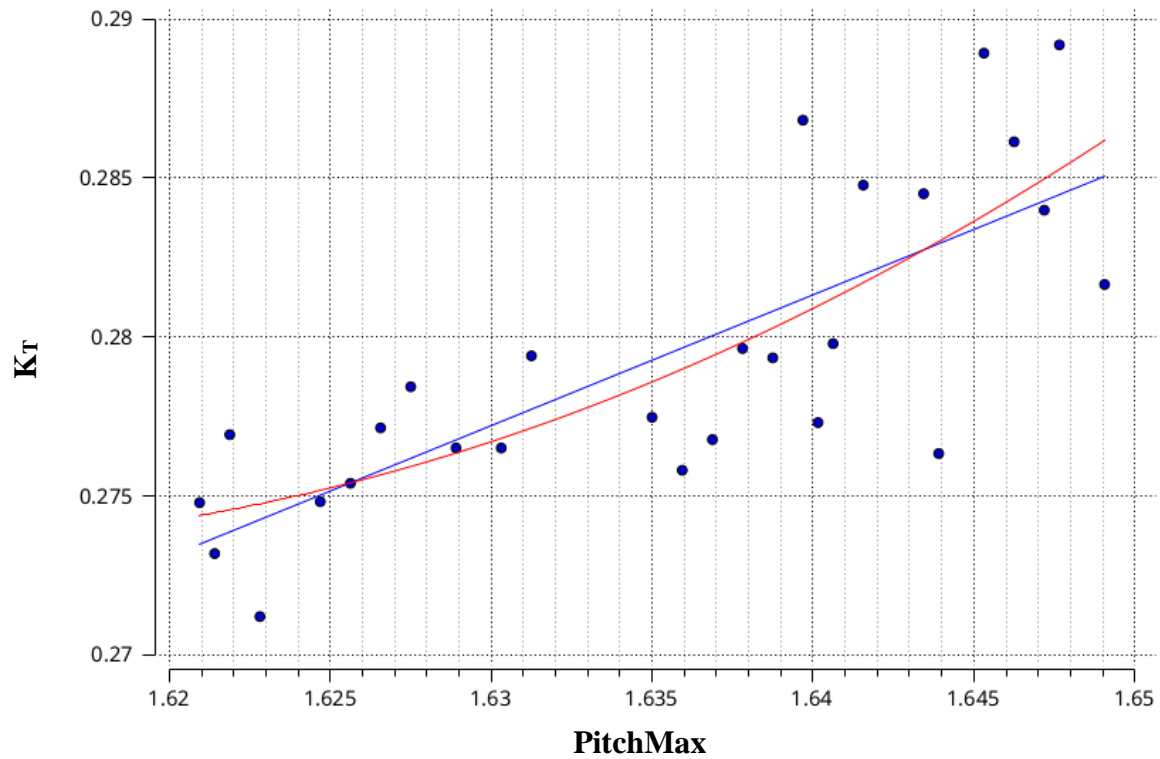


Figure 5.27 Influence of parameter ThicknessStart on open water efficiency

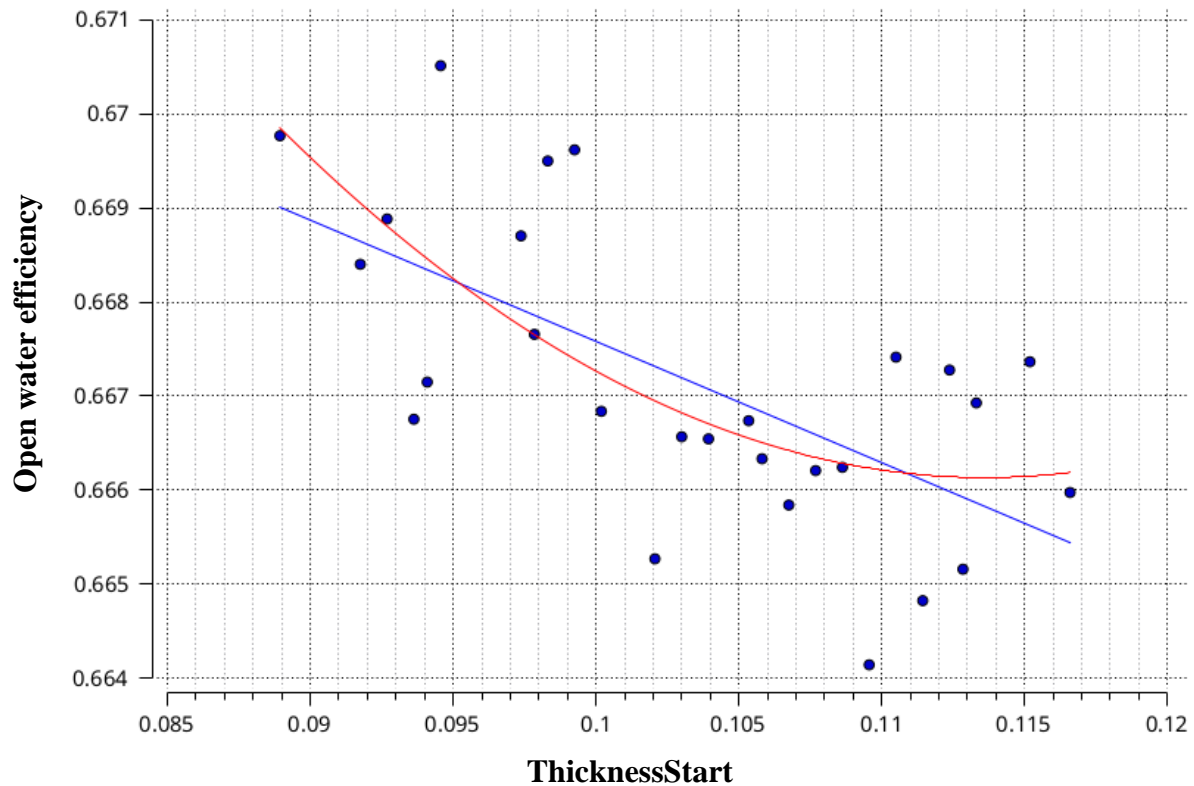


Figure 5.28 Camber1 values for every run

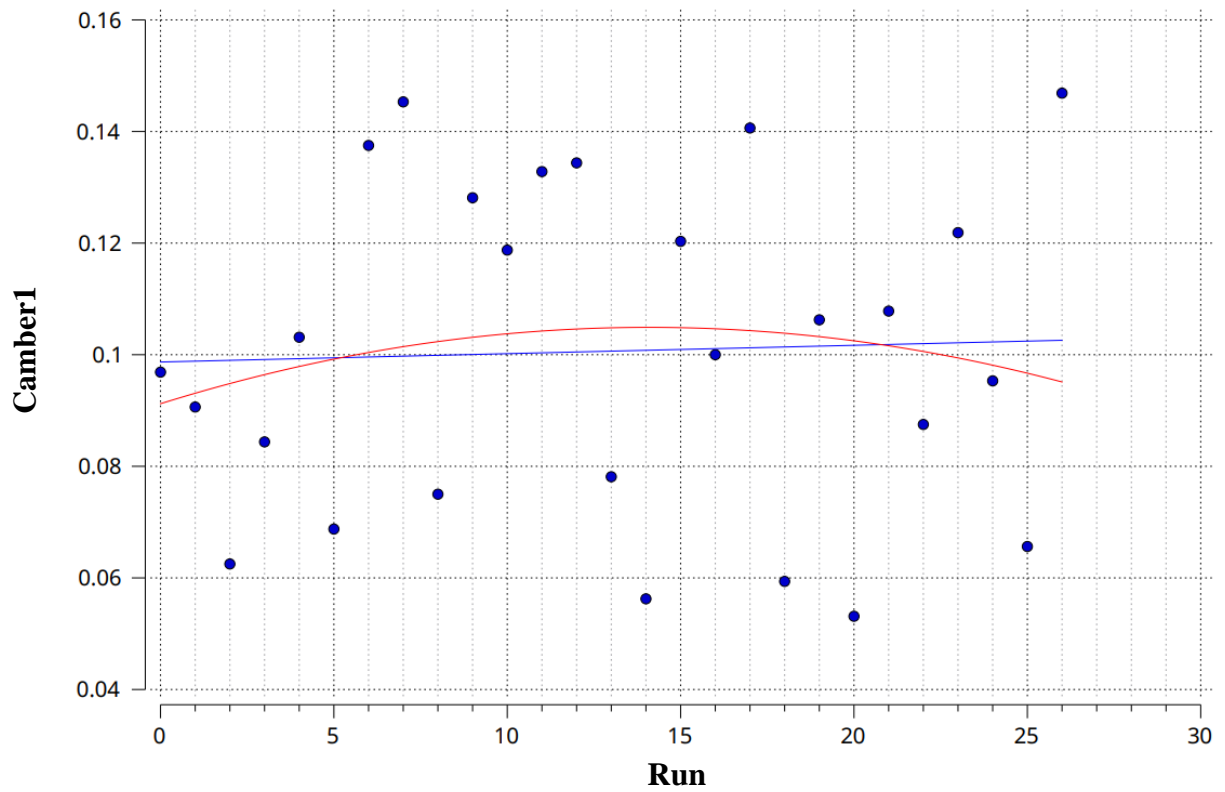
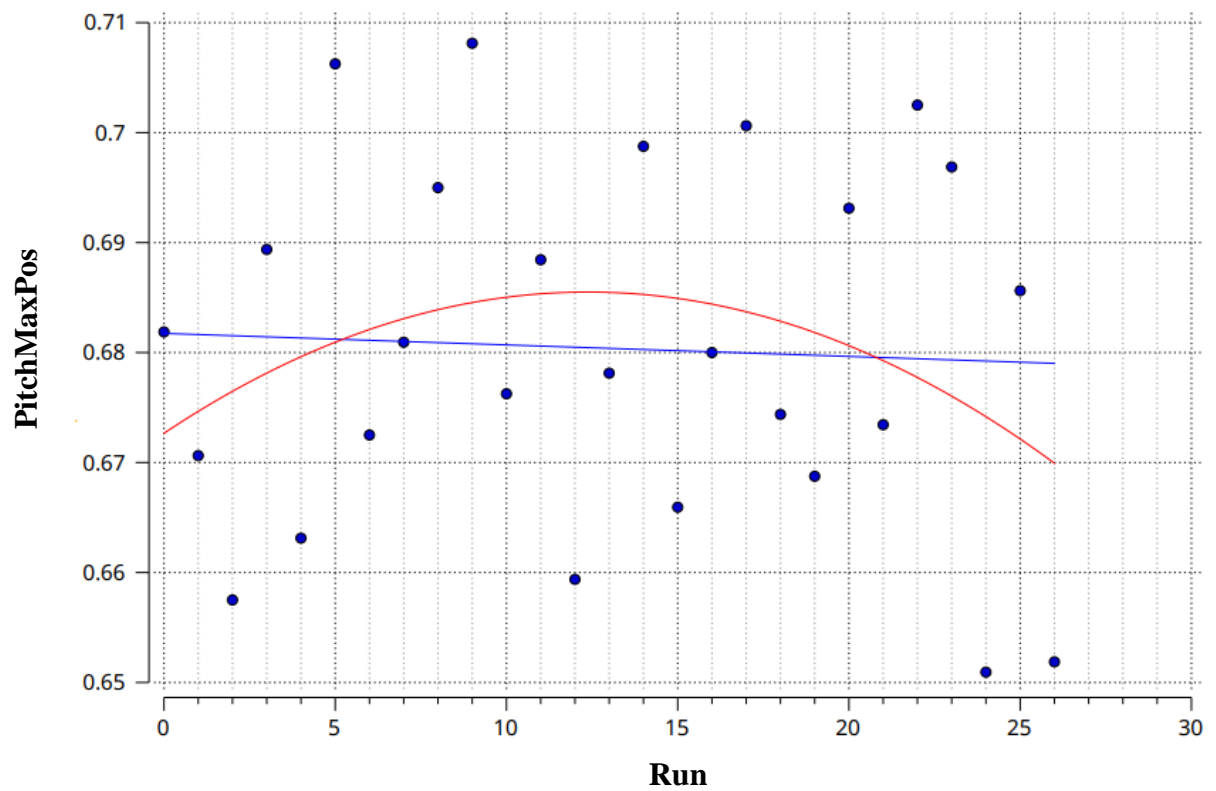


Figure 5.29 PitchMaxPos values for every run



In order to avoid designs with less K_T than the baseline the pitchMax parameter is deactivated. Also, since a smaller design space will be easier to explore, the two thickness related parameters thicknessStart and thicknessEnd are deactivated as well. After all, thinner blades are expected to yield higher efficiency but there are strength limitations that are not considered in this work. One more parameter is activated for the camber function resulting in three parameters and 30 designs are investigated. We now have a new design space, totally different than the previous one and much smaller. The bounds of the parameters are kept the same as in the first Sobol sequence, since they resulted in about $\pm 3\%$ difference for K_T and $10K_Q$.

Table 5.4 New design variables

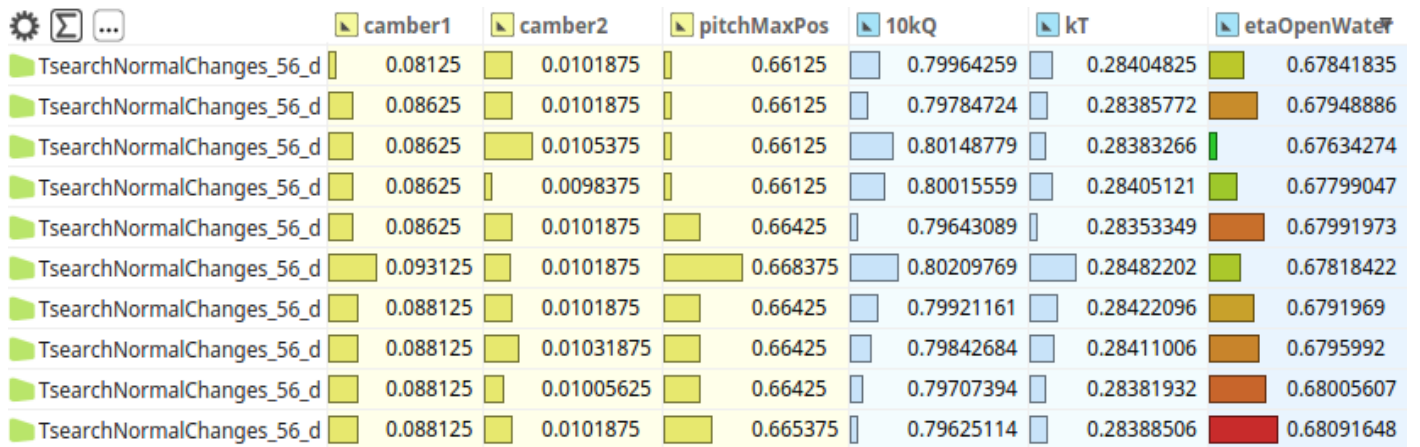
Parameters	Lower Bound	Baseline	Upper Bound
pitchMaxPos	0.65	0.68	0.71
camber1	0.05	0.1	0.15
camber2	0.008	0.01	0.015

pitchMaxPos	camber1	camber2	10kQ	kT	etaOpenWater
0.66125	0.08125	0.0101875	0.79964746	0.28404876	0.67841542
0.6753125	0.0640625	0.009640625	0.79965628	0.28271388	0.67521978
0.6696875	0.1484375	0.008109375	0.79214388	0.28000368	0.67508902
0.670625	0.121875	0.01215625	0.80846397	0.28556653	0.6746026
0.6509375	0.1296875	0.014671875	0.80286127	0.28346844	0.67431929
0.663125	0.134375	0.00953125	0.80523694	0.28354681	0.67251574
0.68375	0.09375	0.0093125	0.79036484	0.27731852	0.67012009
0.655625	0.096875	0.01390625	0.80140892	0.28091684	0.66946053
0.6546875	0.0734375	0.009859375	0.78968454	0.27648344	0.66867774
0.6678125	0.1015625	0.014015625	0.81328976	0.28438691	0.66782962
0.7071875	0.1359375	0.010734375	0.81583134	0.28522847	0.6677192
0.70625	0.05625	0.0119375	0.79935848	0.27943254	0.66763142
0.6884375	0.1421875	0.012046875	0.81470843	0.28479793	0.66763023
0.696875	0.128125	0.01259375	0.81950411	0.28642354	0.6675118
0.6809375	0.0796875	0.011171875	0.80756914	0.28212135	0.66720243
0.6771875	0.0859375	0.014234375	0.81672117	0.28507597	0.6666351
0.704375	0.090625	0.00996875	0.81118857	0.28295482	0.66618776
0.674375	0.140625	0.01346875	0.81837435	0.28529346	0.665796
0.6996875	0.0984375	0.011609375	0.81471381	0.28391351	0.66555256
0.6846875	0.1234375	0.013359375	0.81688301	0.28451922	0.66520135
0.69125	0.13125	0.0136875	0.81974084	0.28548932	0.66514246
0.7053125	0.1140625	0.013140625	0.81730134	0.28461928	0.66509469
0.6725	0.0875	0.012375	0.81157243	0.28260986	0.66506088
0.7025	0.1375	0.008875	0.81318554	0.28313684	0.66497928
0.708125	0.109375	0.01478125	0.81938718	0.28527501	0.66493002
0.7034375	0.0671875	0.013796875	0.82245941	0.28615171	0.66448204
0.68	0.1	0.0115	0.81109481	0.28216368	0.6644019
0.66875	0.06875	0.0145625	0.81141028	0.28181058	0.66331248
0.6575	0.1125	0.014125	0.8035868	0.27835546	0.66155861
0.67625	0.10625	0.0084375	0.80464613	0.27768543	0.65909732

Figure 5.30 30 Sobol designs

The simulation results from the second Sobol sequence show that indeed almost all designs with less K_T than the baseline are avoided. Many designs are found that perform better than the baseline. The best design corresponds to a 1.94% increase in open water efficiency attributed to a 2.16% increase in K_T and only 0.22% increase in $10K_Q$.

At this point, the exploration phase has been finished and the exploitation phase initiates. Starting from the best Sobol design which is the one corresponding to an 1.94% increase in open water efficiency the T-Search algorithm is executed that will search locally around the promising candidate trying to detect an even better design. The objective that is set to the T-search algorithm is to minimize $10K_Q$. In contrast to the Sobol algorithm, every design generated by the T-search algorithm depends on the simulation result of the previous design. The algorithm is stopped after 10 iterations with a 0.4% decrease in $10K_Q$ compared to the best Sobol design.



	camber1	camber2	pitchMaxPos	10kQ	kT	etaOpenWater
TsearchNormalChanges_56_d	0.08125	0.0101875	0.66125	0.79964259	0.28404825	0.67841835
TsearchNormalChanges_56_d	0.08625	0.0101875	0.66125	0.79784724	0.28385772	0.67948886
TsearchNormalChanges_56_d	0.08625	0.0105375	0.66125	0.80148779	0.28383266	0.67634274
TsearchNormalChanges_56_d	0.08625	0.0098375	0.66125	0.80015559	0.28405121	0.67799047
TsearchNormalChanges_56_d	0.08625	0.0101875	0.66425	0.79643089	0.28353349	0.67991973
TsearchNormalChanges_56_d	0.093125	0.0101875	0.668375	0.80209769	0.28482202	0.67818422
TsearchNormalChanges_56_d	0.088125	0.0101875	0.66425	0.79921161	0.28422096	0.6791969
TsearchNormalChanges_56_d	0.088125	0.01031875	0.66425	0.79842684	0.28411006	0.6795992
TsearchNormalChanges_56_d	0.088125	0.01005625	0.66425	0.79707394	0.28381932	0.68005607
TsearchNormalChanges_56_d	0.088125	0.0101875	0.665375	0.79625114	0.28388506	0.68091648

Figure 5.31 T-Search results

Overall, a 2.3% increase in open water efficiency is achieved coming from a 2.11% increase in K_T and an insignificant 0.2% decrease in $10K_Q$ compared to the baseline design. It is important to mention that this design is not necessarily the global optimum since the local search with the T-Search algorithm was executed only for the best Sobol design. It is possible that starting from another Sobol design, there would be more room for improvement with the local search. However, due to the limited resources, the local search is carried out only for one candidate.

5.3 Process Automation and Validation

As discussed in the Software section, any operation in CAESES can be executed with commands inside a Feature Definition and any design process can be scripted and automated. During the course of this thesis, the process of importing a propeller model, splitting the blade, generating the periodic domain, performing the necessary Boolean operations and then coupling the geometry to OpenFOAM has been performed a number of times for different propellers. In this context, an effort is made to automate the entire process from geometry import to the calculation of the open water characteristics. The goal is to minimize the manual effort needed to get a rough calculation of the open water characteristics of a propeller.

To begin with, the Software Connector containing the OpenFOAM configuration can be exported as a file and be imported in a new project with a new propeller. Moreover, in any OpenFOAM dictionary connected as an input file, for example the **meshDict** that controls the mesh generation, parameters can be introduced as entries. In this way, basic configurations that change in the CFD setup, for example the inlet velocity, rpm, end time of the simulation, density and so on, can be controlled by parameters through CAESES and easily be adjusted. Geometrical features like the diameter or pitch can be linked to these parameters reducing the manual effort needed.

Automating the import and splitting of the blade could not be achieved. There are too many ways and file formats to import a propeller. The imported geometry could include the hub and shaft or not, it can be a BrepPart or an STL and taking all of these parameters under consideration would be an exhaustive process of doubtful results. As expected, another bottleneck is the grid generation process. Grid generation with cfMesh is a trial and error procedure, performed by trying out settings, visualizing the resulting mesh and checking the convergence of the computations. However, the configuration of the grid generator can be automated somehow. The workaround that is used is to keep cell size ratios between for example the cell size in the volume refinement and the leading edge constant and as a function of the diameter. The mesh configuration used for the PPTC is used as a guideline.

No obstacle exists in the automation of the domain construction process. These are operations performed in CAESES and they can be easily encapsulated inside a Feature Definition. Thus, a Feature Definition is created that given two points, one on the leading and one on the trailing edge, the watertight blade geometry as well as a Brep containing the hub, shaft and cap and some basic parameters like the propeller diameter and the size of the domain, it will automatically generate the needed surfaces and perform the necessary operations to export a Brep ready to be given to the grid generator. The generation of the hub, shaft and cap is also scripted and automated within a separate Feature Definition.

In Figure 5.32, part of the code is presented responsible for the generation of two circles needed for the domain construction. On the same Figure on the right, the executed feature exports the final Brep along with the colors needed to apply the boundary conditions. In the end, the only manual effort needed is the preparation of the blade surface (splitting, merging with colors).

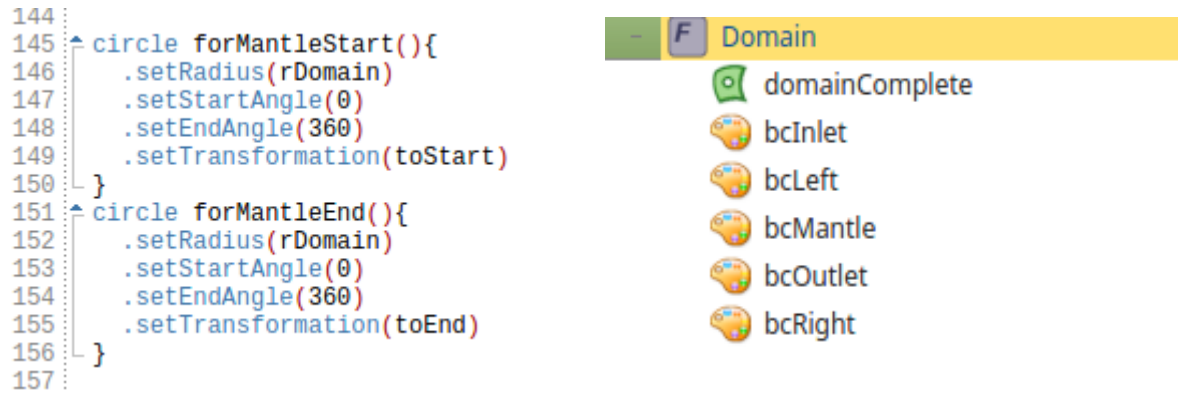


Figure 5.32 Example of the feature (left), Feature output (right)

In order to investigate the flexibility of the OpenFOAM setup and the Feature Definitions that have been developed, simulations are carried out for two more powerboat propellers. The propeller geometry of these two propellers is available in an IGES file format. Their performance has been evaluated using the Vortex Lattice Method (VLM) and the results are provided to the author for comparison. The first propeller is a Wageningen B-series propeller depicted on the left, and a custom design on the right.

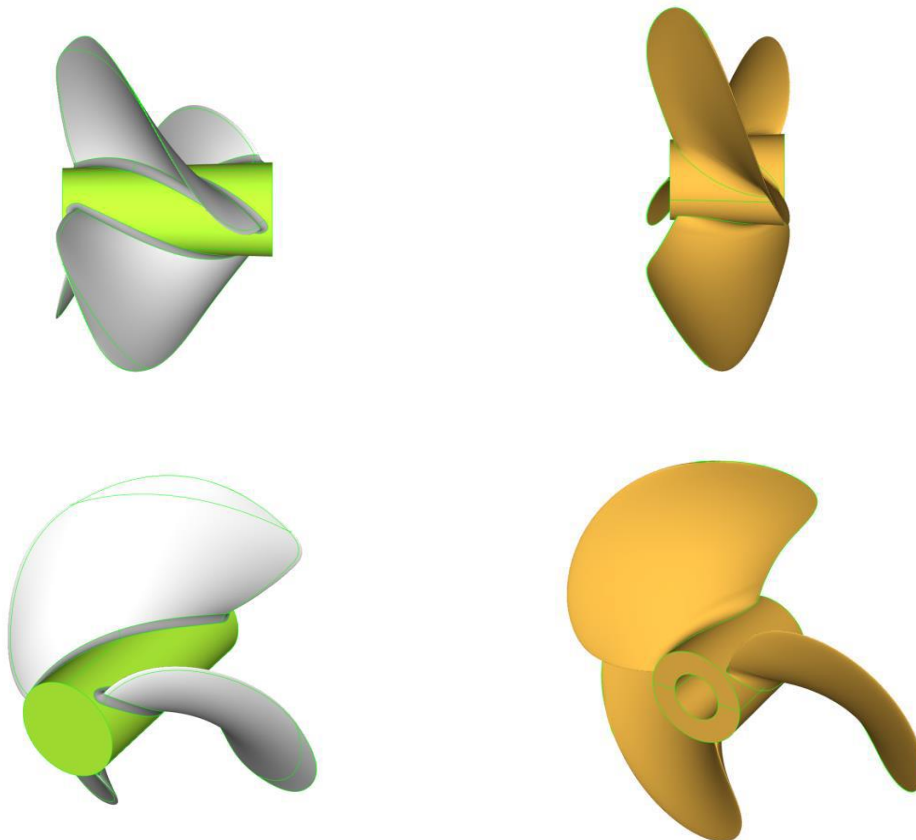


Figure 5.30 Views of the Wageningen B-series and of the custom design [55]

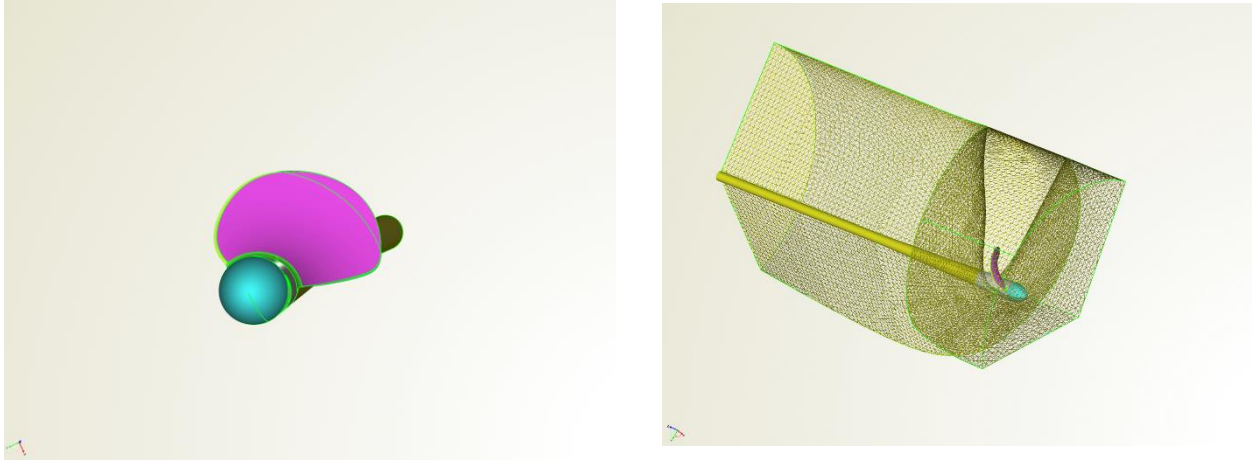


Figure 5.31 Wageningen B-Series geometry preparation (left), domain construction (right)

The propeller blade has to be treated manually first. The edges of the blade are separated and connected into the same Brep again but with different colors. This will allow the control of the cell size on every part separated. The feature definition that has been created is used to generate the hub shaft and cap and the two solids are then combined into a single Brep as seen in Figure 5.34 on the left. The feature definition responsible for generating the domain is then used, as seen in Figure 5.34 on the right. The mesh settings are adjusted according to the propeller diameter and the final mesh consists of 2135613 cells.

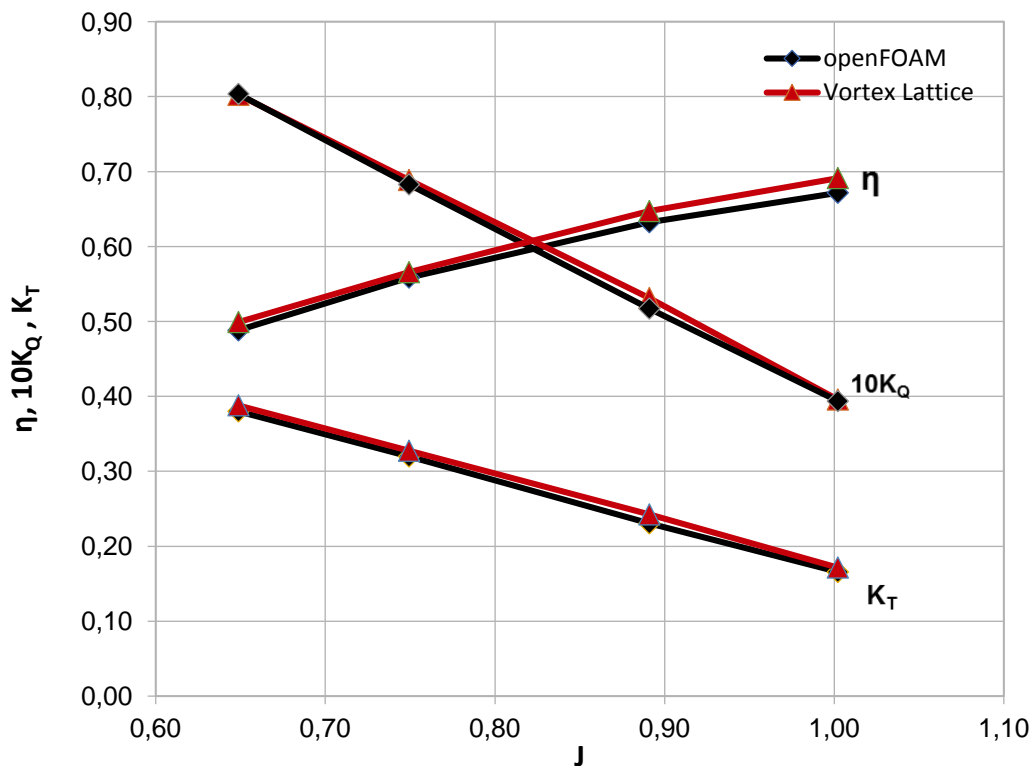


Figure 5.32 Open water chart for Wageningen B-series

Table 5.5 Open water characteristics obtained with OpenFOAM and Vortex lattice for the Wageningen B-Series propeller

OpenFOAM					
Vs[m/s]	n[rps]	J	K_T	10K_Q	η₀
2.5720	11.7300	0.6487	0.3799	0.8034	0.4882
6.1730	24.2800	0.7494	0.3197	0.6828	0.5584
9.2600	30.5700	0.8911	0.2306	0.5172	0.6323
16.3760	48.0700	1.0021	0.1658	0.3938	0.6715
Vortex Lattice					
Vs[m/s]	n[rps]	J	K_T	10K_Q	η₀
2.5720	11.7300	0.6487	0.3879	0.8025	0.4990
6.1730	24.2800	0.7494	0.3275	0.6888	0.5659
9.2600	30.5700	0.8911	0.2425	0.5313	0.6472
16.3760	48.0700	1.0021	0.1715	0.3958	0.6911

**Figure 5.33 Pressure distributions on the suction side (left) and the pressure side (right) for J=0.6487**

The open water characteristics obtained from the viscous openFOAM analysis are in agreement Vortex Lattice method. The maximum percentage difference is observed is 5.16% for K_T for advance coefficient 0.8911. For the open water efficiency, the maximum percentage difference is 2.91% for advance coefficient 1.002.



Figure 5.34 Custom made geometry preparation (left), domain construction (right)

The same procedure is followed for the custom-made propeller as well. The blade is split manually, and the rest part of the process is performed automatically using feature definitions. The mesh consists of 2531350 cells.

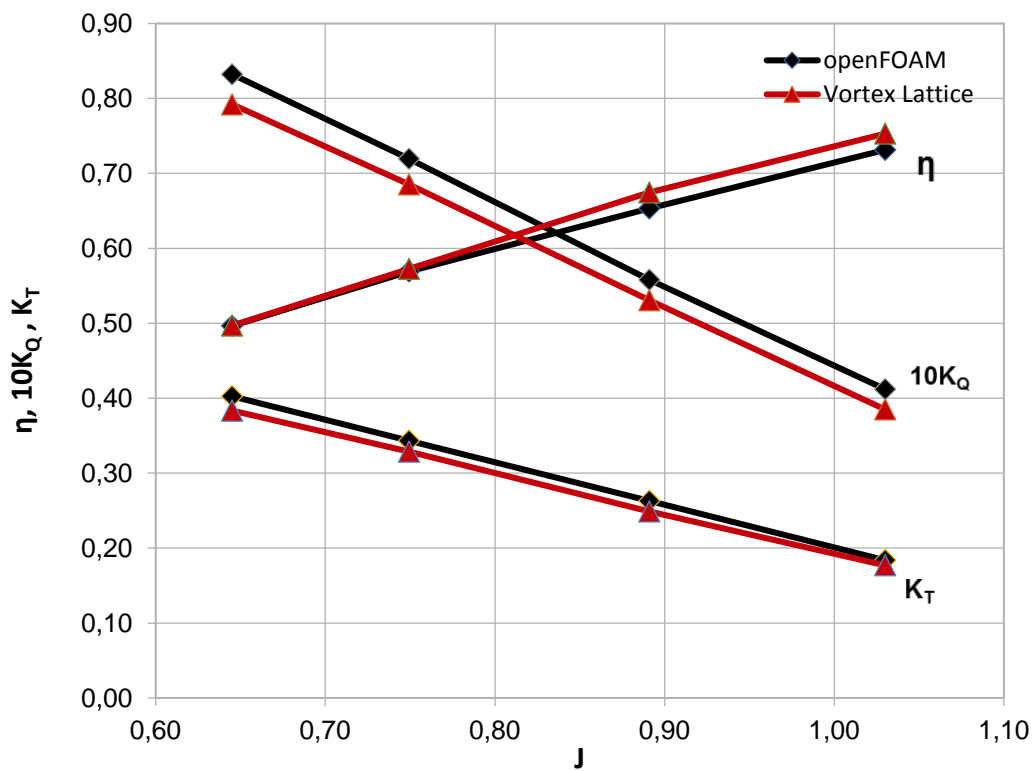
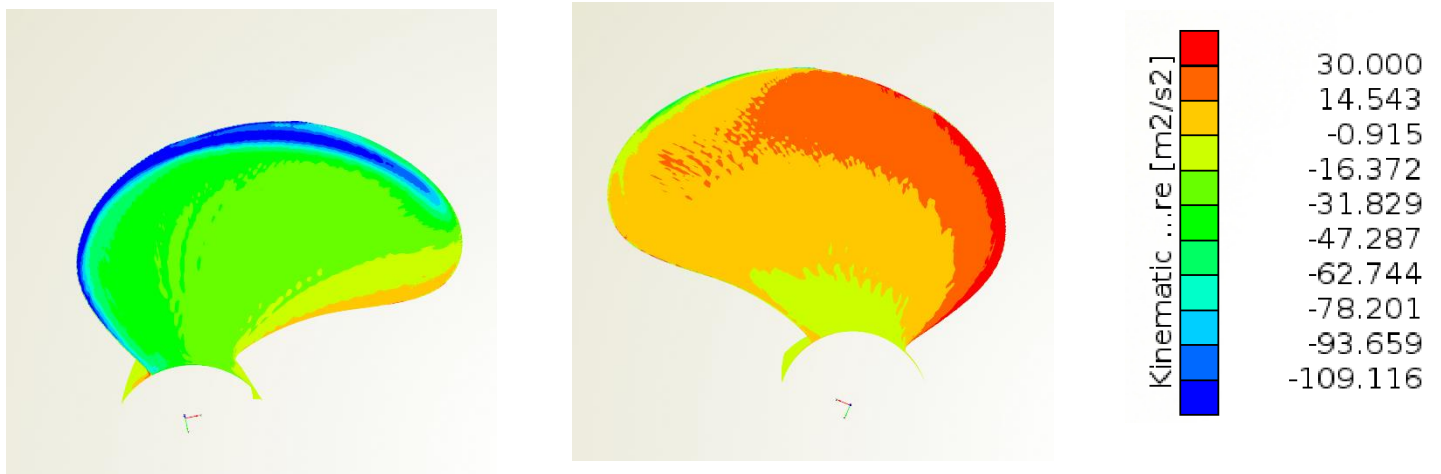


Figure 5.35 Open water chart for Wageningen B-series

Table 5.6 Open water characteristics obtained with OpenFOAM and Vortex lattice for the Wageningen B-Series propeller

OpenFOAM					
Vs[m/s]	n[rps]	J	K_T	10K_Q	η_0
2.5720	11.7300	0.6450	0.4025	0.8323	0.4964
6.1730	24.2800	0.7494	0.3433	0.7193	0.5692
9.2600	30.5700	0.8911	0.2631	0.5582	0.6534
16.3760	48.0700	1.0300	0.1838	0.4120	0.7313
Vortex Lattice					
Vs[m/s]	n[rps]	J	K_T	10K_Q	η_0
2.5720	11.7300	0.6450	0.3835	0.7922	0.4969
6.1730	24.2800	0.7494	0.3288	0.6851	0.5724
9.2600	30.5700	0.8911	0.249	0.5307	0.6744
16.3760	48.0700	1.0300	0.1769	0.3850	0.7530



For this propeller, the differences observed are slightly higher. More specifically, $10K_Q$ is predicted up to 6.5% higher ($J=1.03$) with OpenFOAM compared to the VLM and K_T up to 5.3% ($J=0.8911$). For open water efficiency the maximum percentage difference is 3.2% for $J=0.8911$.

Overall, for both propellers, the open water efficiency calculated with OpenFOAM is slightly lower for all advance coefficients, due to the high prediction of K_Q . However, the OpenFOAM analysis is in agreement with the VLM that the custom design performs better than the Wageningen – B series.

6.1 Conclusions

The goal of the present work was to evaluate the propeller open water characteristics using open-source CFD and secondly to develop a closed optimization loop inside CAESES, automating the CFD tools utilized in order to optimize a propeller. The workflow is designed to be as computationally cheap as possible, using the one blade passage approach, the MRF method to model the rotation and the $k-\omega$ SST to model the turbulence. The validation is performed with the Potsdam Propeller Test Case by visualizing the pressure and velocity fields and by comparing the open water characteristics with experimental data. A maximum percentage difference of 1.66% is observed for K_T , a 5.35% for K_Q and a maximum of 4.5% difference on the open water efficiency. Grid independence is achieved using about 1.8 million cells corresponding to four hours of computation time with an i5 processor of 2.5GHz. A parametric model is constructed inside CAESES for the PPTC. The model is designed so that the parameters have maximum impact on the blade shape. A closed optimization loop is realized within CAESES, where the execution of all the OpenFOAM and grid generation utilities are automated. A two-phase approach is followed for the optimization, starting with the exploration phase that is carried out with the Sobol sequence and the exploitation phase performed with the Tangent Search Method. The best design shows a 2.3% increase in open water efficiency with a 2.1% increase in K_T . Partial automation of the methodology used was achieved aiming to accelerate the connection of any new propeller model. The features developed were put to use with two powerboat propellers. The results acquired are in agreement with relevant Vortex Lattice results.

Throughout this thesis, CAESES has been used as a pre-processor, blade design tool, parametric CAD modeler, software connector and the core of the optimization process. It proved to be a versatile and powerful software with unlimited capabilities. To fully take advantage of these capabilities, the designer needs to use the feature programming language. When it comes to propellers, a number of entities and pre-programmed scripts called features are available that accelerate and simplify the propeller design process resulting in watertight models ready for meshing.

CfMesh is a simple, robust and fast unstructured grid generator that can be very efficient for simple shapes. Even though it served sufficiently for the task at hand, configuring its settings and manipulating the quality of the resulting mesh proved to be a very demanding, time-consuming trial and error process. Without the pre-processing and preparation of the blade with CAESES that allowed the local control of the cell sizes, the desirable mesh quality may not have been achieved.

OpenFOAM is a powerful simulation toolbox that can serve as a reliable tool for propeller analysis. Visualization of the flow field can be done with the open-source pre-processor Paraview. The lack of licensing costs and issues make it tempting choice for CFD analysis. With minimum effort, openFOAM simulations can be configured to run in parallel, reducing simulation time, opening the way to full scale simulations. However, due to its open-source nature, the learning curve for a new user is quite steep since the only assistance provided comes from the online community.

6.2 Suggestions for future work

Most of the choices made and approaches followed in the thesis project were related to the computational cost. Provided that more computational power is available, transient simulations could be carried out using the openFOAM solver PimpleDyMFOAM that allows for dynamic meshes. The transient solver is expected to yield more accurate predictions of the propeller performance. Another issue of interest is the effect of the periodic boundaries on the results of the simulations. This modification would be very simple and easy to perform since the mesh and solver settings do not require changes.

Using the parallelization capabilities of openFOAM, full scale simulations would be viable as well. The scale effects on propeller open water performance are always a subject of interest in marine hydrodynamics. With variable propeller geometry easily generated with CAESES, numerical simulation results can be compared for model scale and full scale propellers with specific geometrical features (highly skewed, different blade area ratios).

Another suggestion is to use and validate commercial grid generators like ANSA, Pointwise or gridPro. These tools are compatible with openFOAM and provide maximum control on the boundary layer resolution. Thus, the control of the y^+ values on the propeller blades would not be an issue. This will allow the use of wall functions and will bring down the cell count allowing for more designs to be explored. However, their automation will require more effort and considerations than cfMesh.

Finally, regarding propeller optimization, a very interesting application would be to connect CAESES to a BEM code that will be able to evaluate a great number of designs in short time. Then, global optimization could be performed using an evolutionary algorithm like NSGA-II that is already implemented in CAESES.

Bibliography

- [1] T. TURUNEN, T. SIIKONEN, J. LUNDBERG and R. BENSOW, "Open-water computations of a marine propeller using OpenFOAM".
- [2] J. Carlton, *Marine Propellers and Propulsion*, 2012.
- [3] F. M. White, *Viscous Fluid Flow*, 2006.
- [4] "https://openfoamwiki.net/index.php/See_the_MRF_development," 2018. [Online].
- [5] S. P. Polyzos, *Calculation of the Hydrodynamic Resistance of Appendages on Conventional Vessels*, Athens, 2017.
- [6] C. J. Greenshields, "OpenFOAM Programmer's Guide," 2015.
- [7] T. Holzmann, *Mathematics Numerics, Derivations and OpenFOAM*, Leoben, 2017.
- [8] H. Jasak, *Error analysis and estimation for the finite volume method with application to fluid flows*, London: PhD Thesis, Imperial College of Science, Technology and Medicine, 1996.
- [9] S. Patankar, *Numerical Heat Transfer and Fluid Flow*, Taylor and Francis, 1980.
- [10] "www.wikiversity.org," [Online].
- [11] S. Harries, *Practical shape optimization using CFD*, Berlin: Friendship Systems, 2015.
- [12] K. Hochkirch, S. Harries and C. Abt, *Modeling meets Simulation - Process integration to improve design*, 2004.
- [13] C. Abt and S. Harries, *Qualitative assessment of available geometric modelling techniques*, 2007.
- [14] J. Thompson, *Handbook of grid generation*, 1999.
- [15] J. Kortelainen, "Meshing Tools for Open Source CFD - A Practical Point of View," 2009.
- [16] J. Rhoads, "Effects of grid quality on solution accuracy," *OpenFOAM Workshop 2014*, 3 July 2014.
- [17] F. Vesting, *Marine Propeller Optimisation - Strategy and Algorithm Development*, Gothenburg, Sweden, 2015.
- [18] "https://en.wikipedia.org/wiki/Multi-objective_optimization," 2018. [Online].

- [19] "https://en.wikipedia.org/wiki/Mathematical_optimization," 2018. [Online].
- [20] E. F. Campana, D. Peri, Y. Tahara, A. Pinto and F. Stern, "A comparison of global optimization methods with application to ship design.," *Processings of the 5th Osaka Colloquium on Advanced Research on Ship Viscous Flow and Hull Form Design* , 2005.
- [21] W. Luo and L. Lan, "Design Optimization of the Lines of the Bulbous Bow of a Hull Based on Parametric Modeling and Computational Fluid Dynamics Calculation," *Mathematical and Computational Applications*, 2016.
- [22] S. Harries, K. Lorentz, J. Palluch and E. Praefke, "Appification of Propeller Modeling and Design via CAESES," 2018.
- [23] J. Jung, K. Lee, I. Song and J. Han, "Design of marine propellers using genetic algorithm," 2007.
- [24] J.-H. Chen and Y.-S. Shih, "Basic design of a series propeller with vibration consideration by genetic algorithm," *Marine Science and Technology*, 2007.
- [25] J.-b. Suen and J.-s. Kouth, "Genetic algorithms for optimal series propeller design," *Built Enviroment*, 1999.
- [26] "https://en.wikipedia.org/wiki/Sobol_sequence," 2018. [Online].
- [27] R. Hilleary, *The Tangent Search Method of Constrained Minimization*, Monterey US: Postgraduate School, 1966.
- [28] M. Save and W. Prager, "Structural Optimization," *Mathematical Programming*, vol. 2, no. Mathematical Programming, 1990.
- [29] Friendship Systems, "CAESES Documentation," 2018.
- [30] K. Mizzi, Y. K. Demirel, C. Banks, O. Turan, P. Kaklis and M. Atlar, "Design optimisation of Propeller Boss Cap Fins for enhanced propeller performance," *Applied Ocean Research*, 2017.
- [31] CFDDirect, "The Architects of OpenFOAM," 2018. [Online].
- [32] C. J. Greenshields, "OpenFOAM User Guide," The OpenFOAM Foundation, 2018.
- [33] D. F. Juretic, "CfMesh User Guide," Zagreb, 2015.
- [34] SVA, "Schiffbau-Versuchsanstalt Potsdam," 2018. [Online].
- [35] S. Subhas, "CFD Analysis of a Propeller Flow and Cavitation," *International Journal of Computer Applications* 55 , October 2012.

- [36] T. Watanabe, "Simulation of steady and unsteady cavitation on a marine propeller using RANS CFD code," *Fifth international Symposium on Cavitation*, November 2003.
- [37] J. Yao, "On the propeller effect when predicting hydrodynamic forces for manoeuvring using RANS simulations of captive model tests," 2015.
- [38] J. A. C. Y. B. TJR Hughes, "Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement," *Computer methods in applied mechanics and engineering*, 2005.
- [39] L. Davidson, *Fluid mechanics, turbulent flow and turbulence modeling*, 2018.
- [40] P. A. Keun Woo Shin, "CFD Analysis of Scale Effects on Conventional and Tip-Modified Propellers," *Fifth International Symposium on Marine Propulsors*, 2017.
- [41] T. Turunen, "Analysis of Multi-Propeller Marine Applications by Means of Computational Fluid Dynamics," *Master's Thesis*, 2014.
- [42] "OpenFOAM Documentation," [Online]. Available: www.openfoam.com.
- [43] W. M. H K Versteeg, *An Introduction to Computational Fluid Dynamics*, 2007.
- [44] H. Nilsson, "Rotating machinery training at OFW10," 2015.
- [45] S. C. S. Salim, "Wall y+ strategy for dealing with wall-bounded turbulent flows," *Proceedings of the International MultiConference of Engineers and Computer Scientists 2009*, 2009.
- [46] A. AlOnazi, *Design and Optimization of OpenFOAM-based CFD Applications for Modern Hybrid and Heterogeneous HPC Platforms*, Dublin, 2013.
- [47] Gambit, "User guide".
- [48] "CFD online," [Online]. Available: www.cfd-online.com.
- [49] F. Moukalled, L. Mangani and M. Darwish, *The Finite Volume Method in Computational Fluid Dynamics, An Advanced Introduction with OpenFOAM and Matlab*, Springer, 2016.
- [50] Engys, "A comprehensive tour of snappyhexmesh," 7th OpenFOAM Workshop, 2012.
- [51] L. DA-Qing, "Validation of RANS predictions of open water performance of a highly skewed propeller with experiments," *Conference of Global Chinese Scholars on Hydrodynamics*, 2002.
- [52] G. Haller, "An objective definition of a vortex," *J. Fluid Mech.*, 2005.
- [53] F. Feruglio, "Hydrodynamic study of a bow of a combatant hull," 2018.

- [54] "https://en.wikipedia.org/wiki/Sobol_sequence," 2018. [Online].
- [55] S. Harries, K. Lorentz, J. Palluch and E. Praefke, "Appification of Propeller Modeling and Design via CAESES".

Appendix A

With cfMesh, a number of propellers have been tried, good results have been obtained in comparison with the experimental data and the overall experience with cfMesh is satisfactory. However, an issue that persists is the generation of prism layers at the blade wall while maintaining the quality of the mesh in acceptable levels. While this has been achieved for the propellers that have been tried, it has proven to be a very time-consuming trial and error process. In this context, a promising solution would be to use a hybrid approach, with a structured grid around the blade including viscous layer blockings combined with an unstructured mesh for the rest of the domain created by an automatic unstructured grid generator like cfMesh which will easily create high quality hexahedral cells in the areas that are easy to discretize. After some research, no open-source tool is found to support the functionality of a smooth structured grid around a twisted geometry like the propeller blade. However, a tool called Cfdmsh is found, a python library structured meshing based on the grid generator SALOME. SALOME is an open source tool that supports automation using python scripts, it is widely used both for CFD and FEM simulations and shows good compatibility with OpenFOAM.

The library (cfd-mesh) is quite limited in 3D, however it is possible generate viscous layer blockings around extruded geometries or surfaces of revolution but also around twisted wings. The last case is selected for evaluation, and the first issue to be examined is the possibility of combining the two meshes in a file format OpenFOAM can handle.

OpenFOAM offers two utilities for the connection of different meshes, these are StitchMesh and MergeMesh. MergeMesh takes the meshes from two different cases and merges them into the master case. It reads the system/controlDict of both cases and uses the startTime, so special attention needs to be paid if the meshes are moving. The utility is triggered with the command:

```
$mergeMeshes <case1> <case2>
```

With <case1> as the master case. In the unified mesh, the two meshes will keep their boundary conditions. StitchMesh is responsible for coupling two uncoupled mesh regions, belonging to the same case. The utility is triggered with the command:

```
$stitchMesh masterPatch slavePatch
```

For partially overlapping patches < -partial> < -toleranceDict> can be added after <slavePatch> that will help couple non-conforming patches.

SALOME exports the mesh in UNV file format, and different patches can be distinguished. Separating the external surface mesh as a different patch is important for the stitching of the meshes later. The mesh can be converted to OpenFOAM format using the command `ideasUnvToFoam <filename.unv>` with the unv file located in a valid OpenFOAM folder structure, that is including a valid controlDict file. It is then possible to run the mesh conversion routine, and this will create a folder called **polyMesh** within the *constant* folder, with the mesh in OpenFOAM format including the patches separated.



Figure 1. Geometry to be meshed including the surface mesh

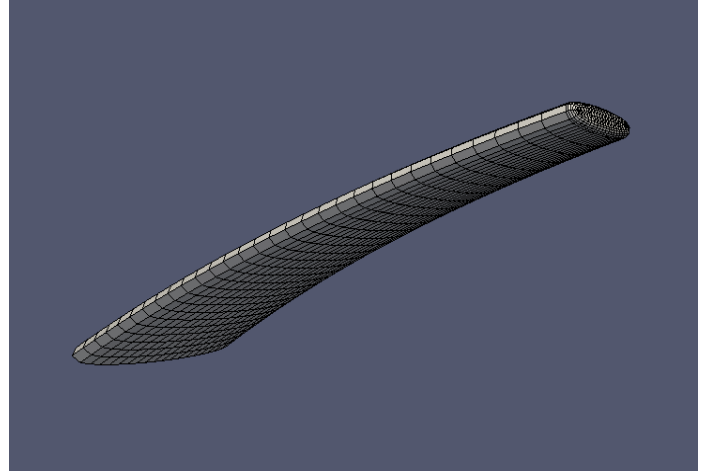


Figure 2. Structured grid around the bended wing

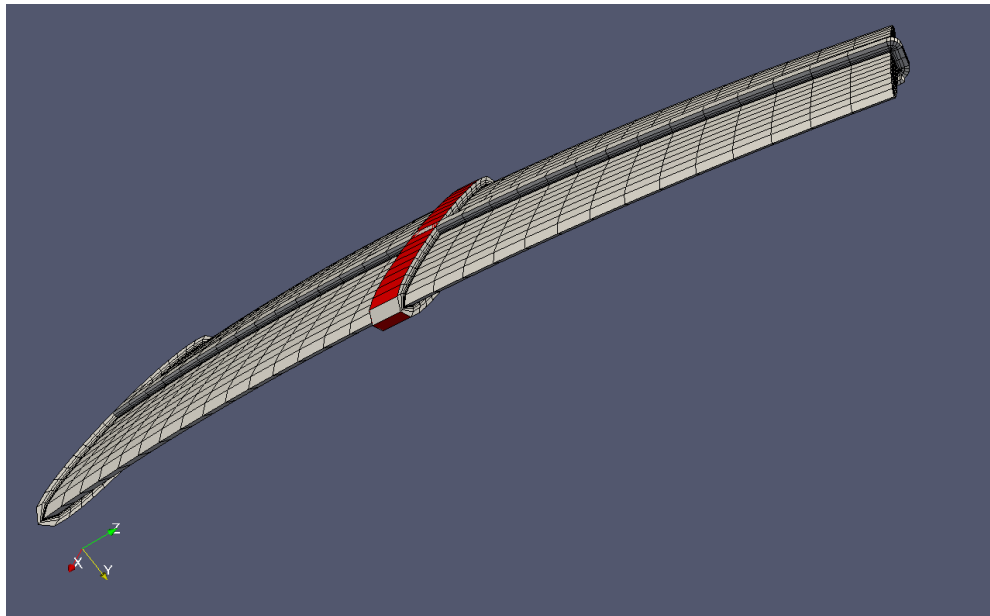


Figure 3. Slices of the grid around the bended wing

After the creation of the structured mesh around the wing as seen above the next step is to generate the surrounding mesh with cfMesh and another issue that arises is the file format the cfMesh can handle. CfMesh requires a closed STL as an input, together with the mesh settings. Salome supports the functionality of exporting the mesh as an STL file. For the creation of the domain including the wing CAESES is used, with the outer boundary of the bended wing's mesh imported as an STL into CAESES, boundaries for the domain are created and all are exported in STL format using the trimesh operations available in CAESES. The boundaries are placed randomly since at this point the stitching and merging of the meshes is examined.

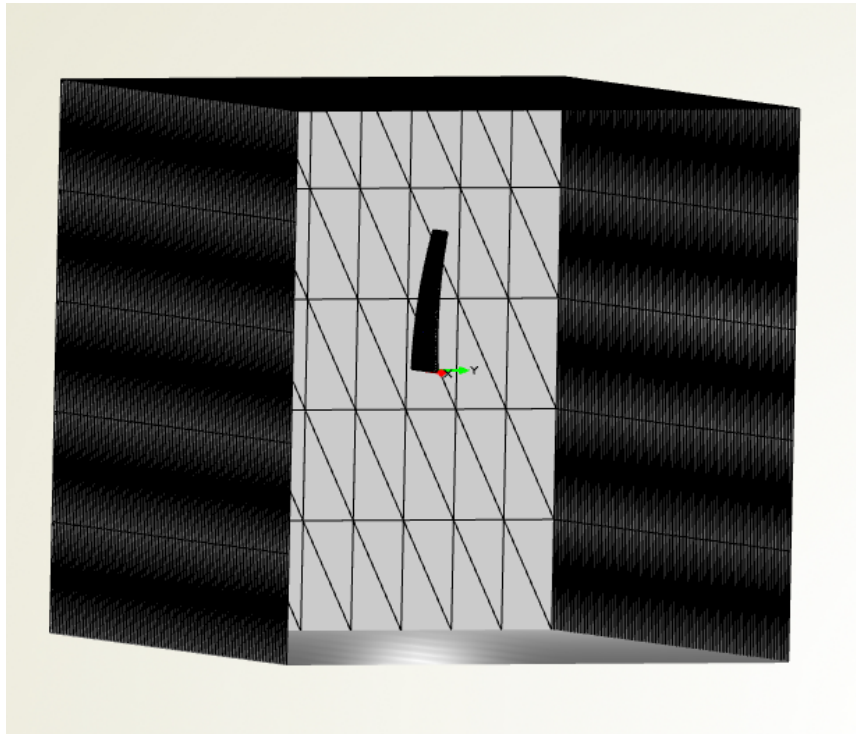


Figure 4. Domain generation

With the mesh for the domain created with cfMesh, the two meshes are merged with the mergeMesh command successfully. However, all the efforts that have been made in order to stitch the two meshes were unsuccessful due to the mismatch of the surface meshes to be coupled. The underlying reason seems to be the fact that cfMesh creates its own surface mesh and does not conform to the given STL. For the stitching of the two patches, the surface mesh corresponding to the wing is specified in cfMesh, and the cell size can be controlled locally. Different cell sizes are tried for the wing area as seen below, however the discontinuities and gaps at the interface of the two grids could not be avoided. In the three figures below, one can see the two patches, grey with black edges for SALOME and red with blue edges for cfMesh.

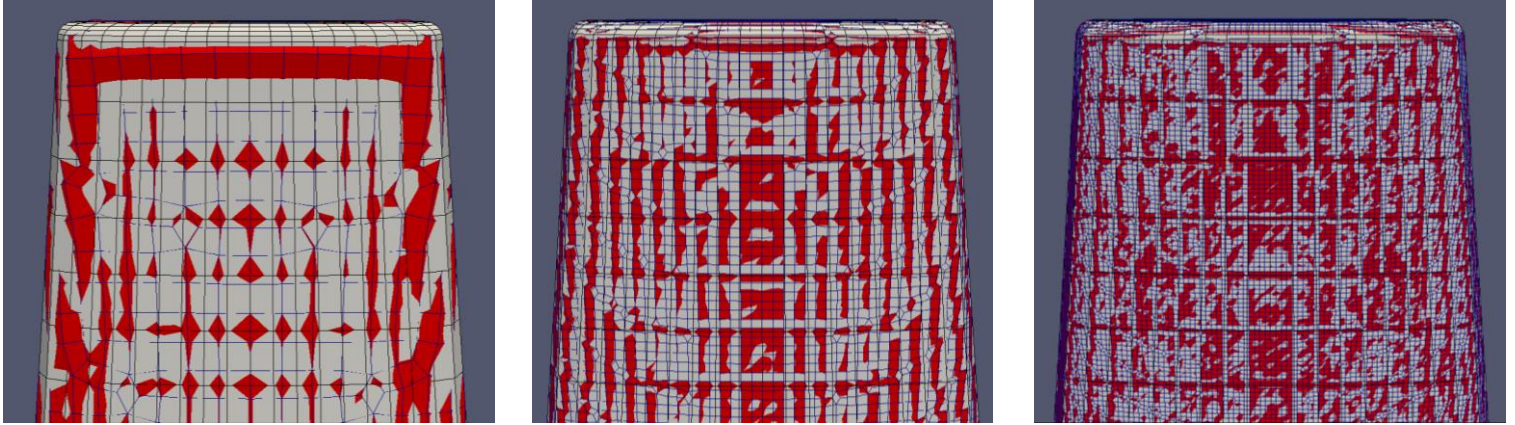


Figure 5. Alterations on the mesh resolution to increase interface matching

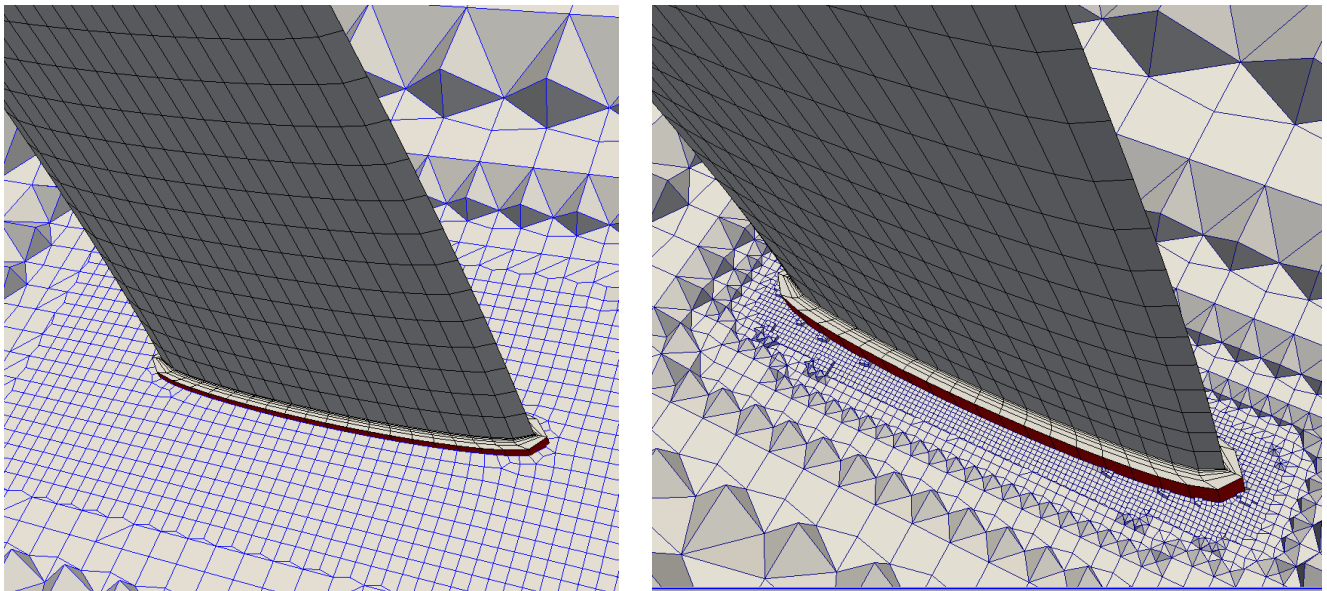


Figure 6. Alterations on the mesh resolution 3D view

Appendix B

The finite volume schemes and solver settings used are presented below.

```

/*-----*- C++ -*-----*\
|=====|
|| / F i e l d | OpenFOAM: The Open Source CFD Toolbox |
|| / O p e r a t i o n | Version: 4.1 |
|| / A n d | Web: www.OpenFOAM.org |
|| \ V M a n i p u l a t i o n | |
|*-----*/
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    location "system";
    object fvSolution;
}
// ***** //
solvers
{
    p
    {
        solver GAMG;
        smoother GaussSeidel;
        tolerance 1e-05;
        relTol 0.01;
    }
    U
    {
        solver smoothSolver;
        smoother GaussSeidel;
        nSweeps 2;
        tolerance 1e-08;
        relTol 0.1;
    }
    "(k|omega)"
    {
        solver smoothSolver;
        smoother GaussSeidel;
        tolerance 1e-6;
        relTol 0.1;
    }
}
SIMPLE
{
    nNonOrthogonalCorrectors 3;
    consistent yes;
    residualControl
    {
        p 1e-4;
        U 1e-4;
        "(k|omega)" 1e-4;
    }
}
relaxationFactors
{
    p 0.7;
    U 0.7;
    "(k|omega)" 0.7;
}
// ***** //

```

```

/*----- C++ -----*/
=====
|                               |
\\ / F i e l d | cfMesh: A library for mesh generation |
\\ / O p e r a t i o n |                               |
\\ / A n d | Author: Franjo Juretic |
\\ / M a n i p u l a t i o n | E-mail: franjo.juretic@c-fields.com |
/*-----*/

FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    object fvSchemes;
}

// *****

ddtSchemes
{
    default steadyState;
}

gradSchemes
{
    default Gauss linear;
}

divSchemes
{
    default none;
    div(phi,U) bounded Gauss limitedLinearV 1;
    div(phi,k) bounded Gauss limitedLinear 1;
    div(phi,omega) bounded Gauss limitedLinear 1;
    div((nuEff*dev2(T(grad(U)))) Gauss linear;
}

laplacianSchemes
{
    default Gauss linear corrected;
}

interpolationSchemes
{

```

```
        default          linear;
    }
    snGradSchemes
    {
        default          corrected;
    }
    fluxRequired
    {
        default          no;
        p;
    }
    wallDist
    {
        methodmeshWave;
    }
    // ***** //
```