University of Southampton

Faculty of Engineering and Physical Sciences
Department of Civil, Maritime and Environmental Engineering
Maritime Engineering Research Group

An Efficient and Accurate Overset Grid Technique Applied to Maritime CFD

by

Sébastien Lemaire
PhD
ORCiD: 0000-0002-9959-2100

A thesis for the degree of
Doctor of Philosophy

March 2023

University of Southampton

Abstract

Faculty of Engineering and Physical Sciences
Department of Civil, Maritime and Environmental Engineering

Doctor of Philosophy

An Efficient and Accurate Overset Grid Technique Applied to Maritime CFD

by Sébastien Lemaire

Ship maneuvering and other maritime applications pose complex challenges involving the motion of multiple bodies. The overset method enables simulations of such cases using computational fluid dynamics (CFD) by allowing arbitrary movement of any number of meshes. However, this versatility comes at a cost in terms of performance and accuracy. This research, therefore, aims to provide a modern, efficient and accurate overset method and study its reliability using advanced Verification techniques. To this end, a novel overset method that includes a wide variety of interpolation schemes has been implemented and is the basis for quantitative and qualitative error analysis studies performed on several test cases, as well as an investigation on computational performance.

Quantifying local and global errors, convergence orders, and mass imbalance for different interpolation schemes on several test cases allows to draw guidelines for both overset method users and developers. Moreover, the range of test cases used, such as a realistic 3D unsteady RANS manufactured solution of a recirculation bubble or a rudder-propeller flow, allows to extend the conclusions to many maritime applications. The results show that the choice of interpolation scheme has a significant impact on accuracy. 1^{st} order schemes lower the overall convergence order of the discretisation error, increasing the amount of artefacts visible in the field and resulting in larger high-frequency temporal oscillations. Higher order schemes such as 2^{nd} and 3^{rd} order accurate ones help to limit the error production and maintain 2^{nd} order accuracy of the solver's discretisation scheme. However, the limitations of 1^{st} order schemes do not come with a significant reduction in computational overhead. In fact, the 2^{nd} order *Nearest cell gradient* is even cheaper than the 1^{st} order *Inverse distance* scheme. Finally, the 3^{rd} order *Least squares* scheme, though more expensive, is still a viable option as it shows only a 8% performance overhead after several sequential and parallel programming performance tuning operations. In addition to providing guidelines and analysis methodologies, this work has also produced two opensource tools aimed at helping Verification, with the automatic generation of manufactured solutions and the computation of statistical uncertainties.

Contents

List of Figures

List of Tables

List of Algorithms

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;

7. Parts of this work have been published as:

   - Sébastien Lemaire, Guilherme Vaz, and Stephen R. Turnock. Implementation and Verification of an Explicit Overset Grid Method. In *21st Numerical Towing Tank Symposium (NuTTS)*, Cortona, Italy, 2018
   - Sébastien Lemaire, Guilherme Vaz, and Stephen R. Turnock. On the Need for Higher Order Interpolation with Overset Grid Methods. In *22nd Numerical Towing Tank Symposium (NuTTS)*, Tomar, Portugal, 2019
   - Tiago Gomes, Sébastien Lemaire, and Guilherme Vaz. Code and Solution Verification of Sliding and Overset Grid Methods on Wind Turbine Flows. In *ASME 41th International Conference on Ocean, Offshore and Arctic Engineering*, Hamburg, Germany, 2022
   - Sébastien Lemaire, Guilherme Vaz, Menno Deij-van Rijswijk, and Stephen R. Turnock. On the Accuracy, Robustness, and Performance of High Order Interpolation Schemes for the Overset Method on Unstructured Grids. *International Journal for Numerical Methods in Fluids*, 94(2):152–187, feb 2022. ISSN 0271-2091. doi: 10.1002/fld.5050
   - Sébastien Lemaire, Guilherme Vaz, Menno Deij-van Rijswijk, and Stephen R. Turnock. Influence of Interpolation Scheme on the Accuracy of Overset Method for Computing Rudder-Propeller Interaction. *Journal of Verification, Validation and Uncertainty Quantification*, 8(1), mar 2023. ISSN 2377-2158. doi: 10.1115/1.4056681

Signed:..... Date: March 2023

Acknowledgements

First and foremost, I would like to thank my three supervisors. I am very grateful for their continuous help and guidance that shaped this work and allowed me to complete this thesis:
Dr. Guilherme Vaz: Many thanks for introducing me to CFD seven years ago and for trusting me ever since. Thank you also for your support inside and outside work and for sharing your knowledge and vision that largely transpired in this thesis.
Prof. Stephen Turnock: Thank you for trusting me with this project, for your support, guidance and writing feedback throughout this work. Your insights on the propeller-rudder experiments were also very valuable.
Dr. Menno Deij - van Rijswijk: Thank you very much for always looking out for my well-being and having my best interest at heart. I also greatly appreciated your availability and for maintaining my connection with MARIN when I moved to Portugal.

I am very grateful for the fact that this thesis allowed me to work with a wide variety of teams and I would like to thank each of them for welcoming me. Chronologically starting with the NGCM team and the FSI department at the University of Southampton in the UK, continuing with the R&D department of MARIN in the Netherlands, and finishing with both WavEC and blueOASIS in Portugal. In particular I would like to thank António Maximiano, Bénédicte Dommergues, Benoît LeBlanc, Eduardo Lima, Gem Rotte, João Muralha, Manuel Rentschler, Raphael Madureira, Rui Lopes, Soren Schenke, Sophia Schillai and Stefano Levato. With special thanks going to Maarten Klapwijk, Chiara Wielgosz and Artur Lidtke. Our weekly meetings and chats were always very insightful and often a welcomed distraction. Artur Lidtke deserves a special mention for his help and expertise with the mesh generation of the propeller used in this work. Finally, I would also like to thank Tiago Gomes. Supervising you throughout your thesis helped mine in many ways, and it has been a pleasure to work with you.

Lastly, I must mention and thank my parents, brothers, and friends for their support and for always being there when I visit France.

À mon grand-père Jacques Lemaire

Nomenclature

Symbols

Ω	control volume	[–]
μ	dynamic viscosity	$[Pa.s]$
μ_t	turbulence eddy viscosity	$[Pa.s]$
ν	kinematic viscosity	$[J.s.kg^{-1}]$
ω	vorticity	$[s^{-2}]$
ρ	fluid density	$[kg.m^{-3}]$
C_D	drag coefficient	[–]
C_L	lift coefficient	[–]
C_{mult}	donor point multiplier	[–]
C_m	moment coefficient	[–]
C_{pc}	centre of pressure	[–]
C_p	pressure coefficient	[–]
F_k	polynomial basis function	[–]
J	propeller advance ratio	[–]
K_Q	torque coefficient	[–]
K_T	trust coefficient	[–]
N	number of *donor* cells	
N_{proc}	number of processes	
P/D	propeller blade pitch ratio	[–]
Q_{surf}	mass flux going through surface $surf$	$[kg.m^{-2}.s{-}1]$
Re	Reynolds number	[–]
U_∞	inlet velocity	$[m.s^{-1}]$
U_{discr}	space discretisation error uncertainty	
U_{exp}	experimental uncertainty	
U_{input}	input uncertainty	
U_{itr}	iterative error uncertainty	
U_{num}	numerical uncertainty	
U_{stat}	statistical uncertainty	
U_{time}	time discretisation error uncertainty	
c	chord length	$[m]$
k	turbulence kinetic energy	$[m^2.s^{-2}]$

m	mass	$[kg]$
p	pressure	$[Pa]$
s	span length	$[m]$
t	time	$[s]$
t_{step}	time-step	$[s]$
$\mathbf{v}$	velocity vector	$[m.s^{-1}]$
w	interpolation weight	[–]
y^+	non-dimensional wall-normal distance	[–]

Abbreviations

BEM	boundary element method
BPF	blade passing frequency
CFD	computational fluid dynamics
CFL	Courant-Friedrichs-Lewy number
DCI	domain connectivity information
DNS	direct numerical simulation
EPSRC	engineering and physical sciences research council
FSI	fluid structure interaction
GMRES	generalised minimal residual algorithm for solving nonsymmetric linear systems
HPC	high-performance computing
LGPL	GNU Lesser General Public License
MARIN	maritime research institute Netherlands
MMS	method of manufactured solutions
MPI	message passing interface
QUICK	quadratic upstream interpolation for convective kinematics
RANS	Reynolds-averaged Navier-Stokes equations
RBF	radial basis functions
RPM	revolution per minute
ReFRESCO	reliable & fast RANS equations (solver for) ships (and) constructions offshore
SIMPLE	semi-implicit method for pressure linked equations
SPH	smoothed-particle hydrodynamics
TST	transient scanning technique
UKRI	UK research and innovation
VoF	volume of fluid
lhs	left hand side
rhs	right hand side

Chapter 1

Introduction

1.1 Origin of overset method

The challenge of designing modern ships that are capable of achieving complex manoeuvres or hold dynamic positions within an ocean environment requires a detailed understanding of the flow interactions between rotating propellers and moving control surfaces. Historically, model tests have been used to perform manoeuvring studies. With the increase of computational resources available to design groups and research institutes, CFD (computational fluid dynamics) is more and more used early on in the design process. To perform such simulations, the CFD methods used need to allow relative motion of bodies, accurately capture boundary layers and be efficient enough to perform the computations in a reasonable amount of time.

CFD solvers can be divided into two main families: Lagrangian and Eulerian. In most cases, Lagrangian methods are 'meshless' and Eulerian methods are 'meshed' methods. The Lagrangian approach models the fluid using particles and solves the fluid equations (e.g. the Navier-Stokes equations) for each particle. The most common Lagrangian method is called smoothed-particle hydrodynamics (SPH). With this approach, the motion of bodies such as ships, propellers, and rudders is straightforward to handle because there is no theoretical limitation on their movement. Moreover, simulation of sharp and accurate free surfaces, which is important in a maritime contexts, is one of the strength of these methods. They, however, have more difficulties when capturing viscous effects and are better suited for inertial ones such as sloshing [98, 112] or impacts problems [90]. Another drawback of Lagrangian methods is their performance; they are more costly to use and are best suited for simulations in confined spaces or for short simulation times, where their modelling strength can be fully utilised.

More conventional CFD uses the Eulerian approach, in which, instead of modelling the fluid as particles, the domain containing the fluid is discretised on a mesh. The Navier-Stokes equations are then solved for each cell of this mesh. This has a large performance

benefit, because, unlike particles that move with the flow and for which controlling their density locally is not easy [26], the density of cells of the mesh can be tuned in advance for the application. This means that the computational resources can be assigned to the locations of interest or locations which will improve the accuracy of the simulation the most. For example, for maritime applications, it is common to see meshes refined close to the free surface, in the propeller vicinity and wake, and in general on any boundary to accurately capture the boundary layer. This mesh structure also allows easier handling of modern highly parallel high-performance computing (HPC) resources as the domain can be split into smaller ones and the computation distributed evenly on each processing core. Amongst others, these are the main reasons why the Eulerian approach is the most widely used in the industry, and has applications in every aspect of maritime engineering [96]. The need for a mesh as a support for the equation solving, however, prevents easy relative motion of bodies as meshes are usually body fitted. Due to the high demand for complex computation and body motion, several methods have been designed to solve this shortcoming.

One of these methods is called deforming mesh. If an object needs to me moved, the mesh around it needs to follow it, and as the name suggests this method deforms the mesh to comply with the motion. This allows keeping a single mesh to perform the computation and maintain the mesh's topology which is good for performance and parallelisation reasons. It is, however, limited to small relative motion. While it can be enough to simulate a floating buoy [94, 102, 130] as shown in Figure 1.1, it cannot cope with more complex movements like a propeller rotating.

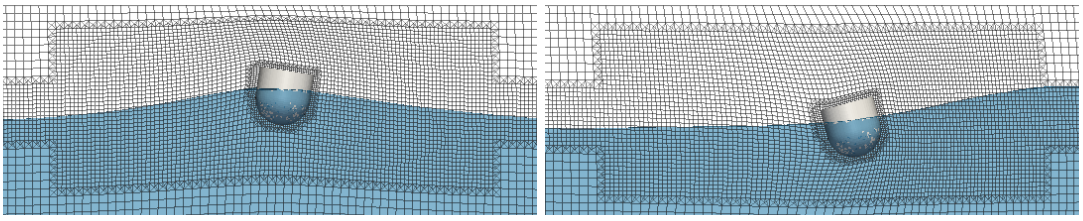


FIGURE 1.1: Deforming meshes used to simulate a floating buoy by Ransley et al. [102].

For such cases, the most commonly used method is sliding grids. For simulating a propeller, for example, instead of having only a single mesh to discretise the entire fluid domain, two meshes are used; a cylindrical one around the propeller and another one with a cylindrical hole in it where the propeller would sit. The cylindrical mesh can then rotate inside the hole of the other mesh. Similarly to single mesh methods, equations are solved for each cell of both meshes, the only difference lies in the interface between the two meshes. There, interpolation is performed to transfer information from one mesh to the other and the interpolated field data (like pressure, fluid velocity etc.) are treated as boundary conditions on the meshes. While this method can be used in many situations like rotating propellers [67, 99] as shown in Figure 1.2, airfoils changing their angle of attack [50] or even roll decay of ships [69], the motion is limited to translation or rotation

only and needs to be known *a priori*. Moreover the requirement for a fixed interface can also be a limiting factor as, for example, while both a rudder and a propeller only rely on rotations, they are often located too close to each other to accommodate both circular domains to be placed. Finally, the need for interpolation to transfer information adds another source of error that needs to be studied and kept under control. It should be noted that sliding and deforming grids can be used together in the same computation allowing for a wider range of motion to be simulated. Toxopeus and Bhawsinka [118] for example used them to compute the flow around a ship entering a lock as seen in Figure 1.3.

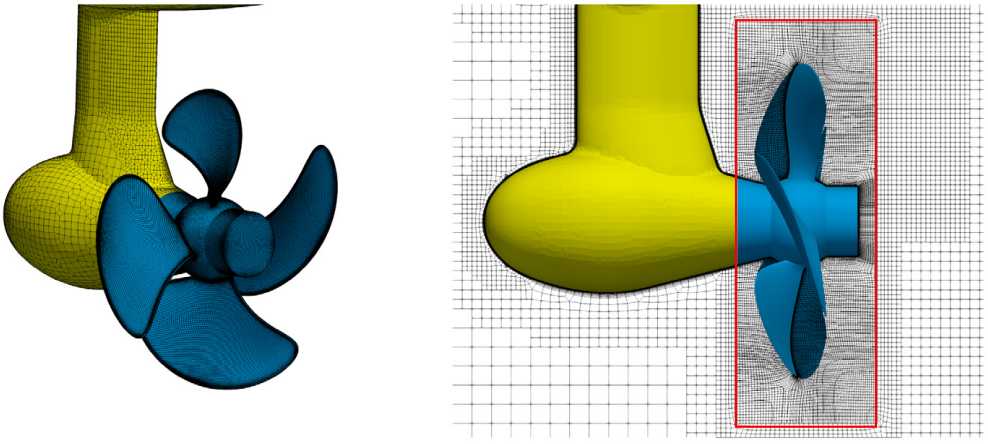


FIGURE 1.2: Propeller mesh rotating using sliding meshes by Lidtke et al. [67], the limit of the two meshes is marked in red.

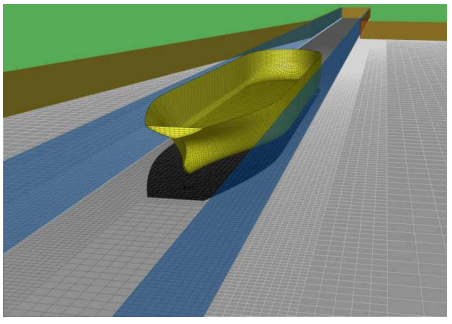


FIGURE 1.3: Sliding interface (in blue) used in combination with deforming meshes by Toxopeus and Bhawsinka [118] to simulate a ship entering a lock.

Finally, the overset method, also called chimera or overlapping grid method, overcomes the shortcomings of both the deforming and sliding grid methods by allowing arbitrary motion of any number of bodies while still permitting good quality body fitted meshes to be used. Like the sliding grid method, it relies on different meshes around each body but does not need a predefined interface between them. The different meshes are overlapped and the method disables cells that are under another mesh while using, like with sliding grids, interpolation to transfer field data at the fringe of each mesh. Because the interface cells (*fringe* cells) and disabled ones (*hole* cells) can be updated dynamically during the

computation, the method enables arbitrary motion of each mesh. As an example, Figure 1.4 shows a ship with rotating propeller, rudder, and six degrees of freedom on the hull itself. It uses 30 structured meshes in total. One can note that even meshes that do not move relative to each other are meshed with different grids, like the domain below and above the free surface, or the left and right part of the hull. This is another benefit of the overset method. Better mesh quality can be achieved by meshing parts of the domain independently, which is also particularly interesting on structured grid CFD solvers as it enables the meshing of complex geometries albeit at the cost of additional overset interfaces.

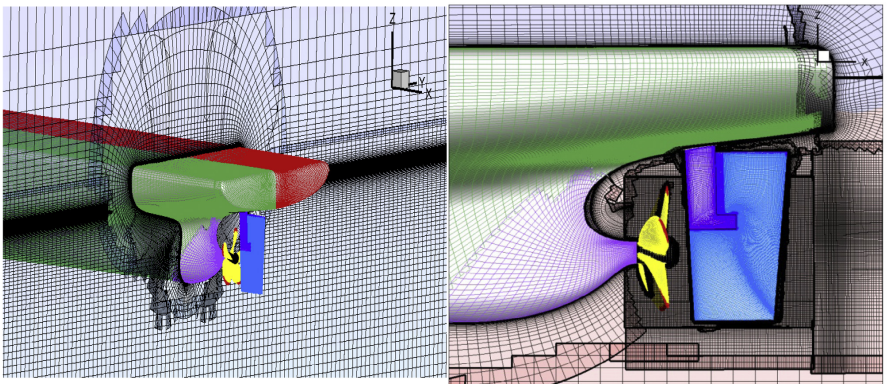


FIGURE 1.4: Overset grid assembly for a fully appended ship with a horn rudder from Mofidi and Carrica [77].

Historically, the overset mesh method was designed by NASA for aerospace application. Benek et al. [4, 5] were the first to publish a detailed description of the method and Dougherty et al. [31] applied it first to airfoils. At the time most CFD codes only used structured meshes and, like for the ship example presented above, the overset method was used primarily for its ability to mesh complex geometries even without motion.

In the early nineties, the first applications to the maritime field started. Cheng-Wen et al. [25] used the NASA code OVERFLOW to simulate a submarine sail and sail plane with an overset meshes to link the two meshes together. Even though no motion was used during the computation, the overset mesh approach allowed to simulate the sail plane at 10 and 20 degrees angle of attack without the need to generate different meshes. At Texas A&M University, HC Chen worked on various ship simulations requiring mesh motion like in Chen and Chen [24] where a ship approaching and leaving a harbour quaywall is simulated using two overset meshes, or Kang et al. [53] where prescribed ship motion is computed to analyse the wave generation. The two-thousands saw a diversification of overset implementations. Hadzic [47] wrote a thesis on the development of a 2D overset method and R Noack started Suggar [86], an external library to be integrated in already existing CFD solvers to add overset capability with the goal of being robust and requiring minimal work for its integration on the CFD solver. It lead to several successful integrations in CFDShip-IOWA [15] or OpenFOAM [7] and now ReFRESCO with the present work [63]. It is only after 2010 that all the major commercial CFD

solvers used nowadays implemented overset capabilities. STAR-CCM+ [111] introduced it in 2012 with version 7.02, FINE™/Marine [28] implementation started in 2016, same year as Ansys Fluent [23] with version R17. Finally, an open source version is also available in OpenFOAM since v1706, and foam-extend got it in 2018 [43]. Similarly to Suggar, several other overset libraries appeared, like OPERA [19] tailored towards OpenFOAM or TIOGA [107]. The latter is, contrary to Suggar or OPERA, opensource released under the LGPL license (GNU Lesser General Public License), allowing its integration for free in CFD solvers. All these implementations lead to the accessibility of the overset mesh method to a larger number of researchers and engineers.

This broader accessibility has been accompanied by an increase in overset method related publications as shown in Figure 1.5 displaying the number of overset papers in the google scholar database against their publication year. Since both 'overset' and 'chimera' nomenclatures are used interchangeably the graph plots the number of paper returned when searching for the keywords 'overset cfd' and 'chimera cfd'. Finally, to compensate for the overall increase in publication and other potential bias in the google scholar database the data presented is normalised by the overall number of publication present in the database each year following the methodology proposed in Orduna-Malea et al. [93]. From this graph, two trends can be distinguished: from 1980 to around 2010, a steady growth is observed while from 2010 onwards the number of paper increases more drastically. It can also be noted that the terminology 'chimera', even though used as much as 'overset' until 2000 appears to not be used as much afterwards and doesn't experience the same increasing trend as 'overset' does.

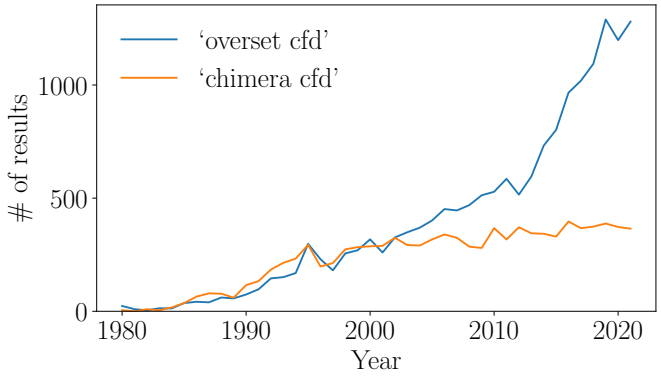


FIGURE 1.5: Number of papers published per year containing the keywords 'overset cfd' or 'chimera cfd'. Data extracted from google scholar database and normalised by the total number of papers per year in the database.

The democratisation of the tools means that the overset method is, nowadays, used for a wide variety of applications in the maritime field. From propeller rudder interactions [132] to fully appended ships in waves [17] or submarines [71] to assess maneuverability. Offshore structures and wind turbines also benefit from overset meshes with easier meshing and motion like in [56].

The overset method is, however, not exempt from limitations as it has an impact on the accuracy, performance and robustness of a computation. In such context, an accuracy loss means that a new error source is introduced in the field that overpowers existing ones. With the overset method, the accuracy loss is caused by the interpolation step needed to transfer field data like velocity or pressure from one mesh to the other. Several interpolation schemes can be used to achieve this, varying the number of *donor* cells required in the other mesh, the theoretical order of accuracy, the robustness or even the computational cost. As highlighted by Chandar et al. [23], the interpolation step is crucial to the overset method as it directly influences the solution. Low order schemes, for example, generating spurious oscillations on force coefficients. This step is however not often studied in detail. Most research either doesn't mention the interpolation method used [6, 17, 24, 114, 130] or mention the name of the method without sufficient details [126]. The lack of detail renders the reproducibility of the results difficult and makes the choice of appropriate scheme harder. It is only in very recent years that a limited number of more thorough research into the interpolation methods have been conducted. Chandar et al., for example, compared the interpolation methods of StarCCM+, AnSYS Fluent and OPERA in [23], although with a limited number of interpolation methods tested. Chandar [20] [21] investigate the interpolation methods available in OPERA (Inverse distance, and Least Squares Polynomial interpolation) on several test cases, including manufactured solutions, in order to assess the error levels and interpolation orders of each scheme. More recently, the library TIOGA was used to test several interpolation schemes with varying order of accuracy on different test cases in Sharma et al. [113]. This trend is, however, not large enough and shared only by a handful of research groups. In view of the large increase of overset usage and application range more thorough studies looking at the accuracy of the overset method are needed.

Besides solution accuracy, the overset method also degrades mass conservation. Indeed, one of the key feature of the finite volume method used by most Eulerian CFD solvers is its ability to conserve quantities, like mass, to a known level of accuracy (related to the residual level of the associated transport equation). This is achieved by equilibrating fluxes going in and out of each cell through its faces and is described in detail in Chapter 2. With the overset method, however, cells from different meshes do not share common faces anymore consequently eliminating the theoretical inherited mass conservation trait of the method. While mass defect is not often reported in the literature, it leads to unphysical pressure fluctuations as reported by Völkner et al. [126]. To reduce these effects and help accuracy, several methods were designed to enforce mass conservation in overset computations. For example, Hadzic [47] proposed a global method for conserving mass by modifying cells volume artificially to compensate for mass imbalance. It is however not using any local information, but globally adds a fraction of the total mass imbalance to all internal cells which limits it's physical correctness. Völkner et al. [126] proposed an improved method; instead of adding the total mass imbalance equally in the entire domain, each *fringe* cell receiving an interpolated value get a mass variation

based on the divergence of the velocity between the *donor* cells and the *fringe* cell. They show that their method manages to reduce the pressure fluctuations as well as the mass imbalance. Finally, Chandar [21] presents an alternative method based on fluxes correction within the iterative process. Even though these methods manage to overcome some of the consequences of not conserving mass at the overset interface they are not ubiquitous, likely because they are not always straightforward to implement, easy to generalise and can be computationally expensive. More recently, Chandar and Sitaraman [22], however, addresses these points by designing a new mass conservation method that does not add performance overhead while being easy to implement which may improve adoption in the future.

Finally, the overset method is known to lead to performance overhead. This is due to the computation of the interpolation itself as well as the computation of the domain connectivity (finding which cells are to be discarded etc.) requiring a lot of parallel communication between the different processes. Even though performance of the overset method is not often reported, Ohashi [91] show that 60% of the computation is dedicated to overset tasks, meaning that a similar computation without overset would be 2.5 times faster. For OpenFOAM, Gatin et al. [43] reports that a third of the computational time is dedicated to the overset method. Moreover, besides needing computational time to perform its own tasks, the overset method can also degrade the iterative convergence, hence needing more iterations to reach the same level of residuals. However, like for interpolation errors, performance and scalability of the different overset implementations are almost never reported.

Table 1.1 outlines the characteristics of the three mesh methods presented in this section. Even though it has some limitations, the overset method is the only one able to handle arbitrary motion of multiple meshes and hence multiple bodies.

To summarise, the main challenges to overcome in any overset implementation are to ensure the interpolation errors are minimised and predictable so that their influences on the solution are quantifiable as well as being computationally performant enough to be used in an engineering context.

TABLE 1.1: Summary of the characteristics of the different mesh methods presented.

Method	Motion handled	Limitation
Re-meshing	• Arbitrary motion of multiple meshes • Body deformation	• Accuracy loss (interpolation) • Extremely expensive • Hard to implement in a general way
Immersed boundary	• Arbitrary motion of multiple meshes • Body deformation	• No boundary layer • High frequency noise with motion
Deforming grids	• Small amplitude • Body deformation	• May worsen mesh quality
Sliding grids	• Predefined rotation and translation	• Accuracy loss (interpolation) • Loss of mass conservation
Overset grids	• Arbitrary motion of multiple meshes	• Accuracy loss (interpolation) • Loss of mass conservation • Decrease of robustness • Performance overhead

1.2 Project background

This PhD project is a collaboration between the University of Southampton in the UK and MARIN (Maritime Research Institute Netherlands) in the Netherlands and funded by MARIN and by the UKRI (UK Research and Innovation) more specifically the EPSRC (Engineering and Physical Sciences Research Council). Its original impulse was to add overset method capabilities to ReFRESCO, the finite volume CFD solver developed by MARIN and its partners. As such, time was spent in the University of Southampton, MARIN but also in WavEC and blueOASIS in Portugal, the later two being development partners of MARIN and ReFRESCO.

1.3 Aim & Objectives

The aim of the project is to develop an efficient and accurate overset method, particularly targeted towards maritime applications. As seen in the first section of this Chapter, the overset method brings many possibilities in terms of computation complexity or ease of mesh generation, but also comes with several challenges like performance overhead, accuracy loss or robustness issues. The developed method should then be made more *reliable* by quantifying and reducing the errors associated with it, and more *efficient* by minimising its computational cost. Detailed objectives are as follow:

- **Design and implement a novel overset method for unstructured grid solvers targeted to maritime applications**. This work should produce a detailed description of the overset method architecture chosen and justify the design choices made.
- **Assess the robustness, accuracy and efficiency of overset methods**. An overset method should be robust to various mesh assemblies and reliably maintain the accuracy of the underlying discretisation. Finally, this should be achieved with minimal performance overhead. When analysing errors, emphasise should be given to quantifying and qualifying error generation and propagations to better understand the influence of the overset method on the solution.
- **Draw guidelines on the usage and implementation of the overset method for maritime applications**. Overset capable solvers often implement several interpolation methods though the impact of this choice is not often known well enough. Guidelines should help users as well as advise overset method developers.
- **Help the research community by producing opensource tools to assist code Verification and uncertainty quantifications**. As every new implementation needs to perform error assessment and uncertainty quantifications, tools developed and used in the present research should be published to better help the community.
- **Push for higher error analysis standards in maritime CFD**. Verification and Validation should be done thoroughly to show how useful and efficient the methodologies are at helping research.

1.4 Publications

As part of this research, two journal papers and four conference papers were published or have been submitted:

- **Lemaire S.**, Vaz G., Turnock S. Implementation and Verification of an Explicit Overset Grid Method. 2018. *21st Numerical Towing Tank Symposium (NuTTS).*
- **Lemaire S.**, Vaz G., Turnock S. On the Need for Higher Order Interpolation with Overset Grid Methods. 2019. *22nd Numerical Towing Tank Symposium (NuTTS).*
- **Lemaire S.**, Vaz G, Deij-van Rijswijk M., Turnock S. On the Accuracy, Robustness, and Performance of High Order Interpolation Schemes for the Overset Method on Unstructured Grids. 2022. *International Journal for Numerical Methods in Fluids.* Volume 94, Issue 2. DOI: 10.1002/fld.5050

- Klapwijk M., **Lemaire S.**. And... Action! Setting the Scene for Accurate Visual CFD Comparisons Using Ray Tracing. 2021. *Journal of Marine Science and Engineering*. Volume 9, Issue 10: 1066. DOI: 10.3390/jmse9101066
- Gomes T., **Lemaire S.**, Vaz G., Lau F. Verification Study of Sliding and Overset Grid Methods using the Method of Manufactured Solutions on a Wind Turbine flow. 2021. *23rd Numerical Towing Tank Symposium (NuTTS)*.
- Gomes T., **Lemaire S.**, Vaz G. Code and Solution Verification of Sliding and Overset Grid Methods on Wind Turbine Flows. 2022. *ASME 41th International Conference on Ocean, Offshore and Arctic Engineering*.
- **Lemaire S.**, Vaz G., Deij-van Rijswijk M., Turnock S. Influence of Interpolation Scheme on the Accuracy of Overset Method for Computing Rudder-Propeller Interaction. 2023. *Journal of Verification, Validation and Uncertainty Quantification*. Volume 8, Issue 1: 011002. DOI: 10.1115/1.4056681

1.5 Novelty

This research brought several new aspects to the field of CFD and the study of overset meshes in particular, namely:

- The usage of Verification procedure and in particular of complex RANS manufactured solution for the analysis of overset mesh method errors.
- The thorough and systematic analysis of a wide range of overset interpolation schemes, from first to fourth order accurate.
- The quantification of absolute errors as well as uncertainties on integral quantities, mass fluxes imbalance, instantaneous and time averaged coefficients and error propagations on a variety of flows.
- The qualification of overset error sources and propagations patterns depending on the scheme and the flow characteristics.
- The release of two opensource tools to help Verification; PyMMS [65] for the generation of manufactured solutions, and PyTST [66] for the computation of statistical uncertainties. Since their release, both of them have been used in the literature, by Gomes et al. [44, 45] for PyMMS and by Lidtke et al. [67], Klapwijk et al. [59] and Wang et al. [128] for PyTST.
- The design and publication of the code architecture of a novel overset method.
- The publication of a set of guidelines on overset interpolation schemes taking into consideration their accuracy, robustness and performance [63, 64].

1.6 Structure of the Thesis

After this introduction chapter, Chapter 2 establishes the main theoretical foundation for the CFD computations used in this work. It presents the finite volume approach of solving Navier-Stokes and highlights the connections with the overset method. It also includes a description of the CFD solver used throughout this work.

Since the study of overset interpolation errors is critical to the accuracy quantification of an implementation, it is important to also study the other error sources appearing in CFD computation. To this end, Chapter 3 explains the principle of Verification, Validation as well as the different error sources appearing in any CFD computation. This Chapter then details the method used to perform code Verification on complex test cases: the method of manufactured solutions. Lastly, two opensource tools to first generate manufactured solutions and second to quantify statistical uncertainties are presented.

The decisions made during the implementation of the overset method directly affect the efficiency and accuracy of the computations. Chapter 4, then, explains in a literature review some design choices made by other CFD solvers and goes into further details with the implementation done as part of this work.

Next, the three following Chapters test the accuracy and robustness of the newly developed overset implementation with increasingly complex and realistic test cases. Chapter 5 assesses the implementation correctness of the different interpolation schemes in isolation, outside of any overset or CFD computation. Moreover, this Verification step allows to draw some preliminary conclusions on the robustness of some schemes. Then, Chapter 6 performs code Verification on actual overset CFD computation of a Poiseuille flow – a steady, 2D, low Reynolds number test case – followed by the study of a 3D unsteady high Reynolds number RANS manufactured solution representing a recirculation bubble. The Verification is enabled by the fact that, for both test cases, an exact analytical solution is known, hence allowing for the errors to be probed in the entire domain. This leads to a detailed understanding of their propagation and sources. Lastly, Chapter 7 studies the real life example of a rudder behind a propeller at a drift angle. This final test case permits to draw conclusions more directly applicable to overset method users helped with the knowledge gained by the previous Verification.

In each of the results chapters a wide range of interpolation schemes are tested, their difference in terms of robustness and accuracy are, however, not enough to draw useful guidelines as these need to be analysed in lights of their respective performance overhead. This is the reason why Chapter 8 studies the performance and parallel scalability of all the components composing any overset implementation, with a focus on the interpolation schemes themselves to help decision making.

Finally, Chapter 9 draw concluding remarks on the methodology developed in this work as well as precise guidelines for overset users and developers with a focus on maritime CFD.

One can note that literature reviews are performed throughout the thesis covering areas relevant for the discussion at hand.

Chapter 2

Fundamentals of finite volume CFD and overset method

Mesh based CFD solvers are all built upon common principles, they, however, do not exactly share the same methodologies. Some of them trade the versatility of unstructured meshes for the potential increased accuracy of structured meshes for example. Similarly, most solvers share the same overall solving mechanism without sharing exactly the specifics of their discretisation methods.

Instead of providing an exhaustive list of methods to build a CFD solver, which is available in the literature [40], this Chapter details all the core steps common to any solvers as well as some specific components particularly interesting for the overset method in general, the implementation done as part of this work or the test cases and methodologies that are part of this thesis.

This Chapter begins with details about the CFD solver used for this work with section 2.1. Then, section 2.2 presents the governing equations solved by CFD codes followed with section 2.3 detailing the methods used to numerically solve the previously mentioned governing equations. Next, section 2.4 shows the structures of CFD solvers and how equations are solved on HPC systems. Finally, section 2.5 gives an overview of the overset method and how it is integrated with a CFD solver.

2.1 Baseline CFD solver

The research conducted in this work is done on an existing CFD solver: ReFRESCO (REliable & Fast Rans Equations (solver for) Ships (and) Constructions Offshore) [122]. It is a research and commercial CFD code optimised, verified and validated for maritime applications. It is being developed, verified and validated at MARIN[1] in collaboration

[1] https://marin.nl

with several other organisations like IST (Instituto Superior Técnico in Lisbon, Portugal), the University of Southampton in the UK, TU Delft (Technical University of Delft, the Netherlands) or blueOASIS in Portugal.

ReFRESCO is a viscous-flow CFD code that solves multiphase (unsteady) incompressible flows using the Navier-Stokes equations, complemented with turbulence models, cavitation models and volume-fraction transport equations for different phases. The equations are discretised using a finite volume approach with cell centered collocated variables, in strong conservation form, and a pressure correction equation based on the SIMPLE algorithm is used to ensure mass conservation [58]. Time integration is performed implicitly with 1^{st} or 2^{nd} order backward schemes. At each implicit time-step, the non-linear system for velocity and pressure is linearised with Picard's method and either a segregated or coupled approach is used. In the latter, the coupled linear system is using a SIMPLE-type preconditioner. A segregated approach is always adopted for the solution of all other transport equations. The implementation is face based, which permits grids with elements consisting of an arbitrary number of faces (hexahedrals, tetrahedrals, prisms, pyramids, etc.), and if needed h-refinement (hanging nodes).

Moving, sliding and deforming grids, as well automatic grid adaptation were available before the current work and coupling with structural equations-of-motion (rigid-body 6DOF), and flexible-body FSI is possible. Code parallelisation is done using MPI (Message Passing Interface) and domain decomposition computed by ParMETIS [110]. The core of the code is mainly implemented in free form Fortran 2003 with some module taking advantage of Fortran 2008 and 2016 standards.

2.2 Governing equations

This section presents the governing equations that dictate fluid behaviours, the notation employed follows the one used in Ferziger and Peric [40].

2.2.1 Navier-Stokes

Fluid dynamics is based on conservation principles, stating that mass, momentum or energy are conserved over time. In this section they are all applied to 'control volumes' which is a spacial region of the domain. First, the conservation of mass also called continuity equation can be written as:

$$\frac{\partial}{\partial t}\int_{\Omega}\rho d\Omega + \int_{S}\rho \mathbf{v}\cdot\mathbf{n}dS = 0. \tag{2.1}$$

It can be interpreted as follow: for a control volume Ω, the evolution of its mass over time (first term) is equal to the mass fluxes that go through the control volume surface S (second term). Hence at any given time, in order to conserve mass, for any control volume Ω, this equation has to be satisfied.

Then, the conservation of momentum is the second fundamental equation of CFD, it is derived from Newton's second law of motion: $\frac{dm\mathbf{v}}{dt} = \sum \mathbf{f}$, and applied to a control volume Ω results in:

$$\frac{\partial}{\partial t}\int_{\Omega} \rho \mathbf{v} d\Omega + \int_{S} \rho \mathbf{v}\mathbf{v} \cdot \mathbf{n} dS = \sum \mathbf{f}. \tag{2.2}$$

The right hand side is constituted by surface forces acting on the control volume like pressure, normal and shear stresses etc. or volume forces like gravity. It can be shown that these forces can be written as:

$$\sum \mathbf{f} = \int_{S} \mathbf{T} \cdot \mathbf{n} dS + \int_{\Omega} \rho \mathbf{b} d\Omega. \tag{2.3}$$

With $\mathbf{b}$ representing the volume forces per unit mass and $\mathbf{T}$ defining the surface forces due to pressure and stresses, it is the molecular rate of the transport of momentum. By assuming that the fluid is Newtonian, T can be written as:

$$\mathbf{T} = -\left(p + \frac{2}{3}\mu \; div \; \mathbf{v}\right)\mathbf{I} + 2\nu\mathbf{D}, \tag{2.4}$$

$$\mathbf{D} = \frac{1}{2}\left[grad \; \mathbf{v} + (grad \; \mathbf{v})^T\right], \tag{2.5}$$

with p the static pressure, ν the dynamic viscosity, $\mathbf{I}$ the unit tensor and $\mathbf{D}$ is the strain rate tensor.

The equations governing the conservation of mass and the conservation of momentum are called the Navier-Stokes equations.

2.2.2 General transport equation

Besides mass and momentum, the concept of conservation can be applied to other scalar quantities, and, in a similar fashion, a generic transport equation for a quantity ϕ can be derived:

$$\frac{\partial}{\partial t}\int_{\Omega} \rho \phi d\Omega + \int_{S} \rho \phi \mathbf{v} \cdot \mathbf{n} dS = \sum f_{\phi}. \tag{2.6}$$

f_ϕ represents the transport of ϕ besides convection, this includes its production or destruction as well as its diffusion. Since diffusion is always present, a generic definition of f_ϕ^d is as follows, using a gradient approximation:

$$f_\phi^d = \int_S \Gamma grad\phi \cdot \mathbf{n} dS. \tag{2.7}$$

With Γ the diffusivity of ϕ.

2.2.3 RANS approach to turbulence modelling

Turbulent flows are very common in engineering applications and in maritime ones in particular as they appear in wakes of hydrofoils, ships, propellers or offshore structures. Turbulence itself is characterised by being a highly unsteady three dimensional flow behaviour, often used as an example for chaotic phenomena. However, it presents coherent structures and a large range of time and length scales. Since fully resolving turbulence is not yet a realistic approach for engineering problems due to the computational resources required, modelling turbulence partially or entirely is needed.

Instead, because instantaneous fluctuating quantities are not always needed in engineering context, the RANS (Reynolds Averaged Navier-Stokes) approach simplifies the equations by considering time averaged and ensemble averaged quantities instead. Since the complexity of turbulence lies in these unsteadinesses removing them simplifies the requirements in terms of mesh densities and time-steps, hence decreasing the computational cost. When introducing 'Reynolds averages' in the Navier-Stokes equations however, new terms appear that require modelling to be handled. The different RANS turbulence models then differ from one another by the way and complexity these new terms are coped with.

In detail, the RANS approach decomposes any quantity as the sum of a time averaged or ensemble averaged one and a fluctuating one:

$$\phi(x,t) = \overline{\phi}(x) + \phi'(x,t) \tag{2.8}$$

with $\overline{\phi}(x)$ a time average in case of a steady flow, or an ensemble average (also called 'Reynolds average') for an unsteady flow, and $\phi'(x,t)$ the fluctuating part. For example, it follows that $\overline{\phi'} = 0$.

By replacing each term by the sum of its averaged and fluctuating components in the Navier-Stokes equations presented in the previous section, and removing null terms, the following equations are found for incompressible flows without body forces in Cartesian coordinates:

$$\frac{\partial(\rho\overline{u}_i)}{\partial x_i} = 0, \tag{2.9}$$

$$\frac{\partial(\rho\overline{u}_i)}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho\overline{u}_i\overline{u}_j + \rho\overline{u_i'u_j'}\right) = -\frac{\partial\overline{p}}{\partial x_i} + \frac{\partial\overline{\tau}_{ij}}{\partial x_j}, \tag{2.10}$$

With $\overline{\tau}_{ij}$ the mean viscous stress tensor components:

$$\overline{\tau}_{ij} = \mu\left(\frac{\partial\overline{u}_i}{\partial x_j} + \frac{\partial\overline{u}_j}{\partial x_i}\right). \tag{2.11}$$

And for a generic scalar field ϕ:

$$\frac{\partial(\rho\overline{\phi})}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho\overline{u}_j\overline{\phi} + \rho\overline{u_j'\phi'}\right) = \frac{\partial}{\partial x_j}\left(\Gamma\frac{\partial\overline{\phi}}{\partial x_j}\right). \tag{2.12}$$

In these equations, the Reynolds stresses, $\rho\overline{u_i'u_j'}$, and turbulent scalar flux, $\rho\overline{u_i'\phi'}$, appear and cannot be expressed by mean quantities or simplified. Therefore, a model is needed to close the system of equations. One common approach is to use the Boussinesq hypothesis, which introduces the following relation to approximate the Reynolds stresses:

$$-\rho\overline{u_i'u_j'} = \mu_t\left(\frac{\partial\overline{u}_i}{\partial x_j} + \frac{\partial\overline{u}_j}{\partial x_i}\right) - \frac{2}{3}\rho\delta_{ij}k. \tag{2.13}$$

This new equation, however, introduces two new quantities, the eddy viscosity, μ_t, and turbulent kinetic energy, k, both requiring transport equations to be solved. The way these new quantities are modeled is where most RANS turbulence models differ. One equation models like the Spalart-Allmaras [117] and Menter [73] both focus on the eddy viscosity ignoring the kinetic energy term. Then two equations models like the $k-\omega$ SST from 2003 [74] or $k-\sqrt{k}l$ [75] models for example introduce two transport equations from which μ_t and k can then be deduced. Because approximations are made with the use of models, each model has a range of applications for which it is designed and has been validated and calibrated against.

2.3 Numerical methods

The governing equations presented in the previous section are partial differential equations for which exact solutions are most of the time unknown. Solving them numerically is then the only possible option, and it relies on both space and time discretisation. To achieve this, several methods can be employed. As seen in Chapter 1 a mesh based

Eulerian approach is used in this work, more particularly space discretisation follows the finite volume method.

2.3.1 Finite volume method

The finite volume approach discretises the solution domain into control volumes or cells forming a mesh. As seen in Figure 2.1, cells do not overlap and each cell face is shared by two cells except on domain boundaries. That way, each point of the domain is inside one and only one cell. In the method presented here, and used for this study, the cell centres hold a computational node where each quantity is recorded (like p, $\mathbf{v}$ etc.), though other implementations may choose cell vertices to assume this role.

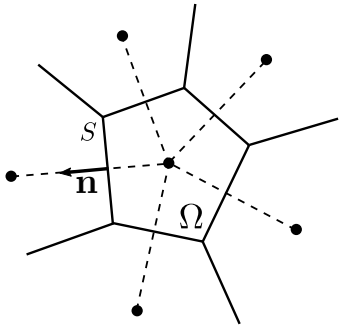


FIGURE 2.1: Example of cell acting as a control volume with the finite volume approach.

Since each cell defines a control volume, the equations presented in the previous section have to hold true for every single one of them. Taking as an example the integral form of the conservation of a quantity ϕ for a steady flow:

$$\int_S \rho\phi\mathbf{v}\cdot\mathbf{n}dS = \int_S \Gamma grad(\phi)\cdot\mathbf{n}dS + \int_\Omega q_\phi d\Omega. \tag{2.14}$$

With S the surface area of the control volume, $\mathbf{n}$ the control volume normal pointing outwards and Ω defining the control volume. In order to solve this equation each term has to be analytically computed which is done by approximating the integrals.

The integral approximation can be done in many different ways and has an influence on both the accuracy of the solution and the robustness of the solving mechanism. Assuming the cell centre value is known, the simplest way to compute a volume integral is to multiply the cell centre value by the cell's volume:

$$\int_\Omega q d\Omega \approx q_p \Delta\Omega, \tag{2.15}$$

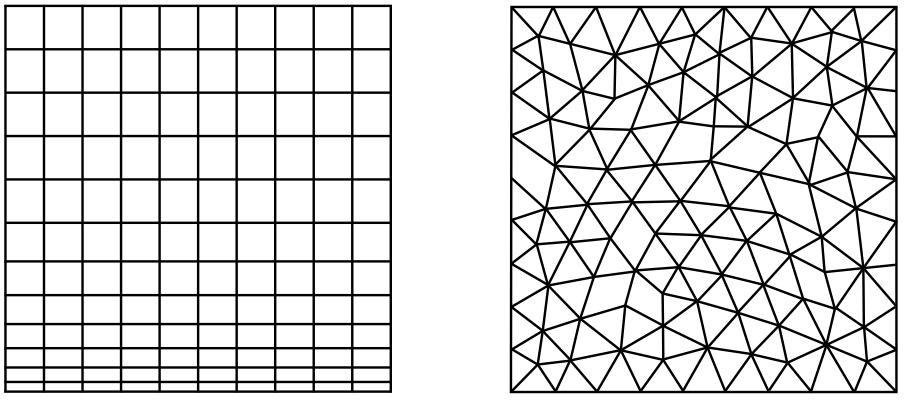
where q_p is the value of q at the cell centre and $\Delta\Omega$ is the volume of the cell. Similarly, surface integrals can be approximated using each face surface area and the value of the

quantity of interest at its centre:

$$\int_S f dS \approx \sum_{faces} f_c S_f, \tag{2.16}$$

with f_c the value of f at the face centres and S_f the face's area. Unlike with the volume integral, however, face centre's values are usually not known and need to be reconstructed from the neighbouring cell centre's values. Once again, a wide range of schemes exist to perform face centre reconstructions and detailing them is out of the scope of this Chapter.

As stated in Chapter 1, meshes can be structured or unstructured, and this has an influence on the scheme that can be used. Figure 2.2 shows an example of both types, in 2D. Structured meshes enforce each interior cell to have exactly four neighbouring cells (and six in 3D). On the other hand, unstructured meshes do not have any formal limitations on the number of faces or neighbours per cell, hence allowing for a greater versatility in the mesh topology and easier mesh generation. By having a fixed number of neighbours per cell and such cell organisation, structured meshes can, however, benefit from more powerful schemes. This is mainly because, unlike with unstructured meshes, each cell's second, third, etc. layer of neighbours are trivially easy to gather and use without storing any extra connectivity information. For this reason, on unstructured meshes (and associated code), the accuracy of the surface integrals and face centre are either 1^{st} or 2^{nd} order as higher order would require more knowledge than the direct neighbours for each cell. 1^{st} order schemes, albeit being less accurate, are still used for their better robustness.



(A) Structured grid (B) Unstructured grid

FIGURE 2.2: Example of 2D structured and unstructured meshes.

2.3.2 Gradient computation

Either because it directly appears in the transport equation or because it is needed to apply corrections (like cell excentricity or non-orthogonality) and approximate integrals,

the gradient of any quantity is needed when solving numerically the Navier-Stokes equations. One common way to compute them is to rely on the Gauss's theorem which links the surface integral of a quantity and the volume integral of its gradient:

$$\int_{\Omega} \mathbf{grad(f)}\ d\Omega = \int_{S} f\mathbf{n}dS \tag{2.17}$$

with $\mathbf{n}$ the outward normal facing normal and f the quantity of interest. Taking the same assumptions regarding volume and surface integrals as stated in the previous sections results in:

$$\mathbf{grad(f)} \approx \frac{1}{\Delta\Omega} \sum_{faces} f_c \mathbf{n}S. \tag{2.18}$$

Then, if the face centre and volume integral approximations are 2^{nd} order accurate, the resulting gradient computation is also 2^{nd} order accurate. Once again, several alternative methods exist for computing the gradient of a quantity in a finite volume context.

2.3.3 Time discretisation

Similarly to space, for unsteady flows, time also needs to be discretised to solve Navier-Stokes equations. This is usually done using a single, constant, time-step (t_{step}) throughout the entire computation, and, each step, equations are solved taking into account previous ones. In this work, the 'Three time level Method' is used and is presented in this section. It is a fully implicit scheme meaning that, to compute time-step $n+1$, fluxes and other quantities are also evaluated at time-step $n+1$, contrary to explicit schemes for which the solution at time-step $n+1$ directly depends on quantities computed at time-step n. The implicit formulation allows the scheme to be unconditionally stable regardless of the time-step size (t_{step}) picked, allowing for larger time-steps to be used.

For example, when taking the differential form of the transport equation for a generic quantity ϕ:

$$\frac{\partial \rho\phi}{\partial t} = -div(\rho\phi\mathbf{v}) + div(\Gamma grad(\phi)), \tag{2.19}$$

the three time level method would result in the following discretisation:

$$\rho\frac{3\phi^{n+1} - 4\phi^{n} + \phi^{n-1}}{2t_{step}} = -div(\rho\phi^{n+1}\mathbf{v}) + div(\Gamma grad(\phi^{n+1})), \tag{2.20}$$

and, by re-arranging the terms:

$$\rho\frac{3}{2t_{step}}\phi^{n+1} + div(\rho\phi^{n+1}\mathbf{v}) - div(\Gamma grad(\phi^{n+1})) = \frac{2\rho}{t_{step}}\phi^{n} - \frac{\rho}{2t_{step}}\phi^{n-1}. \tag{2.21}$$

With ϕ^n denoting the value of ϕ at time-step n. This means that, for this scheme, the value of ϕ of the current (n) and previous $(n-1)$ time iterations are needed to compute the next one $(n+1)$. Compared to an explicit scheme, this results in a larger storage requirement, though, besides being stable due to its implicit nature, this scheme is also 2^{nd} order accurate.

2.4 Resolution of equations and CFD code structure

Once all the equations are discretised in both time and space, as seen in the previous section, the system of equations can be solved. In this section, details are given regarding the solving mechanism, the general structure of a CFD solver and the challenges brought by parallel resolution on HPC systems.

2.4.1 Linear system of equations

For each cell of the domain, a set of coupled algebraic equations needs to be solved. Moreover, for each cell, its neighbouring cells centre values appear in its own equation, effectively coupling all the algebraic equations together.

For each quantity, a matrix and its accompanying right hand side vector are built. The matrix is square and has as dimension the number of cells in the domain. For example, when considering cell i having two neighbouring cells j and k, the equation for the quantity ϕ to be solved can be written as follow:

$$\alpha_i \phi_i^{n+1} + \alpha_j \phi_j^{n+1} + \alpha_k \phi_k^{n+1} = \beta_n \phi_i^n + \beta_{n-1} \phi_i^{n-1} + q. \tag{2.22}$$

Where α_i, α_j, α_k, β_n, β_{n-1} and q are coefficients that include a dependency to $\mathbf{v}$, p, etc. If a product of ϕ was present in the equation, a linearisation step is needed and it would place one of the ϕ component inside the associated α or β coefficient. For each quantity, the system of equation can be assembled into a matrix equation as shows here for the quantity ϕ:

$$\begin{pmatrix} \ddots & & & \\ & & & \\ & & & \\ \alpha_j & \alpha_k & \alpha_i & \ddots \end{pmatrix} \begin{pmatrix} \phi_j^{n+1} \\ \phi_k^{n+1} \\ \\ \phi_i^{n+1} \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ \\ \beta_n \phi_i^n + \beta_{n-1} \phi_i^{n-1} + q \\ \\ \end{pmatrix}, \tag{2.23}$$

$$A\phi^{n+1} = b. \tag{2.24}$$

It can be noted that, because each cell has a limited number of neighbours, matrix A is sparse, moreover it has non zero diagonal elements. For efficiency reasons, and because of the linearisation step, the system is solved iteratively using methods taking advantage of such shape. If linearisation was needed the α and β coefficients are updated with the latest ϕ value computed. At each step of the process, convergence can be monitored by computing the residual vector ϵ defined by:

$$\epsilon = A\phi - b. \tag{2.25}$$

This residual can then serve as a stopping criteria for the iterative resolution process. In practice, the iterative resolution also often includes a pre-conditioning step that, without changing the solution of the system, will modify A and b to improve robustness and convergence. In this work, the B-Jacobi (for block-Jacobi) pre-conditioner is used associated with GMRES (Generalised Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems) [109] method for solving all equations except the continuity one for which the conjugate gradient method is used instead.

2.4.2 General structure of CFD solvers

Solving each quantity individually does not couple them, the pressure, for example, depending on the velocity and *vice versa*. The coupling is done in a segregated way, solving each equation in turn. More precisely, the momentum and continuity equation, solving for the velocity and pressure correction, are solved first using, in this work, the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) [40] algorithm. Then any additional equation (for turbulence quantities for example) is solved subsequently. This process then loops until a convergence stopping criteria is reached before advancing to

the next time-step. This stopping criteria is either residual based, *i.e.* once each equation's residuals reaches a user-defined threshold the time-step advances, or directly after a set number of iteration has been performed. These set of loops are called outerloops.

Figure 2.3 summarises the structure of the different iterative processes involved in a CFD computation.

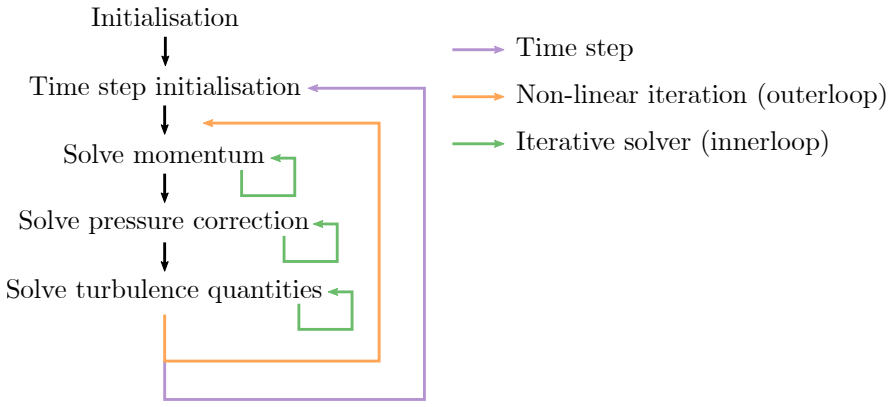


FIGURE 2.3: Summary of the different iterative processes appearing in a CFD computation.

2.4.3 HPC parallelisation

Modern HPC architectures use an aggregation of compute nodes, each of which is composed of one or multiple CPUs, each then with the capability of running multiple processes in parallel. CFD solvers need some adaptation in their resolution to accommodate for it. While such architecture allows great versatility in the raw performance that can be assigned to a computation by modifying the number of CPU cores used, it has some limitations. Namely, the memory is shared only amongst each node, and MPI (Message Passing Interface) is needed for inter-node communication. In order to optimise the available resources the load of the computation needs to be balanced evenly on each processing core. This is achieved with domain decomposition. Instead of solving the entire problem at once, the mesh supporting the computational domain is subdivided and each processing core is assigned a partition of it. Good load balancing is achieved when each processing core is assigned the same number of cells. To this end, partitioning algorithms are utilised, in this work this step is handled by the external library ParMETIS (Parallel Graph Partitioning and Fill-reducing Matrix Ordering) [110].

Effectively, with domain decomposition, a global matrix system as showed in equation 2.23 is never generated and solved, instead, each processing core assembles a matrix containing the cells that are part of their domain. Communication between the different sub-domains is achieved by adding 'ghost cells' at their border that duplicate the data located on a neighbouring sub-domain like presented in Figure 2.4. Because inter-node communication, which is at the core of the parallelisation, is an expensive step the partitioning algorithm, besides maintaining load balancing, also tries to minimise the

amount of ghost cells needed. After each innerloop, ghost cells are updated to account for the updated field of each domain. Further details can be found in Hawkes [48] and Hawkes et al. [49] as they provide state of the art overviews of CFD performance on HPC.

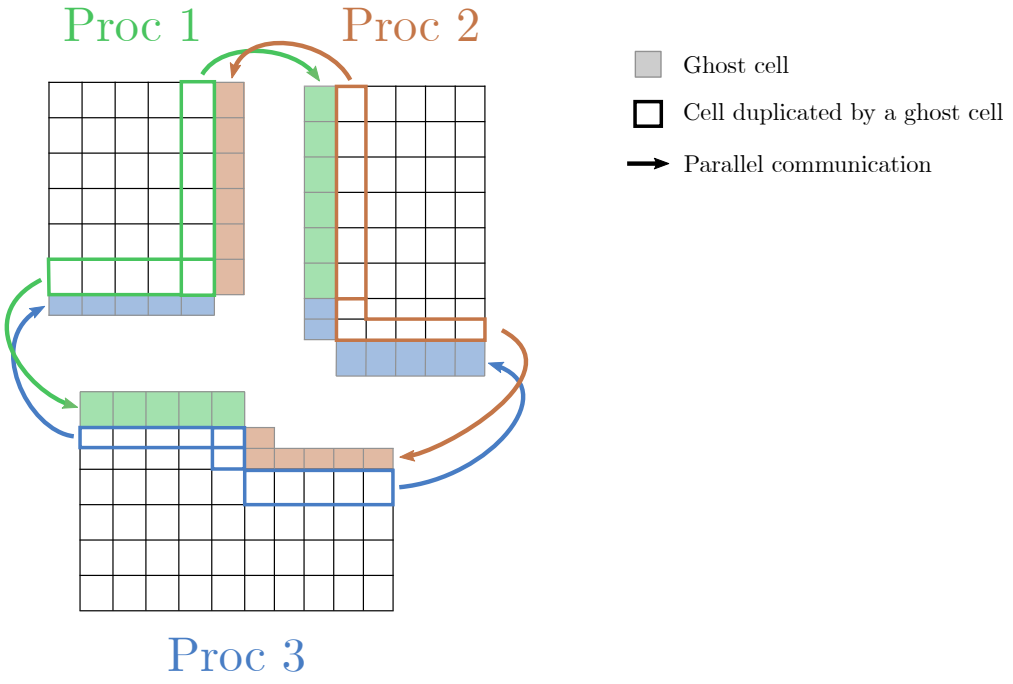


FIGURE 2.4: Parallel domain decomposition with three processes showing ghost cells and parallel communications.

2.5 The overset method

The overset grid method, also called chimera method, works by overlapping several meshes in a single computation allowing easier mesh generation and their arbitrary relative motion, allowing in turns relative motion of bodies. The method dynamically assigns status to particular cells to generate an interface where information can be exchanged between meshes. On most cells, the finite volume method is solved as presented in the previous sections, but, at the fringe of each mesh, cells receive interpolated information from another mesh to ensure the continuity of the domain. Due to the overlap, other cells need to be disabled. Overall, each cell in the domain is assigned a status with the following nomenclature:

- *Hole* cell: a field cell that is outside the boundary of the domain. It can come from a grid entirely embedded in another one (the background grid will have *hole* cells), or from a grid being partially outside the boundary of the domain.
- *Fringe* cell (also called receptor cell in the literature): a *fringe* cell is a cell adjacent to a hole or at the boundary of an embedded grid. It will act as boundary cell

for its grid and get its field value from the interpolation of *donor* cells of another mesh.

- *Orphan* cell: an *orphan* cell is a *fringe* cell that does not have enough *donor* cells to compute its interpolated value. *Orphan* cells exist when grids don't have enough overlap or when the position of *hole* cells was not correctly computed. As a fallback, its field value is usually then computed as the average of its direct neighbouring cells.

- *In* cell: this is a traditional cell, every cell that is not a *hole*, a *fringe* or an *orphan* cell will be an *in* cell. The finite volume method is solved conventionally without any special treatment related to the overset grid assembly.

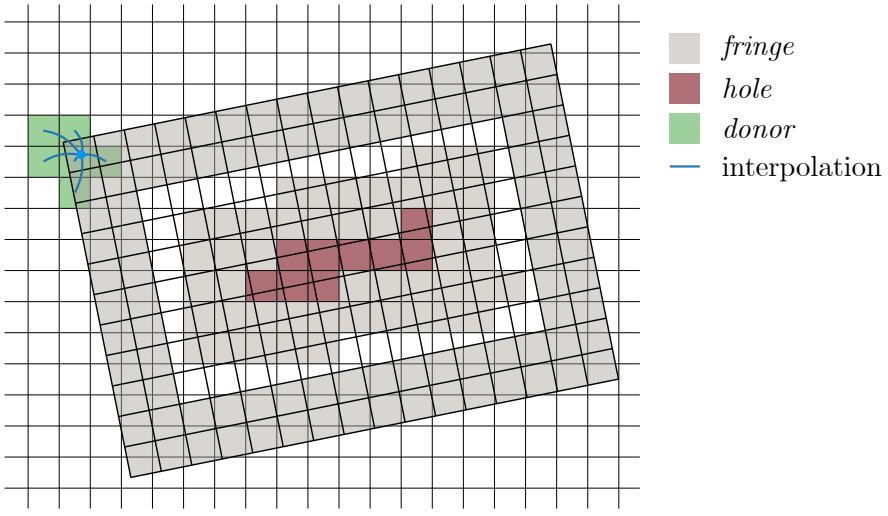


FIGURE 2.5: Example of overset assembly with its nomenclature. *Donor* cells for the top left *fringe* cell are displayed.

Figure 2.5 shows an example of a mesh assembly, with two cartesian meshes overlapped. The background mesh contains *hole* cells as well as two layers of *fringe* cells around it to act as the boundary and the foreground one also has two layers of *fringe* cells along its boundary. Having two layers allows for the gradient to be computed accurately as the *fringe* cells directly next to *in* cells are surrounded by either *in* or *fringe* cells. Finally, an example of a set of *donor* cells is presented with the, here, six cells used to compute the interpolated value of the top left *fringe* cell. Any *in* cell can be a *donor* cell and a *donor* cell can be used for several *fringe* cells interpolations. The DCI (domain connectivity information) is the name given to the specification of *fringe*, *hole* and *orphan* cells, sometimes associated with interpolation weights and *donor* cells. It is computed only based on the geometry and topology of the different grids using a 'hole cutting' algorithm and hence needs to be recomputed each time there is grid motion, which is usually every time-step. Details about how cells status are computed is out of the scope of this work but detailed algorithms can be found in the literature [19, 86, 88, 107]. Finally, different interpolation schemes used in an overset context are presented in Chapter 4 section 4.4.

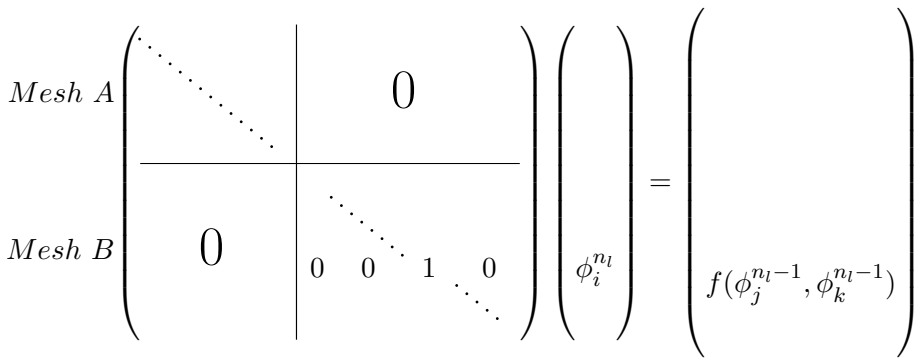


FIGURE 2.6: Explicit formulation of an overset coupling with two meshes (A and B). ϕ is the quantity to be solved, n_l the current outer iteration, i the index of a *fringe* cell from $Mesh\ B$ and j and k its associated *donor* cells from $Mesh\ A$. Finally f is the overset interpolation function.

To incorporate the cell status information and interpolated data inside the solver and perform the actual coupling between meshes two approaches can be used; either explicitly or implicitly. In both cases, it involves modifying the matrix system of equation from equation 2.23 as, for example, *fringe* cells do not depend on their neighbours anymore.

In an explicit form, the interpolated values are placed on the right hand side of the system of equations and a unitary diagonal is used for *fringe* cells. This allows to have minimal changes on the system of equations and matrices. The right hand side interpolation is then updated every outerloop from the new data of the *donor* cells. This coupling is presented in Figure 2.6, for the example using two meshes where cell i is a *fringe* cell of $Mesh\ B$, cells j and k are its associated *donor* cells belonging to $Mesh\ A$ and f the interpolation function. If cells belonging to $Mesh\ A$ and $Mesh\ B$ are ordered successively like it is the case here, because they do not share any 'neighbours' the top right and bottom left corners of the system are filled with zeros. In this formulation, the interpolation is lagging one outerloop (n_l being the current outerloop index here) which can slow down the iterative convergence. This has the benefits of large versatility in the interpolation function f. Moreover, the structure of the matrix itself is not modified by the method, when a cell becomes a *fringe* one (due to a mesh motion for example) the modifications only affect the diagonal, right hand side and the locations associated with the neighbours of the *fringe* (that need to be changed to zeros).

Contrary to an explicit coupling, with an implicit one, the interpolation is not computed directly, instead interpolation weights are placed on the left hand side of the system as shown with Figure 2.7. With this method, the interpolation has to be a linear combination of the *donor* cells' values, and is defined here by: $\phi_i^{n_l} = w_j\phi_j^{n_l} + w_k\phi_k^{n_l}$, with i the *fringe* cell index, j and k the *donor* cells ones and w_j and w_k the associated interpolation weights. Because updated information is used for the interpolation, this coupling improves the iterative convergence of the resolution. At the cost, however, of a more complex implementation – the matrix layout being modified –, and more expensive innerloops. This added computational cost comes from having to update *donor* cell data

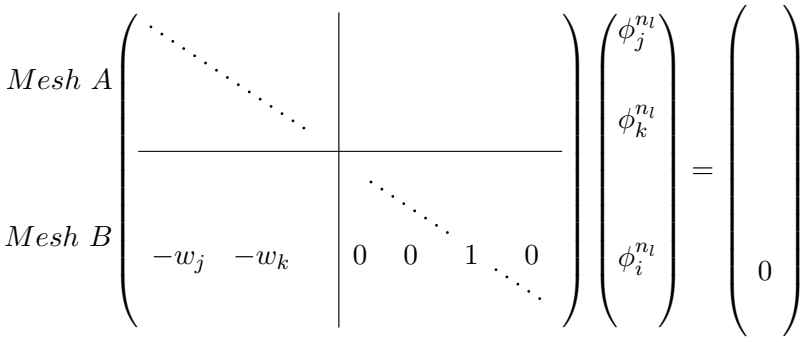


FIGURE 2.7: Implicit formulation of an overset coupling with two meshes (A and B). ϕ is the quantity to be solved, n_l the current outer iteration, i the index of a *fringe* cell from $Mesh\ B$ and j and k its associated *donor* cells from $Mesh\ A$ with w_j and w_k the interpolation weights. The interpolation is then defined by $\phi_i = w_j\phi_j + w_k\phi_k$.

within innerloops using parallel communications whenever the *fringe* and *donor* cells are not stored and solved on the same processing core. In terms of accuracy, however, when iteratively solved to the same residual levels, the two methods should give the same solution within the margin of the iterative error.

Finally, the two approaches can be used together as some interpolation schemes may have a linear component that would be implicitly handled (by modifying the matrix) and non-linear ones that will be placed to the right hand side. Moreover, solver may choose to handle some equations with implicit formulations and others explicitly to improve iterative convergence only for the most challenging ones and benefit from the performance gain for the other ones.

To summarise, Figure 2.8 shows the structure of a CFD solver with an explicit overset implementation when mesh motion is performed every time-step. Details concerning implementation decisions and interpolation methods are given in Chapter 4.

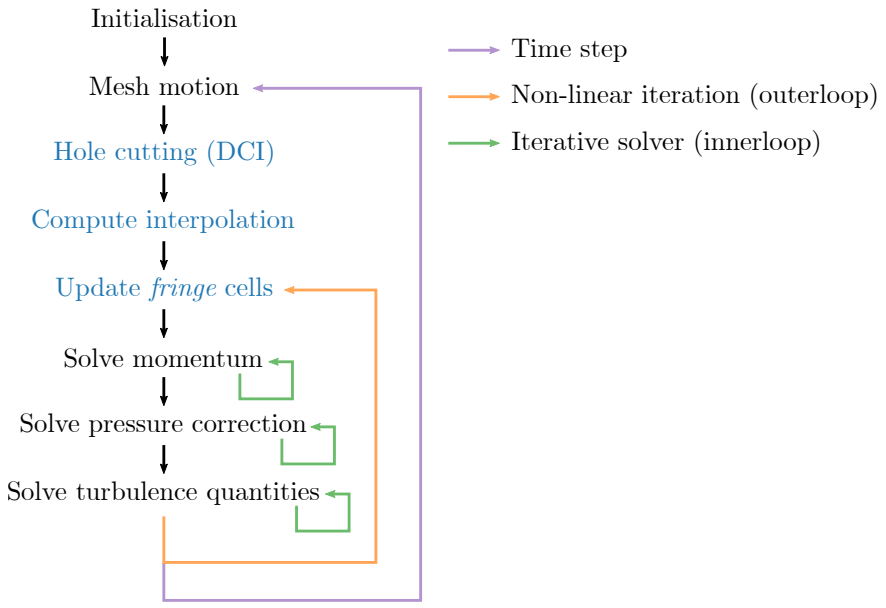


FIGURE 2.8: Solution workflow with explicit overset coupling. Steps marked in blue are specific to an overset computation.

2.6 Summary

As seen in this Chapter, CFD solvers are complex pieces of software with many different components that interact with each other. For example, the construction of the resolution matrix is influenced by the time and space discretisation schemes, the mesh topology, parallelisation, and overset connectivity information. Additionally, all the different components must cope with domain decomposition and, if possible, rely only on local information rather than integral quantities. In the process of developing a CFD solver from the continuous Navier-Stokes equations, several approximations are made. The next Chapter is dedicated to the analysis of these approximations and the errors associated with them.

Chapter 3

Verification and Validation

Any numerical simulation comes with assumptions and approximations that lead to inaccuracies affecting the produced results. In order to ensure the reliability of a computation it is essential to study the error sources and their importance. Following Roache [104] nomenclature an error is the signed difference between the computed solution and an exact value. That exact solution might, however, not always exist, if so, uncertainties can still be computed and used as they define an interval where the exact value should be within a certain degree of confidence (usually 95%).

Verification is a purely mathematical study where numerical errors or uncertainties are quantified. It does not depend on any experimental data but when possible an analytical exact solution might be used to compute errors, this is called code Verification. The quantification of error uncertainties without the knowledge of an exact values is called solution Verification. Validation, on the other hand, requires experimental data to compare to as it is meant to quantify how physically accurate a computation is. Since numerical errors are always present, Verification should always be performed prior to any Validation study. In this Chapter, the different error sources appearing in a CFD computation are introduced together with methods designed to quantify errors and uncertainties. Finally, two open source tools developed to perform Verification studies in this work are presented.

3.1 Error sources

3.1.1 Round off error

Round off errors relate to the number of digits or resolution available when manipulating numbers due to the finite representation of numbers by computers. This type of error cannot be measured as it would require infinite precision to reach the 'exact' solution, and

therefore estimating the round off uncertainty is then only possible by running several computations using different precisions (like single, double or quadruple precision) and estimating the error. The single, double etc. precision refers to the number of bits used to encode a float number in the computer memory. A single precision float, for example, uses 32-bit when double precision one uses 64-bit.

As it is commonly accepted, for practical CFD applications, double precision arithmetic errors should be suitably small to be negligible compared to other sources [34]. Hence, in this work, double precision is used and the round off errors are not further investigated. It should be noted, however, that as meshes get finer and higher order interpolations are needed, double precision inaccuracies might start to become predominant [33].

3.1.2 Iterative error

As seen in Chapter 2, in CFD, iterative methods are used both in linear resolution of the transport equations (innerloops) and, in segregated implementations, within non-linear steps (outerloops). Both these iterative processes contribute to the iterative errors. In theory they could be reduced to machine accuracy and reach the round off error. In practice, however, the complexity of the problems prevents such convergence. Either equations cannot be converged that low or it would require a number of iterative steps too high making the computation not worth the accuracy gain. The iterative process has to be stopped once a 'satisfactory' level of error is achieved. From Eça and Hoekstra [33], the iterative error needs to be at least two or three orders of magnitude smaller than the discretisation error to be considered negligible.

To quantify the iterative error, a computation reaching residuals of the order of the round off error is needed. Because it is not practical for most engineering problems, methods like the one presented in Eça et al. [39] are used. By running computations varying the number of iterative steps (or of residuals), iterative uncertainty can be estimated. Figure 3.1 shows the method described by Eça et al. [39] in which the quantity of interest (here K_T) is plotted against residual levels which are here controlled by the number of outerloop (n_{loop}) per time-step. From the error modelling an uncertainty estimation can be computed for each residual level and is presented with the vertical bars. This example is detailed in Chapter 7.

It should be noted that, even though there is a relation between the residual levels and the iterative error, there is no quantitative law linking the two quantities. From a single computation, the only conclusions that can be drawn are qualitative: lowering residuals leads to a decrease in the iterative error.

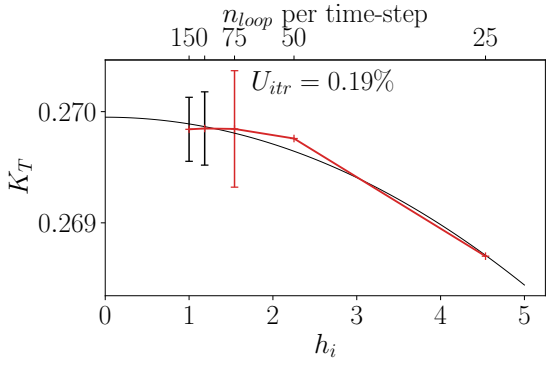


FIGURE 3.1: Integral quantity convergence with residual decrease (h_i being a metric of the residuals level) and uncertainty computed as per Eça et al. [39] methodology.

3.1.3 Discretisation error

The discretisation of partial differential equations in space and time is an approximation to their continuous nature and the source of the discretisation errors. Its study is of particular importance as it is usually (or should be) the predominant error source in CFD computations. Space discretisation is governed by two aspects: the grid quality which include cell sizes, their shape and the mesh topology (presence of hanging nodes or highly skewed cells etc.) as well as the schemes used for the discretisation. In unsteady computations, time discretisation is, similarly, governed by the time-step size and the time discretisation scheme. In order to estimate it, systematical grid and time-step refinements are often used, and, as an example, the method proposed by Eça and Hoekstra [34] models the error using a truncated power series expansion. For space discretisation the error is estimated by:

$$e_i = \alpha h_i^p + o(h_i^{p+1}), \tag{3.1}$$

with α a constant, p the observed order of convergence, and h_i the typical cell size. This model assumes that the set of grids used are in the asymptotic range to justify the use of power series expansion and that the grid's density can be represented by a single parameter (the typical cell size). In practice, this translates to having geometrically similar grids for which the grid refinement ratio is constant and cells from different meshes have the same orthogonality, skewness, etc. On such meshes, h_i can be defined by: $h_i = 1/N_i^{1/3}$ in 3D or $h_i = 1/\sqrt{N_i}$ in 2D, with N_i the number of cells in grid i.

By neglecting higher order terms, the order of convergence of two consecutively refined grid is measured by:

$$\begin{aligned} \frac{e_i}{e_{i-1}} &= \frac{\alpha(h_i)^p}{\alpha(h_{i-1})^p} \\ \frac{e_i}{e_{i-1}} &= \left(\frac{h_i}{h_{i-1}}\right)^p \\ \log\left(\frac{e_i}{e_{i-1}}\right) &= p\log\left(\frac{h_i}{h_{i-1}}\right) \\ p &= \frac{log(\frac{e_i}{e_{i-1}})}{log(\frac{h_i}{h_{i-1}})} \end{aligned} \tag{3.2}$$

If higher order terms can be neglected, the order p can be computed with any two grids with different refinement. In practice, convergence studies are done with more than two grids, and a least squares approach is used to compute the observed order of convergence. Moreover, modelling the error using only equation 3.1 can be limiting as real data may include noise and the fit not good enough to use the model. To this end, Eça et al. [37] suggest additional fit functions of order below two to be used when discretisation schemes are at most 2^{nd} order. The methodology presented by Eça et al. [37] is used to estimate the discretisation error, the observed order of convergence but also an estimation of the uncertainty as an interval that contains the exact solution with 95% confidence. Even though this section mainly mentioned space discretisation, a similar procedure can be adopted with time discretisation by varying the time-step instead of the typical cell size.

In this work, on cases where an exact solution is known, the exact discretisation error can be computed directly and will be analysed, however, when such solution is not known error estimations as well as uncertainties will be used following Eça and Hoekstra [33], Eça et al. [37] methodology.

3.1.4 Overset interpolation errors

In an overset computation, the different meshes exchange information using interpolation. Depending on the interpolation scheme used, the error generated varies and can then propagate away from the overset interface. In itself, the overset interpolation error should be part of the discretisation error, however, in this work, a distinction will be made between the error generated by interpolation on *fringe* cells (called interpolation error) and the space discretisation error common to any CFD computation related to the use of cells in the rest of the domain.

3.1.5 Statistical error

Because of the finite physical time of a computation, any time averaging is affected by statistical errors. The longer the time averaging window, the smaller the statistical error. However, the limit is often the available computational resources and simulation wall clock time. Quantifying the error itself would need knowledge of the time average over an infinite long period of time, hence only the statistical uncertainty can be computed. In this work the method proposed by Brouwer et al. [11, 13] is used. It employs auto-covariance and computes, from any time signal, a 95% confidence interval band around the computed time average.

Any CFD computation is affected by startup effects, which are due to the unideal nature of the initial conditions prescribed that pollute the first time-steps of the simulation. When time averaging is needed, the signal has to be monotonous and this transient portion discarded. Using its statistical uncertainty estimation, Brouwer et al. [12] developed a method to detect this transient portion called the Transient Scanning Technique (TST). It works by computing the uncertainty on longer and longer portions of the entire signal always including its end and displaying the successive uncertainties on a log-log plot against the signal length. It is illustrated in Figure 3.2, where on the top a time signal with an initial transient is present, and on the bottom, the TST analysis. The -1 slope highlights a monotonous portion and the sudden rise of the uncertainty (called hockey stick) corresponds to the initial transient that should be discarded.

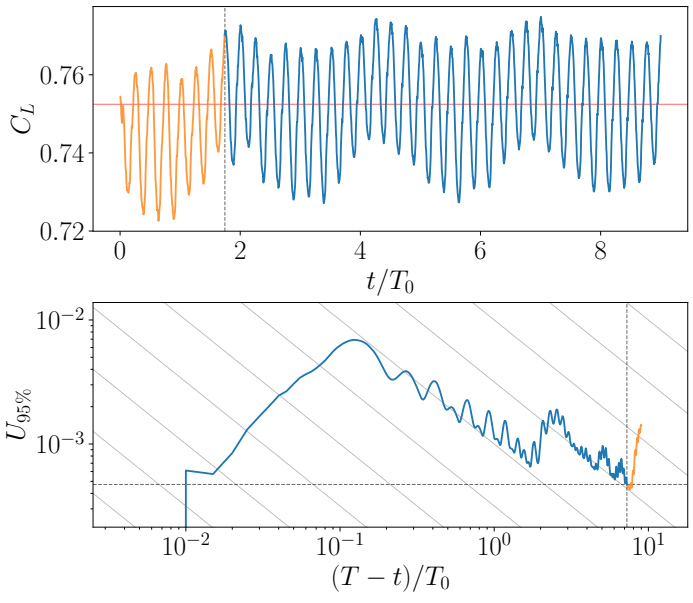


FIGURE 3.2: Example of statistical uncertainty analysis performed using the Transient Scanning Technique by [12] on a lift coefficient signal over time.

3.1.6 Additional errors

Finally, additional error sources can be found:

- Programming errors: these come from bugs in the implementation of the CFD solver itself and may affect the produced solution. Such errors can be checked using code Verification. In this work it is performed in Chapters 6 using analytical and manufactured solutions.
- Input errors: when two sets of data are produced in different ways (simulation and experiments for example), some uncertainty is present in the input given to the two sets. For example, the inlet velocity of a wind tunnel might have been measured with its own uncertainty, hence, when replicating the setup in a CFD computation, the inlet velocity cannot exactly replicate the experimental one. This can be quantified using methods presented in Katsuno et al. [54] that estimates the input uncertainty but is out of the scope of this thesis.

3.1.7 Modelling error

Modelling errors are due to physical approximations done when simplifying the models to be implemented often to save computational resources. In this work, for example, the RANS approach is used to perform turbulence modelling instead of solving it. Besides the model themselves used to simplify physical phenomena, boundary conditions are also approximated as physically accurate ones are not always obtainable.

Before analysing and assessing the modelling error via Validation, all the other numerical error sources need to be contained or quantified using Verification. Validation is then done by comparing computational results against either experimental data or more accurate computations like DNS (direct numerical simulation). Following the ASME V&V 20 standard [1] an procedure to estimate the interval that contains the modelling error can be devised. It compares $E = \phi_{CFD} - \phi_{exp}$, the difference between the CFD and experimental data for the quantity of interest ϕ with U_{val} the Validation uncertainty. The latter is made of three components: U_{num}, the estimated numerical uncertainty (from the CFD computation), U_{exp} the estimated experimental uncertainty and U_{input} the input uncertainty. [1] then proposes $U_{val} = \sqrt{U_{num}^2 + U_{input}^2 + U_{exp}^2}$. The complexity and potential cost of quantifying U_{input} means that it is often assumed to be null [33]. From these two quantities, different conclusions can be drawn:

- if $|E| \geq U_{val}$: the modelling error is likely of similar magnitude to E, and, if it is acceptably low for the practical purpose of the application, the model can be accepted and considered good enough to capture the physics for the analysed quantity.

- if $|E| \leq U_{val}$: if U_{val} is large no conclusion concerning the modelling error can be drawn and the Validation uncertainty needs to be reduced (with improved experiments or simulations). Otherwise, if U_{val} is small enough, then Validation can be considered acceptable.

It should be noted that these conclusions mention 'acceptably low' quantities. These need to be assessed in lights of the application and the precision required.

3.2 Method of Manufactured Solutions

The method of manufactured solutions or MMS is used to design (or manufacture) test cases. The main benefit of this method is that an exact solution field is known for each quantity (velocity, pressure etc.), hence the numerical errors can be probed in the entire domain without modelling errors and without requiring estimations. On computations that have negligible iterative and round off errors, the MMS allows direct access to the discretisation error in the entire domain or, in an overset context, the sum of both interpolation and discretisation errors.

MMS are traditionally used for code Verification [103, 105]: by checking that the convergence order of the discretisation error is matching the discretisation scheme order, one verifies that programming errors are, at most, also decreasing with the same order. Gomes et al. [45] for example, designed a solution resembling a wind turbine flow to assess interpolation errors when using either a sliding grid method or the overset method. Lovato et al. [68] used the method for non-newtonian fluid code Verification of multiphase flows, allowing to highlight spurious velocities appearing at the free surface when using the VOF (Volume of Fluid) method. Finally, Eça et al. [37] designed a RANS turbulent manufactured solution of a recirculating bubble in a boundary layer also to perform code Verification. This last solution is used in Chapter 6 to perform error analysis with overset meshes. Effectively, several code Verification studies have been done on ReFRESCO using the method of manufactured solutions [33, 35–38, 44, 45, 68] which give trust and credit to the tool on which the overset method is build upon.

The MMS works by adding a specific source term to each equation. Since the manufactured fields are 'user defined' they do not verify the equations being solved. The source term is needed to restore the equation balance. In practice, the design and use of a Manufactured Solution is done in three steps:

- First, an analytical definition of each field quantity (velocity, pressure, eddy viscosity, etc.) needs to be defined. In theory, the MMS is a purely mathematical exercise, so the fields do not have to be physically realistic. However, testing the models outside of their usual conditions can potentially lead to unexpected results

[35]. Moreover, approaching real phenomena often results in stronger conclusions. Therefore, fields are usually designed to be not too far from realistic physical behaviours.

- Next, each equation being solved (momentum, continuity, turbulence model etc.) needs to be analytically derived using the field quantities defined at the previous step. This process can be done manually (like in [35]), or using a symbolic mathematics framework like Sympy [76] or Wolfram Mathematica [52]. The imbalance in the equation is then equal to the source term.

- Finally, the source terms need to be added in the CFD solver system of equations, they are placed on the right hand side of the system. As they are only a function of space and time and, in particular, do not depend on the solution being computed such source terms can often be easily added via the usercoding capabilities of the CFD package.

By design, the MMS has some limitations though. Primarily, the analytical formulation of each quantity used in the model equations needs to be differentiable in space and time at least once, and twice for some quantities. Although some attempts were made to overcome this limitation [105], these methods are not ubiquitous. Nathan Woods and Starkey [83], for example, successfully designed an integral variant of the method of manufactured solutions to perform Verification of shock-capturing codes. Then, the models themselves need to be uniquely defined analytically. For example the use of limiters (like the $max()$ function) in some models is not compatible with the MMS, as these functions are not differentiable since they do not have a unique definition of the gradient. This was noticed by Eça et al. [35] when the CFD implementation of the $k-\omega$ SST model was not producing the expected convergence order.

Even though the MMS is not widely used for maritime applications, it is, for example, more common in aerospace engineering [9, 55, 84, 108, 123]. Its ability to perform code Verification on very complex and realistic flows makes it a powerful tool for precisely analysing errors and their behaviours.

3.3 PyMMS: an opensource framework for generating Manufactured Solutions

As described in the previous section, one of the key steps when generating a manufactured solution is the generation of the source terms. To this end a framework was developed and implemented in Python using the Sympy symbolic mathematics library [76]. As no openly available framework was found with the needed capabilities, this code was made opensource and libre under the MIT license in Lemaire [65].

The framework allows the user to provide the analytical formulation of field quantities like velocity, pressure, and eddy viscosity (when using a turbulence model). From them, it will compute the source terms analytically and produce a compilable Fortran source module file that contains the definition of each field quantity, its analytical derivatives and the source terms of each equation. This module can then be integrated easily as usercoding in the CFD solver. Besides the momentum and continuity equations, PyMMS includes the Spalart Allmaras turbulence model [117] (both the original formulation and the 'noft2' variant) as well as the one equation eddy viscosity model from Menter [73]. Additionally, its object oriented structure makes it easy to implement new models without having to modify the core of the code.

Since manufactured solutions are used to verify code implementation of other tools, their own code correctness is essential to asses error sources. PyMMS was verified by comparing its resulting source terms to another derivation of a similar MS done this time by hand. The test case described in [35] was replicated using Sympy and its source terms were generated with PyMMS. The original fortran code from the authors was compared to the one PyMMS generated by probing it in thousands of random location and time combinations and the difference between the two implementations was found to be within the margin of the round off error of the two fortran codes. Hence verifying the correctness of PyMMS.

As an example, a basic input file for PyMMS is presented in Algorithm 3.1. It defines the velocity components as well as a constant pressure field and runs the fortran code generation. Algorithm 3.2 is then a portion of the generated code. It can be noted that some parameters are left in the source code for easier edition without requiring a new generation of the source terms.

```
from sympy import *
from PyMMS import PyMMS

# Definition of parametric symbols and their default value
x, y, z, t = symbols('x y z t')
T = symbols('Period')
A = symbols('A')
B = symbols('B')
global_vars = [(T, 1), (A, 10), (B, 5)]

# Definition of field functions
U = A*cos(2*pi*t/T)*cos(x)
V = B*cos(2*pi*t/T)*cos(y)
W = - Integral(diff(U, x)+ diff(V, y), (z, 0, z))
P = Integer(1)

# MMS generation and export to file
mms = PyMMS(Nu=1, rho=1, U=U, V=V, W=W, P=P, turbulence_model="none")
```

```
mms.compute_sources()
mms.export_module("module-basic-example.F90", global_vars=global_vars)
```

ALGORITHM 3.1: Example of input file for running PyMMS, written in python.

```
module MMS
implicit none
REAL*8, parameter :: Period = 1
REAL*8, parameter :: A = 10
REAL*8, parameter :: B = 5
contains
!...
REAL*8 elemental function source_MOM_U(x, y, z, t)
REAL*8, intent(in) :: x
REAL*8, intent(in) :: y
REAL*8, intent(in) :: z
REAL*8, intent(in) :: t
REAL*8, parameter  :: pi = 3.14159265358979 32d0

source_MOM_U = -2*A**2*sin(x)*cos(x)*cos(6.2831853071795865d0*t/Period) &
      **2 - A*B*sin(y)*cos(x)*cos(6.2831853071795865d0*t/Period)**2 - A &
      *(-A*sin(x)*cos(6.2831853071795865d0*t/Period) - B*sin(y)*cos( &
      6.2831853071795865d0*t/Period))*cos(x)*cos(6.2831853071795865d0*t &
      /Period) + A*cos(x)*cos(6.2831853071795865d0*t/Period) - 2*pi*A* &
      sin(6.2831853071795865d0*t/Period)*cos(x)/Period
end function
!...
end module
```

ALGORITHM 3.2: Extract of the generated fortran module file created by the previous source code showing the source term for the first component of the momentum equation.

3.4 PyTST: an opensource Transient Scanning Technique analysis tool

To perform the Transient Scanning Technique developed by Brouwer et al. [12] a python library and tool were developed under the MIT license [66]. They take as input either an ASCII text file containing the signal data or any python arrays and produce the two graphs needed to make the analysis: the signal itself, and the log-log plot of statistical uncertainty against the window size. Moreover, for easier analysis, the two plots are colour coded and interactive. Selecting part of the signal to discard will highlight the corresponding uncertainty and *vice versa*.

Figure 3.3 shows the window when running the tool interactively. The mouse can be used to zoom in and out as well as move the cut-off location to recompute the signal mean and statistical uncertainty. For performance reasons, the code is parallelised to run on shared memory computers. Moreover, a `step-size` parameter is introduced to only compute the uncertainty each `step-size` sample. This has the effect of lowering the resolution of the bottom uncertainty graph but drastically reduces the number of calculation to run. It should be noted that the uncertainties themselves are not affected by this parameter as they use the full length original input and not a downsampled version of it.

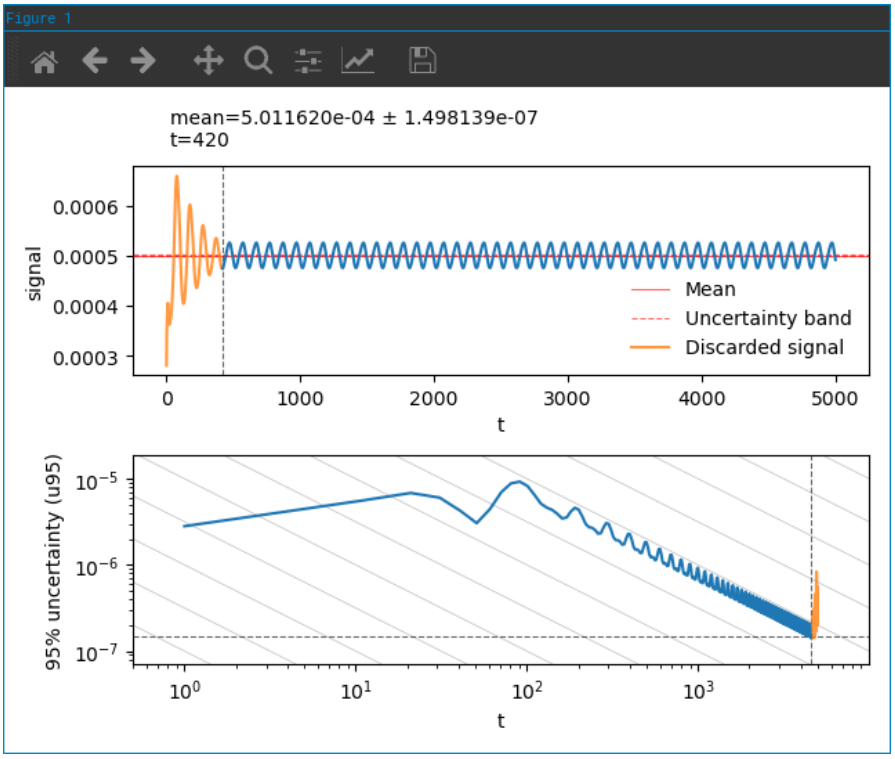


FIGURE 3.3: Example window when running pyTST interactively. It shows the original signal on the top and the statistical uncertainty on the bottom. Both plots are synchronised so that moving the cut-off location manually on either plot updates the other one.

3.5 Summary

The error sources discussed in this Chapter are always present in CFD computations, making their analysis difficult in isolation. To address this issue, methods have been developed to either quantify the strength of these errors or reduce them to negligible levels. In this work, tools developed in Eça and Hoekstra [33], Eça et al. [37], as well as the Transient Scanning Technique by Brouwer et al. [13] and the method of manufactured solutions are used to analyse the overset implementation and isolate interpolation errors. These errors are directly influenced by the interpolation scheme used and its implementation, which are described in the following Chapter.

Chapter 4

The Overset method implementation

The overset method developed in this work was designed to allow the use of higher order interpolation schemes and to be computationally performant. These two requirements directly influenced the implementation. In this Chapter, details are given about overset method implementations found in the literature and an overview of the architecture developed for this project. Furthermore, a detailed presentation of the different interpolation schemes that can be used by this implementation is provided.

4.1 General workflow

Any overset implementation is composed of different steps. First the hole cutting algorithm computes cells status like *hole*, *in* or *fringe*, from the geometry and topology of the different meshes as well as user defined 'rules' (giving informations on boundaries roles and sometimes grid priorities). The methods used can be complex and vary between implementation, although two main methods can be distinguished, *implicit* or *explicit* hole cuttings (not to be confused with the implicit or explicit coupling between meshes). In an explicit formulation, the hole cutting is done directly using the mesh geometry, the body surface mesh cutting a hole in the other meshes. On the other hand, with an implicit approach, the decision to cut a grid is purely based on local grid information, often the grid with the best resolution will cut a hole in all the other grids that are locally overlapping it. The main drawback of implicit hole cutting being that it is more complex to compute and more computationally demanding than explicit hole cutting, but as it relies only on local information it is easier to parallelise.

Once the hole cutting is performed, *donor* cell(s) need to be searched for each *fringe* cell. In most implementations on unstructured meshes, a first *donor* cell is found as

the cell directly overlapping its associated *fringe* cell (but from another mesh), and all the direct neighbours of this first *donor* cell are also tagged as *donor* for this particular *fringe* cell. This means that there is no control over the exact number of *donor* cells collected per *fringe* cell. On structured meshes, *donor* searching methods can make use of the mesh topology to easily and cheaply collect more than the first layer of neighbours around the first *donor* cell.

It should be noted that the hole cutting and *donor* search method chosen do not have a direct influence on the accuracy of the solution but their combination with an interpolation method can. The selection of *donor* cells far from the *fringe* cell centre location leading to extrapolation can, for example, be considered a poor selection and result in large errors. Other examples *donor* selection leading to accuracy losses are given in Chapter 5.

Subsequently, once *donor* cells are found, the interpolation itself can be performed. Depending on the scheme used and the coupling implemented, this is done explicitly or implicitly as presented in Chapter 2. When the interpolation can be formulated as a linear combination of the *donor* information, besides being usable in an implicit formulation, in an explicit implementation the weights associated to each *donor* cell can be stored and reused when ever the *donor* values are updated for the new interpolated value and the meshes did not move relative to each other. In practice weights can be reused for each outerloop and when the motion is periodic. Hence saving the cost of recomputing the weights themselves.

4.2 Overset in various CFD solvers

Depending on the implementation, the different components of the overset method are either done by an external library or by the flow solver itself. The use of external libraries or separate implementations is often justified by large requirement differences between the CFD solver and the overset method. In terms of parallelisation for example, the solver uses domain decomposition and relies mainly on local information for its operation, while an explicit hole cutting step requires knowledge of the entire mesh geometry.

One of the libraries is called Suggar (renamed Suggar++ in 2009 after a major update) and is mainly developed by R. Noack [86, 88]. The primary objective behind its development is to facilitate the integration of overset capabilities in any CFD code. This is done using several high level functions accessible via a Fortran or C interface, and the library allows use of both structured and unstructured grids, cell and node centered solvers etc. [87]. Several integrations of Suggar++ are reported in the literature, like CFDShip-Iowa [16], or UNCLE with the work detailed in Boger and Dreyer [6]. More recently, two implementations of the coupling were reported for the open source framework OpenFOAM, the first one using Suggar++ by Boger et al. [8] called FoamedOver, the second one by

Shen et al. [114] coupling with Suggar this time. Another library implementing the overset method is OPERA, contrary to Suggar it seems targeted towards OpenFOAM only, it has, for example, been used in the foam-extend fork [46]. Both OPERA and Suggar/Suggar++ are closed source, which is not the case for TIOGA (Topology Independent Overset Grid Assembler) as it is published under the LGPL license. Its main developers are from the University of Wyoming [10, 116] and, like Suggar/Suggar++ it can also work with a wide range of mesh type (structured, unstructured, cell or node centered etc.), though, unlike Suggar/Suggar++, it does not support polyhedra cells. Finally, it has also been used for multi-solver simulations (using different CFD solver on different meshes).

The libraries Suggar++ and Suggar are both using an explicit method hole cutting method, though Suggar++ has a more accurate algorithm by using the exact geometry of the surfaces instead of a simplified one in Suggar. This allows for tighter holes to be cut and a minimisation of the overlap [88]. On the other hand, OPERA and TIOGA are using implicit approaches described in Gopalan et al. [46] and Gopalan et al. [46] respectively. For TIOGA, its implicit calculation overhead is compensated its capability to run fully in parallel using MPI, whereas Suggar++ parallelisation is done either with shared memory only or is limited to a single process per mesh. TIOGA's parallelisation and implementation has, however, the consequence of having its DCI influenced by the domain decomposition [27].

In contrast, several implementations do not get their overset capability using external libraries but the implementation is done within the flow solver itself. This is the case of the ESI-openCFD fork of OpenFOAM since 2017, StarCCM+ has this feature since version 2012 with version 7.02 [111], FINE/Marine (Isis CFD) since 2016 [28], FRESCO+ from HSVA since 2017 [126], ANSYS-Fluent since version R17 released in 2016 [23] or INSEAN flow solver from the Italian Ship Model Basin Rome, Italy since 2006 [82].

Regardless of the way the overset method is implemented, interpolation is a core component of any overset code as it has a direct impact on the accuracy of the solution. The literature, however, does not always describe in detail the interpolation method used. The *donor* cell collection method is not often mentioned, and the actual interpolation scheme is either not mentioned, or not described in enough detail to be understood or re-implemented [6, 17, 24, 114, 126, 130]. Table 4.1 summarises the data found in the literature, and where possible, a nomenclature similar to the one used in the presented work is used. Moreover, unstructured and structured solvers are distinguished as the *donor* collection can be done very easily with a structured code, and the interpolation schemes can take advantage of this. This is, for example, the case for UP_GRID solver [60], which generates a spline in the entire domain for the interpolation. Similarly, Suggar++, when used with structured codes, can use a *Polynomial tensor* interpolation requiring n^3 *donor* cells, a search for these being trivial on a structured grid. It must be noted that interpolation orders higher than two are only possible with OPERA library

using a *Least squares* approach. Overall, most of the other solvers implement a *Nearest cell* and an *Inverse distance* scheme (both 1^{st} order). Also, even though the *donor* cells collection method is not always described, at least three of the six solvers are using the closest cell's direct neighbours. This means that the number of *donor* cells collected for each *fringe* cell depends on the grid topology and can not be changed. Finally, several schemes make use of the gradient at the *donor* cell centre locations to achieve 2^{nd} order interpolations. In 2012, Quon and Smith [100, 101] also tested the use of Radial Basis Functions (RBF) for overset interpolation, but, even though positive conclusions were drawn, it seems that no other overset implementation followed its trend, possibly because most overset implementations rely on having only the direct neighbours of the closest cell as *donor* cells, the RBF method, however, requires more *donor* cells, making its implementation and parallelisation more challenging.

TABLE 4.1: Summary of interpolation methods used in various overset implementations.

Solver	Type	Scheme	Number of *donor* cells	Order
Fresco+ [126]	Unstructured	*Nearest cell*	1	1
		Inverse distance	Neigh[a]	1
		Inverse Simplex Volume[b]	4	1 or 2
StarCCM+ [111]	Unstructured	*Inverse distance*	4	1
		Shape Function		1
		Gradien Least Squares[f]	Neigh[a]	2
ANSYS-Fluent [23]	Unstructured	Linear [d]	4	1
		Inverse distance		1
		Gradient Least Squares[f]	Neigh[a]	2
foam-extend [43, 127]	Unstructured	*Nearest cell*	1	1
		Average		1
		Inverse distance		1
ISIS CFD/FINE Marine [28]	Unstructured	*Least squares*	Neigh[a]	2
OPERA (OpenFOAM) [20, 21, 23, 46]	Unstructured	Linear[d]	4	1
		Inverse distance		1
		Least squares	-[e]	2-4
Suggar++[c] [86–89]	Both	*Nearest cell*	1	1
		Inverse distance	Neigh[a]	1
		Laplacian	Neigh[a]	2
		Least squares	Neigh[a]	2
		Dual grid	-	2
	Structured	*Polynomial tensor*	n^3	n
LAVA [57]	Unstructured	*Least squares*	Neigh[a]	
	Structured	*Polynomial tensor*	8	2
Chimera Grid Tools (CGT) [18]	Structured	*Polynomial tensor*	8	2
UP_GRID (NMRI) [60, 92]	Structured	Ferguson spline		
Pusan University [85]	Structured	*Barycentric*		

[a] The *donor* cells are the closest cell and its direct neighbours

[b] Interpolation weights based on the inverse simplex volumes, it is made 2nd order by using *donor* cells gradients

[c] Suggar++ is a library being used by several solvers like CFD-Shipflow[72], OVERFLOW[42] and some implementations of OpenFOAM[7, 114]

[d] Using vertices to construct a tetrahedron and perform the interpolation

[e] OPERA is not using a fixed number of *donor* cells and therefore the interpolation order is not guaranteed, a pseudo-inverse method is used if not enough *donor* cells are collected and the system is under-determined

[f] The least squares approach is not coupled with a polynomial function here. The gradient at the *donor* cell centre is used to extrapolate a field value at the interpolation location from each *donor* (like the *Nearest cell gradient* scheme), then a least squares approach is used as an averaging method between the values

4.3 Implementation of the overset method

4.3.1 General design decisions

The goals of the implementation of the overset method in ReFRESCO done as part of this research were to have a state of the art technique with the help of existing implementations while still keeping maximum control over key components that are relevant for the accuracy of the method. This goal was achieved by combining external libraries and in-house code. Suggar++ [88] and its companion library DiRTlib [87] were picked for their proven robustness and easy integration with various solvers. They provide all the components needed to add overset capabilities to any CFD code with hole cutting, *donor* search, interpolation (with and without weights), recomputation of the DCI in case of mesh motion or change (due to deformation) and can be integrated with structured and unstructured solvers written in C, C++ or Fortran. For this work, however, to keep control over the interpolation steps and experiment with a wide variety of schemes, only the hole cutting and cell status assignment are done by Suggar++ leaving the *donor* search and interpolation to ReFRESCO. In detail, Suggar++ is given the meshes and a setup file and returns an IBLANK array. This integer array gives for each cell in the domain its status: *fringe*, *hole* or *in*. Section 4.3.2 details the implementation of the *donor* search, and the various interpolation schemes implemented are presented in section 4.4.

The coupling with the flow solver is done explicitly for all equations. The pressure field is also being interpolated on *fringe* cells. The interpolated values are directly placed in the field together with a zero pressure correction on the right hand side of the pressure correction system. It should be noted that, for the pressure correction equation, this method is equivalent to an implicit coupling with null interpolation weights. Finally, for schemes that allow it, interpolation weights are computed and stored in memory to be re-used when grids are not moving to decrease computational cost.

Figure 4.1 presents the structure of the overset code. It includes a cache system for IBLANK data. Indeed, as the computation of the DCI done by Suggar++ can be expensive and only relies on the mesh positions, computations with a periodic or prescribed motion can take advantage of it by storing IBLANK data. With the cache system, every time-step, if the position of the meshes has not been computed before, Suggar++ is run and the cells statuses that it returns are stored to file for future use. The cache file itself only stores the global cell id number of every *fringe* and *hole* cell in a binary form and the file name encodes the mesh positions to avoid the use of look up tables. This cache system allows to run Suggar++ a minimum amount of times. For example, when running several computations changing only the interpolation scheme, a single set of cache files can be used.

Once the IBLANK data is loaded the *donor* search is performed for each *fringe* cell, this step is detailed in section 4.3.2. Then, if the interpolation scheme allows it, interpolation weights are computed. Every outerloop, the field values of *donor* cells changes and the interpolation needs to be recomputed. First, the *donor* values are transmitted to the process that will perform the interpolation itself which is the process storing the first *donor* cell. Then the interpolation is performed and the interpolated value is transferred to the process that has the corresponding *fringe* cell. This methodology offers some load balancing and minimises communications as *donor* cells mostly belong to the process that computes the interpolation. All of the exchanges use point to point communications as it allows versatility and data is only transferred between the processes that require it, hence minimising the amount of data exchanged.

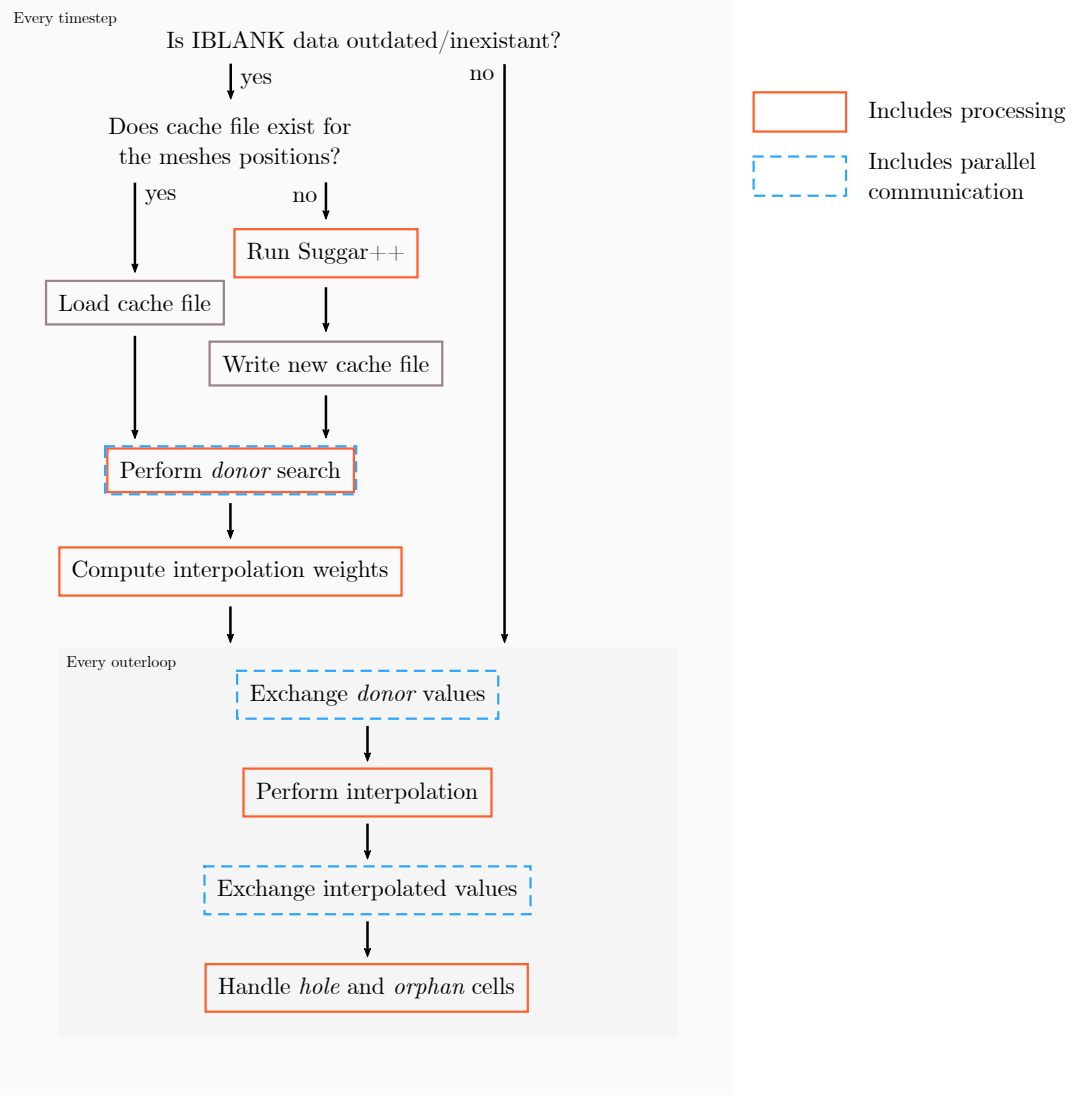


FIGURE 4.1: Flow chart of the overset implementation.

4.3.2 Donor search implementation

The *donor* cell search was designed and implemented to minimise the number of parallel synchronisation steps and the amount of data being communicated between MPI processes. The algorithm is given in pseudocode in Figure 4.2. The general idea to select *donor* cells is to first find the cell that contains the interpolation location (*fringe* cell centre) and then gather its neighbours, followed by the neighbours of the neighbours etc., until enough *donor* cells have been gathered for the scheme used. The order in which neighbours of the same *layer* are gathered is based on their distance to the location of *fringe* cell centre. This method is different from most overset *donor* search because the number of cells being gathered is not limited to the first *layer* of neighbours allowing to improve the robustness of some schemes as demonstrated in Chapter 5. The performance of this approach is tested in Chapter 8.

```
1  For cell in global_fringe_cell list
2  | If donor cell found in local domain
3  |   gather 1st donor cell ■
4  |   add cell to local_fringe_cell list
5
6  fix duplicates & orphans
7
8  While nb_donor < requested_nb_donor
9  | For cell in local_fringe_cell list
10 | | gather next layer of donor cells ■, ■, ■
11 |
12 | exchange neighbour domain cells
13
14 prepare future communication
```

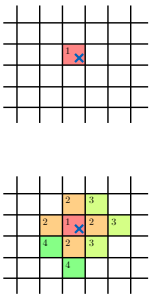


FIGURE 4.2: *Donor* search algorithm presented in pseudocode. Steps circled in light blue are parallel communications. Similarly, 'local' and 'global' refer to the parallel domain decomposition.

In detail, first, the first *donor* cell is found for each point where the interpolation is needed. This is done locally on each process and facilitated with an octree search. If no local suitable *donor* cell is found for a particular *fringe* cell it means the *donor* cells will be on another process and the loop continues. Once this is done, each *fringe* cell should be associated with one and only one *donor* cell, this is checked with a parallel communication. If it is not the case, either a *fringe* cell is on the edge of two *donor* cells that belong to two different processes and both of them are gathered, this is easily fixed by discarding one of them. Otherwise, if a *fringe* cell has no *donor* cell, the cell with the closest cell center is gathered as *donor* cell as a fallback. This prevents the presence of *orphan* cells at the cost of a potentially poorer *donor* selection. For each local *donor* cell found at the previous step, its neighbours are gathered as *donor* cells themselves, followed by the neighbours of the second *donor* cell etc. Parallel communication is needed after each set of neighbours is gathered to allow the search to expand to the neighbouring parallel domain when needed. Finally, once enough *donor* cells have been gathered for each *fringe* cell, a last communication step is used to prepare future exchanges after

which each process will know on which sub domain their interpolation will be computed to be able to retrieve it later on.

This implementation minimises the amount of data being exchanged and favours a few larger parallel communications steps over many small ones. For example, in a 3D Cartesian mesh (each cell having 6 neighbours), if 20 *donor* cells are needed for each *fringe* cell, the `while` loop will be run four times in total, leading to five exchange steps.

4.4 Interpolation methods

As part of this work, seven different interpolation schemes are implemented for overset computations. This section details their mathematical characteristics. For consistency, the location of the interpolation is denoted by $\boldsymbol{x} = (x, y, z)$, the cell centre of a *fringe* cell. N is the number of *donor* points needed by the scheme, in an overset context, a *donor* point is the cell centre of a *donor* cell. The *donor* cell centre locations are $\boldsymbol{x_i}$ with $1 \leq i \leq N$. Finally, ϕ is the field to be interpolated and the interpolated field is named $\tilde{\phi}$. The goal of the interpolation is then to get $\tilde{\phi}(\boldsymbol{x})$ using $\boldsymbol{x_i}$ and $\phi(\boldsymbol{x_i})$ for $1 \leq i \leq N$.

4.4.1 Nearest Cell

This scheme has only a single *donor* cell and the interpolated value is equal to the *donor* cell value. It is a 1^{st} order scheme.

$$\tilde{\phi}(\boldsymbol{x}) = \phi(\boldsymbol{x_1}). \tag{4.1}$$

4.4.2 Nearest Cell Gradient

This scheme also uses a single *donor* with the addition, however, of its gradient at the cell centre to correct the interpolated value. The gradient used here does not need to be computed specifically for the interpolation as it is already computed and used by the flow solver as mentioned in section 2.3.2. The use of the gradient makes it a 2^{nd} order scheme.

$$\tilde{\phi}(\boldsymbol{x}) = \phi(\boldsymbol{x_1}) + \nabla\phi(\boldsymbol{x_1}) \cdot (\boldsymbol{x} - \boldsymbol{x_1}). \tag{4.2}$$

4.4.3 Inverse Distance

The *Inverse distance* interpolation scheme is sometimes called a weighted average method. The number of *donor* cells N can be freely chosen and the interpolation is defined as follow:

$$\tilde{\phi}(\boldsymbol{x}) = \frac{\sum_{i=1}^{N} w_i \phi(\boldsymbol{x_i})}{\sum_{i=1}^{N} w_i}. \tag{4.3}$$

$$w_i = \frac{1}{||\boldsymbol{x} - \boldsymbol{x_i}||^p}. \tag{4.4}$$

The weights w_i decay with the inverse of the distance to the interpolation location $\boldsymbol{x}$ ($||.||$ being the 2-norm), and this decay is controlled by the power p. In other words, p controls the smoothness of the resulting field, a lower p-value will produce a smoother field as it decreases the relative difference between each weights. The extreme being with $p = 0$ where the *Inverse distance* scheme is equivalent to a geometric average. Tests made with this method show that the p value has little influence on the accuracy of the scheme, as a balance $p = 3.5$ is used in this work. This is a 1^{st} order scheme, and, as the weights $(w_j / \sum_{i=1}^{N} w_i)$ are bounded between 0 and 1, the interpolated value will be bounded between the minimum and maximum field value of its *donor* cells. Also, using a single *donor* cell makes it equivalent to the *Nearest cell* scheme.

4.4.4 Polynomial

Polynomial interpolation schemes generate a polynomial function that will pass through all the *donor* cells values and the evaluation of this polynomial function at the interpolation location ($\boldsymbol{x}$) is the interpolated value. A polynomial function can be expressed in the form,

$$P(\boldsymbol{x}) = P(x, y, z) = \sum_{k=1}^{N} \alpha_k F_k(\boldsymbol{x}), \tag{4.5}$$

where N is the number of coefficients α_k, and F_k are the polynomial basis functions. Two different sets of basis functions are presented in the following sections. The first one named *Complete Polynomial* will simply be called *Polynomial* interpolation in the rest of the work, and the second one is called *Polynomial tensor* interpolation.

4.4.4.1 Complete Polynomial

This type of *Polynomial* interpolation, creates a degree n polynomial function that has all the terms of degree n and below, hence the adjective *complete*. Therefore, in 2D,

for a polynomial of degree n, the basis functions will be all $F = x^r y^s, r + s \leq n$ with $r, s \in \mathbb{N}$. Similarly, in 3D, $F = x^r y^s z^t, r + s + t \leq n$ with $r, s, t \in \mathbb{N}$

To find the polynomial function that passes through all of the *donor* cell values, the α_k coefficients need to be found:

$$\sum_{k=1}^{N} \alpha_k F_k(\boldsymbol{x_i}) = \phi(\boldsymbol{x_i}), \quad 1 \leq i \leq N. \tag{4.6}$$

Which can be rewritten as:

$$[A_{k,m}][\alpha_k] = [rhs_k], \quad 1 \leq k, m \leq N. \tag{4.7}$$

With

$$[A_{k,m}] = F_k(\boldsymbol{x_m}) \tag{4.8}$$

$$[rhs_k] = \phi(\boldsymbol{x_k}) \tag{4.9}$$

Once the system is solved and the α_k are obtained, the interpolated value is given by:

$$\tilde{\phi}(\boldsymbol{x}) = \sum_{k=1}^{N} \alpha_k F_k(\boldsymbol{x}) \tag{4.10}$$

There is uniqueness in the polynomial function generated, however different basis functions can be chosen to improve the conditioning of the system to be solved. Eça [32] tested different sets of basis functions but did not experience major changes in the results. Moreover, as there is uniqueness in the generated polynomial function, this method is equivalent to the Lagrange polynomial interpolation sometimes found in the literature.

For a polynomial of degree n, its interpolation order is $n+1$. Table 4.2 shows the number of *donor* cells needed for the *Polynomial* interpolation.

TABLE 4.2: Number of *donor* cells N depending on the dimension of the problem and the degree of the polynomial function n.

Dimension	**N (number of *donor* cells)**			
	n (degree)	$n = 1$	$n = 2$	$n = 3$
2D	$\frac{(n+1)(n+2)}{2}$	3	6	10
3D	$\sum_{i=0}^{n} \frac{(i+1)(i+2)}{2}$	4	10	20

4.4.4.2 Polynomial tensor

The *Polynomial tensor* interpolation is similar to the previously presented complete polynomial interpolation; the only difference lies in the basis functions.

The basis functions used by the *Polynomial tensor* interpolation of degree n will be, in 2D, all the terms of the product: $(\sum_{r=0}^{n} x^r)(\sum_{s=0}^{n} y^s)$, and in 3D, the terms of $(\sum_{r=0}^{n} x^r)(\sum_{s=0}^{n} y^s)(\sum_{t=0}^{n} z^t)$. In other words, the basis functions in 2D are $F = x^r y^s$ with $0 \leq r, s \leq n$ and in 3D $F = x^r y^s z^t$ with $0 \leq r, s, t \leq n$. Compared with the complete polynomial basis functions of the same degree, the *Polynomial tensor* adds some crossterms of degree higher than n.

TABLE 4.3: Number of *donor* cells N depending on the dimension of the problem and the degree for the *Polynomial tensor* interpolation.

Dimension	**N (number of *donor* cells)**			
	n (degree)	$n = 1$	$n = 2$	$n = 3$
2D	$(n+1)^2$	4	9	16
3D	$(n+1)^3$	8	27	64

In 2D and 3D the *Polynomial tensor* interpolation is sometimes called bi-linear (2D degree 1), bi-cubic (2D degree 2), or tri-linear (3D degree 1), tri-cubic (3D degree 2), etc. This kind of interpolation is often used in finite element methods. To interpolate inside a square in 2D or a cube in 3D, the bi-linear or tri-linear interpolations allow the use of the vertices as *donor* points.

4.4.5 Least squares

The *Least squares* interpolation also relies on complete polynomial functions but more *donor* cells than the number of unknown coefficients α_k are collected. The system is solved by minimising the sum of the squares of the errors at each *donor* cells location. Increasing the number of *donor* points improves the robustness of the method.

In this work, the number of *donor* cells collected is defined using the donor point multiplier C_{mult}, it is the ratio between the number of *donor* cells and the number of unknowns in the polynomial function, formally defined with $N = \lceil C_{mult} N_{terms} \rceil$, with $C_{mult} \in \mathbb{R}, C_{mult} > 1$, $\lceil\ \rceil$ the ceil operator, N is the number of *donor* cells and N_{terms} is the number of terms in the polynomial (number of unknown coefficients α_k).

The polynomial function is then:

$$P(\boldsymbol{x}) = \sum_{k=1}^{N_{terms}} \alpha_k F_k(\boldsymbol{x}). \tag{4.11}$$

The *Least squares* approach is to find α_k that minimise the sum of the squared errors S:

$$S = \sum_{i=1}^{N} \Big(\phi(\boldsymbol{x_i}) - \sum_{k=1}^{N_{terms}} \alpha_k F_k(\boldsymbol{x_i}) \Big)^2. \tag{4.12}$$

The minimisation of S is computed by finding the roots of its derivative regarding each α_l

$$\frac{\partial S}{\partial \alpha_l} = -2 \sum_{i=1}^{N} F_l(\boldsymbol{x_i}) \Big(\phi(\boldsymbol{x_i}) - \sum_{k=1}^{N_{terms}} \alpha_k F_k(\boldsymbol{x_i}) \Big) = 0 \tag{4.13}$$

Which is equivalent to the $N_{terms} \times N_{terms}$ system:

$$[A_{l,m}][\alpha_l] = [rhs_l], \quad 1 \leq l, m \leq N_{terms} \tag{4.14}$$

With:

$$A_{l,m} = \sum_{i=1}^{N} F_l(\boldsymbol{x_i}) F_m(\boldsymbol{x_i}) \tag{4.15}$$

$$rhs_l = \sum_{i=1}^{N} F_l(\boldsymbol{x_i}) \phi(\boldsymbol{x_i}) \tag{4.16}$$

4.4.6 Barycentric

The *Barycentric* interpolation goal is to assign to each *donor* point a weight that would make the interpolation location the centre of gravity of the system and then use these weights as interpolation weights. However, such weights can be found only if the interpolation location is inside the convex polyhedron made by all the *donor* points. In order to make sure the interpolation location is inside this polyhedron, the cell vertices are used as *donor* points instead of *donor* cell centres.

First, the cell that contains the interpolation location is found. This cell is subdivided into tetrahedrons formed by the cell centre and one triangulated face, and the tetrahedron sub cell that contains the interpolation location is selected. Barycentric weights are computed for the vertices of this sub cell. The *Barycentric* interpolation finds weights w_i associated with the vertices v_i that will lead to the interpolation location $\boldsymbol{x}$ being the barycentre or centre of mass of the system. With $\bar{\phi}_{vert_i}$, the value at one vertex of the sub-cell, one can compute the value at the interpolation location with:

$$\tilde{\phi}(\boldsymbol{x}) = \frac{\sum_{i=1}^{N} w_i \bar{\phi}_{vert_i}}{\sum_{i=1}^{N} w_i} \tag{4.17}$$

In order to get the values at the vertices of the sub-cell, four different methods can be used. First, the values at the vertices of the cell can be obtained in two different ways:

$$\bar{\phi}_{vert} = \frac{1}{N_{adj}} \sum_{j=1}^{N_{adj}} \phi(\boldsymbol{x_j}), \quad (type\ 1\ and\ 3) \tag{4.18}$$

$$\bar{\phi}_{vert} = \frac{1}{N_{adj}} \sum_{j=1}^{N_{adj}} \left(\phi(\boldsymbol{x_j}) + \nabla\phi(\boldsymbol{x_j}) \cdot (\boldsymbol{x_j} - \boldsymbol{x_{vert}})\right), \quad (type\ 2\ and\ 4) \tag{4.19}$$

With N_{adj} the number of cells adjacent to the vertex and $\boldsymbol{x_j}$ the cell centre location of an adjacent cell.

Second, the value at the cell centre (which is the last vertex of the sub-cell) is already known, and can be used directly (type 1 and 2), or it can be reconstructed by taking the arithmetic average of the cell vertex values (type 3 and 4). Table 4.4 summarises the different *Barycentric* interpolation types.

TABLE 4.4: Summary of the different ways to get the values at the vertices of the sub-cell.

		Face vertices	
		cell centre only	cell centre + gradient
Cell centre	cell centre	type 1	type 2
	reconstructed	type 3	type 4

For this method, it is not trivial to derive an interpolation order, Chapter 5.2 however shows a 2nd order for the tested case.

4.4.7 Interpolation schemes overview

Of the interpolation schemes presented, only the polynomial ones (*Polynomial*, *Polynomial tensor* and *Least squares*) can be used with higher order as there is no limit in the theoretical degree of the function they generate. It should, however, be noted that with higher orders, the number of *donor* cells required increases drastically.

When performing an interpolation one can directly get the interpolated value $\tilde{\phi}(\boldsymbol{x})$. Though, in order to perform an implicit coupling or to precompute interpolation weights, explicitly computing these weights is required. $\tilde{\phi}(\boldsymbol{x})$ is then computed as a linear combination of the $\phi(\boldsymbol{x_i})$ for $1 \leq i \leq N$. It can be written under the form:

$$\tilde{\phi}(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \phi(\boldsymbol{x_i}). \tag{4.20}$$

The coefficients w_i are the interpolation weights and are placed in the left hand side of the system in an implicit coupling. The *Nearest cell* and *Inverse distance* schemes yield a trivial implicit formulation. The *Nearest cell gradient* scheme, on the other

hand, cannot be written fully implicitly since the gradient at the *donor* cell is pre-computed and does not directly depend on the neighbouring cell center values alone. A semi-implicit formulation is however possible with the gradient term placed on the right hand side and the rest on the left hand side in the matrix. Concerning polynomial interpolations (*Polynomial*, *Polynomial tensor* and *Least squares*) one needs to express the α_k coefficients in terms of $\phi(\boldsymbol{x_i})$ and reorganise the terms of the equation to get an implicit form. To this end, the inverse of the matrix A (Equation 4.8 or Equation 4.15) is explicitly needed. Appendix A details the construction of w_i for both Polynomial and Least Squares approaches.

For the *Polynomial* and *Polynomial tensor*, w_i is expressed by:

$$w_i = \sum_{k=1}^{N} F_k(\boldsymbol{x}) A^{-1}_{k,i}. \tag{4.21}$$

And for the *Least squares* approach, w_i can be expressed as:

$$w_i = \sum_{k=1}^{N_{terms}} F_k(\boldsymbol{x}) \Bigg(\sum_{l=1}^{N_{terms}} F_l(\boldsymbol{x_i}) A^{-1}_{k,l} \Bigg). \tag{4.22}$$

The barycentric interpolation could also be rewritten in an implicit form since the weights w_i only depend on the interpolation location $\boldsymbol{x}$ and the value at the vertices is a linear combination of the cell centre values, though the implementation used in this work does not use this and only direct interpolation is available.

Finally, some schemes will always result in bounded interpolation, meaning that the interpolated value lies between the minimum and maximum values of the *donor* cell values. Such a characteristic is useful as it guaranties the robustness of the scheme, though only low order schemes have such a feature. Table 4.5 summarises the different schemes presented in this Chapter and their main characteristics.

TABLE 4.5: Summary of the different interpolation schemes available for the current overset implementation and their main characteristics. The number of *donor* cells for the barycentric interpolation depends on the topology and is here given as an example for a Cartesian mesh.

Scheme	**Number of *donor* cells**		**Order (theoretical)**	**Boundedness**
	2D	3D		
Nearest Cell	1	1	1	yes
Nearest Cell gradient	1	1	2	no
Inverse Distance	N	N	1	yes
Polynomial ($degree = 1$)	3	4	2	no
Polynomial ($degree = 2$)	6	10	3	
Polynomial ($degree = 3$)	10	20	4	
Polynomial ($degree = n$)	$\frac{(n+1)(n+2)}{2}$	$\sum_{i=0}^{n} \frac{(i+1)(i+2)}{2}$	$n+1$	
Polynomial Tensor ($degree = 1$)	4	8	2	no
Polynomial Tensor ($degree = 2$)	9	27	3	
Polynomial Tensor ($degree = 3$)	16	64	4	
Polynomial Tensor ($degree = n$)	$(n+1)^2$	$(n+1)^3$	$n+1$	
Least Squares ($degree = 1$)	> 3	> 4	2	no
Least Squares ($degree = 2$)	> 6	> 10	3	
Least Squares ($degree = 3$)	> 10	> 20	4	
Least Squares ($degree = n$)	$> \frac{(n+1)(n+2)}{2}$	$> \sum_{i=0}^{n} \frac{(i+1)(i+2)}{2}$	$n+1$	
Barycentric (type 1)	6	18		yes
Barycentric (type 2)	6	18		no
Barycentric (type 3)	9	27		yes
Barycentric (type 4)	9	27		no

4.5 Summary

The design choices made when developing this overset method have a direct impact on the accuracy of the resulting computations because interpolation errors made on *fringe* cells affect the rest of the solution through their neighboring *in* cells. In addition to accuracy, the code architecture and parallelisation decisions also impact performance. Chapters 5 to 7 focus on the accuracy and robustness of the method, while Chapter 8 analyses performance results. It is important to consider both accuracy and performance when providing useful guidelines.

Chapter 5

Verification of interpolation schemes

The first step in assessing the accuracy of the overset method, and of interpolation schemes in particular, is to test them in isolation outside of any CFD computation. This allows for better control over the input to verify their good behavior and ensure that they follow their theoretical order of convergence. Furthermore, polynomial based interpolation schemes are known to suffer from robustness issues with certain cell arrangements, leading to ill-conditioned systems that are difficult to solve. Therefore, the robustness of polynomial based schemes is first tested in section 5.1, and the error levels and convergence orders of all the schemes implemented in this work are analysed in section 5.2.

5.1 Robustness of Polynomial based interpolations

The robustness of a scheme is defined by how likely the interpolation is to result in very large errors for a set of *donor* cells. A more robust scheme will also have a narrower error distribution for a given function to interpolate. In summary, the interpolation error of a robust interpolation scheme is not influenced by the locations of the *donor* cells nor the underlying mesh, making it more predictable.

In this section, the robustness of polynomial based interpolation is tested. This includes the *Polynomial*, *Polynomial tensor* and *Least squares* schemes, all of which are examined at degrees 1 to 3, resulting in 2^{nd} order to 4^{th} order interpolations. Furthermore, concerning the *Least squares* scheme, since the number of *donor* points is not fixed by the method, several C_{mult} values are tested. Finally, this section focuses only on polynomial based schemes as they are the only ones known to present potential robustness

issues, all the other ones being intrinsically bounded (or bounded with an added gradient correction for *Nearest cell gradient*).

5.1.1 Methodology

5.1.1.1 *Donor* points locations

In a finite volume cell centred CFD solver, the overset interpolation *donor* points are located at the cell centres of *donor* cells associated to a *fringe* cell. In this study, because the interpolation itself is not affected by the underlying mesh but only by the cell centre locations, in order to test thousands of *donor* point arrangements quickly, no mesh is used. Instead, *donor* points are placed in a partially random way in the 2D $[0, 1] \times [0, 1]$ square domain. Two different methods were designed and implemented to mimic what different cell centre locations on a CFD mesh could look like. The first method is meant to replicate a Cartesian or structured grid and will be called *Cartesian* in this section. If N *donor* points are needed for the interpolation, a 'virtual' Cartesian grid of size $\lfloor\sqrt{N} + 1\rfloor + 1$ is created in the $[0, 1] \times [0, 1]$ domain. For example, for 6 *donor* points, the Cartesian grid is 4×4. Then N different cells are randomly selected in the grid and the cell centres are taken as *donor* point locations. Figure 5.1 shows some examples for the selection of six *donor* points.

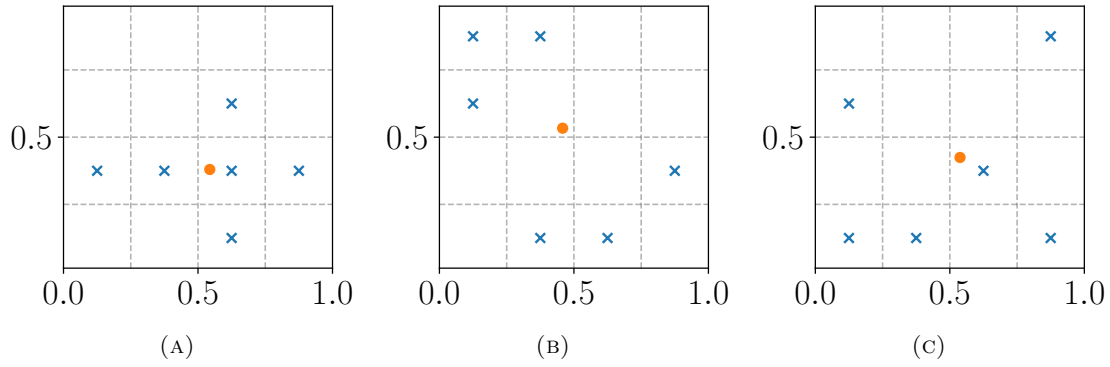


FIGURE 5.1: Example of randomly selected *donor* points on a grid used in this study using the *Cartesian* method. The orange dot is the reconstructed interpolation location.

The second method is meant to replicate non Cartesian, unstructured meshes and will be referred to as *Random* in the following. The same process for generating a 'virtual' grid is done, but this time instead of taking the grid cell centre as *donor* point, a random point at a minimum distance of 1/3 of the faces is taken. This method is purposefully not completely random in the *donor* point placement because cell centres in a real CFD mesh are, for example, never clustered close together. Figure 5.2 shows the result of this method when the chosen cells on the grid are the ones from Figure 5.1.

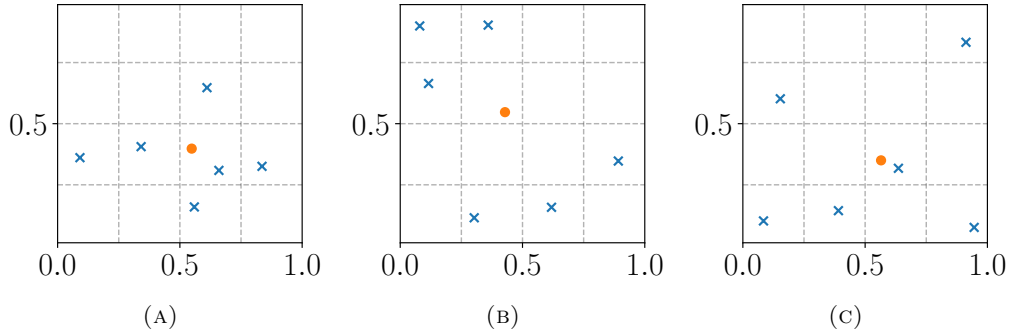


FIGURE 5.2: Example of randomly selected *donor* points based on the *Random* method. The reconstructed interpolation location is denoted by the orange dot.

5.1.1.2 Error estimation

Instead of solving fluid equations, the field to interpolate is describe by f as follows:

$$f(x, y) = \sin(-2x) + \cos\left(\frac{3y}{2}\right) \tag{5.1}$$

This function f was chosen because it is bounded between -1 and 1, it does not show any oscillation in the $[0, 1] \times [0, 1]$ domain and is constructed from non-polynomial functions. Figure 5.3 shows this function in 3D plot.

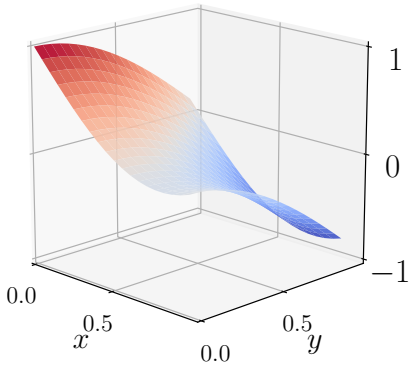


FIGURE 5.3: Function f to be interpolated.

In order to compute the error of a particular scheme given a set of *donor* points, an interpolation location is recreated from the *donor* point locations. To avoid extrapolation the interpolation location is set as the centre of the *donor* points (mean x and y values) plus a random offset between 0 and 0.05 in both directions. The error is then the difference between the interpolated value at that location and the evaluation of f also at that location as shown with Equation 5.2. The magnitude of the error for a given scheme and set of *donor* points is not meaningful by itself because it depends on the function f chosen. It is however used here to compare schemes and *donor* point

locations relative to each other. Several functions f were tested and the conclusions drawn in the following section were not affected by this change.

$$\begin{aligned} x_{loc} &= mean(x_i) + rand(0, 0.05) \\ y_{loc} &= mean(y_i) + rand(0, 0.05) \\ e &= |f(x_{loc}, y_{loc}) - f_{interp}(x_{loc}, y_{loc})| \end{aligned} \tag{5.2}$$

5.1.2 Results

Figure 5.4 summarises the error distributions for each scheme. For each histogram, 50 000 sets of *donor* points were generated using the *Random* method, and the distribution of the log of the error is shown. The median error is also displayed by a vertical line. Finally, the standard deviation of the log of the error as well as the percentage of interpolation returning an error higher than 1 is displayed below each histogram.

From Figure 5.4, several observations can be made. Firstly, for each scheme, increasing the polynomial degree decreases the error. Among the three schemes, the *Least squares* interpolation performs the best, as the error median is either similar or lower, and the distribution is more concentrated around the median. This can also be seen with the lower standard deviation, indicating that interpolation errors higher than the median are less likely. Furthermore, among all the random sets of *donor* points drawn, none of them yield high errors, while a small percentage of the *Polynomial* and *Polynomial tensor* ones result in errors greater than 1. One of the characteristics of the *Least squares* method is the overdetermination of the interpolation, which means that the number of *donor* points can be freely determined as long as it is larger than the number of unknowns in the reconstructed polynomial function. To this end, the donor point multiplier (C_{mult}) can be set. This study is shown in Figure 5.5, presenting, for a range of donor point multipliers, the associated histogram of error for a degree 2 *Least squares* scheme. Using more *donor* points tends to slightly reduce the median error in this case, this conclusion cannot however be generalised to other support functions f (not shown here). Nevertheless, a general conclusion can be drawn: by increasing C_{mult}, the spreading of the errors higher than the median reduces leading to a more robust interpolation. Moreover, using only one more *donor* point than the *Polynomial* interpolation ($N = 7$) already shows some positive effects with median error reduced from $10^{-2.03}$ to $10^{-2.19}$.

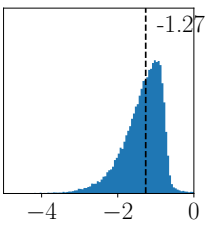
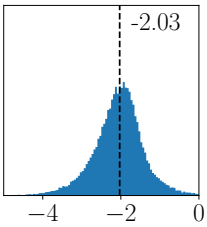
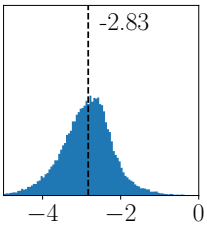
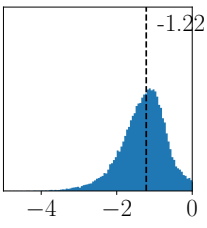
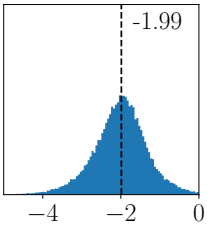
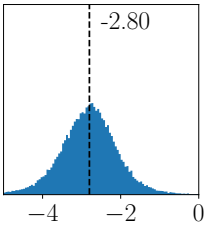
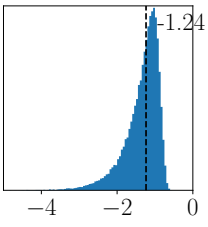
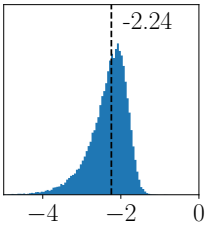
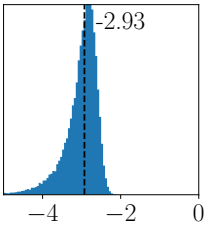
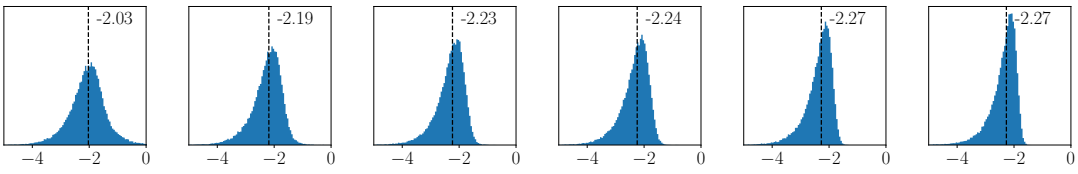
	Degree 1	Degree 2	Degree 3
Polynomial			
Standard deviation	0.544	0.660	0.699
% of data with error > 1	0.3 %	0.4 %	0.1 %
Polynomial Tensor			
Standard deviation	0.682	0.720	0.751
% of data with error > 1	3.5 %	0.7 %	0.1 %
Least Squares			
Standard deviation	0.470	0.506	0.438
% of data with error > 1	0.0 %	0.0 %	0.0 %

FIGURE 5.4: Histogram of the log of errors when *donor* points are created using the *random* method. The log of the median error is displayed with dotted lines and the *Least squares* results were done with a donor point multiplier of 1.5. This means that, for the same degree, the *Least squares* results use 1.5 times more *donor* points than the *Polynomial* ones.



Pol degree 2 $N = 6$ | $C_{mult} = 1.2$ $N = 7$ | $C_{mult} = 1.3$ $N = 8$ | $C_{mult} = 1.5$ $N = 9$ | $C_{mult} = 2.0$ $N = 12$ | $C_{mult} = 2.5$ $N = 16$

FIGURE 5.5: Influence of the donor point multiplier (C_{mult}) on the error distribution. Histograms show the log of the errors for degree 2 *Least squares* interpolations. The leftmost histogram is a *Polynomial* interpolation as it is mathematically equivalent to a *Least squares* interpolation with $C_{mult} = 1$ (and $N = 6$).

When generating *donor* points using the *Cartesian* method, error histograms look sensibly similar to the *Random* method but present a larger proportion of very high errors. This is illustrated by Table 5.1 displaying the percentage of interpolations leading to errors higher than 1. For instance, the *Polynomial* interpolation results in around 10% of

high errors, and the percentage of high errors for the *Polynomial tensor* varies between 30% to 70%. Only the *Least squares* approach is robust when the donor point multiplier is high enough. A value of 1.5 is, in this case, sufficient to prevent any divergent results, though, even a donor point multiplier of 1.3 shows a very small percentage of high error results.

TABLE 5.1: Percentage of interpolations resulting in errors higher than 1 for each scheme.

	Method	Degree 1	Degree 2	Degree 3
Polynomial	*Rand*	**0.3 %**	**0.4 %**	**0.1 %**
	Cart	**9.6 %**	**13.0 %**	**10.7 %**
Polynomial tensor	*Rand*	**3.5 %**	**0.7 %**	**0.1 %**
	Cart	**29.8 %**	**53.2 %**	**68.1 %**
Least squares $C_{mult} = 1.5$	*Rand*	0.0 %	0.0 %	0.0 %
	Cart	0.0 %	0.0 %	0.0 %
Least squares $C_{mult} = 1.3$	*Rand*	0.0 %	0.0 %	0.0 %
	Cart	**0.5 %**	**0.1 %**	0.0 %

To better understand the root of these high errors, Figure 5.6 shows examples of *donor* point locations leading to either high or low errors when using the *Random* or the *Cartesian* method. It then becomes clear that, in both cases, high errors are associated with sets of *donor* points that show spacial collinearity. This also explains why the *random* method leads to far less high errors when compared to *Cartesian* one.

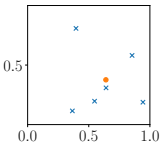
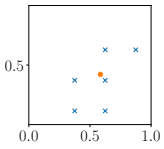
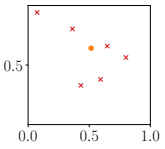
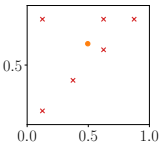
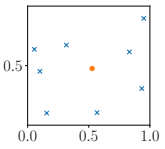
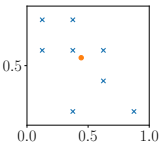
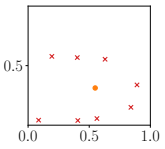
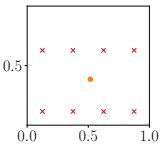
	Low error		**High error**	
	Random	*Cartesian*	*Random*	*Cartesian*
Polynomial				
Interpolation error	2.52×10^{-05}	7.46×10^{-06}	$2.13 \times 10^{+02}$	$9.38 \times 10^{+44}$
Least squares				
Interpolation error	5.03×10^{-07}	7.48×10^{-07}	7.76×10^{-02}	Singular matrix

FIGURE 5.6: Example of *donor* point locations leading to low or high interpolation errors.

In a real CFD context, it can be important to detect high interpolation errors. These errors can be correlated with the condition number (κ) of the system of linear equations used to perform the interpolation. The condition number measures the sensitivity of the system to variations in the right-hand side of the equations. Figure 5.7 shows the relationship between interpolation errors and the condition number. In all of the plots, high interpolation errors are consistently associated with high condition numbers. Furthermore, the distinct separation between low and high errors and low and high condition numbers makes it easy to use the condition number as a criterion for detecting

high interpolation errors. There are only a few 'false positives' where low errors are associated with relatively high condition numbers in the case of degree 3 polynomial interpolation.

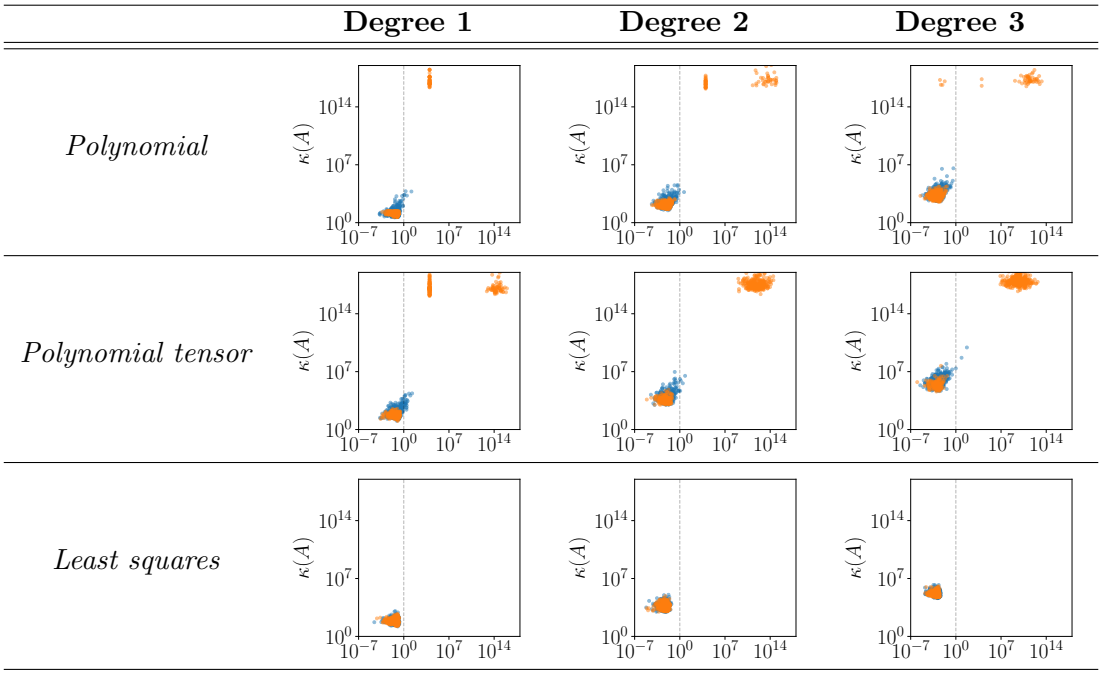


FIGURE 5.7: Interpolation error against condition number κ of the system. *Least squares* interpolation uses a C_{mult} of 1.5. Blue data points are using the *Random donor* point location method and orange ones the *Cartesian* method.

5.1.3 Conclusions

Studying polynomial based schemes in isolation allowed to make statistical analysis of the interpolation error and link it to *donor* point locations. *Polynomial* and *Polynomial tensor* schemes are seen to have worse robustness compared to the *Least squares* approach, showing both larger median errors as well as a larger spread of the errors. Moreover, particularly on structured meshes, they more often lead to very high errors due to poor system conditioning. With the *Least squares* scheme, on the other hand, robustness can be controlled by the number of *donor* points used for the interpolation, increasing it resulting in smaller spread of the errors. In the cases presented in this section, a donor point multiplier of 1.5 is enough to remove any diverging results, but adding a single *donor* point ($C_{mult} = 1.2$) is already enough to improve the robustness over a *Polynomial* interpolation.

The robustness is directly linked with the *donor* point locations, their alignment worsening the conditioning and leading to higher errors. In a CFD context, cell centre alignments are quite common, with structured meshes or in the prism layer for example. Since the interpolation method has to be resilient to such cases the *Least squares* is the best candidate amongst the three polynomial based schemes tested. Finally, the

condition number of the system to solve could be used to detect, prior to computing the interpolation, whether the error will be high or not.

5.2 Code Verification of interpolation schemes

Prior to making overset computations, it is important to test interpolation schemes and the *donor* search method implementation to verify the error produced and their order of convergence. In this section, like in the previous one, analytical fields are interpolated, but, this time, real meshes are used together with the complete CFD solver. The *donor* search as well as the interpolation scheme computations are the ones used in the overset implementation even though overset meshes are not used here.

5.2.1 Methodology

The domain is a 2D square of size 2×2 and five Cartesian meshes (16×16, 32×32, 64×64, 128×128 and 256×256 named G1 to G5) were generated. The field to interpolate represents a Taylor-Green vortex solution. This set of equations are often used in Verification studies because they are solutions to the Navier-Stokes equations, though, in this section, such property is not required as no simulation is performed, instead, the fields have been selected only as example fields for the velocity and pressure components. The analytical equations are set up in each cell centre as per Equation 5.3.

$$\begin{aligned} U_x &= -\cos(\pi x)\sin(\pi y) \\ U_y &= \sin(\pi x)\cos(\pi y) \\ P &= -\frac{1}{4}\left(\cos(2\pi x) + \cos(2\pi y)\right) \end{aligned} \tag{5.3}$$

The interpolations are performed in 500 pre-determined randomly picked locations of the grid, and used for each scheme tested and each of the five grids. Moreover, a minimum distance to the boundaries of the domain of 0.25 was prescribed when generating the interpolation locations to prevent the *donor* cell search to be affected by the boundaries. Figure 5.8 shows the coarsest grid and the interpolation locations.

Each scheme presented in Chapter 4 section 4.4 is tested, *Polynomial* and *Least squares* up to a degree 5 and *Polynomial tensor* up to degree 4. Finally, the *Least squares* schemes used $C_{mult} = 1.5$, and the *Inverse distance* scheme used 10 *donor* cells.

For each grid and interpolation scheme, $N_j = 500$ interpolations are performed and the L_∞ norm of the error is recorded.

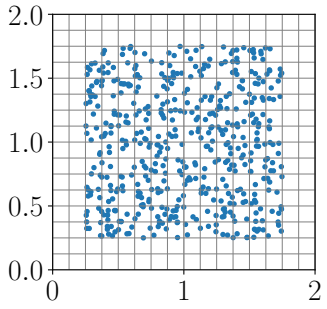


FIGURE 5.8: Locations of the interpolations. 500 different locations randomly picked at a minimum distance of 0.1 to the domain boundary (the coarsest grid of 16×16 is displayed).

5.2.2 Results

Figure 5.9 shows the L_∞ norm of the error and the grid convergence order for each scheme. It is worth noting that the *Polynomial tensor* scheme produced unreliable results due to its lack of robustness, ranging from a lack of grid convergence (degree 1) to diverging results and very high errors for degree 2 to degree 4. In this case, the Cartesian nature of the mesh used is a worst-case scenario for this type of scheme. In contrast, all of the other schemes showed converging trends with grid refinement, each achieving the convergence order predicted by their mathematical description. Furthermore, the *Barycentric* interpolation behaved like a 2^{nd} order scheme regardless of its type. It is also worth mentioning that schemes with the same order of convergence tend to produce similar error levels, with the *Polynomial* scheme resulting in slightly lower errors compared to the *Least squares* scheme.

Overall, this study verifies the correct implementation of the schemes used in this work, and provides further insight into the poor robustness of the *Polynomial tensor* interpolation.

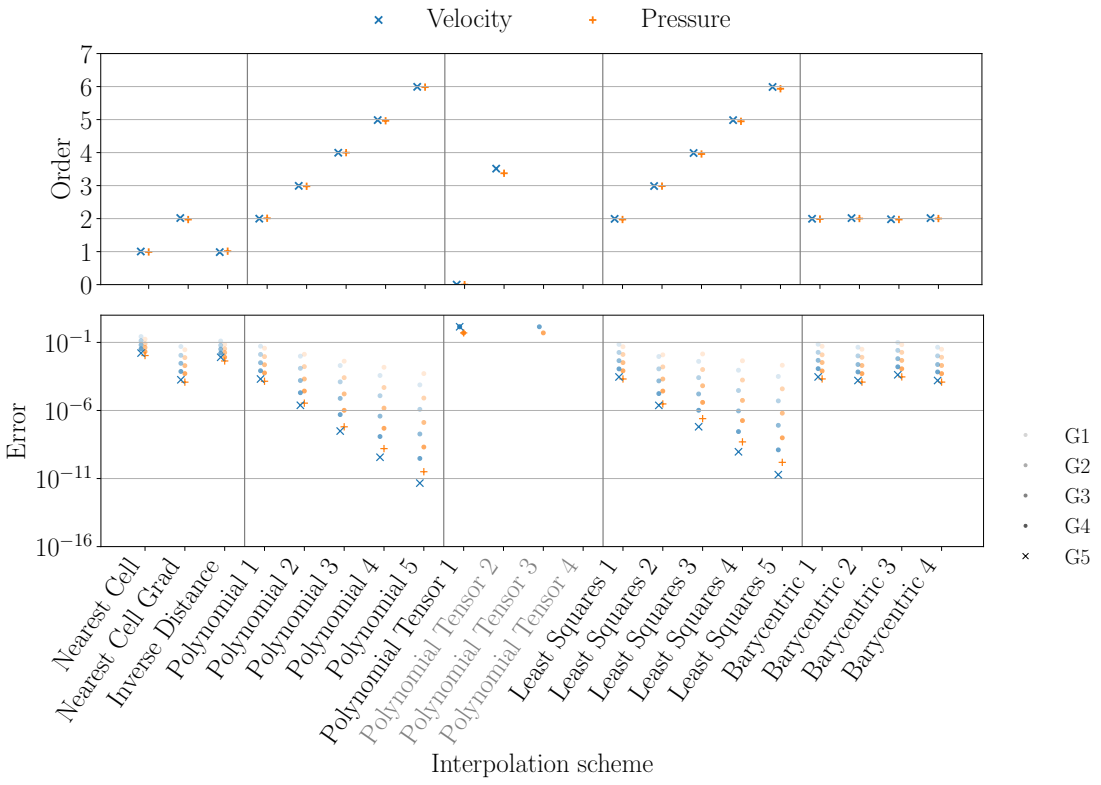


FIGURE 5.9: L_∞ norm of the error and associated convergence order for the different interpolation schemes.

Chapter 6

Code Verification and error analysis on flows with analytical solution

After testing the different interpolation schemes in isolation in Chapter 5, overset computations are here run to perform code Verification and analyse errors generated by the overset interface and their influence on the results. As seen in Chapter 3, in a CFD computation, discretisation and interpolation errors are only accessible when the exact solution of the problem is known. To achieve this objective the present Chapter uses two methods on two distinct test cases; the first one is a 2D steady computation of the Poiseuille flow, a low Reynolds number pipe flow for which an analytical solution to the Navier-Stokes equations is known. The second case is more complex and closer to real-life maritime applications as it resembles a high Reynolds number flat-plate flow together with a pulsating recirculation bubble. This test case, designed by Eça et al. [35], involves an unsteady RANS manufactured solution including a one equation turbulence model [117]. In both cases, errors can be probed in the entire domain and mass imbalance quantified. Besides testing the different interpolation schemes, several meshes and cell types, and meshes relative orientations are considered. Always having in mind space and time refinements, as it is crucial for a proper Verification study. Finally, as it is possible for the cases studied, overset grid solutions will be compared to computation done on single grids, without overset.

6.1 Poiseuille flow test case

6.1.1 Case definition

The *Poiseuille* flow analytical formulation can be derived directly from the Navier-Stokes equations. Considering a 2D domain of length L in the x direction and height h in the y direction with the inlet at $x = 0$, outlet at $x = L$ and walls at $y = 0$ and $y = h$. After taking into account the different boundary conditions (non slip walls, and $P = 1$ at the outlet), the analytical solution is presented in Equation 6.1.

$$U_x(y) = \frac{Ph}{2\mu} y \left(1 - \frac{y}{h}\right) \quad , \quad U_y = 0 \quad , \quad \frac{\partial p}{\partial x} = -P \quad , \quad p(x) = -P(x - L) + 1 \quad . \tag{6.1}$$

In this study, the constant pressure gradient is $\partial p/\partial x = -8\mu$, the domain height is unity ($h = 1$), the length of the domain is two ($L = 2$) and the kinematic viscosity is $\nu = 0.1$, resulting in a height based Reynolds number of $Re_h = 10$. Figure 6.1 shows U_x as defined by Equation 6.1.

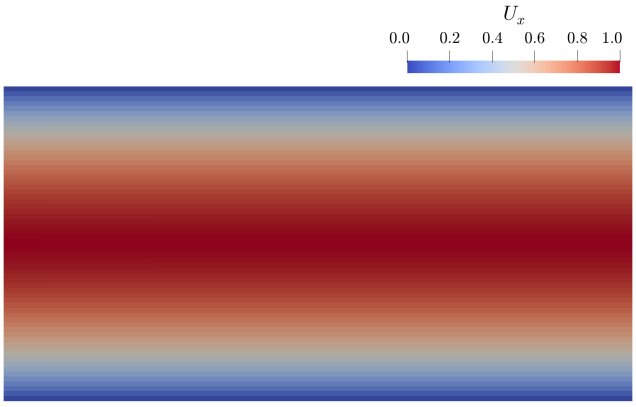


FIGURE 6.1: Exact axial velocity field for the Poiseuille flow test case.

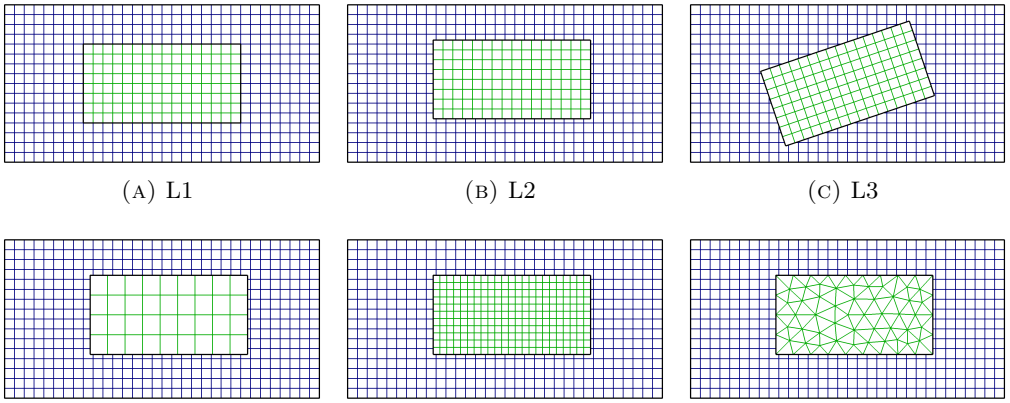
In order to test the interpolation in different conditions, six overset mesh layouts were designed. These are meant to represent various aspects of real overset meshes, with variation in refinement, relative positioning of the meshes and grid topology. For each layout, five grid assemblies were generated with different refinements. Each grid assembly is composed of two grids:

1. a Background grid common to all layouts, which is a Cartesian grid of ratio 2:1, as per Table 6.1
2. and a Foreground grid, also a 2:1 rectangular grid but four times smaller than the Background grid. Its position and type is different for each layout:
    - L1: the Foreground grid is a Cartesian grid on which the cell size is the same as the one of the Background grid, it is also centred in both directions relative

to the Background grid. In the overlap region there is a one-to-one match between cells of both grids.

- L2: the Foreground grid is similar to layout L1 but the global positioning of the grid is offset by about 0.694 cells in the x-direction and 0.416 cells in the y-direction (the absolute offset depending on the grid refinement).
- L3: the Foreground grid is similar to layout L1 but it is rotated by about 18.65 degrees around the centre of the domain (the decimals were randomly chosen in order to avoid cell alignments).
- L4: The Foreground grid is still Cartesian but the cells are 1.6 times larger in each direction compared to the Background grid. The grid has the same offset from the centre of the domain as L2.
- L5: The Foreground grid is 1.5 times finer in both directions than the Background grid. The grid has the same offset from the centre of the domain as L2.
- L6: The Foreground grid is unstructured composed of triangles, and its cell count is kept close to the cell count of L1. The Foreground grid has the same offset from the centre of the domain as L2.

It can be noted that, even if some of them are structured, the CFD solver considers them as unstructured and does not take advantage of their topology with the *donor* search or interpolation as presented in Chapter 4.



(A) L1 (B) L2 (C) L3

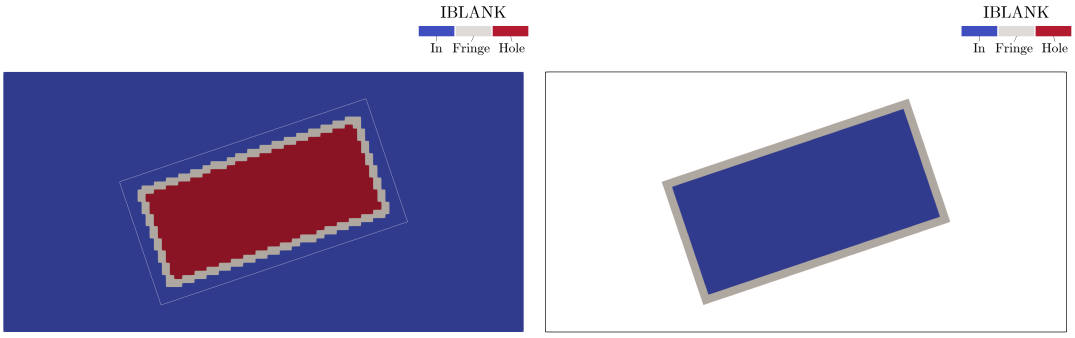
(D) L4 (E) L5 (F) L6

FIGURE 6.2: Coarsest grid used for each layout. The Background grid is displayed in blue and the Foreground grid in green.

In this test case, steady computations are performed where only the momentum equation and the pressure correction (via the SIMPLE method) are solved. At the inlet, the analytical velocity profile is set, and the analytical pressure is enforced at the outlet. The top and bottom of the domain use non-slip wall boundary conditions. Iterative convergence is ensured by letting the infinity norm of the residuals fall below 10^{-14} and convective and diffusive fluxes are discretised using 2^{nd} order schemes.

TABLE 6.1: Different grid sizes used for the Poiseuille flow test case.

	Background L{1-6}		**Foreground** L1 - L2 - L3		L4		L5		L6	
		N_i		N_i		N_i		N_i		N_i
G1	32×16	512	16×8	128	9×4	36	23×11	253	unstr	112
G2	64×32	2 048	32×16	512	19×9	171	47×23	1 081	unstr	518
G3	128×64	8 192	64×32	2 048	39×19	741	95×47	4 465	unstr	2 232
G4	256×128	32 768	128×64	8 192	79×39	3 081	191×95	18 145	unstr	9 234
G5	512×256	131 072	256×128	32 768	159×79	12 561	383×191	73 153	unstr	37 944



(A) Background grid (B) Foreground grid

FIGURE 6.3: Cell status for the layout L3 and grid G3 computed by Suggar++.

Figure 6.3 shows the IBLANK array for the L3 layout computed by Suggar++. Two layers of *fringe* cells are used at each boundary for the CFD solver to reconstruct gradients properly on neighbouring *in* cells. Since the test case does not strictly require the use of overset meshes (there is no moving body, etc.) for each refinement, a computation using only the Background grid was additionally done for comparison purposes, it will be called *no overset* in the following.

In this study, all the interpolation schemes presented in section 4.4 were tested: *Nearest cell* and *Inverse distance* using 4 and 10 *donor* cells as well as *Barycentric* interpolation of type 2. For polynomial based interpolation, *Least squares* and *Polynomial* interpolation of degree 1, 2 and 3, as well as *Polynomial tensor* interpolation of degree 1 were used. It should be noted that since the Poiseuille flow solution is a degree 2 polynomial function, a degree 2 or higher polynomial based interpolation method should be able to interpolate the field exactly.

As seen in Chapter 5, the robustness of polynomial based interpolation is heavily influenced by *donor* cell locations and is particularly sensitive to cell centre alignments. In this test case, *Polynomial* interpolation of order 3 and the *Polynomial tensor* interpolation lead to diverging computations due to their high errors. Subsequently, results from these computations are not shown in the following study.

6.1.2 Error level analysis

Figure 6.4 shows the infinity norm of the error on the velocity field for a selection of schemes (and the layout L3) against grid refinement. For the *no overset* computation, this error is only the space discretisation error, that, as expected with 2nd order discretisation, also decreases in 2nd order when refining the grid. On overset computations, however, the error shown is a combination of discretisation and overset interpolation errors. With the *Least squares* 2 scheme (a 3rd order scheme), the error level is similar to the *no overset* computation, suggesting that the interpolation error is lower than the discretisation error. To analyse errors on all of the schemes and the different layouts, Figure 6.5 shows a different representation of the same data with the infinity norm of the error for each grid per scheme as well as the order of convergence.

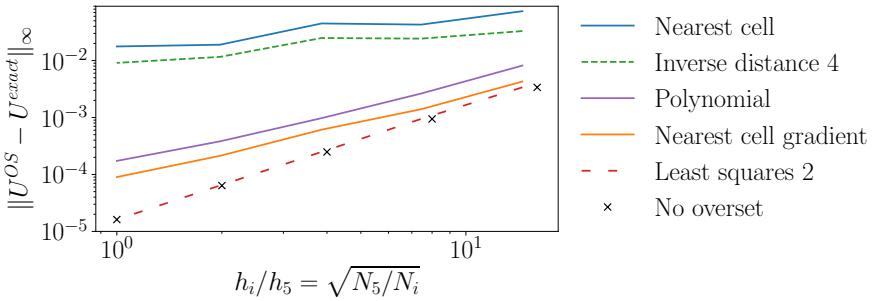
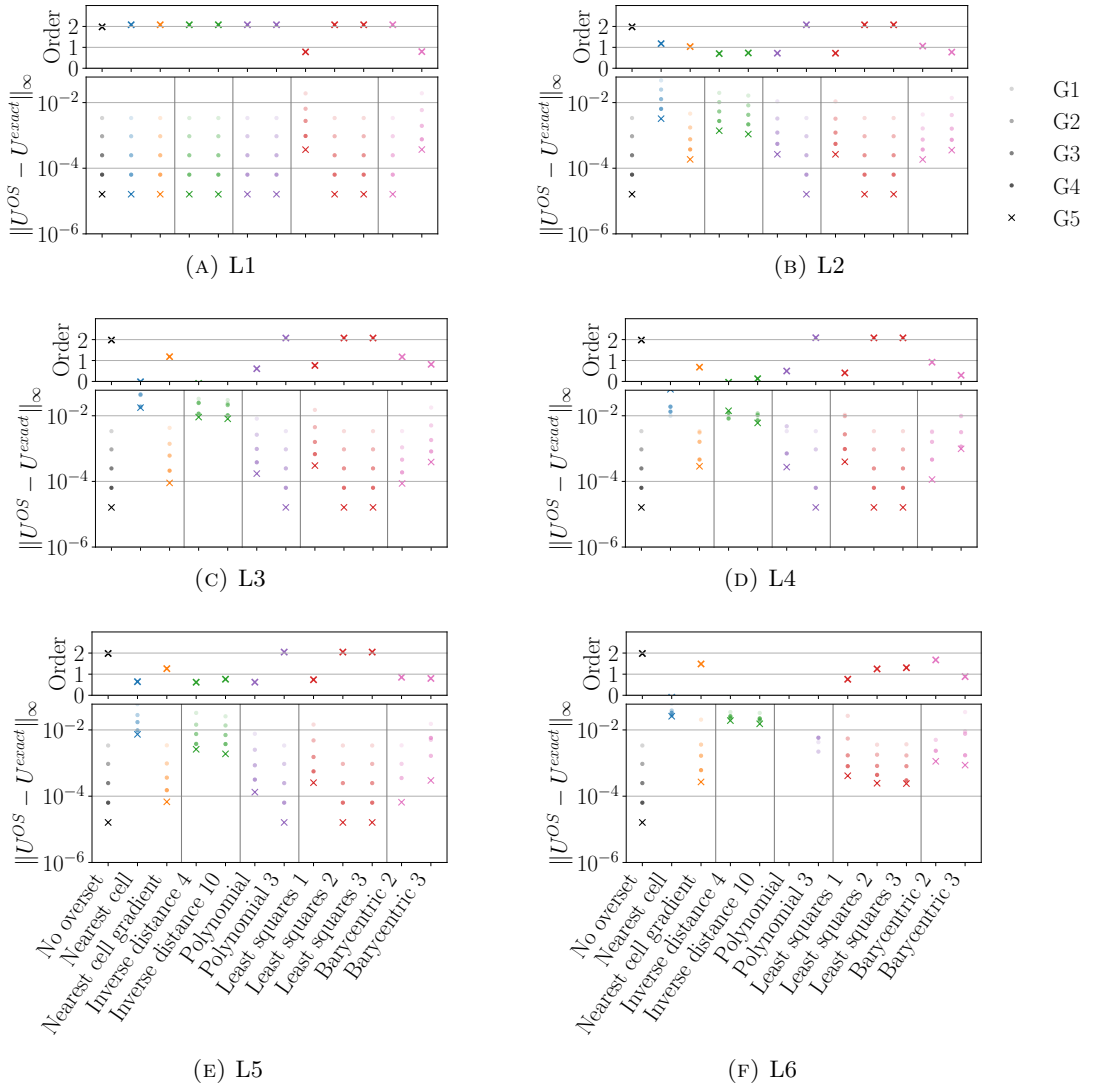


FIGURE 6.4: Infinity norm of the velocity field error against grid refinement (G5 to G1) for the layout L3. For readability reasons, only a selection of schemes is displayed.

- Layout L1: each scheme shows a converging trend with errors similar to the case run without overset except for the degree one *Least squares* and *Barycentric* type 3, which both result in higher errors. This is because neither of them guarantee that an interpolation computed on a cell centre returns the cell centre value. Even though only the velocity error is shown here, the error on the pressure has a similar behaviour.

- Layouts L2 to L5 show very comparable results for each scheme, giving confidence that the conclusions drawn here are not specific to the mesh layout. Every single scheme shows a converging trend when refining the grid with the exception of *Inverse distance* and *Nearest cell* on layouts L3 and L4 (both of them being only 1st order). Increasing the number of *donor* cells for the *Inverse distance* scheme slightly helps to reduce the error but not to the level of other, higher order, interpolation schemes. Then, 2nd order schemes (*Nearest cell gradient*, *Polynomial* 1 and *Least squares* 1), as well as the *Barycentric* ones, show intermediate results, with proper convergence when refining the grids, but still showing errors higher than the discretisation error alone (*no overset* case). The convergence order computed from this error varies between 1 and 1.5. Furthermore, degree 1 *Least squares* computations show slightly higher errors than the degree 1 *Polynomial* on these



(A) L1 (B) L2

(C) L3 (D) L4

(E) L5 (F) L6

FIGURE 6.5: Infinity norm of the error and convergence order on the velocity for the Poiseuille case. Cross marker denote the finest grid.

layouts, which can be explained by the fact that the *Least squares* approach uses more *donor* cells, thereby using field information further away from the interpolation location. Finally, schemes with theoretical order of 3 or higher (*Polynomial* 3 and *Least squares* 2 and 3) are the only ones to show error levels similar to the case without overset grids, suggesting that the interpolation error is lower than the discretisation error.

- Layout L6 is the only case introducing an unstructured grid for the Foreground mesh. Since triangular cells reduce the grid quality and increase the intrinsic discretisation error on *in* cells, the comparison with the non-overset case is harder to make. Nevertheless, *Nearest cell gradient* and *Least squares* have both similar error levels for the finest grids, though degree 2 and 3 *Least squares* are better for coarser grids.

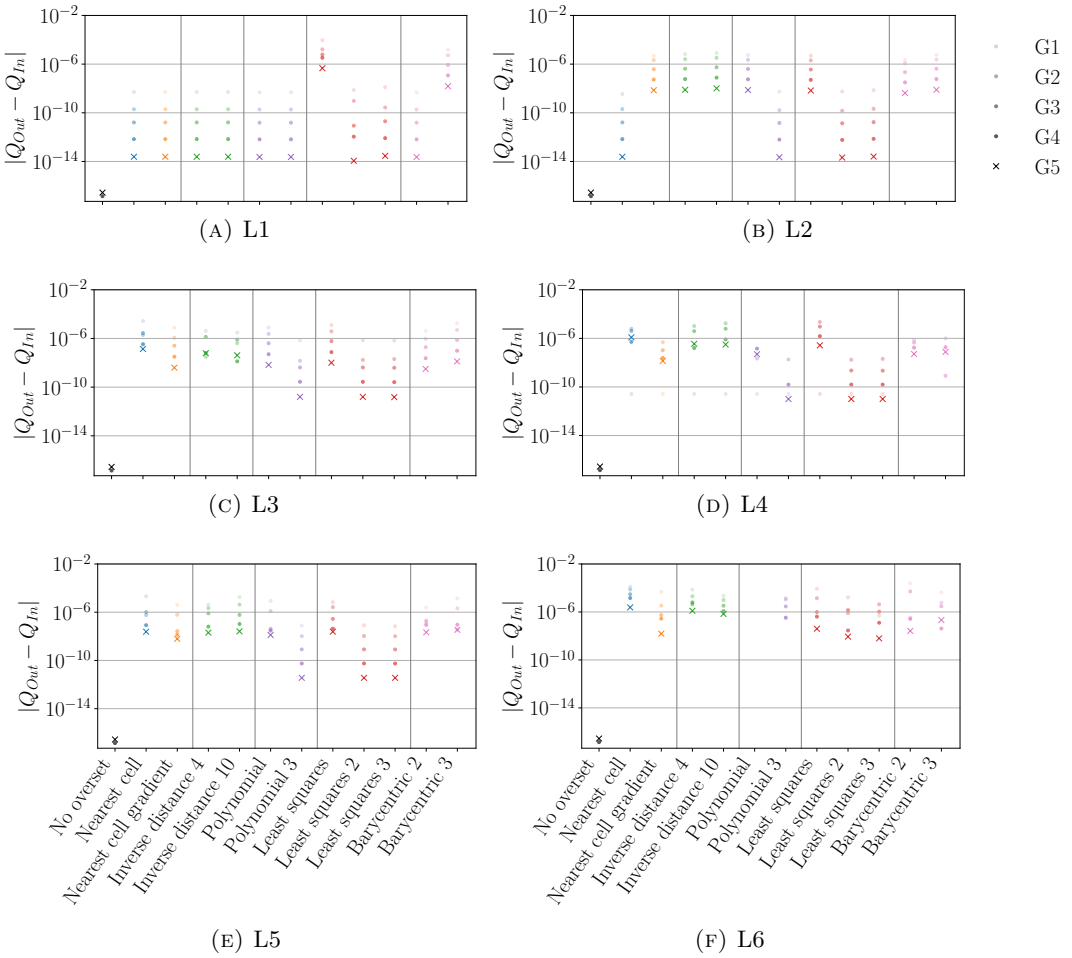
The schemes showing the lowest error overall (*Least squares* 2 and 3) have worse convergence order on this layout. As mentioned, this convergence order combines both discretisation errors and overset interpolation errors, each having their own order of convergence. On these computations, the discretisation error is likely to have a lower order than the interpolation one, thereby lowering the overall order of convergence. However, the *Nearest cell gradient* computation shows a different behaviour as the interpolation error is dominating (with an order of 1.5 like seen on the other layouts). The combined convergence order is 1.5. Upon refining the grids, the discretisation error will likely become dominant, lowering the combined convergence order.
As stated previously, *Polynomial* based interpolation schemes are sensitive to *donor* cells location. On this layout, both *Polynomial* computations resulted in poor robustness with very high interpolation errors leading to the computation divergence.

6.1.3 Mass imbalance study

Upon convergence of the SIMPLE algorithm, mass imbalance, which is the difference between the inflow and outflow mass fluxes, is of the same order of magnitude as the iterative error. When using overset grids, however, the global mass conservation that is inherent to the finite volume method is lost due to the overset interpolation happening on *fringe* cell centres and breaking the uniqueness of fluxes going in and out of every single cell faces. This source of mass imbalance does not depend on iterative error but is rather related to the interpolation error and the overset method itself.

Figure 6.6 shows the mass imbalance for each scheme. It was previously observed that on layout L1, overset computations did not show errors higher than the ones without overset. It is visible here, however, that this does not imply that the mass is fully conserved. It can be noted that for each scheme, the mass imbalance decreases with grid refinement.

The layout L2 is a translation of L1. Therefore the *Nearest cell* scheme is computing exactly the same solution for the two layouts since the *donor* and *fringe* cells are the same, explaining the similar mass imbalance on these two layouts. Furthermore, the mass imbalance is several orders of magnitude lower than the error levels shown in the previous section, which implies that schemes with comparable error magnitudes can have a different mass imbalance. This is, for example, the case for the degree 2 *Least squares* scheme. Even if Figure 6.5 shows similar trends for layouts L2 and L3, the mass imbalance is three orders of magnitude lower on L2 for the finest grids. Similarly to the error plots, it can be seen that schemes with a theoretical order strictly higher than two behave similarly on all grid layouts.



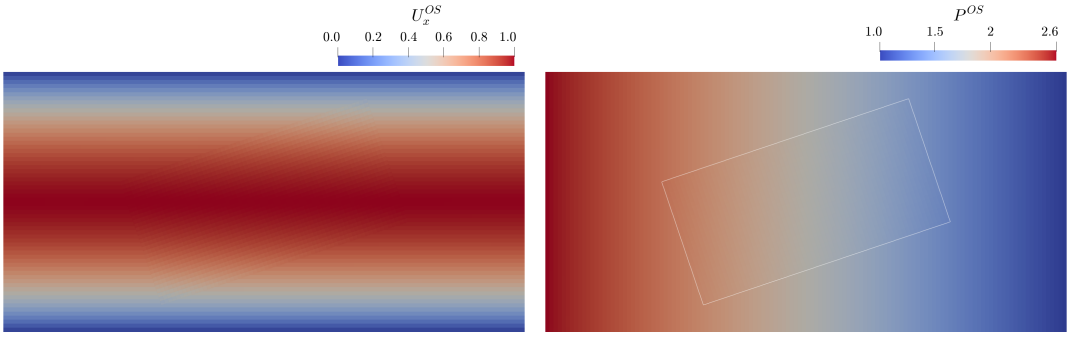
(A) L1 (B) L2 (C) L3 (D) L4 (E) L5 (F) L6

FIGURE 6.6: Difference between inflow and outflow mass fluxes for the Poiseuille case measuring the mass imbalance caused by the overset method.

6.1.4 Flow behaviour and errors location

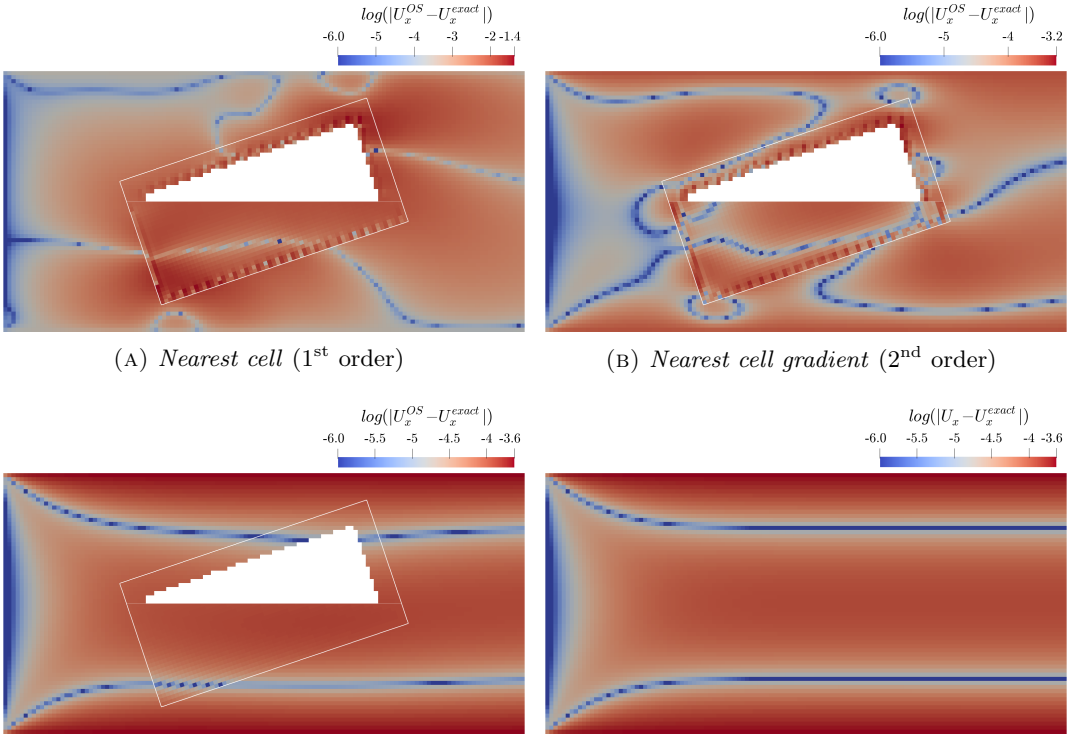
In this section, only the *Nearest cell*, *Nearest cell gradient*, and degree 2 *Least squares* schemes will be analysed as they can be considered representative of 1st, 2nd and 3rd order interpolation schemes respectively. Field visualisation for the degree 1 *Least squares* being relatively similar to the *Nearest cell gradient* one for example. Figure 6.7 shows the velocity and pressure fields for layout L3 and *Nearest cell gradient* scheme, though, from it no artefact can be observed. For analysing errors and the effects of interpolation schemes, comparison with the exact solution is needed. Hence, Figure 6.8 shows, for the layout L3 and set of grids G3, the log of the difference between the exact solution and the overset computation for the velocity field. On these figures, only the bottom half of the Foreground mesh solution is displayed to visualise *fringe* cells of the Background grid that the Foreground mesh would otherwise overlap.

Firstly, outside of the overlap region a couple of error patterns can be discussed. As the velocity is prescribed at the inlet, it is where its error is minimal. Then, the two 'blue



(A) Axial velocity field (B) Pressure field

FIGURE 6.7: Velocity and pressure fields for the layout L3 and grid set G3, using the *Nearest cell gradient* interpolation scheme.



(A) *Nearest cell* (1st order) (B) *Nearest cell gradient* (2nd order)

(C) *Least squares* degree 2 (3rd order) (D) Computation without overset

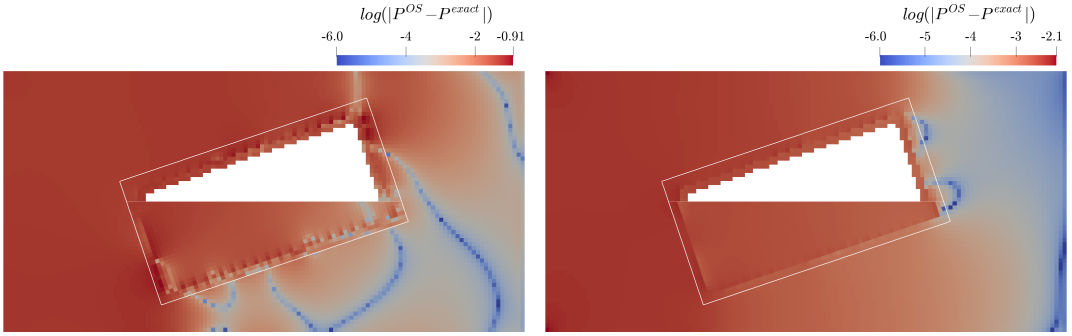
FIGURE 6.8: Log of the error between overset computations and exact solution for the velocity field. Note that the scale is adapted for each plot with the maximum error always being the upper range of the scale. The Foreground mesh is only half visible in order to visualise *fringe* cells of the Background mesh.

lines' appearing on the visualisation without overset only show a change of sign of the error which is maximal close to the walls and in the centre of the domain.

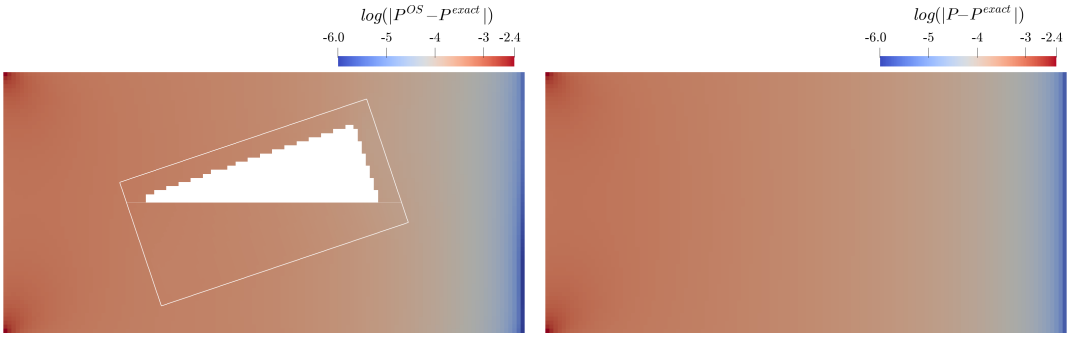
When focusing on the overset method and comparing the different interpolation schemes, *Nearest cell* and *Nearest cell gradient* show similar error patterns: *fringe* cells have higher errors and can clearly be distinguished from the rest of the flow even if the computation using the *Nearest cell gradient* scheme has lower errors than the one using the *Nearest cell* scheme. With the degree 2 *Least squares* scheme, on the other hand, no discontinuities are visible even in the overlap region. The highest errors are at the top and bottom boundaries and not close to the overset interface. For this scheme, the errors are quite similar in terms of magnitude and location to a computation done without overset suggesting that the interpolation error is lower than the discretisation one in the entire domain. When looking at the pressure field, Figure 6.9, comparable conclusions can be drawn as degree 2 *Least squares* scheme does not display any artefact related to the overset process. It can be noted that the errors generated by the *Nearest cell gradient* scheme on the pressure result in a smoother field than for the velocity, but a step is still present between *fringe* and *in* cells. Visible on both the Foreground and the Background mesh. Finally, the *Nearest cell* scheme shows point-to-point oscillations on *fringe* cells for both grids and presents higher errors.

The Background grid data can also be directly compared to the computation done without overset as shown in Figure 6.10 for the velocity field. For the *Nearest cell* and *Nearest cell gradient*, the maximum difference is similar to the error analysed above. In contrast, the *Least squares* 2 figure shows differences one order of magnitude lower than the error (computed when comparing against the analytical solution) and still has a smooth field without visible effects of the overset method. This suggests that the interpolation error is one order of magnitude lower than the discretisation error.

To summarise, artefacts of the overset method, specifically of the interpolation step, are visible for interpolation schemes of order 2 or lower. When using higher order schemes, the error made with overset grids is negligible compared to discretisation error without overset grids.

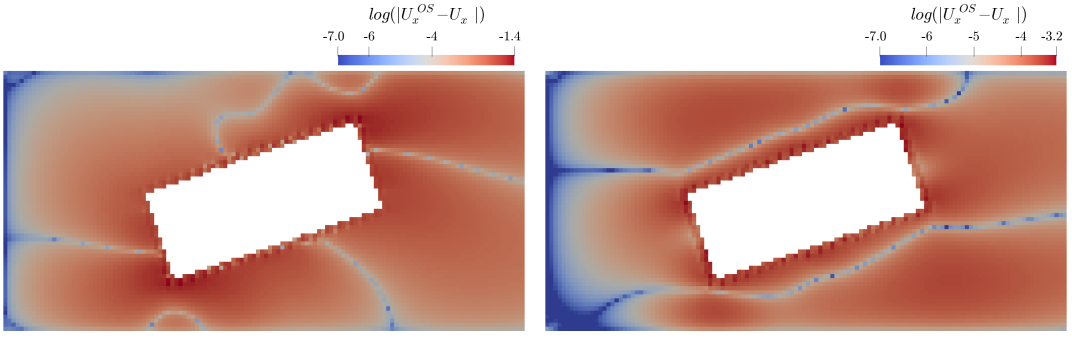


(A) *Nearest cell* (1st order) (B) *Nearest cell gradient* (2nd order)

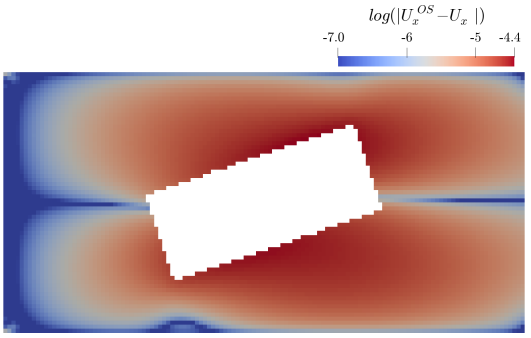


(C) *Least squares* degree 2 (3rd order) (D) Computation without overset

FIGURE 6.9: Log of the error between overset computations and exact solution for the pressure field.



(A) *Nearest cell* (1st order) (B) *Nearest cell gradient* (2nd order)



(C) *Least squares* degree 2 (3rd order)

FIGURE 6.10: Log of the difference in velocity between a computation done with and without overset.

6.2 Recirculation bubble URANS manufactured solution

6.2.1 Introduction

This test case, designed by Eça et al. [35], is a 3D unsteady manufactured solution of a high Reynolds number ($Re = 10^7$) recirculation bubble. Even though no constrains formally exist in the design of a manufactured solution, Eca et al. focused on designing a realistic unsteady high Reynolds number boundary layer to assess the discretisation errors and turbulence models in conditions close to real engineering problems. In the present work, the Verification of the CFD solver discretisation is not the goal, however, the use of this particular manufactured solution allows a detailed study of interpolation error generations and propagations in a realistic engineering context and at Reynolds numbers the method encounters in maritime applications.

6.2.2 Case definition

The manufactured solution defines the velocity, pressure and $\tilde{\nu}_t$ turbulent eddy viscosity in the entire domain and the bubble grows and disappears periodically following a sine wave with a period $T = 5$. A slice of the velocity field can be seen in Figure 6.11 with the inlet on the left and outlet on the right. Finally, a detailed description of the solution can be found in Appendix B and Eça et al. [35] should be read for an explanation of the design choices that lead to the different fields.

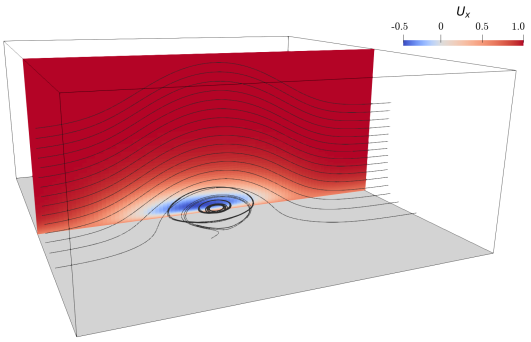


FIGURE 6.11: Recirculation bubble used as a manufactured solution, the slice is coloured by the x axial velocity. On this representation, the inlet is on the left, non-slip wall at the bottom and outlet on the right.

The domain is an empty box of size $x = [0.1; 1]$, $y = [0; 0.4]$, and $z = [0; 1]$. The bottom boundary is a non-slip wall, the inlet a Dirichlet boundary condition with each quantity set to their analytical formulation, and every other boundary uses a Neumann boundary condition with the exact analytical gradient set for each quantity.

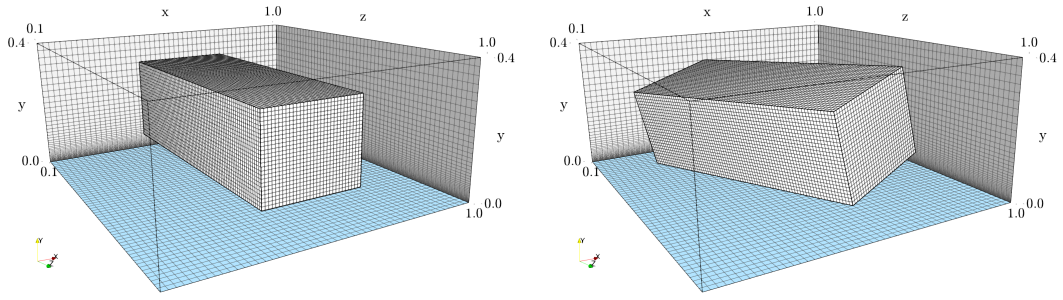
For this test case, two different overset mesh layouts were tested. In both of them the Background grid of the size of the entire domain is used together with a smaller

Foreground grid. The Foreground grid is a Cartesian mesh of size $0.3 \times 0.25 \times 0.8$ and the Background grid is a structured mesh with a refinement towards the wall to reach a y^+ below 1 for all grids. In this study, five different refinements were used for both the Background and Foreground meshes, and Table 6.2 summarises their dimensions and cell counts. The name of each grid layout (*Grid n*) can be seen as the number of cells in the x-direction for the Background grid, and every other dimension scales with it in order to have a set of geometrically similar grids.

The difference between the two overset layouts is only in the rotation of the Foreground grid, on layout L1, the Foreground grid is in the centre of the domain, and aligned with the global axis. When on layout L2, the Foreground grid is also centred in the domain but two rotations are applied to it. First in the y-direction of 18.90 degrees and then of 12.89 degrees in the z-direction. Figure 6.12 shows the coarsest mesh for both layout and Figure 6.13 displays the associated IBLANK information. Similarly to the Poiseuille test case, two layers of *fringe* cells are placed at the boundary of the Foreground mesh and outside *hole* cells on the Background mesh.

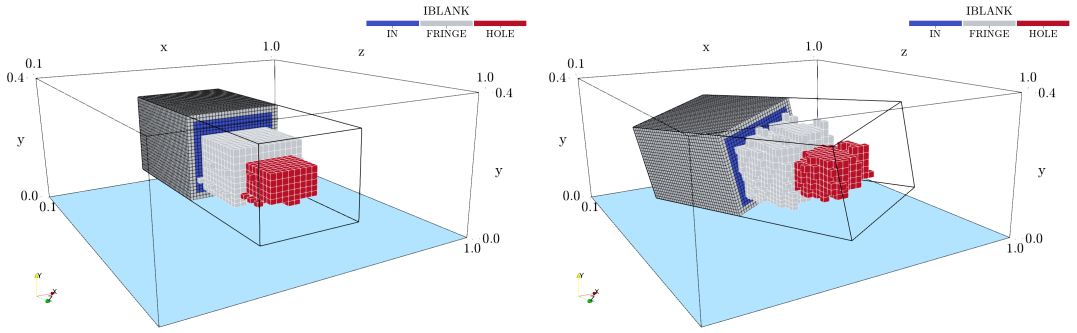
TABLE 6.2: Details of the different grids used for the recirculation bubble test case.

	Background		**Foreground**	
	$n_x \times n_y \times n_z$	N_i	$n_x \times n_y \times n_z$	N_i
Grid 50	$50 \times 80 \times 55$	220 000	$30 \times 25 \times 80$	60 000
Grid 60	$60 \times 96 \times 66$	380 160	$36 \times 30 \times 96$	103 680
Grid 70	$70 \times 112 \times 77$	603 680	$42 \times 35 \times 112$	164 640
Grid 80	$80 \times 128 \times 88$	901 120	$47 \times 40 \times 128$	240 640
Grid 90	$90 \times 144 \times 100$	1 296 000	$53 \times 45 \times 144$	343 440
Grid n	$n \times \frac{0.4n}{0.25} \times \frac{n}{0.9}$	$\approx \frac{16}{9}n^3$	$0.6n \times 0.5n \times 1.6n$	$\approx 0.48n^3$



(A) L1 (B) L2

FIGURE 6.12: Coarsest grid (Grid 50) with the two different layouts used for the recirculation bubble test case.



(A) L1 (B) L2

FIGURE 6.13: Overset domain connectivity as computed by Suggar++ on the coarsest grid setup (Grid 50). Black cells belong to the Foreground grid and white cells belong to the Background grid.

For this study, turbulence is modelled by the Spalart-Allmaras model [117] and the pressure-velocity coupling is done by the SIMPLER algorithm as it showed better iterative convergence characteristics than SIMPLE. Time integration is performed by an implicit three time level scheme (2^{nd} order) and the 2^{nd} order QUICK scheme is used for convective fluxes discretisation.

The period of the manufactured solution is set to $T = 5$ and the time-step is $t_{step} = 0.01$, leading to a maximum Courant number of 0.3 for the coarsest grid (Grid 50) and 0.7 for the finest grid (Grid 90). The validity of this time-step was checked by doing a time-step refinement study with $t_{step} = 0.05$, $t_{step} = 0.01$ and $t_{step} = 0.001$. With a time-step of $t_{step} = 0.01$, a period is modeled by 500 time-steps.

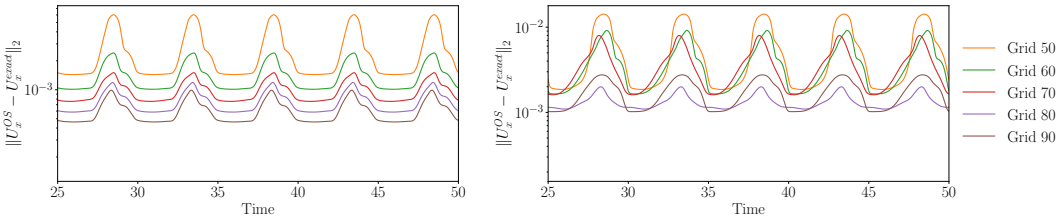
The statistical uncertainty for any measured quantity is kept below 2% by simulating 5000 time-steps (10 periods) and the first two periods of any computation is discarded to remove the startup transient effect (according to the transient scanning technique results).

For this study, the two Layouts L1 and L2, and all the five grid setups were computed using overset. For each grid combination, *Inverse distance* (using seven *donor* cells), *Nearest cell gradient*, *Least squares* of degree 1, 2 and 3, and *Barycentric* interpolation of type 2 were used. For comparison purposes, a set of computations was also done using only the Background grid without overset grids.

Iterative error was controlled by performing enough outerloops to keep the infinity norm of the residuals for all equations below 10^{-6}. The same computations were performed with infinity norm of residuals below 10^{-7} but no changes were seen on the computed error fields suggesting that the iterative error was negligible compared to the discretisation one, therefore 10^{-6} was kept for the remainder of the study.

6.2.3 Time evolution of errors

For this test case, the error norms were calculated every time-step for each field quantity. As an example, Figure 6.14a shows them for the *Nearest cell gradient* on layout L1. On this graph, the five different grid refinements are plotted and all show similar behaviours with the error decreasing with mesh refinement. In time, the error is oscillating with a period equal to the solution period ($T = 5$). As suggested by the low statistical uncertainty, the differences between each period are negligible. Finally, it should be noted that the maximum error occurs just after the middle of the period when the amplitude of the recirculation bubble reaches its maximum size. These observations hold for the two layouts and all the quantities as well as the set of computations done without overset meshes. However, when observing the results coming from the computation using the *Inverse distance* interpolation scheme on layout L1 in Figure 6.14b, the convergence behaviour is less clearly visible. For example, Grid 60 and 70 show similar error levels and the amplitude is also not consistent throughout the refinements.



(A) *Nearest cell gradient* (B) *Inverse distance*

FIGURE 6.14: L_2 norm of the error on U_x over five flow periods on layout L1. In this plot, the timetrace of each grid refinement is shown. Figure (a) shows a proper converging trend when it is not as clear when *Inverse distance* is used.

6.2.4 Error level analysis

Figure 6.15 plots the time averaged L_2 norm of the error for each quantity, each interpolation scheme, each grid setup and both layouts. In this study, the L_2 norm is used as it was found to be more representative of the situation compared to the L_∞ norm because the maximum error was often linked to a domain boundary and not the overset assembly. Considering the error regarding U_x for layout L1 (first column and first line), each scheme shows grid convergence, with finer grids leading to lower errors, except for the *Inverse distance* scheme (in green). Computations done without overset show error levels similar to the *Nearest cell gradient*, degree 2 and 3 *Least squares*, and *Barycentric* interpolation approaches. It is noticeable that the errors without overset are even slightly higher than when using the *Nearest cell gradient* on layout L1. This is because the Foreground grid is more refined than the Background grid. Hence the overset computations lead to lower discretisation errors when compared to the non overset ones. Though this is not as true anymore for layout L2, where the Foreground grid is

rotated, resulting with part of it within the lower boundary layer. In this region, the Background grid is finer than the Foreground one, leading to larger discretisation errors. Overall, for layout L2, the two effects cancel each other out, and the error levels are similar without overset and with a *Nearest cell gradient* scheme.

Consistently for each quantity and both layouts, the *Nearest cell gradient* and *Barycentric* schemes perform as well as the non-overset computations. The degree 2 and 3 *Least squares* interpolation perform here, at best, as well as the *Nearest cell gradient* in layout L1, but slightly worse in layout L2. As mentioned before, on layout L1, the *Inverse distance* scheme shows the highest error levels and no clear grid convergence trend. This implies that the interpolation error is higher than the discretisation error. For layout L2, however, the *Inverse distance* scheme performs better, with lower errors and a converging trend, suggesting that the scheme is more sensitive to the *donor* cells' location than the other tested ones. The degree 1 *Least squares* scheme always performs worse than the degree 2 one, and the degree 2 and 3 *Least squares* schemes perform comparably. Summarising, from the error plots only, the *Nearest cell gradient*, degree 2 and 3 *Least squares* and *Barycentric* schemes are the best candidates as they add no or a minimal amount of error in the field compared to the case without overset. This suggests that the interpolation error is lower than the discretisation error.

Regarding the convergence order, the computations done without overset show a 2^{nd} order convergence for each quantity, as expected from the theoretical order of discretisation of the schemes used, and therefore verifying the non-overset solution. The *Nearest cell gradient*, degree 2 and 3 *Least squares* and *Barycentric* computations also show a 2^{nd} order convergence as well. However, the *Inverse distance* scheme shows, depending on the quantity and layout, between 1^{st} and 2^{nd} order convergence. Finally, the degree 1 *Least squares* results in 2^{nd} order convergence for layout L1 but slightly lower order on L2.

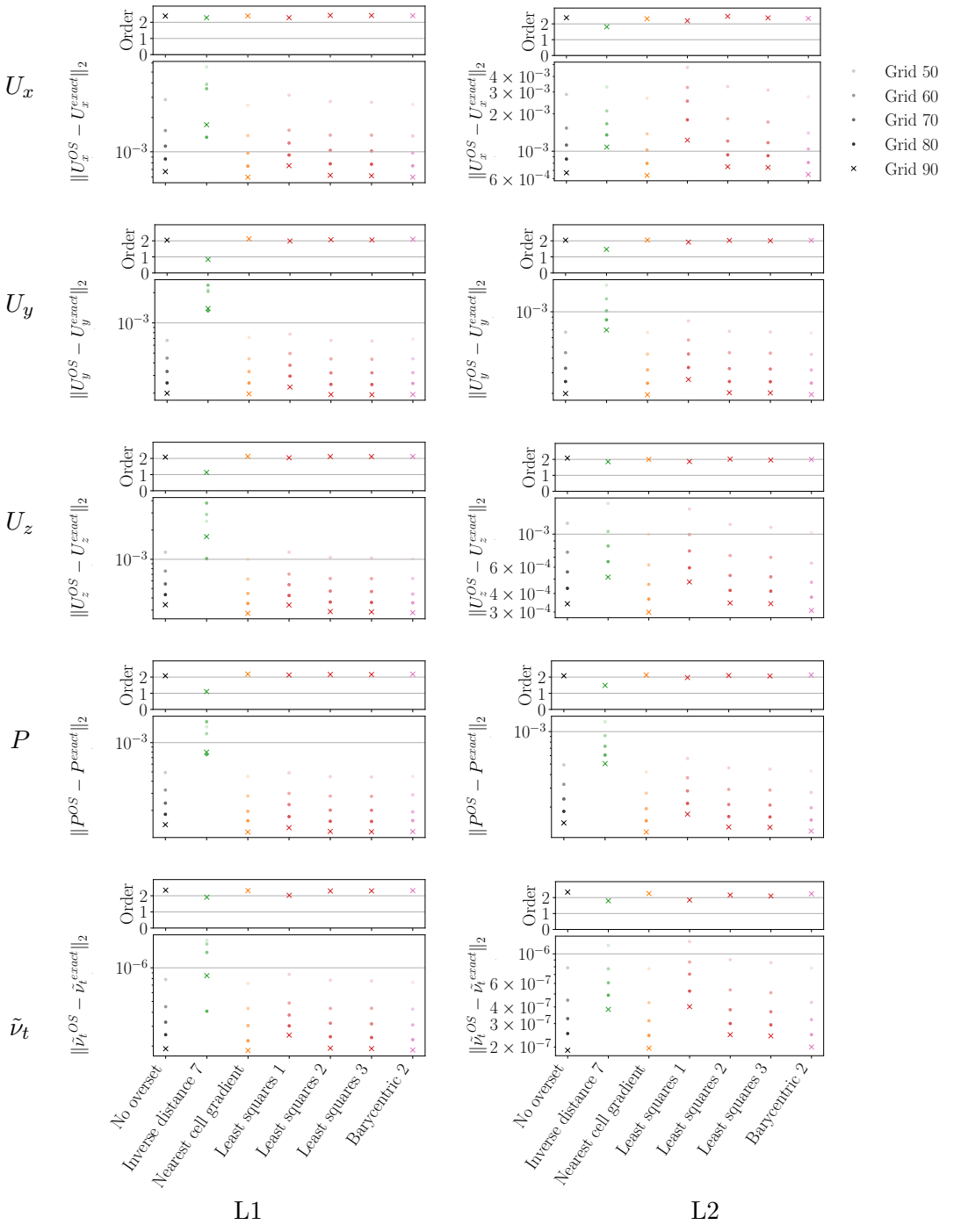
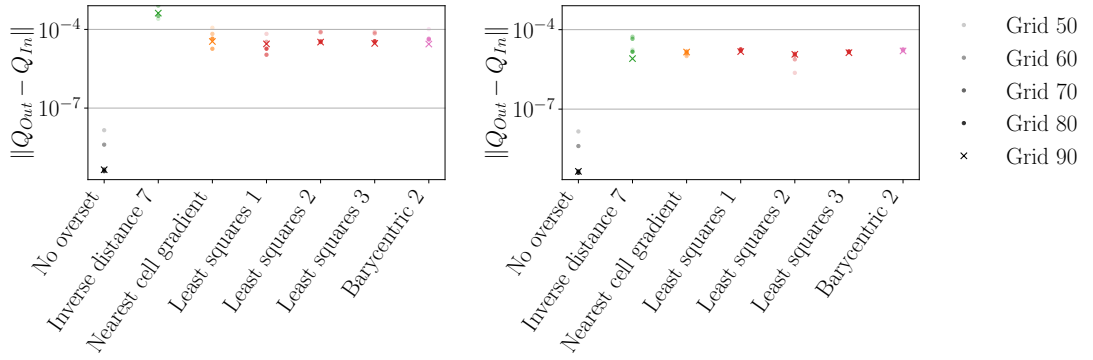


FIGURE 6.15: Comparison of error levels for each quantity depending on the interpolation scheme used for the recirculation bubble test case. Errors plotted are the time average of the L_2 norm of the error for each quantity. Convergence order is displayed above each quantity.

6.2.5 Mass imbalance study

Figure 6.16 shows the mass imbalance for the different computations. Without overset meshes, the imbalance is capped by the iterative error and is around 10^{-7} for every grid. When using overset meshes, if, like in this study, no particular treatment is done to the overset interface, mass imbalance is introduced. Independently of the scheme being used, the relation between cell sizes and mass imbalance is complex: finer meshes not always leading to lower errors. For the two 2^{nd} order schemes (*Nearest cell gradient* and *Least squares* 1), Grid 50 to 80 show a converging trend on layout L1, but Grid 90 has higher mass imbalance. This trend is also not present on layout L2. None of the other schemes show monotonic convergence trend and *Inverse distance* has, in general, the highest errors.



(A) L1 (B) L2

FIGURE 6.16: Mass imbalance for different interpolation schemes for the two tested grid layouts on the recirculation bubble test case.

6.2.6 Flow behaviour and error location

Snapshots of the entire field solution were saved at $t = 497.5$ when the recirculation bubble is largest and, as seen in the previous section, when errors are also higher. In this section only the layout L2 is being analysed in detail, though similar conclusions can be drawn from layout L1.

Figure 6.17 shows the velocity field when using the *Inverse distance* scheme, where no artefacts from the overset interpolation are visible in the field. When looking at the error field for the velocity, however, the effects of the overset meshes become apparent. In Figure 6.18, the log of the velocity error is plotted for each computation using Grid 80. The lowest errors are visible upstream of the overset region for all of the computations. When using the *Inverse distance* scheme, it is particularly visible that the error made in the overset region is being convected downstream with the flow. Some artefacts of the overset interpolation are also visible downstream of the Foreground mesh when using the *Nearest cell gradient* interpolation, but less than with *Inverse distance*. On the

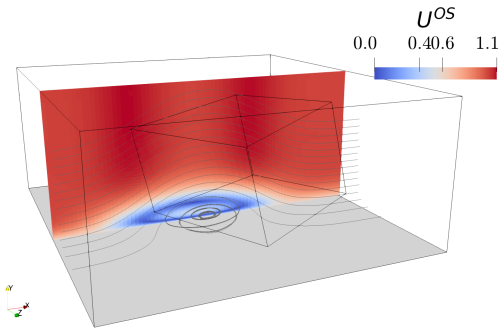
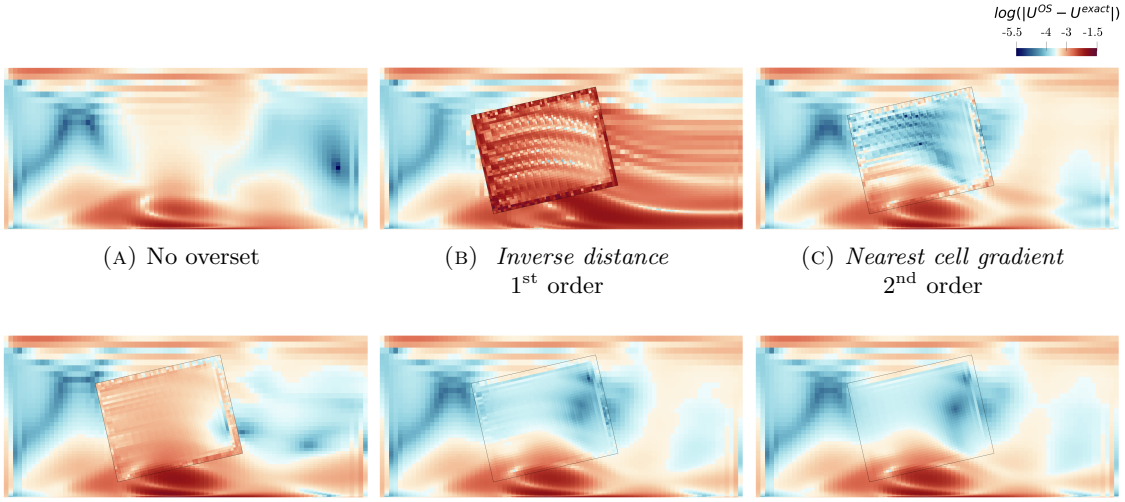


FIGURE 6.17: Velocity magnitude slice for the *Inverse distance* computation on Grid 80.



(A) No overset

(B) *Inverse distance* 1st order

(C) *Nearest cell gradient* 2nd order

(D) *Least squares* degree 1 2nd order

(E) *Least squares* degree 2 3rd order

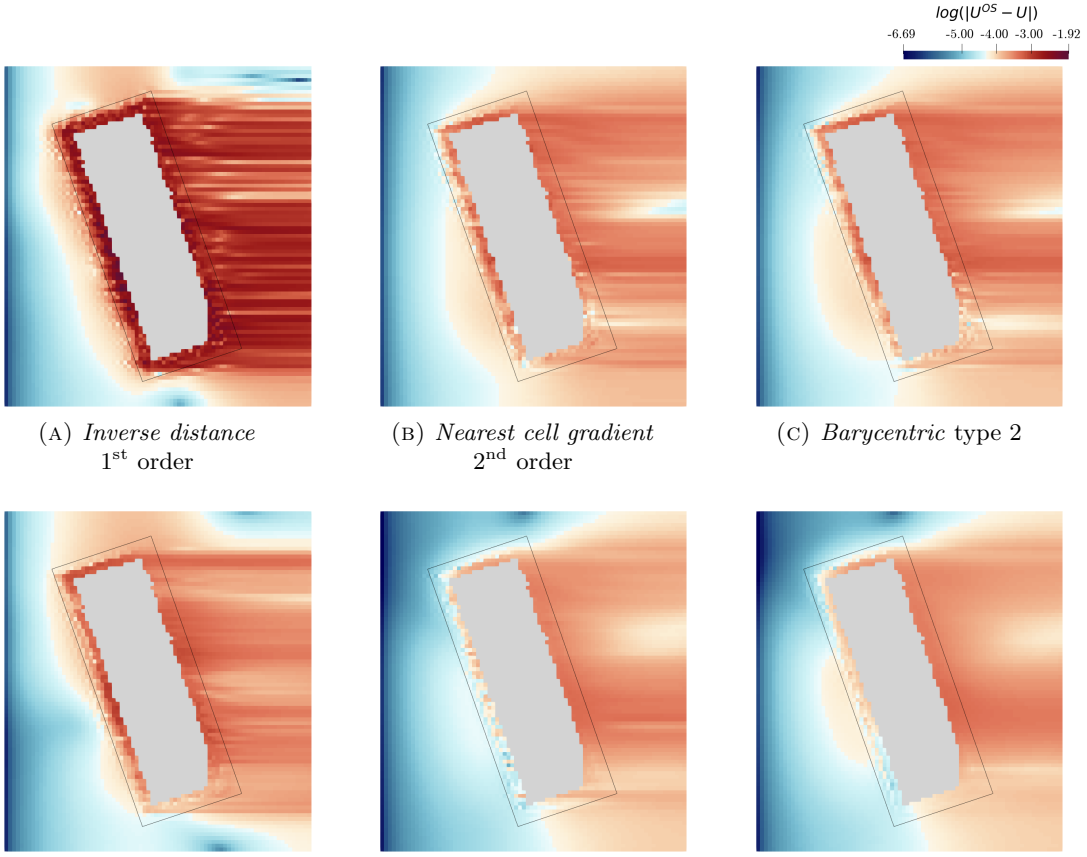
(F) *Least squares* degree 3 4th order

FIGURE 6.18: Velocity error magnitude shown with a side view of the domain (inlet on the left, wall at the bottom) for layout L2. The slice is taken in the middle of the domain ($z = 0.5$).

Foreground mesh itself, both the *Inverse distance* and *Nearest cell gradient* computations show a chequerboard error pattern following the flow streamlines. This is because the interpolation error created at the upstream *fringe* cells of the Foreground grid is being convected downstream. Neither scheme produce a smooth field, meaning that two neighbouring *fringe* cells can hold very different interpolated data creating artefacts still visible downstream of the layer of *fringe* cells. The *Least squares* computation, on the other hand, produces a much smoother interpolation, and higher order method makes the field even smoother. This leads to less visible artefacts of the interpolation even though the error levels are comparable to the *Nearest cell gradient* scheme. When comparing the *Least squares* and non-overset computation, one can notice that, in the area where the Foreground grid is, the error is lower in the overset computation, which confirms that having a finer overset mesh helps with decreasing the discretisation error despite the overset method. Moreover, the fact that the *inlet* part of the Foreground grid is better aligned with the flow streamlines may also help lower the errors for the

overset computations in this region. Finally, even when no overset meshes are used, the recirculation bubble region is where the highest errors are located.

In order to analyse only the overset method error on the Background grid, Figure 6.19 shows the difference between overset and non overset computations. Compared to previous error plots, what stands out is that, even with the degree 2 *Least squares* interpolation, interpolation errors are being convected. Even though the *Least squares* produces a smoother field compared to the *Nearest cell gradient* one, errors are still present. Interpolation artefacts are also visible on the *fringe* cells upstream of the Foreground mesh.



(A) *Inverse distance* 1^{st} order

(B) *Nearest cell gradient* 2^{nd} order

(C) *Barycentric* type 2

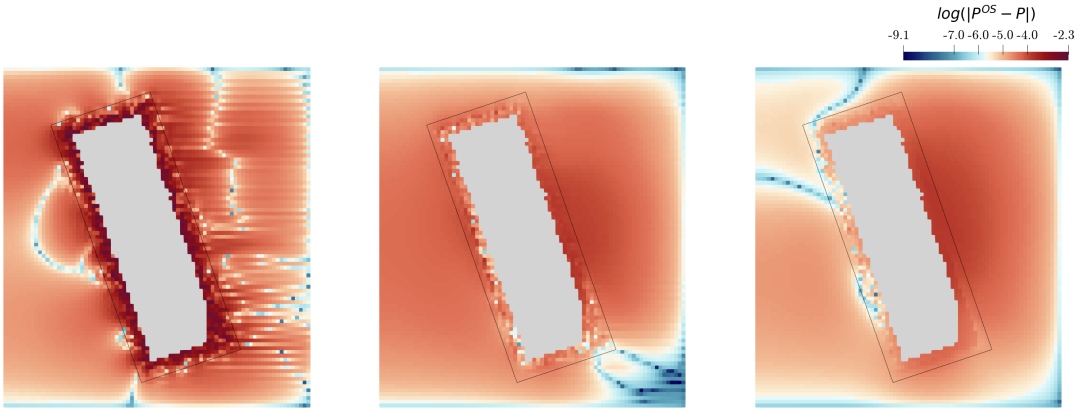
(D) *Least squares* degree 1 2^{nd} order

(E) *Least squares* degree 2 3^{rd} order

(F) *Least squares* degree 3 4^{th} order

FIGURE 6.19: Velocity difference between overset and non overset computation shown with a top view of the domain (inlet on the left) for layout L2. The slice is taken in the middle of the domain ($y = 0.2$).

Finally, Figure 6.20 shows the pressure difference between the overset and non overset computations. Similarly to the velocity comparison, 1^{st} order *Inverse distance* scheme shows higher errors also convected downstream with a clear chequerboard pattern. No large differences are, however, visible between *Nearest cell gradient* and *Least squares* schemes. The latter only displaying smoother *fringe* cells region though the error convection is not apparent.



(A) *Inverse distance* 1st order

(B) *Nearest cell gradient* 2nd order

(C) *Least squares* degree 2 3rd order

FIGURE 6.20: Pressure difference between overset and non overset computation shown with a top view of the domain (inlet on the left) for layout L2. The slice is taken in the middle of the domain ($y = 0.2$).

6.3 Concluding remarks

From this work, several preliminary conclusions can be drawn from two distinct but complementary test cases. As expected, higher order schemes produce lower interpolation errors than lower order ones. In both cases, 1st order schemes like the *Inverse distance* generate errors that significantly affect the underlying discretisation scheme of the flow solver. At the other end of the spectrum, 3rd order schemes were always sufficient to maintain the 2nd order convergence of the global discretisation. This means that the interpolation error produced by the overset interface is lower than the discretisation error. In addition to resulting in different error levels, the interpolation schemes influence the 'smoothness' of the error field. This is partly due to the number and location of the *donor* cells. The *Nearest cell gradient* or *Barycentric* schemes, for example, only use very local information (with a single *donor* cell) whereas the *Least squares* interpolation can use more than 10 *donor* cells. As a result, the *Least squares* approach was found to produce smoother fields among all the schemes tested. Its robustness, which can be controlled by increasing the number of *donor* cells used, and its accuracy make it an ideal candidate for interpolation in an overset context.

The two test cases were designed to study different features and test the overset implementation and interpolation schemes under different conditions. First, the recirculation bubble manufactured solution is a high Reynolds test case ($Re = 10^7$) compared to the Poiseuille flow one ($Re = 10$). As a result, convection plays a predominant role in the recirculation bubble case, and interpolation errors produced by the overlap are convected downstream. However, on diffusion dominant flows like the Poiseuille test case, the overset assembly perturbed the entire domain (even upstream of the overlap). This difference in the transport of the errors also impacts the smoothness of the field.

In fact, a low Reynolds number leads to a smooth field regardless of the interpolation scheme. In contrast, at high Reynolds number, the wake of the overset meshes is as smooth as the interpolation on *fringe* cells. The two test cases also differ in how 2^{nd} order interpolation schemes perform. With the recirculation bubble test case, 2^{nd} order interpolation results in interpolation errors lower than the discretisation ones. On the other hand, on the Poiseuille flow test case, a 3^{rd} order or higher interpolation scheme is required to achieve accurate results.

Chapter 7

Case study: Analysis of propeller-rudder interaction

Studies on analytical or manufactured solutions, such as those presented in the previous Chapter, are essential for gaining detailed knowledge on error propagation and the overall effect of the overset method on accuracy. However, these solutions are not as complex as the engineering applications the overset method is often used for. Therefore, conclusions drawn from these solutions are insightful, but must be checked on real-life problems. It is, for example, difficult to deduce quantitative conclusions in terms of integral quantities for real engineering problems. For this reason, this Chapter focuses on the complex and realistic problem of a rudder behind a propeller subjected to a drift angle, replicating experiments from Molland and Turnock [79]. It includes three overset meshes and mesh motion for the propeller rotation.

7.1 Introduction

Molland and Turnock [79, 81], Turnock [120] conducted a series of experiments testing a variety of rudder designs, rudder positions relative to the propeller while also applying a drift angle to the entire system. Their goal was to assess maneuverability, and the change of propeller and rudder characteristics under different conditions. Now that CFD has become more ubiquitous by getting more accessible and more accurate, such test cases can be replicated computationally. Phillips et al. [97] and Villa et al. [125] both did it by using RANS-BEM coupling, which requires no moving objects in the domain and uses a single mesh. Such coupling, allows the propeller to be only modelled hence lowering the cell count and making the computation a lot faster at the cost of higher modelling errors. In order to simulate the propeller instead of modelling it, Badoe et al. [3] used sliding meshes for its rotation. While this approach is commonly used for propeller flows, rudders, often, cannot use it due to the lack of space to account

for two cylindrical domains. Hence requiring a new mesh for each rudder angle, or the use of mesh deformation when possible. As such limitations are lifted with the overset grid method, it is often used on rudder-propeller assemblies. Replicating another set of experiments, Yilmaz et al. [132] used an overset grid on the rudder while the propeller was simulated with a sliding mesh. Di Mascio et al. [29] also used overset grids, but this time for both the rudder and every single propeller blade, resulting in a total of six overset grids.

Rudder-propeller flows are not only simulated in isolation, but are an inherent part of any full ship computation, making the conclusion drawn in this Chapter applicable for a wider range of computations. Zhirong et al. [133], for example, simulated a complex manoeuvre of a ship with two propellers using six degrees of freedom with six overset grids, one per propeller, one per rudder, one for the rest of the ship and finally a background one. Similarly, Carrica et al. [17] simulated the same vessel but this time combining a total of 37 meshes. Because the CFD solver used, CFDShip-IOWA, is limited to structured grids, even non-moving components had to be meshed independently and the overset grid method used to assemble them.

To optimise computational resource usage, in this Chapter, three interpolation schemes are tested: *Inverse distance*, *Nearest cell gradient* and *Least squares* degree 2 as they are considered representative of 1^{st}, 2^{nd} and 3^{rd} order interpolation schemes. Besides analysing the differences of these schemes on integral quantities, mass imbalance, flow features and pressure distribution, a detailed solution Verification study is performed to estimate the time and space discretisation uncertainties, and the iterative and statistical uncertainties. Finally, Validation of the rudder flow is performed against a set of experiments done in similar conditions by Molland and Turnock [79].

7.2 Problem setup

7.2.1 Experiments presentation

Molland and Turnock [78, 79, 81] conducted a series of experiments to study propeller - rudder interactions in various realistic ship manoeuvring conditions. Done in Southampton R.J. Mitchell's wind tunnel, the experiments included a centre board upstream of the propeller, the propeller itself and a rudder. Besides varying the rudder angle of attack, the entire assembly could be rotated to replicate different ship drift angles. Overall, air inlet speed, propeller rotation speed, rudder and drift angles were varied, and several centre board and rudder shapes were tested. The propeller used was a four bladed modified B4.40 Wageningen propeller, an 800 mm diameter, 0 degree rake and 0.95 mean pitch ratio propeller for which design details are found in Turnock [119] and can be seen in Figure 7.1. The rudder used in the present work is named rudder $n°2$ in the

experiments description: a NACA0020 profile straight rudder with a chord of 667 mm and a span of 1000 mm. Finally, the long centre board, the one used in this work, is 2690 mm long and 1018 mm high. Load cells on the rudder assembly and propeller allowed the recording, amongst other quantities, of lift, drag and moment coefficients for the rudder and torque and moment coefficients for the propeller. Furthermore, for some runs, pressure was recorded on the rudder and centre board surfaces.

The present work focuses on replicating a subset of the input conditions done experimentally: a drift angle of -7.5 degrees, a propeller advance ratio of $J = 0.51$, an inlet velocity of 10 m/s with rudder angles of 10 and 20 degrees. This set of conditions was picked for their flow behaviour complexity and the larger amount of experimentally recorded quantities.

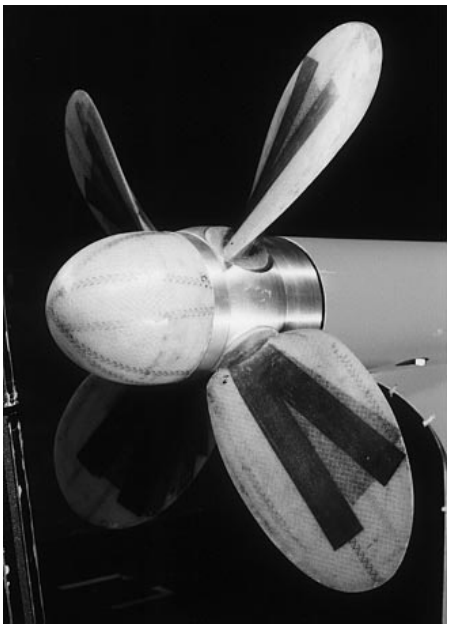


FIGURE 7.1: Photo of the propeller used during the experimental campaigns [80, 120].

7.2.2 Numerical setup

7.2.2.1 Grid and Overset setup

As stated in section 7.2.1, the numerical setup replicates the long centre board and rudder $n°2$ from Molland et al. To accommodate each component, three sets of meshes were generated, a first one containing the empty tunnel and centre board, then a cylindrical domain containing the propeller geometry and, finally, a body-fitted rudder mesh. With the overset method, the propeller can rotate and rudder can be set to any angle of attack without the need for re-meshing. Figure 7.2 shows the domain geometry and dimensions and Figure 7.3 defines the different angles used (angle of attack and drift angle) and shows the outline of the propeller and rudder domains. It should be noted

that, compared to Molland et al.'s experiments, the propeller is rotating in the opposite direction. Corresponding changes in drift and rudder angle sign conventions were made when comparing CFD with Experimental data.

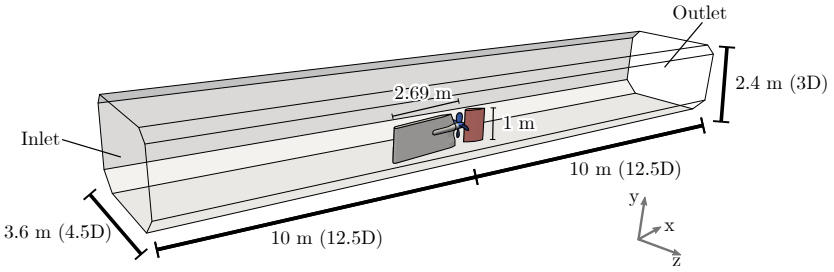


FIGURE 7.2: Computational domain dimensions replicating the cross section of the R.J Mitchel wind tunnel.

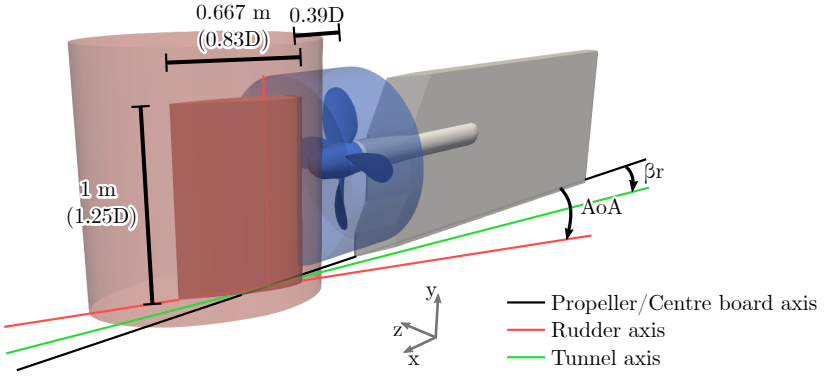


FIGURE 7.3: Definition of the different coordinate and angle systems. AoA is the rudder angle of attack while βr defines the drift angle of the assembly.

The tunnel mesh is fully structured, made with Pointwise [14]. Its cross section is at the R.J Mitchel wind tunnel's dimension, 3.6 m wide, 2.4 m high. It is also 20 m long with the rudder leading edge in the centre of the domain. The centre board's y^+ was kept below 30, and refinements were made where the propeller and rudder would be positioned to have enough cells in the overlap region.

The propeller mesh, also fully structured, was generated using GridPro [95] with some custom preprocessing tools made by MARIN [41] using Rhino [106]. To ease the meshing, especially at the blade root, the geometry of the propeller itself does not exactly replicate the shape used in the experiments. Instead of the modified B series propeller, the CFD analysis done with a conventional Wageningen B4.40 [121] with a constant pitch ratio of $P/D = 0.95$. Figure 7.4 compares both geometries. Because of this difference, Validation is performed for K_T-equivalent computations to produce a similar flow wake on the rudder. The detailed procedure is explained in section 7.5. y^+ is kept below three

on the blades with a mean y^+ below 1, and below 100 on the hub. The mesh diameter of 1 m or $1.25D$ allows for sufficient clearance at the blade tip for overset *fringe* cells.

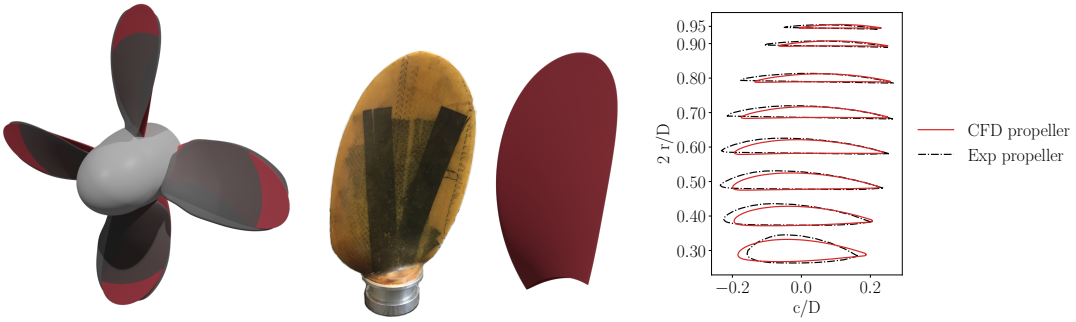


FIGURE 7.4: Comparison of the propeller blade shapes used during the experimental campaign [79] (black) and in this CFD study (red).

Finally, the rudder mesh, also fully structured, was generated with Pointwise. It is an O-grid that extends above the rudder tip. Refinements at the leading and trailing edges were made and a y^+ below 1 was maintained.

The original tunnel mesh was designed without drift angle with the centre board aligned with the tunnel inlet-outlet. To prevent re-meshing, meshes with drift angles were generated using deforming grids by rotating the centre board part of the tunnel mesh.

For each component, a set of geometrically similar meshes with different refinements was generated, Table 7.1 summarises the different cell counts. As seen in Figure 7.5 and 7.6, the overset cell status generated by Suggar++ places two layers of *fringe* cells distributed on each side of the interface. Moreover, *hole* cells are placed in the propeller mesh (due to the rudder overlap) and in the tunnel mesh where the propeller and rudder meshes are positioned.

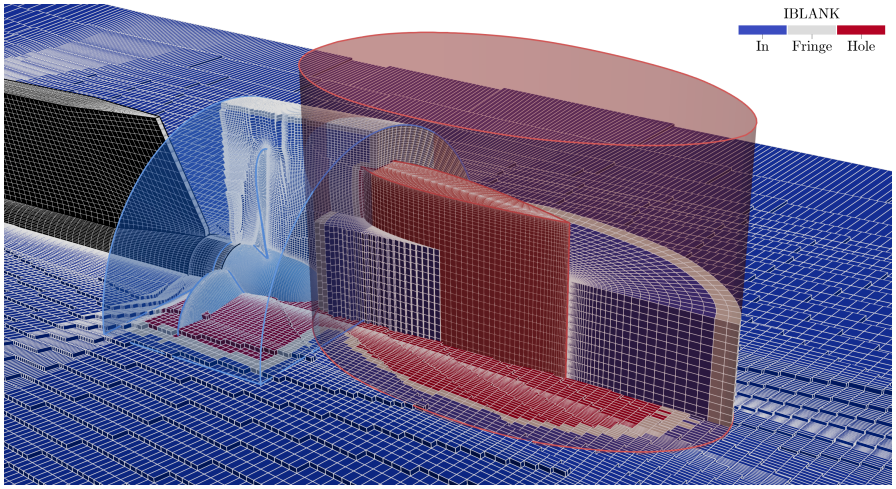


FIGURE 7.5: 3D view of the different meshes coloured with IBLANK information for the coarsest assembly G1. The wireframe of each mesh is also coloured differently with the tunnel in white, the propeller in blue and the rudder in red.

TABLE 7.1: Refinements ratio (h_i) and cell counts for the different meshes generated.

	$\mathbf{h_i}$	**Total**	**Tunnel**	**Propeller**	**Rudder**
G1	1.51	$10.8 \cdot 10^6$	$6.6 \cdot 10^6$	$3.2 \cdot 10^6$	$1.0 \cdot 10^6$
G2	1.36	$14.8 \cdot 10^6$	$9.8 \cdot 10^6$	$3.4 \cdot 10^6$	$1.6 \cdot 10^6$
G3	1.21	$21.4 \cdot 10^6$	$13.1 \cdot 10^6$	$6.2 \cdot 10^6$	$2.1 \cdot 10^6$
G4	1.0	$37.4 \cdot 10^6$	$23.1 \cdot 10^6$	$10.8 \cdot 10^6$	$3.6 \cdot 10^6$

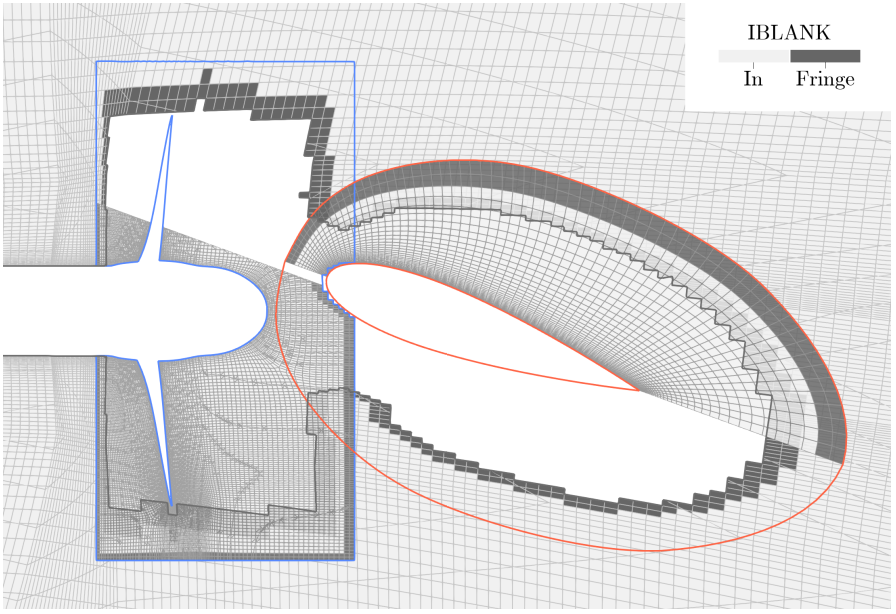


FIGURE 7.6: Top view of the different meshes showing IBLANK information for the coarsest grid assembly G1. Only the lower two thirds of the propeller (blue) and top half of the rudder (red) meshes are displayed to reveal the tunnel mesh in the background.

7.2.2.2 Computational setup

In this study, turbulence modelling is done with the $k - \sqrt{k}L$ two equations turbulence model [75], chosen for its better robustness compared to $k - \omega$ SST models. Momentum and pressure correction equations are coupled using the SIMPLE methods and all equations are solved in a segregated way. Convective fluxes for the momentum and turbulence equations are discretised by the 2^{nd} order QUICK scheme (with limiters), and a three-time level scheme is used for 2^{nd} order in time discretisation.

The side walls of the tunnel are modeled using a slip boundary condition to account for their blockage effect without the need to refine the boundary layer, as would be required for a non-slip wall, in order to reduce computational cost. The rudder, propeller and centre board surfaces are modeled using a non-slip wall boundary condition. Automatic wall functions are used, where for $y^+ < 5$ (viscous-layer) a fully resolved boundary layer treatment is employed, for $y^+ > 30$ (log-layer) wall functions are used, and in-between (in the buffer layer) a blending between the two approaches is used. At the outlet a pressure boundary condition is used and the inflow velocity ($U_\infty = 10\ m/s$) is set at the

inlet of the tunnel (aligned with the tunnel itself) together with an eddy viscosity ratio of 0.01.

Three interpolation schemes are used for the overset grid method interpolation. 1st order *Inverse distance*, 2nd order *Nearest cell gradient*, and a 3rd order polynomial-based *Least squares* schemes. The latter uses 25 *donor* cells per *fringe* cell to overdetermine the system of linear equations and help robustness ($C_{mult} = 2.5$).

7.2.2.3 Analysed quantities

In this study, integral quantities related to both the propeller and rudder are assessed, which are defined in the following equations,

$$J = \frac{U_\infty}{nD}, \quad K_T = \frac{T}{\rho n^2 D^4}, \quad K_Q = \frac{Q}{\rho n^2 D^5}, \quad C_L = \frac{L}{\frac{1}{2}\rho U_\infty^2 cs},$$
$$C_D = \frac{D}{\frac{1}{2}\rho U_\infty^2 cs}, \quad C_m = \frac{M_y}{\frac{1}{2}\rho U_\infty^2 c^2 s}, \quad C_{pc} = 100 \times \left(\frac{C_m}{\sqrt{C_L^2 + C_D^2}} + \frac{0.2}{c} \right). \quad (7.1)$$

J is the propeller's advance ratio, K_T its thrust coefficient, computed from the thrust T, and K_Q its torque coefficient computed from the propeller torque Q. Concerning the rudder, C_L is the lift coefficient, C_D the drag coefficient, C_m its moment coefficient, here calculated at 30% chord and C_{pc} the location of the centre of pressure along the chord. Also, U_∞ is the inlet velocity ($U_\infty = 10\ m/s$), n the propeller rotation speed in revolution per second, D the propeller diameter ($D = 0.8\ m$), ρ the air density ($\rho = 1.225\ kg/m^3$), c the chord length ($c = 0.667\ m$) and s the rudder span ($s = 1\ m$).

For each quantity, time averaging is done over an integer number of blade passing periods and statistical uncertainty is quantified in section 7.3.3 using Brouwer et al. [12, 13] methodology. Time averaged velocity and pressure fields are also recorded.

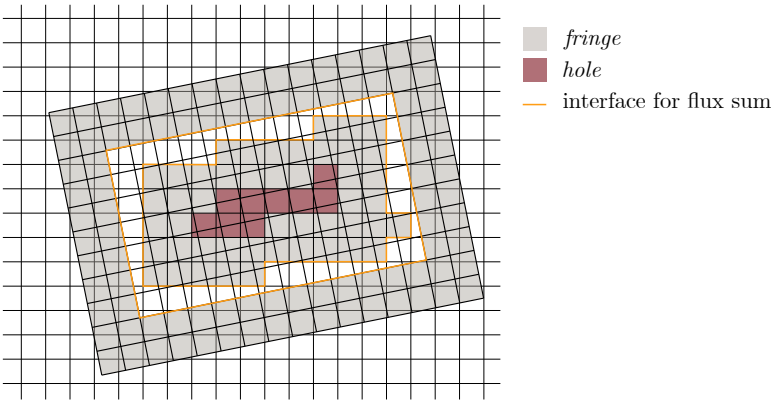


FIGURE 7.7: Overset meshes schematic highlighting (in orange) the faces where signed mass fluxes are being summed up to compute mass imbalance.

Finally, given that the overset grid method implementation used in this study does not guarantee total mass flux conservation, monitoring the mass imbalance is of particular importance. To this end, signed mass fluxes between *fringe* cells and *in* cells are summed and recorded over time for each mesh individually, thereby effectively measuring the mass imbalance on each mesh. The cells where the summation is performed are shown in Figure 7.7. In total four quantities are analysed: total mass fluxes going in and out of the domain via the inlet and outlet (Q_{total}), and fluxes going through each of the overset 'interfaces' on the tunnel (Q_{tunnel}), propeller (Q_{prop}) and rudder (Q_{rudder}) meshes. One should note that, besides overset grid interpolation, mass fluxes are governed by the pressure correction equation, hence a non-zero value is a marker of overset grid interpolation errors and/or iterative errors for that particular equation.

7.3 Verification studies

7.3.1 Iterative uncertainty

Iterative uncertainty quantification has been done following the method presented in Eça and Hoekstra [34], Eça et al. [39]. For this, a set of five computations on grid G2 with the rudder at 10 degrees angle of attack was done with a varying number of outerloops per time step of 25, 50, 75, 100 and 150. Each computation resulted in different residuals iterative convergence, and Figure 7.8 shows, for each equation solved, the time averaged residual level at the end of each time-step depending on the number of outerloop per time-step. First, apart from the pressure correction equation (denoted P), all the equations are not affected by the number of outerloops per time-step. This is mainly because their convergence stall within the first 25 outerloops each time-step.

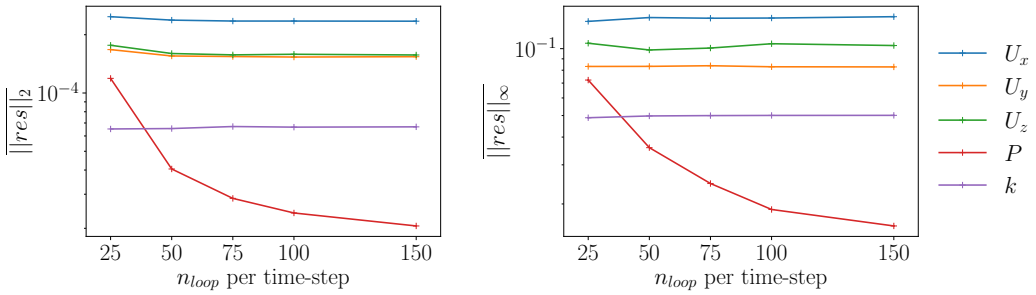


FIGURE 7.8: Time averaged L_2 and L_∞ residuals for an increasing number of outerloops (n_{loop}) per time-step. The second turbulence equation residuals are omitted for clarity, but they are relatively constant and three orders of magnitudes lower than the turbulence kinetic energy (k) residuals.

Following Eça et al. [39], a least squares fit is performed on the quantities of interest against their relative residuals levels. In this study, the L_∞ norm of residuals levels for the pressure correction equation were selected as they are the only one affected by the number of outerloops. The proposed fit function is:

$$\phi(\epsilon_i) = \phi_0 + \alpha e^{F(\epsilon_i)\beta}, \tag{7.2}$$

with ϕ the quantity of interest and ϵ_i the residuals level achieved. Both functions $F(\epsilon) = ln(\epsilon)$ and $F(\epsilon) = -1/\epsilon$ were tested and the first one is selected for its better fit to the data. Results are shown in Figure 7.9 for the propeller's K_T and rudder's C_L. A converging trend is clearly achieved for the residuals as a function of the number of outerloops, and iterative uncertainty can be computed. In the rest of this work, 75 outerloops per time-step will be used as a balance between iterative error (here its uncertainty is estimated at 0.2% for K_T and 0.02% for C_L) and computational cost. It has to be noted, however, that because only the pressure correction equation residuals are reduced with more outerloops, this study only partially evaluates iterative error. In fact, reduction of all the other residuals would be needed to fully assess the iterative uncertainty. Moreover, iterative error is here studied on a single mesh, but Eça et al. [39] showed that iterative error can influence the study of discretisation errors. Nonetheless, given the relatively low uncertainties found for integral quantities further study was considered outside of the scope of this work.

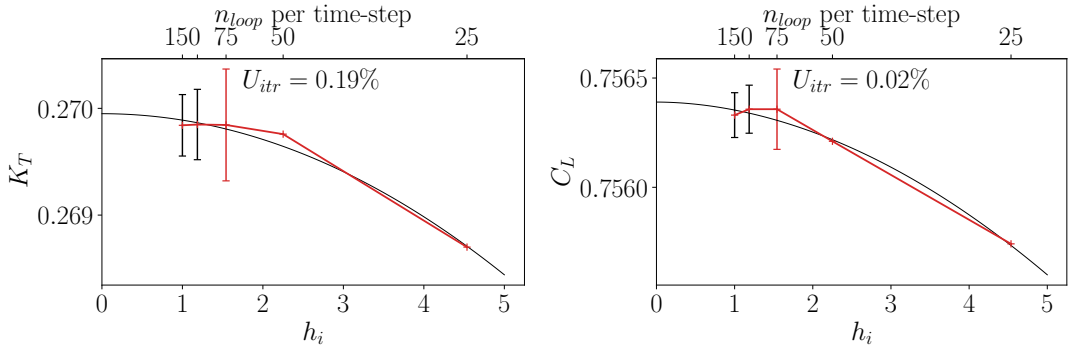


FIGURE 7.9: K_T and C_L convergence with the pressure correction residuals, $h_i = \frac{\overline{||Pres\ i||_\infty}}{||Pres\ 0||_\infty}$. Bars show the iterative uncertainty and in red the one corresponding to the number of outerloops selected for the rest of this work.

7.3.2 Time discretisation uncertainty

To assess time discretisation errors, a set of five computations with varying time-steps was used (see Table 7.2). The number of time-steps ranged from 400 per propeller revolution to 50, leading to an average Courant number from 0.7 to 5. For computational resources reasons, this test has been performed on mesh G2 only even though some variations should be expected with finer meshes and the increase of Courant number. Figure 7.10 plots K_T and C_L against time-step refinements, each quantity showing convergence trends. Using Eça and Hoekstra [34] method, the time-step uncertainty on K_T is estimated at 0.15% when using 1.8° rotation per time-step, and 2% for C_L. This time-step is selected to be used in the rest of the current study. One should note, however, that for

the finest mesh assembly (G4) this choice leads to an average Courant number of around 2.2 which would result in similar K_T uncertainty but, likely, a higher C_L uncertainty, looking at the trend of each convergence plot.

TABLE 7.2: Time-steps tested to compute time discretisation uncertainty. The rightmost two columns show the time-step compared to the propeller rotation speed ($J = 0.51$ leading to 1460 RPM).

$\mathbf{h_i}$	$\mathbf{t_{step}}$	$\overline{\mathbf{CFL}}$	$\mathbf{t_{step}/rot}$	$^\circ/\mathbf{t_{step}}$
1.00	$1.027 \cdot 10^{-4}$ s	0.66	400	0.9°
1.33	$1.370 \cdot 10^{-4}$ s	0.89	300	1.2°
2.00	$2.055 \cdot 10^{-4}$ s	1.33	200	1.8°
2.67	$2.740 \cdot 10^{-4}$ s	1.77	150	2.4°
4.00	$4.110 \cdot 10^{-4}$ s	2.66	100	3.6°
8.00	$8.219 \cdot 10^{-4}$ s	5.32	50	7.2°

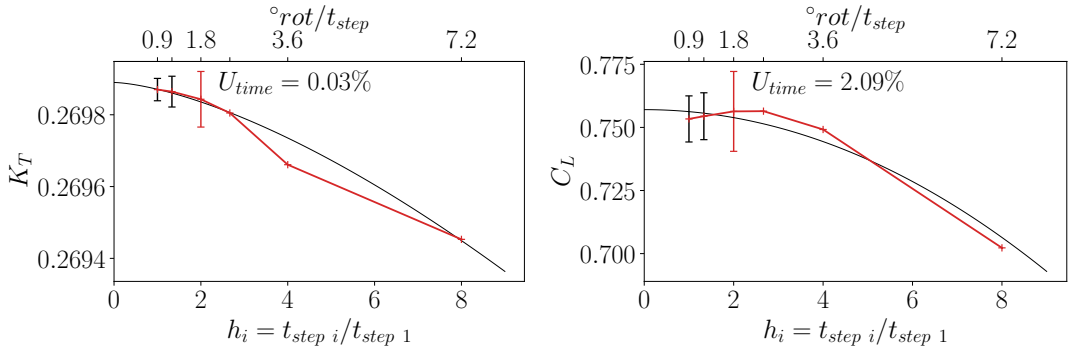
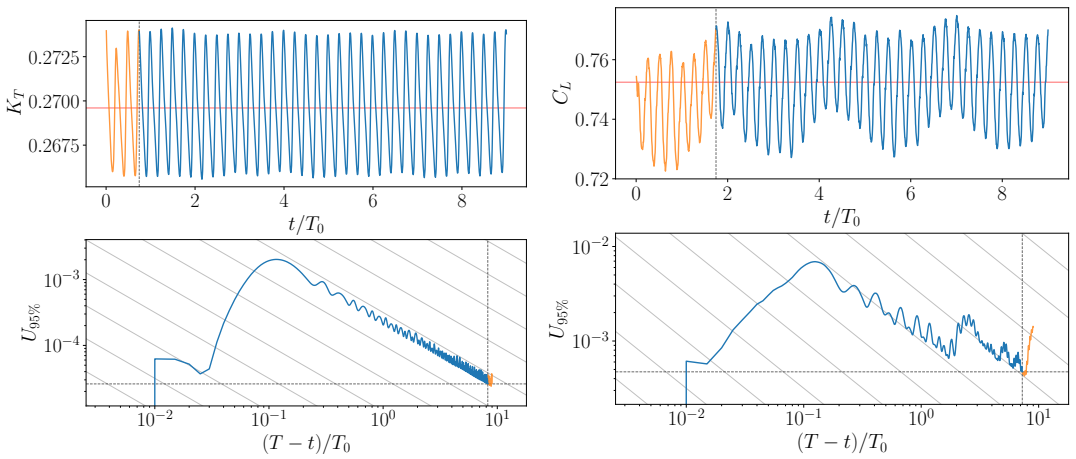


FIGURE 7.10: K_T and C_L convergence with time-step refinement. The error bars show time discretisation uncertainties and are computed using Eça and Hoekstra [34] methodology.

7.3.3 Statistical uncertainty

To speed up simulations, each computation presented in this work is initialised from a fully developed computation on a coarser mesh. After this initialisation, each computation is run for at least nine propeller rotations. The transient scanning technique is then used on each quantity to detect and discard the transient portion of the results and compute the statistical uncertainty. As shown in Figure 7.11, computations done with 10 degrees rudder angle converge quickly to a steady state and show a clear oscillatory behaviour in sync with the blade passing frequency (four times per rotation). Rudder forces (like C_L) additionally display secondary oscillatory behaviour about 10 times larger than the blade passing one (or 2.5 propeller period). Figure 7.11 shows that the statistical uncertainty is below 0.5% for this set of computations and analysed quantities. While not displayed here, this conclusion holds for the other mesh refinements, overset grid interpolation schemes and quantities.

When setting the rudder at 20 degrees, this behaviour changes for grid assemblies G1 to G3 (the coarsest ones) as they introduce an extra long transient phase lasting more than



(A) $\overline{K_T} = 0.26960 \pm 2.6 \cdot 10^{-5}$ (B) $\overline{C_L} = 0.7525 \pm 4.7 \cdot 10^{-4}$

FIGURE 7.11: Propeller (K_T) and rudder (C_L) forces coefficients time histories with the rudder at 10 degrees angle of attack. The statistical uncertainty on the mean is computed using the transient scanning technique [12] and shown on the bottom plot for each quantity. The transient portion (orange) is removed from the computation of the mean. T_0 is the rotation period of the propeller and T is the simulation time.

40 propeller periods seen in Figure 7.12. As this transient portion needs to be simulated but discarded this set of computations was deemed too expensive to be computed. As a result, at 20 degrees rudder angle, only the results for mesh G4 (the finest one) will be shown and discussed. These were run for 12 propeller rotations and lead to similar statistical uncertainties as the 10 degrees rudder angle ones.

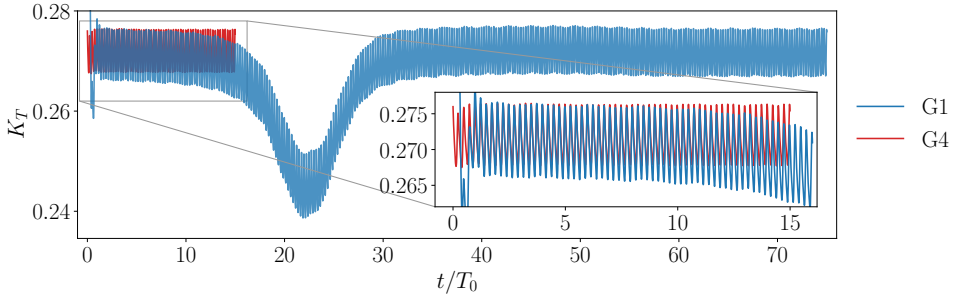


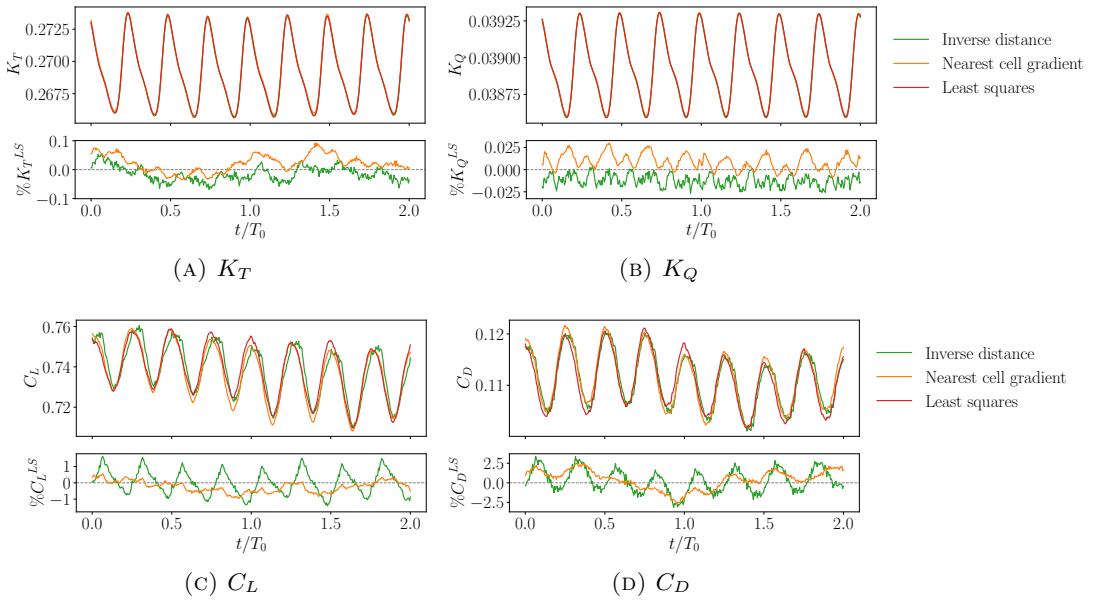
FIGURE 7.12: Propeller K_T when the rudder is set at 20 degrees angle on the coarsest (G1) and finest (G4) meshes.

7.4 Impact of interpolation schemes

7.4.1 Integral quantities

As seen in section 7.3.3, propeller forces coefficients K_T or K_Q have a clear oscillatory behaviour in sync with the blades rotation. These oscillations are also consistent over time, likely because the propeller inlet, even in the wake of the angled centre board, is

steady. The results obtained when using the three interpolation schemes tested share this characteristic, and, overall, have very close behaviour as seen in Figure 7.13. Rudder forces, however, exhibit larger differences and seem less smooth. Being in the wake of the propeller and having the flow cross at least one more overset interface brings more time variation and greater changes between the different interpolation schemes. The *Inverse distance* shows up to 1% difference on C_L and 2.5% on C_D compared the *Least squares* computation, but only 0.05% on K_T. The same trend is also true for the *Nearest cell gradient*, albeit with smaller differences. Apart from different amplitudes and general shape of the oscillation, it appears that using the *Inverse distance* scheme results in 'noisier' forces, once again, specially on the rudder lift and drag.



(A) K_T (B) K_Q

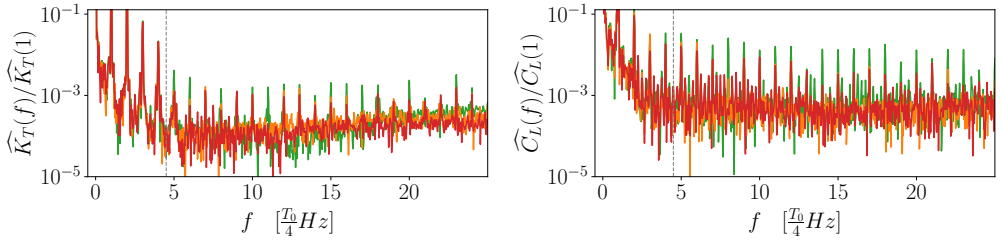
(C) C_L (D) C_D

FIGURE 7.13: Force coefficients time history for the three interpolation schemes tested at 10 degrees rudder angle. The bottom plot is a comparison with the *Least squares* scheme results, computed with: $\%\phi^{LS} = 100 \cdot \frac{\phi - \phi^{LS}}{\phi^{LS}}$

To quantify these differences, and in particular the spurious oscillations between the interpolation schemes, a spectral analysis is shown in Figure 7.14. First, doing a Fourier transform on the force coefficients time histories confirms that, overall, propeller force coefficients have fewer higher-frequency harmonics than rudder coefficients. Moreover, the high frequencies are mainly harmonics of the blade passing frequency. In addition to having more harmonics, the rudder quantities have higher white noise levels too. Differences between the interpolation schemes can also be observed. The *Inverse distance* computation appears to have more harmonics than the other two tested, as well as overall higher intensities outside of harmonics.

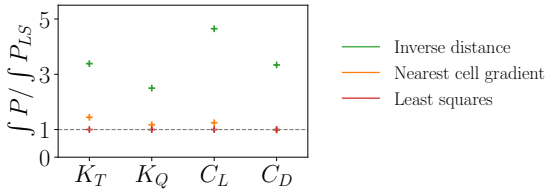
Computing the power spectral density of high frequencies (higher than 4.5 times the blade passing frequency) highlights the energy stored in these high frequencies and is here used as a measure of the overall 'smoothness' of the signals. Such quantity,

normalised by the *Least squares* density, is plotted in Figure 7.14c. It shows the *Nearest cell gradient* computation having very similar levels to the *Least squares* one. The *Inverse distance* computation, on the other hand, regardless of the quantity has two to five times more energy in these high frequencies. These results are here displayed for 10 degrees angle of attack on grid G4 but are similar to the 20 degrees rudder angle or other grid refinements.



(A) Fourier transform of K_T

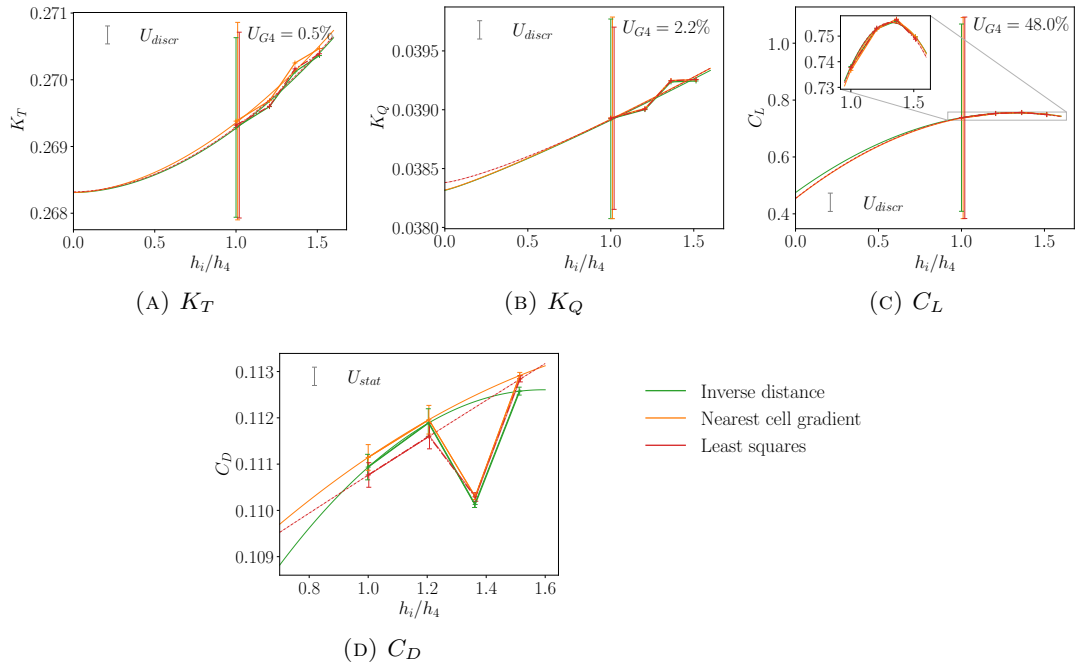
(B) Fourier transform of C_L



(C) Power spectral density of high frequencies (higher than 4.5 BPF) normalised by the *Least squares* computation ones.

FIGURE 7.14: Fourier transforms of K_T and C_L. Frequencies are normalised by the blade passing frequency ($4/T_0$) and spectra by the level of the first harmonic (corresponding to the blade passing frequency). Plot 7.14c compares the integration of the power spectra for high frequencies (higher than 4.5 times the blade passing frequency, denoted with the vertical dashed line on the Fourier transform plots).

Time averaging the force coefficient eliminates these high frequencies and leads to very similar results for the three interpolation schemes. This is observed in Figure 7.15, where each quantity is plotted against grid refinement. K_T, for example, tends to be only marginally over-predicted on the *Nearest cell gradient* results compared to the other two schemes but with a difference of less than 0.03% it can be considered as an insignificant change in an engineering context. For propeller forces, the convergence with grid refinement is clear and monotonic which leads to low uncertainty of 0.5% and 2.2% for K_T and K_Q respectively. In comparison, the rudder forces show non-monotonic behaviours with less clear convergence trends. On C_L the 'bell shaped' convergence leads to 48% uncertainty. C_D, on the other hand, has a different behaviour. In fact, in Figure 7.15d, the statistical uncertainty is this time plotted as it is of similar magnitude to the mesh refinements changes, which then confounds the fitting and computation of the discretisation uncertainty. Even though the three schemes show very close trends, the uncertainty varies from 0.5% to 11% and this results in very different fitting functions. This issue could be solved by either running the simulation for longer to decrease the



(A) K_T (B) K_Q (C) C_L

(D) C_D

FIGURE 7.15: Time average force coefficients against grid refinement for the three interpolation schemes tested at 10 degrees rudder angle. For each of them, except C_D, discretisation uncertainty (U_{discr}) is displayed. For C_D, statistical uncertainty (U_{stat}) is shown instead.

statistical uncertainty, or by using more meshes with different refinements to make the fit less influenced by small variations[1].

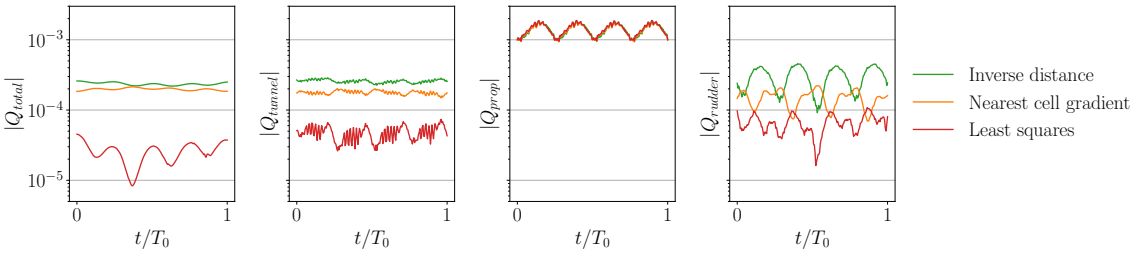
Overall, this shows that the spurious oscillations shown in the previous section mainly lead to zero mean noise, and do not lead to sustained changes to forces as the three interpolation schemes result in very similar time averaged values. Also, propeller forces show monotonic 2^{nd} order converging trends whereas rudder ones are non-monotonic. The discretisation uncertainty estimation, however, is one order of magnitude larger than the difference between the three schemes suggesting that the discretisation error is higher than the interpolation error made on overset interfaces.

As explained in section 7.2.2.3, mass fluxes are not explicitly conserved by the overset grid method, and consequently, a mass imbalance is introduced and influenced by the interpolation scheme. Figure 7.16 shows the sum of mass fluxes going through each mesh over time and for each scheme. First, once again, the blade passing frequency is discernible on each of the quantities, even though it is not the only frequency component. The comparison of the different schemes shows that, for the *total* mass imbalance (Figure 7.16a), the *Least squares* scheme shows results an order of magnitude lower than the other two, but they all share similar oscillation amplitudes (albeit not obvious due to the log scale). The *tunnel* interface mass imbalance has, then, levels similar or slightly higher than the *total* ones but with extra frequencies components. Next, the *prop*

[1] Eça et al. methodology is known to be pessimistic and highly sensitive to small spread of refinements and small number of refinement levels.

interface shows a drastically different behavior with higher flux imbalance and only minor differences between the interpolation schemes. Finally, the *rudder* mesh has a behavior close to the *tunnel* one with, however, larger oscillations amplitudes.

The difference in smoothness and higher frequency components between the *total* mass imbalance and those on the *tunnel* or *rudder* is explained by the direct proximity of the latter two to *fringe* cells. The inlet and outlet of the domain are further away from overset meshes, which means that the oscillations are dampened due to numerical diffusion and iterative errors. The higher imbalance seen on the propeller mesh is either related to the iterative error, the highest pressure correction residuals being close to the blade tips, or due to the fact that the propeller mesh is the only one moving. Regardless of the reason, the error source driving this phenomenon appears to be different from the one seen on the other three quantities as it is not influenced by the interpolation scheme, it has a different level and it is not influenced in the same way by mesh refinements (not shown here). The interpolation made on the propeller mesh still deteriorates the mass imbalance, but its effect is overshadowed by another error source. Overall, it is, however, not clear how and if errors made on fluxes propagate through an overset interface. Here, for example, the high flux imbalance of the propeller mesh does not seem to affect the other two ones. One beginning of explanation resides with the fact that fluxes are computed on cell faces while the interpolation uses exclusively cell centre information partially dissociating the different meshes.



(A) Q_{total} (B) Q_{tunnel} (C) Q_{prop} (D) Q_{rudder}

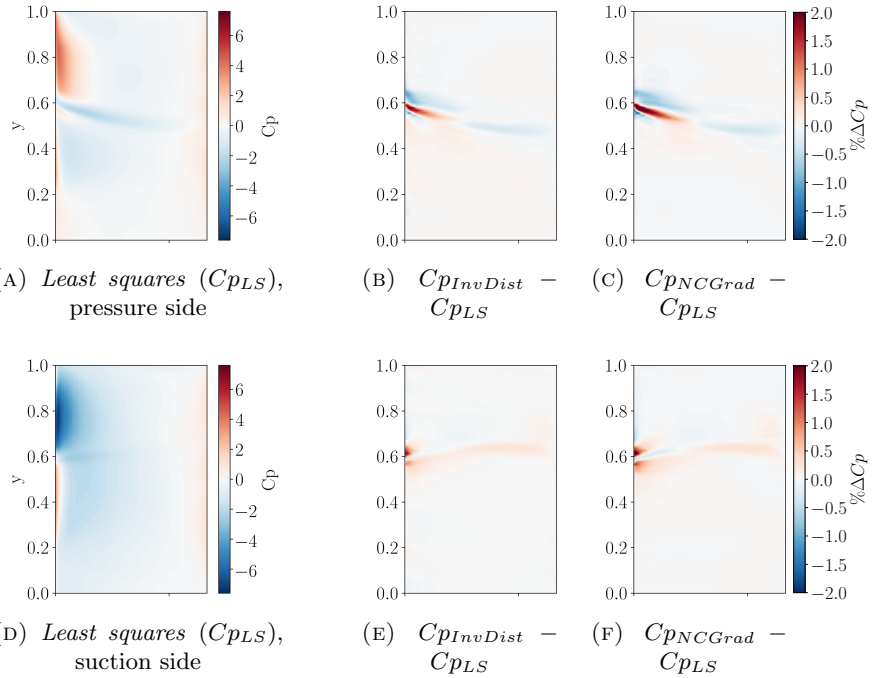
FIGURE 7.16: Sum of fluxes going through each overset interface (Q_{tunnel}, Q_{prop} and Q_{rudder}) and by the inlet and outlet of the domain (Q_{total}) for the three interpolation schemes tested on the finest mesh (G4) and at 10 degrees rudder angle. Fluxes are normalised by the inlet mass flux.

7.4.2 Pressure on the Rudder

The time averaged pressure coefficient on the rudder surface is shown in Figure 7.17 at 10 degree rudder angle and in Figure 7.18 at 20 degrees. In both figures, the *Least squares* results are displayed in the first column, and the subsequent two plot differences with *Inverse distance* and *Nearest cell gradient* respectively. Since the height or span of the rudder is $s = 1\ m$, no distinction will be made between the height of a section and its span-wise ratio (y/s) here.

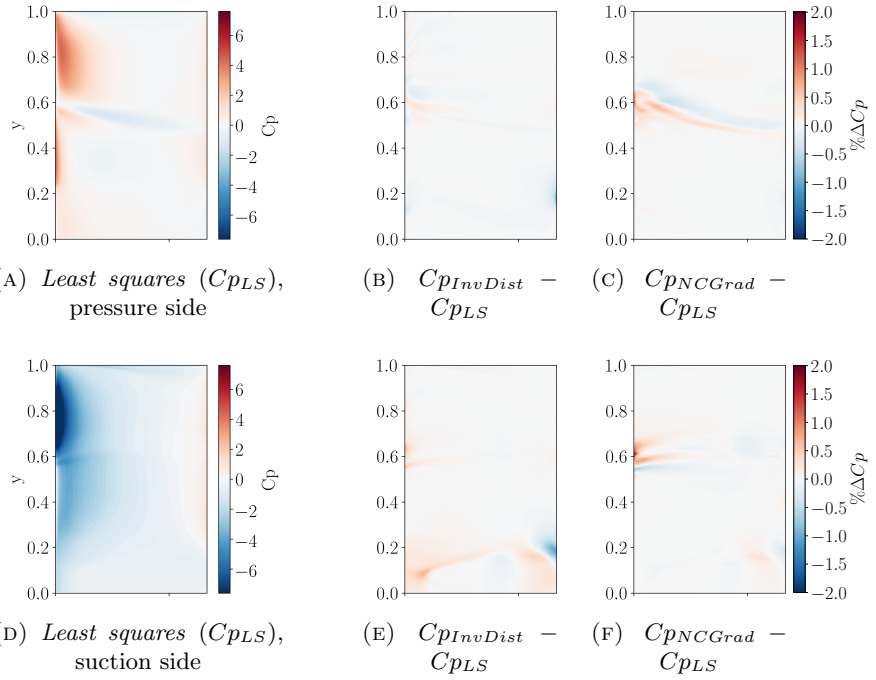
At both 10 and 20 degrees, from 0.2 m to 1 m height, the leading edge of the rudder sees high and low pressure regions directly related to the propeller rotation. At 0.6 m height the impact of the hub vortex is clearly visible and creates a lower pressure trail. On the suction side, this trail goes down due to the high pressure region above it and goes up on the pressure side, again due to the higher pressure region this time in the lower part of the rudder. Being in the wake of the propeller, the high pressure regions have levels higher than the stagnation pressure. Moreover, as expected, a higher rudder angle leads to higher pressure differences between the two sides of the rudder, generating more lift.

When comparing the different interpolation schemes the main variations are in the hub vortex region with at most 3% differences at 10 degrees angle of attack. In comparison, at 20 degrees, differences are lower with maximum around 1% or lower. Surprisingly, however, in both cases, the *Inverse distance* results look closer to *Least squares* than *Nearest cell gradient* are. These differences, being very localised and of relatively small magnitude, have likely a marginal contributions to the overall pressure lift and drag. For a finer comparison, line plots along the rudder's chord are shown in Figure 7.19. They highlight the fact that, outside the hub vortex region, *Least squares* results are actually closer to *Nearest cell gradient* ones than they are to *Inverse distance*, though, again, the overall small differences explain the minimal variation in lift and drag observed in the previous section.



(A) *Least squares* (Cp_{LS}), pressure side
(B) $Cp_{InvDist} - Cp_{LS}$
(C) $Cp_{NCGrad} - Cp_{LS}$
(D) *Least squares* (Cp_{LS}), suction side
(E) $Cp_{InvDist} - Cp_{LS}$
(F) $Cp_{NCGrad} - Cp_{LS}$

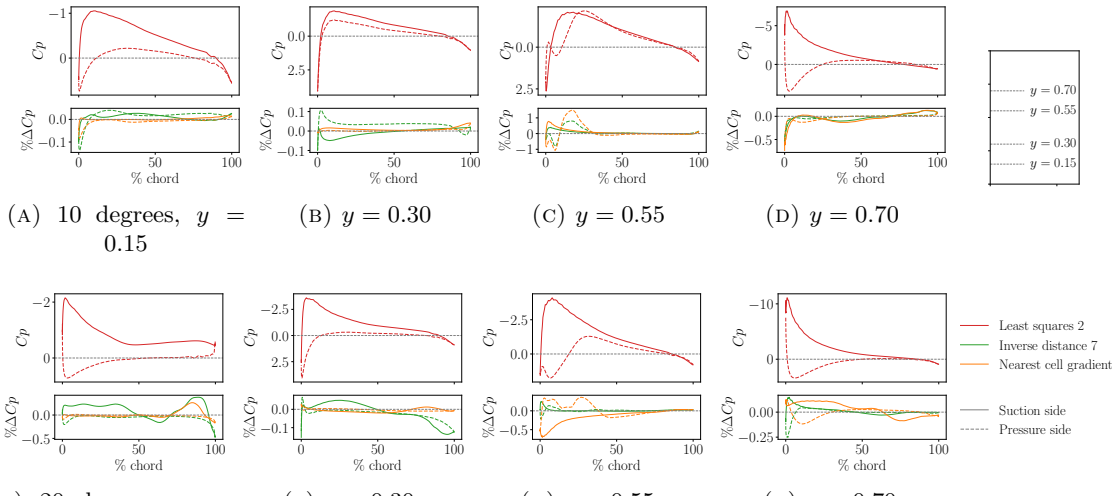
FIGURE 7.17: Pressure coefficient on the rudder's surfaces at 10 degrees angle of attack for grids G4. The leading edge is shown on the left of the frame. The first column shows *Least squares* results while the other two display its difference with *Inverse distance* and *Nearest cell gradient* respectively, normalised by the amplitude of Cp over the rudder surface.



(A) *Least squares* (Cp_{LS}), pressure side (B) $Cp_{InvDist} - Cp_{LS}$ (C) $Cp_{NCGrad} - Cp_{LS}$

(D) *Least squares* (Cp_{LS}), suction side (E) $Cp_{InvDist} - Cp_{LS}$ (F) $Cp_{NCGrad} - Cp_{LS}$

FIGURE 7.18: Pressure coefficient on the rudder's surfaces at 20 degrees angle of attack for grids G4. The leading edge is shown on the left of the frame. The first column shows *Least squares* results while the other two display its percentage of difference with *Inverse distance* and *Nearest cell gradient* respectively, normalised by the amplitude of Cp over the rudder surface.

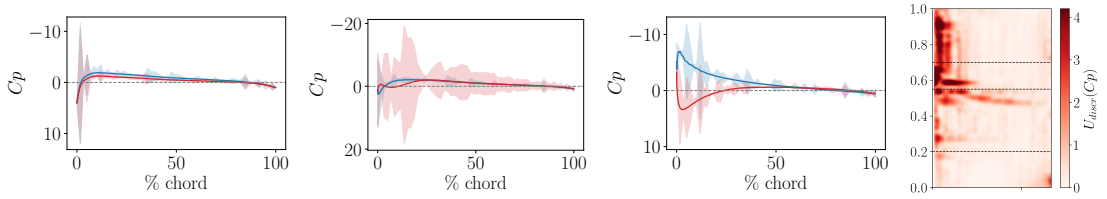


(A) 10 degrees, $y = 0.15$ (B) $y = 0.30$ (C) $y = 0.55$ (D) $y = 0.70$

(E) 20 degrees, $y = 0.15$ (F) $y = 0.30$ (G) $y = 0.55$ (H) $y = 0.70$

FIGURE 7.19: Pressure coefficient on the rudder's surface at various height. The first line at 10 degrees angle of attack, and the second one at 20. For each section, the bottom plot is a comparison with *Least squares* results.

Finally, similarly to the integral quantities, using the four sets of meshes space discretisation uncertainties can be computed for the rudder's pressure distribution. Figure 7.20 displays it at several slices and on the rudder's pressure side. On average, the uncertainty is 0.6 Cp overall, but it is a lot higher in the hub vortex region. This means that, regardless of the interpolation scheme, a reduction of cell sizes in this region would be needed to reduce discretisation errors, and doing so would likely contribute to lowering the lift and drag errors as well. Finally, the differences between the schemes may also be reduced in this region with further grid refinements.



(A) $y = 0.20$ (B) $y = 0.55$ (C) $y = 0.70$ (D) Discretisation uncertainty on the pressure side

FIGURE 7.20: Space discretisation uncertainties for the local pressure coefficient. Data for the *Least squares* computation at 10 degrees rudder angle. In line plots, the pressure side is coloured in red when the suction side is in blue.

7.4.3 Velocity field

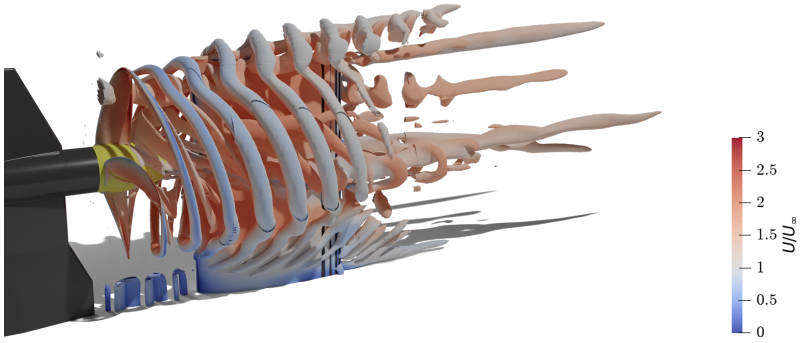
To better understand the error patterns and their influence on the flow field, this section focuses on the analysis of the velocity field. Figure 7.21 shows, for the different interpolation schemes, iso-surfaces of Qcriterion [51]. This quantity is a scalar field computed from the velocity components following equation 7.3, where Ω is the vorticity tensor and S strain rate tensor. It is used to highlight coherent structures in a turbulent flow by displaying shear layer vortices.

$$Q = \frac{1}{2}(||\Omega||^2 - ||S||^2) \tag{7.3}$$

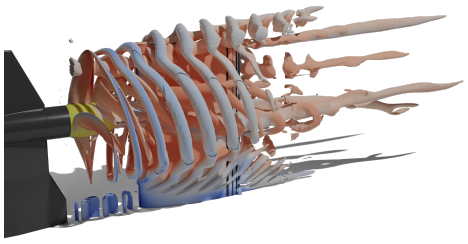
For such flow, it renders some features clearly identifiable like helices formed by blade tip vortices or the hub vortex. The later being split in two vortices by the rudder's leading edge. At 10 degrees angle of attack, the rudder also generates two vortices at the leading edge and the trailing edge of its tip section, the one on the leading edge then blends with the wake of the centre board placed upstream of the propeller. Finally, flow acceleration in the propeller wake is visible by the higher velocity (in red), but also, by the increase of helix pitch for vortices closer to the hub vortex. Besides these flow features, visualisation artefacts from the overset meshes are also visible. On blades' tip vortices, for example, a 'step' is seen wherever the vortex crosses an overset boundary. These discontinuities are

intrinsic to the overset method as separate iso-surfaces are constructed for each mesh. The difference in sizes and cell locations on both sides of the overset interface means that the iso-surfaces do not perfectly match across the boundary.

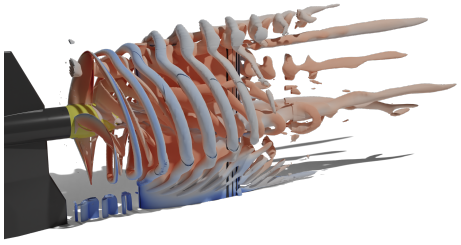
When comparing the different interpolation schemes, however, only marginal changes can be seen with such visualisation. The three schemes all show the same main flow features as described in the previous paragraph. Moreover, lower order schemes (like *Inverse distance*) are not excessively diffusing vortices at the overset interface compared to higher order ones like is sometimes the case for free surface flows [70]. In fact, only the wake and hub vortex are slightly different, but this is likely due to the chaotic characteristics of the flow in these regions rather than to the interpolation method itself.



(A) *Least squares*



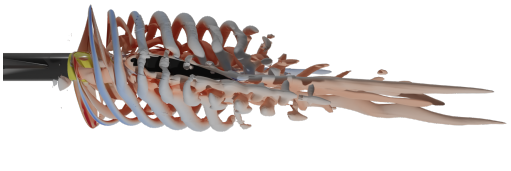
(B) *Inverse distance*



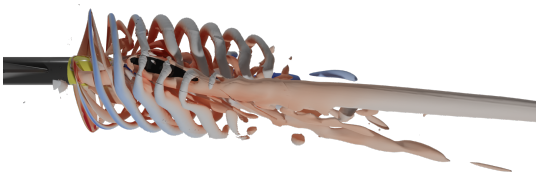
(C) *Nearest cell gradient*

FIGURE 7.21: Iso-surface of Qcriterion coloured by velocity for the tested interpolation schemes with a 10 degrees rudder angle on mesh assemblies G4.

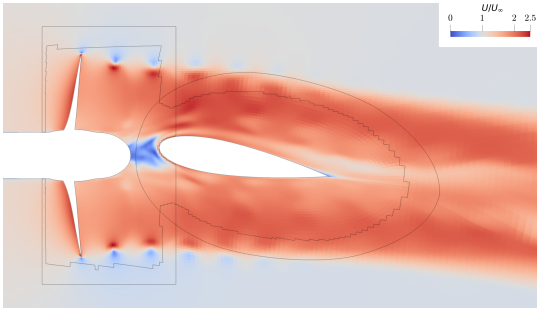
In order to analyse the flow more quantitatively, Figure 7.22 shows a top view of the domain with, from top to bottom, iso-Qcriterion, the velocity field at the propeller hub height with *Least squares* interpolation scheme, and a comparison with *Inverse distance* and *Nearest cell gradient* computations, the left column showing 10 degrees rudder angle and the right ones 20 degrees.
From the iso-Qcriterion and velocity slices, Figures 7.22a to 7.22d, the flow features highlighted in the previous section are still noticeable. Furthermore, at 20 degrees, flow separation can be observed at the rudder's trailing edge, the wake being diverted towards the suction side even more. In both cases this wake orientation is not only due to the rudder angle but also to the entire system orientation the propeller and centre board being at a drift angle of 7.5 degrees and therefore misaligned with the tunnel and



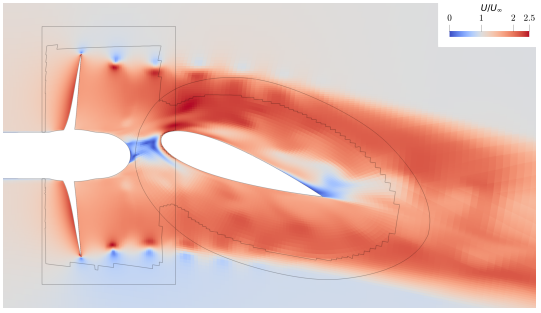
(A) *Least squares* iso-Qcriterion, 10 degrees



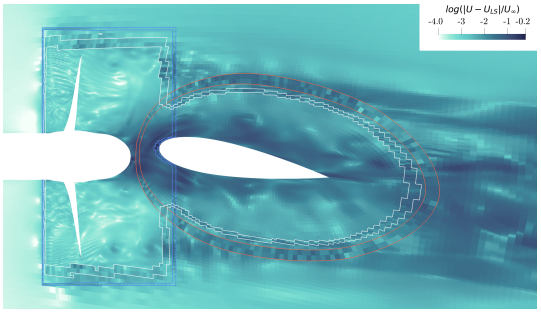
(B) *Least squares* iso-Qcriterion, 20 degrees



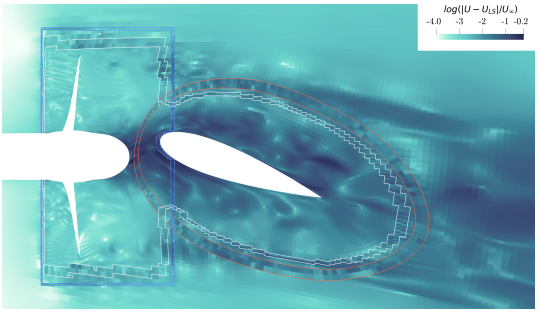
(C) *Least squares*, 10 degrees



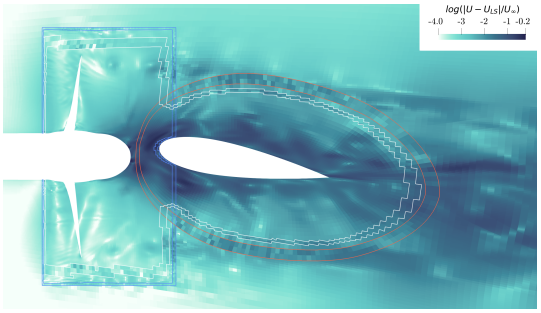
(D) *Least squares*, 20 degrees



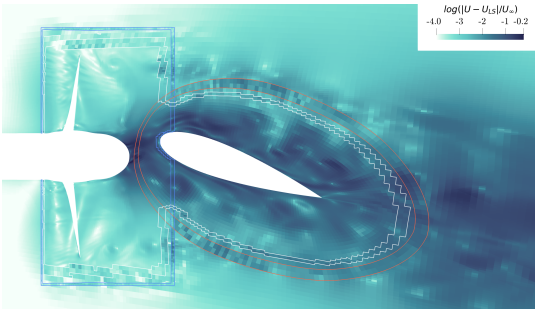
(E) *Inverse distance*: $log(|U_{InvDist} - U_{LS}|/U_\infty)$, 10 degrees



(F) *Inverse distance*, 20 degrees

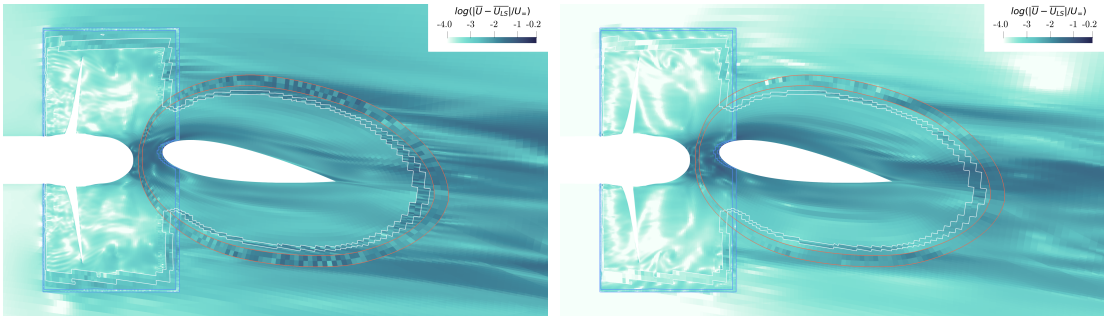


(G) *Nearest cell gradient*: $log(|U_{NCGrad} - U_{LS}|/U_\infty)$, 10 degrees



(H) *Nearest cell gradient*, 20 degrees

FIGURE 7.22: Comparison of the different interpolation schemes flow. 7.22a and 7.22a show iso-Qcriterion for the *Least squares* computations, 7.22c and 7.22d show the velocity field for the same computation, and finally 7.22e to 7.22h compares *Inverse distance* and *Nearest cell gradient* velocity fields to *Least squares* ones. In the last set of plots, *fringe* cells are highlighted in white for the tunnel mesh, blue for the propeller one and finally red for the rudder. 10 and 20 degrees rudder angles are shown in the left and right columns respectively.



(A) *Inverse distance* (B) *Nearest cell gradient*

FIGURE 7.23: Top view of the domain showing a comparison of the time averaged velocity fields with *Least squares* computation, at 10 degrees rudder angle. *fringe* cells are highlighted in white for the tunnel mesh, blue for the propeller and red for the rudder.

domain inlet. When comparing the other interpolation schemes with the *Least squares* computation, the hub vortex area shows the largest differences, once again likely due to the chaotic property of the flow at that location. Larger differences are also noticeable on each propeller tip vortex also coinciding with the rudder *fringe* cells region. These differences are finally convected downstream. On *fringe* cells themselves, the difference field is not as smooth as elsewhere because of interpolation errors, the *Inverse distance* computation showing more interpolation artefacts than the *Nearest cell gradient* one with larger cell-to-cell variations.

Finally, it is interesting to notice that the comparison of time averaged velocity fields, as seen in Figure 7.23, intensifies these cell-to-cell variations. Indeed, while other differences seem to average out, the differences located on *fringe* cells are still present and emphasised. This can be explained by the fact that the rudder and tunnel meshes are not moving, *fringe* cells then always have the same set of *donor* cells, leading to consistently over or under predicting their interpolated field values. These differences then, once again, follow streamlines and propagate downstream. This effect, however, affects less the *Nearest cell gradient* plot as it has both overall lower differences and smoother *fringe* cells regions compared to the *Inverse distance* one.

Even though these plots do not directly show interpolation or discretisation errors, the large difference in methodology between the three schemes demonstrates that the *Nearest cell gradient* and *Least squares* methods create fewer artefacts, produce overall smoother interpolated fields and have lower interpolation errors compared to the *Inverse distance* scheme. This conclusion is in line with the work done in Chapter 6 using manufactured solution for which errors could be probed directly. With only the rudder-propeller flow study, however, it is not possible to state which of *Nearest cell gradient* or *Least squares* computations have less interpolation errors and lead to a smoother interpolated fields.

7.5 Rudder flow Validation

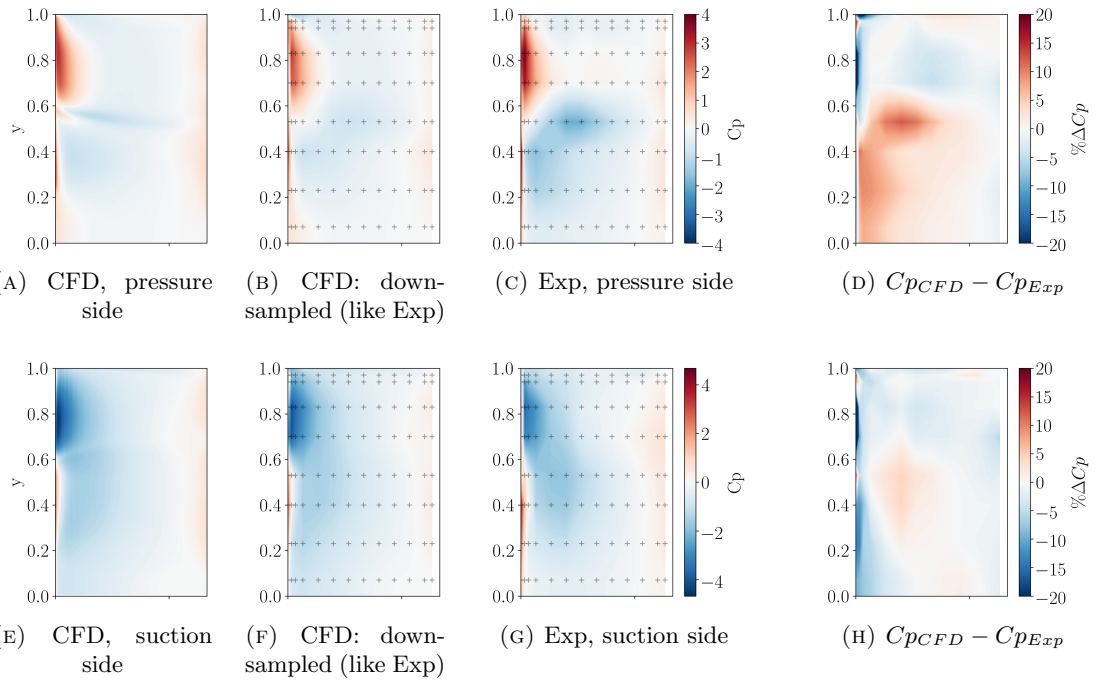
In this section, the validation is focussed on the rudder because as long as the thrust loading generated by the propeller is correct [81] then the rudder's performance is the most sensitive to errors induced by the interpolation scheme as onset flow goes through propeller and then into rudder overset mesh itself.

As discussed in section 7.2, the propeller geometry used in the CFD analysis is different from the experimental one, as a result, direct Validation of the entire setup is not possible. Rudder characteristics can however be validated using K_T equivalent computations. Experiments done at $J = 0.51$ with a -7.5 degrees drift angle and a 10 degrees rudder angle led to $K_T = 0.242$ [79]. Thus, in order to replicate the propeller wake, albeit with a different propeller geometry, a CFD computation has been done at $J = 0.61$ with similar drift and rudder angles which resulted, on grid G4, in $K_T = 0.235$. This is less than 3.5% away from the experimental value and, considering the Verification study done at $J = 0.51$, it was deemed close enough for the purpose of this Validation. The rest of this section, then, compares experimental data with this CFD computation done at $J = 0.61$, starting with Table 7.3 showing integral quantities. As said, K_T is, by design, close to the experimental one, K_Q is, however, quite far off with 20% difference. Which could be expected when using such methodology. On the other hand, rudder force coefficients are within 10% of the experimental ones, with C_L even below 1%. These values justify the K_T-equivalent methodology and considering the Verification study and resulting numerical uncertainties concur in the Validation of the rudder flow. The moment coefficient taken at 30% chord sees a larger difference which led to a 4.3% chord length offset in the centre of pressure C_{pc}.

TABLE 7.3: Comparison between CFD and experimental [79] integral quantities for the K_T equivalent Validation.

	CFD	**Exp**	**comp**
J	0.61	0.51	
K_T	0.2350	0.243	-3.28%
K_Q	0.0354	0.045	-21.32%
C_L	0.5903	0.587	0.56%
C_D	0.1055	0.114	-7.46%
C_m	-0.0520	-0.078	-33.37%
C_{pc} (in % chord)	21.3174	17.02	4.29

In the experimental campaign, the rudder was equipped with pressure probes. This pressure was, however, recorded only on cases without the upstream centre board which can have some effects on the pressure distribution. The comparison of the rudder pressure, then, should be taken with care and is here only qualitative as it compares data with a centre board (CFD) and without (Exp). Figure 7.24 displays the pressure on the rudder for both CFD and experimental results. The first column shows raw CFD



(A) CFD, pressure side (B) CFD: downsampled (like Exp) (C) Exp, pressure side (D) $Cp_{CFD} - Cp_{Exp}$

(E) CFD, suction side (F) CFD: downsampled (like Exp) (G) Exp, suction side (H) $Cp_{CFD} - Cp_{Exp}$

FIGURE 7.24: Pressure coefficients on both sides of the rudder surface, comparison of CFD and experimental data from Molland and Turnock [79]. The first two columns show the CFD results, both raw and downsampled to the probes locations, then the third one show experimental data. Finally, the last column compares the two data sets.

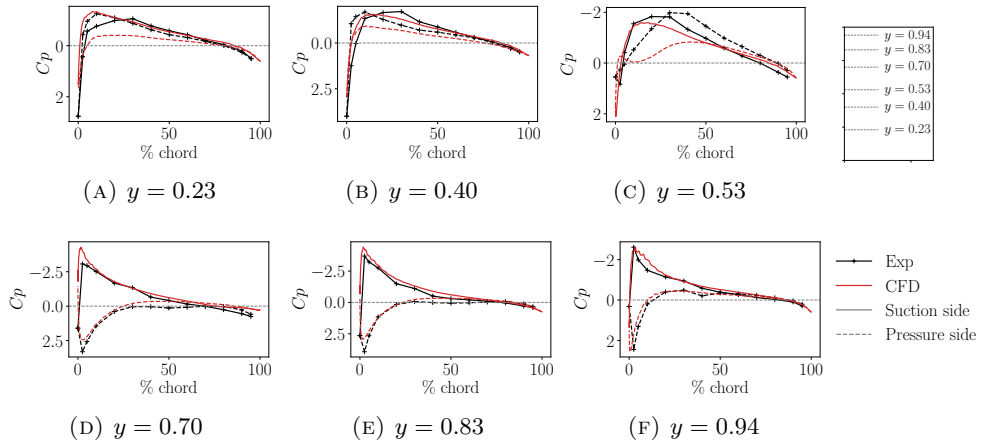
pressure coefficient on both sides when the second one downsamples it at the locations where the experimental pressure probes were placed for easier comparison with the third column, i.e. the experimental results. Finally, the last column compares the two sets of results.

Due to the low amount of pressure probes, some features disappear from the experimental and downsampled plots like the distinct lower pressure region of the hub vortex or the extent of the higher pressure region at the trailing edge. Overall, however, both the CFD and experimental plots show a high pressure region at the top of the pressure side, a similar low pressure region on the suction side etc. The CFD computation has lower C_p amplitude with high C_p regions not as high as experimental ones, and, in general, higher C_p in low pressure regions. These observations are highlighted in Figure 7.25 which shows chord-wise slices of C_p at various span heights. From them, it is more clearly visible that the top part of the rudder shows better correlation with experimental data and the hub vortex section ($y = 0.53$) significantly underestimate the pressure side C_p while the rest of the slope is better predicted.

The differences seen on the pressure profile can have various origins. Firstly, it can be related to the lack of match between the experimental and CFD setups, with the difference in propeller geometry, the K_T equivalent methodology or the lack of centre board in this sets of results in the experiments. Secondly, it can also come from modelling errors directly as the CFD simulations use unsteady RANS approach to model turbulence

(using the $k - \sqrt{k}L$ model). Thirdly, it can come from discretisation errors, as seen in section 7.4.2, the hub vortex region is where the discretisation uncertainty is the largest. Finally, it can also come from the overset interpolation upstream, and interpolation errors combined with the hub vortex could here create larger differences. It should also be noted that the literature related to this set of experiments did not provide uncertainty quantifications on the measurements, though it would likely be below the differences seen here.

To conclude, even though some differences can be originating from the K_T-equivalent methodology and different geometries, the very close correlation of rudder coefficients like C_L, C_D or the centre of pressure as well as the overall good capture of the features present on the rudder surface lead toward a favorable Validation of the flow. The circumstances of this study, however, make it harder to quantify the degree of confidence of this Validation.



(A) $y = 0.23$ (B) $y = 0.40$ (C) $y = 0.53$

(D) $y = 0.70$ (E) $y = 0.83$ (F) $y = 0.94$

FIGURE 7.25: Pressure coefficient on the rudder's surface at various span sections comparing CFD and experimental data from Molland and Turnock [79]

7.6 Concluding remarks

In this Chapter, a comprehensive solution Verification and error source analysis as well as Validation allowed to confirm and expand the conclusions drawn on analytical and manufactured solutions done in Chapter 6. Indeed with the simulation of a rudder-propeller interaction flow, a problem of direct importance for any ship computation, this Chapter also introduces mesh motion with the propeller rotation.

Similarly to the manufactured solution study, the 1st order *Inverse distance* continues to produce more artefacts on *fringe* cells compared to the 2nd order *Nearest cell gradient* and 3rd order *Least squares* schemes. Moreover, these interpolation errors are not cancelled out when time averaging the velocity field, highlighting the fact that errors on non moving *fringe* cells are consistently over- or under-predicted. These artefacts,

then convect downstream, and are the cause, together with the mesh motion, of high frequencies oscillations on the different force coefficients on the propeller and rudder. A temporal spectral analysis quantified these oscillations as having more than twice the energy as the ones produced by the 2$^{\text{nd}}$ order *Nearest cell gradient* or 3$^{\text{rd}}$ order *Least squares* schemes. Besides impacting the accuracy, those spurious oscillations can be an error source when doing common propeller acoustics studies, and cavitation flow simulations. It is important to note, however, that these oscillations do not result in significant changes in time averaged force coefficients as the differences produced is far smaller than the discretisation uncertainty. Similarly, time averaged rudder pressure coefficients were also comparable for the three tested schemes, with the largest difference in the hub vortex region which is also where the space discretisation uncertainty is maximum.

Another source of error of the overset method is its lack of mass flux conservation between grids that have been overset, even in a theoretically conservative finite volume discretisation. By inspecting each mesh individually, it was observed that the 3$^{\text{rd}}$ order accurate *Least squares* interpolation achieved mass flux imbalance one order of magnitude lower than the other two schemes, and the *Nearest cell gradient* performed better than the *Inverse distance* scheme. Here lies one of the difference with the recirculation bubble manufactured solution, for which the different interpolation schemes had minor to no effects on the flux conservation. It can be related to the complexity of the flow leading to higher differences in mass conservation or the addition of mesh motion with this test case.

Finally, the Validation of the rudder flow, done with K_T-equivalent computations, showed promising results. Integral quantities were found within 10% of the experimental ones and Cp on the rudder surface captured the main complex flow features such as the hub vortex, high and low pressure regions due to the propeller rotation. The uncertainties related to the different propeller geometries and K_T-equivalent computation should be lifted to fully perform the Validation study.

Chapter 8

Performance and scalability of the overset method

The accuracy of the overset method and the influence of interpolation schemes has been thoroughly studied in Chapter 5, 6 and 7. This is, however, not enough to give guidelines as the performance aspect of the overset method also needs to be considered. In this Chapter, the different components of the overset implementation are analysed in terms of performance and parallel scalability.

8.1 Introduction

In the literature, performance is sometimes discussed, though not often with a lot of detail. The general consensus, however, is that the overset method can be expensive. Gatin et al. [43] for example reports that a third of the computational time is dedicated to the overset method and Ohashi [91] observes 60% with their implementation. Windt et al. [129] compares the early overset implementation done in OpenFOAM (v1706) to the deforming mesh method in the same CFD solver and concludes that the overset method is 3.7 times more expensive in serial and also does not scale as well as the deforming mesh method, which makes the overset method very expensive in this case. Besides measuring the overall computational time of the method, looking at the different interpolation schemes can be informative. This is what Verma and Hemmati [124] did for the OpenFOAM implementation. They tested the *Inverse distance* scheme available for the overset method in this version against *Least squares* and 'CellVolumeWeight' a conservative interpolation method. They found the *Least squares* to be 1.4 times slower than the *Inverse distance* but more than two times faster than the conservative method. Though, given the limited difference found on integral quantities they considered the *Inverse distance* scheme to be sufficient. In the literature, papers describing 'new' interpolation schemes often compare their performance with more traditional ones. Like in

Sharma et al. [113] where a *RBF* implementation is compared to a *Polynomial* scheme or in Noack et al. [89] where a Dual-grid interpolation method is compared to the *Least squares* scheme already available in Suggar++. It should be noted, however, that these comparisons are often not made in view of the overall computational time, which can be deceiving. Indeed, having a scheme 1.4 times more expensive than another one is relevant only if the interpolation computation itself takes a significant amount of the overall computational time or if it does not scale well. Finally, besides comparing different interpolation schemes, some studies look at ways to improve performance overall, regardless of the interpolation method being used. Shen et al. [115] or Martin et al. [71] for example, designed a *lagged* method for a coupling with Suggar++ (with OpenFOAM and CFDShip-Iowa respectively). Since Suggar++ does not scale well with core count, several independent instances are run in serial working on different time-steps ahead of time effectively limiting the idle time of the solver at the cost of some inaccuracies in the DCI computation (due to the prediction of the mesh locations). Finally, Djomehri et al. [30] and Wissink and Meakin [131] looked at MPI asynchronisation and load balancing respectively to reduce idle time due to the parallel execution.

In contrast to the analysis of a scheme's accuracy, for which it's mathematical expression is enough to allow reproducibility, performance is highly implementation dependant. It is, however, still relevant to study it thoroughly to see the effects of implementation decisions, isolate bottlenecks, compare the different schemes between each other, and overall give achievable timing and scaling for other implementations to compare to.

In this Chapter, the raw performance and scalability of each of the steps forming the overset method as implemented in this work are analysed; always in light of the overall computational time. After a presentation of the methodology used, the DCI computations done by Suggar++ are examined in section 8.2.2, then the *donor* search and interpolation are studied in section 8.2.3. Finally, very few studies discuss the importance of the iterative convergence in a performance context, because an explicit overset coupling will affect the iterative convergence, as more outerloops are required to reach the same level of iterative error. Section 8.3 of this Chapter discusses this aspect of performance by comparing computations with and without overset.

8.2 Overset method performance

8.2.1 Methodology and setup

Computations done in this Chapter were run on MARIN's cluster Marclus4, a 4200 cores machine hosted in-house. Each computational node is composed of a dual-socket Intel Xeon (E5-2660 v3 @ 2.60GHz) 10 cores CPU, leading to 20 cores per node and Infiniband HBA is used for inter-node communications. The code is compiled using Intel

compiler and Intel MPI for the parallelisation (2018.4). For each computation, the time of each routine was recorded using PETSc [2] solver logging capabilities, and, on parallel runs, the time of the slowest process is used.

Most of the measurements were run on Grid 160 of the recirculation bubble test case, a 1M cells mesh assembly composed of two grids presented in section 6.2.2. Some results also come from the grid G4 of the rudder propeller test case, a 37M cells mesh assembly made of three grids. In both cases, the computation purposefully requires a new DCI computation, *donor* search and interpolation every time-step, even on the recirculation bubble test case, to emulate the performance with mesh motion. Finally, the interpolation is updated every outerloop, using 90 outerloops per time-step.

To have sufficient statistical convergence on the timing results, each function was run at least 500 times and the timing includes the computation itself as well as the parallel communication required. Moreover, to prevent data caching between consecutive runs, functions are not run in isolation but inside the real CFD computation. Each computation is repeated using 1-1000 processes as summarised in Table 8.1 for cases using the recirculation bubble assembly.

TABLE 8.1: Average number of cells and *fringe* cells per process (for the recirculation bubble case). Computations up to 20 cores are run on a single node. Here, N_{proc} denotes the number of processes used for the parallelisation, N_i is the total number of cells and N_{fringe} is the total number of *fringe* cells.

$\mathbf{N_{proc}}$	$\mathbf{N_i/N_{proc}}$	$\mathbf{N_{fringe}/N_{proc}}$
1	9 216 000	259 837
2	4 608 000	129 918
10	921 600	25 983
20	460 800	12 991
40	230 400	6 495
80	115 200	3 247
160	57 600	1 623
300	30 720	866
460	20 034	564
620	14 864	419
800	11 520	324
1000	9 216	259

For comparison purposes, individual timings of functions are expressed as a percentage of a baseline set of computations for which the timing of overset and interpolation methods has been subtracted. The ratio between the interpolation time ($T_{interpolation}$) and the time taken by the rest of the computation ($T_{overall} - T_{interpolation}$) is used. This means that this number could be higher than 100 if the time taken by the interpolation is higher than the time taken by the rest of the computation. Similar measures are computed for the *donor* searching method.

$$\%\ of\ total\ time = 100 \times \frac{T_{interpolation}}{T_{overall} - T_{interpolation}} \tag{8.1}$$

Table 8.2 shows the interpolation schemes and parameters used in this analysis. *Least squares*, *Inverse distance* and *Nearest cell* interpolation use interpolation weights that are computed once per time-step and reused within each outerloop, while other schemes need the interpolation to be fully recomputed every outerloop (note that the *donor* search is done only once per time-step).

TABLE 8.2: Interpolation schemes and parameter tested in this study.

Scheme	**Nb *donor* cells**
Nearest cell	1
Nearest cell gradient	1
Inverse distance	7
	15
Least squares degree 1	6
Least squares degree 2	20
Least squares degree 3	34
Barycentric type 2	1

In this section, strong scalability results are presented. Strong scalability shows how the timing of a computation evolves when more processing power (here more cores) are used. It is opposed to weak scalability for which a constant processing power is used on increasingly more expensive problems (in CFD this would be done with increasing the cell count). Strong scalability has been chosen because it is more relatable and useful in an engineering context where the cell count is dictated by the desired accuracy and the core count is then selected based on the efficiency of the particular solver and hardware.

8.2.2 DCI computation

In the present implementation, the DCI and IBLANK information are computed by the external library Suggar++ in serial. In a computation with mesh motion, the DCI needs recomputing at every time-step and Figure 8.1 shows the time taken as a percentage of the CFD computation depending on the number of cores used for the flow solver (Suggar++ always using a single core).

The lack of scalability is apparent from the increasing relative time compared to the CFD computation, and goes up to 30% when 1000 cores are used. On such a mesh, due to the scalability of the CFD solver itself, simulations are usually run on 500 to 750 cores leading to an overhead of around 20 to 25%. Then, Figure 8.2 shows the library runtime relative to the grid cell count for both the recirculation bubble case and the rudder behind propeller one. Besides the different cell counts between the two cases, the complexity of the assembly also increases with the rudder propeller case having three

grids and the presence of bodies (centre board, propeller and rudder). In both cases, the library seems to scale linearly with the cell count but with a steeper gradient on more complex cases. Finally, on the rudder behind propeller case running on grid G4 (37M cells) on only 500 cores, the DCI computation running each time-step takes about 30% of the total runtime.

In both cases, the DCI computation takes a significant amount of time relative to the CFD solver itself and is increasingly more expensive as the number of cells increases, because, unlike the solver, it won't benefit from using more cores. In practice, however, when the meshes motion is periodic, like in this work, the DCI does not need to be computed as part of the CFD computation. Here, as explained in section 4.3, IBLANK cache files are used to isolate the DCI computation from the CFD one. The DCI is then run on a minimal amount of cores and for only a single period. When the motion is not pre-determined, other methods like presented in Martin et al. [71] or Shen et al. [115] can be used to reduce the CFD solver idle time.

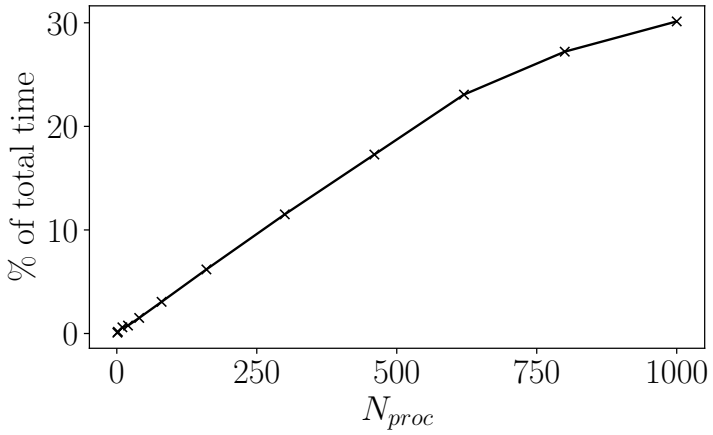


FIGURE 8.1: Time taken by Suggar++ when computing the DCI at every time-step as a percentage of the CFD solver's time when running the recirculation bubble test case on Grid 160.

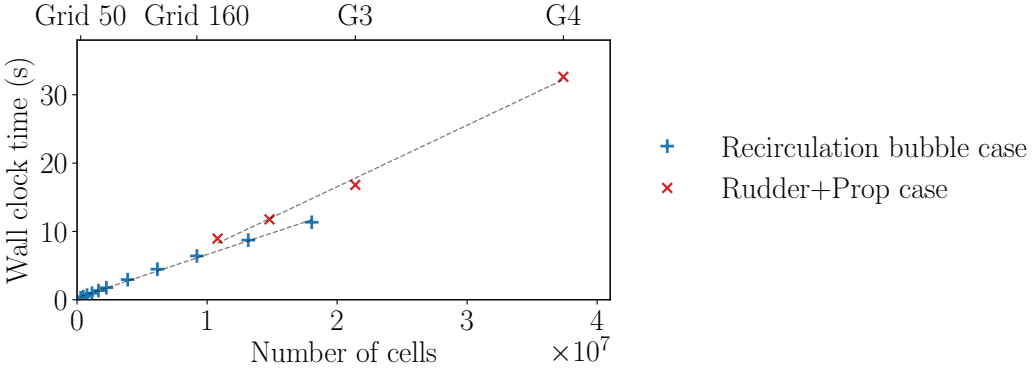


FIGURE 8.2: Wall clock time taken by Suggar++ to compute the DCI depending on the grid cell count for two different sets of meshes assemblies.

8.2.3 Donor search and interpolation

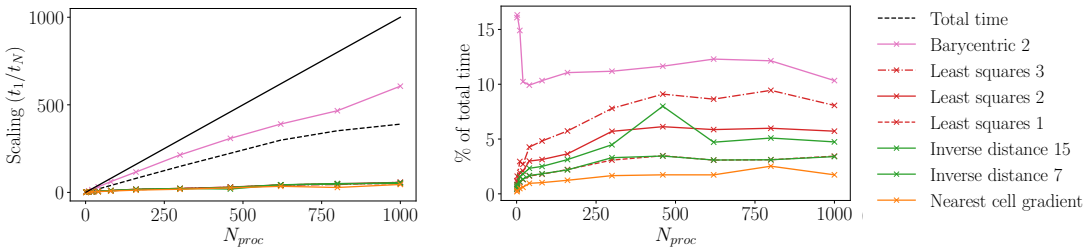
In this implementation, after the computation of the DCI, which provides the location of *fringe*, *hole* and *in* cells, the *donor* search is performed by the CFD solver, then followed by the computation of the interpolation itself. In this section, the performance of the interpolation is discussed before the *donor* search method as, depending on the interpolation scheme, a different amount of *donor* cells are required influencing the performance of the search.

8.2.3.1 Interpolation

For the interpolation performance, the timing considers the parallel exchange of *donor* cells field data and the interpolation computation itself, either by computing interpolation weights, computing the interpolated value directly or just using the already computed interpolation weights depending on the scheme. The interpolation step is concluded with the parallel exchange of the interpolated values to the process that requires them.

Figure 8.3a shows the strong scalability of interpolation schemes computation. The wall clock time is scaled based on the serial computation (using a single core). And, apart from the scalability of the different schemes themselves, the *Total time* line shows the scaling of the rest of the CFD code. With the solid black line denoting an ideal scalability, the CFD code scalability is seen to degrade slightly faster when using more than 600 cores.

Amongst the schemes tested, the *Barycentric* interpolation is the one that scales the best, with constant scalability even at a high core count. All the other schemes show similar levels of poor parallel scalability. This scalability behaviour is explained by the way the parallelisation is implemented. Like most finite volume method CFD codes, the solver uses domain decomposition where each core gets only a part of the full grid. In the current implementation, when using overset meshes, the interpolation is computed by the process that stores the first *donor* cell to minimise parallel communications. Since the *donor* cells are not scattered evenly in the entire domain, different loads are experienced by different cores, thereby affecting the scalability. This is shown in Figure 8.4 where the scalability of the most loaded core (in number of interpolations to perform) is plotted against the number of cores and compared with the scalability of the *Least squares* computation. This 'Max core load' number depends only on the mesh, the layout and the domain decomposition but not on anything hardware specific. From this plot, a close relation between the most loaded core and the scalability of the interpolation is seen. This suggests that the poor scalability is mainly related to the load balancing of the interpolation but to the parallel communication, for example. However, a different load balance method spreading the load over a wider range of cores should improve



(A) Strong scalability (B) Timing relative to Total time

FIGURE 8.3: Scalability and timing of different interpolation schemes compared to the CFD computation without overset (dashed line on scalability plot). The solid black line on shows ideal scalability.

scalability but also increase communications of *donor* cells field data. From this zoomed in plot, one can also note that the scalability is quite consistent over the range of process used as it forms a straight line.

The *Barycentric* interpolation is using a slightly different parallelisation method compared to the other schemes. It makes it scale very well, but it also makes it more time-consuming. Figure 8.3b shows the relative time each interpolation takes compared to the full CFD computation. The *Barycentric* interpolation is the most expensive out of all the tested schemes, especially at low core counts. Furthermore, the poor scalability of all the other schemes does not have a large influence on their timing. Overall, the timing degrades from 1 to about 50 cores, but stagnates when using more. Once again this is related to the domain decomposition, once the domain is subdivided enough to have some cores filled with *donor* cells, using more cores will keep a good scalability only reducing the amount of work done by each core.

To summarise, except for the *Barycentric* interpolation, the interpolation does not need more than 8% of the total time, with a minimum of 2% for the *Nearest cell gradient*, making the use of 2$^{\text{nd}}$ or 3$^{\text{rd}}$ order schemes affordable and in some case even cheaper than 1$^{\text{st}}$ order *Inverse distance*.

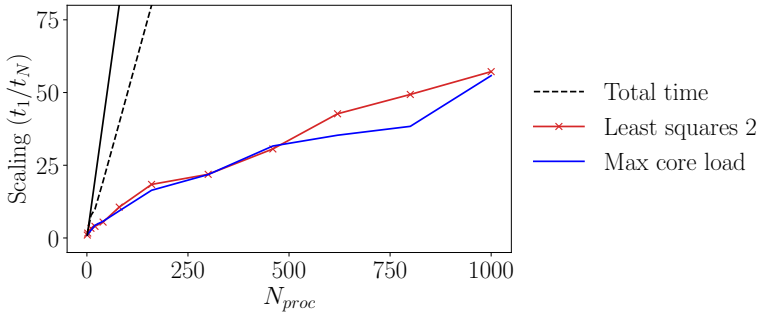
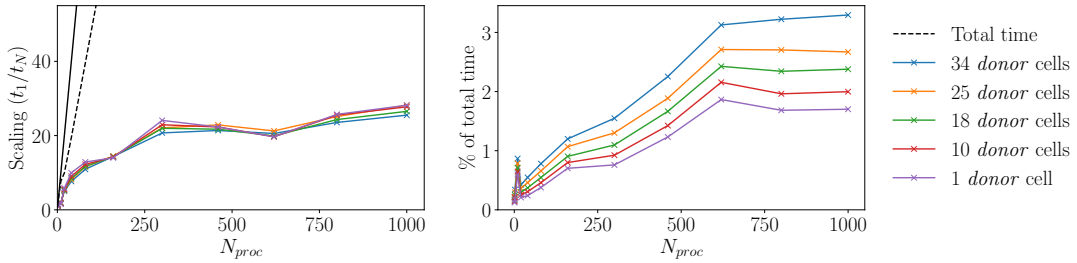


FIGURE 8.4: Scalability of the *Least squares* interpolation and of the number of interpolation computed by the most loaded core.

8.2.3.2 Donor search

For the *donor* cell searching, the timing includes the entire algorithm detailed in section 4.3.2 with the search for *donor* cells themselves plus a final parallel communication step to prepare the future interpolation. Similar to the interpolation performance analysis, Figure 8.5a shows the strong scalability of the *donor* search. This time, the scalability is worse than earlier as it is only 20 times faster on 1000 cores compared to 1. The scalability itself, however, is not really influenced by the amount of *donor* cells needed. Nonetheless, similar to the interpolation, even though the scalability is not ideal, the actual time taken by the algorithm is not dramatically high as shown in Figure 8.5b with a maximum below 3% of the total CFD computational time. It is also important to note that, once the first *donor* cell is gathered, the subsequent *donor* cells are a lot cheaper to collect. For example, requesting 15 *donor* cells is only about 20% more expensive than requesting a single *donor* cell. This again makes schemes that need more *donor* cells affordable.



(A) Strong scalability (B) Timing relative to Total time

FIGURE 8.5: Scalability and timing of the *donor* searching methods compared to a CFD computation without overset (dashed line on scalability plot). The solid black line on shows ideal scalability.

8.2.3.3 Combined performance

Finally, Figure 8.6 shows the combined time taken by both the *donor* searching and the interpolation itself taking into account the differences in *donor* cells requirements of each scheme. Overall, the fastest scheme is the *Nearest cell gradient* since it needs only a single *donor* cell and is trivially easy to compute, i.e. no system to solve like polynomial based schemes. More complex schemes like *Least squares*, however, only take 8% of the total time at most for a degree 2 (3^{rd} order scheme). The *Inverse distance* scheme, is seen to take between 5 and 7% of the total time depending on the number of *donor* cells required making it more expensive than the *Nearest cell gradient* but a cheaper option compared to the *Least squares* scheme. Even though this scalability study was done on the recirculation bubble test case, the rudder propeller one present similar results as shown with Figure 8.7, extending the findings to a largely different mesh assembly and cell counts.

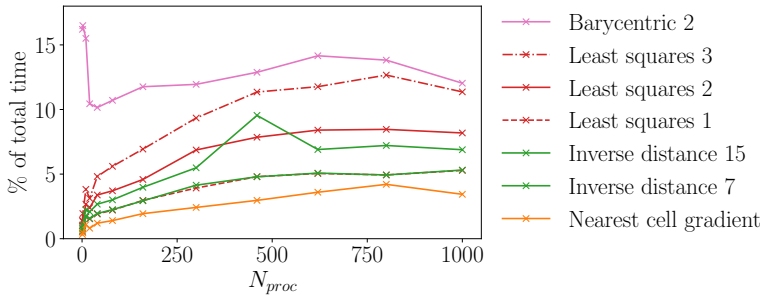


FIGURE 8.6: Time taken by both the *donor* searching and interpolation for all the tested schemes.

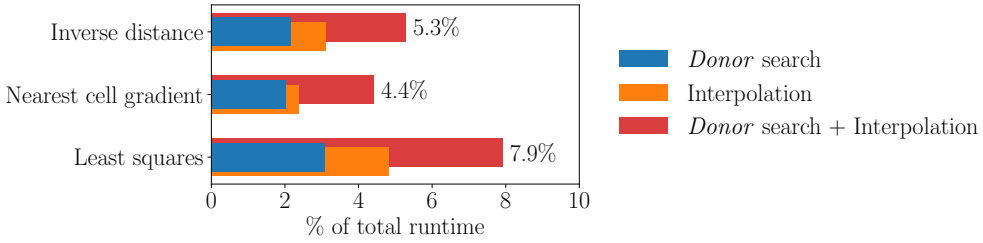


FIGURE 8.7: Time spent in overset related functions as a percentage of the total computation runtime for mesh G4 of the rudder propeller test case when using 500 cores.

8.3 Iterative convergence

The coupling between the different meshes modifies the system of equations as, for example, *fringe* cells do not depend on their direct neighbours. In this implementation the coupling is done explicitly, meaning that, as explained in Chapter 4, interpolated values are placed on the right hand side of the system corresponding to each *fringe* cell. While being easier to implement and requiring less parallel communications than an implicit coupling, it degrades the iterative performance of the solver by less tightly coupling the different meshes. In fact, interpolated data is updated only once per outerloop. In this section, the number of outerloops for reaching a L_∞ norm of 10^{-6} for all the equations residuals for the recirculation bubble test case is analysed. In this work, the momentum and turbulence equations were solved using the GMRES algorithm, and the pressure correction was solved with a CG solver. A block Jacobi pre-conditioner was used for all equations. It has to be noted that every computation done here uses the same under-relaxation parameters which may not be optimal, and individual runs could potentially be optimised to have better convergence behaviour.

Figure 8.8 shows the average number of outerloops needed for each computation as well as the difference with the non-overset run. Overall, layout L1 has better iterative convergence as, consistently, the different schemes need fewer outerloops to converge than on layout L2. Based on these results though, it is hard to draw a clear conclusion for each scheme. It appears that the convergence order or accuracy of the scheme does not play a significant role in the iterative convergence. This can be observed as the

Inverse distance scheme converges as well as a degree 2 *Least squares*. The degree 1 *Least squares* on layout L2 here shows a poorer convergence though, it was also having higher errors as presented in section 6.2.

Compared to the case without overset, however, it is clear that the coupling requires, in this case, about 50% more outerloops to achieve similar convergence. Since there is no reason for an implicit overset coupling to perform better than a single mesh computation, this means that, at most, the wall clock time could be reduced by around 30%. Concerning the comparison of the different schemes, it is safe to say that they do not affect meaningfully the iterative convergence unless a clear divergence behaviour is witnessed (like with layout L2 and *Least squares* 1).

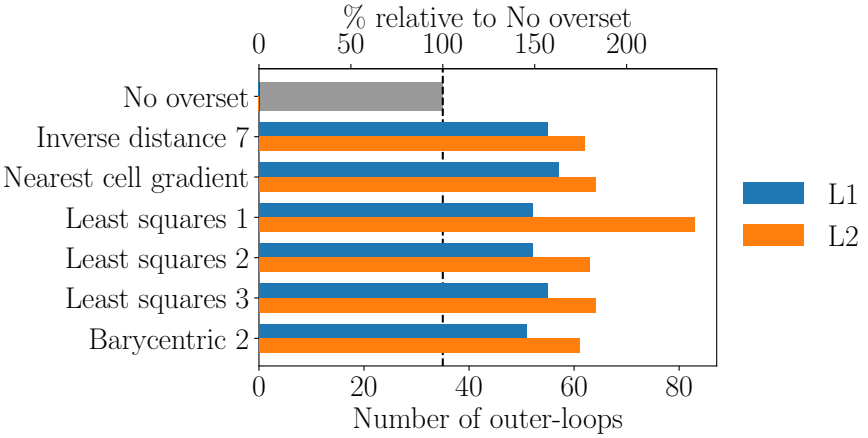


FIGURE 8.8: Average number of outerloops needed to converge each time-step to L_∞ residuals of 10^{-6} for each equation. Results are presented for layout L1 and L2 of the recirculation bubble manufactured solution on Grid 80.

8.4 Conclusion

In this Chapter, the different components affecting the performance of an overset computation were analysed. In this implementation, the lack of parallel scalability of the DCI computation resulted in an overhead of 20 to 30%, which could increase on more complex cases or if more cores are used. This means that any method to either reduce the amount of calls to the DCI or to reduce the idle time can have a large impact on the overall wall clock time. This is the reason why the IBLANK cache system was designed and used in all the test cases of this work, allowing to precompute the cell status data whenever the mesh motion can be predicted (like the periodic motion of the propeller's rotation).

Next, the interpolation and *donor* search, even with their non-ideal parallel scalability, has less than 8% overhead for the 3$^{\text{rd}}$ order *Least squares* scheme. 1$^{\text{st}}$ order *Inverse distance* is slightly cheaper with 5% to 7% overhead, and the 2$^{\text{nd}}$ order *Nearest cell*

gradient only has a 3% to 5% overhead. These differences, below 5% of a computation runtime, are small enough to justify the use of high order schemes for any overset computation. The benefits in terms of accuracy, reduce spacial and temporal noise and better mass conservation give them a clear advantage over their 1^{st} order counterparts. Interestingly, using only a single *donor* cell, the *Nearest cell gradient* scheme is both cheaper than the *Inverse distance* and more accurate (2^{nd} order versus 1^{st} order for *Inverse distance*). Higher order *Least squares*, due to the larger number of *donor* cells required and more complex scheme itself is more expensive with 10% to 12% overhead, close to the *Barycentric* interpolation. Though in both these cases the added cost does not give improved accuracy compared to 3^{rd} order *Least squares* making them less relevant for CFD solvers with 2^{nd} order spacial discretisation.

Finally, regardless of the interpolation scheme used, the explicit coupling of the meshes means that iterative convergence is affected, and, with the recirculation bubble test case, leads to 50% more outerloops compared to the same case without using overset. This performance hit can be partially compensated by using implicit coupling, at the cost of more expensive innerloops.

Chapter 9

Concluding remarks

9.1 Conclusion

Design and implement a novel overset method for unstructured grid solvers targeted to maritime applications.
The overset method is a powerful and essential feature of modern CFD software. It enables complex motions of multiple bodies, which is not feasible with other techniques. This project aimed to develop a new, efficient, and accurate method specifically for maritime applications. This was achieved by combining the reliability of external libraries for hole cutting computation and in-house development for integration, *donor* search, and interpolation. This architecture allowed for state-of-the-art features and versatility in interpolation methods. The accuracy and reliability of the overset method were tested on a variety of test cases, from academic examples like a low Reynolds 2D Poiseuille flow to complex industry applications like rudder-propeller interactions. Additionally, efforts were made to improve the method's performance, and both serial and parallel performance were measured.

Assess the robustness, accuracy and efficiency of overset methods.
In details, the robustness of the method, and of the interpolation schemes in particular was assessed first on a dedicated isolated test for polynomial schemes, and throughout the work by varying the mesh topologies and test cases. It was found that the *Polynomial* and *Polynomial tensor* schemes often resulted in high errors due to the resolution of an ill-conditioned system, particularly at high interpolation order. The *Least squares* scheme, on the other hand, even if also based on polynomial functions, could be made robust by using a larger number of *donor* cells. Depending on the meshes and interpolation order, from 1.5 to 2.5 more *donor* cells than the number of unknowns were required to guarantee its accuracy. All the other interpolation schemes tested were either bounded by construction, meaning that they cannot present large errors, ensuring their

robustness, or were based directly on the solver's gradient computations, also offering good robustness.

Interpolation always introduces errors. In the context of a computation using the overset method, these interpolation errors are directly present in the domain on *fringe* cells and affect the solution. Studying them is an essential step in any simulation involving the overset method. In this work, the different interpolation schemes were tested on three main test cases. The first two, a Poiseuille flow and a high Reynolds number manufactured solution of a recirculation bubble, allowed for code Verification as the exact solution for each of the fields was known. These test cases permitted to quantify and qualify accurately the errors present in the solution, allowing for discretisation and overset interpolation errors to be isolated and studied. From these, it can be determined whether or not the overset interpolation errors are higher than the intrinsic discretisation error coming from the mesh discretisation. Moreover, the convergence characteristics of the different errors can be specified using different cell sizes. Having an exact solution also allowed for the direct visualisation of errors, their propagation, pattern, smoothness etc. These findings from the first two test cases allowed for a better analysis of the final test case of this thesis, the interaction between a rudder and a propeller, for which, like in any industrial test case, no exact solution is known.
The importance of the method of manufactured solutions was highlighted by the fact that most conclusions drawn from the rudder-propeller test case were already known from the recirculation bubble test case. Because they can be designed to be realistic and close to particular problems and conditions they are an ideal tool to generalise and investigate errors in detail.

Finally, performance and efficiency of the implementation is essential for industrial use. For this reason, development efforts were focused at minimising parallel communications both in terms data being exchanged and number of communication steps, reducing cache misses, and optimising serial performance. In this work, the external library Suggar++ was used for the hole cutting to get cell status and the *donor* search and interpolations are both done inside the CFD solver. This architecture allowed to rely on the proven robustness of the library in terms of mesh assembly for complex cases and to keep complete control over the interpolation itself, which is a core component of the method's accuracy. While the library's parallel performance could be seen as a bottleneck, the *donor* search and interpolation computations were measured to take at most 8% of the total runtime for a 3^{rd} order *Least squares* interpolation or 4% for the 2^{nd} order *Nearest cell gradient* scheme. These low relative performance overheads allow to recommend these interpolation methods.

Draw guidelines on the usage and implementation of the overset method for maritime applications.

From the analysis of the robustness, error quantification and qualification, and performance as well as the variety of test cases used general guidelines concerning the overset method usage and its development in the context of finite volume unstructured grid discretisation CFD codes can be drawn. They are presented here:

- 1^{st} order interpolation schemes, such as *Inverse distance*, should be avoided. They, most of the time, provide worse accuracy without being faster than some of their 2^{nd} or 3^{rd} order counterparts. In cases where the interpolation error can be measured, they do not maintain the 2^{nd} order convergence of the code spacial discretisation scheme. In addition to generating larger errors, they produce more noise in both space and time, which, with moving meshes, translates to the production of high frequency variations of integral quantities that can be particularly detrimental when acoustic data is required (or absolute values of the pressure, in general). However, the results from 1^{st} order interpolation schemes should not be completely discarded. On complex flows, such as the rudder-propeller interaction, comparing time averaged integral quantities like force coefficients does not show a significant difference with other, higher order, schemes. This is because, on such cases, the discretisation error is higher than the overset interpolation error, and time averaging reduces the effect of the high frequency oscillations.

- 2^{nd} and 3^{rd} order interpolation schemes are both viable options for overset interpolation. The choice between them should then be directed by their different accuracy-performance balance. The 2^{nd} order *Nearest cell gradient*, by using a single *donor* cell, is the most affordable option as it requires only around 3-4% performance overhead for the *donor* search and interpolation. However, it does not perform as well as 3^{rd} order schemes at low Reynolds number and, in general, shows slightly worse mass conservation properties. To overcome these limitations, the 3^{rd} order *Least squares* scheme is the best option tested in this work. By using an overdetermined system with more *donor* cells than unknowns, it maintains good robustness compared to other polynomial based approaches, especially when the *donor* search cannot take advantage of the topology of a structured mesh. Moreover, it is still a viable option with only about 8% performance overhead. It also resulted in interpolation errors one order of magnitude lower than the discretisation error (for the recirculation bubble test case), making its impact on the solution's accuracy minimal.

- 4^{th} order and higher interpolation schemes are not necessary for overset interpolation when the solver's discretisation is 2^{nd} order, as they do not improve accuracy and are more expensive than 3^{rd} order schemes. Using the 4^{th} order *Least squares* scheme, however, resulted in a slightly 'smoother' field.

- Limiting the *donor* search to only the first layer of neighbours, as it is done in most implementations, can be detrimental to the robustness of *Least squares* interpolation. Allowing to gather more *donor* cells reduces the probability of high errors without being much more expensive. Indeed, since the search for the first *donor* cell is the bottleneck, subsequent new *donor* cells can be gathered using the mesh's topology at a minimal cost, even on unstructured grids.

- Parallel performance limitations of explicit hole cutting computations can be partially circumvented by using a cache system when possible (e.g. on periodic motion). Otherwise, a well parallelised explicit approach or other bypass mechanism (such as the *lagged* system presented in [115]) should be considered. Without scalability, the DCI computation in this implementation can accounts for 30% runtime overhead and increases with core count.

Help the research community by producing opensource tools to assist code Verification and uncertainty quantifications.
Verification has been a core component of this research and the source of most of the conclusion drawn. For this reason, the tools developed and used as part of it were made open source to help future research, not necessarily aimed at the overset method. This includes PyMMS [65], a python library that generates compilable code for the source terms of manufactured solution. The Navier-Stokes equations, as well as Spalart-Allmaras and one equation eddy viscosity model by Menter, are implemented, and the architecture allows for easy addition of extra models. Then, PyTST [66] is a library and interactive graphical interface that implements the Transient Scanning Technique to assess statistical uncertainty and detect transient portion of a signal.

Push for higher error analysis standards in maritime CFD.
In addition to the findings and conclusions, this work also aims to showcase the benefits and capabilities of Verification and Validation for CFD in the hope that more comparative studies will be done in the future. Additionally to analysing errors, they help build trust in the methods and provide a clearer path for further investigations as bottlenecks and shortcomings are identified.

9.2 Future work

The findings of this work lead to several ways to continue the research. First, the tool made to generate manufactured solution can be used more extensively. Following the collaboration with Gomes et al. [45], which lead to the production of a wind turbine manufactured solution, more realistic manufactured solution could be designed and published to serve as baseline cases for the study of overset or CFD solvers in general. Then, it would be worthwhile to investigate the use of an implicit coupling instead of an

explicit one, as it should improve iterative convergence at the cost of more expensive innerloops. An implicit coupling can reduce wall clock time and since most of the schemes were designed to use interpolation weights, they would not need any adaptation.

The work on error analysis and the influence of different interpolation methods should be continued for free surface flows. Manufactured solution could also be designed for these cases and used to test the implementation and behaviours of the interpolation schemes. The *donor* search might also require some adaptation as *donor* cells from one phase should not be used for the interpolation of another phase. Constraining the *donor* search to a single phase is a possibility to explore. Lastly, the large difference in density between the different fluids (1000:1 for water-air free surface) means that mass conservation can become a more important issue. Therefore, mass conservation methods should be investigated for such cases.

Over the years, several method for conserving mass with overset computations have been published, but no ubiquitous method seems to have emerged. Each publication designing and testing a new method. A study comparing different methodologies in terms of accuracy, spurious oscillation dampening, performance and ease of integration into existing solvers should be undertaken.

As seen in Chapter 8 on performance, the parallelisation of the interpolation could be improved with better load balancing at the cost of more inter-process communication. A balance between load balance and communication could be found to improve overall performance. Moreover, implicit hole cutting libraries should be further investigated or developed as they benefit from better parallel performance.

Finally, Validation should be continued and performed on a larger variety of industrial cases to potentially expand the conclusions of this work.

Appendix A

Implicit formulation of Polynomial schemes

To be able to use an implicit coupling, the interpolated value need to be a linear combination of the *donor* cell values, therefore under the form:

$$\tilde{\phi}(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \phi(\boldsymbol{x_i}), \tag{A.1}$$

with w_i the interpolation weights.

A.1 Polynomial and Polynomial Tensor schemes

In section 4.4.4.1, the system to be solved is:

$$[A_{k,m}][\alpha_k] = [rhs_k], \quad 1 \leq k, m \leq N. \tag{A.2}$$

With

$$\begin{aligned} A_{k,m} &= F_l(\boldsymbol{x_m}) \\ rhs_k &= \phi(\boldsymbol{x_k}) \end{aligned} \tag{A.3}$$

By inverting the matrix A, one can get explicitly the α coefficients:

$$\alpha_k = \sum_{i=1}^{N} A^{-1}_{k,i} \phi(\boldsymbol{x_i}) \tag{A.4}$$

And reconstruct the polynomial function:

$$\begin{aligned}\tilde{\phi}(\boldsymbol{x}) &= \sum_{k=1}^{N} \alpha_k F_k(\boldsymbol{x}) \\ &= \sum_{k=1}^{N} \left(\sum_{i=1}^{N} A^{-1}_{k,i} \phi(\boldsymbol{x_i}) \right) F_k(\boldsymbol{x}) \\ &= \sum_{i=1}^{N} \phi(\boldsymbol{x_i}) \left(\sum_{k=1}^{N} F_k(\boldsymbol{x}) A^{-1}_{k,i} \right) \\ &= \sum_{i=1}^{N} w_i \phi(\boldsymbol{x_i})\end{aligned} \tag{A.5}$$

With

$$w_i = \sum_{k=1}^{N} F_k(\boldsymbol{x}) A^{-1}_{k,i} \tag{A.6}$$

A.2 Least Squares scheme

In section 4.4.5, the $N_{terms} \times N_{terms}$ system needed to be solved is:

$$[A_{l,m}][\alpha_l] = [rhs_l], \quad 1 \leq l, m \leq N_{terms} \tag{A.7}$$

With:

$$\begin{aligned} A_{l,m} &= \sum_{i=1}^{N} F_l(\boldsymbol{x_i}) F_m(\boldsymbol{x_i}) \\ rhs_l &= \sum_{i=1}^{N} F_l(\boldsymbol{x_i}) \phi(\boldsymbol{x_i}) \end{aligned} \tag{A.8}$$

By inverting the matrix A, one can get explicitly the α coefficients:

$$\begin{aligned} \alpha_k &= \sum_{l=1}^{N_{terms}} A^{-1}_{k,l} rhs_l \\ &= \sum_{l=1}^{N_{terms}} A^{-1}_{k,l} \left(\sum_{i=1}^{N} F_l(\boldsymbol{x_i}) \phi(\boldsymbol{x_i}) \right) \end{aligned} \tag{A.9}$$

Then the complete polynomial function can be reconstructed:

$$\begin{aligned} \tilde{\phi}(\boldsymbol{x}) &= \sum_{k=1}^{N_{terms}} \alpha_k F_k(\boldsymbol{x}) \\ &= \sum_{k=1}^{N_{terms}} \left[\sum_{l=1}^{N_{terms}} A^{-1}_{k,l} \Big(\sum_{i=1}^{N} F_l(\boldsymbol{x_i}) \phi(\boldsymbol{x_i}) \Big) \right] F_k(\boldsymbol{x}) \\ &= \sum_{i=1}^{N} \phi(\boldsymbol{x_i}) \left[\sum_{k=1}^{N_{terms}} F_k(\boldsymbol{x}) \Big(\sum_{l=1}^{N_{terms}} F_l(\boldsymbol{x_i}) A^{-1}_{k,l} \Big) \right] \\ &= \sum_{i=1}^{N} w_i \phi(\boldsymbol{x_i}) \end{aligned} \tag{A.10}$$

With

$$w_i = \sum_{k=1}^{N_{terms}} F_k(\boldsymbol{x}) \Big(\sum_{l=1}^{N_{terms}} F_l(\boldsymbol{x_i}) A^{-1}_{k,l} \Big) \tag{A.11}$$

Appendix B

Recirculation bubble manufactured solution equations

This appendix presents the manufactured solution designed by Eça et al. [35] for a high reynolds recirculation bubble.

The domain is an empty box, with a wall at the bottom ($y = 0$) and the flow going in the x direction. The inlet is at $x = 0.1$ and outlet at $x = 1$. The domain expands in the z direction from $z = 0$ to $z = 1$. Each quantity is composed of two parts, a *base* flow, and a *perturbation* flow. The *base* flow defines a typical turbulent boundary layer flow and stays constant over time and throughout the domain in the z direction while the *perturbation* flow represents the recirculation bubble, and evolves in time and in the z-direction.

f_{time} defines the time variation of each quantity. As presented in Equation B.1, it is a sine wave oscillating between 0.2 and 1 with a period T.

$$f_{time}(t) = 0.2 + 0.4\left(1 + sin\left(\pi\left(\frac{2t}{T} - 0.5\right)\right)\right). \tag{B.1}$$

The *base* flow for the x component of the velocity is defined in u_x^b (Equation B.2) and represents a boundary layer flow. The shape of the recirculation bubble itself is defined using three parameters, a_1, a_2 and a_3 describing respectively the magnitude of the bubble, the location of centre of the bubble ($x = 0.5$, $y = 1/a_2$) and finally the decay of the bubble with distance to $x = 0.5$. u_x^p (Equation B.3) defines this 2D recirculation bubble.

$$u_x^b(x, y) = \sum_{i=1}^{3} \alpha_i^u \cdot tanh\left(a_i^u y x^{-b_i^u} Re^{1-b_i^u}\right) \tag{B.2}$$

$$u_x^p(x,y) = \left(1 - tanh\left(a_3\left(x^2 - x + 0.25\right)\right)\right) a_1 y e^{-a_2 y} \tag{B.3}$$

The x component of the velocity u_x is defined in Equation B.4. u_z is defined in Equation B.5. And finally, u_y (Equation B.6) is defined from u_x and u_z to satisfy the continuity equation.

$$u_x(x,y,z,t) = u_x^b(x,y) + u_x^p(x,y) \cdot sin^2(\pi z) \cdot f_{time}(t) \tag{B.4}$$

$$u_z(x,y,z,t) = \frac{\partial u_x^p}{\partial x} \cdot \frac{sin^2(2\pi z)}{4\pi} \cdot f_{time}(t) \tag{B.5}$$

$$u_y(x,y,z,t) = -\int_0^y \frac{\partial u_x}{\partial x} + \frac{\partial u_z}{\partial z} dy \tag{B.6}$$

The pressure field is defined in order to have a zero gradient normal to each domain boundary, and a pressure of 0 at the outlet ($x = x_{max}$). Equation B.9 defines C_p, similarly to u_x, a *base* flow is added to a fluctuating in time and in the z direction component.

$$P_x(x) = x\left(x\left(\frac{x}{3} - \frac{x_{min} + x_{max}}{2}\right) + x_{min}x_{max}\right) + 1 + \frac{x_{max}^3}{6} - \frac{x_{min}x_{max}^2}{2} \tag{B.7}$$

$$P_y(y) = y^2\left(\frac{y}{3} - \frac{y_{max}}{2}\right) + 1 + \frac{y_{max}^3}{6} \tag{B.8}$$

$$\begin{aligned} Cp\,(x,y,z,t) = {} & P \cdot log\,(P_x) \cdot log\,(P_y) + \\ & P_b \cdot cos\left(\frac{3\pi}{2} \cdot \frac{x - x_{min}}{x_{max} - x_{min}}\right)^2 \cdot cos^2\left(\frac{\pi}{2} \cdot \frac{y}{y_{max}}\right) \cdot sin^2(\pi z) \cdot f_{time}(t) \end{aligned} \tag{B.9}$$

Finally, the eddy viscosity is defined to have a near wall behaviour close to the wall ($y = 0$) and decays exponentially in the outer region as defined in Equation B.12. One can note that Equation B.12 is different from the one found in Eça et al. [35]. The one implemented in this work follows the source code used by the solution's authors.

$$p_{tm}^p = \left(1 + \left(a_1^d + a_2^d \cdot tanh\left(a_3^d\left(x^2 - x + 0.25\right)\right) - 1\right)\right) \frac{sin^2(\pi \cdot z)}{x^{0.8}} f_{time}\,(t) \tag{B.10}$$

$$y^+(x,y,z,t) = \sqrt{Re \cdot \frac{\partial u_x}{\partial y}\Big|_{y=0}} \cdot y \tag{B.11}$$

$$\tilde{\nu_t}(x,y,z,t) = \left(\left(\kappa y^+ - \tilde{\nu}_{out}\right) + \tilde{\nu}_{out}\right) \frac{e^{-y \cdot p_{tm}^p \cdot Re^{0.2}}}{Re} \tag{B.12}$$

The different variables being used in the current study are shown in Table B.1, they follow *case A* from [35].

TABLE B.1: Variables being used in the recirculation bubble manufactured solution (*case A* from [35])

Variable	**Value**
T	5
Re	10^7
P	500
P_b	0.25
x_{min}	0.1
x_{max}	1
y_{min}	0
y_{max}	0.4
$\tilde{\nu}_{out}$	1
κ	0.41

i	**1**	**2**	**3**
a_i	-140	40	16
α_i^u	0.35	0.4	0.25
a_i^u	0.0792	0.000063	0.005
b_i^u	0.2	0.2	0.2
a_i^d	0.4	0.6	10

The following source code implements the equations presented in this section in Sympy and runs PyMMS [65] to generates the fortran source file containing the source terms of this manufactured solution.

```
from sympy import *
from PyMMS import PyMMS

i = symbols('i', integer = True)
x, y, z, t = symbols('x y z t')

########################################
# Case initialisation
########################################
# user defined variables:
T = symbols('Period')
Re = symbols('Re')
As_1, As_2, As_3 = symbols('As_1 As_2 As_3')
Acp_1, Acp_2 = symbols('Acp_1 Acp_2')
rho = symbols('Rho')

# Definition of default values
global_vars = [(Re, 10**7),
               (As_1, -140),
```

```
                (As_2, 40),
                (As_3, 16),
                (Acp_2, 0.25),
                (Acp_1, 500),
                (T, 5),
                (rho, 1)]

# Common variable for the test case
Nu = 1/Re # L=1 and V=1

# Domain start
Xmin = 0.1

# Domain end
Xmax = 1

# Domain height (Ymin=0)
Ymax = 0.4

# Eddy viscosity at outlet
Emext = 1

As = [As_1, As_2, As_3]
Al  = Array([0.0792, 0.000063, 0.005])
Bl  = Array([0.2, 0.2, 0.2])
Alf = Array([0.35, 0.4, 0.25])
Aem = Array([0.4, 0.6, 10])
A1 = Al[i]*Re**(1-Bl[i])

########################################
# Definition of field functions
########################################

################################
# time component
Ftime = 0.2 + 0.4*(1+sin(pi*(2*t/T-0.5)))

################################
# Ums
A1 = Al[i]*Re**(1-Bl[i])
T1 = A1*y/x**Bl[i]
Upms = tanh(T1)
Byus = As[0]*y/exp(As[1]*y)
Bxus = 1-tanh(As[2]*(x**2-x+0.25))
Usms = Bxus*Byus

Ums = Sum(Alf[i]*Upms, (i, 0, 2)) + Usms*sin(pi*z)**2*Ftime

################################
# Wms
Wms = Derivative(Usms, x)*sin(2*pi*z)**2/(4*pi)*Ftime
```

```
71
72 ################################
73 # Vms = - Integral(Derivative(Ums, x)+ Derivative(Wms, z), (y, 0, y))
74 Byvs = As[0]*(y+1/As[1])/As[1]/exp(As[1]*y)
75 Vsms = (Byvs-As[0]/As[1]**2)*Derivative(Bxus, x)
76 Vpms = Bl[i]*x**(Bl[i]-1)/A1*log(Upms+1) + Bl[i]*y*(Upms-1)/x
77 Vms = Sum(Alf[i]*Vpms, (i, 0, 2)) + Vsms*(sin(2*pi*z)*cos(2*pi*z)+sin(pi*
       z)**2)*Ftime
78
79 ################################
80 # Cp
81 Pcpx = x*(x*(x/3-(Xmin+Xmax)/2)+Xmin*Xmax)+1+Xmax**3/6-0.5*Xmin*Xmax**2
82 Pcpy = y*y*(y/3-0.5*Ymax)+1+Ymax**3/6
83 Pcpsx = 1.5*(x-Xmin)*pi/(Xmax-Xmin)
84 Pcpsy = 0.5*pi*y/Ymax
85
86 Cpms = Acp_1*log(Pcpx)*log(Pcpy) + Acp_2*(cos(Pcpsx)**2 * cos(Pcpsy)**2)*
       sin(pi*z)**2*Ftime
87
88 ################################
89 # Nu_t_tild
90 Gx = tanh(Aem[2]*(x**2-x+0.25))
91 Fx = (1+(Aem[0]+Aem[1]*Gx-1)*Ftime*sin(pi*z)**2)/x**0.8
92 Pdl = Fx*y*Re**0.2
93 TWMS = diff(Ums, y).subs([(y, 0)])
94 Yplms = sqrt(Re)*sqrt(TWMS)*y
95
96 Nu_t_tildms = (exp(-Pdl)*(0.41*Yplms-Emext)+Emext)/Re
97
98
99 ########################################
100 # MMS generation and export
101 ########################################
102 mms = PyMMS(Nu=Nu,
103             rho=rho,
104             U=Ums,
105             V=Vms,
106             W=Wms,
107             P=Cpms,
108             Nu_t_tild=Nu_t_tildms,
109             wall_dist=y,
110             turbulence_model="SA-noft2")
111
112 mms.compute_sources()
113 mms.export_module("MMS-SA-noft2.F90",
114                   global_vars=global_vars)
```

ALGORITHM B.1: Source code for the definition of the recirculation bubble manufactured solution using PyMMS [65].

References

[1] *Standard for Verification and Validation in Computational Fluid Dynamics and Heat Transfer*. ASME, 2009. ISBN 9780791832097.

[2] Shrirang Abhyankar, Jed Brown, Emil M. Constantinescu, Debojyoti Ghosh, Barry F. Smith, and Hong Zhang. PETSc/TS: A Modern Scalable ODE/DAE Solver Library. *arXiv*, V(212):1–29, jun 2018. ISSN 23318422.

[3] Charles Erzan Badoe, Alexander B. Phillips, and Stephen R. Turnock. Influence of Drift Angle on the Computation of Hull–Propeller–Rudder Interaction. *Ocean Engineering*, 103(0):64–77, jul 2015. ISSN 00298018. doi: 10.1016/j.oceaneng.2015.04.059.

[4] John A Benek, Joseph L Steger, and F Carroll. Dougherty. A Flexible Grid Embedding Technique with Application to the Euler Equations. In *6th Computational Fluid Dynamics Conference Danvers*, Danvers, MA ,U.S.A., jul 1983. American Institute of Aeronautics and Astronautics. doi: 10.2514/6.1983-1944.

[5] John A Benek, Pieter G Buning, and Joseph L Steger. A 3-D Chimera Grid Embedding Technique. In *7th Computational Physics Conference*, page 10, Reston, Virigina, jul 1985. American Institute of Aeronautics and Astronautics. doi: 10.2514/6.1985-1523.

[6] David A Boger and James Dreyer. Prediction of Hydrodynamic Forces and Moments for Underwater Vehicles Using Overset Grids. In *44th AIAA Aerospace Sciences Meeting*, number January, pages 1–13, Reston, Virigina, jan 2006. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-039-0. doi: 10.2514/6.2006-1148.

[7] David A Boger, Ralph W. Noack, and E.G. Paterson. Dynamic Overset Grid Implementation in OpenFOAM. In *5th OpenFOAM Workshop*, volume 21, page 24, Gothenburg, Sweden, 2010.

[8] David A Boger, E.G. Paterson, and Ralph W. Noack. FoamedOver: a Dynamic Overset Grid Implementation in OpenFOAM. In *10th Symposium on Overset Composite Grids and Solution Technology*, Moffet Field, CA, USA, 2010.

[9] Ryan Bond, Patrick Knupp, and Curtis Ober. A Manufactured Solution for Verifying CFD Boundary Conditions, Part II. In *43rd AIAA Aerospace Sciences Meeting and Exhibit*, Reston, Virigina, jan 2005. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-064-2. doi: 10.2514/6.2005-88.

[10] Michael J. Brazell, Jayanarayanan Sitaraman, and Dimitri J. Mavriplis. An Overset Mesh Approach for 3D Mixed Element High-Order Discretizations. *Journal of Computational Physics*, 322:33–51, oct 2016. ISSN 00219991. doi: 10.1016/j.jcp.2016.06.031.

[11] Joris Brouwer, Jan Tukker, and Martijn van Rijsbergen. Uncertainty Analysis of Finite Length Measurement Signals. *The 3rd International Conference on Advanced Model Measurement Technology for the EU Maritime Industry*, (February), 2013.

[12] Joris Brouwer, Jan Tukker, and Martijn van Rijsbergen. Uncertainty Analysis and Stationarity Test of Finite Length Time Series Signals. In *4th International Conference on Advanced Model Measurement Technology for the Maritime Industry*, 2015.

[13] Joris Brouwer, Jan Tukker, Yvette Klinkenberg, and Martijn van Rijsbergen. Random Uncertainty of Statistical Moments in Testing: Mean. *Ocean Engineering*, 182 (April):563–576, jun 2019. ISSN 00298018. doi: 10.1016/j.oceaneng.2019.04.068.

[14] Cadence. Pointwise. URL https://www.pointwise.com.

[15] Pablo M. Carrica, Robert V. Wilson, Ralph W. Noack, and Frederick Stern. Ship Motions Using Single-Phase Level Set with Dynamic Overset Grids. *Computers & Fluids*, 36(9):1415–1433, 2007. ISSN 00457930. doi: 10.1016/j.compfluid.2007.01.007.

[16] Pablo M. Carrica, A Castro, J. Ezequiel Martin, and Ralph W. Noack. Overset Grid Technology Applied to Maneuvers of Marine Vehicles Background for ship hydrodynamics applications. In *11th Symposium on Overset Composite Grids and Solution Technology*, number October, Dayton, Ohio, 2012.

[17] Pablo M. Carrica, Farzad Ismail, Mark Hyman, Shanti Bhushan, and Frederick Stern. Turn and Zigzag Maneuvers of a Surface Combatant using a URANS Approach with Dynamic Overset Grids. *Journal of Marine Science and Technology*, 18(2):166–181, jun 2013. ISSN 0948-4280. doi: 10.1007/s00773-012-0196-8.

[18] William M Chan. Overset grid technology development at NASA Ames Research Center. *Computers & Fluids*, 38(3):496–503, 2009. ISSN 0045-7930. doi: 10.1016/j.compfluid.2008.06.009.

[19] Dominic D.J. Chandar. Development of a Parallel Overset Grid Framework for Moving Body Simulations in OpenFOAM. *Journal of Applied Computer Sciences & Mathematics*, 9(2):22–30, 2015. doi: 10.4316/JACSM.201502004.

[20] Dominic D.J. Chandar. Assessment of Interpolation Strategies and Conservative Discretizations on Unstructured Overset Grids in OpenFOAM. In *2018 AIAA Aerospace Sciences Meeting*, number January, pages 1–15, Reston, Virginia, jan 2018. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-524-1. doi: 10.2514/6.2018-0828.

[21] Dominic D.J. Chandar. On Overset Interpolation Strategies and Conservation on Unstructured Grids in OpenFOAM. *Computer Physics Communications*, 239: 72–83, jun 2019. ISSN 00104655. doi: 10.1016/j.cpc.2019.01.009.

[22] Dominic D.J. Chandar and Jayanarayanan Sitaraman. A Flux Correction Approach for the Pressure Equation in Incompressible Flows on Overset Meshes in OpenFOAM. *Computer Physics Communications*, 273:108279, apr 2022. ISSN 00104655. doi: 10.1016/j.cpc.2021.108279.

[23] Dominic D.J. Chandar, Bharathi Boppana, and Vasanth Kumar. A Comparative Study of Different Overset Grid Solvers Between OpenFOAM, StarCCM+ and Ansys-Fluent. In *2018 AIAA Aerospace Sciences Meeting*, Reston, Virginia, jan 2018. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-524-1. doi: 10.2514/6.2018-1564.

[24] Hamn-Ching Chen and Miaomou Chen. Chimera RANS Simulation of a Berthing DDG-51 Ship in Translational and Rotational Motions. *International Journal of Offshore and Polar Engineering*, 8(3):182–191, 1998. ISSN 10535381.

[25] Lin Cheng-Wen, Scott Percival, and Eugene Gotimer H. Application of Chimera Composite Grid Scheme to Ship Appendages. Technical report, Naval Surface Warfare Center, Bethesda Maryland, USA, Bethesda, Maryland, USA, 1995.

[26] L. Chiron, G. Oger, M. de Leffe, and D. Le Touzé. Analysis and Improvements of Adaptive Particle Refinement (APR) through CPU time, Accuracy and Robustness Considerations. *Journal of Computational Physics*, 354:552–575, feb 2018. ISSN 00219991. doi: 10.1016/j.jcp.2017.10.041.

[27] Menno Deij-van Rijswijk and Auke van der Ploeg. 80259-5-RD: Interpolation and Overset Grids. Technical report, MARIN, Wageningen, NL, 2021.

[28] Garbo Deng, Alban Leroyer, Emmanuel Guilmineau, Patrick Queutey, Michel Visonneau, and J Wackers. CFD Simulation of PMM Motion in Shallow Water for the DTC Container Ship. In *4th International Conference on Ship Manoeuvring in Shallow and Confined Water with Special Focus on Ship Bottom Interaction*,

pages 93–98, Hamburg, Germany, 2016. ISBN 9783939230380. doi: 10.18451/978-3-939230-38-0.

[29] Andrea Di Mascio, Giulio Dubbioso, Roberto Muscari, and Mario Felli. CFD Analysis of Propeller-Rudder Interaction. In *International Ocean and Polar Engineering Conference*, number June 21-26, pages 946–950, Kona, Big Island, Hawaii, USA, 2015. International Society of Offshore and Polar Engineers (ISOPE). ISBN 978-1-880653-89-0.

[30] M.J. Djomehri, R. Biswas, Mark A. Potsdam, and R.C. Strawn. An Analysis of Performance Enhancement Techniques for Overset Grid Applications. In IEEE, editor, *Proceedings International Parallel and Distributed Processing Symposium*, page 9, Nice, France, 2003. IEEE Comput. Soc. ISBN 0-7695-1926-1. doi: 10.1109/IPDPS.2003.1213158.

[31] F Carroll. Dougherty, John A Benek, and Joseph L Steger. On Applications of Chimera Grid Schemes to Store Separation. Technical report, NASA, 1985.

[32] Luís Eça. Polynomial Interpolation in Unstructured Grids. Technical Report September, IST, 2009.

[33] Luís Eça and Martin Hoekstra. Verification and Validation for Marine Applications of CFD. *International Shipbuilding Progress*, 60(1-4):107–141, 2013. ISSN 0020868X. doi: 10.3233/ISP-130083.

[34] Luís Eça and Martin Hoekstra. A Procedure for the Estimation of the Numerical Uncertainty of CFD Calculations Based on Grid Refinement Studies. *Journal of Computational Physics*, 262:104–130, apr 2014. ISSN 00219991. doi: 10.1016/j.jcp.2014.01.006.

[35] Luís Eça, Martin Hoekstra, and Guilherme Vaz. Manufactured Solutions for Steady-Flow Reynolds-Averaged Navier-Stokes Solvers. *International Journal of Computational Fluid Dynamics*, 26(5):313–332, 2012. ISSN 10618562. doi: 10.1080/10618562.2012.717617.

[36] Luís Eça, Guilherme Vaz, and Martin Hoekstra. Assessing Convergence Properties of Rans Solvers With Manufactured Solutions. *European Congress on Computational Methods in Applied Sciences and Engineering*, (Eccomas), 2012.

[37] Luís Eça, Guilherme Vaz, and Martin Hoekstra. Code Verification of ReFRESCO With a Statistically Periodic Manufactured Solution. In *ASME 33rd International Conference on Ocean, Offshore and Arctic Engineering*, page V002T08A015. American Society of Mechanical Engineers, jun 2014. ISBN 978-0-7918-4540-0. doi: 10.1115/OMAE2014-23258.

[38] Luís Eça, Christiaan M. Klaij, Guilherme Vaz, Martin Hoekstra, and Filipe Pereira. On Code Verification of RANS Solvers. *Journal of Computational Physics*, 310(January):418–439, apr 2016. ISSN 00219991. doi: 10.1016/j.jcp.2016.01.002.

[39] Luís Eça, Guilherme Vaz, and Martin Hoekstra. On the Role of Iterative Errors in Unsteady Flow Simulations. In *21st Numerical Towing Tank Symposium (NuTTS)*, pages 2–7, Cortona, Italy, 2018.

[40] Joel H Ferziger and Milovan Peric. *Computational methods for fluid dynamics*. Springer Science & Business Media, 2020. ISBN 3540420746.

[41] Evert-jan Foeth and Menno Deij-van Rijswijk. Remodeling the B-series Geometry in a CAD Environment. In Mario Felli and Cecilia Leotardi, editors, *6th International Symposium on Marine Propulsors*, number May, Rome, 2019. National Research Council of Italy, Institute of Marine Engineering (CNR-INM), Via di Vallerano 139, 00128 Rome, Italy.

[42] Norman Foster and Ralph W. Noack. High-Order Overset Interpolation Within An OVERFLOW Solution. In *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, number January, pages 1–8, Reston, Virigina, jan 2012. American Institute of Aeronautics and Astronautics. ISBN 978-1-60086-936-5. doi: 10.2514/6.2012-728.

[43] Inno Gatin, Vuko Vukcevic, Hrvoje Jasak, and I Lalovic. Manoeuvring Simulations using the Overset Grid Technology in FOAM-extend. In *32nd Symposium on Naval Hydrodynamics*, number August, page 10, Hamburg, Germany, 2018.

[44] Tiago Gomes, Sébastien Lemaire, Guilherme Vaz, and Fernando Lau. Verification Study of Sliding and Overset Grid Methods using the Method of Manufactured Solutions on a Wind Turbine flow. In *23rd Numerical Towing Tank Symposium (NuTTS)*, Mülheim an der Ruhr, Germany, 2021.

[45] Tiago Gomes, Sébastien Lemaire, and Guilherme Vaz. Code and Solution Verification of Sliding and Overset Grid Methods on Wind Turbine Flows. In *ASME 41th International Conference on Ocean, Offshore and Arctic Engineering*, Hamburg, Germany, 2022.

[46] Harish Gopalan, Rajeev Jaiman, and Dominic D.J. Chandar. Flow Past Tandem Circular Cylinders at High Reynolds Numbers using Overset Grids in OpenFOAM. In *53rd AIAA Aerospace Sciences Meeting*, pages 1–20, Reston, Virginia, jan 2015. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-343-8. doi: 10.2514/6.2015-0315.

[47] H. Hadzic. *Development and Application of a Finite Volume Method for the Computation of Flows Around Moving Bodies on Unstructured, Overlapping Grids*. PhD thesis, Technische Universitat Hamburg, 2005.

[48] James Hawkes. *Chaotic Methods for the Strong Scalability of CFD*. PhD thesis, University of Southampton, 2017.

[49] James Hawkes, Guilherme Vaz, Alexander B. Phillips, S. J. Cox, and Stephen R. Turnock. On the Strong Scalability of Maritime CFD. *Journal of Marine Science and Technology*, 23(1):81–93, mar 2018. ISSN 0948-4280. doi: 10.1007/s00773-017-0457-7.

[50] Guanghua He, Weijie Mo, Yun Gao, Zhigang Zhang, Jiadong Wang, Wei Wang, Pengfei Liu, and Hassan Ghassemi. Modification of Effective Angle of Attack on Hydrofoil Power Extraction. *Ocean Engineering*, 240:109919, nov 2021. ISSN 00298018. doi: 10.1016/j.oceaneng.2021.109919.

[51] J. C. R. Hunt, A. A. Wray, and P. Moin. Eddies, streams, and convergence zones in turbulent flows. In *Center for Turbulence Research Proceedings of the 1988 Summer Program*, number 1970, pages 193–208, 1988.

[52] Wolfram Research, Inc. Mathematica, Version 13.1. URL https://www.wolfram.com/mathematica. Champaign, IL, 2022.

[53] Chang-Ho Kang, Hamn-Ching Chen, and Erick T Huang. Chimera RANS/LAPLACE Simulation of Free Surface Flows Induced by 2D Ship Sway, Heave, and Roll Motions. In *8th International Offshore and Polar Engineering Conference*, pages 320–327, Montreal, Canada, 1998. International Society of Offshore and Polar Engineers.

[54] Eduardo Tadashi Katsuno, Artur K. Lidtke, Bülent Düz, Douwe Rijpkema, João L.D. Dantas, and Guilherme Vaz. Estimating Parameter and Discretization Uncertainties using a Laminar–Turbulent Transition Model. *Computers & Fluids*, 230:105129, nov 2021. ISSN 00457930. doi: 10.1016/j.compfluid.2021.105129.

[55] Aaron Katz and Venkateswaran Sankaran. Mesh Quality Effects on the Accuracy of CFD Solutions on Unstructured Meshes. *Journal of Computational Physics*, 230(20):7670–7686, aug 2011. ISSN 00219991. doi: 10.1016/j.jcp.2011.06.023.

[56] Andrew C. Kirby, Michael J. Brazell, Zhi Yang, Rajib Roy, Behzad R. Ahrabi, Michael K. Stoellinger, Jayanarayanan Sitaraman, and Dimitri J. Mavriplis. Wind Farm Simulations using an Overset HP-Adaptive Approach with Blade-Resolved Turbine Models. *The International Journal of High Performance Computing Applications*, 33(5):897–923, sep 2019. ISSN 1094-3420. doi: 10.1177/1094342019832960.

[57] Cetin C. Kiris, Jeffrey A. Housman, Michael F. Barad, Christoph Brehm, Emre Sozer, and Shayan Moini-Yekta. Computational Framework for Launch, Ascent, and Vehicle Aerodynamics (LAVA). *Aerospace Science and Technology*, 55:189–219, 2016. ISSN 12709638. doi: 10.1016/j.ast.2016.05.008.

[58] Christiaan M. Klaij and C. Vuik. SIMPLE-type Preconditioners for Cell-Centered, Colocated Finite Volume Discretization of Incompressible Reynolds-Averaged Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 71(7):830–849, mar 2013. ISSN 02712091. doi: 10.1002/fld.3686.

[59] Maarten Klapwijk, Thomas P. Lloyd, Guilherme Vaz, M. van den Boogaard, and Tom van Terwisga. Exciting a Cavitating Tip Vortex with Synthetic Inflow Turbulence: A CFD Analysis of Vortex Kinematics, Dynamics and Sound Generation. *Ocean Engineering*, 254:111246, jun 2022. ISSN 00298018. doi: 10.1016/j.oceaneng.2022.111246.

[60] Hiroshi Kobayashi and Yoshiaki Kodama. Developing Spline Based Overset Grid Assembling Approach and Application to Unsteady Flow Around a Moving Body. *Journal of Mathematics and System Science*, 6(9):339–347, sep 2016. ISSN 21595291. doi: 10.17265/2159-5291/2016.09.001.

[61] Sébastien Lemaire, Guilherme Vaz, and Stephen R. Turnock. Implementation and Verification of an Explicit Overset Grid Method. In *21st Numerical Towing Tank Symposium (NuTTS)*, Cortona, Italy, 2018.

[62] Sébastien Lemaire, Guilherme Vaz, and Stephen R. Turnock. On the Need for Higher Order Interpolation with Overset Grid Methods. In *22nd Numerical Towing Tank Symposium (NuTTS)*, Tomar, Portugal, 2019.

[63] Sébastien Lemaire, Guilherme Vaz, Menno Deij-van Rijswijk, and Stephen R. Turnock. On the Accuracy, Robustness, and Performance of High Order Interpolation Schemes for the Overset Method on Unstructured Grids. *International Journal for Numerical Methods in Fluids*, 94(2):152–187, feb 2022. ISSN 0271-2091. doi: 10.1002/fld.5050.

[64] Sébastien Lemaire, Guilherme Vaz, Menno Deij-van Rijswijk, and Stephen R. Turnock. Influence of Interpolation Scheme on the Accuracy of Overset Method for Computing Rudder-Propeller Interaction. *Journal of Verification, Validation and Uncertainty Quantification*, 8(1), mar 2023. ISSN 2377-2158. doi: 10.1115/1.4056681.

[65] Sébastien Lemaire. PyMMS: Generation of RANS Manufactured Solution for CFD using Sympy. January 2021. doi: 10.5281/zenodo.4428181. URL https://github.com/nanoseb/pymms.

[66] Sébastien Lemaire and Maarten Klapwijk. PyTST: Python Library and Command Line Tool Performing the Transient Scanning Technique. January 2021. doi: 10.5281/zenodo.4428158. URL https://github.com/nanoseb/pytst.

[67] Artur K. Lidtke, Thomas P. Lloyd, Frans Hendrik Lafeber, and Johan Bosschers. Predicting Cavitating Propeller Noise in Off-Design Conditions using Scale-Resolving CFD Simulations. *Ocean Engineering*, 254:111176, jun 2022. ISSN 00298018. doi: 10.1016/j.oceaneng.2022.111176.

[68] Stefano Lovato, Serge L. Toxopeus, Just W. Settels, Geert H. Keetels, and Guilherme Vaz. Code Verification of Non-Newtonian Fluid Solvers for Single- and Two-Phase Laminar Flows. *Journal of Verification, Validation and Uncertainty Quantification*, 6(2), jun 2021. ISSN 2377-2158. doi: 10.1115/1.4050131.

[69] A Lungu. A Sliding Grid Based Method for the Roll Decay Simulation. *IOP Conference Series: Materials Science and Engineering*, 591(1):012052, aug 2019. ISSN 1757-8981. doi: 10.1088/1757-899X/591/1/012052.

[70] Marin Lauber and Pandeli Temarel. Acquisition of Maneuvring Characteristics of Ships using RANS CFD. In *22nd Numerical Towing Tank Symposium (NuTTS)*, pages 1–6, Tomar, Portugal, 2019.

[71] J. Ezequiel Martin, Thad Michael, and Pablo M. Carrica. Submarine Maneuvers Using Direct Overset Simulation of Appendages and Propeller and Coupled CFD/Potential Flow Propeller Solver. *Journal of Ship Research*, 59(1):31–48, mar 2015. ISSN 00224502. doi: 10.5957/JOSR.59.1.140053.

[72] J. Ezequiel Martin, Ralph W. Noack, and Pablo M. Carrica. Overset Grid Assembly Approach for Scalable Computational Fluid Dynamics with Body Motions. *Journal of Computational Physics*, 390:297–305, aug 2019. ISSN 00219991. doi: 10.1016/j.jcp.2019.04.009.

[73] Florian R Menter. Eddy Viscosity Transport Equations and Their Relation to the k-ϵ Model. *Journal of Fluids Engineering*, 119(4):876–884, dec 1997. ISSN 0098-2202. doi: 10.1115/1.2819511.

[74] Florian R Menter, M Kuntz, and Robin Blair Langtry. Ten Years of Industrial Experience with the SST Turbulence Model. In K Hanjalic, Y Nagano, and M Tummers, editors, *Turbulence, Heat and Mass Transfer 4*, pages 625 – 632. Begell House, Inc., 2003.

[75] Florian R Menter, Yury Egorov, and D. Rusch. Steady and Unsteady Flow Modelling Using the k-skL Model. In *Proceedings of the International Symposium on Turbulence, Heat and Mass Transfer*, pages 403–406, New York, 2006. Begell House. ISBN 1-56700-229-3. doi: 10.1615/ICHMT.2006.TurbulHeatMassTransf.800.

[76] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller,

Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. SymPy: Symbolic Computing in Python. *PeerJ Computer Science*, 3(1):e103, jan 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103.

[77] Alireza Mofidi and Pablo M. Carrica. Simulations of Zigzag Maneuvers for a Container Ship with Direct Moving Rudder and Propeller. *Computers & Fluids*, 96:191–203, jun 2014. ISSN 00457930. doi: 10.1016/j.compfluid.2014.03.017.

[78] Anthony F. Molland and Stephen R. Turnock. Wind tunnel test results for a model ship propeller based on a modified Wageningen B4.40. Technical report, University of Southampton, Southampton, UK, 1990.

[79] Anthony F. Molland and Stephen R. Turnock. Wind Tunnel Tests on the Effect of a Ship Hull on Rudder-Propeller Performance at Different Angles of Drift. Technical report, University of Southampton, Southampton, UK, 1995.

[80] Anthony F. Molland and Stephen R. Turnock. A Propeller Thrust and Torque Dynamometer for Wind Tunnel Models. *Strain*, 38(1):3–10, 2002. ISSN 00392103. doi: 10.1046/j.0039-2103.2002.00001.x.

[81] Anthony F. Molland and Stephen R. Turnock. *Marine Rudders, Hydrofoils and Control Surfaces*. Elsevier, butterwort edition, 2022. ISBN 978-0-12-824378-7. doi: 10.1016/C2020-0-01238-7.

[82] Roberto Muscari, Riccardo Broglia, and A Di Mascio. An Overlapping Grids Approach for Moving Bodies Problems. *16th International Offshore and Offshore and Polar Engineering Conference Proceedings*, 4:243–248, 2006. ISSN 1098-6189.

[83] C. Nathan Woods and Ryan P. Starkey. Verification of Fluid-Dynamic Codes in the Presence of Shocks and Other Discontinuities. *Journal of Computational Physics*, 294:312–328, aug 2015. ISSN 00219991. doi: 10.1016/j.jcp.2015.03.055.

[84] Chris Nelson and Christopher Roy. Verification of the Wind-US CFD Code Using the Method of Manufactured Solutions. In *42nd AIAA Aerospace Sciences Meeting and Exhibit*, Reston, Virigina, jan 2004. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-078-9. doi: 10.2514/6.2004-1104.

[85] Van Tu Nguyen, Duc Thanh Vu, Warn Gyu Park, and Chul Min Jung. Navier–Stokes Solver for Water Entry Bodies with Moving Chimera Grid Method in 6DOF Motions. *Computers & Fluids*, 140:19–38, 2016. ISSN 00457930. doi: 10.1016/j.compfluid.2016.09.005.

[86] Ralph W. Noack. SUGGAR: a General Capability for Moving Body Overset Grid Assembly. In *17th AIAA Computational Fluid Dynamics Conference*, volume 5117,

pages 1–21, Toronto, Ontario, Canada, 2005. ISBN 9781624100536. doi: 10.2514/6.2005-5117.

[87] Ralph W. Noack. DiRTlib: A Library to Add an Overset Capability to your Flow Solver. In *17th AIAA Computational Fluid Dynamics Conference*, number June 2005, pages 1–20, Toronto, Ontario, Canada, jun 2005. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-053-6. doi: 10.2514/6.2005-5116.

[88] Ralph W. Noack and David A Boger. Improvements to SUGGAR and DiRTlib for Overset Store Separation Simulations. In *47th AIAA Aerospace Sciences Meeting*, number January, pages 5–9, Orlando, Florida, 2009. ISBN 978-1-60086-973-0. doi: 10.2514/6.2009-340.

[89] Ralph W. Noack, Nicholas J. Wyman, Greg McGowan, and Cameron Brown. Dual-Grid Interpolation for Cell-Centered Overset Grid Systems. In *AIAA Scitech 2020 Forum*, volume 1 PartF, pages 1–32, Reston, Virginia, jan 2020. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-595-1. doi: 10.2514/6.2020-1407.

[90] G. Oger, A. Vergnaud, B. Bouscasse, J. Ohana, M. Abu Zarim, M. De Leffe, A. Bannier, L. Chiron, Y. Jus, M. Garnier, S. Halbout, and D. Le Touzé. Simulations of Helicopter Ditching using Smoothed Particle Hydrodynamics. *Journal of Hydrodynamics*, 32(4):653–663, aug 2020. ISSN 1001-6058. doi: 10.1007/s42241-020-0044-y.

[91] Kunihide Ohashi. A New Approach for Handling Body Motion by Combining a Grid Deformation Method and an Overset Grids Technique. *Ocean Engineering*, 213(August):107836, oct 2020. ISSN 00298018. doi: 10.1016/j.oceaneng.2020.107836.

[92] Kunihide Ohashi, Takanori Hino, Hiroshi Kobayashi, Naoyuki Onodera, and Nobuaki Sakamoto. Development of a Structured Overset Navier–Stokes Solver with a Moving Grid and Full Multigrid Method. *Journal of Marine Science and Technology*, 24(3):884–901, sep 2019. ISSN 0948-4280. doi: 10.1007/s00773-018-0594-7.

[93] Enrique Orduna-Malea, Juan M. Ayllón, Alberto Martín-Martín, and Emilio Delgado López-Cózar. Methods for Estimating the Size of Google Scholar. *Scientometrics*, 104(3):931–949, sep 2015. ISSN 0138-9130. doi: 10.1007/s11192-015-1614-6.

[94] Johannes Palm and Claes Eskilsson. Facilitating Large-Amplitude Motions of Wave Energy Converters in OpenFOAM by a Modified Mesh Morphing Approach. *Proceedings of the European Wave and Tidal Energy Conference*, pages 2107–1–2107–8, 2021. ISSN 27066940.

[95] PDC. Gridpro. URL https://www.gridpro.com.

[96] Milovan Peric and Volker Bertram. Trends in Industry Applications of Computational Fluid Dynamics for Maritime Flows. *Journal of Ship Production and Design*, 27(04):194–201, nov 2011. ISSN 2158-2866. doi: 10.5957/jspd.2011.27.4.194.

[97] Alexander B. Phillips, Stephen R. Turnock, and Maaten Furlong. Accurate Capture of Propeller-Rudder Interaction using a Coupled Blade Element Momentum-RANS Approach. *Ship Technology Research*, 57(2):128–139, apr 2010. ISSN 0937-7255. doi: 10.1179/str.2010.57.2.005.

[98] C. Pilloton, A. Bardazzi, A. Colagrossi, and S. Marrone. SPH Method for Long-Time Simulations of Sloshing Flows in LNG Tanks. *European Journal of Mechanics - B/Fluids*, 93:65–92, may 2022. ISSN 09977546. doi: 10.1016/j.euromechflu.2022.01.002.

[99] Patrick Queutey, Garbo Deng, Jeroen Wackers, Emmanuel Guilmineau, Alban Leroyer, and Michel Visonneau. Sliding Grids and Adaptive Grid Refinement for RANS Simulation of Ship-Propeller Interaction. *Ship Technology Research*, 59(2):44–57, apr 2012. ISSN 0937-7255. doi: 10.1179/str.2012.59.2.004.

[100] Eliot W. Quon and Marilyn J. Smith. Advanced Interpolation Techniques for Overset CFD. In *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, number January, Reston, Virigina, jan 2012. American Institute of Aeronautics and Astronautics. ISBN 978-1-60086-936-5. doi: 10.2514/6.2012-305.

[101] Eliot W. Quon and Marilyn J. Smith. Advanced Data Transfer Strategies for Overset Computational Methods. *Computers & Fluids*, 117:88–102, aug 2015. ISSN 00457930. doi: 10.1016/j.compfluid.2015.04.023.

[102] E.J. Ransley, D. Greaves, A. Raby, D. Simmonds, and M. Hann. Survivability of Wave Energy Converters using CFD. *Renewable Energy*, 109:235–247, aug 2017. ISSN 09601481. doi: 10.1016/j.renene.2017.03.003.

[103] Patrick J. Roache. Code Verification by the Method of Manufactured Solutions. *Journal of Fluids Engineering*, 124(1):4–10, mar 2002. ISSN 0098-2202. doi: 10.1115/1.1436090.

[104] Patrick J. Roache. *Fundamentals of Verification and Validation*. Hermosa Publishers, 2009. ISBN 978-0913478127.

[105] Patrick J. Roache. The Method of Manufactured Solutions for Code Verification. In Claus Beisbart and Nicole J. Saam, editors, *Computer Simulation Validation*, Simulation Foundations, Methods and Applications, pages 295–318. Springer International Publishing, Cham, 2019. ISBN 978-3-319-70765-5. doi: 10.1007/978-3-319-70766-2_12.

[106] Robert McNeel & Associates. Rhinoceros. URL https://www.rhino3d.com.

[107] Beatrice Roget and Jayanarayanan Sitaraman. Robust and Efficient Overset Grid Assembly for Partitioned Unstructured Meshes. *Journal of Computational Physics*, 260:1–24, mar 2014. ISSN 00219991. doi: 10.1016/j.jcp.2013.12.021.

[108] Christopher Roy, Curtis Ober, and Tom Smith. Verification of a Compressible CFD Code Using the Method of Manufactured Solutions. In *32nd AIAA Fluid Dynamics Conference and Exhibit*, Reston, Virigina, jun 2002. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-113-7. doi: 10.2514/6.2002-3110.

[109] Youcef Saad and Martin H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, jul 1986. ISSN 0196-5204. doi: 10.1137/0907058.

[110] Kirk Schloegel, George Karypis, and Vipin Kumar. Parallel Static and Dynamic Multi-Constraint Graph Partitioning. *Concurrency and Computation: Practice and Experience*, 14(3):219–240, mar 2002. ISSN 1532-0626. doi: 10.1002/cpe.605.

[111] Eberhard Schreck and Milovan Peric. Overset Grids in STAR-CCM+: Methodology, Applications and Future Developments. In *STAR Japanese Conference*, 2012.

[112] J.R. Shao, H.Q. Li, G.R. Liu, and M.B. Liu. An Improved SPH Method for Modeling Liquid Sloshing Dynamics. *Computers & Structures*, 100-101:18–26, jun 2012. ISSN 00457949. doi: 10.1016/j.compstruc.2012.02.005.

[113] Ashesh Sharma, Shreyas Ananthan, Jayanarayanan Sitaraman, Stephen Thomas, and Michael A. Sprague. Overset Meshes for Incompressible Flows: On Preserving Accuracy of Underlying Discretizations. *Journal of Computational Physics*, 428: 109987, mar 2021. ISSN 00219991. doi: 10.1016/j.jcp.2020.109987.

[114] Zhirong Shen, Decheng Wan, and Pablo M. Carrica. Dynamic overset grids in OpenFOAM with application to KCS self-propulsion and maneuvering. *Ocean Engineering*, 108:287–306, nov 2015. ISSN 00298018. doi: 10.1016/j.oceaneng.2015.07.035.

[115] Zhirong Shen, Decheng Wan, and Pablo M. Carrica. Dynamic Overset Grids in OpenFOAM with Application to KCS Self-Propulsion and Maneuvering. *Ocean Engineering*, 108:287–306, nov 2015. ISSN 00298018. doi: 10.1016/j.oceaneng.2015.07.035.

[116] Jayanarayanan Sitaraman, Matthew Floros, Andrew M. Wissink, and Mark A. Potsdam. Parallel Domain Connectivity Algorithm for Unsteady Flow Computations using Overlapping and Adaptive Grids. *Journal of Computational Physics*, 229(12):4703–4723, 2010. ISSN 0021-9991. doi: 10.1016/j.jcp.2010.03.008.

[117] Philippe R Spalart and S. Allmaras. A One-Equation Turbulence Model for Aerodynamic Flows. In *30th Aerospace Sciences Meeting and Exhibit*, Reston, Virigina, jan 1992. American Institute of Aeronautics and Astronautics. doi: 10.2514/6.1992-439.

[118] Serge L. Toxopeus and K Bhawsinka. Calculation of Hydrodynamic Interaction Forces on a Ship Entering a Lock Using CFD. In *4th International Conference on Ship Manoeuvring in Shallow and Confined Water with Special Focus on Ship Bottom Interaction*, pages 305–314, Hamburg, Germany, 2016. doi: 10.18451/978-3-939230-38-0_34.

[119] Stephen R. Turnock. Computer Aided Design and Numerically Controlled Manufacture of a Split Mold for a Composite Model Ship Propeller. Technical report, University of Southampton, 1990.

[120] Stephen R. Turnock. A Test Rig for the Investigation of Ship Propeller/Rudder Interactions. Technical report, University of Southampton, Southampton, UK, 1990.

[121] WPA van Lammeren, J D van Manen, and MWC Oosterveld. The Wageningen B-screw Series. *Schip en Werf*, 5:88–103, 1970.

[122] Guilherme Vaz, Frederick Jaouen, and Martin Hoekstra. Free-Surface Viscous Flow Computations: Validation of URANS Code FreSCo. In *Volume 5: Polar and Arctic Sciences and Technology; CFD and VIV*, pages 425–437. ASMEDC, jan 2009. ISBN 978-0-7918-4345-1. doi: 10.1115/OMAE2009-79398.

[123] Subrahmanya Veluri, Christopher Roy, Shelley Hebert, and Edward Luke. Verification of the Loci-CHEM CFD Code Using the Method of Manufactured Solutions. In *46th AIAA Aerospace Sciences Meeting and Exhibit*, Reston, Virigina, jan 2008. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-128-1. doi: 10.2514/6.2008-661.

[124] Suyash Verma and Arman Hemmati. Performance of Overset Mesh in Modeling the Wake of Sharp-Edge Bodies. *Computation*, 8(3):66, jul 2020. ISSN 2079-3197. doi: 10.3390/computation8030066.

[125] Diego Villa, Andrea Franceschi, and Michele Viviani. Numerical Analysis of the Rudder–Propeller Interaction. *Journal of Marine Science and Engineering*, 8(12): 990, dec 2020. ISSN 2077-1312. doi: 10.3390/jmse8120990.

[126] S. Völkner, Jörg Brunswig, and Thomas Rung. Analysis of Non-Conservative Interpolation Techniques in Overset Grid Finite-Volume Methods. *Computers & Fluids*, 148:39–55, apr 2017. ISSN 00457930. doi: 10.1016/j.compfluid.2017.02.010.

[127] Vuko Vukcevic and Hrvoje Jasak. Overset Mesh Library in Foam-Extend, 2018.

[128] Yu Wang, Hamn-Ching Chen, Arjen Koop, and Guilherme Vaz. Hydrodynamic Response of a FOWT Semi-Submersible Under Regular Waves using CFD: Verification and Validation. *Ocean Engineering*, 258:111742, aug 2022. ISSN 00298018. doi: 10.1016/j.oceaneng.2022.111742.

[129] Christian Windt, Josh Davidson, Benazzou Akram, and John V. Ringwood. Performance Assessment of the Overset Grid Method for Numerical Wave Tank Experiments in the OpenFOAM Environment. In *ASME 37th International Conference on Ocean, Offshore and Arctic Engineering*, volume 10, pages 1–10, Madrid, Spain, 2018. ISBN 9780791851319. doi: 10.1115/OMAE2018-77564.

[130] Christian Windt, Josh Davidson, Dominic D.J. Chandar, and John V. Ringwood. On the Importance of Advanced Mesh Motion Methods for WEC Experiments in CFD-based Numerical Wave Tanks. In R. Bensow Ringsberg and J., editors, *VIII International Conference on Computational Methods in Marine Engineering MARINE 2019*, pages 145–156, Gothenburg, Sweden, 2019. International Center for Numerical Methods in Engineering (CIMNE).

[131] Andrew M. Wissink and Robert L. Meakin. On Parallel Implementations of Dynamic Overset Grid Methods. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM) - Supercomputing '97*, volume 1, pages 1–21, New York, New York, USA, 1997. ACM Press. ISBN 0897919858. doi: 10.1145/509593.509608.

[132] Naz Yilmaz, Batuhan Aktas, Mehmet Atlar, Patrick A. Fitzsimmons, and Mario Felli. An Experimental and Numerical Investigation of Propeller-Tudder-Hull Interaction in the Presence of Tip Vortex Cavitation (TVC). *Ocean Engineering*, 216 (May):108024, nov 2020. ISSN 00298018. doi: 10.1016/j.oceaneng.2020.108024.

[133] Shen Zhirong, Wan Decheng, and Pablo M. Carrica. RANS Simulations of Free Maneuvers with Moving Rudders and Propellers Using Overset Grids in OpenFOAM. In *SIMMAN workshop on Verification and Validation of Ship Maneuvering Simulation Methods*, number December, Lyngby, Denmark, 2014.