VPMFoam

A 2D Incompressible Hybrid Eulerian-Lagrangian Solver for External Aerodynamics

Pasolari, R.

DOI
10.4233/uuid:f9713f44-9b0b-40e0-9793-3fa10e6a9ba6

Publication date
2025

Document Version
Final published version

Citation (APA)
Pasolari, R. (2025). *VPMFoam: A 2D Incompressible Hybrid Eulerian-Lagrangian Solver for External Aerodynamics*. [Dissertation (TU Delft), Delft University of Technology].
https://doi.org/10.4233/uuid:f9713f44-9b0b-40e0-9793-3fa10e6a9ba6

VPMFoam

A 2D INCOMPRESSIBLE HYBRID EULERIAN-LAGRANGIAN SOLVER FOR EXTERNAL AERODYNAMICS

RENTION PASOLARI

VPMFoam

A 2D Incompressible Hybrid Eulerian-Lagrangian Solver for External Aerodynamics

VPMFoam

A 2D Incompressible Hybrid Eulerian-Lagrangian Solver for External Aerodynamics

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus, Prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board of Doctorates
to be defended publicly on Thursday 3, July 2025 at 15:00 o'clock

by

Rention PASOLARI

Master of Science in Mechanical Engineering and Aeronautics,
University of Patras, Greece,
born in Këlçyrë, Albania.

This dissertation has been approved by the candidate's promotors

Promotor: Prof. dr. ir. C.J. Simão Ferreira
Promotor: Dr. ir. M.I. Gerritsma
Copromotor: Dr. ir. A.H. Van Zuijlen

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. ir. C.J. Simão Ferreira,	Delft University of Technology
Dr. ir. M.I. Gerritsma,	Delft University of Technology
Dr. ir. A.H. Van Zuijlen,	Delft University of Technology
Independent members:	
Dr. A. Goude,	Uppsala University, Sweden
Prof. dr .ir. S. Hickel,	Delft University of Technology
Dr. ir. R.J. Labeur,	Delft University of Technology
Dr. G. Papadakis,	National Technical University of Athens, Greece
Prof. dr. D.A. von Terzi	Delft University of Technology, reserve member

Keywords: Eulerian Methods, Lagrangian Methods, Coupling, Hybrid Method, OpenFOAM, Vortex Particle Methods, External Aerodynamics

Printed by: Ipskamp

Front & Back: The cover illustrates an abstract fusion of code and flow. Designed by Kiriakos Kalogerinis - Portfolio.

ISBN 978-94-6518-072-4

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

To my parents,
and my brother.

Στους γονείς μου,
και στον αδερφό μου.

Acknowledgements

To whoever holds these pages in hand, hello! If you're reading this part of my dissertation, it means we've reached the end. Wow! I still remember the day of my first interview for this Ph.D. position like it was just yesterday. One of my earliest conversations with Carlos sticks with me, *"Rention, these years will fly by, and before you know it, you'll be at the end, preparing your dissertation."* And he was absolutely right. Here I am, completing my dissertation, and I still can't believe how fast the time has passed.

I started this journey in October 2020, which, let's be honest, wasn't exactly the best time to start a Ph.D., especially abroad. Covid-19 was everywhere. My first experiences in the Netherlands were lockdowns, loneliness and anxiety (oh, and a scam when I thought I rented a house but just ended up losing money. Be careful, don't do what I did!). Covid stuck around for a while, and it became a big part of my Ph.D. experience. It definitely marked this chapter of my life, but looking back now, it makes me even prouder that I made it through. In the end, some challenges really do make you stronger.

This four-year journey has been a true rollercoaster, full of highs and lows, moments of self-doubt, and unexpected strength. I questioned my abilities, my place in academia, and even my passion for research. At times, I wasn't sure I deserved to be here at all. It took conversations with friends, colleagues, and supervisors to realize that many go through this, often facing the weight of "Impostor Syndrome" alone. There were days I thought seriously about quitting, calling my parents and my brother for reassurance. But looking back now, I'm proud I didn't give up.

And it wasn't all struggle, far from it. Alongside the difficult moments, there were many times of joy, growth, and discovery. I still remember the thrill of seeing perfect results, the excitement of having my first paper accepted, and the engaging discussions with supervisors and colleagues who truly understood and challenged my thinking. There was a deep sense of satisfaction in finally solving a problem that had kept me up at night, and a quiet joy in those unexpected "aha!" moments.

I didn't make it here alone. This dissertation wouldn't exist without the people who stood by me, those who supported me openly or quietly, sacrificing their time and energy. The next paragraphs are for them.

First, I'd like to thank **TU Delft** and everyone who works here to make our Ph.D. journey smoother. A big shoutout to the **secretaries**, the **Graduate School staff**, **course tutors**, **cleaning staff**, **IT**, **professors**, **technicians**, **administrative staff**, and everyone behind the scenes who keeps things running.

I would like to thank and acknowledge all involved in developing, maintaining, expanding, and administering the **DelftBlue** supercomputer. Over the years, I used DelftBlue for

most of my simulations, and I couldn't have achieved these results without it. The team behind it does an incredible job, from offering courses to answering questions whenever needed. Your work has made a huge difference. Thank you!

I owe a great debt of gratitude to my supervisors, **Prof. dr. ir. C.J. Simão Ferreira** and **Dr. ir. Alexander van Zuijlen**. First of all, thank you both for trusting me and choosing me to work with you over these years. I truly enjoyed our collaboration.

Carlos, you helped me become more independent, always trusting me and coming up with brilliant ideas to test new features or track down bugs in the code. Our brainstorming sessions were a highlight. But most of all, thank you for your attitude and the way you treated me. You taught me to believe in myself and not to undervalue my work or results. You always reminded me that being happy, smiling, and taking care of my health are the top priorities, more important than anything else we do. That caring attitude really stayed with me, and I truly appreciate all your kind words. They meant a lot to me and will stay with me as I move forward.

Sander, your expertise in CFD was invaluable. You were always there to help me understand tricky concepts, and even for quick meetings whenever I needed them. You sat down with me, going through my code line by line to find bugs that I couldn't see myself. Thanks for trusting me and getting me involved in the OpenFOAM seminars and supervising an Honours Programme, that really helped me grow. I am also very grateful for your help in translating my summary and propositions into Dutch. I deeply appreciate all the time and help you generously gave me throughout this journey.

You both set the example of what excellent supervision looks like: knowledgeable, supportive, and truly invested in your students. I truly appreciate that. Thank you!

I also want to thank my promotor, **Dr. ir. Marc Gerritsma**. You may have joined my Ph.D. journey towards the end, but your support was crucial in helping me reach the finish line. Thank you for being part of my Go/No-Go meeting and for your invaluable feedback. Thank you as well for handling everything during the final stretch to make the defense possible. Your help was truly appreciated.

A special thanks to the members of my defense committee for agreeing to be part of this important moment: **Dr. A. Goude**, **Prof. dr. ir. S. Hickel**, **Dr. ir. R.J. Labeur**, and **Dr. G. Papadakis**. I truly appreciate the time and energy you've devoted to reviewing my work and participating in my defense. I would also like to thank **Prof. dr. D.A. von Terzi** for serving as a reserve member and for all the invaluable help you provided in finalizing the procedural aspects of my Ph.D. when I needed it most.

Many thanks to the people I collaborated with on organizing the Ph.D. OpenFOAM seminars. Whether we worked together for a long time or just briefly, **Dennis**, **Mehtab**, **Shubham**, **Flavio**, **Bram**, I really enjoyed it, guys. It was a great experience!

I would like to express my appreciation to the two Bachelor groups I co-coached during the DSE project. Our communication and collaboration really helped me grow. A big thank you as well to the supervisors of these projects, **Dr. ir. Julien van Campen** and **Dr. ir. Hans Kuiper**, as well as my fellow co-coachs, **Jin Maruhashi** and **Victor Poorte**.

A special thank you goes to **Jakub Siemaszko**, the Honours Bachelor student I had the great chance to co-supervise during his project. You did brilliant work and helped me grow in so many ways. Good luck with your Master's!

I am deeply thankful to the people whose work I built on. First of all, **Carlos Baptista**, you helped me get started in this research area with some really great brainstorming sessions. Even after our meetings, you stayed interested in my progress. The code you developed in OpenFOAM is brilliant, and it's been a huge part of my work, so thank you so much for that. You also have an amazing ability to explain difficult concepts in such a relaxed, easy-going way that makes everything seem simple. I also want to thank **Dr. Artur Palha**, who developed the majority of the code I based on. We only met once during Covid-19 era via video call, but you still helped me with a part of the code you worked on years ago, and you were willing to assist even though you weren't involved in the project anymore. I also learned a lot from your coding style, even though we didn't work together directly. Your code taught me what well-structured, properly documented code should look like. Thank you for that!

I would also like to thank **Dr. Spyros Aleiferis**, who has supported me since my undergraduate years. You consistently helped me navigate the next steps in my academic journey, offering invaluable guidance during my Master's and practical advice drawn from your own experience. Your support and availability whenever I had questions have been truly appreciated. I'm also grateful to **Dr. Polycarpos Papadopoulos**, my supervisor during my undergraduate and postgraduate studies before the Ph.D. You introduced me to CFD, taught me how to work with OpenFOAM, and gave me solid foundations that I carried into my Ph.D. and still rely on. You also helped me clear my mind when I was searching for a Ph.D., encouraged me to look for opportunities abroad, and supported me throughout that process. Thank you for all your help. I hope we'll collaborate again in the future.

An especially heartfelt thank you to my therapist. Going through psychotherapy has been the best gift I've given to myself. You helped me find my way back when I was close to falling apart, and for that, I'm deeply grateful. To everyone reading this, I encourage you to give this gift to yourself. Trust me, it's worth it!

I want to thank all of my colleagues from these past years. With some of you, I had the chance to grow closer and share deeper conversations. With others, our exchanges were more brief. But in every case, whether through small talk, group meetings, or our sessions in Carlos' team, I believe you always gain something from the people around you, consciously or unconsciously. I feel very lucky to have met such incredible individuals. I've learned so much, not only academically but also about different cultures, perspectives, and ways of thinking. I truly hope that each of us finds his own path after this big chapter called the Ph.D. You are always welcome at my place in Greece. I also hope that, in the future, we'll get the chance to collaborate. I truly trust and respect your work. A warm thank you to: **Mehtab**, **Kiran**, **Jingna**, **David**, **Adhyanth**, **Guanqun**, **Rishikesh**, **Andrea**, **Flavio**, **Erik**, **Cristina**, **Abhratej**, **Shyam**, **Deepali**, **Ricardo**, **Ali**, **Yanan**, **Sebastiano**, **Jatinder**, **Livia**, **Anand**, **Likhitha**, **Simone**, **Mihir**, **Abhyuday**, **Matteo**, **Kaj**, **Clem**, **Jiaxin**, **Jelle**, **Shantanu**, **Helena**, **Haoyuan** and **André**.

Friends are the family you choose, so a heartfelt thank you goes to my friends. It's incredible to be surrounded by people you love, who make you feel at home and part of a family. We all know that part of who we are is made up of little things we've taken from our friends, and for me, I'm sure that all the conversations we've had and all the time we've spent together have shaped who I am today. Of course, I could write pages of memories and individual notes to each of you, but then the acknowledgements would end up longer than the actual thesis.

I want to start with my friends from back home. It's rare to have such a big group of people from your school years still by your side today, even if we're far apart most of the time. **Alexandros ("Alekos")**, you've always been there for me, every single time I needed it. I truly appreciate it! **Antonis (Marousis)**, I'll never forget the memories we made in Patras. Thank you, and best of luck with your Ph.D., you're almost there! **Antonis ("Toni")**, my 12-year school desk partner. Keep being real! I'm really grateful we've been friends since our childhood. Thanks for everything! **Argyris ("Argy")**, the guy I truly believe can achieve anything he sets his mind to. I'm really happy and proud to have met you! **Achilleas**, you supported me a lot, especially during tough times, like when I moved to the Netherlands. I really appreciate that! **Dimitris ("Ypogeios")**, the guy for every job. A part of this Ph.D. happened thanks to you. I'd still be struggling with Ubuntu without your help. Thanks, mate! **Dimitris ("Gozive")**, the doctor of the group and the warmest hug you can get. You reached the finish line man, well done! **Eleni (Livanou)**, I really look back all the times we shared during our summers. Thanks for all your support and for being there, especially when I started this journey, I truly appreciate it. **Giorgos (Michalettos)**, probably the person who understood me the most during the Ph.D., being on the same path himself. Thanks for all the support and conversations. You're up next! **Giorgos (Pissanos)**, we've shared amazing summer moments, unforgettable memories. Thanks, mate! **Harris (Veliotis)**, the most talented person I've met. I genuinely admire you and fully believe in you! Thanks, man! **Ioanna (Rothou)** I truly thank you for all your support and for being there with me in the Netherlands during a tough time! **Kiriakos (Kalogerinis)**, first of all, thank you for the amazing thesis cover. I may have met you only a few years ago, but the bond we've built is incredible. Thanks for all the time we've spent on our never-ending (literally) coffee sessions. **Konstantinos ("Velo")**, the smartest guy with the greatest sense of humor. I'm really greatful for all the time we've spent together. **Labros (Kourkoulis)**, for sure the most knowledgeable guy around. Even though we met recently, your support has meant a lot to me. Thanks! **Manolis (Georgakakos)**, thanks for all the support, it's always a joy being around you. **Panagiotis ("Psilos")**, the sweetest guy you'll ever meet and a true example of a good friend. Thanks man! **Panagiotis (Kalivitis)**, we've spent countless hours talking. We share a similar mindset, and that connection has really helped shape who I am. Thanks! **Panagiotis (Kompis)**, unforgettable memories with you, especially during our holidays. **Theodora (Konstantinakou)**, thank you for all the support and the time we've shared together. I truly admire the way you fight for your dream.

Knowing he gets a little touchy sometimes, I have to give a special shoutout to my "roommate" **Panagiotis (Georgakakos)**. Thank you for all the time we've spent together, both in Greece and the Netherlands. You've been a big part of this journey, and I'm grateful

for all the memories we've shared. Thank you for all the endless conversations and the love and support I've received from you. Publicly, I have to say, you somehow have the unique talent of being both the best and the worst roommate at the same time, and honestly, I wouldn't trade it for anything! Thanks for everything! And of course, along with you, I want to thank **Effie (Kolonia)**. Thank you for being by my friend's side and for all the help and support you've given me!

I'm truly grateful to have met all of you. Those memories are like a safe place I can always come back to, and I'll cherish them forever. Thanks for everything! And you know I'm always here for you, no matter what.

Continuing, I want to thank all the amazing friends I met during my university years, whom I now consider family. Each of you has played a crucial role in my journey, and I hope our paths will cross again someday. These years have truly been the best of my life. **Ilias (Latee)**, I'm sure you'll achieve the dream you've set very soon. **Giorgos (Frantzis)**, a human Wikipedia. I really admire your mindset and the person you are. **Christos ("Pappous")**, one of the truest guys I've ever met; all the best with your Ph.D., I know you can achieve anything. **Konstantinos (Bartzis)**, I've got tons of great moments with you, brother! Always stay true! **Nikos ("Doppler")**, thanks for all the support during our times in Patras and the Netherlands, it meant a lot. **Marios (Kotsampasakis)**, I can remember thousands of moments together, and I really cherish them. Thanks for everything, man. **Stratos (Polyzos)** I have no doubt you'll go far, mate. **Alex (Tsokanos)**, thanks for all the moments we spent in Patras, I really miss those days. **Haroula (Sfetsa)**, you really supported me throughout our university years, and I'm so happy we've reconnected now. **Rigers**, the sweetest guy, I feel lucky to have met you and being friends. **Dimitris (Dimitriou)**, though we got to know each other better toward the end of university, I really admire your mindset and dedication. And of course, the guys from Zevgolatio, even though we weren't in the same city, I have many great memories with you in Patras. **Andreas ("Gennaios")**, you hold a special place in my heart, as I've told you before. **Andreas (Kokkas)** and **Michalis (Menounos)**, thanks for all the times we shared together, guys!

Of course, I want to thank all the incredible friends I made in the Netherlands, I'm lucky to have shared so many great times with you all. Thank you, **Apostolos**, **Alex**, **Charis**, **Chrysanthos**, **Gerasimos (Ntoukas)**, **Nikos (Bias)**, **Harry**, **Ilias**, **Manuel**, **Nikol** and **Penelope**. A special shoutout goes to **Stratis**. Thanks for all the help you gave me and for the countless supportive conversations. You're truly one of a kind, bro! And of course, there's **Elina**. Thank you for all the support and for introducing me to the amazing people above.

And of course, life is a continuous journey, and along the way, we meet new people who we choose to keep close. I want to thank **Ilias (Lykouridis)** for all the great moments and conversations we've shared, I truly admire you, man! I also want to thank **Dimitris (Stathis)** for all the good times we've spent together, and I really hope there are many more to come!

I want to thank everyone who has been part of my journey. Whether we shared big moments or small, each of you has had an impact on my life. I really appreciate all the support and memories along the way.

A very special thank you goes to **Asimina**. You've been by my side through the toughest part of my Ph.D. journey, especially during these last two years. Thank you for all the support you've given me, through my highs and lows. There were times when I was distant or didn't handle things as well as I could have, but you still understood me and helped me find my way. I'm deeply grateful for your patience, our long talks, and for being there during my toughest moments. Thank you for listening, even when I was at my most vulnerable. Along with you, I want to thank your family: your parents for their kindness, and your sisters and brother. They are all truly wonderful.

Of course, the last paragraphs are dedicated to my family. This is the least I can do to thank you. **Brother**, you have supported me in every step of my life, making me feel secure that you can be there whenever I need you. You are a fighter and I really admire you, love ya! And of course, along with you, I want to thank **Sabina** for being on your side, and for giving birth to my little niece **Chloe**! Thank you!

Μάνα και **Πατέρα**. Σας ευχαριστώ για όλα. Έχετε θυσιάσει όλη σας τη ζωή για τον αδερφό μου και εμένα. Πήρατε το τεράστιο ρίσκο να αφήσετε την πατρίδα σας και να μεταναστεύσετε σε μια ξένη χώρα, κάτω από τις πιο δύσκολες συνθήκες, χωρίς να ξέρετε ούτε μία λέξη στα Ελληνικά, με δύο πολύ μικρά παιδιά, για να έχουμε εμείς ένα καλύτερο μέλλον. Σε όλη σας τη ζωή έχετε παλέψει, δουλεύοντας κάθε μέρα 10, 12, 14, ακόμη και 16 ώρες, για να μας προσφέρετε τα πάντα. Και ξέρετε τι; Τα καταφέρατε! Ό,τι κι αν έχω καταφέρει εγώ, δεν το θεωρώ τίποτα μπροστά σε όσα καταφέρατε εσείς. Εσείς είστε οι πραγματικοί μαχητές. Ό,τι είμαι σήμερα, το χρωστάω σε εσάς. Σας ευχαριστώ για όλα και σας αγαπώ με όλη μου την καρδιά! Αυτό το βιβλίο είναι αφιερωμένο σε εσάς και στον αδερφό μου!

I want to close this section not with an acknowledgment but with a personal reflection. Over the years, I have grown not only as a researcher but as a person, because in the end, our profession is just one part of who we are. What I've truly learned is that what we have, above all, is one another. I often think of a lyric from a song by the Greek band Βόρεια Αστέρια (Voreia Asteria), which says: "*ό,τι έχουμε είμαστε εμείς*" (*all we have is us*). No one is truly alone, and no one should ever feel that way. We are here to support, uplift, and stand beside one another. Solidarity and compassion are not ideals, they are necessities. Life can pull us away from that path, but we must hold onto it. Every choice for kindness, no matter how small, makes a difference. No one has ever been harmed by love or peace.

Help, because tomorrow, you will be the one in need. I end with this thought because in difficult times, we are reminded of what truly matters. I wish peace, health, and freedom to each of us, and to all of humanity.

Rention Pasolari
June 2025

Contents

Summary

The rapid advancement of science and technology is driven by the goal of improving life on Earth while maintaining environmental responsibility. This progress is evident across numerous fields. In wind energy, the continuous development of new wind turbine designs and innovative wind farm configurations is crucial for accelerating the energy transition. The aerospace industry constantly introduces advanced airplane and helicopter designs to improve performance and efficiency. Urban planning increasingly relies on detailed analysis to optimize airflow within cities, addressing environmental and health concerns. These are just a few examples where innovation is driven by the need to enhance performance and sustainability.

A common requirement across these diverse fields is the need for an in-depth understanding of aerodynamics. Accurate aerodynamic analysis is essential for enhancing design, efficiency, and effectiveness. However, the fast-paced nature of modern advancements limits reliance on experimental methods alone, as these can be time-consuming, costly, and impractical for all possible configurations. Consequently, combining experimental and computational studies becomes crucial.

This is where Computational Fluid Dynamics (CFD) comes into play. The development of efficient and accurate CFD tools has become essential in scientists' and engineers' hands to explore aerodynamics quickly and understand the physics of the flows. This fact has driven the present research. The primary goal of this dissertation is the development of a computational tool that is both accurate and efficient for exploring external aerodynamics simulations.

The main approaches in CFD today are the Eulerian and Lagrangian approaches, each comprising a family of methods. Eulerian methods, like the Finite Volume Method (FVM) and Finite Element Method (FEM), have been extensively used in exploring external aerodynamics, with their greatest advantage being accuracy in capturing the boundary layers. However, due to the diffusive nature of these methods, artificial diffusion is introduced into the flow, damping the vortex structures, which are crucial in many applications driven by strong body-vortex interactions. Moreover, the study of multibody objects often requires special treatments, especially for mesh generation, making the simulations extremely costly.

On the other hand, Lagrangian methods, like the Vortex Particle Method (VPM), are excellent for studying flows with a high presence of vortices, as they can preserve the vortex structures without damping them. Additionally, the particles participating in the flow are self-adaptive, satisfy the far-field boundary conditions automatically, and allow for easy implementation of multiple bodies into the simulation. However, resolving the boundary layer is very challenging and often very costly due to the inability to use anisotropic elements, making them less ideal for predicting aerodynamic forces.

Lagrangian solvers have become very popular in the last two to three decades, primarily due to the significant advancements in computer hardware, especially GPUs,

which enable very fast calculations. This has encouraged engineers to explore ways to leverage this advantage in CFD, leading to the development of hybrid solvers that couple Eulerian and Lagrangian solvers. In this coupled approach, Eulerian solvers can be applied near the solid body to accurately and efficiently resolve the boundary layer region, while Lagrangian solvers preserve the vortex structures further from the body.

Building on this approach, this dissertation introduces a hybrid Eulerian-Lagrangian solver, named *VPMFoam*, developed to combine the strengths of both methods while minimizing their limitations. This is achieved by integrating OpenFOAM, a widely used open-source CFD software, with a Lagrangian VPM. The primary goal is to create an accurate tool focused on external aerodynamics that can efficiently handle cases with strong body-vortex interactions and multibody scenarios while maintaining a lower computational cost compared to pure Eulerian solvers. OpenFOAM was chosen primarily for its open-source flexibility and its extensive user base in academia and industry, offering broad access to this tool.

The solver's development focuses on a 2D version, with validation conducted through a step-by-step approach, starting with simple cases that exclude solid bodies. Once the successful coupling of the two solvers is verified, validation proceeds with cases involving solid bodies, such as flow around a cylinder. In these cases, the solver accurately predicts fluid flow and aerodynamic coefficients, showing strong agreement with established Eulerian solvers and effectively preserving the vorticity field in the wake. Further validations address dynamic mesh motions and multibody applications.

Following validation, the solver is applied to more realistic scenarios, including the static and dynamic stall of an airfoil and the simulation of hybrid Vertical Axis Wind Turbines using actuator models. A performance analysis of the codes efficiency is also presented, highlighting the solvers capability to conduct fast and accurate simulations.

By effectively combining the strengths of both Eulerian and Lagrangian approaches, *VPMFoam* addresses key challenges in aerodynamic analysis, particularly in cases with strong body-vortex interactions, and lays a strong foundation for further applications, including potential 3D extensions. This makes *VPMFoam* a valuable asset for applications requiring both high fidelity and computational efficiency.

Samenvatting

De snelle wetenschappelijke en technologische vooruitgang wordt gedreven door het doel om het bestaan op aarde te verbeteren met inachtneming van verantwoordelijkheid jegens het milieu. Deze vooruitgang is evident over meerdere gebieden. Voor een versnelling in de energie transitie, is het in windenergie cruciaal om continue nieuwe windturbines te ontwerpen en innovatieve windpark inrichtingen te ontwikkelen. De vliegtuigindustrie introduceert constant geavanceerde vliegtuig en helikopter ontwerpen voor betere prestaties en efficiëntie. Stedenbouwkundige planning vraagt steeds gedetailleerdere analyses van luchtstromingen binnen steden om milieu- en gezondheidszorgen mee te nemen. Dit zijn slechts een paar voorbeelden waar innovatie zowel een verbetering van prestaties duurzaamheid beoogd.

Een gemeenschappelijke deler tussen deze gebieden is de noodzaak om een diep begrip van aerodynamica te hebben. Nauwkeurige analyse van aerodynamica is essentieel voor verbetering van ontwerp, efficiëntie en effectiviteit. Echter, door de snelle vooruitgang die de huidige tijd kenmerkt, kan niet meer alleen vertrouwd worden op experimentele methodes, doordat deze veel tijd en geld kunnen kosten, en onpraktisch zijn om alle mogelijkheden te onderzoeken. Het gevolg is dat een combinatie van experimenteel een rekenonderzoek noodzakelijk wordt.

Hier komt Computational Fluid Dynamics (CFD) om de hoek kijken. Ontwikkeling van efficiënte en nauwkeurige programmas die in handen van wetenschappers en ingenieurs de mogelijkheid bieden om snel aerodynamisch onderzoek te doen en de fysica van stromingeen te bevatten, is essentieel geworden. Hierdoor wordt het huidige onderzoek gedreven. Het primaire doel van dit proefschrift is de ontwikkeling van een rekenkundig programma ter onderzoek van externe aerodynamica op een zowel nauwkeurige als efficiënt manier.

Binnen CFD zijn de meest voorkomende technieken gebaseerd op een Euleriaanse en Lagrangiaanse methodes, die beide families van methodes behelzen. Euleriaanse methodes, zoals de Eindige Volume Methode (EVM) of Eindige Elementen Methode (EEM), zijn veelvuldig toegepast in het onderzoek naar externe stromingen, met als grootste voordeel de nauwkeurigheid waarmee grenslagen gerepresenteerd kunnen worden. Echter, het diffuse karakter van deze methodes zorgt voor een toevoeging van kunstmatige diffusie in de stroming, die wervel structuren dempt, terwijl deze wervel structuren juist cruciaal zijn in vele toepassingen waarbij sterke interactie tussen wervels en object een grote rol speelt. Daarbij komt nog dat bij de studie van meerdere objecten in een stroming vaak nog speciale technieken nodig zijn, vooral wat betreft rekenrooster generatie, die de simulatie nog duurder maken.

Aan de andere kant zijn Lagrangiaanse methodes, zoals de Wervel Deeltjes Methode (WDM), uitmuntend in het bestuderen van stromingen met veel wervels, aangezien deze methodes de wervelstructuren kunnen behouden zonder deze te dempen. Daarbij zijn werveldeeltjes die de stroming representeren zelf-adaptief, ze voldoen intrinsiek aan

verre-veld randvoorwaarden en bieden een eenvoudige manier om meerdere objecten in een stroming te implementeren. Echter, het representeren van de grenslaag is uitdagend en vaak erg kostbaar, doordat de methode geen anisotrope werveldeeltjes toestaat, waardoor de methode minder geschikt is om aerodynamische krachten te voorspellen.

Lagrangiaanse programmas zijn de afgelopen drie decennia erg populair geworden voornamelijk door de grote ontwikkeling in computer hardware en dan met name in GPUs, die het zeer snel uitvoeren van berekeningen mogelijk maakt. Hierdoor aangespoord hebben onderzoekers gezocht naar manieren om dit voordeel uit te buiten in CFD, wat geleid heeft tot de ontwikkeling van hybride codes, die Euleriaanse en Lagransiaanse methodes koppelen. In deze koppeling kunnen Euleriaanse methodes dicht bij het object toegepast worden, waardoor de grenslaag nauwkeurig en efficiënt opgelost kan worden, terwijl de Lagrangiaanse methode de wervelstructuren verder af van het object blijft behouden.

Hierop voortbordurend introduceert dit proefschrift een hydride Euleriaans- Lagrangiaanse code, genaamd VPMFoam, ontwikkeld om de sterktes van beide methodes te exploiteren en hun beperkingen te minimaliseren. Hiertoe wordt OpenFOAM, wijds toegepaste open-source software, geïntegreerd met een Lagrangiaanse WDM. Het primaire doel is om een nauwkeurig simulatie programma te creëren voor externe aerodynamica, en dat efficiënt omgaat met toepassingen waarin sterke wervel-object interacties en meerdere objecten een rol spelen, waarbij een kleinere rekentijd nodig is in vergelijking met een puur Euleriaanse aanpak. OpenFOAM is gekozen met name vanwege de flexibiliteit en de brede toegankelijkheid die open-source met zich meebrengt en de grote gebruikersgroep van OpenFOAM binnen de academische en industriële gemeenschap.

De ontwikkeling van het programma richt zich met name op een 2D versie, waar stap voor stap de validatie wordt verricht, waarbij eerst eenvoudige toepassingen zonder objecten worden bekeken. Wanneer de koppeling tussen beide methodes succesvol is geverifieerd, gaat de validatie verder met toepassingen met een object, zoals de stroming om een cilinder. In deze toepassingen worden de stroming en aerodynamische coëfficiënten nauwkeurig voorspeld, waarmee een goede overeenkomt met gevestigde Euleriaanse programmas en een effectief behoud van vorticiteit in het zog aangetoond wordt. Verdere validatie behelst dynamische rekenrooster beweging en toepassingen met meerdere objecten.

Na de validatie wordt het programma toegepast op realistischer scenarios, waaronder statische en dynamische overtrek van een vleugelprofiel en de simulatie van een hydride Verticale As Wind Turbine gerepresenteerd als krachtveld model. Ook wordt de efficiëntie van het programma geanalyseerd en de eigenschap van het programma om snel en nauwkeurig simulaties te draaien aangetoond.

Door een effectieve combinatie van de sterktes van de Euleriaanse en de Lagrangiaanse methodes, kan VPMFoam de cruciale uitdagingen in aerodynamische analyse aan, met name bij toepassingen met sterke wervel-object interacties, en legt VPMFoam een sterk fundament voor verdere toepassingen, waaronder mogelijk 3D uitbreidingen. Dit maakt VPMFoam een waardevolle aanwinst voor toepassingen waarbij hoge betrouwbaarheid en rekenefficiëntie noodzakelijk zijn.

I

Foundation: Theory and Validation

1

Introduction

In this introductory chapter, the dissertation sets the stage for the research topic by highlighting the importance of Computational Fluid Dynamics (CFD) in both academic research and industrial applications. The chapter begins by discussing the two primary approaches used in CFD: the Eulerian and Lagrangian, which are distinct yet complementary in exploring the same physical phenomena. The advantages and limitations of each approach are briefly discussed, and through this analysis, certain challenges in the field are identified, indicating potential directions for future research. This discussion serves as a precursor to the primary objective of this dissertation, which aims to address specific gaps in CFD and propose solutions to some of these challenges. Subsequently, the research questions upon which this dissertation is built are presented. The chapter concludes with an overview of the dissertation's structure, including a graphical representation of the content to be covered.

1.1. The role of CFD in modern engineering

Computational Fluid Dynamics (CFD) is a sophisticated computational tool that bridges theoretical fluid mechanics and real-world applications, offering a detailed visualization and analysis of fluid flow behaviors. It leverages advanced numerical methods and computational resources to predict fluid motion, heat transfer, and related phenomena with remarkable accuracy, serving as an indispensable resource in engineering design and analysis. The applicability of CFD spans various engineering disciplines, with this dissertation narrowing its focus to external aerodynamics, the field that explores the airflow around wind turbines, helicopters, airplanes, etc.

Before the first computers were invented, fluid mechanics scientists relied solely on analytical solutions, experiments, and empirical expressions that came either from experiments or analysis after making certain assumptions. While analytical solutions are generally the most accurate solution that can be obtained, they come with extreme limitations. First of all, Navier-Stokes (N-S) equations, which govern the fluid behavior, are inherently complex equations. They are highly non-linear, allowing for analytical solutions only in simplified scenarios. This includes cases with simple geometries, like the Hagen-Poisseuille [1] flow in a cylindrical pipe, which describes the laminar flow of an incompressible Newtonian fluid through a long, straight cylindrical pipe. Analytical solutions often require multiple assumptions to address a specific problem since various phenomena can be involved, like heat transfer, chemical reactions, or the presence of porous media, which renders analytical solutions impractical. Another major limitation of analytical solutions is their inability to deal with turbulence effectively. Turbulence is a common and highly complex phenomenon in most real-world flows, which has been extensively studied; however, it remains an ongoing challenge with no universal solution and requires constant adaptation.

Despite their limitations, analytical solutions have historically played a crucial role in understanding fluid behavior. However, when analytical approaches fall short, physical experiments become indispensable. In many fields of research, physical experiments have traditionally been the primary method for investigating fluid flow phenomena. Experiments are widely used and essential for conducting new research. However, they face several challenges and limitations. Depending on the field of research, they can be unaffordable, especially if sophisticated equipment or materials are required. Taking the wind energy field as an example, specifically multibody problems like those in a wind farm, wind tunnel experiments are impractical due to space constraints. They can be prohibitively expensive in terms of both time and money. This is because each modification to the configuration of wind turbine models requires starting the entire experiment from scratch.

Given the limitations of analytical solutions in tackling complex problems and the impracticality of experiments in certain cases, the importance of CFD as a critical tool becomes undeniably clear. CFD can quickly provide results for many scenarios at a lower cost and with great accuracy. However, it is important to note that CFD can also be resource-intensive, requiring significant computational effort, especially for complex simulations.

1

1.2. Eulerian and Lagrangian approaches in CFD

Today, the field of CFD primarily relies on two approaches: the Eulerian and Lagrangian. Although these approaches significantly differ, they aim to explore the same physical phenomena based on the same equations but from different perspectives.

Eulerian solvers are the most popular choice in CFD. The Eulerian framework focuses on computing fluid properties at specific locations within a given space as the fluid flows through it (see Figure 1.1).

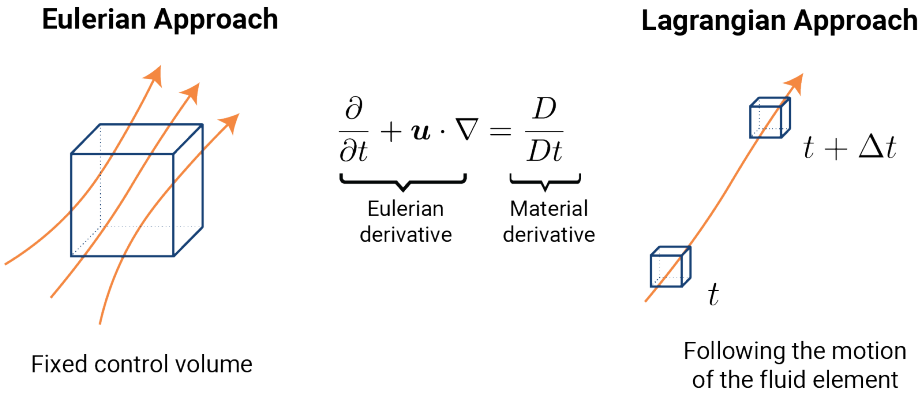


Figure 1.1: In the Eulerian approach (**left**) fluid properties are measured at specific locations within a given space as the fluid flows through it. In the Lagrangian approach (**right**) the observer tracks a specific fluid parcel as it moves through space and time, i.e., from an inertial frame of reference.

Various methods are employed in the Eulerian description concerning the discretization technique, including Finite Differences [2], Finite Volumes [3–5], Finite Elements [6, 7], and Spectral Elements [8]. Generally, these methods excel in resolving regions close to solid boundaries, including the boundary layer, and can efficiently capture viscous phenomena and vorticity generation near the solid structure. This advantage stems from their ability to handle anisotropic elements, cells that are elongated in specific directions to capture the flow features more accurately. For instance, around an airfoil, the cells near the surface are often stretched in the direction parallel to the flow (see Figure 1.2). This orientation allows the mesh to follow the gradients in the boundary layer, effectively capturing the rapid changes in velocity and vorticity that occur close to the surface, while minimizing computational cost in regions with lower flow variation.

However, their dissipative nature, especially in sparsely meshed regions, presents challenges that are not quickly addressed. Employing higher-resolution meshes significantly increases computational costs, necessitating powerful hardware and parallel processing. Moreover, constructing an accurate mesh can be complex and time-consuming, as solution quality heavily depends on achieving precision in meshing, which is challenging in intricate geometries. Furthermore, explicit time schemes (if chosen) impose strict time-step restrictions governed by the Courant-Friedrichs-Lewy (CFL) condition due to the advective terms [9]. Additionally, setting appropriate far-field boundary conditions often requires large computational domains, even when the primary interest is focused near the bluff body, which increases the computational effort. A potential solu-

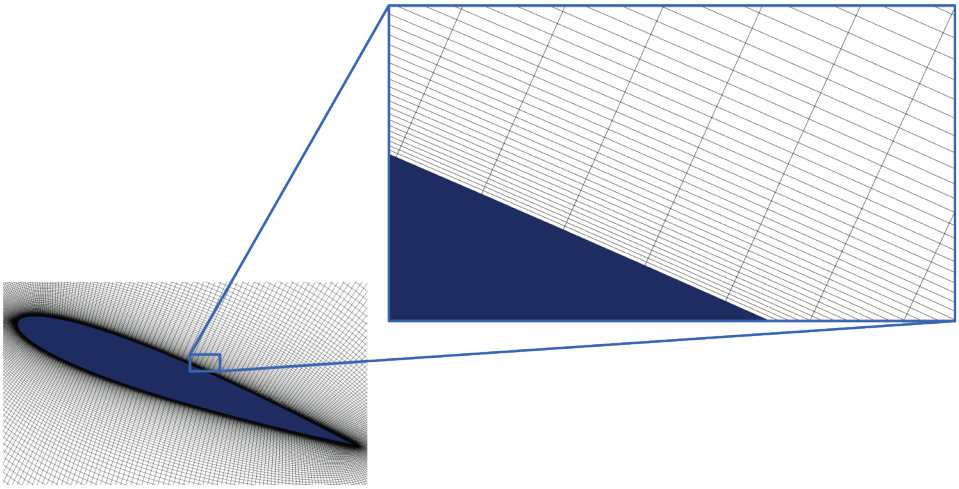


Figure 1.2: Illustration of anisotropic mesh cells around an airfoil, with a close-up showing elongated cells near the surface.

tion is the use of Adaptive Mesh Refinement (AMR) techniques, allowing for dynamic mesh adaptation based on simulation requirements through localized refinement or coarsening [10].

In contrast, Lagrangian solvers are becoming increasingly popular as they overcome many limitations Eulerian solvers face. In the Lagrangian framework, the observer tracks a specific fluid parcel as it moves through space and time, i.e., from an inertial frame of reference (see Figure 1.1). The properties of these moving particles characterize the flow. The Vortex Particle Method (VPM) is a well-known Lagrangian method wherein particles traverse the flow field, carrying vorticity. It has seen numerous applications in external aerodynamics, such as those documented by Pan et al. [11]. Extensive analyses of VPM are available in the books by Cottet et al. [12] and Winckelmans [13], with a detailed review provided by Mimeau et al. [14]. Another well-known method in the Lagrangian family is Smooth Particle Hydrodynamics (SPH), which solves the compressible N-S equations in velocity-pressure form, with particles carrying the flow's pressure and density. Lagrangian methods offer several advantages, particularly in external aerodynamics. Notably, they exhibit significantly lower numerical dissipation (theoretically zero in ideal cases) compared to conventional mesh-based numerical methods. This low dissipation arises because there are no grid-based truncation errors, and particles move solely according to the local flow velocity, eliminating the need for artificial diffusive terms. As a result, these methods enable more accurate simulations of wake dynamics, efficiently capturing far-field wake effects that would otherwise be challenging with mesh-based approaches. Furthermore, the free movement of vortex particles suits areas of high vorticity, obviating the need to resolve physically insignificant regions. Additionally, boundary conditions at infinity are naturally satisfied, as there are no physical boundaries constraining the flow, eliminating the need for the large domains required by Eulerian solvers. Their linear nature also allows for the application of acceleration tech-

niques [15–17], significantly enhancing speed.

Despite their advantages, VPMs face challenges, particularly when imposing boundary conditions near solid surfaces, requiring supplementary solvers like the vortex panel method [18] or immersed body techniques [19]. Moreover, their efficiency diminishes near solid boundaries due to the necessity of a large number of particles. Lastly, flow strain can distort the vortex particle configuration, potentially leading to computational inaccuracies [20].

1.3. External aerodynamics and hybrid solvers

Examining the strengths and limitations of the two primary approaches in CFD reveals substantial opportunities for innovation. This dissertation specifically focuses on the potential of these emerging tools in advancing external aerodynamics, an area with immense possibilities for development.

External aerodynamics, a field dominated by advection-driven flows, encompasses a wide range of engineering applications, such as wind turbines, rotors, propellers, helicopters, airplanes, trains, and buildings. The computational study in this field poses significant challenges, especially in scenarios characterized by strong body-vortex interactions, such as the flow around Vertical Axis Wind Turbines (VAWTs) [21], where the turbine blades are in constant interaction with the wake. The development of efficient and accurate simulation tools for such flows is crucial. However, as previously mentioned, Eulerian solvers suffer from artificial diffusion, leading to distorted wakes and overly diffused solutions, coupled with high computational costs. Conversely, Lagrangian solvers struggle to provide high-fidelity solutions in the presence of solid bodies within the flow.

These challenges have prompted engineers to explore the possibility of combining the advantages of both approaches while mitigating their limitations, leading to the development of hybrid Eulerian-Lagrangian solvers. The state-of-the-art developments in these hybrid solvers are discussed in detail in Chapter 4. Generally, in this hybrid approach, the Eulerian solver is applied near solid surfaces to accurately capture boundary layer phenomena, while the Lagrangian solver is used in wake regions to minimize numerical diffusion and preserve flow structures. Additionally, when parallelized, hybrid solvers enable high-speed computations. This hybrid approach is emerging as a promising alternative in external aerodynamics, where accurately capturing wake dynamics is essential, along with achieving computational efficiency.

1.4. Objective and research questions

The present dissertation centers on the development of a hybrid Eulerian-Lagrangian solver, with a particular focus on applications in external aerodynamics. A new computational software, named *VPMFoam*, is introduced, coupling a VPM with OpenFOAM, an open-source software toolkit that is highly regarded and widely used in both academia and industry. OpenFOAM was selected for this project due to its extensive use, numerous built-in functions, and proven effectiveness across various applications. Its open-source nature makes it ideal for integrating with other software and in-house solvers, as well as for developing new features.

The goal was to develop a hybrid tool that would benefit a large portion of the CFD

community, leveraging the extensive online resources, continuous development, and active troubleshooting support available. This research combines various coupling techniques from the literature on hybrid solvers, integrating them within OpenFOAM. These techniques, along with the hybrid solvers used in this study, are discussed in detail in Chapter 4.

The solver will be thoroughly verified and validated, and applied to aerodynamic cases, providing a basis for comparing the performance of hybrid solvers against existing Eulerian and Lagrangian solvers in terms of both accuracy and efficiency. The principal aim of this dissertation can be outlined as follows:

> The objective of this dissertation is to develop and validate a two-dimensional hybrid Eulerian-Lagrangian solver by coupling OpenFOAM with a Lagrangian Vortex Particle Method to simulate external aerodynamic flows. Designed for both accuracy and efficiency, this solver leverages the strengths of Eulerian methods close to solid boundaries and Lagrangian methods in wake regions, making it particularly suited to multibody problems involving body-vortex interactions. The validation of the solver is essential to establish it as a reliable tool for aerodynamic analysis.

In addressing the objective, the following research questions are explored, with corresponding chapters indicated:

1. How is OpenFOAM coupled with an in-house VPM solver, and what modifications to OpenFOAM are necessary to enable this coupling? *(Chapters 3 and 4)*
2. How can the Eulerian and Lagrangian solvers be effectively combined in a hybrid approach? *(Chapter 4)*
   (a) How can the computational domain be decomposed to facilitate the effective coupling of the two solvers?
   (b) What methods and techniques can be employed to achieve a stable and seamless integration of the two solvers?
   (c) Which regions of the computational domain should be handled by the Eulerian solver and which by the Lagrangian solver to optimize both efficiency and accuracy?
   (d) What steps are involved in the solver's evolution within a single time-step to maintain accurate coupling?
3. Can a hybrid simulation achieve the same level of accuracy as purely Eulerian simulations? *(Chapters 5 to 7 and 9 to 11)*
   (a) How does the accuracy of the hybrid solver compare to that of a pure Eulerian solver?
   (b) What factors contribute to the accuracy of the hybrid solver?
4. How can dynamic mesh simulations be effectively managed in the hybrid configuration? *(Chapters 4, 6 and 10)*

   (a) What modifications are required in OpenFOAM to enable the hybrid solver to handle dynamic meshes?

   (b) How does the accuracy of dynamic mesh simulations in the hybrid solver compare to that of pure Eulerian simulations?

5. How can the hybrid solver be applied to multibody simulations? *(Chapter 7)*

   (a) How does the hybrid solver handle complex multibody problems effectively?

   (b) What advantages does the hybrid solver offer over purely Eulerian solvers for multibody simulations?

6. What factors drive the computational cost of hybrid simulations, and how can these costs be minimized? *(Chapters 8 and 13)*

   (a) How does the computational cost of the hybrid simulation compare to that of a pure Eulerian simulation?

   (b) What strategies can be implemented to reduce computational costs in the coupled simulation system?

1.5. Outline of the dissertation

The dissertation is structured into four distinct parts, tracing the pathway taken to address the research questions. The outline of the dissertation is presented below, while Figure 1.3 provides a graphical representation of its core components.

Part I: Foundation - Theory and Validation

This foundational part of the dissertation begins by introducing the dissertation's objectives and culminates in the validation of the developed solver. It opens with an overview of the two component solvers, followed by an in-depth examination of the coupling techniques employed. The solver is then tested through various cases to verify its capability to accurately simulate different flow scenarios. The part ends with an assessment of the efficiency of the solver.

- **Chapter** 1 introduces the topic, providing background on the research, outlining the dissertation's objectives, and presenting the core research questions. It also highlights the significance of the developed solver in advancing the field of CFD.

- **Chapter** 2 presents the solver used as the Lagrangian component of the hybrid approach. It begins with an overview of Lagrangian solvers, particularly the VPM, followed by a mathematical description of the method (with further details in Appendix A). The chapter then explains how the method is applied in the current context and includes a preliminary test case to verify the solver and assess its convergence.

- **Chapter** 3 introduces OpenFOAM as the Eulerian component of the hybrid solver. It starts with an overview of the Finite Volume Method (FVM) (with additional information in Appendix B), followed by details on the specific OpenFOAM solver

used. The chapter's focus is on the modifications made to OpenFOAM to enable its coupling with the in-house Lagrangian solver, as well as an assessment of its convergence.

- **Chapter** 4 focuses on *VPMFoam*, the developed hybrid solver. The chapter begins with a literature review on the topic of hybrid Eulerian-Lagrangian solvers, similar to *VPMFoam*. It then presents the state-of-the-art of the developed solver and discusses the coupling strategy employed, detailing each step in the process. The chapter concludes with an overview of the main software and libraries, as well as the hardware utilized in the simulations presented later in the dissertation.

- **Chapter** 5 marks the beginning of the verification and validation process for the developed solver. Initially, the chapter focuses on the verification of the solver by ensuring that the coupling between the Eulerian and Lagrangian components functions correctly and that the implementation accurately solves the intended equations. This is achieved through comparisons with analytical solutions and results from other software. Subsequently, the chapter presents fundamental validation cases, encompassing static scenarios both with and without the presence of solid bodies. These cases are carefully selected to demonstrate the solver's accuracy in representing physical phenomena under static conditions, thereby establishing its reliability. Through these verification and validation efforts, it is confirmed that all processes implemented into the solver and presented in the previous chapters are functioning effectively.

- **Chapter** 6 focuses on the verification and validation of the solver in dynamic cases. The solver's accuracy is tested in scenarios involving translational, rotational, and oscillatory motions, confirming its capability to accurately simulate all possible motions that a solid object can undergo.

- **Chapter** 7 addresses the treatment of multibody simulations to ensure the solver's accuracy in handling complex interactions and dynamics in multibody scenarios.

- **Chapter** 8 concludes the first part of the dissertation by examining the efficiency and performance of the solver. Specifically, the performance of *VPMFoam* is assessed in detail, with an analysis of the computational cost of each step in a hybrid simulation, compared to a corresponding simulation in OpenFOAM. Additionally, guidelines for further accelerating the solver are proposed and discussed.

Part II: Applied Case Studies

This part of the dissertation focuses on the application of *VPMFoam* to more realistic engineering problems. Specifically, the solver is applied to various practical scenarios: a static airfoil case, a pitching airfoil simulation, and a simulation of a hybrid Darrieus-Savonius wind turbine using actuators. These applications demonstrate the solver's versatility and effectiveness in addressing complex, real-world engineering challenges.

- **Chapter** 9 presents the simulation of a static airfoil. The airfoil is analyzed under different angles of attack, identifying the stall point. The results are compared with corresponding data from the literature.

- **Chapter** 10 focuses on the simulation of a pitching airfoil, specifically investigating dynamic stall at different reduced frequencies. The results are compared with those obtained from simulations performed in OpenFOAM.
- **Chapter** 11 presents the simulation of a hybrid Darrieus-Savonius wind turbine using actuator models. The hybrid turbine is simulated using different approaches for modeling the turbines (within the Eulerian subdomain or outside), demonstrating the solver's versatility. The results showcase that various approaches can be employed to solve the same problem, and they are compared with corresponding simulations performed in OpenFOAM.

Part III: Closing Remarks

This part serves as the epilogue of the dissertation, focusing on the conclusions drawn from the analyses conducted throughout the text. It offers insightful reflections that encapsulate the essence of the dissertation. The section concludes with a broader discussion of the solver, highlighting its potential applications, future objectives, and recommendations for further research, laying the groundwork for continued exploration and development in the field. Specifically:

- **Chapter** 12 summarizes the conclusions drawn from the dissertation.
- **Chapter** 13 presents recommendations for enhancing the solver's efficiency, outlines proposed developments to make it a more comprehensive tool, suggests various tests to expand its capabilities, and highlights potential applications.

Part IV: Supplementary Material

Part IV consists of additional information which completes the dissertation.

- **Appendix** A: Mathematics of the Vortex Particle Method
- **Appendix** B: Finite volume discretization
- **Appendix** C: Source code of pimpleFoam solver
- **Appendix** D: *VPMFoam* folder structure
- **Appendix** E: Calculation of pressure gradient using vortex particles
- Bibliography
- List of Abbreviations
- Nomenclature
- Curriculum Vitæ
- List of Publications

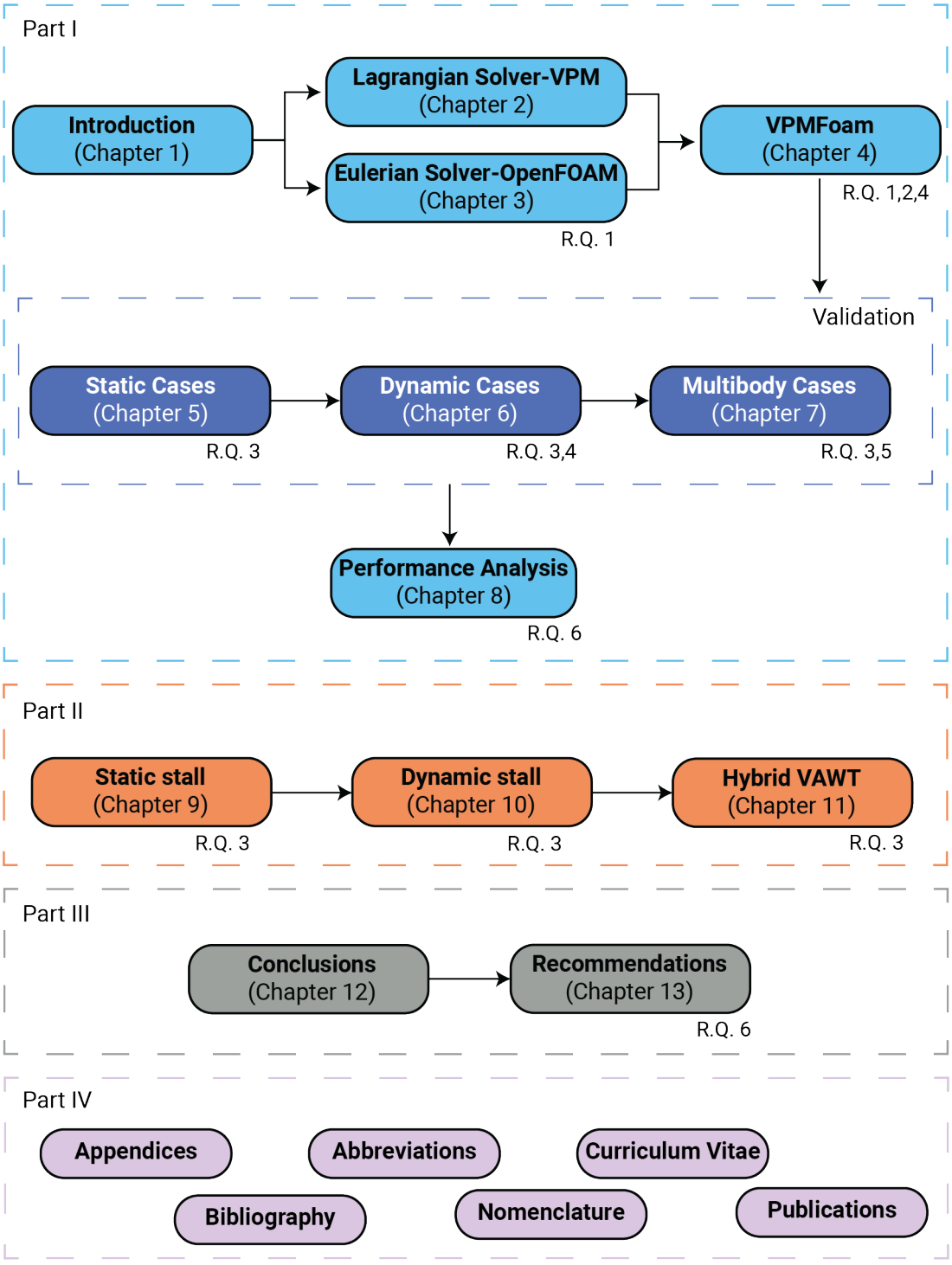


Figure 1.3: A graphical representation of the dissertation's outline.

2

The Lagrangian solver - Vortex Particle Method

The previous chapter was an introductory chapter to the research topic. This chapter deals with the Lagrangian component of the hybrid solver. It begins with a brief introduction to Lagrangian methods in fluid dynamics, focusing specifically on the Vortex Particle Method, the method that is employed here. The chapter discusses both the advantages and limitations of this method, as well as its mathematical formulation. Following this, the modeling approach employed in this study is examined. The chapter culminates in a verification of the solver and an assessment of its accuracy and convergence.

Parts of this chapter have been published in: **R. Pasolari**, C.J. Ferreira, and A. van Zuijlen, Coupling of OpenFOAM with a Lagrangian vortex particle method for external aerodynamic simulations, Physics of Fluids, vol. 35, no. 10, p. 107 115, Oct. 2023, ISSN: 1070-6631. DOI: 10.1063/5.0165878. [22]

2.1. Introduction

In fluid mechanics, the Lagrangian approach is a group of methods characterized by describing the flow as particles representing fluid elements, while the observer follows a specific fluid parcel (or sets of parcels) as it moves through space and time, i.e., in the inertial frame of reference. This characteristic distinguishes it from the Eulerian approach, where the observer remains fixed, and measures flow properties at this stationary point (more information about the Eulerian approach is discussed later in Chapter 3). In the Lagrangian approach, the flow is described through the properties carried by the particles. As Mimeau et al. [14] note, standard methods in the Lagrangian family for fluid dynamics include the SPH and the VPM. The SPH deal with the compressible N-S equations in velocity-pressure formulation, and the particles carry the flow pressure and density. Conversely, in VPM (the method employed in this dissertation), the incompressible N-S equations are solved in a velocity-vorticity formulation. As the method's name reveals, the particles carry the vorticity of the flow. For further details on VPM, readers can consult the comprehensive book by Cottet et al. [12], which describes the theory and the numerical practices of this method, while an extensive review of the vortex methods from their inception to recent developments, can be found in the review paper by Mimeau et al. [14].

2.2. Advantages and limitations

The VPM is broadly utilized, especially in external aerodynamics, due to the significant advantages it has demonstrated in this field. Here, these benefits are discussed in detail, alongside providing additional background information on the method:

- **Reduction of numerical diffusion.** The method is inherently well-suited to capturing flow physics, as it directly follows the evolution of fluid particles (vortices) and preserves sharp vorticity gradients, significantly reducing the numerical diffusion (theoretically zero) that mesh-based methods often introduce. This reduction in diffusion is possible because there are no grid-based truncation errors, and particles move solely according to the local flow velocity, eliminating the need for artificial diffusive terms. This accuracy is particularly advantageous in external aerodynamics, where accurately capturing wake dynamics is essential, especially in multibody systems and body-vortex interactions, where precise wake representation is vital.

- **Adaptability of vortex particles.** The vortex particles are free to move according to the local velocity field acting on them, which is induced by other particles, the freestream flow, or surface elements. As a result, they naturally accumulate in areas of high vorticity, while regions with negligible vorticity remain particle-free. This selective concentration eliminates the need to resolve regions without any physical importance, reducing the computational effort and enhancing the general efficiency of the method. For instance, Figure 2.1 demonstrates this adaptability through the particles' distribution in the case of the flow around a circular cylinder. It clearly shows particle accumulation in the wake regions where the von Kármán vortex street develops, leaving other less critical areas unresolved. Addi-

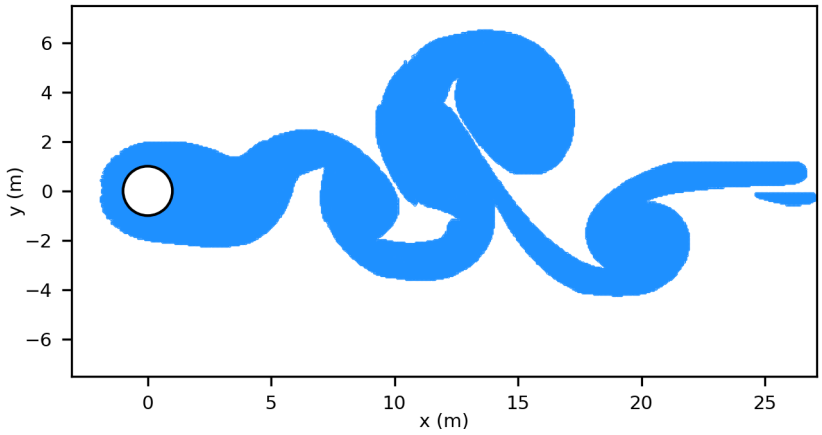


Figure 2.1: The distribution of the vortex particles for an arbitrary moment of the case of the flow around a circular cylinder. The particles accumulate in the wake regions where vorticity is present, leaving other less critical areas unresolved.

tionally, new particles are continuously introduced into the regions of interest, a process that can be achieved utilizing methods such as Eulerian-based seeding, lifting line theories, or directly, when the circulation generated by an object is predetermined. This ensures that the flow field remains accurately represented even as particles are convected out of the domain.

- **Satisfaction of far-field boundary conditions.** The boundary conditions in the far-field are automatically satisfied, as the domain is not confined by any physical boundaries. This is a crucial advantage over mesh-based solvers, where the computational domain must be predetermined and often enlarged to apply realistic boundary conditions, increasing the computational load in this way.

- **Vorticity field transportation by particles.** The fact that the method relies on the vorticity field is also very noteworthy. Vorticity is a highly relevant field in aerodynamics, closely associated with the aerodynamic lift force. This association is articulated by the Kutta-Joukowski theorem [23], which establishes a direct connection between circulation (a macroscopic measure of vorticity) and the lift force exerted on an aerodynamic body.

- **Support for larger time-steps.** Additionally, VPM is not constrained by the CFL criterion, which is a limitation for mesh-based explicit solvers that connect the time-step to the minimal grid size and the advection velocity. Mimeau et al. [14] notes that a related quantity, the so-called Lagrangian Courant-Friedrichs-Lewy (LCFL) number, can be defined based on the velocity gradient using the following expression:

$$\Delta t \leq \frac{LCFL}{|\nabla \mathbf{u}|_\infty} = \frac{1}{|\nabla \mathbf{u}|_\infty}$$

This condition is often less restrictive than the CFL criterion in Eulerian methods, since particles in Lagrangian vortex methods move with the flow. As a result, the relative motion between neighboring particles is typically smaller than the absolute fluid velocity crossing fixed grid cells. It is important to note, however, that the CFL and LCFL numbers are not directly comparable. In mesh-based explicit methods, exceeding a CFL number of 1 leads to numerical instability, and even in implicit methods, a high CFL may reduce accuracy. In contrast, in vortex particle methods, a high LCFL number generally does not cause instability, but may lead to a loss of accuracy.

Barba [24] uses the ω_{max}^{-1} criterion, where the time-step should be lower than the inverse of the maximum vorticity in the flow. This criterion is based on estimating the time-step such that the particles can follow a curved path accurately.

- **Acceleration potential.** The vortex particles in VPM are vortex flow elements, one of the elementary flows in fluid dynamics [23]. Their mathematical expression satisfies Laplace's equation, indicating that their solutions are linear. This principle implies that the velocity field at any point in the flow can be determined as the sum of the velocities induced by each vortex. From a programming perspective, this characteristic facilitates parallel processing of the advection problem using either Central Processing Units (CPUs) [25] or Graphics Processing Units (GPUs) [26, 27]. Furthermore, the Fast Multipole Method (FMM) [15–17], a numerical algorithm utilizing a hierarchical tree structure for efficient particle interaction computations, can be employed. FMM can reduce the number of calculations from N^2 to $Nlog(N)$, where N represents the number of particles. In practice, FMM is often considered to have $\mathcal{O}(N)$ complexity, since the $\mathcal{O}(NlogN)$ component stems from the sorting, which is typically not the dominant contributor to the overall computational cost.

  Additionally, fast Poisson solvers can also be used to efficiently recover the velocity field from the vorticity distribution, accelerating computations in vortex methods.

On the other hand, the challenges faced by the method are primarily related to the accuracy and efficiency of the solver in treating solid boundaries within the flow. The method has not yet advanced to the point of utilizing anisotropic elements. Consequently, capturing the flow structures near solid objects relies on the use of a vast number of particles, which negatively impacts the solver's efficiency. Moreover, imposing boundary conditions on solid surfaces is a challenging task. The solver requires an additional method, such as the vortex panel method [18] or the immersed body technique [19], to achieve accurate boundary conditions. Finally, the nature of the solver, which allows particles to move freely, leads to strains in the flow that cause the vortex particle configuration to lose its structure. This distortion can result in computational inaccuracies [20]. This phenomenon requires special treatment, which will be discussed later in this chapter.

2.3. Mathematical description

The VPM is based on the vorticity field. Vorticity is a vector field that represents the local spinning of the fluid at each point. Mathematically is defined as the curl of the velocity field:

$$\boldsymbol{\omega} = \nabla \times \mathbf{u} \tag{2.1}$$

The actual rotation rate (angular velocity) of the fluid element is $\frac{1}{2}\boldsymbol{\omega}$, meaning that vorticity represents twice the local angular velocity.

Another important quantity in VPM is the circulation. The circulation (denoted as Γ) is strongly related to the vorticity field since it measures the total rotation around a closed loop in the flow. It is defined as the line integral of the velocity field around a closed contour C:

$$\Gamma = \oint_C \mathbf{u} \cdot \mathbf{dl} \overset{\text{Stokes' Theorem}}{=} \iint_S (\nabla \times \mathbf{u}) \cdot \mathbf{dS} = \iint_S \boldsymbol{\omega} \cdot \mathbf{dS} \tag{2.2}$$

Here, C is the boundary of the surface S, and the orientation of C follows the right-hand rule relative to the direction of the surface normal vector dS.

2.3.1. Governing equations

The most common formulation of the incompressible N-S equations with constant viscosity in CFD is in terms of velocity and pressure (Equation 2.3).

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} \tag{2.3}$$

where

- $\mathbf{u}$ is the velocity field.
- t is the time.
- ∇ is the nabla operator.
- p is the pressure field.
- ρ is the mass density.
- ν is the kinematic viscosity.

Taking the curl of Equation 2.3 will end up in the velocity-vorticity formulation of the Equation 2.4 (detailed derivation is presented in Appendix A):

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\,\mathbf{u} + \nu \nabla^2 \boldsymbol{\omega} \qquad \text{in 3D} \tag{2.4a}$$

$$\frac{\partial \omega}{\partial t} + (\mathbf{u} \cdot \nabla)\,\omega = \nu \nabla^2 \omega \qquad \text{in 2D} \tag{2.4b}$$

In the formulation of Equation 2.4, the pressure field is absent (the curl of the gradient of a scalar field is always zero). This means that there is no need for velocity-pressure coupling techniques like the Semi-Implicit Method for Pressure-Linked Equations (SIMPLE), Pressure Implicit with Splitting of Operators (PISO), and PIMPLE algorithms in Eulerian solvers, which often increase the computational cost of the simulations. As one can see in Equation 2.4, in two dimensions, the vortex stretching term $(\omega \cdot \nabla)\,\mathbf{u}$ is eliminated (all terms are zero), while the vorticity field has only one non-zero component and thus is a scalar. Consequently, the set of 2D incompressible equations can be summarized as:

$$\frac{D\omega}{Dt} = \nu \nabla^2 \omega \qquad \text{N-S equations in 2D} \tag{2.5a}$$

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{incompressibility constraint} \tag{2.5b}$$

$$\nabla \times \mathbf{u} = \omega \qquad \text{velocity-vorticity relation} \tag{2.5c}$$

$$\omega(\mathbf{x}, t) = \omega_0(\mathbf{x}) \qquad \text{initial vorticity} \tag{2.5d}$$

while their boundary conditions are:

$$\lim_{|\mathbf{x}| \to \infty} \mathbf{u}(\mathbf{x}, t) = \mathbf{U}_{inf} \qquad \text{velocity at infinity} \tag{2.6a}$$

$$\lim_{|\mathbf{x}| \to \infty} \omega(\mathbf{x}, t) = 0 \qquad \text{vorticity at infinity} \tag{2.6b}$$

2.3.2. Discretization into point vortex elements

In VPM the fluid is discretized into vortex particles which carry the vorticity field of the flow. The continuous vorticity field is approximated by a sum of Dirac delta functions :

$$\omega(\mathbf{x}) \approx \sum_{i=1}^{N} \Gamma_i \delta(\mathbf{x} - \mathbf{x}_i) \tag{2.7}$$

where:

- Γ_i is the circulation of the i-th vortex particle.
- $\mathbf{x}_i = (x_i, y_i)$ is the position of the i-th vortex particle.

The velocity field induced by a vortex particle located at $\mathbf{x}_i$ at any point $\mathbf{x}$ can be retrieved from the vorticity field using the 2D Biot-Savart law:

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi} \Gamma_i \frac{(\mathbf{x} - \mathbf{x}_i)^\perp}{\|\mathbf{x} - \mathbf{x}_i\|^2} \tag{2.8}$$

where $(\mathbf{x} - \mathbf{x}_i)^\perp$ is the perpendicular vector to $\mathbf{x} - \mathbf{x}_i$, defined as:

$$(\mathbf{x} - \mathbf{x}_i)^\perp = (y - y_i, -(x - x_i))$$

The velocity and vorticity fields are linear solutions, so the corresponding total fields can be written as the linear combination of the contribution of all the particles. The total velocity field is obtained by summing the freestream velocity ($\mathbf{U}_{inf}$) with the induced

velocity field. Hence, it concludes with Equation 2.9 (detailed derivation is presented in Appendix A):

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi}\sum_{i=1}^{N}\Gamma_i\frac{(\mathbf{x}-\mathbf{x}_i)^{\perp}}{\|\mathbf{x}-\mathbf{x}_i\|^2} + \mathbf{U}_{inf} \tag{2.9}$$

2.3.3. Discretization into mollified vortex elements

In order to obtain a smooth representation of the vorticity field instead of a spurious one produced by Dirac distributions and to avoid the singularity that arises from point vortices, mollified kernels can be used. These kernels provide the particles with a finite core, resulting in smooth induced velocity and vorticity fields. The smoothing function (cutoff function) is selected to ensure the conservation of total circulation. Here, a second-order Gaussian kernel is utilized:

$$K(\mathbf{x}) = \frac{1}{k\pi\sigma^2}\exp\left(-\frac{\|\mathbf{x}\|^2}{k\sigma^2}\right) \tag{2.10}$$

where

- $K(\mathbf{x})$ is the value of the Gaussian function at the point $\mathbf{x}$.
- k is the Gaussian cutoff parameter. Here $k = 2$.
- σ is the standard deviation of the Gaussian distribution, which controls the width of the kernel. In VPM it represents the core size of the mollified particles.

Using the Gaussian kernel, the total velocity and vorticity fields can be written as (detailed derivation is presented in Appendix A):

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi}\sum_{i=1}^{N}\Gamma_i\left(1-\exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}\right)\right)\frac{(\mathbf{x}-\mathbf{x}_i)^{\perp}}{\|\mathbf{x}-\mathbf{x}_i\|^2} + \mathbf{U}_{inf} \tag{2.11a}$$

$$\omega(\mathbf{x}) = \sum_{i=1}^{N}\Gamma_i\frac{1}{2\pi\sigma^2}\exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}\right) \tag{2.11b}$$

2.3.4. Initialization of the vortex particles

The initialization of the vortex particles is very important, as it plays a crucial role in the accuracy and convergence of the method, as will be shown in detail in the last section of this chapter. Here, an important quantity for the vortex particles must be defined, which is the overlap (or overlap ratio), denoted as λ and defined as:

$$\lambda = \frac{h}{\sigma} \tag{2.12}$$

The overlap is the ratio between the particles' spacing (h) and their core size (σ), indicating how much two particles overlap (Figure 2.2). As explained in the literature, vortex particles need to overlap with each other to achieve a smooth representation of the velocity field [12, 24].

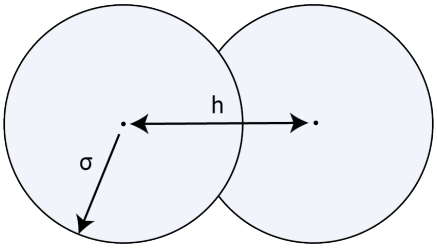


Figure 2.2: The overlap of two neighboring particles. The particles' spacing is denoted as h, while σ represents the core size of the particles.

Assuming a known vorticity field, the vortex particles can be initialized using the circulation that occupies their local area (Equation 2.13).

$$\Gamma_p = \omega_p h^2 \tag{2.13}$$

While VPM avoids many of the numerical diffusion issues common in Eulerian methods, this initialization formula still introduces a form of diffusion. Specifically, the finite core size of each particle causes its vorticity to be distributed over a surrounding region, effectively smoothing the field. However, unlike in Eulerian schemes, this diffusion does not grow with time and remains bounded. Moreover, it is controllable through the choice of particle core size, allowing a balance between accuracy and smoothness in the velocity field reconstruction.

2.3.5. Evolution of the vortex particles

The VPM is primarily used for describing incompressible, inviscid flows. However, to make the model more realistic and capable of solving problems with greater accuracy, it is essential to consider the viscous effect. This means that the processes occurring during the evolution of the flow are advection and diffusion. Chorin [28] proposed a viscous splitting algorithm, where these two processes are decoupled and solved sequentially: first, the particles are advected, and then they are diffused. The velocity-vorticity formulation of the N-S equations for an incompressible fluid with constant viscosity in 2D and in terms of the material derivative is given by the following expression:

$$\frac{D\omega}{Dt} = \frac{\partial \omega}{\partial t} + (\mathbf{u}\cdot\nabla)\omega = \nu\nabla^2\omega \tag{2.14}$$

Thus, by employing the viscous splitting algorithm, the evolution process can be divided into the following steps.

- **Advection**

$$\frac{D\omega}{Dt} = 0 \tag{2.15}$$

  This means that the vorticity of a fluid particle remains constant as it moves through the flow. Physically, this represents the conservation of vorticity for each fluid par-

ticle in the absence of diffusion and external forces. Hence, the advection of vorticity by the flow can be described by tracking the positions of fluid parcels, leading to the Ordinary Differential Equations (ODEs):

$$\frac{d\mathbf{x}_p}{dt} = \mathbf{u}\left(\mathbf{x}_p\right) \tag{2.16a}$$

$$\frac{d\omega_p}{dt} = 0 \tag{2.16b}$$

Equation 2.16 is solved using a 4^{th} order Runge-Kutta integration scheme to introduce stability to the method and allow for greater time-steps.

- **Diffusion**

$$\frac{D\omega}{Dt} = \nu\nabla^2\omega \tag{2.17}$$

Hence, the diffusion, in the Lagrangian framework, can be expressed as the following ODEs:

$$\frac{d\mathbf{x}_p}{dt} = 0 \tag{2.18a}$$

$$\frac{d\omega_p}{dt} = \nu\Delta\omega_p \tag{2.18b}$$

The solution of the diffusion equation is more complicated than that of advection, making modeling essential. This topic is covered in the next section.

It is important to recognize that the accuracy of the viscous splitting algorithm is constrained by the order of the splitting scheme itself, regardless of the numerical method used within each substep. The original splitting method introduced by Chorin [28] is first-order accurate in time, as it simply decouples the advection and diffusion processes and solves them sequentially within a full time-step. This results in a splitting error that limits the overall temporal accuracy of the method.

Higher-order splitting schemes, such as the second-order Strang splitting [29], reduce this error by symmetrically ordering the substeps: performing a half-step of diffusion, followed by a full-step of advection, and concluding with another half-step of diffusion. This arrangement improves the temporal accuracy of the coupled scheme, especially when paired with high-order solvers for each subproblem. In this work, the first-order Chorin splitting scheme is adopter. Nevertheless, a 4^{th} order Runge-Kutta integration scheme is used here. While this does not increase the global time accuracy beyond first order, it significantly improves the accuracy of particle trajectory integration and enhances the stability of the method, allowing for larger time-steps.

2.4. Modeling techniques

With the basic mathematics governing the VPM established, the next step is to address the modeling techniques utilized in the solver. These techniques focus on the redistribution process of the particles, the diffusion process, and the methods employed to control the excessive increase in the number of particles within the flow.

2.4.1. Redistribution

As previously stated, one of the disadvantages of the VPM is that particles lose their initial spatial distribution due to the strains in the flow, as the particles are free to move. Many studies have shown that this distortion can lead to inaccuracies [20, 24] due to the loss of sufficient overlap. To visualize this, Figure 2.3 shows the initial and final (after $1.0s$ simulation time) distribution of the particles in an arbitrary diffusing vortex case. Figure 2.3a illustrates that without employing any special technique to preserve their structure, the particles become clustered due to the strains of the flow field, completely losing their overlap in some regions.

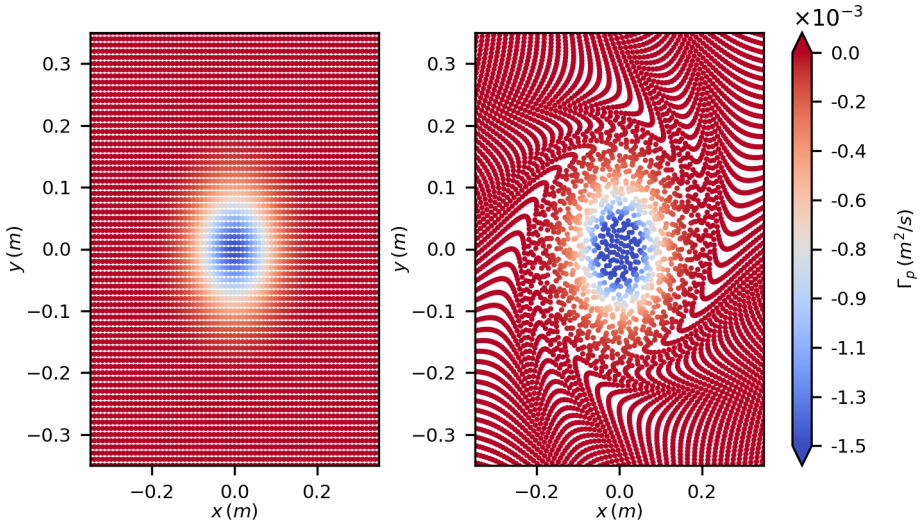
One method to address this issue is Beale's circulation processing [30], where the circulation of the particles is recalculated to adjust the particles' volume and maintain overlap, using an iterative method to solve the produced system. The problem with this method is that it is not guaranteed to converge [24]. Another method is the rezoning of the particles, as mentioned by Barba [24]. In this method, the particles are re-initialized into a structured grid. The vorticity is obtained at the new particle locations using the vorticity induced by the previous particles. The new particles are then re-initialized using the formula in Equation 2.13. However, this method is computationally expensive.

The method employed here is the redistribution (or remeshing) of the vortex particles. This technique is based on the interpolation of the particles' circulation from the distorted particles to the new structured particles. Let Γ_p be the circulation of one particle in the new grid, and $\hat{\Gamma}_q$ be the circulation of a distorted particle. The new circulations of the particles are calculated using Equation 2.19.

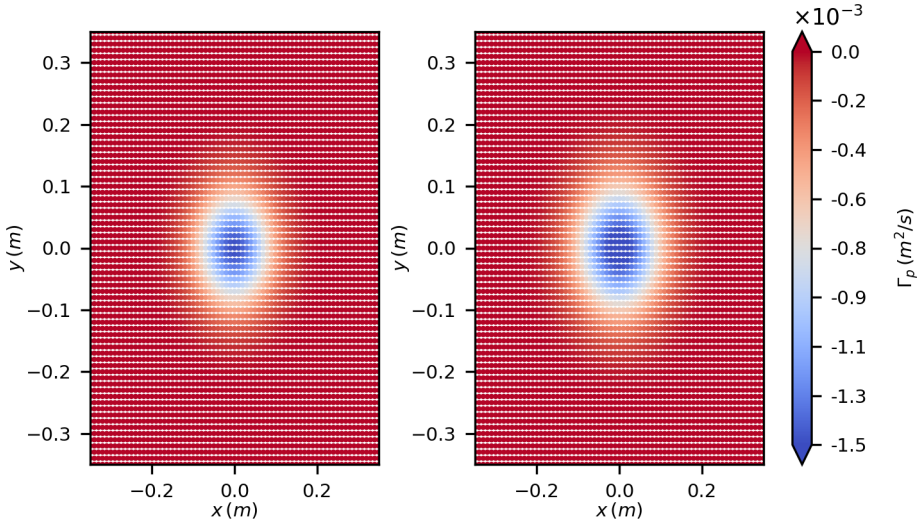
$$\Gamma_p = \sum_q \hat{\Gamma}_q W\left(\frac{x_p - \hat{x}_q}{h}\right) \tag{2.19}$$

Redistribution in 2D or 3D grids is constructed as Cartesian tensor products of 1D kernels. Many different interpolation kernels have been used in this context, with the most common being the interpolation kernels of the Λ and M families. The interpolation kernel used here is the M_4' kernel, which has been extensively employed [24, 31]. This kernel is third-order accurate, piecewise smooth, and B-spline based, using four support nodes in each direction (see Figure 2.4). The 1D kernel is defined as:

$$M_4'(\eta) = \begin{cases} 1 - \frac{5}{2}\eta^2 + \frac{3}{2}|\eta|^3 & \text{if } 0 \le |\eta| \le 1 \\ \frac{1}{2}(1-|\eta|)(2-|\eta|)^2 & \text{if } 1 \le |\eta| \le 2 \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$



(a) Distribution of vortex particles without employing the redistribution process.



(b) Distribution of vortex particles after employing the redistribution process.

Figure 2.3: Distribution of vortex particles for the case of a diffusing vortex. On the left, the initial distribution of the particles is illustrated, while on the right the distribution after 1.0s of simulation. In the first case the particles are evolved without employing the redistribution process, while in the second case, a redistribution process is employed.

where,

$$\eta = \frac{x_p - \hat{x}_q}{h} \tag{2.21}$$

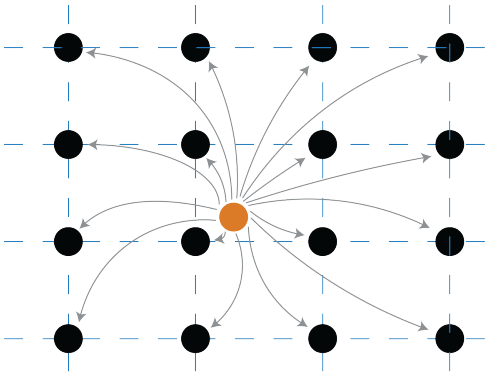


Figure 2.4: Graphical representation of the interpolation of one particle's circulation to 16 neighboring nodes using four support nodes in each direction.

Figure 2.5 shows the distribution of the kernel in 1D. Using this kernel and considering that the remeshing is applied in 2D, the interpolation takes the form shown in Equation 2.22.

$$\Gamma_p = \sum_q \hat{\Gamma}_q M_4'\left(\frac{x_p - \hat{x}_q}{h}\right) M_4'\left(\frac{y_p - \hat{y}_q}{h}\right) \tag{2.22}$$

The effect of the redistribution process is illustrated in Figure 2.3b. In the context of this dissertation, the redistribution is applied every f_{red} time-steps. Generally, it can occur after more than one time-step, but here an $f_{red} = 1$ is chosen.

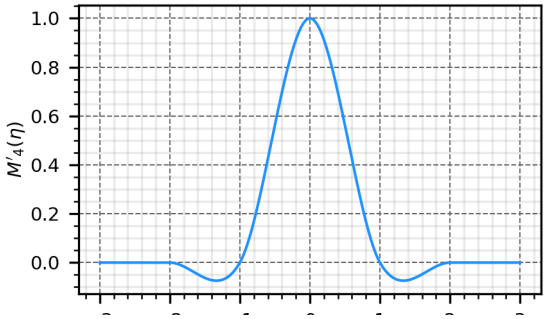


Figure 2.5: The distribution of the M_4' kernel in 1D.

2.4.2. Diffusion

Diffusion is a crucial part of the VPM when dealing with viscous flows. Over the years, many different techniques have been developed to model the diffusion process. Some

of these are probabilistic, such as the Random Walk Method (RWM) [28], while most are deterministic, including the Particle Strength Exchange (PSE) [32], Core Spreading Model (CSR) [33], Diffusion Velocity Model (DVM) [34], and the Vortex Redistribution Method (VRM) [35]. A brief description of these techniques can be found in the review by Mimeau et al. [14] and in the doctoral thesis of Barba [24]. In the context of the present dissertation, the following methods are employed:

- **Core Spreading Model**
  In this model, the solution of the Equation 2.18 is used. The vorticity field of the diffused particle evolving in time can be written as:

$$\omega(\mathbf{x},t) = -\frac{\Gamma_p}{4\pi\nu t} exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{4\nu t}\right) \tag{2.23}$$

  This equation is equivalent to the vorticity field of a finite core-size particle if the core radius is defined as $\sigma^2 = 2\nu t$. This implies that the particle expands its core over time, simulating the diffusion process. This method is computationally efficient as it does not introduce new particles into the simulation. However, achieving a smooth and accurate representation of the vorticity field requires particles with a small core size. In this specific method, the presence of particles with a wide core is inevitable. For this reason, reliance on this diffusion model alone is insufficient; instead, it is combined with other diffusion methods. Specifically, the CSR is employed in regions where an extremely precise flow description is not necessary, particularly in distant areas downstream from the solid body when simulating the aerodynamics of bluff bodies.

- **Vortex Redistribution Method**
  The primary diffusion method used throughout this dissertation is the VRM, specifically in the form proposed by Tutty [36]. This method is based on the redistribution method of Shankar et al. [35], but instead of redistributing circulation between vortex elements, the circulation carried by each particle is independently transferred onto a small set of neighboring nodes. This method can perform the redistribution and diffusion in a single step. Similar to the redistribution method covered above, the circulation of the vortex particles is updated as:

$$\Gamma_p = \sum_q \hat{\Gamma}_q W_{qp} \tag{2.24}$$

  Γ_p is the circulation of the updated particle p, while $\hat{\Gamma}_q$ is the circulation of the particle q before the process, and W_{qp} is the weight of the transfer from q to p. The method is structured in such a way that the conservation of vorticity, center of vorticity, linear momentum, and angular momentum is enforced, since the weights W_{qp} are calculated using the following set of equations:

$$\sum_p W_{qp} = 1 \tag{2.25a}$$

$$\sum_p W_{qp}(x_p - x_q) = \sum_q W_{qp}(y_p - y_q) = 0 \tag{2.25b}$$

$$\sum_p W_{qp}(x_p - x_q)^2 = \sum_q W_{qp}(y_p - y_q)^2 = 2h_v \tag{2.25c}$$

$$\sum_p W_{qp}(x_p - x_q)(y_p - y_q) = 0 \tag{2.25d}$$

where $h_v = \sqrt{\Delta t / Re}$ is the characteristic diffusion distance over time Δt (with Re being the Reynolds number). Similar to the redistribution process, the kernel is calculated for a 1D case, and for higher dimensions tensor product calculations are used.

For the 1D case, consider a vortex of strength Γ, placed at $x = x_v$, where $x_i \leq x_v \leq x_{i+1}$ (see Figure 2.6).

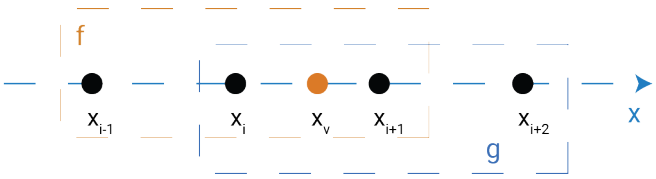


Figure 2.6: This is a graphical representation of the derivation for the VRM of Tutty [36] in 1D. A vortex is placed at $x = x_v$. The strengths of the interpolation can be carried out by taking the center node as i (weights f), or the node $i+1$ (weights g).

Tutty [36] defines the distances $\Delta = (x_v - x_i)/h$ and $\Delta_1 = (x_v - x_{i+1})/h$. A solution to the redistribution equations, taking the node i as the center node is:

$$f_i = 1 - 2\left(\frac{h_v}{h}\right)^2 - \Delta^2 \tag{2.26a}$$

$$f_{i-1} = \frac{1}{2}\left(1 - f_i - \Delta\right) \tag{2.26b}$$

$$f_{i+1} = \frac{1}{2}\left(1 - f_i + \Delta\right) \tag{2.26c}$$

while the weight for all the other grid points is zero. A second solution, taking the node $i+1$ as the center node is:

$$g_{i+1} = 1 - 2\left(\frac{h_v}{h}\right)^2 - \Delta_1^2 \tag{2.27a}$$

$$g_i = \frac{1}{2}\left(1 - g_{i+1} - \Delta_1\right) \tag{2.27b}$$

$$g_{i+2} = \frac{1}{2}\left(1 - g_{i+1} + \Delta_1\right) \tag{2.27c}$$

while the weight for all the other grid points is zero. Since both are solutions, a linear combination of both can be used:

$$F_p = (1-\Delta) f_p + g_p \Delta, \qquad p = i-1, i, i+1, i+2 \tag{2.28}$$

With this process, the circulation is redistributed into three points if the element is exactly on a grid point, and into four points if the element is between two grid points. Tutty [36] states that a condition for this redistribution process is:

$$\frac{h_v}{h} < \frac{1}{\sqrt{2}} \tag{2.29}$$

For the 2D case, the kernel is computed as the tensor product of two 1D kernels:

$$W_{p,l} = F_p G_l, \qquad p = i-1, ..., i+2, \;\; l = j-1, ..., j+2 \tag{2.30}$$

where G_l is the kernel in the y direction, and j is the grid point in the y direction, similar to i in the x direction.

2.4.3. Control of the particles' number

One main issue of particle methods is that the number of particles in the flow can increase uncontrollably. For example, consider the case of the flow around a cylinder. New particles should be created near the wall where the vorticity is generated, and then advected into the wake, continually increasing the number of particles. Moreover, the total number of particles also increases due to the redistribution and diffusion processes. In the redistribution process, new particles are added at the edges to conserve the center of vorticity and higher orders. During the diffusion process, if a VRM or PSE model is used, it makes sense to add particles at the edges to physically represent the diffusion process.

The uncontrolled increase in the number of particles can make the simulation computationally highly expensive, often without any gain in accuracy. For example, during the redistribution and diffusion processes, many particles with extremely low circulation are created, as well as particles that are diffused and weakened in strength, carrying minimal circulation. Additionally, particles far away from the region of interest increase the cost of the simulation without adding significant information to the flow. For this reason, the following techniques are used to control the total number of particles.

- **Population control**
  In the population control process, particles that carry minimal circulation can be removed. The parameters that need to be defined here are the local circulation threshold Γ_{loc} and the global circulation threshold Γ_{glob}. Particles with circulation lower than Γ_{loc} can be removed, but only when the total circulation from all these particles is lower than Γ_{glob}. This process is illustrated in the following pseudocode:

Algorithm 1: Population control process

```
function POPULATION_CONTROL
    loop

    for each particle p do
        if |Γ_p| < Γ_loc then
            flag particle p
        end if
    end for
    Γ_flagged_sum ← Σ|Γ_p| for all flagged particles
    if Γ_flagged_sum < Γ_glob then
        remove flagged particles
    else
        Γ_loc ← 0.1 × Γ_loc
    end if
    repeat until a set of particles is removed
end function
```

The process is employed every $f_{pop.control}$ steps during the simulation. An option is included for specifying the thresholds, either by providing absolute values for the threshold circulations or by using values relative to the maximum absolute circulation in the flow. The population control process has also been utilized in other VPM codes [24, 31].

- **Control far-field particles**
  As previously mentioned, particles may accumulate in regions where their direct influence on quantities of interest, such as aerodynamic forces on a body is minimal. For example, in the case of flow around a cylinder, particles may be convected far downstream (e.g., over 100 diameters behind the cylinder), where their contribution to near-body forces becomes negligible. However, since these particles still carry vorticity, they remain part of the global circulation, and their removal must be handled carefully. It is crucial that particle removal does not introduce unphysical effects or alter key flow features near the region of interestsuch as the body forces on the aerodynamic object being studied. As Stock et al. [37] explain, the particles cannot be removed directly, as the absence of vorticity on the other side of the plane where particles are removed causes the remaining vorticity to reflect off the interface. Instead, the particles are gradually weakened until they reach the

population control threshold and are then removed. A pseudocode of this process is shown here:

Algorithm 2: Control far-field particles

```
function CONTROL_FARFIELD
    initialize flagged_particles ← {}
    for each particle p do
        if (x_p) > x_farfield or (y_p) > y_farfield then
            add p to flagged_particles
        end if
    end for
    for each particle p in flagged_particles do
        Γ_p ← 0.5 × Γ_p
    end for
end function
```

2.5. Accuracy and convergence study

The accuracy and convergence of the VPM is a very important and complex topic that has been extensively studied in the literature [24, 30, 38]. In the doctoral thesis of Barba [24], it is highlighted that the accuracy of the discretization of a vorticity field in VPM is highly dependent on the cutoff function that approximates the delta function (a Gaussian here), the cutoff parameter k (here $k = 2$), and the way that the particles are initialized. In the same thesis, a comprehensive analysis of the accuracy of VPM in 2D is presented, considering various cutoff functions, different particle placements (square vs triangular vs stretched triangular lattice), and different cutoff parameters of the Gaussian function.

Here, the convergence tests will be applied only for the Gaussian kernel with $k = 2$, which is the one implemented in the VPM solver used here[1]. To validate the solver and assess its convergence rate, the stationary Lamb-Oseen vortex test case [39] is used. The Lamb-Oseen vortex is characterized by a finite core and diffuses over space and time. The analytical solutions for the velocity and vorticity fields induced by the vortex are presented in Equation 2.31.

$$u_\theta = \frac{\Gamma_v}{2\pi r}\left[1.0 - exp\left(-\frac{r^2}{4\nu(t+\tau)}\right)\right] \quad , \quad u_r = 0 \tag{2.31a}$$

$$\omega = \frac{\Gamma_v}{4\pi\nu(t+\tau)} exp\left(-\frac{r^2}{4\nu(t+\tau)}\right) \tag{2.31b}$$

where, u_θ is the circumferential velocity, u_r is the radial velocity and ω is the vorticity. Γ_v is the strength of the vortex, t is the simulation time, τ is the time constant (for smooth distribution of the vorticity field), ν is the kinematic viscosity and r is the distance from the core center.

[1] The original code was developed by Palha et al. [31] in the context of pHyFlow solver. Significant modifications have been made to this work.

The Lamb-Oseen vortex is discretized into vortex particles with a spacing h, core radius σ, and strength Γ_i^0. To account for the diffusion effect that occurs during discretization, the value of $\sigma^2/2\nu$ is subtracted from the time $t+\tau$. This is known as "time shift correction" [24]. Therefore, the strength assigned to each particle i, using the initialization provided in Equation 2.13, is:

$$\Gamma_i^0 = \omega_i^0 \cdot h^2 = \left\{ \frac{\Gamma_\nu}{4\pi\nu(t+\tau-\sigma^2/2\nu)} \exp\left(-\frac{r_i^2}{4\nu(t+\tau-\sigma^2/2\nu)} \right) \right\} \cdot h^2 \tag{2.32}$$

The accuracy and convergence of the solver are measured using the E_2 and E_∞ norms for the velocity and vorticity errors, calculated using Equation 2.33, at the locations of the vortex particles, with L and A corresponding to the Lagrangian and analytical solutions, respectively.

$$E_2^u = \sqrt{\sum_i |u_{L,i} - u_{A,i}|^2 h^2 + \sum_j |v_{L,j} - v_{A,j}|^2 h^2} \tag{2.33a}$$

$$E_\infty^u = \frac{\max|u_L - u_A| + \max|v_L - v_A|}{\max|u_A| + \max|v_A|} \tag{2.33b}$$

$$E_2^\omega = \sqrt{\sum_i |\omega_{L,i} - \omega_{A,i}|^2 h^2} \tag{2.33c}$$

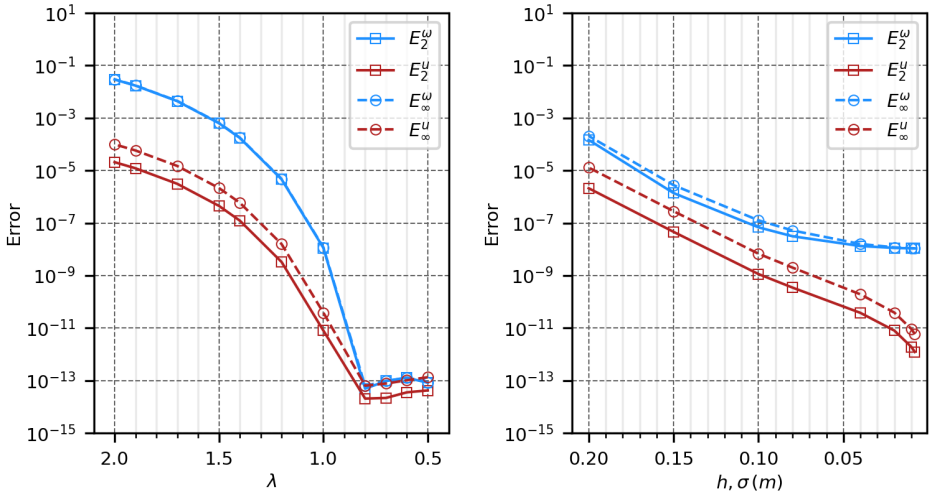
$$E_\infty^\omega = \frac{\max|\omega_L - \omega_A|}{\max|\omega_A|} \tag{2.33d}$$

2.5.1. Initialization errors

As already mentioned, the initialization of the particles plays a crucial role in the accuracy of the solver. Here, since the kernel of the particles is predefined, the parameters that affect the initial solution are the particles' spacing h, their core size σ, and the overlap ratio $\lambda = h/\sigma$. The influence of the initial solution on these parameters is tested in the Lamb-Oseen case similar to that used by Barba [24]. A vortex with $\tau = 4.0\,s$ (quite diffused), $\nu = 0.01\,m^2/s$, and $\Gamma_\nu = 1.0\,m^2/s$ located at $(x_0, y_0) = (0.0, 0.0)\,m$ is used, and the particles are initialized in the domain $[-2.5, 2.5]^2$. The choice of the domain ensures that the region where circulation is present is covered, leaving out circulation less than $10^{-14}\,m^2/s$.

Figure 2.7 illustrates the initialization error for the velocity and the vorticity field in two different cases. In the first case (Figure 2.7a), the core size σ is kept constant and equal to $0.02\,m$, while the overlap ratio changes. In the second case (Figure 2.7b), the overlap ratio λ is kept constant and equal to 1.0, while the core size and the particle spacing are changing simultaneously. Figure 2.7a shows that as the overlap goes from 1.0 to 2.0, the initialization errors for all the norms are increasing. This makes sense since higher values of λ mean that the particles are getting apart and losing their overlap. For $\lambda = 1.0$, the vorticity errors are of order $\mathcal{O}(10^{-8})$ and the velocity errors of order $\mathcal{O}(10^{-11})$, which can be considered a satisfying case. Further decrease in the overlap ratio drops the errors close to machine round-off values. It can be concluded that values of overlap $0.8 \leq$

$\lambda \leq 1.0$ produce highly satisfactory results. Reducing the overlap beyond these values increases the computational cost (due to the introduction of more particles) without providing further improvements in accuracy. So this range of values is a good trade-off between the accuracy and the computational cost. Throughout this dissertation, $\lambda = 1.0$ is used in all the simulations.



(a) Constant $\sigma = 0.02\,m$.

(b) Constant $\lambda = 1.0$.

Figure 2.7: Initialization errors for the case of a Lamb-Oseen vortex with $\Gamma_\nu = 1.0\,m^2/s$, $\nu = 0.01\,m^2/s$ and $\tau = 4.0\,s$, using the VPM solver.

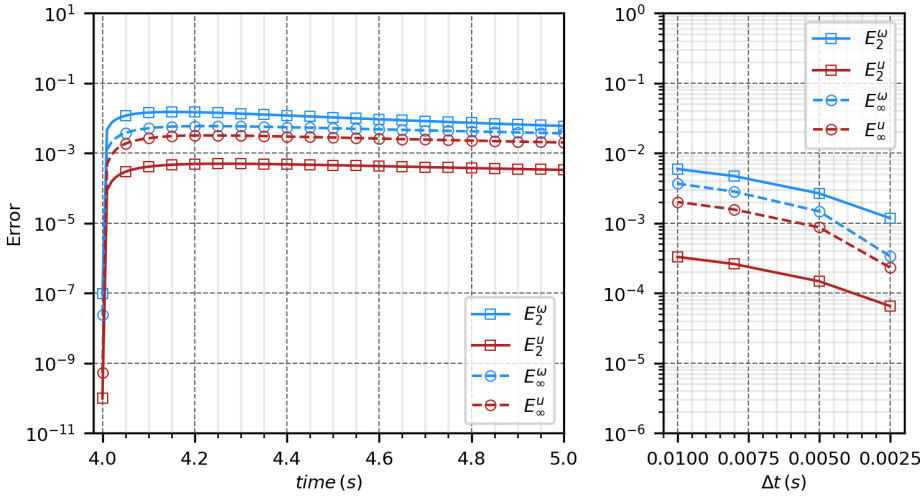
Then, keeping the overlap ratio constant at 1.0, the values of h and σ are modified simultaneously. Lower values mean better resolution. In the semilog plot of Figure 2.7b, the errors show to drop in a straight line from $h = \sigma = 0.2\,m$ approximately up to $h = \sigma = 0.05\,m$, and then the vorticity error is shown to bound to values of order $\mathcal{O}(10^{-8})$, while the velocity errors decrease further approaching machine round-off errors. These lines in a logarithmic plot would not be straight lines, showing that the initialization error is not linear. In the simulations presented throughout this dissertation, different values are used depending on the case being simulated.

2.5.2. Effect of the time evolution and redistribution

With the order of the initialization errors known, the next step is to measure the errors after evolving the solution over time. For this test, the Lamb-Oseen vortex case is used again, but with different parameters. The previous case was highly viscous and initially diffused, which significantly increased the computational cost. Furthermore, due to the time-step condition of the diffusion process (see Equation 2.29), extremely small time-steps were required. For these reasons, a vortex with $\Gamma_\nu = 1.0\,m^2/s$, $\tau = 0.25\,s$, $\nu = 0.005\,m^2/s$, located at $(x_0, y_0) = (0.0, 0.0)\,m$ is used. The particles are initialized in the

domain $[-0.6, 0.6]^2$, with $h = \sigma = 0.01\,m$.

Figure 2.8a shows the time history error of the velocity and vorticity fields from the initial time $t = \tau$ up to $t = \tau + 1.0\,s$. It can be seen that the initial errors for the velocity and the vorticity are of orders $\mathcal{O}(10^{-10})$ and $\mathcal{O}(10^{-7})$, respectively. From the very first time-step, the errors increase by about 7 orders of magnitude for the velocity and 5 orders of magnitude for the vorticity. This phenomenon has been identified in other studies [24], and it occurs because of the redistribution.



(a) Time history errors.

(b) Time-step convergence.

Figure 2.8: Time history errors and time-step convergence for the case of a Lamb-Oseen vortex with $\Gamma_\nu = 1.0\,m^2/s$, $\nu = 0.005\,m^2/s$ and $\tau = 0.25\,s$, using the VPM solver with $h = \sigma = 0.01\,m$.

The first redistribution introduces a significant error in the fields, but it does not increase further in subsequent redistribution processes. Redistribution is a crucial step in the present VPM because, without it, particles would lose their overlap (see Figure 2.3), which would increase errors, particularly localized errors, and could potentially cause the solution to diverge. Therefore, although the initial redistribution error is present, it is manageable as it does not escalate further.

Figure 2.9 shows the initial and final local E_2^ω errors of the simulation in logarithmic scale. It is shown that the highest errors are present around the core of the vortex, where the highest values and gradients occur. After the redistribution, the highest errors remain in the same regions but are many orders of magnitude higher. Figure 2.8b illustrates the final errors using different time-steps. Reducing the time-step reduces the errors, albeit at a low order, since the errors introduced by the redistribution seem to be dominant.

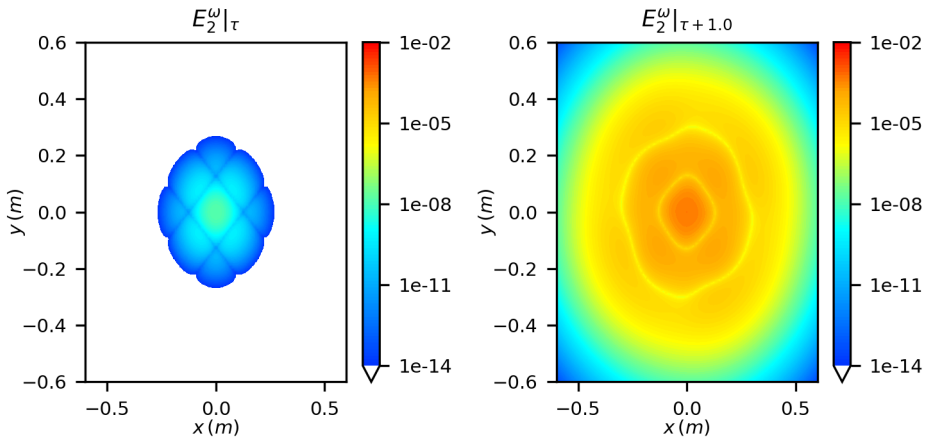


Figure 2.9: Localized E_2^ω error for the initial and the final state of a Lamb-Oseen vortex with $\Gamma_v = 1.0\,m^2/s$, $\nu = 0.005\,m^2/s$ and $\tau = 0.25\,s$, using the VPM solver with $h = \sigma = 0.01\,m$.

2.6. Conclusions

The accuracy and convergence of the VPM are highly influenced by initialization parameters and the redistribution process. Overlap ratios between 0.8 and 1.0 were found to provide optimal results, minimizing velocity and vorticity errors. While smaller timesteps marginally improved accuracy (Figure 2.8b), redistribution errors remained the dominant source of error throughout the simulations. Despite this, the method demonstrated robustness and stability, and, along with all the advantages discussed above, VPM can be considered a reliable approach for capturing wake dynamics and preserving vorticity.

3

The Eulerian solver - OpenFOAM

The previous chapter focused on the Lagrangian component of the hybrid solver. This chapter addresses the Eulerian component starting with a brief introduction to the main Eulerian methods currently employed in CFD. Subsequently, the focus shifts to the FVM, which is explored in greater detail. The chapter discusses both the advantages and the disadvantages of the method and then proceeds to describe OpenFOAM, the software utilized in the context of this dissertation. The chapter continues with detailed discussions on the modifications implemented in OpenFOAM to facilitate the coupling and concludes with a convergence study.

Parts of this chapter have been published in:

- **R. Pasolari**, C.J. Ferreira, and A. van Zuijlen, Coupling of OpenFOAM with a Lagrangian vortex particle method for external aerodynamic simulations, Physics of Fluids, vol. 35, no. 10, p. 107 115, Oct. 2023, ISSN: 1070-6631. DOI: 10.1063/5.0165878. [22]
- **R. Pasolari**, C.J. Ferreira, A. van Zuijlen and C.F. Baptista, Dynamic Mesh Simulations in OpenFOAM: A Hybrid Eulerian-Lagrangian Approach, Fluids 9, 51, (2024), DOI: 0.3390/fluids9020051. [40]

3.1. Introduction

Eulerian solvers are the most popular choice in CFD. The Eulerian framework entails the computation of fluid properties at specific locations within a given space as the fluid flows through it. Various methods concerning discretization techniques are employed in the Eulerian description. The Finite Difference Method (FDM) is considered the oldest method [41] used in CFD. It replaces the partial derivatives of the governing equations with finite difference expressions derived from Taylor series expansions [42]. The method is primarily suited for structured meshes. The Finite Element Method (FEM) employs piecewise polynomial functions on finite elements to describe the variation of the unknown flow fields [42]. It introduces the concept of weighted residuals to measure the errors of the approximate functions. FEM is preferred over FDM due to its ability to handle complex geometries. The Spectral Element Method (SEM) differentiates itself from the others by employing global approximations through truncated Fourier series or Chebyshev polynomial series [42]. The SEM uses high-degree piecewise polynomials. Lastly, the Lattice Boltzmann Method (LBM) offers a distinct approach within the Eulerian framework. Unlike methods that use macroscopic equations, LBM is based on microscopic models and mesoscopic kinetic equations [43]. The fundamental idea behind the method is to construct simplified kinetic models that encompass the essential physics of micro- and mesoscopic processes so that the macroscopic averaged properties obey the desired macroscopic equations.

The FVM was intentionally left to be described last since it is the method employed in this work. FVM converts the Partial Differential Equations (PDEs), which encapsulates conservation laws across differential volumes, into discrete algebraic equations applicable over finite volumes. This conversion process begins with the geometric domain being discretized into finite, non-overlapping volumes. Then, the PDEs are transformed into algebraic equations by integrating them over each discrete element. FVM ensures strict conservation of mass, momentum, and energy by evaluating face fluxes at the finite volume boundaries. It mandates that the fluxes entering a volume must precisely match the fluxes exiting the adjacent volumes, ensuring the conservation of the properties. This method excels in handling unstructured meshes, showcasing its adaptability and efficiency in various computational domains.

3.2. Advantages and limitations

Eulerian solvers have been widely used in the field of CFD, in numerous applications. An application of the FDM can be found in [2]. Similarly, applications of the FEM are documented in [6, 7]. References to the SEM are found in [8, 44], while applications of the LBM are highlighted in [45]. Furthermore, the FVM is extensively discussed in [3–5], showcasing its significance and versatility in CFD research. In general, these methods have shown great performance when they resolve regions close to solid boundaries including the boundary layer. They are capable of capturing the viscous phenomena and vorticity generation that takes place in the vicinity of the solid structure in a very efficient way, taking advantage of the anisotropy of their elements. However, their dissipative nature, especially in regions where the mesh is sparse, is a bottleneck that cannot be easily overcome. Using higher-resolution meshes would increase the computational cost,

making the method inefficient and the use of powerful hardware and parallel running essential. Moreover, explicit Eulerian methods impose strict restrictions on the time-step because of the advective terms (CFL condition [9]). In addition, while imposing far-field boundary conditions is straightforward, the primary limitation lies in the need for large computational domains. This requirement increases computational cost, even when the region of interest is localized near the bluff body. Unstructured meshes can partially address this by allowing a coarser mesh in the outer regions; however, this approach falls back to the problem of artificial diffusion in those coarse mesh areas. One potential approach to address the challenge of the high computational cost involves the implementation of AMR techniques, whereby the mesh undergoes dynamic adaptation based on the simulation requirements through localized mesh refinement or coarsening [10].

3.3. Discretization in Finite Volume Method

As already mentioned, FVM is the method employed in the context of this dissertation, so the focus is limited to this method. Here, the discretization process used in FVM, as implemented in OpenFOAM, is presented. Figure 3.1 illustrates a 3D cuboid finite volume. This control volume features a central point where variables are computed (collocated grid) and six faces where fluxes are determined. The central node is denoted by the letter P (for polar), with adjacent nodes labeled N (north) at the top, S (south) at the bottom, E (east) on the right, W (west) on the left, and T (front) and B (back) representing the front and back nodes, respectively. The nodes are designated with uppercase letters, each of which is associated with a lowercase letter that identifies the corresponding face of the cell. It is important to note that this notation and configuration assume a perfectly cuboidal control volume; for cells of different shapes or in unstructured meshes, the geometric relationships vary.

To keep things simple, this section focuses solely on the discretization of the diffusion term in a 3D volume. For a complete discretization of all terms and additional details on the schemes used, please refer to Appendix B.

Let ϕ be a variable of the flow (e.g., velocity, temperature, etc.). The diffusion term of the N-S equations for this variable, after integrating over the control volume, is the Left Hand Side (LHS) of Equation 3.1. The volume integral is then transformed into a surface integral using the Gauss Theorem, as shown on the Right Hand Side (RHS) of Equation 3.1.

$$\iiint_V \left(\nabla \cdot (\nu \nabla \phi)\right) dV \underset{\text{Theorem}}{\overset{\text{Gauss}}{=}} \oiint_S \nu \nabla \phi \cdot \hat{\mathbf{n}} dS \tag{3.1}$$

The integral can be transformed into a summation over the faces around the control volume:

$$\oiint_S \nu \nabla \phi \cdot \hat{\mathbf{n}} dS \approx \sum_f \nu_f \nabla \phi_f \cdot \mathbf{S}_f =$$

$$= \nu_n \nabla \phi_n \cdot \mathbf{S}_n + \nu_s \nabla \phi_s \cdot \mathbf{S}_s + \nu_w \nabla \phi_w \cdot \mathbf{S}_w + \nu_e \nabla \phi_e \cdot \mathbf{S}_e + \nu_b \nabla \phi_b \cdot \mathbf{S}_b + \nu_t \nabla \phi_t \cdot \mathbf{S}_t$$

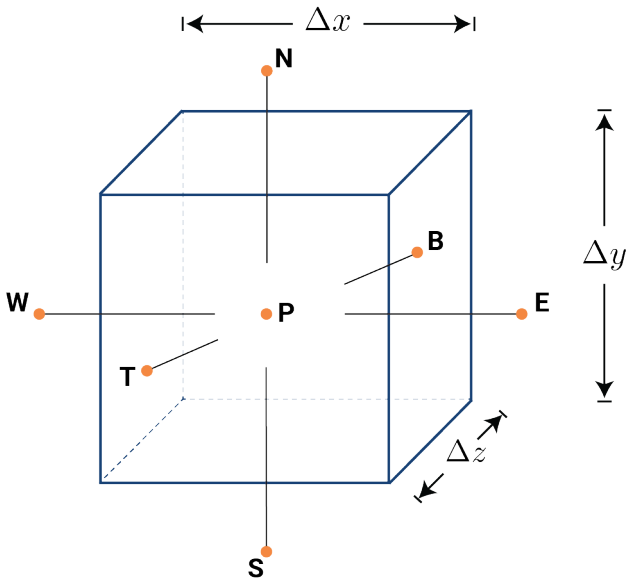


Figure 3.1: A cuboid finite volume. The cuboid has a central point (denoted by the letter P), where the flow fields are computed, and six faces where the fluxes are determined. The six central nodes of the adjacent cells are also illustrated, denoted by the letter of their direction: N (north) at the top, S (south) at the bottom, E (east) on the right, W (west) on the left, T (front) and B (back).

Now the values of the gradient and the diffusion coefficient (here the kinematic viscosity ν) should be transferred from the faces to the cell center. For the diffusion coefficient, the most common scheme used is *linear*, so the coefficient is calculated as an interpolation between the cell P and the cell adjacent to the face. For example, the value of ν_n is approximated as:

$$\nu_n = \frac{\nu_P + \nu_N}{2}$$

This averaging assumes that the cells P and N are of equal size and symmetrically arranged. If the grid is non-uniform or the cells differ in size, a proper linear interpolation weighted by the distances to the face from each cell center should be used. The gradient is calculated using the surface normal gradient schemes, denoted as *snGradSchemes* in OpenFOAM. When the face is orthogonal, the gradient is calculated as:

$$\nabla\phi_n = \frac{\phi_N - \phi_P}{|\mathbf{d}|}\hat{\mathbf{n}}_f$$

where $\mathbf{d} = \mathbf{x}_N - \mathbf{x}_P$ is the vector between the centers of the neighboring cells P and N, and $\hat{\mathbf{n}}_f$ is the unit normal vector to the face. This approximation is valid when the face is orthogonal, i.e., when $\mathbf{d}$ is aligned with the face normal.

When the face is not orthogonal, i.e., when the vector connecting the centers of adjacent control volumes is not aligned with the face normal, a non-orthogonal correction must be applied. In OpenFOAM, the surface-normal gradient is typically decomposed

into two components. The first is the orthogonal contribution, while the second accounts for the non-orthogonality between the face normal and the vector connecting the neighboring cell centers. OpenFOAM implements several non-orthogonal correction strategies which control the accuracy and stability of the scheme depending on the mesh quality. For more information please refer to the book by Greenshields et al. [46].

3.4. OpenFOAM

In the scope of this dissertation, the OpenFOAM software is employed. OpenFOAM is a widely used and highly respected open-source software for CFD, which employs the FVM. The holder of the copyrights of OpenFOAM is *The OpenFOAM Foundation* [47], whose purpose is to manage and distribute OpenFOAM as free, open-source software for the benefit of its users. *The OpenFOAM Foundation* releases an official version of OpenFOAM annually, using a numerical and sequential versioning system (with OpenFOAM v12 being the most recent today), recognized for its stability and robustness, and an extended version, referred to as *dev* version, which includes additional features not yet part of the standard release. *ESI-OpenCFD* [48] also has a license to distribute OpenFOAM, releasing two main versions each year. These versions are identified by the year and the version number, with the most recent being v2412 as of the day this dissertation is written. The version of OpenFOAM used for the development of the hybrid solver is OpenFOAM v9 [49] from *The OpenFOAM Foundation*.

OpenFOAM has been chosen among numerous software options available for several reasons. Specifically:

- OpenFOAM is open-source and customizable. Its open-source nature makes it a perfect option for coupling with in-house solvers (and other open-source software), allowing for seamless integration and modification to meet specific research requirements. As an open-source software, OpenFOAM aligns with the principles of open science, promoting transparency, reproducibility, and collaborative research.
- It comes with a wide range of features, including a vast library for turbulence modeling, dynamic mesh modeling, heat transfer, etc. This spares researchers the effort of re-implementing these features from scratch. When coupling with other in-house solvers, these features remain accessible, often requiring only minor modifications.
- It has a strong track record in academia and industry. The software has been widely used and validated across a broad range of applications, providing confidence and reliability to users.
- OpenFOAM is continuously developed and improved. With at least three versions released annually, OpenFOAM remains at the cutting edge of CFD technology.
- It is highly versatile and scalable, meaning it can run on a range of hardware, from a laptop to large-scale High-Performance Computing (HPC) clusters.

These factors make OpenFOAM an appealing choice for both academia and industry, which is why it was selected for this project. The aim was to create a tool that would

interest a large community and be familiar, or at least provide substantial resources for learning and troubleshooting within the software. Figure 3.2 illustrates a survey conducted by *Resolved Analytics* [50] among 624 CFD users. It shows that OpenFOAM is one of the most used tools in the field, with 12% of respondents using it. At the same time, *Enlyft* [51] mentions in its database 150 companies that use OpenFOAM, with 71% being large companies (more than 1,000 employees and over 1,000 million dollars in revenue). This statistic indicates that OpenFOAM is not only competitive in the academic sphere but also holds a significant standing in the industrial market.

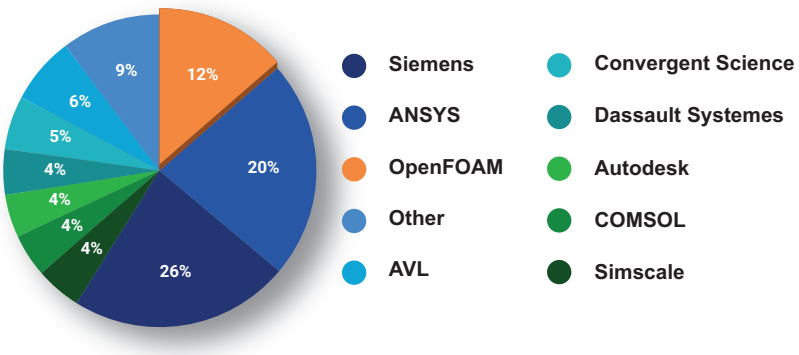


Figure 3.2: A pie chart with the percentage of users of the different CFD software. OpenFOAM is the third in order software with the most users according to the survey. The pie has been reproduced from Reference [50].

3.5. The modified OpenFOAM solver

The implementation of the finite volume method in OpenFOAM has been extensively covered in bibliography [46, 52]. As the basis of the hybrid solver developed here, *pimpleFoam* [53] is used, an inherent OpenFOAM solver that is designed to solve transient, incompressible, and turbulent (and laminar) flows. It uses the PIMPLE loop for the correction of the velocity and pressure fields and has been extensively used in many applications [54, 55].

In OpenFOAM v9, the solvers are applications, meaning that when a user invokes a solver (e.g., *pimpleFoam*), a while loop is triggered that runs from the initial time to the simulation's end time. The details of this loop's formulation can be found in Appendix C. However, this particular formulation is not very suitable for accommodating the coupling of OpenFOAM with an additional solver, as is the case here.

In this hybrid approach, coupling steps must occur immediately before and after each Eulerian step to establish the necessary coupling. Moreover, in the context of a moving mesh, actions need to be taken in the interim. This is because, once the mesh has moved, boundary conditions must be recalculated for the updated coordinates of the boundary faces.

Various coupling techniques are available, with one effective option being *preCICE* [56], which facilitates efficient, numerically stable coupling between different solvers by

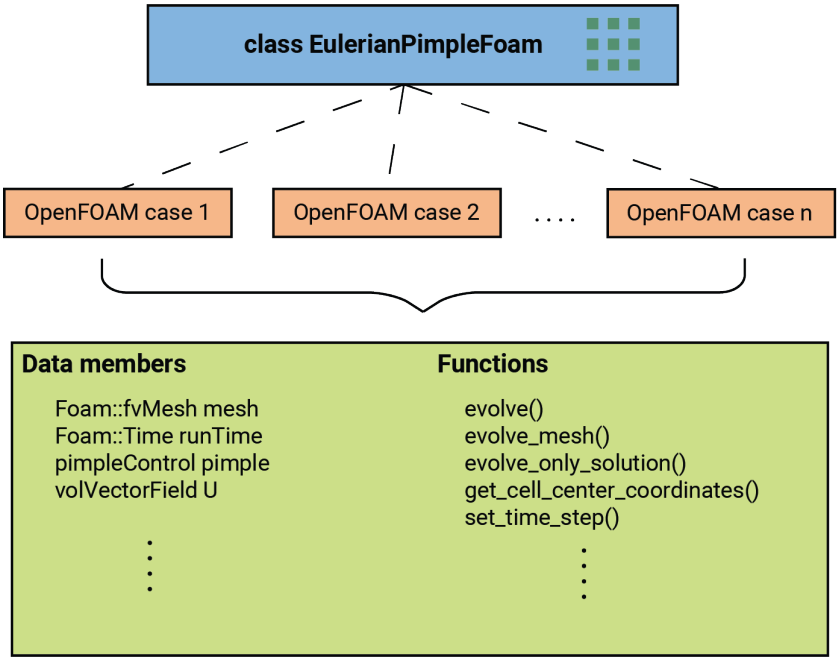


Figure 3.3: The custom *EulerianPimpleFoam* class' structure.

enabling data exchange in partitioned multi-physics simulations. However, a different approach has been chosen here. Specifically, the original structure of *pimpleFoam* was modified, resulting in the creation of a new class-based solver called *EulerianPimpleFoam*[1]. All the processes within the Eulerian solver are now encapsulated as member functions of this class, providing enhanced control and coordination. In the hybrid solver, at any given time, the Eulerian component can halt its operations, export data, receive new inputs, and establish efficient communication with the Lagrangian solver. This structure facilitates a more organized and flexible interaction between the Eulerian and Lagrangian components. Additionally, the modified *EulerianPimpleFoam* solver can operate independently of the hybrid setup, making it versatile for various Eulerian-only simulations. The solver's structure is visually depicted in Figure 3.3.

Another noteworthy advantage of this solver's specific structure is its ability to handle multiple OpenFOAM cases simultaneously. Each OpenFOAM case can be instantiated as an object of the class and executed independently. This capability enables the simulation of multibody scenarios, with Lagrangian particles facilitating interconnections between various Eulerian regions.

It should be noted here that newer versions of OpenFOAM (v11 and later) have implemented modular solvers that provide similar flexibility, replacing traditional solvers like *pimpleFoam* with structures such as *incompressibleFluid*. These updates enable more

[1]The original code was developed by Carlos F. Baptista (personal communication). Significant modifications have been made to this work.

versatile solver functionality, facilitating multi-physics coupling in a way that parallels the custom modularity introduced in this work. However, this flexibility was not available in earlier versions of OpenFOAM, including OpenFOAM v4, which was used by Carlos F. Baptista, and OpenFOAM v9, which was used in this dissertation. These earlier versions necessitated custom development to achieve the modularity and solver flexibility described here.

3.6. Convergence study

The convergence of the OpenFOAM solver will be assessed using the Lamb-Oseen vortex case [39]. The analytical solutions for the velocity and vorticity fields induced by the vortex were presented in Equation 2.31. A square computational domain with length L is used, as shown in Figure 3.4.

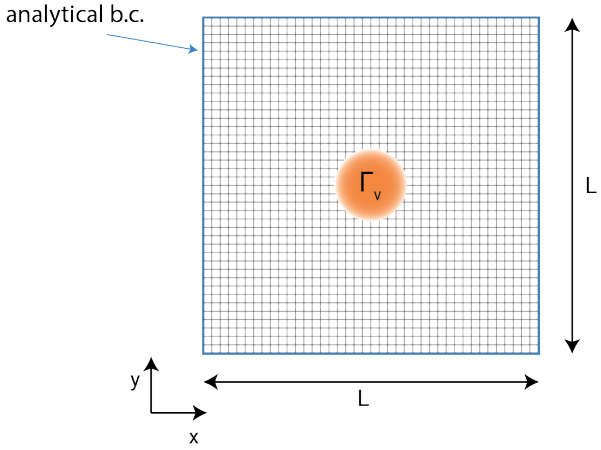


Figure 3.4: The configuration of the stationary Lamb-Oseen vortex in an unbounded domain, using OpenFOAM.

The analytical expression for the velocity field is used for the initialization of the velocity field, as well as for the velocity boundary conditions at every time-step, while the *fixedFluxPressure* boundary condition is used for the pressure. The parameters common to all simulations in this convergence study are summarized in Table 3.1, while the discretization schemes are summarized in Table 3.2.

3.6.1. Spatial convergence

For the spatial convergence, eight different mesh resolutions are used. From sparse to dense mesh, the cell size h_{cell} is reduced by a factor of 2, but as it gets smaller, a factor of $\sqrt{2}$ is used. The time-step is chosen to be sufficiently small, so in every case, the CFL number remains small enough (the maximum CFL number for the densest mesh is 0.025), ensuring that the influence of the temporal discretization does not affect the solution. Figure 3.5 illustrates the errors at the initial time and after 1.0 s. Specifically, the E_2 and E_∞ norms are calculated. These errors are computed as shown in Equation 3.2,

Table 3.1: Simulation parameters for the case of a stationary Lamb-Oseen vortex in an unbounded domain, using OpenFOAM.

Parameter	Symbol	Value	Dimension
Vortex strength	Γ_v	-0.05	m^2/s
Initial position	(x_0, y_0)	$(0.5, 0.5)$	m
Freestream velocity	(U_x^{inf}, U_y^{inf})	$(0.0, 0.0)$	m/s
Lamb-Oseen time constant	τ	4.0	s
Kinematic viscosity	ν	5×10^{-4}	m^2/s
Simulation time	t_{sim}	1.0	s
Length	L	1.0	m

Table 3.2: Discretization schemes for the simulations of the stationary Lamb-Oseen vortex in unbounded domain, using OpenFOAM.

Terms	OpenFOAM notation	Scheme	Order
Temporal terms	*ddtSchemes*	*backward*	2^{nd}
Gradient terms	*gradSchemes*	*Gauss linear*	2^{nd}
Divergence terms	*divSchemes*	*Gauss linearUpwind*	2^{nd}
Laplacian terms	*laplacianSchemes*	*Gauss linear orthogonal*	2^{nd}
Interpolations	*interpolationSchemes*	*linear*	2^{nd}
Surface Gradients	*snGradSchemes*	*orthogonal*	2^{nd}

using the strength of the vortex Γ_v, the initial radius of the vortex $R_v = \sqrt{2.0\nu\tau}$, and the surface area of the cell h_{cell}^2 for normalization (square cells are used here):

$$E_2^u = \frac{1}{\Gamma_v}\sqrt{\sum_i \left[(u_{E,i} - u_{A,i})h_{cell}\right]^2 + \sum_j \left[(v_{E,j} - v_{A,j})h_{cell}\right]^2} \tag{3.2a}$$

$$E_\infty^u = \frac{R_v}{\Gamma_v}\left[\max(|u_E - u_A|) + \max|v_E - v_A|\right] \tag{3.2b}$$

$$E_2^\omega = \frac{R_v}{\Gamma_v}\sqrt{\sum_i \left[(\omega_{E,i} - \omega_{A,i})h_{cell}\right]^2} \tag{3.2c}$$

$$E_\infty^\omega = \frac{R_v^2}{\Gamma_v}\max(|\omega_E - \omega_A|) \tag{3.2d}$$

Here, E and A correspond to the Eulerian and analytical solutions, respectively.

Along with the errors in Figure 3.5, a second-order line is plotted. As can be seen, the initial velocity error is zero because the velocity field was initialized using the analytical solution. However, the vorticity field error is finite due to the curl operation ($\nabla\times$) calculated in OpenFOAM. Since second-order schemes are used for gradients and interpolation, it is evident that after the second mesh, the vorticity errors reach the asymptotic range, which is second-order as expected. On the right side of the figure, the errors at

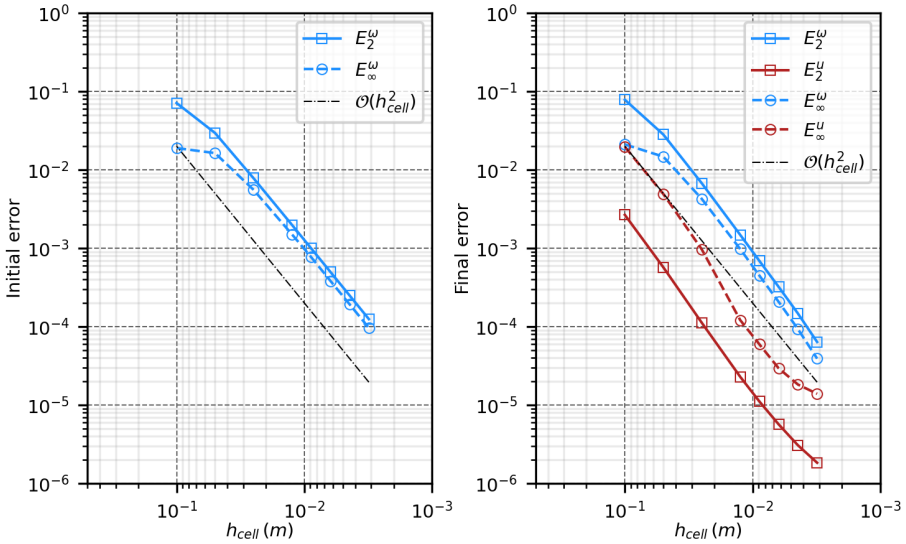


Figure 3.5: Spatial convergence study for the case of a stationary Lamb-Oseen vortex in an unbounded domain, using OpenFOAM.

the end of the simulation are presented. Here, the velocity error is also present but lower than the vorticity error. The convergence rate here is close to second-order, especially for the E_2^ω,E_∞^ω and E_2^u. Achieving the exact convergence rate in such cases, where analytical boundary conditions are used and the computational domain is very small, is challenging. The boundary conditions applied to the boundary of the Eulerian field are highly dependent on the number of boundary faces. A single value is calculated as a boundary condition and applied across the face, whereas, in reality, this should be a distribution. A more accurate approach would be to integrate the velocity over the face and divide it by its length to find a better approximation of the boundary condition. This can lead to nonlinearities in the convergence rates.

3.6.2. Temporal convergence

In a similar fashion as the spatial convergence study, the temporal convergence study is performed. A dense mesh ($h_{cell} = 0.015625\,m$) is chosen to ensure the solution is independent of the spatial resolution. Five different time-steps are used, and the results of the temporal convergence study are shown in Figure 3.6. Of course, since the same mesh is used across different simulations, the initial vorticity error is constant, as expected.

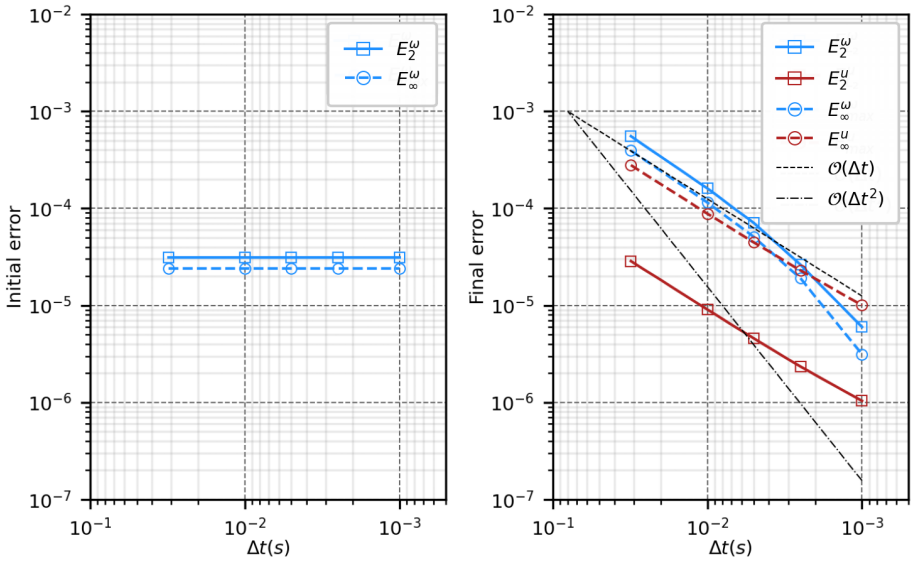


Figure 3.6: Temporal convergence study for the case of a stationary Lamb-Oseen vortex in an unbounded domain, using OpenFOAM.

3.7. Conclusions

The convergence study for the *EulerianPimpleFoam* solver demonstrated nearly second-order spatial convergence, as expected from the second-order schemes used. After the second mesh refinement, the errors in both velocity and vorticity fields aligned well with the theoretical accuracy, with some small deviation for the velocity.

For temporal convergence, the rate was between first- and second-order. This discrepancy is due to the use of a very small computational domain with analytical boundary conditions. Applying these conditions to such small domains can introduce nonlinearities, making it difficult to achieve exactly the accuracy of the schemes.

4

The coupled solver - *VPMFoam*

The previous two chapters presented the components of the hybrid solver. This chapter begins the discussion on the developed hybrid solver. It starts with a literature review on hybrid solvers, followed by a presentation of the state-of-the-art VPMFoam solver and the fundamental concepts applied. Following this, the coupling process is examined in detail, with an elaboration on each step involved. A flowchart illustrating the general application of the code is both discussed and presented graphically. The chapter concludes by enumerating the various software techniques and packages used in the solver's process and optimization, providing a comprehensive overview of the tools and methodologies utilized.

Parts of this chapter have been published in:

- **R. Pasolari**, C.J. Ferreira, and A. van Zuijlen, Coupling of OpenFOAM with a Lagrangian vortex particle method for external aerodynamic simulations, Physics of Fluids, vol. 35, no. 10, p. 107 115, Oct. 2023, ISSN: 1070-6631. DOI: 10.1063/5.0165878. [22]
- **R. Pasolari**, C.J. Ferreira, A. van Zuijlen and C.F. Baptista, Dynamic Mesh Simulations in OpenFOAM: A Hybrid Eulerian-Lagrangian Approach, Fluids 9, 51, (2024), DOI: 0.3390/fluids9020051. [40]

4.1. Literature review

Researchers, having identified the advantages and limitations of both the Eulerian and Lagrangian solvers presented in Chapter 2 and Chapter 3, considered how to exploit the benefits of both while mitigating their limitations. They recognized the value of tools that can accurately resolve boundary layers close to solid bodies and wakes with high precision without introducing artificial diffusion while also limiting computational costs. Consequently, they began exploring methods to couple mesh-based and mesh-free solvers, aiming to leverage the advantages of each while minimizing their disadvantages. This led to the development of hybrid methods.

The first study to propose a hybrid Eulerian-Lagrangian solver in this context was the pioneering research by Christiansen [57], which is considered the first Vortex-In-Cell (VIC) method. In this study, different parts of the governing equations are solved either on the mesh or by the vortex particles. Specifically, the advection part is solved by the particles, while the diffusion and vortex stretching are solved on the mesh. The properties are transferred between the two solvers using Particle to Mesh (P2M) and Mesh to Particles (M2P) interpolations. The main problem with this method was the distortion of the vortex particles, as discussed in Chapter 2. After the introduction of remeshing schemes, the VIC method was revitalized and began to be referred to as the Vortex Particle-Mesh method or Remeshed Vortex Method (RVM) [14]. The most demanding step of this method is the calculation of the velocity field induced by the vortex elements using the equation:

$$\nabla^2 \mathbf{u} = -\nabla \times \omega \tag{4.1}$$

The equation is typically solved using either the FMM or Fast Fourier Transform (FFT) methods. As Mimeau et al. [14] state, a distinguishing characteristic of different VIC methods is the frequency at which the particles are remeshed. Some methods remesh the particles after a few time-steps (e.g., every 5 time-steps [58]), while others choose to remesh every single time-step [59], leading to the so-called Semi-Lagrangian Vortex methods.

Another well-known hybrid method was introduced by Cottet [60] and is called the Domain Decomposition Method (DDM). The key concept behind this method is that the computational domain can be decomposed into Eulerian and Lagrangian parts, allowing the mesh-based and mesh-free solvers to be used in different regions. Specifically, the Eulerian solver is used close to solid boundaries to capture viscous phenomena and vorticity generation due to its accuracy in resolving boundary layers, while the Lagrangian solver handles the rest of the domain, efficiently evolving the solution with minimal numerical diffusion introduced into the flow. Initially, the method used an overlapping zone between the Eulerian and Lagrangian subdomains, employing a Schwarz iterative method at every time-step to ensure the stream functions of the Eulerian and Lagrangian subdomains matched completely within this overlap region. However, Daeninck [61] later demonstrated that the two solvers could be coupled without the computationally costly iterative process. Instead, the Lagrangian solver resolves the entire computational domain up to the solid boundary, without focusing on accuracy in this region, as the Eulerian solver already provides an accurate solution there. The solution is then transferred to the Lagrangian field using M2P interpolations.

Since the initial introduction of these hybrid solvers, they have gained increased popularity, especially in the field of external aerodynamics. The RVM and the DDM, as well as combinations of these methods, have evolved and been applied to many applications. Numerous research teams have developed their own solvers using these methods or combinations thereof, with some of them presented in the references [62–70]. Shi et al. [62] coupled an Eulerian solver with a wake solver by transferring the Eulerian solution to the Lagrangian domain and calculating sectional lift coefficients on the body using the Kutta-Jouksowksi theorem. In this specific solver, Fluid-Structure Interaction (FSI) methods are also included. Stock et al. [63] coupled a Vortex Particle-Mesh solver with OVERFLOW, a fully compressible solver, and later [64] coupled a high-order spectral FDM with an open-source VPM. Palha et al. [31] coupled the FEM FEniCS software [71] with a VPM solver. Billuart et al. [65] developed a weak coupling approach between a FDM solver (and also a vertex-centered FVM) and a Vortex Particle-Mesh method in 2D, demonstrating that the total circulation can be conserved without using any boundary conditions for the Lagrangian solver. Papadakis et al. [66] developed a strongly coupled compressible Eulerian-Lagrangian solver and used it for external compressible flows and flows including FSI [72].

4.2. State-of-the-art

As discussed earlier, this dissertation presents the development of a two-dimensional incompressible hybrid Eulerian-Lagrangian solver. Building on the work of Palha et al. [31], this solver advances the capabilities of the original one and also replaces the FEM solver with the FVM solver, as introduced in Chapter 3. The name of the new solver is *VPMFoam*, which stands for "Vortex Particle Method Foam", since it is the result of coupling a VPM solver with the OpenFOAM framework, while the original solver was known as pHyFlow [31].

In contrast to the works of Stock [63, 64], which utilized finite differences and spectral elements for the Eulerian part of the hybrid solver, *VPMFoam* employs a FVM solver. Similarly, while Papadakis et al. [66] developed a compressible solver, the present implementation adopts an incompressible approach. Additionally, compared to the work of Billuart et al. [65], where finite differences and vertex-centered finite volumes were used, *VPMFoam* incorporates a cell-centered FVM solver. This modification allows for exploring the performance of a cell-centered FVM in the context of a hybrid solver and assessing its effectiveness.

Furthermore, the use of OpenFOAM for the Eulerian solver plays a crucial role. As discussed in Chapter 3, OpenFOAM is an open-source software widely respected in both academia and industry, having been rigorously tested across various applications and offering a broad range of features. OpenFOAM's flexibility allows for modifications and seamless integration with other software and in-house codes. The objective of this work is to develop an alternative to existing OpenFOAM solvers that adopt a different approach, deliver reliable results, and reduce computational effort. While hybrid solvers are not intended to fully replace existing solvers, they can, depending on the application, achieve accurate results more efficiently.

The solver is applied to complex flow phenomena, such as the dynamic and static stall of airfoils, providing significant validation in challenging scenarios. These complex

cases demonstrate the solver's robustness and capability to handle intricate flow dynamics, reinforcing its potential for reliable performance in real-world applications.

4.3. Methodology and fundamental concepts

VPMFoam is fundamentally a coupling between an Eulerian solver with a Vortex Particle-Mesh solver that employs the DDM, as described by Daeninck [61]. The two component solvers are coupled through a two-way coupling (Figure 4.1) using a weak formulation, meaning the coupling steps are applied only once per time-step. These steps involve calculating boundary conditions for the Eulerian solver and correcting the Lagrangian field near the solid body. The DDM and the detailed processes for coupling the solvers are presented in this section.

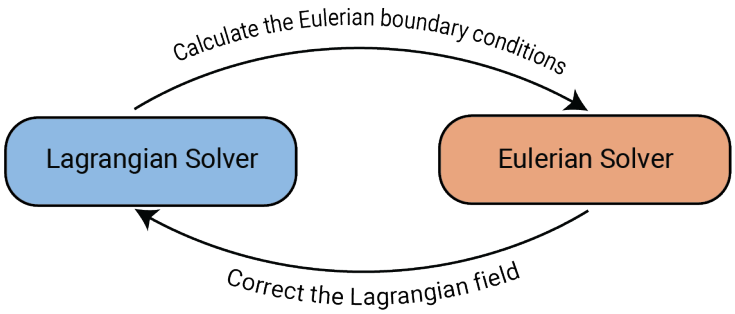


Figure 4.1: Two-way coupling between the Eulerian and the Lagrangian solvers in the context of *VPMFoam*.

4.3.1. Domain decomposition

The decomposition of the computational domain is illustrated in Figure 4.2. The Eulerian solver resolves the region close to the solid body up to the numerical boundary (the outer edge of the Eulerian domain), while the Lagrangian solves the equations of motion for the entire domain. The Lagrangian domain is the entire computational domain, fully overlapping the Eulerian domain.

In this decomposition, the Lagrangian particles do not require an accurate representation of the field near the solid body, as this is handled by the Eulerian solver. However, the Eulerian domain is relatively small and requires appropriate boundary conditions at the numerical boundary.

4.3.2. Calculation of the Eulerian boundary conditions

In OpenFOAM, both velocity and pressure boundary conditions must be specified for the *pimpleFoam* solver. In the hybrid context, the Eulerian domain includes two boundary patches: one at the solid boundary and one at the outer numerical boundary. For the solid boundary, the well-known no-slip condition, or any other appropriate boundary condition, can be applied depending on the specific case. At the outer numerical boundary, since the Lagrangian solution evolves before the Eulerian solution, the evolved Lagrangian particles determine the boundary conditions at the outer edges of the Eulerian

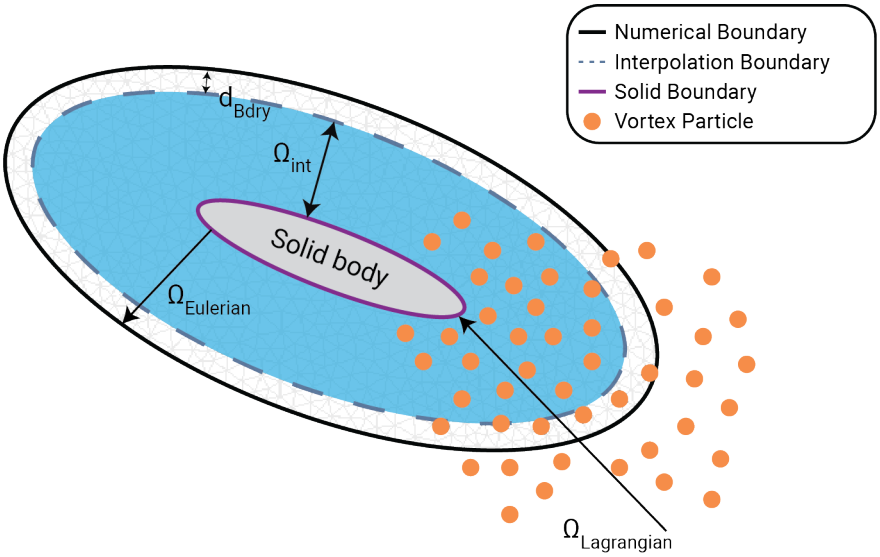


Figure 4.2: The decomposition of the computational domain. The Lagrangian solver covers the entire computational domain, fully overlapping the Eulerian subdomain, which extends from the solid boundary [——] up to the numerical boundary [——]. Interpolations between the Eulerian solution and Lagrangian particles take place within the interpolation domain [ ], which is bounded by the interpolation boundary [– – –].

domain for both velocity and pressure fields. The boundary conditions applied here are as follows:

- Dirichlet boundary conditions for the velocity across the numerical boundary ($u_{n,f}$).
- Neumann boundary conditions for the pressure across the numerical boundary ($\frac{\partial p}{\partial n}|_f$).

The velocity at the faces of the numerical boundary is computed by adding the induced velocity from the particles and the freestream velocity. Meanwhile, the pressure gradient is derived from the N-S equation (Equation 4.2).

$$\nabla p = -\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nu\nabla^2\mathbf{u}\right), \qquad p = p/\rho \tag{4.2}$$

Since the pressure gradient is evaluated at the numerical boundary, which is located farther from the solid body, the influence of viscosity becomes negligible in most cases. Consequently, the flow can often be considered inviscid, allowing the viscous term to be omitted for simplicity. This reduces the equation to:

$$\nabla p = -\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right), \qquad p = p/\rho \tag{4.3}$$

However, in special cases, such as the slipstream of rotating edges where high shear regions exist, the contribution of the viscous term can become significant. In such sce-

narios, neglecting viscosity may lead to inaccurate predictions, and careful consideration is required to ensure appropriate modeling.

Given the induced velocity from all particles, all terms in the RHS can be computed to obtain the pressure gradient (a detailed calculation of the pressure gradient is provided in Appendix E). Then, for use in the boundary condition, the pressure gradient is projected onto the local face-normal direction at the center of each outer boundary face, as illustrated in Figure 4.3.

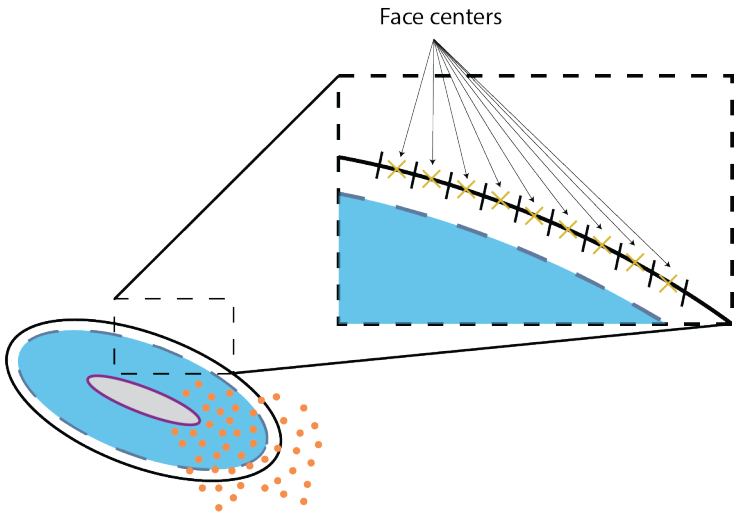


Figure 4.3: The computational domain with a zoomed-in view of the numerical boundary. The highlighted region shows the face centers of the numerical boundary where the velocity and pressure boundary conditions are applied.

As previously discussed, the Lagrangian solver has no strict limit on the time-step, and its accuracy is less dependent on the time-step due to the high-order Runge-Kutta 4 (RK4) integration scheme. Typically, different time-steps are used for the Eulerian and Lagrangian solvers. Specifically, the Lagrangian time-step can be divided into k_E substeps using $\Delta t_E = \frac{\Delta t_L}{k_E}$, where Δt_E and Δt_L are the Eulerian and Lagrangian time-steps, respectively. In this case, the boundary conditions at each substep (t_{int}) are obtained by interpolating between the times t_n and t_{n+1} in the way is presented by Equation 4.4 for the normal velocity $u_{n,f}$ on the face of a boundary cell. The same applies to the pressure boundary condition, as well as to other boundary conditions if needed.

$$u_{n,f}(t_{int}) = u_{n,f}(t_n) + \frac{t_{int} - t_n}{\Delta t_L}[u_{n,f}(t_{n+1}) - u_{n,f}(t_n)] \tag{4.4}$$

OpenFOAM offers a wide range of available boundary conditions that can be used as alternatives. In the context of this dissertation, apart from the fixed pressure gradient boundary condition, the *fixedFluxPressure* boundary condition has been applied, which adjusts the pressure gradient so that the flux on the boundary matches the velocity boundary conditions. Having tested both boundary conditions in the cases presented later, no significant differences were observed. Additional boundary conditions

can be found in the user guide by Greenshields [73], or in the documentation of the *ESI-OpenCFD* [48] version of OpenFOAM [74], but care must be taken to ensure compatibility with *The OpenFOAM Foundation* [47] version.

4.3.3. Correction of the Lagrangian field

The second step in coupling the Lagrangian and Eulerian solvers is the correction of the Lagrangian solution near the solid body. The Lagrangian solver tends to under-resolve the regions close to the aerodynamic body, which introduces errors in the calculation of the Eulerian boundary conditions. To address this issue, the solution of the Eulerian field in this region at the same time level is used to correct the strength of the particles. This correction is performed only when the Eulerian time coincides with the Lagrangian time. At the end of this correction step, the solutions from both solvers should be approximately the same.

The region where this correction takes place, referred to as the interpolation region (Ω_{int} in Figure 4.2), is part of the Eulerian subdomain but excludes a thin layer of cells (around $2-3$ cells) near the numerical boundary. This area is excluded because accurately representing the velocity gradient is challenging near boundaries, which could amplify errors in the computation of vorticity [66]. The principle behind this method is to remove all the particles located inside the interpolation region and create new particles that reproduce the Eulerian vorticity field.

Initially, this was achieved using the Eulerian vorticity field (see Figure 4.4), and the steps of the process are outlined as follows:

1. The particles residing within the interpolation region are identified and removed.
2. New particles are created on the background Lagrangian grid, which is the same grid used for particle redistribution.
3. The vorticity field from the Eulerian solution is exported and interpolated from the cell centers to the particles (here interpolation methods from the *SciPy* [75] library were used). The order of the interpolation method is user-defined, and different methods can be applied in each case (e.g., nearest neighbor, linear, cubic).
4. The strength of the new particles is calculated using the formula $\Gamma_p = \omega_p \cdot h^2$, where h is the nominal separation between particles, ω_p is the vorticity at the particles center, and Γ_p is the assigned circulation of the particle.

Although this method produced good results in cases where the aerodynamic object was static, the results were less satisfactory when simulating moving objects. After reviewing the literature, it was concluded that to accurately represent the vorticity field and conserve the circulation of the flow, it is essential to properly capture the induced fields from the surface of the body. Various approaches have been utilized, such as the vortex panel method used by Palha et al. [31], or the surface elements used in the work of Papadakis et al. [66].

A more recent approach was introduced by Billuart et al. [65], where circulation was conserved up to interpolation accuracy without using panels or surface vortices. The correction process is similar to the one already presented, up to the point where new

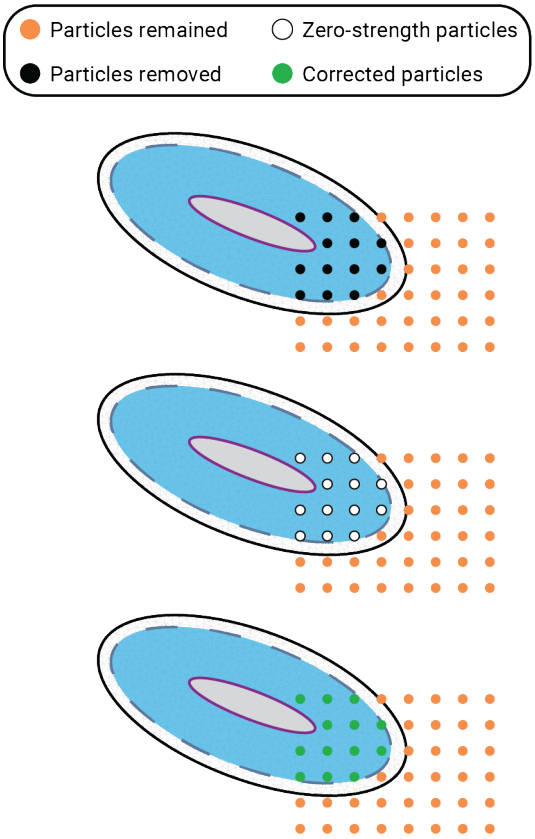


Figure 4.4: The process of correcting the Lagrangian particles inside the interpolation region. First, the particles residing in this region are identified. Then they are removed, and new particles with zero strength are created on the background Lagrangian grid. Finally, strength values are assigned to the new particles using the vorticity exported by the Eulerian solution.

strengths are assigned to the particles. However, instead of using the vorticity field to calculate the particle strengths, the velocity field is employed. For each particle within the interpolation region, the velocity is computed from the Eulerian solution at the four edges of its position on the background Lagrangian grid (see Figure 4.5). The circulation is then obtained by integrating around this closed loop:

$$\Gamma_p \approx \omega_p \cdot h^2 = \oint_L \mathbf{u} \cdot d\mathbf{l} = h\left[\left(v_{i+1/2,j} - v_{i-1/2,j}\right) - \left(u_{i,j+1/2} - u_{i,j-1/2}\right)\right] \tag{4.5}$$

In this method, if an edge lies inside the solid body, the velocity of the body is assigned to that edge (if the body is stationary, a zero velocity is assigned).

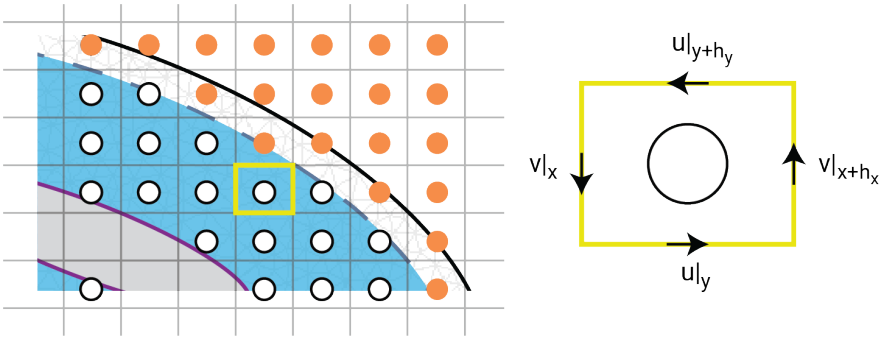


Figure 4.5: Illustration of the particles in the Lagrangian field, with a detailed view of the velocity computed at the four edges surrounding a single particle. This method, introduced by Billuart et al. [65], is used to correct the Lagrangian field in the interpolation region. The velocity values at the edges are employed to compute the circulation of the particle by integrating around the closed loop, ensuring proper correction of the particle strengths based on the Eulerian velocity field.

4.4. Evolution algorithm

The hybrid solver progresses through a series of iterations, with each cycle including two coupling steps as well as the evolution of both the Lagrangian and Eulerian solvers (Chapters 2 and 3). Given the Lagrangian and Eulerian solutions at an arbitrary time instant t_n, the algorithm for progressing to t_{n+1} is described here and is also summarized in the flowchart shown in Figure 4.6.

1. Evolve the Lagrangian solution for a time-step Δt_L. This step includes advecting the particles, redistributing them on the background Lagrangian grid, diffusing the particles, and removing particles during population control and wake control.

2. Calculate the Eulerian boundary conditions at the numerical boundary. The velocity and pressure gradient are computed and assigned to the face centers of the numerical boundary.

3. Evolve the Eulerian solution using an implicit time integration scheme over the interval Δt_L, divided into k_E substeps of size Δt_E. If $k_E > 1$, the boundary conditions are interpolated at each substep using Equation 4.4.

4. Correct the Lagrangian solution in the interpolation region using the more accurate Eulerian solution at t_{n+1}.

4.5. Dynamic mesh motion

In the context of *VPMFoam*, dynamic motions can be simulated easily and efficiently. The hybrid solver enables the entire Eulerian mesh to move as a solid body while Lagrangian particles update the boundary conditions. Papadakis et al. [72] successfully applied this approach in their compressible hybrid solver. This method combines the concepts of overset meshes and solid-body motions (further details on these concepts

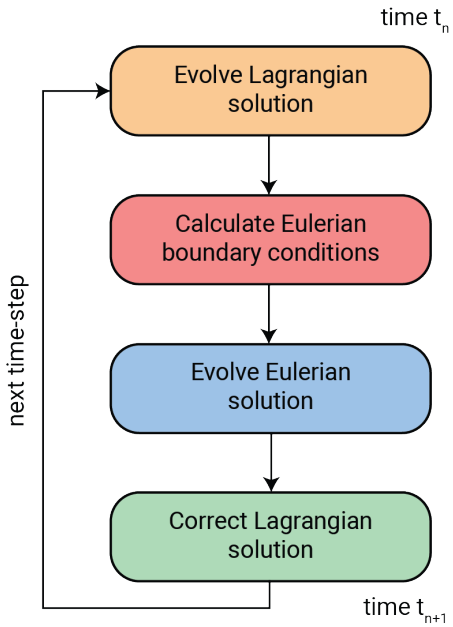


Figure 4.6: Flowchart of the evolution algorithm in *VPMFoam*.

are provided in Chapter 6). The Eulerian mesh moves as a solid body, but instead of a background static Eulerian mesh, it interacts with a background Lagrangian grid where information is interpolated, and boundary conditions are calculated. This technique avoids issues like mesh deformation, which are common with morphing meshes. It is especially beneficial when dealing with multiple bodies, as each bodys mesh can move independently, with their interactions managed solely through Lagrangian particles. For example, in the case of a wind turbine, each blade can be represented as an independent Eulerian mesh, moving independently as a solid body.

In this scenario, a question arises as to how the Eulerian solver accounts for fluid structures, such as vortices, that exist outside its boundary. For example, consider a scenario where a vortex exists beyond the Eulerian domain's boundaries, and the mesh is moving towards it, as depicted in Figure 4.7.

The Lagrangian solver effectively addresses this issue by capturing the vortex and updating the boundary conditions at the new coordinates of the Eulerian mesh. Therefore, it is essential to update the position of the Eulerian mesh before calculating the boundary conditions. Consequently, the evolution of the Eulerian solver can be divided into two distinct steps. First, the mesh undergoes motion, updating the coordinates of cells, faces, and vertices. Second, in a separate step, all operations related to correcting the fields due to the movement and the solution's evolution are performed.

However, to achieve this, modifications are required in the *pimpleFoam* solver. In the original version of *pimpleFoam*, mesh updating and equation solving occur simultaneously. To establish coupling with the Lagrangian solver, these two steps need to be

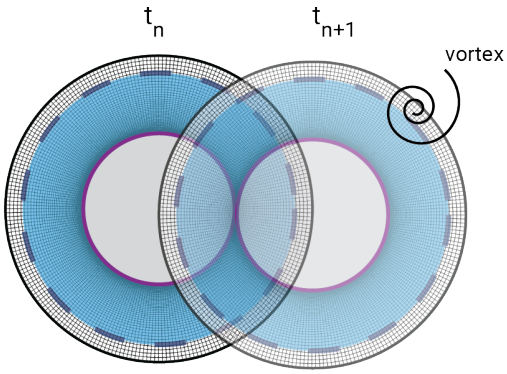


Figure 4.7: A vortex that inserts in the Eulerian subdomain when the Eulerian mesh moves towards it. This is just a graphical representation for clarifying purposes since the body cannot cover this distance in one time-step.

separated. As a result, the mesh is first updated, followed by the calculation of the necessary boundary conditions by the Lagrangian solver, and finally, the Eulerian solution is allowed to evolve. Thus, when conducting a dynamic mesh simulation, the Eulerian evolution comprises two distinct steps: the *evolve_mesh()* and the *evolve_only_solution()* functions. The following code snippets illustrate these functions, and the adapted flowchart in Figure 4.8 includes consideration for mesh motion.

Listing 4.1: The *evolve_mesh()* member function that updates only the mesh of the Eulerian part.

```
void EulerianPimpleFoam::evolve_mesh()
{
    if (LTS)
    {
        #include "setRDeltaT.H"
    }
    else
    {
        #include "CourantNo.H"
        #include "setDeltaT.H"
    }

    runTime++;

    fvModels.preUpdateMesh();

    mesh.update();
}
```

Listing 4.2: The *evolve_only_solution()* member function that evolves the Eulerian solution without moving the mesh.

```
void EulerianPimpleFoam::evolve_only_solution()
{
    Info<< "Time = " << runTime.timeName() << nl << endl;

```

```
    // --- Pressure-velocity PIMPLE corrector loop
    while (pimple.loop())
    {
            if (pimple.firstPimpleIter() || moveMeshOuterCorrectors)
        {
            if (mesh.changing())
            {
                MRF.update();

                if (correctPhi)
                {
                    #include "correctPhi.H"
                }
                if (checkMeshCourantNo)
                {
                    #include "meshCourantNo.H"
                }
            }
        }

        fvModels.correct();
        #include "UEqn.H"

        // --- Pressure corrector loop
        while (pimple.correct())
        {
            #include "pEqn.H"
        }
        if (pimple.turbCorr())
        {
            laminarTransport.correct();
            turbulence->correct();
        }
    }

    vorticity = fvc::curl(U);

    runTime.write();

    Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
        << "  ClockTime = " << runTime.elapsedClockTime() << " s"
        << nl << endl;

}
```

4.6. Software and computational resources

The main part of the solver has been developed in Python 3.9 [76][1]. Specifically, the primary components of the VPM solver are written in Python[1], as is the communication between OpenFOAM and the VPM. Furthermore, the OpenFOAM solver has been

[1] The original code was developed by Palha et al. [31] in the context of pHyFlow solver. Significant modifications have been made to this work.

wrapped[2] using the same version of Python.

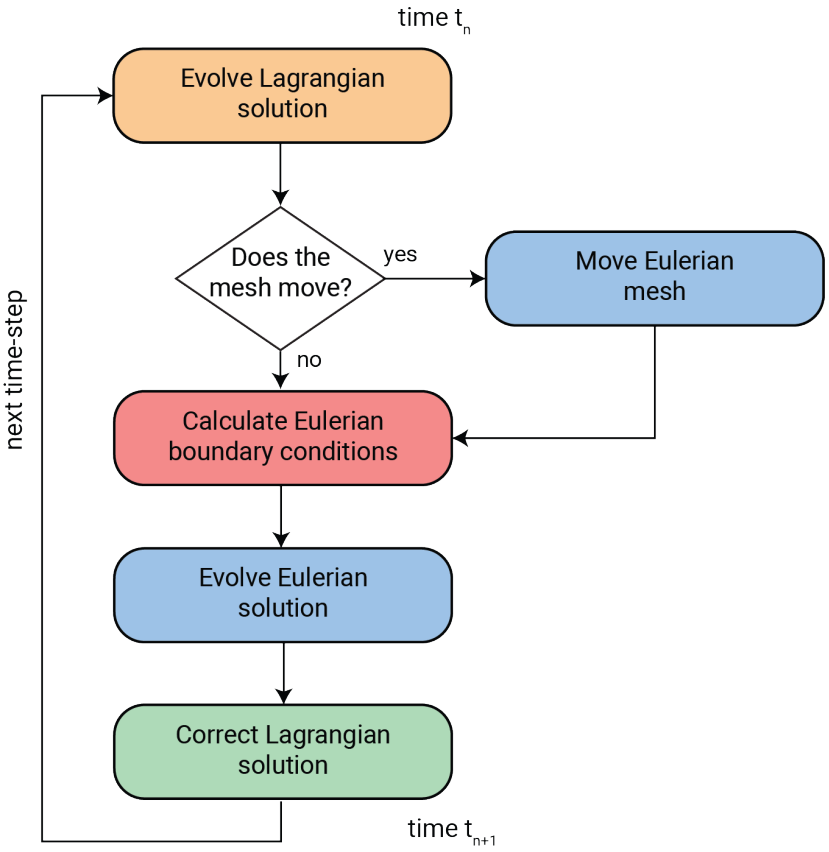


Figure 4.8: Flowchart of the evolution algorithm in *VPMFoam* including consideration for dynamic mesh cases.

Since Python is a high-level language and is generally not optimized for high-speed numerical computations, Palha et al. [31] initially supplemented it with lower-level languages. Additional optimization functions have now been implemented to enhance performance in specific areas. All induced field calculations for the particles can run either in serial or parallel on CPU and GPU. For CPU calculations, C [77], C++ [78], and Cython [79] are used, with parallelization achieved through OpenMP [80]. For GPU-based calculations, CUDA [81] has been employed, along with the CuPy [82] library, a Python library that enables GPU-accelerated computing, to further utilize GPU capabilities.

In addition, a FMM tree code developed by Goude et al. [15] and Engblom [16] has been integrated into the solver to compute induced velocities on both CPU and GPU. The FMM is a hierarchical, tree-based numerical algorithm that efficiently computes particle interactions, reducing the computational complexity from N^2 to as low as $N\log(N)$,

[2] The original wrapping code was developed by Carlos F. Baptista (personal communication), and significant modifications have since been made.

where N is the number of particles.

Most of the simulations presented in this dissertation were performed on the Delft-Blue cluster [83], utilizing type-a and type-b GPU nodes equipped with NVIDIA Tesla V100S 32GB and NVIDIA Tesla A100 80GB GPUs, respectively. Simulations requiring fewer computational resources were conducted on a laptop with an Intel Core i7-10750H 2.60GHz CPU and a GeForce RTX 2070 GPU. A summary of each systems specifications is presented below:

- **GPU type-a (GPU nodes with NVIDIA Tesla V100S 32GB)**
  - CPU cores per node: 48 (across two sockets)
  - CPU per node: 2x AMD EPYC 7402 24C 2.80 GHz
  - RAM per node: 256 GB DDR4 per node (4 NUMA nodes, 64 GB each)
  - GPU per node: 4x NVIDIA Tesla V100S (32 GB VRAM each)
  - Each GPU:
    - FP32 (Single-Precision): Up to 15.7 TFLOPS
    - FP64 (Double-Precision): Up to 7.8 TFLOPS
    - CUDA cores: 5,120
- **GPU type-b (GPU nodes with NVIDIA Tesla A100 80 GB)**
  - CPU cores per node: 64 (across two sockets)
  - CPU: 2x Intel Xeon Gold 6448Y 32C 2.1 GHz
  - RAM: 512 GB DDR4 per node (4 NUMA nodes, 128 GB each)
  - GPU per node: 4x NVIDIA Tesla A100 (80 GB VRAM each)
  - Each GPU:
    - FP32 (Single-Precision): Up to 19.5 TFLOPS
    - FP64 (Double-Precision): Up to 9.7 TFLOPS
    - CUDA cores: 6,912
- **Local Laptop (for lower computational needs)**
  - CPU: Intel Core i7-10750H @ 2.60 GHz (6 cores, 12 threads)
  - RAM: 16 GB DDR4
  - GPU: NVIDIA GeForce RTX 2070 (8 GB VRAM):
    - FP32 (Single-Precision): Up to 7.5 TFLOPS
    - FP64 (Double-Precision): Up to 234 GFLOPS
    - CUDA cores: 2,304

5

Validation - Static Cases

In the previous chapter, the fundamentals of the hybrid solver were discussed. This chapter starts the the validation of VPMFoam. The validation process follows a step-by-step approach, with each step introducing new challenges to progressively assess the solver's capabilities. The process starts with cases where the Eulerian domain remains static. First, the stationary Lamb-Oseen vortex case is presented to evaluate the solver's convergence compared to the pure Eulerian and Lagrangian solvers discussed in previous chapters. Next, the traveling Lamb-Oseen vortex and the dipole flow cases are explored to assess the solver's validity in scenarios involving advection. Subsequently, the flow around a cylinder at $Re = 550$ *is investigated. This examination ensures that the solver can accurately simulate flows involving solid boundaries and predict aerodynamic forces. The chapter ends with discussing the results obtained in the static cases.*

Parts of this chapter have been published in: **R. Pasolari**, C.J. Ferreira, and A. van Zuijlen, Coupling of OpenFOAM with a Lagrangian vortex particle method for external aerodynamic simulations, Physics of Fluids, vol. 35, no. 10, p. 107 115, Oct. 2023, ISSN: 1070-6631. DOI: 10.1063/5.0165878. [22]

5.1. Introduction

To be considered reliable, every new computational tool must undergo validation across a range of test cases. The validation process for the *VPMFoam* solver follows a step-by-step order. The validation begins with the simplest cases of flows without the presence of any solid bodies, to assess the validity of the developed tools. Subsequently, solid bodies are introduced into the flow. This chapter focuses on static cases, where the computational domain remains static, and on scenarios where only one solid object is present, leaving the study of dynamic cases and multibody problems for Chapter 6 and Chapter 7 respectively. All test cases presented are restricted to laminar flow regimes, in line with the overall focus of this dissertation. The validity and accuracy of the solver are assessed using the following cases:

1. **Stationary Lamb-Oseen vortex.** This case allows for the verification of the coupling procedure between the Eulerian and Lagrangian solvers. It is mainly used for assessing the solver's convergence compared to the pure Eulerian solver and the Lagrangian solver presented in Chapter 2 and Chapter 3 respectively.

2. **Traveling Lamb-Oseen vortex.** This case tests the ability of the solver to accurately propagate information out of the Eulerian subdomain. Additionally, it serves as a validation for the hybrid solver in a mixed advection-diffusion problem.

3. **Dipole flow.** In this scenario, another case without the presence of any solid body is encountered where the dipole traverses the Eulerian domain without the influence of a freestream velocity. This particular test case offers valuable insights since it allows for a comparison with the hybrid FEM-VPM solver developed by Palha et al. [31].

4. **Flow around a cylinder at low Reynolds number ($\mathbf{Re = 550}$).** This case is a commonly used benchmark for new solvers and allows for a more demanding case validation. The presence of a solid body can validate that the vorticity captured by the Eulerian solver is transferred correctly to the Lagrangian solver. The focus of this case is primarily on the quantitative validation of the aerodynamic coefficients and the Strouhal number.

5.2. Stationary Lamb-Oseen vortex

The Lamb-Oseen vortex [39] describes a vortex with a finite core that diffuses over space and time, as mentioned in previous chapters. The analytical solutions for the velocity and the vorticity field induced by the vortex were presented in Chapter 2, but they are reiterated here for reference:

$$u_\theta = \frac{\Gamma_v}{2\pi r}\left[1.0 - exp\left(-\frac{r^2}{4\nu(t+\tau)}\right)\right] \quad , \quad u_r = 0 \tag{5.1a}$$

$$\omega = \frac{\Gamma_v}{4\pi\nu(t+\tau)} exp\left(-\frac{r^2}{4\nu(t+\tau)}\right) \tag{5.1b}$$

u_θ is the circumferential velocity, u_r is the radial velocity and ω is the vorticity. Γ_v is the strength of the vortex, t is the simulation time, τ is the time constant (for smooth distribution of the vorticity field), ν is the kinematic viscosity and r is the distance from the core center.

Here, a Lamb-Oseen vortex with strength Γ_v is located at the center of a square domain of length L. The vortex remains stationary throughout the simulation, while it diffuses in space and time. The configuration of the simulation, as well as the Eulerian mesh, are presented in Figure 5.1.

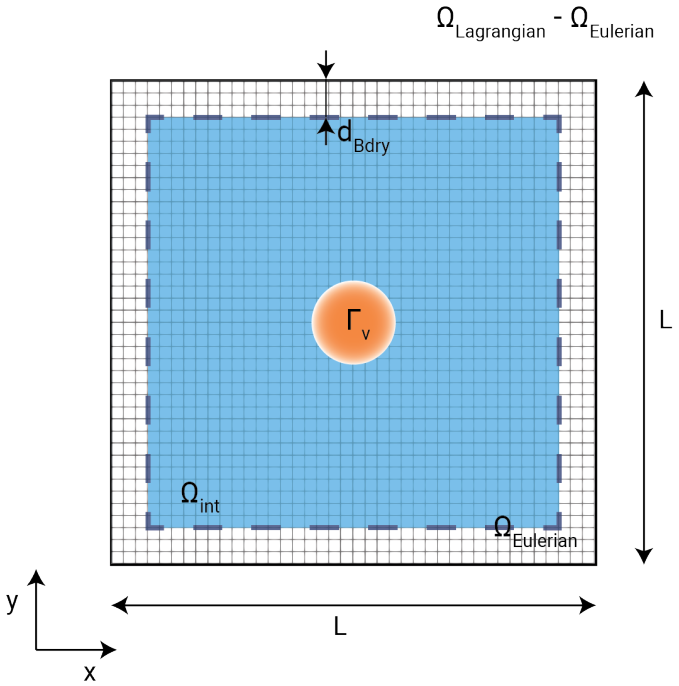


Figure 5.1: The configuration of the stationary Lamb-Oseen vortex using *VPMFoam*. The vortex is located at the center of the square domain, and it diffuses in space and time.

The simulation parameters are listed in Table 5.1. This test case, as previously mentioned, assesses the hybrid solver's convergence, similar to the pure VPM case (see Chapter 2) and the pure OpenFOAM case with analytical boundary conditions (see Chapter 3). For consistency, the same mesh densities with Chapter 3 are used, ranging from 10×10 to 320×320, with particle spacing matched to the cell size in each case. Errors are measured for both the Lagrangian part and the Eulerian part of the hybrid solver based on the relations in Equation 2.33 and Equation 3.2 respectively.

Figure 5.2 presents the spatial convergence results for the stationary Lamb-Oseen vortex, measured within the Lagrangian component of the hybrid solver. The data illus-

Table 5.1: Simulation parameters for the case of a stationary Lamb-Oseen vortex using *VPMFoam*.

Parameter	Symbol	Value	Dimension
Particle strength	Γ_v	-0.05	m^2/s
Domain edge length	L	1.0	m
Initial position	(x_0, y_0)	$(0.5, 0.5)$	m
Freestream velocity	(U_x^{inf}, U_y^{inf})	$(0.0, 0.0)$	m/s
Lamb-Oseen time constant	τ	4.0	s
Kinematic viscosity	ν	5×10^{-4}	m^2/s
Time-step	$\Delta t_L, \Delta t_E$	0.001	s
Overlap ratio	λ	1	–
Interpolation domain offset	d_{Bdry}	0.1	m
Population control thresholds	$(\Gamma_{loc}, \Gamma_{glob})$	$(10^{-8}, 10^{-8})$	m^2/s

trate that initialization errors decrease rapidly as particle resolution increases, consistent with findings in Chapter 2 and as shown in Figure 2.7b (note that Figure 2.7b uses a semilog plot, while here a log-log scale is employed). By the end of the simulation, the convergence achieves a second-order rate. It is important to note that the y-axes of the two subplots in Figure 5.2 differ in range, with the second plot showing values several orders higher. This difference arises because, as explained in Chapter 2, the first redistribution of particles introduces a notable increase in errors.

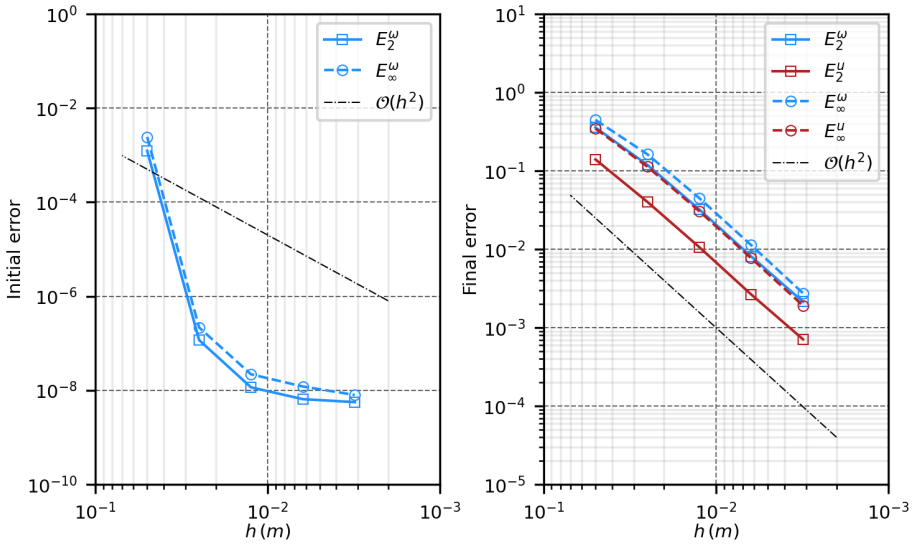


Figure 5.2: Spatial convergence study for the *VPMFoam* solver applied to a stationary Lamb-Oseen vortex, with measurements taken from the Lagrangian component of the hybrid solver. The left plot illustrates the initial error, while the right plot the final error.

Figure 5.3 displays the spatial convergence results measured in the Eulerian component of the hybrid solver. These results align closely with those presented in Chapter 3. The initial error in the vorticity field converges at a second-order rate, due to the curl operation, while the final errors approach second-order convergence. Specifically, the vorticity achieves close to second-order convergence, while the velocity field shows slight deviations from second-order behavior, similar to the results seen in Figure 3.5.

Overall, these findings confirm that the coupling mechanisms within the hybrid solver are effective, allowing it to achieve convergence rates comparable to both the pure VPM solver and OpenFOAM. A more detailed evaluation of the solver's performance will follow in subsequent test cases.

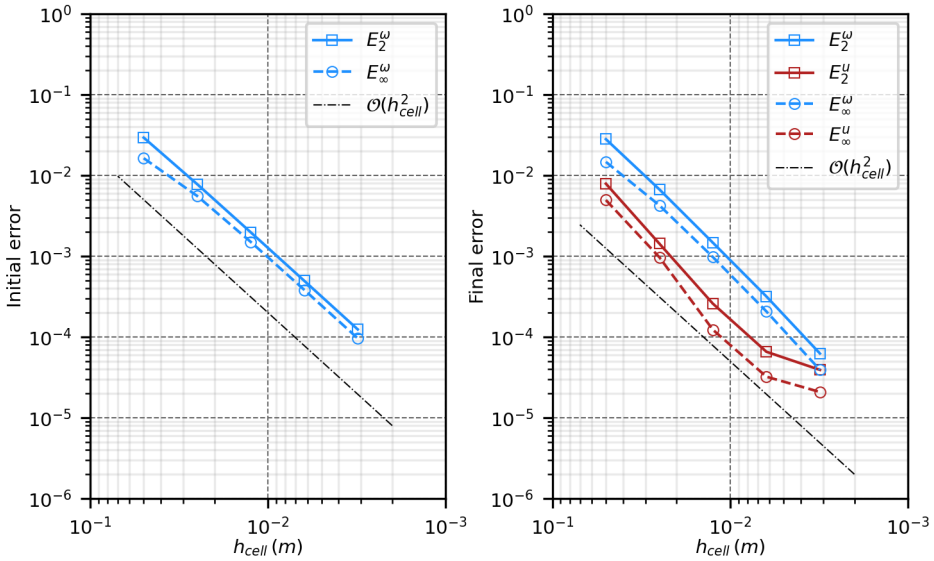


Figure 5.3: Spatial convergence study for the *VPMFoam* solver applied to a stationary Lamb-Oseen vortex, with measurements taken from the Eulerian component of the hybrid solver. The left plot illustrates the initial error, while the right plot the final error.

5.3. Traveling Lamb-Oseen vortex

Now another Lamb-Oseen case is explored. This time the vortex with strength Γ_ν is initially located at the center of a square domain of length L, but it is advected with the freestream velocity U_{inf} as it is illustrated in Figure 5.4. The simulation is executed until the vortex traversed totally out of the Eulerian domain. Table 5.2 provides a summary of the problem parameters, including geometry, and the simulation parameters.

Figures 5.5 and 5.6 display the vorticity and y-component of the velocity field, respectively, for the hybrid and analytical solutions at three distinct time points. The upper portion of the contour plots represents the hybrid solution, while the analytical solution is depicted in the lower portion. The two solutions exhibit perfect agreement at their interface, and the transition between them occurs smoothly without any irregularities.

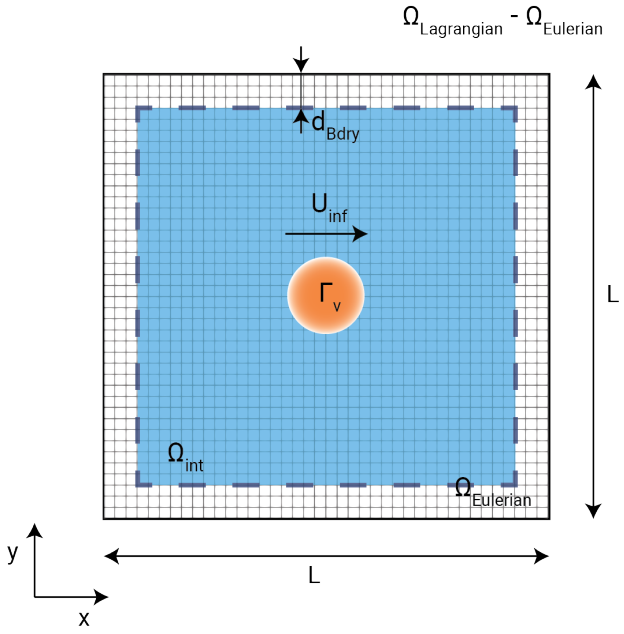


Figure 5.4: The configuration of the traveling Lamb-Oseen vortex using *VPMFoam*. The vortex is initially located at the center of the square domain, and it is advected with a freestream velocity U_{inf} towards the $+x$ direction.

Table 5.2: Simulation parameters for the case of a traveling Lamb-Oseen using *VPMFoam*.

Parameter	**Symbol**	**Value**	**Dimension**
Particle strength	Γ_v	-0.5	m^2/s
Domain edge length	L	1.0	m
Initial position	(x_0, y_0)	$(0.5, 0.5)$	m
Freestream velocity	(U_x^{inf}, U_y^{inf})	$(1.0, 0.0)$	m/s
Lamb-Oseen time constant	τ	4.0	s
Kinematic viscosity	ν	5×10^{-4}	m^2/s
Lagrangian time-step	Δt_L	0.002	s
Overlap ratio	λ	1	–
Vortex particles spacing	h	0.006	m
Interpolation domain offset	d_{Bdry}	$10 \cdot h$	m
Population control thresholds	$(\Gamma_{loc}, \Gamma_{glob})$	$(10^{-8}, 10^{-8})$	m^2/s
Eulerian mesh density	N_{cells}	320×320	–
Eulerian time-step	Δt_E	0.002	s

This is evident even when the vortex approaches and finally exits the Eulerian domain. Figure 5.7 illustrates the relative error of the vorticity field in the three different time instances. The error presented in the figure is calculated as:

$$E^{\omega}_{rel}|_t = \frac{\omega_H|_t - \omega_A|_t}{max\{\omega_A\}|_t} \cdot 100$$

where ω_H and ω_A being the hybrid and the analytical solutions, respectively, measured at the same discretization points. The initial observation reveals a discretization error at the center of the initial vortex, approximately around 1%. As the vortex moves, the primary error remains concentrated at its core, gradually increasing to about 1.5%. Additionally, a minor error, less than 0.3%, emerges at the left edge of the Eulerian domain, but it remains stable throughout the simulation without magnifying further.

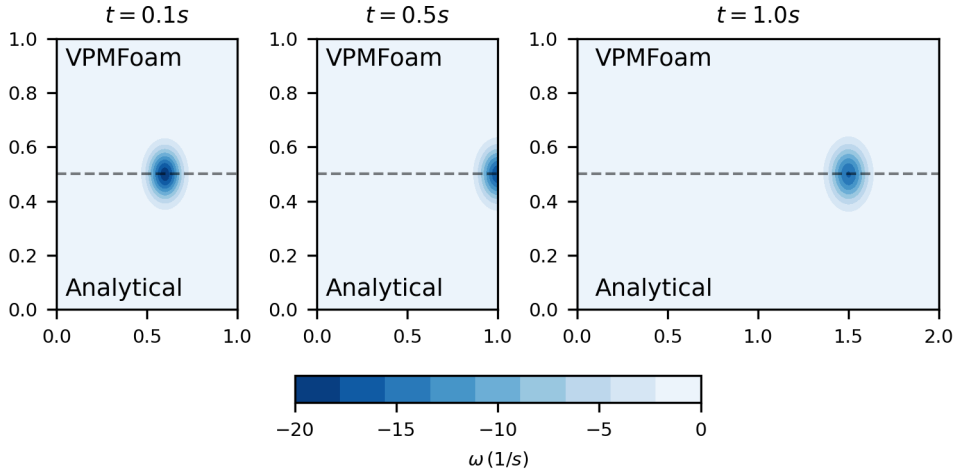


Figure 5.5: The vorticity field for the *VPMFoam* (upper part) and the analytical (lower part) solutions in three different time instances ($t = 0.1s$, $t = 0.5s$, $t = 1.0s$), for the case of a Lamb-Oseen vortex, initially located at the center of a square domain, and advected by a freestream velocity U_{inf}.

5.4. Dipole flow

This case deals with the evolution of a vortex dipole. The specific case was also examined by Palha et al. [31] so it would be interesting to show a comparison between the two solvers. All the parameters are set to be the same, and specifically, the simulation is initialized with a Clercx-Bruneau dipole [84], with the positive monopole located at $(x_1, y_1) = (-1.0, 0.1)\,m$ and the negative monopole at $(x_2, y_2) = (-1.0, -0.1)\,m$. Both the monopoles have a radius $R_v = 0.1\,m$, and the induced vorticity field is calculated as:

$$\omega(x, y, 0) = \omega_0 \left(1 - \frac{{r_1}^2}{R_v^2}\right) e^{-(r_1^2/R_v^2)} - \omega_0 \left(1 - \frac{{r_2}^2}{R_v^2}\right) e^{-(r_2^2/R_v^2)} \tag{5.2}$$

where, $\omega_0 = 299.5283853751/s$ is the characteristic vorticity, and $r_i^2 = (x - x_i)^2 + (y - y_1)^2$. The configuration and the geometry of the case can be seen in Figure 5.8 while the

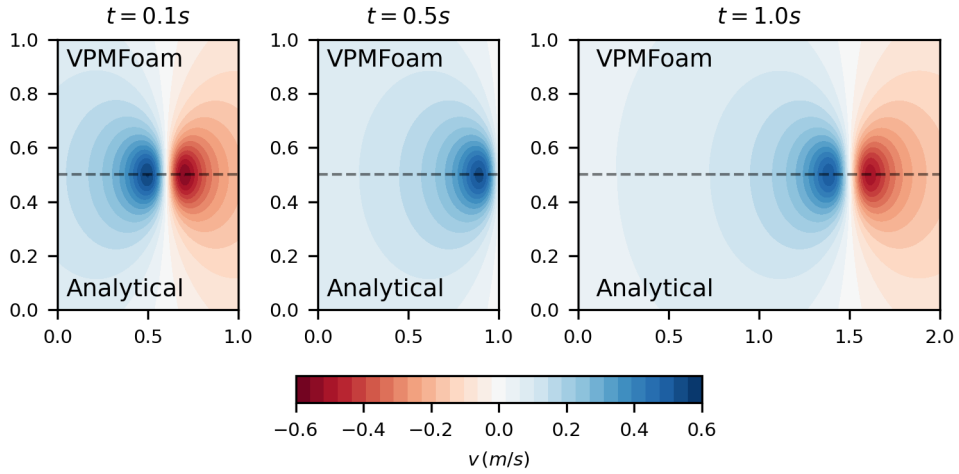


Figure 5.6: The y-component of the velocity field for the *VPMFoam* (upper part) and the analytical (lower part) solutions in three different time instances ($t = 0.1s$, $t = 0.5s$, $t = 1.0s$), for the case of a Lamb-Oseen vortex, initially located at the center of a square domain, and advected by a freestream velocity U_{inf}.

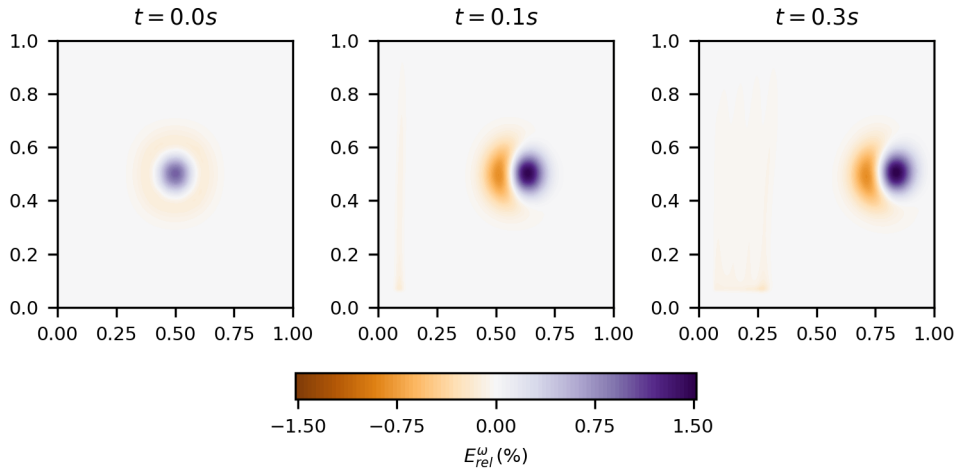


Figure 5.7: The relative vorticity error in three different time instances ($t = 0.0s$, $t = 0.1s$, $t = 0.3s$), for the case of a Lamb-Oseen vortex, initially located at the center of a square domain, and advected by a freestream velocity U_{inf}.

simulation parameters are summarized in Table 5.3.

The maximum vorticity of the dipole during the simulation is depicted in Figure 5.9. In this plot also the FEM and hybrid results from Palha et al. [31] are presented. It can be observed that *VPMFoam* predicts the maximum vorticity in the flow in exactly the same way as the reference. The same decrease in the vorticity when the dipole inserts the Eulerian domain is present here. In addition to this, Figure 5.10 illustrates the vortic-

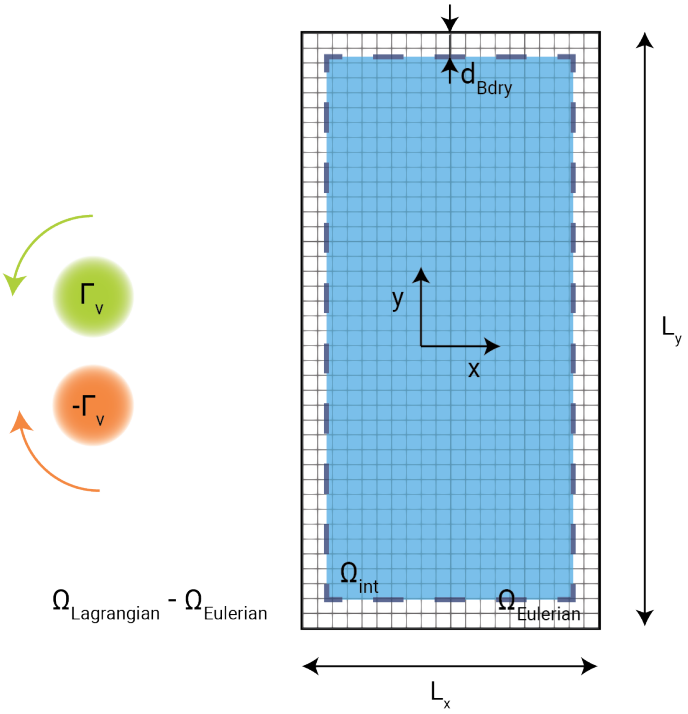


Figure 5.8: The configuration of a vortex dipole, initially located out of the Eulerian subdomain, using *VPM-Foam*.

Table 5.3: Simulation parameters for the case of a dipole flow, using *VPMFoam*.

Parameter	**Symbol**	**Value**	**Dimension**
Domain edge x length	L_x	0.5	m
Domain edge y length	L_y	1.0	m
Initial position of monopole 1	(x_1, y_1)	$(-1.0, 0.1)$	m
Initial position of monopole 2	(x_2, y_2)	$(-1.0, -0.1)$	m
Kinematic viscosity	ν	1.6×10^{-3}	m^2/s
Lagrangian time-step	Δt_L	2.5×10^{-4}	s
Overlap ratio	λ	1	–
Vortex particles spacing	h	5×10^{-3}	m
Interpolation domain offset	d_{Bdry}	$5 \cdot h$	m
Population control thresholds	$(\Gamma_{loc}, \Gamma_{glob})$	$(10^{-14}, 10^{-14})$	m^2/s
Eulerian mesh density	N_{cells}	150×300	–
Eulerian time-step	Δt_E	2.5×10^{-4}	s

ity contours in four different time instances, specifically at times $t = [0.0, 0.2, 0.4, 0.6]s$.

These results are in total accordance with the corresponding results of the reference case. An important comment that must be added here is that the trailing vorticity of the dipole undergoes a noticeable reduction upon traversing the Eulerian subdomain. This reduction is a direct consequence of the increased artificial diffusion introduced by the Eulerian solver.

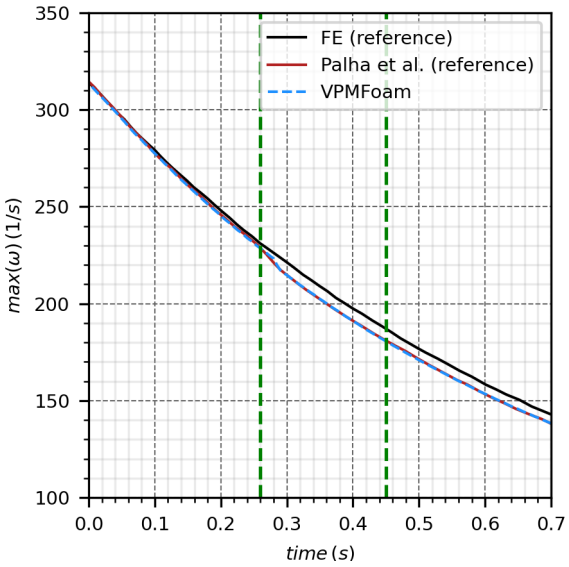


Figure 5.9: The maximum vorticity in the case of the dipole flow, in the time interval $t = 0.0s$ to $t = 0.7s$. The result obtained from *VPMFoam* is compared to the FEM simulation and the hybrid simulation provided by Palha et al. [31].

5.5. Flow around a cylinder at low Reynolds number

The last case to validate the hybrid solver is the flow around a cylinder at $Re = 550$. This is a test case extensively studied in the past (e.g., in the work of Koumoutsakos et al. [85]), and it allows for assessment of the performance of the solver when simulating more realistic problems. The presence of the solid body facilitates benchmarking of the process of capturing vorticity generation and transferring the vorticity field from the Eulerian mesh to the Lagrangian particles.

The vorticity generation, as well as the viscous phenomena, take place in a narrow region close to the solid body and they are responsible for its aerodynamic behavior. For the validation of the solver, the aerodynamic coefficients C_l and C_d, as well as the Strouhal number are compared with the corresponding results of pure Eulerian simulations performed in OpenFOAM, and the results from the hybrid solver developed by Billuart et al. [65]. The computational domain for the pure Eulerian case, as well as the boundary conditions, are illustrated in Figure 5.11. A Grid Convergence Study (GCS) of the OpenFOAM case was performed to establish the reference case for comparison with *VPMFoam*, and it can be seen in Table 5.4.

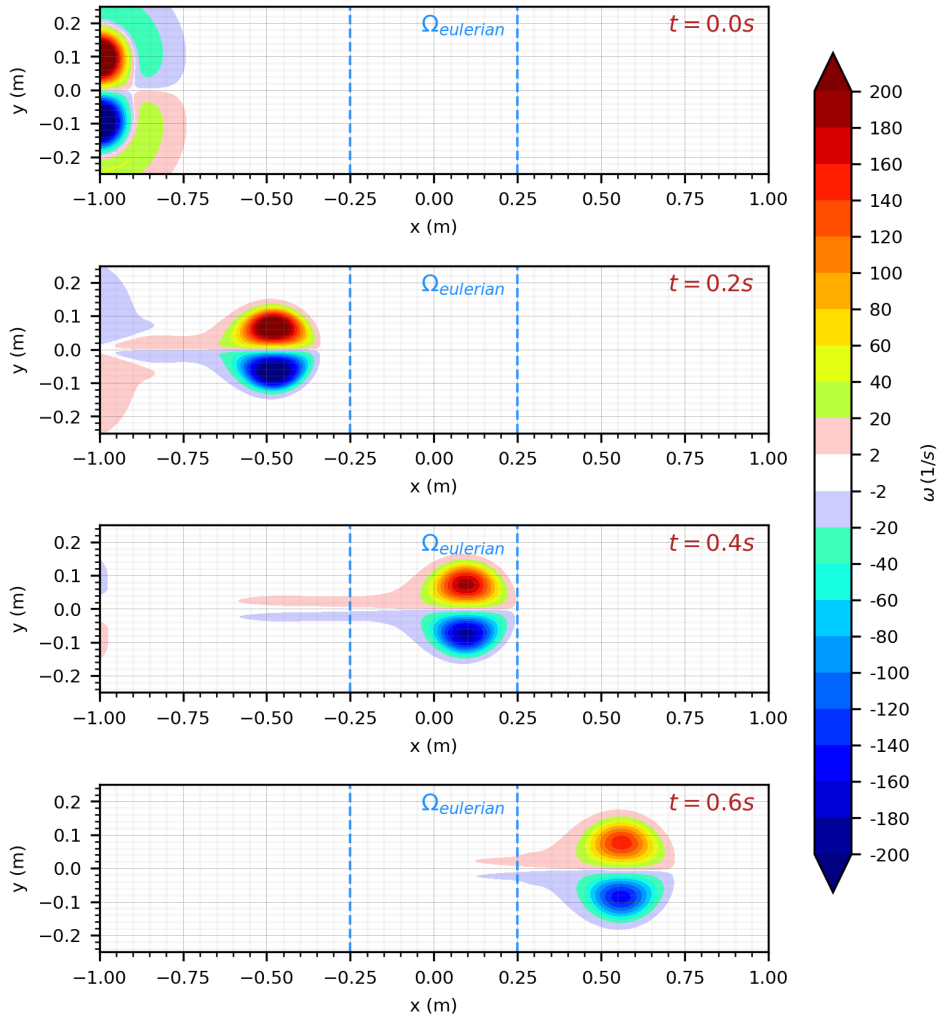


Figure 5.10: The evolution of the vorticity field of the dipole at $t = [0.0, 0.2, 0.4, 0.6]s$, using *VPMFoam*. The colormap of the contours is in accordance with the corresponding colormap in the work of Palha et al. [31].

Figure 5.12 depicts the domain decomposition utilized for the hybrid simulation, where the Eulerian subdomain is confined to a narrow region in proximity to the body extending from the surface of the cylinder to a radius $R_{ext} = 2 \cdot R_{in}$. The parameters employed for this simulation are outlined in Table 5.5. For the Eulerian subdomain, a structured grid (constructed in Gmsh [86]) is used. The cells close to the numerical boundary have an aspect ratio close to 1.0, and their size is similar to that of the core size of the Lagrangian particles in order to reduce the interpolation errors close to the boundary during the correction step [65].

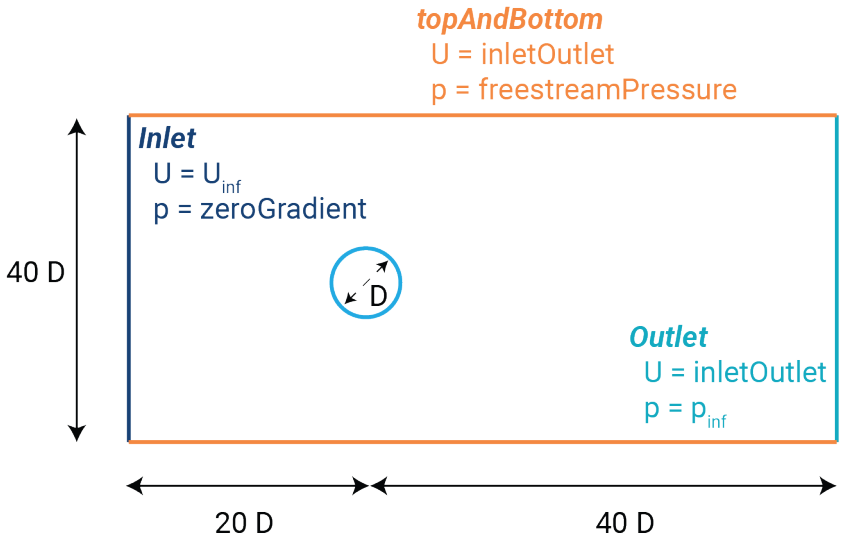


Figure 5.11: The geometry and the boundary conditions for the OpenFOAM case of the flow around a cylinder at $Re = 550$.

Table 5.4: GCS for the OpenFOAM case of the flow around a cylinder at $Re = 550$.

Case	No of cells	$\overline{C_d}$	$\max(C_l)$	Strouhal
Coarse	9550	1.3876	1.0992	0.2333
1^{st} refinement level	19726	1.4128	1.1448	0.2325
2^{nd} refinement level	39293	1.4336	1.1826	0.2271
3^{rd} refinement level	80053	1.4381	1.1963	0.2271
4^{th} refinement level	159982	1.4387	1.1969	0.2271
5^{th} refinement level	318894	1.4389	1.1970	0.2271

The primary goal of this case study is to examine the periodic phase of the flow around a cylinder, rather than focusing on the starting-transition phase. As a result, the validation process emphasizes the fully developed flow regime, where a periodic Von Kármán vortex street can be observed in the cylinder's wake. Nonetheless, the aerodynamic coefficients during the initial phase are also provided to offer a comprehensive overview of the simulation. The comparison on the aerodynamic coefficients with the pure Eulerian case after GCS are depicted in Figures 5.13 and 5.14. The aerodynamic coefficients, as well as the Strouhal number, are compared with the pure Eulerian simulation in OpenFOAM, but also with the results of the hybrid solver presented by Billuart et al. [65]. This comparison in summarized in Table 5.6.

Figure 5.13 demonstrates the aerodynamic coefficients at the initial phase of the flow. It is important to note that this study primarily focuses on investigating the periodic phase. Due to this emphasis, there is a discrepancy between the two solvers during the

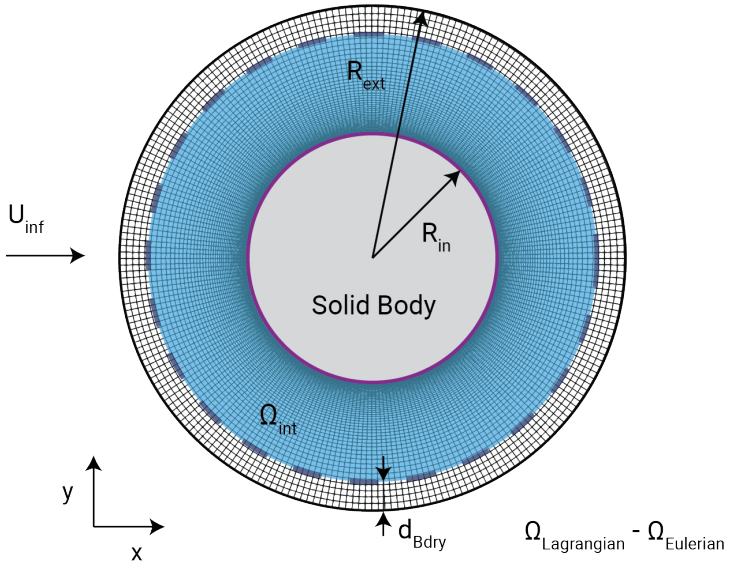


Figure 5.12: The geometry of the flow around a cylinder case at $Re = 550$ for the *VPMFoam* case. The Eulerian mesh is a narrow region close to the solid boundary while the Lagrangian extends indefinitely.

Table 5.5: Problem and simulation parameters for the case of the flow around a cylinder at Re=550, using *VPMFoam*.

Parameter	**Symbol**	**Value**	**Dimension**
Reynolds number	Re	550	–
Simulation time	t	300	s
Lagrangian time-step	Δt_L	2.5×10^{-4}	s
Overlap ratio	λ	1	–
Vortex particles spacing	h	5×10^{-3}	m
Interpolation domain offset	d_{Bdry}	$5 \cdot h$	m
Population control thresholds	$(\Gamma_{loc}, \Gamma_{glob})$	$(10^{-14}, 10^{-14})$	m^2/s
Eulerian time-step	Δt_E	0.002	s
Eulerian mesh density	N_{cells}	51804	–

initial phase. Specifically, no perturbation was introduced to initiate instabilities from the beginning, resulting in different starting phases between the two solvers. The curves depicted in Figure 5.14 represent the periodic phase, and they exhibit great similarity. Both solvers yield the same Strouhal number, with the difference in aerodynamic forces being less than 1.0%. The two lines here have been adjusted to align with each other at the same phase.

Table 5.6: Maximum C_l value, mean C_d value and Strouhal number, along with the relative errors of the aerodynamic coefficients, for the case of flow around a cylinder at $Re = 550$.

Case	$\overline{\mathbf{C_d}}$	$\mathbf{max(C_l)}$	**Strouhal**
OpenFOAM (after GCS)	1.4389	1.1970	0.2271
VPMFoam	1.4270	1.1870	0.2272
Difference	0.827%	0.835%	0.044%
Billuart et al. [65]	1.4270	1.1770	0.2272

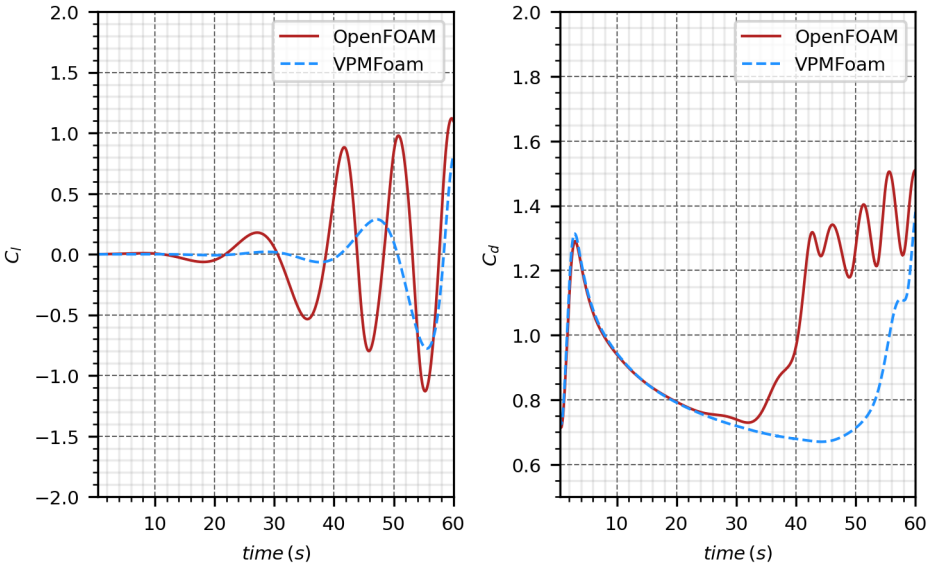


Figure 5.13: Aerodynamic coefficients for the flow around a cylinder at $Re = 550$ for the *VPMFoam* and the OpenFOAM case, at the initial phase.

The outcomes obtained using *VPMFoam* demonstrate excellent concurrence with the corresponding results from the hybrid solver developed by Billuart et al. [65]. The maximum disparity in the lift coefficient is around 1.0%, while the Strouhal number and the mean value of the drag coefficient are identical up to the third decimal place, as presented in Table 5.6.

Figure 5.15 illustrates the vorticity field for the *VPMFoam* and for the OpenFOAM case at four different time instances. In order to have an agreement of the vortical structures at the wake for the two cases, different time instances will be presented under the condition that the two simulations are at similar stages. The first contour is at the beginning of the simulation, where the two vortices are being created at the top and bottom parts of the cylinder. During the transition phase, two similar instances are presented (second contours), but there is a notable difference between them. As mentioned ear-

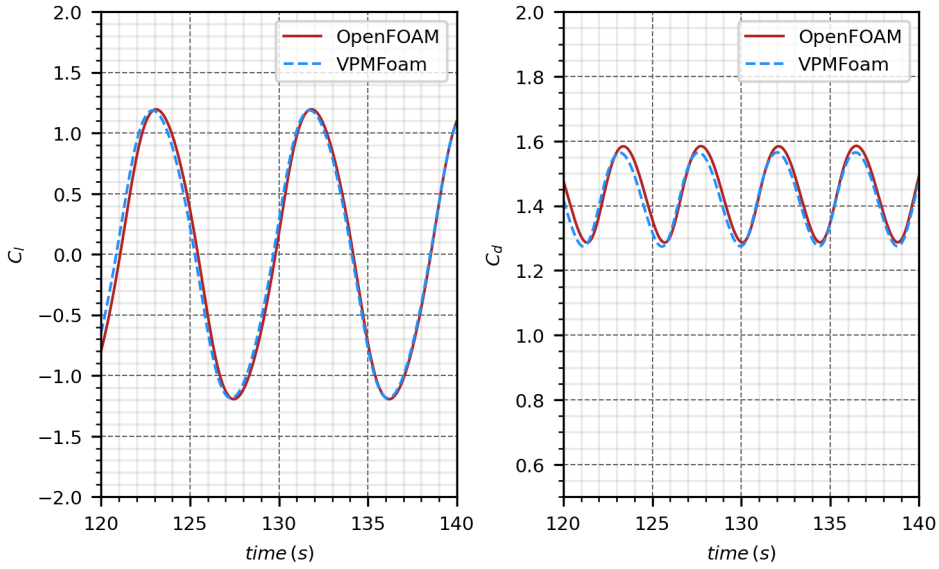


Figure 5.14: Aerodynamic coefficients for the flow around a cylinder at $Re = 550$ for the *VPMFoam* and the OpenFOAM case, at the periodic phase.

lier, this distinction arises due to the fact that the transition in the two solvers does not occur in the same manner. The third and the fourth contours are at a time instance where the lift coefficient is minimum and maximum respectively. It can be seen that the hybrid solver can reproduce the vorticity field in great agreement with the pure Eulerian solver. It is worth mentioning that in the same figure, it can be observed that the vortical structures that are in the far-field appear more diffused in the pure Eulerian simulation, which demonstrates the diffusive nature of Eulerian solvers.

The transition region between the Eulerian and Lagrangian subdomains is a critical aspect of hybrid simulations and can introduce errors if not handled properly. The smooth transition observed in Figure 5.16 indicates that the coupling between the two subdomains is well-implemented and the vorticity field can be accurately transferred between them. This is important for the overall accuracy of the simulation, as errors in this region can propagate throughout the domain and affect the entire solution.

5.6. Conclusions

It has been shown that OpenFOAM can work along the VPM in the framework of a hybrid solver without any crucial complexity. The two solvers can work together without any irregularities introduced by their coupling. *VPMFoam* is capable of simulating flows with and without the presence of solid bodies, showing a great agreement with existing CFD solvers. In all cases, the transition of the solution from the Eulerian to the Lagrangian domain was very smooth. The hybrid solver is capable of predicting the aerodynamic coefficients as well as the Strouhal number with variations less than 1.0%. The present

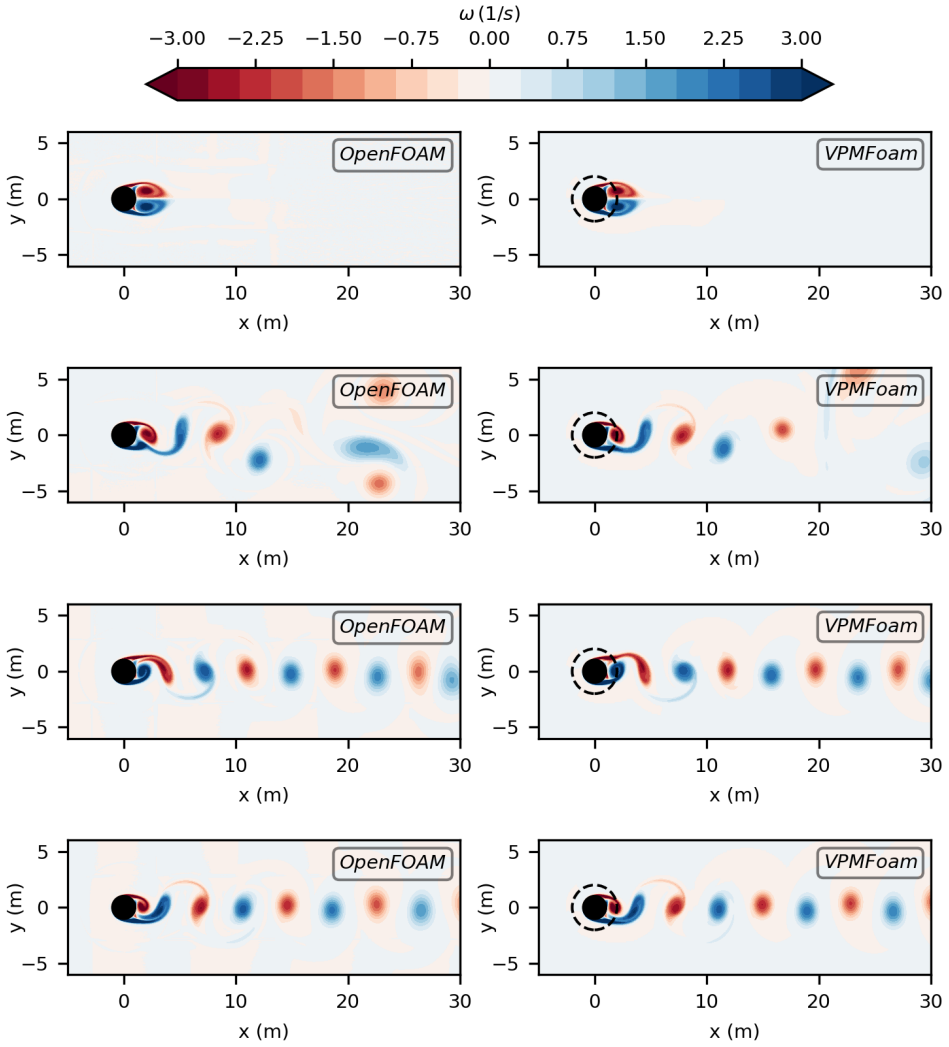


Figure 5.15: The vorticity field contours at four different time instances (starting phase, transition phase, minimum lift, maximum lift) for the case of the flow around a circular cylinder at $Re = 550$, using OpenFOAM and *VPMFoam*. The circle formed by the black dashed line [– – –] defines the Eulerian mesh region.

hybrid solver produces similar results to the existing hybrid solver presented by Billuart et al. [65]. The hybrid solver seems to reduce the artificial diffusion that is present in pure Eulerian simulation. This effect is particularly pronounced in the case of the dipole, where even a minor Eulerian domain can significantly diffuse the trailing vorticity. Furthermore, when considering the flow around the cylinder, a subtle reduction in artificial diffusion becomes apparent in the downstream-traveling vortices.

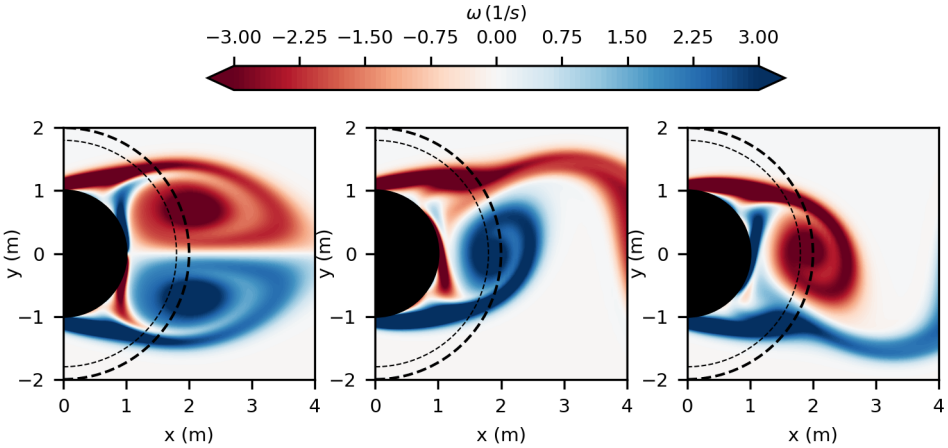


Figure 5.16: The vorticity field contours at three different time instances (starting phase, minimum lift, maximum lift) for the case of the flow around a circular cylinder at $Re = 550$, using *VPMFoam*. The contours depict the region close to the solid boundary to focus on the transition between the Eulerian and the Lagrangian field. Inside the interpolation region, the Eulerian solution is depicted, while the rest is the Lagrangian solution. The outer circle formed by the thick black dashed line [– – –] defines the Eulerian mesh region, while the thinner [------] defines the interpolation region.

6

Validation - Dynamic Cases

This chapter extends the discussion on the validation of the solver. In the previous chapter, the solver was validated through a series of cases where the Eulerian mesh, and consequently the solid object, remained stationary. In this chapter, the solver's capabilities to simulate cases with dynamically moving meshes are analyzed. The chapter begins with an introduction to the dynamic capabilities of the solver and then proceeds to validation, starting with a series of cases where no solid objects are present in the flow, and later moving to cases where solid objects are introduced. A common high-frequency error in dynamic hybrid Eulerian-Lagrangian solvers with solid presence is addressed, and the methods to alleviate this issue are discussed.

Parts of this chapter have been published in:

- **R. Pasolari**, C.J. Ferreira, A. van Zuijlen and C.F. Baptista, Dynamic Mesh Simulations in OpenFOAM: A Hybrid Eulerian-Lagrangian Approach, Fluids 9, 51, (2024), DOI: 0.3390/fluids9020051. [40]
- **R. Pasolari**, J. Pan, C.J. Ferreira, A. van Zuijlen, Flow over traveling and rotating cylinders using a hybrid EulerianLagrangian solver, Computers & Fluids, Volume 279, 106327, (2024), ISSN 0045-7930, https://doi.org/10.1016/j.compfluid.2024.106327. [87]

6.1. Introduction

The dynamic simulation of moving solid objects plays a pivotal role in the field of external aerodynamics. In many scenarios, such as wind turbine rotors, aircraft, and helicopter blades, the objects are inherently in motion. These aerodynamic objects can either undergo forced motion (prescribed motion) or move freely, as seen in Vortex Induced Vibrations (VIV). Conducting dynamic simulations for these scenarios necessitates high-fidelity solutions to accurately capture the complex phenomena involved.

OpenFOAM offers built-in features for handling moving meshes via the *dynamicMeshDict* dictionary. It provides several methods for managing the movement of bodies, with the most notable in aerodynamics being the overset mesh (available only in the *ESI-OpenCFD* [48] versions of OpenFOAM), the morphing mesh, and sliding interfaces using Arbitrary Mesh Interface (AMI) [88]. In the case of morphing mesh, the mesh's topology undergoes alterations by displacing the patch associated with the moving object while preserving the connectivity of the internal cells. Conversely, in the overset mesh approach, a stationary background mesh is employed, and for each moving object, an additional mobile mesh is superimposed on top of the former. This additional mesh moves about the background mesh, and their interaction is achieved through the interpolation of variables between them. Alletto [3] compared morphing and overset mesh methods by simulating the forced and free oscillations of a 2D cylinder, while Wu et al. [89] employed the sliding interfaces method for airfoil simulations. An application of the AMI technique can be found in the work of Pan et al. [11], where a Savonius rotor is simulated. Another method, solid-body motion, is primarily used in internal flow simulations such as sloshing tanks, as demonstrated by Li et al. [90] and Chen et al. [91], but is less commonly employed in external aerodynamics.

In addition to Eulerian solvers, Lagrangian methods have also been utilized for moving body simulations. Alvarez et al. [92] used a VPM to model multirotor aerodynamic interactions, with the propeller represented as a rotary lifting surface. Similarly, Karimi-Zindashti et al. [93] applied a deterministic vortex method with vortex panels to simulate the flow around rotating cylinders. However, Lagrangian solvers often struggle to accurately resolve regions near solid boundaries, necessitating additional solvers for support.

As discussed in Chapter 4, *VPMFoam* enables the entire Eulerian mesh to move as a solid body, while the Lagrangian particles manage the boundary conditions. This approach combines elements of overset mesh techniques with solid-body motion. Rather than interacting with a static background Eulerian mesh, the moving Eulerian mesh communicates with a background Lagrangian grid, where information is interpolated, and boundary conditions are set.

6.2. Validation - No solid bodies

First, the validation process will be conducted using domains without the presence of solid bodies. The validation cases to be used are:

- Translation
- Rotation
- Linear Oscillation
- Combination of motions (multi-motion)

In all the cases that are going to be presented here, the same Eulerian mesh is used. This mesh, illustrated in Figure 6.1, is a structured square mesh with an edge length of L, while the distance of the mesh edges to the edges of the interpolation region is denoted as d_{Bdry}.

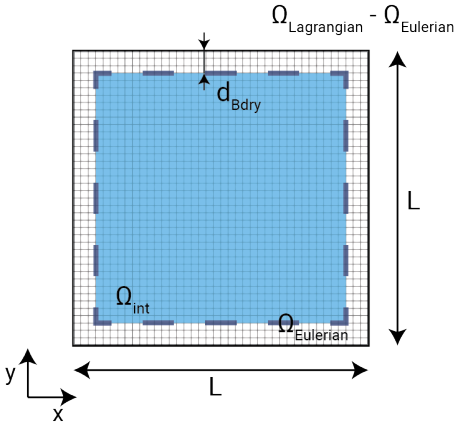
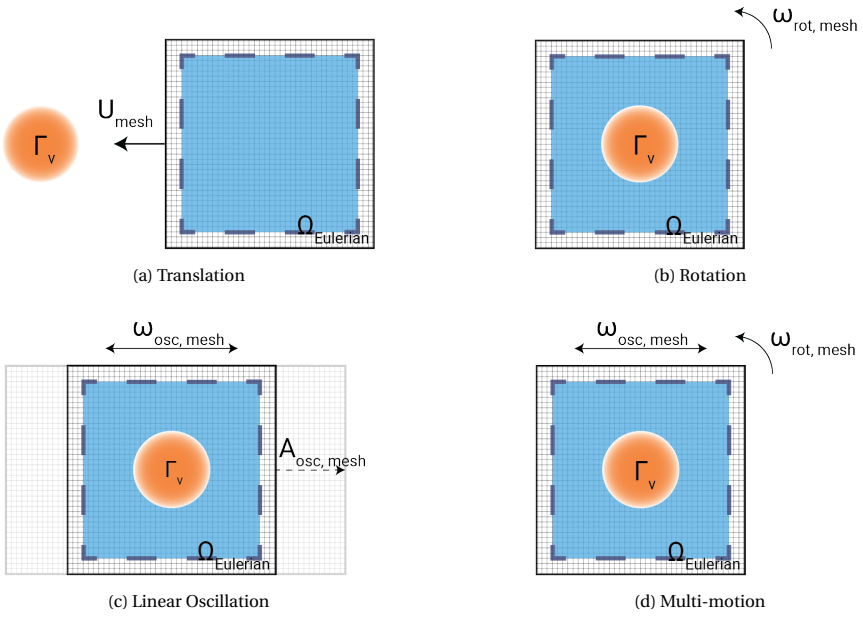


Figure 6.1: The geometry and mesh utilized for the validation of *VPMFoam* through dynamic mesh cases, showcasing the Eulerian domain, the Lagrangian domain, and the interpolation region.



(a) Translation

(b) Rotation

(c) Linear Oscillation

(d) Multi-motion

Figure 6.2: Validation cases featuring a stationary vortex of strength Γ, with the Eulerian subdomain exhibiting relative motion. The following relative motions are considered: **a)** Translation, **b)** Rotation, **c)** Linear Oscillation, and **d)** Linear Oscillation with Rotation (Multi-Motion).

The various cases chosen for validation, as depicted in Figure 6.2, encompass translational motion, rotational motion, linear oscillation, and multi-motion, which combines linear oscillation and rotation. In each of these cases, a stationary Lamb-Oseen vortex [39] is present within the flow. The Eulerian domain is in relative motion to the vortex, following the specified motion patterns. In all cases, a comparison is conducted with analytical results. Table 6.1 provides an overview of the geometry and motion parameters for the various cases presented, while Table 6.2 summarizes the simulation parameters that are consistent across all cases. To ensure a fair comparison among the different motion scenarios, the time-step is adjusted individually in each case to achieve a nearly identical maximum Courant number, which is approximately set to be 1.0.

Table 6.1: Variable parameters associated with the domain motion across the different cases.

Parameter	Symbol	Value	Dimension
Case 1 - Translation			
Initial mesh center	$(x_{E,initial}, y_{E,initial})$	$(0.5, 0.5)$	m
Vortex particle coordinates	(x_v, y_v)	$(0.0, 0.5)$	m
Mesh velocity	$\mathbf{u}_{mesh}$	$[-1.0 \;\; 0.0]$	m/s
Case 2 - Rotation			
Initial mesh center	$(x_{E,initial}, y_{E,initial})$	$(0.5, 0.5)$	m
Vortex particle coordinates	(x_v, y_v)	$(0.5, 0.5)$	m
Mesh rotational speed	$\omega_{rot,mesh}$	1.571	rad/s
Origin of rotation	(x_R, y_R)	$(0.5, 0.5)$	m
Case 3 - Linear oscillation			
Initial mesh center	$(x_{E,initial}, y_{E,initial})$	$(0.5, 0.5)$	m
Vortex particle coordinates	(x_v, y_v)	$(0.5, 0.5)$	m
Axis of oscillation	$axis$	x	–
Mesh oscillation frequency	$\omega_{osc,mesh}$	1.571	rad/s
Origin of oscillation	(x_O, y_O)	$(0.5, 0.5)$	m
Amplitude of oscillation	$A_{osc,mesh}$	0.75	m
Case 4 - Multi-motion			
Initial mesh center	$(x_{E,initial}, y_{E,initial})$	$(0.5, 0.5)$	m
Vortex particle coordinates	(x_v, y_v)	$(0.5, 0.5)$	m
Axis of oscillation	$axis$	x	–
Mesh oscillation frequency	$\omega_{osc,mesh}$	1.571	rad/s
Origin of oscillation	(x_O, y_O)	$(0.5, 0.5)$	m
Amplitude of oscillation	$A_{osc,mesh}$	0.75	m
Mesh rotational speed	$\omega_{rot,mesh}$	1.571	rad/s
Origin of rotation	(x_R, y_R)	$(0.5, 0.5)$	m

In higher-dimensional problems (i.e., beyond 1D), the classical Courant number expression $\mathrm{Co} = \frac{u\Delta t}{\Delta x}$ is no longer sufficient, as there is no unique spatial step Δx in arbitrary

Table 6.2: Simulation parameters for all the moving domain cases, using *VPMFoam*.

Parameter	Symbol	Value	Dimension
Particle strength	Γ_v	-0.5	m^2/s
Freestream velocity	(U_x^{inf}, U_y^{inf})	$(0.0, 0.0)$	m/s
Lamb-Oseen time constant	τ	4.0	s
Kinematic viscosity	ν	5×10^{-4}	m^2/s
Simulation time	t_{sim}	2.0	s
Eulerian mesh density	N_{cells}	320×320	–
Domain edge length	L	1.0	m
Eulerian time-step	Δt_E	0.001	s
Lagrangian time-step	Δt_L	0.001	s
Vortex particles spacing	h	0.006	m
Overlap ratio	λ	1	–
Interpolation domain offset	d_{bdry}	$10 \cdot h$	m
Population control thresholds	$(\Gamma_{loc}, \Gamma_{glob})$	$(10^{-14}, 10^{-14})$	m^2/s

meshes. Instead, the Courant number is computed from the fluxes through the control volume faces using the following expression:

$$\mathrm{Co} = \frac{1}{2}\Delta t \sum_{\text{faces}} \frac{|\mathbf{U}_f \cdot \mathbf{n}_f| A_f}{V_p}$$

Here, $\mathbf{U}_f \cdot \mathbf{n}_f$ is the velocity component normal to face f, A_f is the area of face f, and V_p is the volume of the control cell. Conceptually, the reciprocal of a representative distance Δx can be approximated by the ratio of the total face area to the control volume. However, such a definition would lead to an effective Courant number of zero in incompressible or mass-conserving flows, since the net flux into a cell is zero by conservation of mass. To account for the directional magnitude of the velocity fluxes, the absolute value is taken, and the sum is scaled by a factor of $\frac{1}{2}$, which aligns with the finite volume formulation used in OpenFOAM and other Eulerian solvers.

6.2.1. Translation

The first results to be examined are those of the mesh in translational motion. As depicted in Figure 6.2a, the vortex initially lies outside the Eulerian domain, with the Eulerian mesh traveling across the vortex in the $-x$ direction passing through the vortex. Figure 6.3 displays a comparison of the x and y components of the velocity field for both the analytical and hybrid solutions within the same contour. The contour is divided, with the upper part corresponding to the hybrid solution and the lower part to the analytical solution. The red dashed box denotes the position of the Eulerian mesh at the corresponding time instance. Evidently, there is a substantial agreement between the two solutions.

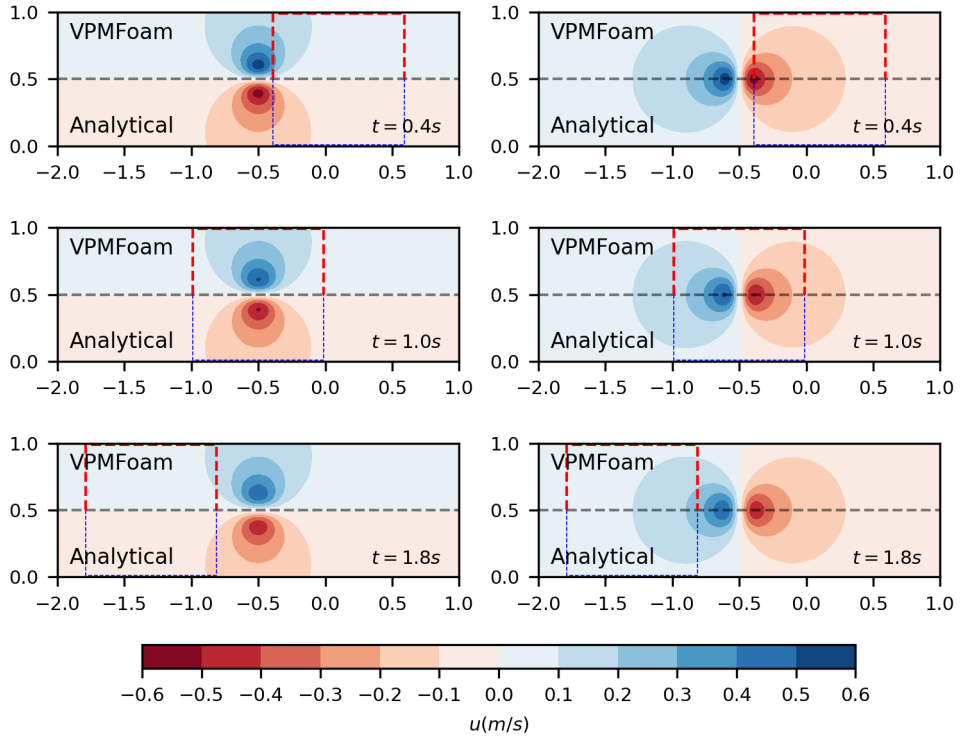


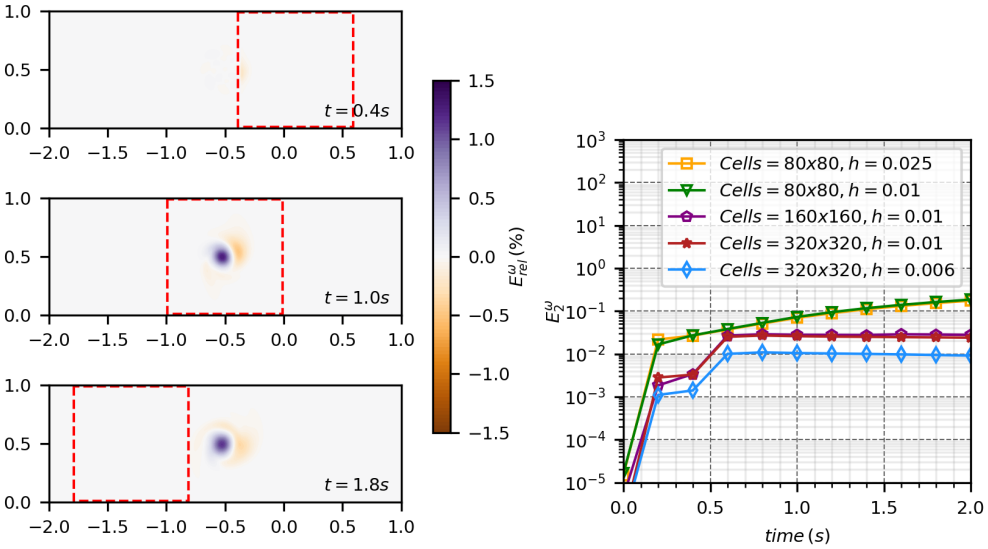
Figure 6.3: Comparison of the velocity field between the hybrid (**top**) and the analytical (**bottom**) solution, for the translational case. The x-component of velocity is depicted on the left, while the y-component is shown on the right. The analytical and hybrid solutions are separated by a black dashed line [– – –] and the Eulerian domain at the corresponding time instance is defined by a red rectangle [- -]. The dashed box extends to the analytical part only for visualization purposes and is illustrated with a thinner blue line.

To assess the error between the two solutions in the vorticity field, Figure 6.4a illustrates the relative error in the vorticity field for the two solutions. The error is computed in a manner that prevents division by very small numbers in regions where vorticity is close to zero, as described by the Equation 6.1.

$$E^{\omega}_{rel}|_t = \frac{\omega_H|_t - \omega_A|_t}{max\{\omega_A\}|_t} \cdot 100 \tag{6.1}$$

where $E^{\omega}_{rel}|_t$ is the relative vorticity error at time t, ω_H the hybrid vorticity and ω_A the analytical vorticity, measured at the same discretization points. It is noteworthy that, for all time instances, the error remains below 1.5%. The error is negligible when the Eulerian domain has not reached the vortex, and there is a sudden increase when it crosses the vortex. This increase in the error is primarily due to discretization and interpolation from the Eulerian field to the Lagrangian field. Figure 6.4b presents a comparison of the E_2-norm of vorticity error for five different cases within the translational domain. These cases employ various mesh sizes and particle spacings to demonstrate the convergence

of the solution as the cell count increases or the particle spacing decreases. In all the results provided, the finest mesh size and particle spacing are used, specifically, 320×320 cells and $h = 0.006m$.



(a) Vorticity error (Translation)

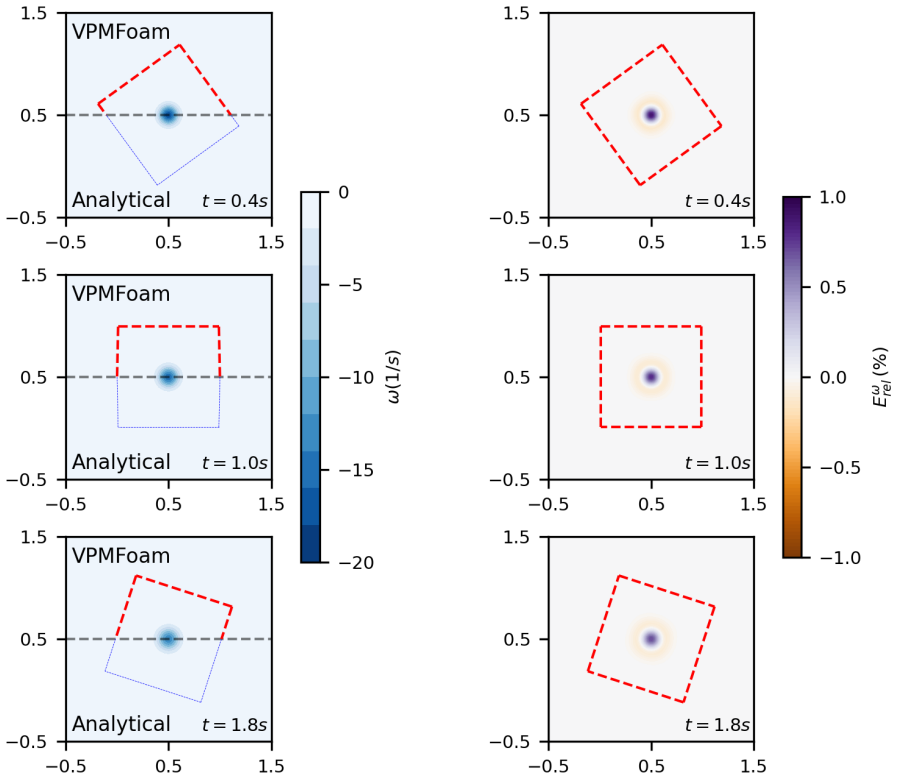
(b) Convergence test (Translation)

Figure 6.4: On the left, the relative error of the vorticity field between the hybrid and the analytical solution for the translational case is illustrated. The Eulerian domain at the corresponding time instance is defined by a red rectangle . On the right, the convergence test for the mesh size and the particles' spacing is depicted.

6.2.2. Rotation

The second validation case involves the pure rotation of the Eulerian domain, as depicted in Figure 6.2b. This time, the vortex is situated within the Eulerian domain, and the domain initiates a rotational motion around its center, coinciding with the center of the vortex's core. In this and the subsequent cases, only the vorticity field and the associated error are presented.

Figure 6.5 illustrates a noteworthy agreement in the vorticity field between the analytical and hybrid solutions. Specifically, the maximum errors are constrained to 1.0%, and this error is present from the beginning of the simulation due to discretization. As the vortex neither enters nor exits the Eulerian domain in this case, no additional errors are introduced.



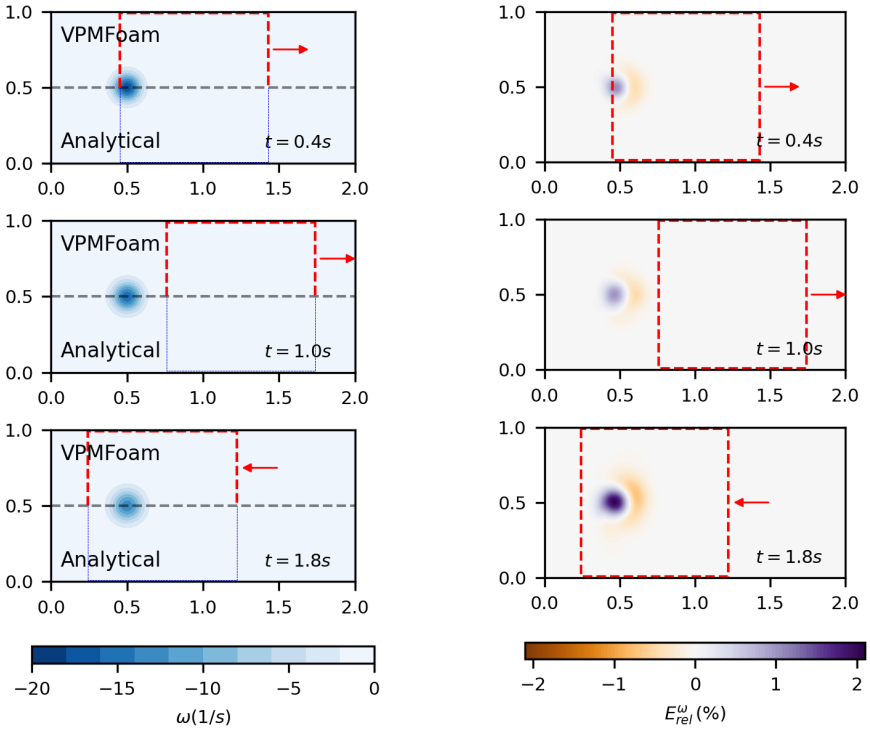
(a) Vorticity (Rotation)

(b) Vorticity error (Rotation)

Figure 6.5: On the left is the comparison of the vorticity field between the hybrid (**top**) and the analytical (**bottom**) solution, for the rotational case. The analytical and hybrid solutions are separated by a black dashed line [– – –] and the Eulerian domain at the corresponding time instance is defined by a red rectangle [- - -]. The dashed box extends to the analytical part only for visualization purposes and is illustrated with a thinner blue line. On the right, the relative error of the vorticity field between the hybrid and the analytical solution for the rotational case is illustrated.

6.2.3. Linear oscillation

The third validation case involves the oscillation of the Eulerian mesh, as it can be seen in Figure 6.2c. The domain moves up to the point that the vortex is entirely outside of the Eulerian mesh, and when it reaches the peak of its oscillation, it returns to its initial position. In Figure 6.6, it is evident that the vorticity field between the analytical solution and the hybrid solution demonstrates a strong agreement. Regarding errors, all errors remain below 2.0%. Notably, in this case, where the vortex is initially within the Eulerian domain, the interpolation error is present from the start of the simulation, and it increases by approximately 0.7% when the Eulerian domain encompasses the vortex for the second time.



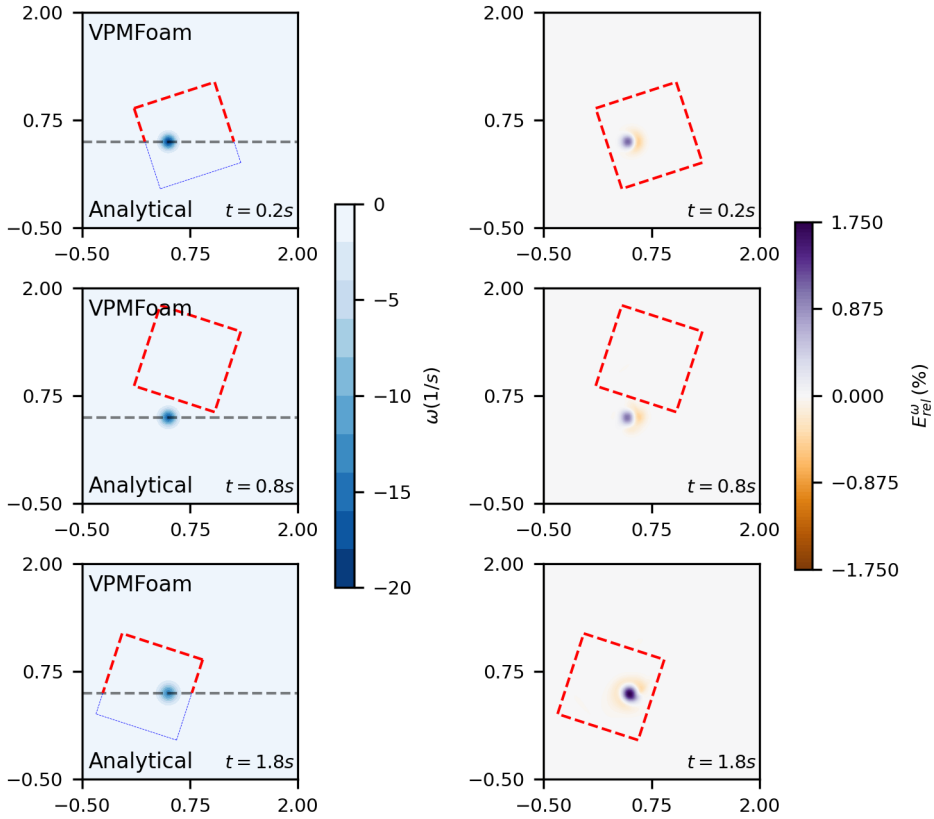
(a) Vorticity (Linear oscillation)

(b) Vorticity error (Linear oscillation)

Figure 6.6: On the left is the comparison of the vorticity field between the hybrid (**top**) and the analytical (**bottom**) solution, for the linear oscillation case. The analytical and hybrid solutions are separated by a black dashed line [– – –] and the Eulerian domain at the corresponding time instance is defined by a red rectangle [- - -]. The dashed box extends to the analytical part only for visualization purposes and is illustrated with a thinner blue line. The red arrow shows the direction that the mesh moves at the specific time instance. On the right, the relative error of the vorticity field between the hybrid and the analytical solution for the linear oscillation case is illustrated.

6.2.4. Multi-motion

In the last validation case the Eulerian mesh performs a more complex motion pattern (Figure 6.2d). Specifically, the vortex is initially located at the center of the Eulerian mesh, when this starts a combined linear oscillation in the x axis and a rotation around the point $(0.5, 0.5)$. This combined motion causes the Eulerian domain to pass through the vortex twice. Figure 6.7a shows a great agreement of the vorticity field between the hybrid and the analytical solution, while the vorticity error presented in Figure 6.7b is always lower than 1.75%. An initial error around 1.0% is present since the start of the simulation, and it grows to 1.75% when the Eulerian mesh passes through the vortex for the second time.



(a) Vorticity (Multi-motion)

(b) Vorticity error (Multi-motion)

Figure 6.7: On the left is the comparison of the vorticity field between the hybrid (**top**) and the analytical (**bottom**) solution, for the multi-motion case. The analytical and hybrid solutions are separated by a black dashed line [– – –] and the Eulerian domain at the corresponding time instance is defined by a red rectangle [ ]. The dashed box extends to the analytical part only for visualization purposes and is illustrated with a thinner blue line. On the right, the relative error of the vorticity field between the hybrid and the analytical solution for the multi-motion case is illustrated.

6.3. Validation - With solid bodies

Having validated the solver in domains without solid bodies, it is now possible to extend its application to more complex cases involving solid bodies. The solver is validated using the traveling cylinder case at $Re = 100$ and the rotating cylinder case at two different rotational speeds at $Re = 200$. Since a combination of these two motions can describe every motion a rigid body can undergo, it is crucial to validate both to achieve a comprehensive analysis of the two-dimensional dynamic case.

The same configuration is employed for the different cases, as depicted in Figure 6.8. The simulated cylinder has a radius R_{in}, and the Eulerian domain extends up to a distance of R_{ext} from the center of the cylinder (the black circle represents the numerical boundary). The Lagrangian solver covers the entire computational domain and is cor-

rected within the interpolation region Ω_{int} (illustrated in blue). Notably, a small layer of cells is excluded from the correction process, denoted as d_{Bdry}. The Eulerian mesh is exclusively comprised of hexahedra. Mesh density increases near the cylinder's surface to capture near-wall phenomena effectively, with the stretching ratio being 1.05 and the height of the first cell 2.7mm (for the base mesh presented below).

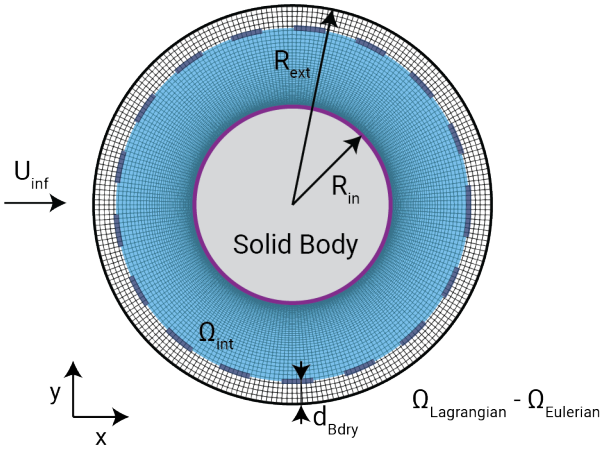


Figure 6.8: The general cylinder configuration used for the validation cases of the dynamic mesh motion in domains with solid bodies presence, using *VPMFoam*.

Some parameters between the different cases are similar, and they can be summarized in Table 6.3. The parameters that differentiate among the cases are mentioned in the corresponding subsections.

Table 6.3: Global simulation parameters for the validation cases of the dynamic mesh motion in domains with solid bodies presence, using *VPMFoam*.

Parameter	**Symbol**	**Value**	**Dimension**
Cylinder's radius	R_{in}	0.5	m
Eulerian domain radius	R_{ext}	1.0	m
Simulation time	t_{sim}	100	s
Overlap ratio	λ	1	–
Interpolation domain offset	d_{Bdry}	0.09	m

6.3.1. Traveling cylinder

The first case to validate the solver is the traveling cylinder. *VPMFoam* has already been validated into the case of a stationary cylinder with a freestream velocity in Chapter 5, but in that case, for $Re = 550$. Here, the validation case presents the opposite situation. The freestream velocity is set to zero while the cylinder travels with a constant velocity U_{mesh} as shown in Figure 6.9 operating at $Re = 100$. In order to have a fair comparison between the static and the dynamic case, the static case at $Re = 100$ is also simulated

here. The initial simulations are executed using a base Eulerian mesh, which consists of 11,040 hexahedra, and is depicted in Figure 6.9.

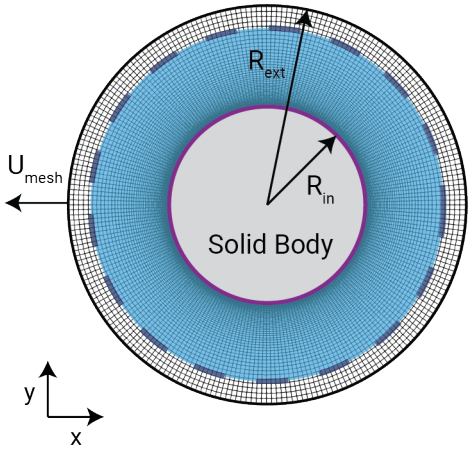


Figure 6.9: The configuration of the traveling cylinder at $Re = 100$, using *VPMFoam*.

A preliminary comparison of the aerodynamic coefficients from the two simulations (static and dynamic) is illustrated in Figure 6.10. While the mean values of the drag and lift coefficients appear similar, the dynamic simulation exhibits noticeable high-frequency oscillations, particularly in the drag coefficient time history, with significant amplitude.

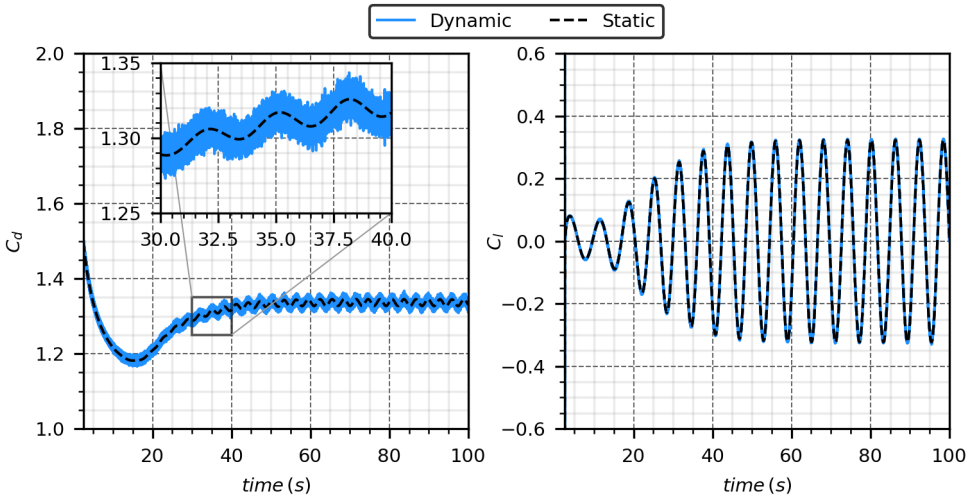


Figure 6.10: Lift and drag coefficients over time for the case of the static and traveling cylinder at $Re = 100$, with the base Eulerian mesh and particle spacing $h = 0.03\,m$, using the hybrid solver.

Upon detailed analysis, it was determined that these oscillations are caused by the movement of the Eulerian mesh over the Lagrangian particles, which are consistently redistributed at the same grid points in global coordinates. This results in a continuous shift of particles near the solid body, where the highest vorticity is recorded, relative to the Eulerian mesh, as shown in Figure 6.11. Consequently, particles close to the solid body at one time-step may end up inside the body at the next, necessitating their removal. This alteration in the distribution of the particles carrying the highest vorticity affects the boundary conditions of the Eulerian solver, thereby inducing oscillations in the forces acting on the body.

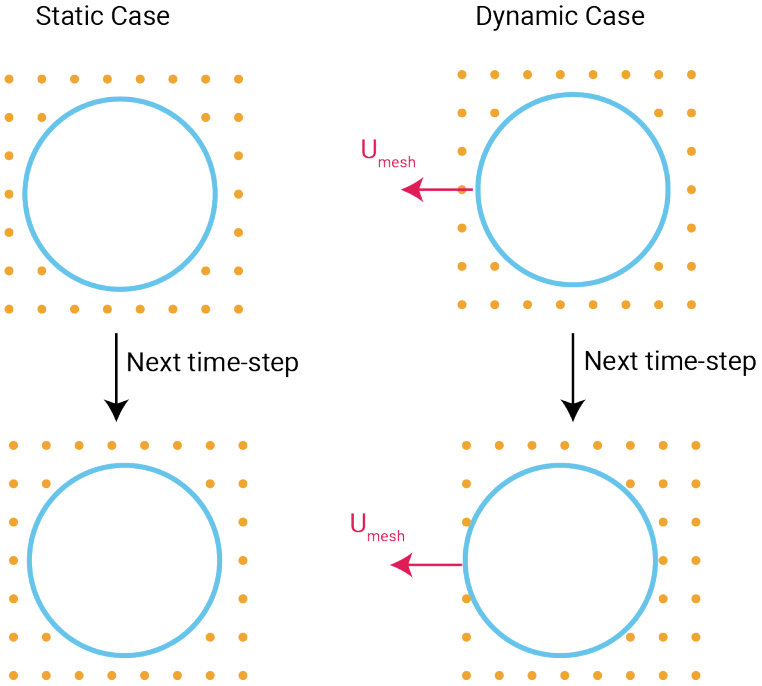


Figure 6.11: Configuration of the Lagrangian particles close to the solid boundary over a time-step. In the static case, the configuration remains the same; for the dynamic case, the distribution of the particles around the solid boundary changes, leading to high-frequency oscillations in the calculation of the Eulerian boundary conditions and, subsequently, on the aerodynamic forces.

However, these oscillations can be mitigated in two ways. First, increasing the resolution of the Lagrangian solver results in a denser particle distribution around the solid body, effectively reducing the relative distances as the mesh advances. Second, by synchronizing the redistribution points of the Lagrangian particles with the motion of the Eulerian mesh, further refinement is achieved. These approaches are validated through the simulations summarized in Table 6.4.

Figure 6.12 displays the first 30 s of the drag coefficient for the six different cases outlined in Table 6.4. On the left, the cases with a spacing of $h = 0.03\,m$ are shown, and on the right, those for $h = 0.015\,m$. Initially, the left subfigure demonstrates that synchro-

Table 6.4: Test cases of the traveling cylinder at $Re = 100$, to demonstrate the alleviation of the high-frequency oscillations on the aerodynamic forces.

Case state	**Particle spacing** (m)	**Eulerian mesh**	**Synchronization**
Static	0.030	Base	-
Static	0.015	Base	-
Dynamic	0.030	Base	False
Dynamic	0.030	Base	True
Dynamic	0.015	Base	False
Dynamic	0.015	Base	True

nizing the particles with the mesh movement significantly reduces the amplitude of the oscillations. Further reductions are observed when the particle spacing is decreased, as evidenced by the comparison between the left blue line and the right one. Moreover, synchronizing the particles with the mesh movement, in addition to reducing particle spacing, significantly minimizes oscillations, bringing results into close alignment with those of the static case.

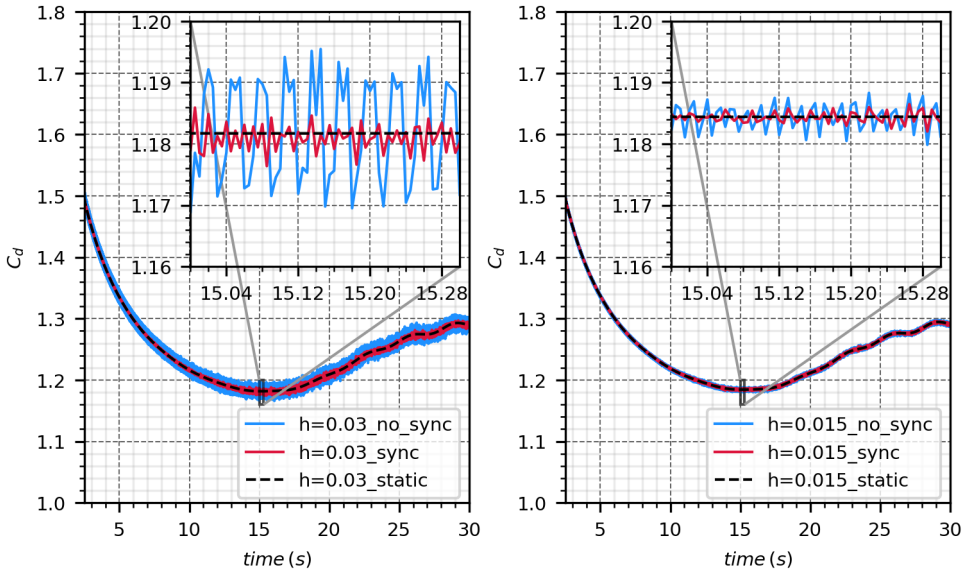


Figure 6.12: Drag coefficient over time for different test cases of the traveling cylinder at $Re = 100$, that demonstrate the alleviation of the high-frequency oscillations on the body forces.

It should be noted that residual oscillations remain due to the continuous interpolation of the particles at different grid points during the synchronization. This is evidenced by running a simulation where the mesh movement per time-step matches the particle spacing, ensuring that the particle distribution relative to the solid body remains con-

stant. In this test case, the time-step was increased to 0.02 s for both static and dynamic scenarios to ensure a fair comparison. The outcomes of this analysis are depicted in Figure 6.13.

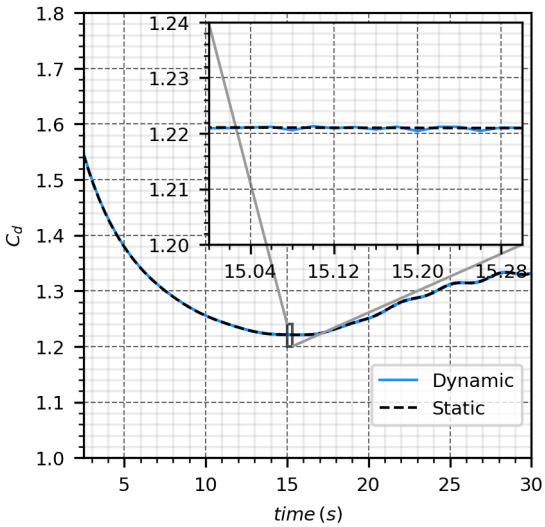


Figure 6.13: Comparison of the drag coefficient between a static and a dynamic case where the mesh displacement is equal to the particles' spacing.

With the issue now addressed, validation of the results for the *VPMFoam* solver can proceed. It is important to note that the remaining simulations for the traveling cylinder case involve the synchronization of the Lagrangian grid with the Eulerian mesh. The results are initially compared with those obtained from the static case, where the cylinder is stationary and a freestream velocity is applied. Subsequently, the dynamic case results are compared with those from a simulation performed purely in OpenFOAM (static simulation), and finally with bibliographical numerical results from Pingjian et al. [94].

Additionally, to perform a convergence test for the Eulerian mesh used in the hybrid simulations, two different meshes, named base and refined mesh, are employed. These are summarized in Table 6.5, alongside the particle spacing parameter and the time-step used in the simulations. Table 6.6 presents the results for the drag coefficient C_d, the lift coefficient C_l, and the Strouhal number. Figure 6.14 shows the drag and lift coefficients over time for the hybrid dynamic case, the hybrid static case, and the OpenFOAM case.

Finally, Figure 6.15 shows the vorticity field in two different instances as the cylinder travels. The left image is at the time $t = 20\,s$ and the right at $t = 100\,s$ where the center of the cylinder is at $x_c = -20\,m$ and $x_c = -100\,m$, respectively. The numerical boundary is represented as a black dashed circle. The vorticity field is smooth, without any inaccuracies to be observed when the vortices generated on the cylinder's surface exit the Eulerian domain.

The above results demonstrate a strong agreement between the hybrid dynamic solver, the static one, and OpenFOAM. Specifically, the hybrid dynamic solution converges to

Table 6.5: Simulation parameters for the case of the traveling cylinder at $Re = 100$, using *VPMFoam*.

Case	**Eulerian Mesh**	**Spacing** (m)	**Time-step** (s)
VPMFoam static	Base (11040)	0.03	0.005
VPMFoam static	Refined (27600)	0.015	0.0025
VPMFoam dynamic	Base (11040)	0.03	0.005
VPMFoam dynamic	Refined (27600)	0.015	0.0025

Table 6.6: Results for the case of the traveling cylinder at $Re = 100$.

Method	C_d	C_l	**Strouhal**
Pingjian et al. [94] static	1.340 ± 0.008	± 0.3130	0.165
OpenFOAM static	1.339 ± 0.008	± 0.3090	0.164
VPMFoam static (base)	1.337 ± 0.009	± 0.3225	0.165
VPMFoam static (refined)	1.337 ± 0.009	± 0.3223	0.165
VPMFoam dynamic (base)	1.337 ± 0.017	± 0.3264	0.165
VPMFoam dynamic (refined)	1.337 ± 0.009	± 0.3223	0.165

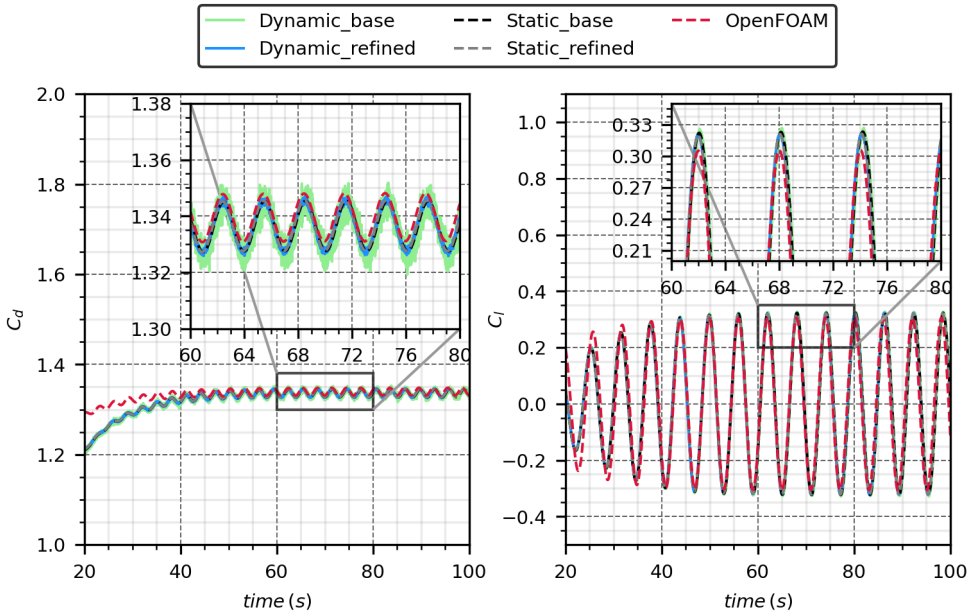


Figure 6.14: Lift and drag coefficient over time for the case of the traveling cylinder at $Re = 100$.

the same values as the hybrid static solution for the lift and drag coefficients, as well as the Strouhal number. When compared with the pure OpenFOAM results, the errors on the lift and drag coefficients are less than 4.0% and 0.2% respectively. Meanwhile, when

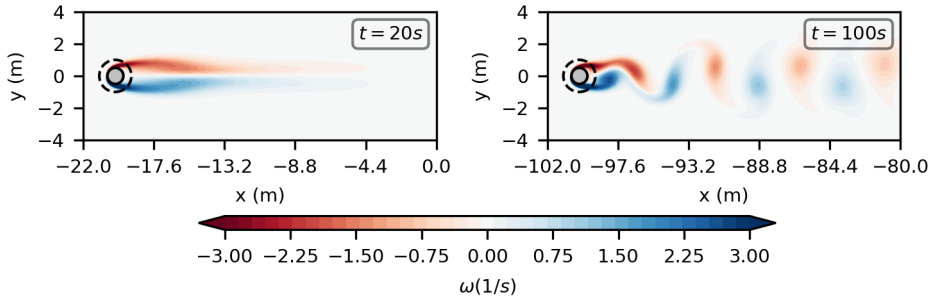


Figure 6.15: Vorticity field for the case of the traveling cylinder at $Re = 100$, using *VPMFoam*. The numerical boundary is represented with a black dashed circle line.

compared with the reference [94], the corresponding errors are 2.8% and 0.22%, respectively. Finally, *VPMFoam* predicts the same Strouhal number as the reference, equal to 0.165, while the pure OpenFOAM value is slightly lower, at 0.164.

6.3.2. Rotating cylinder-Magnus effect

The second validation case is the flow around a rotating cylinder at $Re = 200$. In this case, while the cylinder is spinning through the moving air, it experiences a lift force, known as the Magnus effect. There is a dimensionless parameter that characterizes the flow and relates the tangential velocity on the surface of the cylinder with the freestream velocity, and it is expressed as:

$$\alpha_{spin} = \frac{\Omega_{rot} R_c}{U_{inf}} \tag{6.2}$$

where Ω_{rot} is the rotating speed, R_c is the radius of the cylinder and U_{inf} is the freestream velocity. This parameter is crucial for the rotating cylinder case since it characterizes the flow. Mittal et al. [95] presented the results for the case of $Re = 200$ for value of α_{spin} from 0.0 to 5.0. It is essential that they showed that a von Kármán vortex street is developed in the flow up to $\alpha_{spin} = 1.91$. Up to that point, there is a deflection of the wake, but the periodic vortex street is present. The vortex street is not present for values larger than 1.91, and a steady solution is reached. A second region of instability is observed again at $\alpha_{spin} = 4.4$, while for $\alpha_{spin} \geq 4.8$, multiple solutions emerge, and the flow becomes more complicated. Karimi-Zindashti et al. [93] examined the case of rotating circular cylinders at $0.0 \leq \alpha_{spin} \leq 5.5$ and rotating square cylinders at $0.0 \leq \alpha_{spin} \leq 5.0$ using a deterministic vortex method. Here, *VPMFoam* will be tested in two different cases, specifically $\alpha_{spin} = 0.5$ and $\alpha_{spin} = 2.5$. This allows for comparison before and after the steady solution is reached. The results are compared with those by Mittal et al. [95], as well as with pure OpenFOAM simulations. For the OpenFOAM simulations, AMI is used. The case configuration can be seen in Figure 6.16. This figure illustrates the rotational speed as ω_{mesh}.

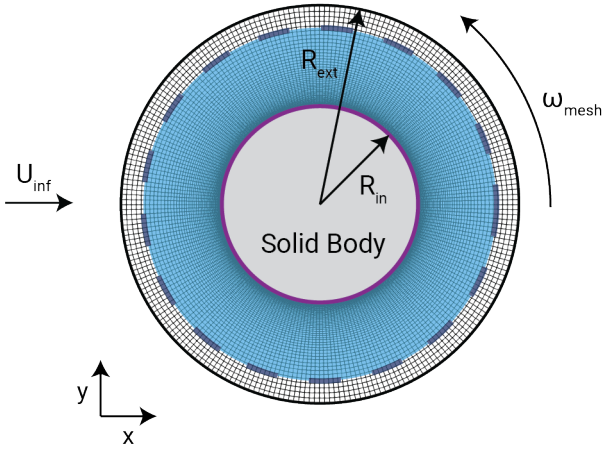


Figure 6.16: The rotating cylinder case configuration, using *VPMFoam*. The cylinder rotates counterclockwise with a rotational speed ω_{mesh}, while a freestream velocity U_{inf} acts in the $+x$ direction.

Case $\alpha_{spin} = 0.5$

For the case of $\alpha_{spin} = 0.5$, a von Kármán vortex street is expected, as Mittal et al. [95] state. Table 6.7 shows the results for the minimum and maximum lift coefficient, and for the mean drag coefficient, for *VPMFoam*, as well as the results obtained by Mittal et al. [95], and the results obtained by OpenFOAM simulations. For the case of $\alpha_{spin} = 0.5$, the base mesh of Table 6.5 is used.

Table 6.7: Results for the case of the rotating cylinder at $Re = 200$ and $\alpha_{spin} = 0.5$.

Case	$\mathbf{min(C_l)}$	$\mathbf{max(C_l)}$	$\overline{\mathbf{C_d}}$
Mittal et al. [95]	−1.910	−0.487	1.255
OpenFOAM	−1.805	−0.577	1.283
VPMFoam	−1.853	−0.520	1.276

Figure 6.17 shows the drag and lift coefficients for the hybrid case and the OpenFOAM simulations. Moreover, Figure 6.18 shows the vorticity field obtained by the hybrid solver simulations for two different time instances: one at the beginning of the simulation ($t = 10s$) and one when a periodicity in the wake is reached ($t = 60s$). As was expected, for this case, a von Kármán vortex street is present in the wake of the flow and deflected compared to the case of a non-rotating cylinder.

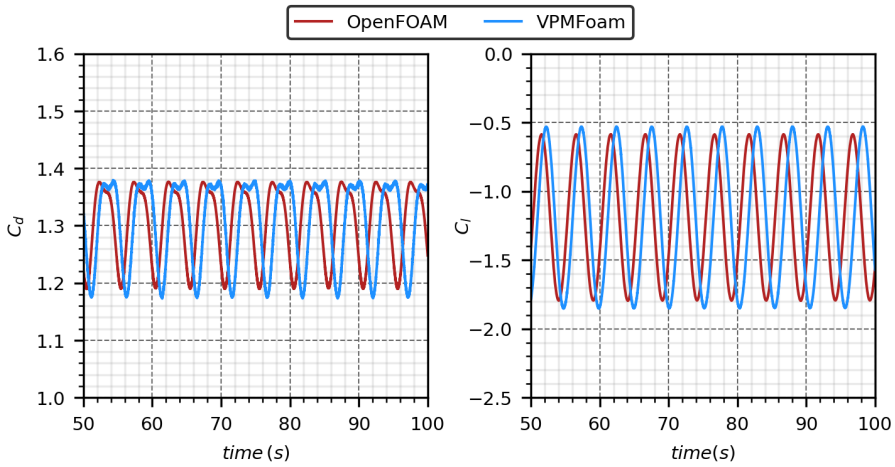


Figure 6.17: Lift and drag coefficients over time for the case of the rotating cylinder at $Re = 200$ and $\alpha_{spin} = 0.5$

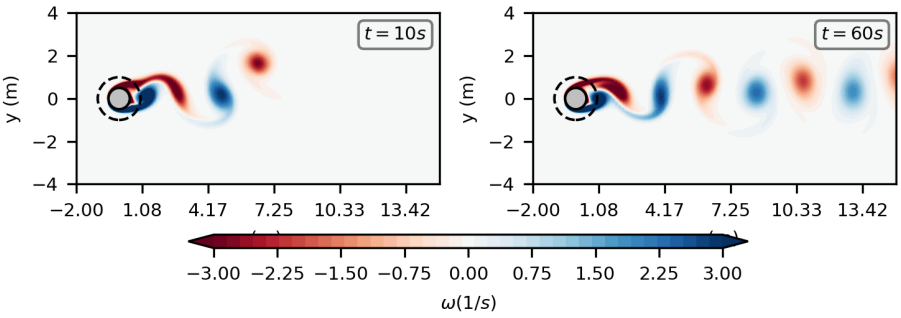


Figure 6.18: Vorticity field for the case of the rotating cylinder at $Re = 200$ and $\alpha_{spin} = 0.5$, using *VPMFoam*. The numerical boundary is represented with a black dashed circle line.

Case $\alpha_{spin} = 2.5$

For the case of $\alpha_{spin} = 2.5$, the von Kármán vortex street is not expected in the flow, as Mittal et al. [95] state. The wake should be steady and deflected. In this simulation, the refined mesh of Table 6.5 is used, while a time-step convergence test is conducted, with the time-step varying from 0.004 s to 0.00025 s. Table 6.8 shows the results for the steady lift coefficient developed on the rotating body for the present method, as well as the results obtained by Mittal et al. [95] and Karimi-Zindashti et al. [93], and the results obtained by OpenFOAM simulations.

Figure 6.19 shows the lift coefficient and a time-step convergence test for the lift coefficient. It can be seen that the hybrid solver has good agreement with the reference results from Mittal et al. [95], with the converged value deviating from the reference value by only 1.2%. Finally, Figure 6.20 shows the vorticity field in two different time instances:

one at the beginning of the simulation and one when the steady wake has been reached.

Table 6.8: Results for the case of the rotating cylinder at $Re = 200$ and $\alpha_{spin} = 2.5$.

Case	C_l
Mittal et al. [95]	-7.680
Karimi-Zindashti et al. [93]	-7.016
OpenFOAM	-7.410
VPMFoam ($\Delta t = 0.004\,s$)	-6.915
VPMFoam ($\Delta t = 0.001 s\,s$)	-7.532
VPMFoam ($\Delta t = 0.00025\,s$)	-7.583

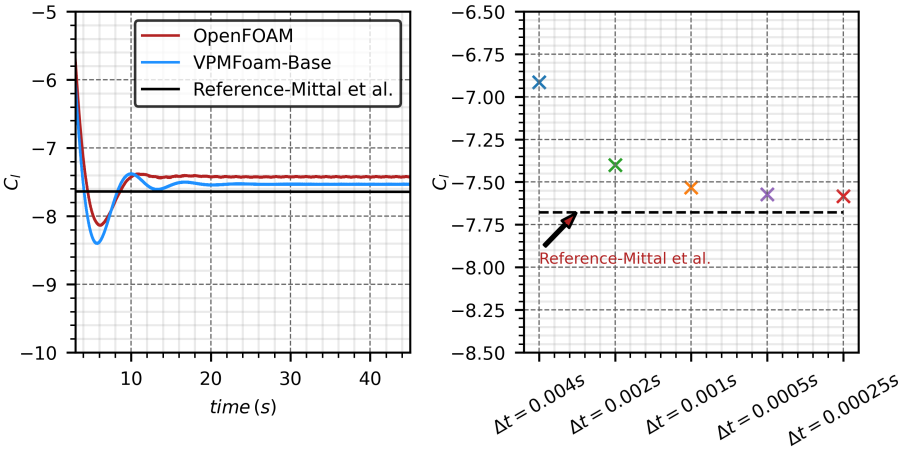


Figure 6.19: On the left is the lift coefficient over time for the hybrid case, the OpenFOAM case, and the reference [95]. The time-step convergence test for the lift coefficient is on the right compared to the reference [95].

It should be noted that in this case, the solver does not experience the issue of high-frequency oscillations in the body forces. This is because the motion of the Eulerian mesh is purely rotational around its center, ensuring that the distribution of the Lagrangian particles around the solid boundary remains constant throughout the simulation.

6.4. Conclusions

The validation of the *VPMFoam* across domains with and without solid bodies has demonstrated its accuracy and stability for various flow scenarios that involve dynamic mesh motion. In domains with no solid bodies, the solver closely matched analytical solutions for translational, rotational, oscillatory, and multi-motion cases, maintaining error levels below 2%. When solid bodies are introduced, the solver effectively handled both traveling and rotating cylinders, achieving strong agreement with reference data for mean lift

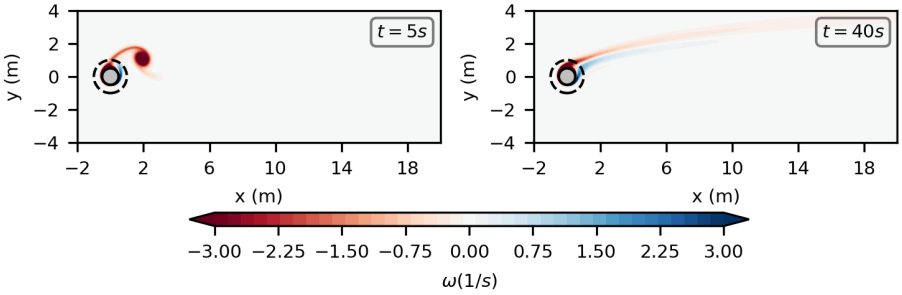


Figure 6.20: Vorticity field for the case of the rotating cylinder at $Re = 200$ and $\alpha_{spin} = 2.5$, using *VPMFoam*. The numerical boundary is represented with a black dashed circle line.

and drag coefficients, as well as the Strouhal number. To address high-frequency oscillations in force predictions, adjustments in Lagrangian particle resolution and synchronization with the Eulerian mesh movement successfully minimized these effects, bringing the dynamic simulations in close alignment with static cases. These results confirm the solvers capability in accurately simulating flows with dynamic mesh motion, validating it as a reliable tool for complex two-dimensional flow scenarios.

7

Validation - Multibody cases

In the preceding two chapters, the hybrid solver was rigorously tested in both static and dynamic scenarios involving single-body simulations. This chapter introduces and validates another capability of the solver: its ability to handle multibody cases. This is achieved by constructing distinct Eulerian subdomains for each solid body, allowing the cases to be processed independently. The interconnection between these regions is facilitated by the Lagrangian solver. The methodology behind this setup is outlined here, and the effectiveness of this feature of the hybrid solver is demonstrated through its application to various geometric configurations of two cylinders.

Parts of this chapter have been published in: **R. Pasolari**, C.J. Ferreira, A. van Zuijlen, "Flow around a pair of 2D cylinders using a hybrid Eulerian-Lagrangian solver", Journal of Physics: Conference Series 2767 (5), 052006 (2024), DOI 10.1088/1742-6596/2767/5/052006. [96]

7.1. Introduction

Multibody problems are frequently encountered in many applications within the field of external aerodynamics. For instance, in the wind energy sector, wind turbines are integral components of a larger system-the wind farm. In addition, a wind turbine itself can be considered and modeled as a multibody problem, with the rotor, tower, and each blade treated as discrete components. Similarly, helicopters, car aerodynamics, and the flow around buildings in a city can also be analyzed using multibody problem techniques. In the case of helicopters, discrete parts can be modeled separately for analysis purposes. Likewise, cars and buildings present multibody problems when studied in groups. The study of multibody problems is crucial because the interaction between different objects significantly alters airflow dynamics compared to standalone objects. Understanding these interactions is essential for optimizing design and performance in various aerodynamic applications.

Multibody problems are most often treated numerically, as conducting experiments can be prohibitively expensive. Even numerical simulations of multibody problems can be extremely costly and sometimes unaffordable. For example, a single body, such as a wind turbine, can be fully resolved using the Eulerian approach like a FVM or FEM solver [97]. However, it is very rare to fully resolve more than one body using these methods due to computational expense. To reduce computational costs, wind turbines in a wind farm are often modeled as actuators [98]. Recently, Lagrangian solvers have become popular for such simulations, offering accelerated solutions but often sacrificing boundary layer resolution [99]. Additionally, the Blade Element Momentum (BEM) method is a commonly used low-fidelity approach that provides fast results in wind turbine simulations [100].

7.2. Multibody problems with *VPMFoam*

Considering the above challenges, *VPMFoam* can help bridge the gap by combining the fast calculations provided by particle methods with the high accuracy of Eulerian solvers in resolving boundary layers. Additionally, this hybrid solver is adept in handling multibody simulations efficiently. Specifically, as shown in Figure 7.1, each cylinder can be treated as an independent OpenFOAM case, solved separately. Particles covering the entire computational domain establish connections between different regions, providing boundary conditions for each separate case. This approach accelerates simulations by assigning each Eulerian case to a different CPU core. Another significant advantage of using *VPMFoam* for multibody simulations is the ease of discretizing the computational domain. In pure Eulerian solvers, mesh generation can often be a challenging and time-consuming task, particularly in multibody cases where objects are positioned close to one another. *VPMFoam* simplifies this process by creating simple meshes around each object independently. Additionally, when some objects are identical or similar, mesh generation becomes even simpler through duplication, combined with translations, rotations, and scaling. This flexibility allows for rapid setup of the computational domain, as the remaining discretization is handled by particles, which are easily created in the Eulerian domains and are free to move.

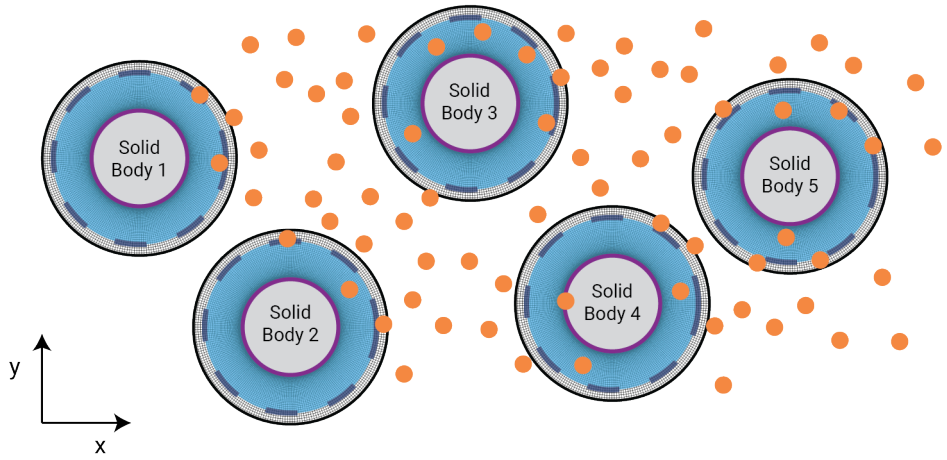


Figure 7.1: An example of a multibody problem involves five circular cylinders in an arbitrary configuration. In the context of the hybrid solver, each cylinder is structured as an independent OpenFOAM case, which can run independently and in parallel on different CPU cores. The particle solver interconnects these different regions by providing boundary conditions to each one, allowing for efficient and coordinated simulations across multiple bodies.

7.3. Validation cases

The validation of this feature of the solver will be conducted through a series of tests involving different configurations of two cylinders. Initially, the two cylinders are positioned in a tandem arrangement (Figures 7.2a and 7.2c). Then, the two cylinders are positioned in a staggered arrangement (Figures 7.2b and 7.2d). The cases illustrated in Figures 7.2a and 7.2b present significant overlap between the Eulerian meshes of the two cylinders. This overlap makes these cases particularly useful for evaluating the solver's performance in handling closely interacting bodies.

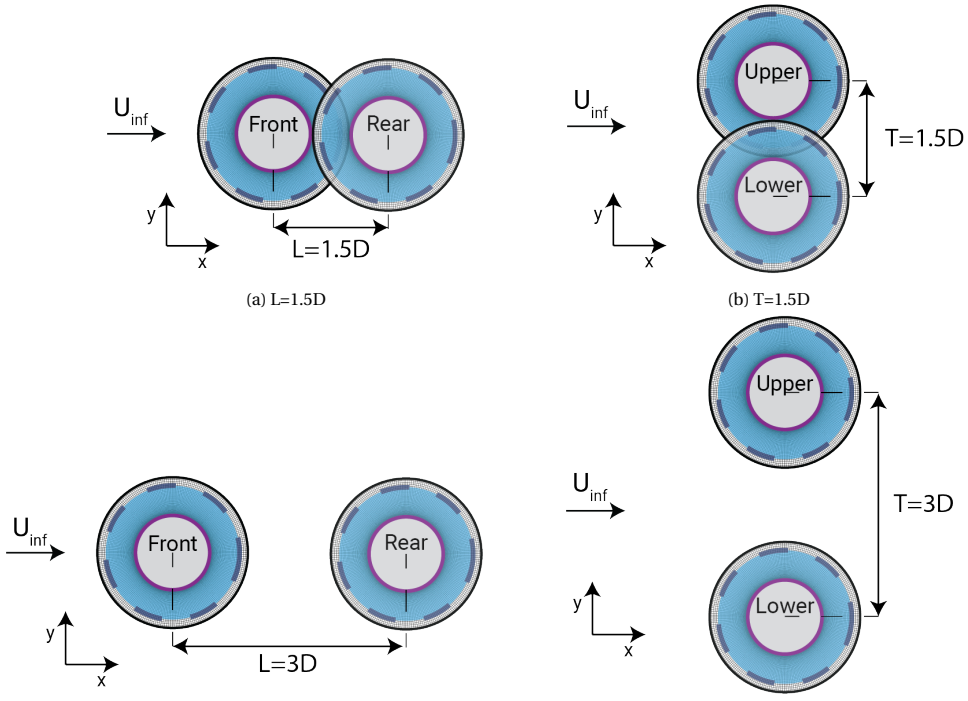
In these cases, the correction of the Lagrangian particles in the overlapping region is performed only once to avoid introducing artificial circulation into the flow. The particles in this region are corrected using the first labeled Eulerian subdomain. Another (more accurate) approach would be to correct the particles according to their proximity to the solid body of each Eulerian subdomain.

In all cases, the Reynolds number is set to $Re = 200$. The mesh used is comprised solely of hexahedra, as illustrated in Figure 7.3. The parameters for the Eulerian solver, the Lagrangian solver, and the general simulation parameters are consistent across all cases and are summarized in Table 7.1.

7.4. Results

7.4.1. Tandem arrangement

For the tandem arrangement of the two cylinders, the results are compared with the corresponding results obtained by Skonecki et al. [101] that used *STAR-CCM+* for their simulations, and by Meneghini et al. [102] where a FEM code was used. Table 7.2 sum-



(a) L=1.5D (b) T=1.5D

(c) L=3D (d) T=3D

Figure 7.2: The four different configurations of the pair of cylinders. Subfigures (a) and (b) show extreme cases where the Eulerian meshes of the two cylinders overlap up to the solid body, while subfigures (c) and (d) illustrate configurations where the cylinders are further apart, without an overlap of their meshes.

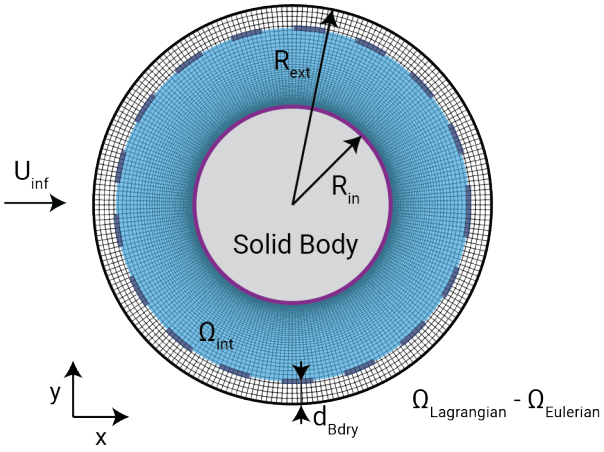


Figure 7.3: The configuration of each cylinder, illustrating the Eulerian, Lagrangian, and interpolation regions, as well as the mesh of the Eulerian domain.

Table 7.1: Simulation parameters for the two cylinder cases at $Re = 200$, using *VPMFoam*.

Parameter	Symbol	Value	Dimension
Kinematic viscosity	ν	0.005	m^2/s
Cylinder's radius	R_{in}	0.5	m
Eulerian mesh density	N_{cells}	11040	–
Eulerian domain radius	R_{ext}	1.0	m
Eulerian time-step	Δt_E	0.008	s
Vortex particles spacing	h	0.027	m
Lagrangian time-step	Δt_L	0.008	s
Interpolation domain offset	d_{Bdry}	$3 \cdot h$	m

marizes the results for both cases in tandem arrangement, comparing the mean drag coefficient and the Strouhal number of the current method with the bibliographical results. It can be seen that the results of the hybrid solver show a strong agreement with the references, for both the front cylinder (denoted with the subscript F) and the rear cylinder (denoted with the subscript R). Figure 7.4a, illustrates the vorticity field for the two tandem arrangements. It can be observed that the vorticity that is produced in the first cylinder, can accurately be convected to the rear cylinder, and then downstream, without any inaccuracies, neither in the case that the two cylinders are placed very close. Finally, Figure 7.5a, shows the aerodynamic coefficients over time, for both the front and rear cylinders.

Table 7.2: Mean drag coefficient and Strouhal number results for the tandem arrangement of the two cylinders. The results are compared with literature values.

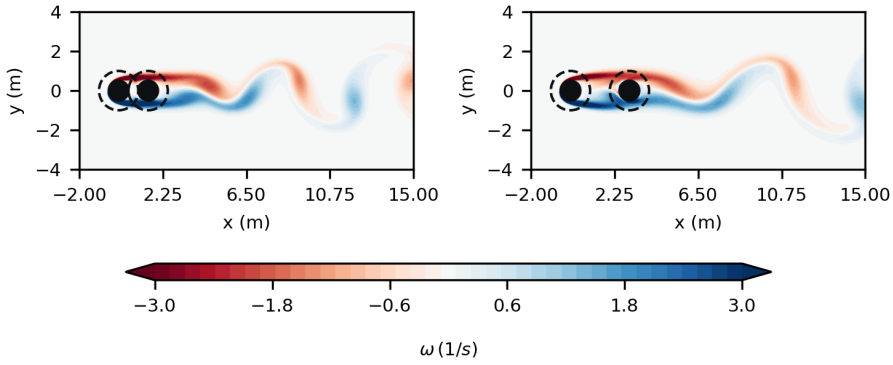
Method	$L = 1.5D$				$L = 3.0D$			
	$\overline{C}_{d,F}$	St_F	$\overline{C}_{d,R}$	St_R	$\overline{C}_{d,F}$	St_F	$\overline{C}_{d,R}$	St_R
Skonecki et al. [101]	1.09	0.170	-0.20	0.170	1.02	0.129	-0.12	0.129
Meneghini et al. [102]	1.06	0.167	-0.18	0.167	1.00	0.125	-0.08	0.125
VPMFoam	1.06	0.167	-0.18	0.167	1.00	0.128	-0.13	0.128

7.4.2. Staggered arrangement

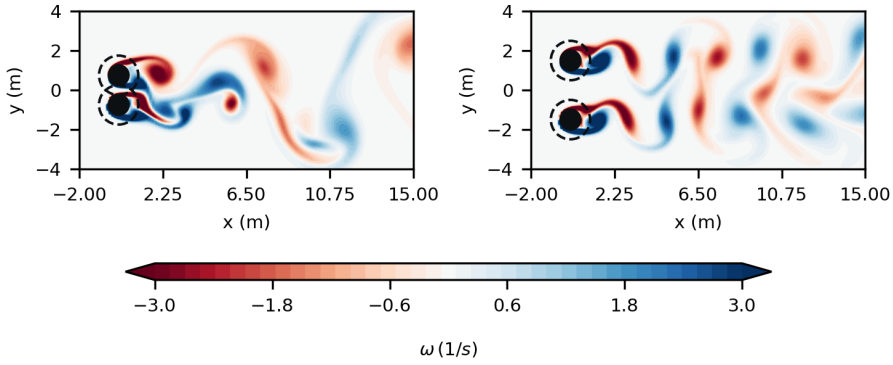
For the staggered arrangement, the results are compared with the same references. Table 7.3 summarizes the results for both cases in staggered arrangement, comparing the mean drag coefficient and the Strouhal number obtained by *VPMFoam* with the bibliographical results. For both the upper cylinder (denoted with the subscript U) and the lower cylinder (denoted with the subscript L), the mean drag coefficient, as well as the Strouhal number, falls between the results obtained from the references. Figure 7.4a, illustrates the vorticity field, while Figure 7.5a, shows the aerodynamic coefficients over time, for both the upper and lower cylinders.

Table 7.3: Mean drag coefficient and Strouhal number results for the staggered arrangement of the two cylinders. The results are compared with literature values.

Method	$T = 1.5D$				$T = 3.0D$			
	$\overline{C}_{d,U}$	St_U	$\overline{C}_{d,L}$	St_L	$\overline{C}_{d,U}$	St_U	$\overline{C}_{d,L}$	St_L
Skonecki et al. [101]	1.55	0.201	1.57	0.204	1.56	0.214	1.56	0.214
Meneghini et al. [102]	1.32	-	1.32	-	1.34	0.174	1.34	0.174
VPMFoam	1.46	0.209	1.48	0.197	1.41	0.198	1.41	0.198



(a) Tandem arrangement.

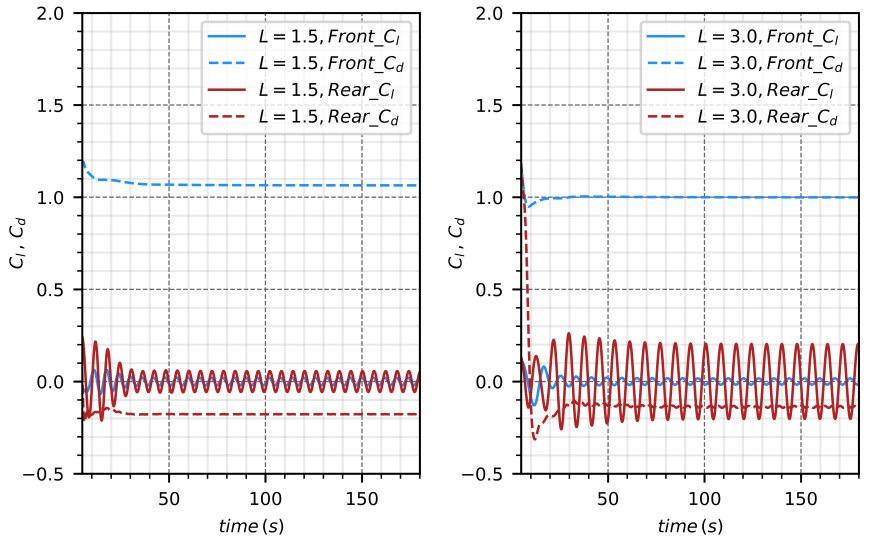


(b) Staggered arrangement.

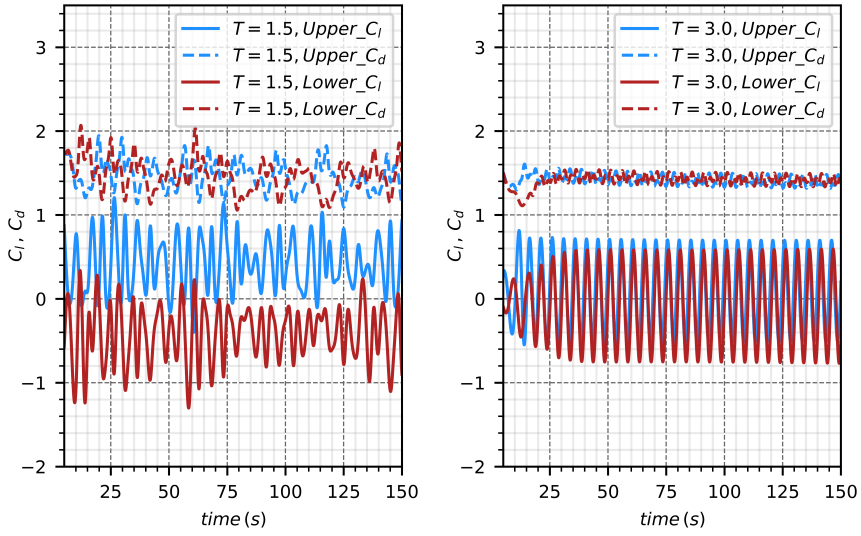
Figure 7.4: Screenshots of the vorticity field for the tandem and the staggered arrangements using *VPMFoam*. The circle formed by the black dashed line [– – –] defines the Eulerian mesh region.

7.5. Conclusions

The results presented above demonstrate the capability of *VPMFoam* to handle multi-body problems, producing outcomes consistent with pure Eulerian solvers. Even in cases where the two meshes overlap significantly (such as when the cylinders are placed



(a) Tandem arrangement.



(b) Staggered arrangement.

Figure 7.5: Time history of the aerodynamic coefficients for the tandem and the staggered arrangements using *VPMFoam*.

very close to each other), the results align with existing literature. The versatility of this approach allows for the extension of the problem to incorporate additional cylinders and multiple moving bodies, establishing it as a robust tool in the field of CFD.

8

Performance evaluation

In the previous three chapters, the hybrid solver was validated in stationary cases, both with and without solid boundaries, in dynamic mesh simulations with and without solid boundaries, and in multibody cases. Now that the solver has been validated, this chapter focuses on its computational efficiency. Specifically, the computational efficiency of the VPMFoam solver is compared to pure OpenFOAM simulations, using the flow around a cylinder case, as presented earlier. Since a direct comparison is challenging, two factors are considered in this chapter: the accuracy in computing the aerodynamic forces and the conservation of wake structures in the wake region.

8.1. Introduction

The *VPMFoam* is designed not only to take advantage of eliminating Eulerian solvers' artificial diffusion in the wake and to harness the flexibility of Lagrangian solvers, but also to maintain computational efficiency. This efficiency implies that the solver should provide high accuracy in both aerodynamic forces and wake structures, while keeping the computational cost comparable to, or even lower than, that of a fully Eulerian or vortex particle simulation.

For instance, running an external aerodynamics case in OpenFOAM can result in accurate predictions of aerodynamic forces, often without a significant computational burden. However, when the focus shifts to preserving wake structures with minimal artificial diffusion, the computational cost can rise considerably.

On the other hand, vortex methods offer rapid computations for vortex interactions with little to no numerical diffusion. However, accurately predicting aerodynamic forces using this method is more challenging. A substantial number of particles are needed in the boundary layer to capture viscous effects, which can make the method computationally expensive.

Here, the performance of *VPMFoam* will be evaluated by comparing its results and computational cost against those of OpenFOAM. A direct comparison with a pure VPM solver is not included, as additional solvers would be required to enforce boundary conditions on the solid surface, which lies beyond the scope of this dissertation. For the evaluation, the flow around a cylinder at $Re = 550$, as presented in Chapter 5, is considered.

The comparison between the two solvers is not straightforward. The key question is: "Which part of the flow requires higher accuracy?". Therefore, accuracy will be assessed based on two criteria:

- Aerodynamic coefficients.
- Preservation of wake structures.

This distinction is crucial, as there is no singular method to assess the performance of the code. Pure Eulerian solvers may offer good accuracy in aerodynamic force predictions, but the wake far downstream of the object may become heavily diffused. Consequently, both factors must be considered to fully evaluate the hybrid solvers objectives. To ensure a fairer comparison, the following requirements will be enforced:

- Two Eulerian simulations will be presented (for the flow around a single cylinder case). In the first, the only requirement is to achieve converged aerodynamic forces. In the second, wake structures must also be preserved, up to 40 diameters downstream.
- One hybrid simulation will be presented, ensuring the convergence of aerodynamic forces to a constant value.
- The near-wall mesh in the hybrid simulation will be identical to that of the full Eulerian simulation.

- Since the computational domain in the hybrid simulation is unbounded, a limit will be imposed on the particles in the freestream direction. Particles crossing this limit will be excluded from the advection and diffusion processes but will continue to advect downstream solely with the freestream velocity.
- For all simulations, the simulation time will be set to $t_{sim} = 150s$, as found in Chapter 5, where both simulations reach a periodic phase by this time.
- Given that the different solvers start from different initial phases, computational time will be measured for the entire simulation and for four periods during the periodic phase to ensure a fair comparison.
- The Eulerian part of the hybrid solver has the potential to be parallelized using OpenFOAMs capabilities. However, this has not been implemented yet in the solver. Therefore, assumptions will be made to estimate performance for a case running on multiple cores.

8.2. Computational resources

For all simulations presented in this study, the DelftBlue cluster [83] was utilized. OpenFOAM simulations were run exclusively on CPUs, using the type-b compute partition of DelftBlue, which is equipped with *Intel Xeon Gold 6448Y 32C 2.1 GHz* CPUs. On the other hand, *VPMFoam* required GPU nodes, as parts of the solver run on GPUs (can run also in CPUs). For this purpose, the GPU type-b partition, featuring both *Intel Xeon Gold 6448Y* CPUs and *NVIDIA Tesla A100* GPUs, was selected. Detailed specifications for both partitions are as follows:

- **Compute type-b (CPU-only nodes for OpenFOAM)**
  - CPU cores per node: 64 (across two sockets)
  - CPU: 2x Intel Xeon Gold 6448Y 32C 2.1 GHz
  - RAM: 256 GB DDR4 per node (2 NUMA nodes, 128 GB each)
- **GPU type-b (GPU nodes for *VPMFoam*)**
  - CPU cores per node: 64 (across two sockets)
  - CPU: 2x Intel Xeon Gold 6448Y 32C 2.1 GHz
  - RAM: 512 GB DDR4 per node (4 NUMA nodes, 128 GB each)
  - GPU per node: 4x NVIDIA Tesla A100 (80 GB each)
  - Each GPU:
    - FP32 (Single-Precision): Up to 19.5 TFLOPS
    - FP64 (Double-Precision): Up to 9.7 TFLOPS
    - CUDA cores: 6,912

These nodes were selected because they are equipped with the same CPUs, allowing for a direct comparison of the two software packages under identical computational resources. For *VPMFoam*, the different processes occurring at each time-step are listed below, along with the corresponding hardware used and software implemented for these simulations. It should be noted that other options are available but are not used in this particular setup:

- **Advection:**
  - Hardware: GPU
  - Method: Fast Multipole Method (FMM)
  - Other options: Direct calculations using OpenMP on CPU, FMM on CPU, or direct calculations on GPU using CUDA.
- **Diffusion with redistribution:**
  - Hardware: Partially in CPU (using OpenMP) and partially in GPU (using CuPy)
  - Other options: Full execution in CPU (serial) without OpenMP, or both parts running entirely on the CPU.
- **Eulerian (OpenFOAM component):**
  - Hardware: CPU
  - Software: OpenFOAM
  - Other options: Not applicable; only available on CPU.
- **Correct Particles:**
  - Hardware: CPU
  - Software: Serial execution in Python
  - Other options: No alternatives available.
- **Population and wake control:**
  - Hardware: CPU
  - Software: Serial execution in Python
  - Other options: No alternatives available.

8.3. OpenFOAM case: converged aerodynamic forces

The computational domain for this simulation was previously introduced in Chapter 5; however, it is presented again here in Figure 8.1 for reference.

For this simulation, the 4^{th} refinement level from Table 5.4 is used, which comprises 159,982 cells. This refinement level provides converged values for the mean drag coefficient, maximum lift coefficient, and the Strouhal number. The simulation is performed with five different levels of parallelization, specifically using 1, 8, 16, 32 and 64 cores.

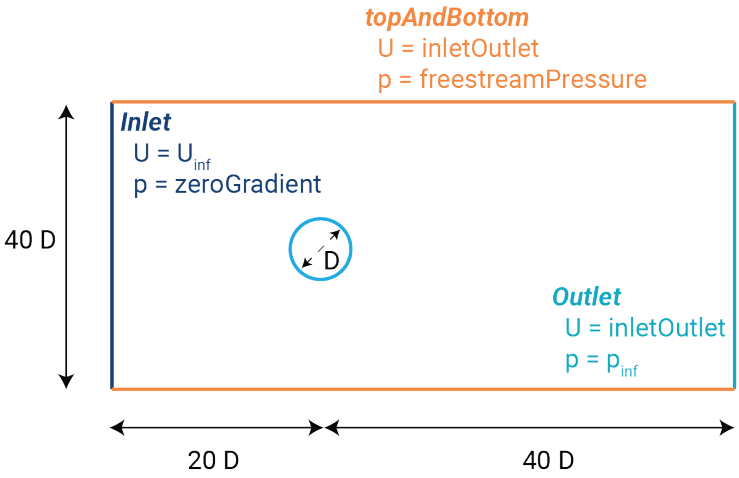


Figure 8.1: The geometry and boundary conditions for the pure Eulerian simulation of flow around a cylinder at $Re = 550$.

The *scotch* decomposition technique [103] is employed to evaluate the speedup achieved through multi-core processing and to determine the maximum possible efficiency gain. Moreover, in all cases, the CPUs are selected from the same node within the cluster to eliminate any inter-node communication latencies. This ensures that the simulation benefits from faster intra-node memory access and avoids potential delays associated with network communication between different nodes.

Figure 8.2 illustrates the computational time for the simulation across the five levels of parallelization (varying core counts). The left subfigure shows the computational cost per time-step, while the right subfigure depicts the total simulation time for each case. Speedup achieved for each core configuration is annotated in black on the right subfigure. Additionally, the computational time for four periods within the periodic phase of the flow is indicated by the hatched region in the left plot and highlighted with a boxed hatch pattern in the right plot, with the corresponding speedup for this time period annotated in white.

The maximum speedup is observed with 32 cores, achieving a total speedup of approximately ×15 for the entire simulation and ×11.5 for the last four periods. The corresponding simulation times for the full run and the four-period segment with 32 cores are 1,730 s and 394 s, respectively. With 64 cores, the speedup is lower, likely due to communication overhead or uneven distribution of tasks, such as the boundary layer, across the cores.

8.4. *VPMFoam* case

For the hybrid case, the computational domain was introduced previously in Chapter 5, and is presented again here in Figure 8.3. This domain corresponds to the inner mesh of the Eulerian grid used earlier and consists of 20,592 cells. The particle spacing is set

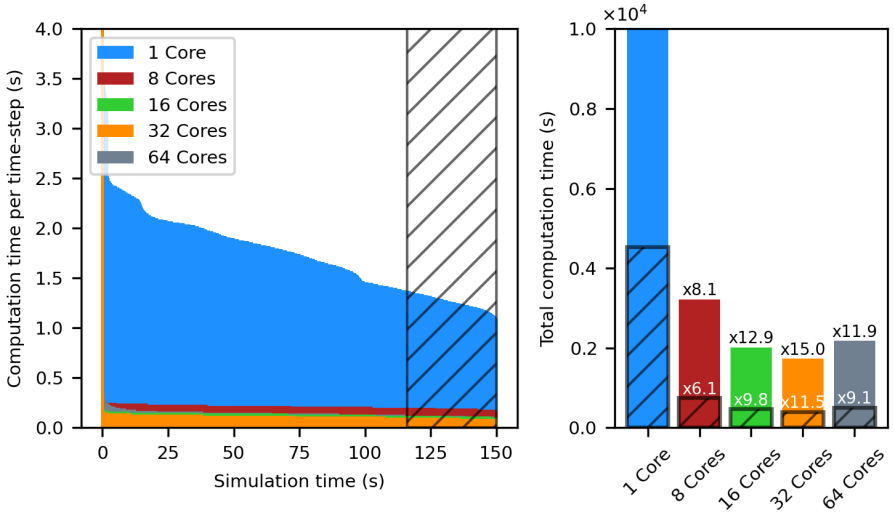


Figure 8.2: Computational time for the Eulerian simulation with 1, 8, 16, 32 and 64 cores (for the case of converged aerodynamic forces). The left subfigure shows the computational cost per time-step, while the right subfigure depicts the total simulation time, with speedup values annotated in black. The computational time for four periods in the periodic phase is marked by the hatched region in the left plot and a boxed hatch pattern in the right plot, with the corresponding speedup annotated in white.

to $h = 0.035\,m$, which approximately matches the size of the cells near the numerical boundary. The time-step used in this simulation is the same as in the pure Eulerian case.

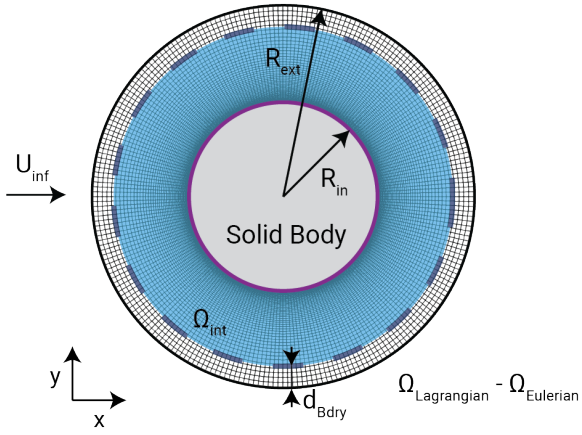


Figure 8.3: The cylinder configuration used for the performance evaluation of *VPMFoam*.

The speedup for the hybrid case can be achieved through several approaches, in-

cluding:

- Increasing the number of CPU cores handling particle redistribution and diffusion, which speeds up these processes.
- Increasing the number of CPU cores allocated to the Eulerian part of the hybrid solver, which would also accelerate the simulation. However, as mentioned earlier, this feature is not yet implemented, but it could be considered for future development.
- Increasing the Lagrangian time–step during the evolution process, which reduces the computational time. In particular, the Lagrangian time-step can be increased while keeping the Eulerian time-step the same, allowing for multiple Eulerian substeps within one Lagrangian step, as discussed in Chapter 4. Caution is required to ensure that the solution's accuracy is not compromised.

To illustrate the potential for accelerating the hybrid simulation, a series of configurations will be evaluated:

- 1 CPU, 1 GPU, $\Delta t_L = \Delta t_E$.
- 8 CPUs, 1 GPU, $\Delta t_L = \Delta t_E$.
- 16 CPUs, 1 GPU, $\Delta t_L = \Delta t_E$.
- 32 CPUs, 1 GPU, $\Delta t_L = 1\Delta t_E$.
- 32 CPUs, 1 GPU, $\Delta t_L = 2\Delta t_E$.
- 32 CPUs, 1 GPU, $\Delta t_L = 3\Delta t_E$.

Figure 8.4 shows the computational time for each part of the solver (advection, diffusion with redistribution, Eulerian, correct particles, and population and wake control) at every time-step for the case with 8 CPUs, 1 GPU, and $\Delta t_L = \Delta t_E$. Figure 8.5 presents the total computation time for all cases, highlighting the time spent on each component of the solver and the speedup achieved for the entire simulation and the last four periods.

It can be observed that the most computationally expensive components are the Eulerian part, advection, and diffusion with redistribution. The advection part, which always runs on a single GPU, does not show speedup but becomes a significant contributor to the total computational time as the number of particles increases. Diffusion, while minimal with a lower number of particles, increases significantly as the number of particles grows. However, by using more CPU cores, the time required for diffusion decreases. As noted, diffusion is divided between CPU and GPU, and increasing the number of CPU cores reduces the CPU portion, while the GPU portion remains consistent beyond 8 CPU cores.

The Eulerian part is the most computationally expensive in the hybrid simulation. However, OpenFOAM's potential for parallel execution in this part has not been utilized here. Therefore, Figure 8.6 presents an estimate of the theoretical total computational time if OpenFOAM were parallelized. This is a rough estimation based on theoretical

speedups from multi-core execution using Message Passing Interface (MPI), but even with more conservative speedups, the Eulerian part would cease to be the most computationally expensive component.

It is important to note that the total number of particles in these simulations is approximately 660,000, which is a large number. No merging techniques were applied, meaning all particles have the same size, and many are concentrated in the wake region.

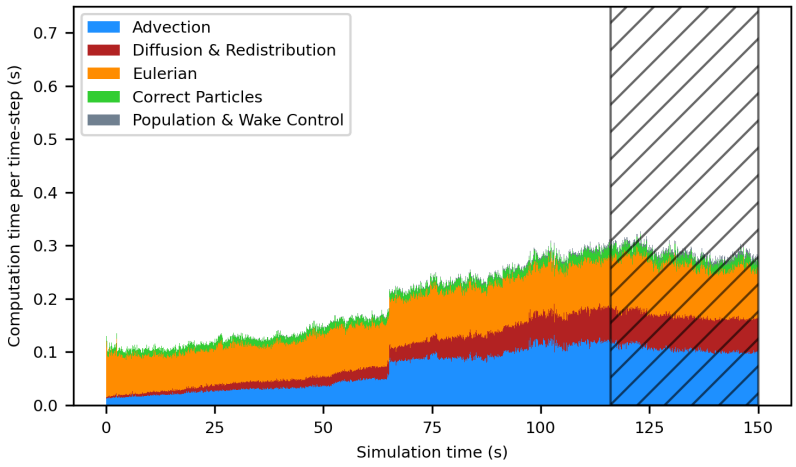


Figure 8.4: Computational time for each component of the *VPMFoam* solver (advection, diffusion with redistribution, Eulerian, correct particles, and population and wake control) at each time-step for the case of 8 CPUs, 1 GPU, and $\Delta t_L = \Delta t_E$.

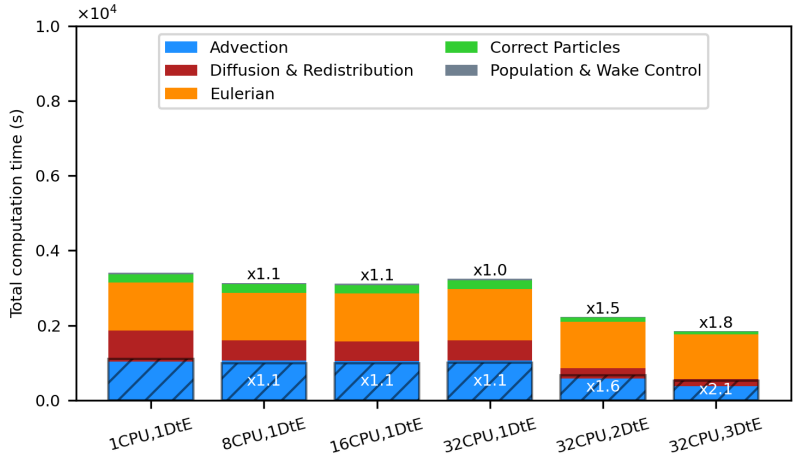


Figure 8.5: Total computational time for all *VPMFoam* cases, showing the time required for each component of the solver and the speedup achieved for the entire simulation and the last four periods.

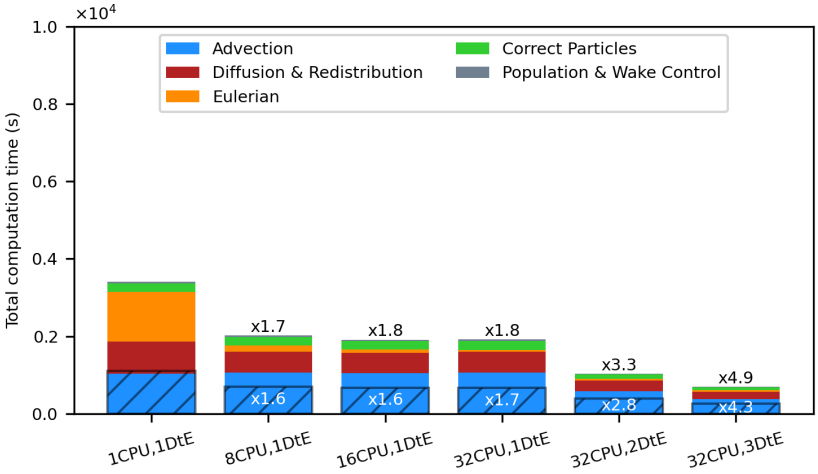


Figure 8.6: Theoretical total computational time for *VPMFoam* if OpenFOAMs parallel capabilities for the Eulerian part were fully utilized, based on rough estimations of possible speedups from multi-core runs using MPI.

In this case, it is ensured that the aerodynamic coefficients have converged, and their values are presented in Chapter 5. However, as previously mentioned, to fully assess the accuracy of the code compared to OpenFOAM, an additional factor must be considered: the preservation of vorticity in the far-field regions.

To address this, the vorticity in the far-field region is illustrated in Figure 8.7. This figure shows a comparison between the pure Eulerian solver (left) and the hybrid simulation (right). It can be observed that while the aerodynamic forces have converged to constant values, the wake vorticity structures in the Eulerian simulation become highly diffused. Even at $x = 30.0\,m$, the structures are significantly diffused, and beyond this point, they lose their vorticity structure even further.

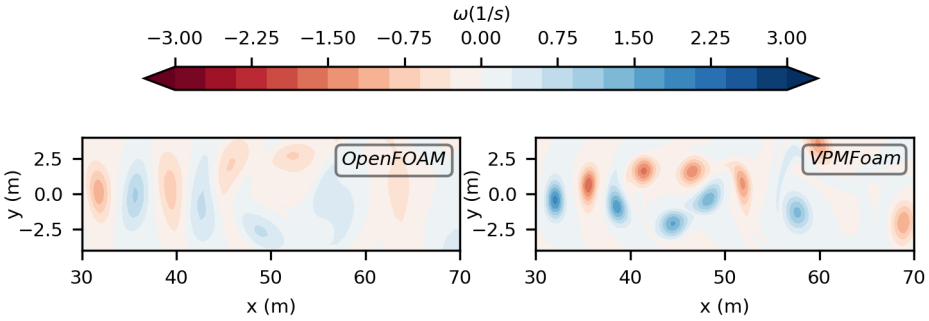


Figure 8.7: Far-field vorticity comparison between the pure Eulerian solver (left) and *VPMFoam* (right), in the case where aerodynamic forces are the only measure of importance for the pure Eulerian simulation.

8.5. OpenFOAM case: minimal wake diffusion

In order to preserve the vorticity structures in the wake region for the pure Eulerian simulation, similar to the hybrid case, an additional simulation is conducted. This time, the wake region is more refined. Specifically, by maintaining the same resolution near the solid body and increasing the resolution in the wake region, the new mesh comprises 718,818 cells. The far-field vorticity field for this case is shown in Figure 8.8, demonstrating that the pure Eulerian simulation now approaches the accuracy of the hybrid solver in preserving vorticity structures. For this case, the simulation time for different numbers of CPU cores is shown in Figure 8.9.

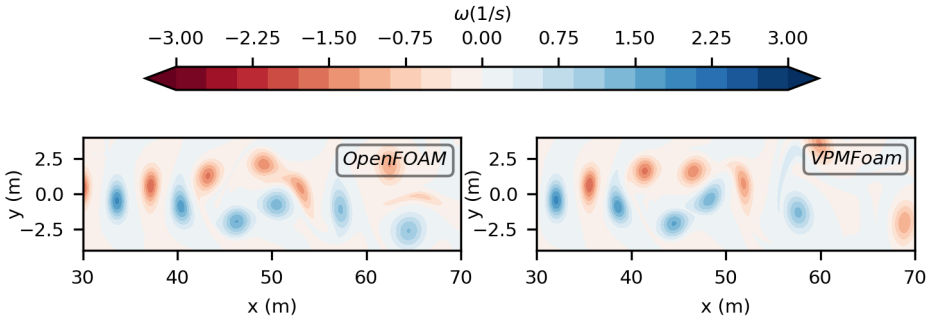


Figure 8.8: Far-field vorticity comparison between the pure Eulerian solver (left) and *VPMFoam* (right), for the case where preserving wake structures is also a focus in the pure Eulerian simulation.

The maximum speedup is observed with 64 cores, likely due to the increased workload per core and reduced relative communication overhead. The speedup is approximately ×54, with the full run taking 3,155 s and the four-period segment taking 722 s. The speedup shows a converging trend as the number of cores increases from 8 to 64, indicating that the limit is likely reached around this point. Beyond 64 cores, the speedup would likely decrease, as additional cores would require CPUs from different nodes, introducing inter-node communication overhead.

8.6. Conclusions

Considering the results above, *VPMFoam* demonstrates great efficiency compared to OpenFOAM, particularly in cases where eliminating artificial diffusion in the wake is crucial. The hybrid solver can complete the full simulation in 1,911 s using multiple CPU cores and the same time-steps for both the Eulerian and Lagrangian parts. This time is further reduced to 1,029 s when the Lagrangian time-step is twice as large as the Eulerian, and to 690 s when it is three times larger. This performance is impressive, especially when compared to the best time achieved by the pure Eulerian simulation, which was 1,730 s with 32 cores for the first case, and 3,155 s for the second, more demanding case.

Additionally, the ease of grid generation in the hybrid solver highlights its efficiency even further. In Eulerian simulations, creating the computational grid often requires special meshing techniques, particularly when dealing with complex geometries, whereas in the hybrid solver, only the region near the solid body needs to be meshed. The rest of

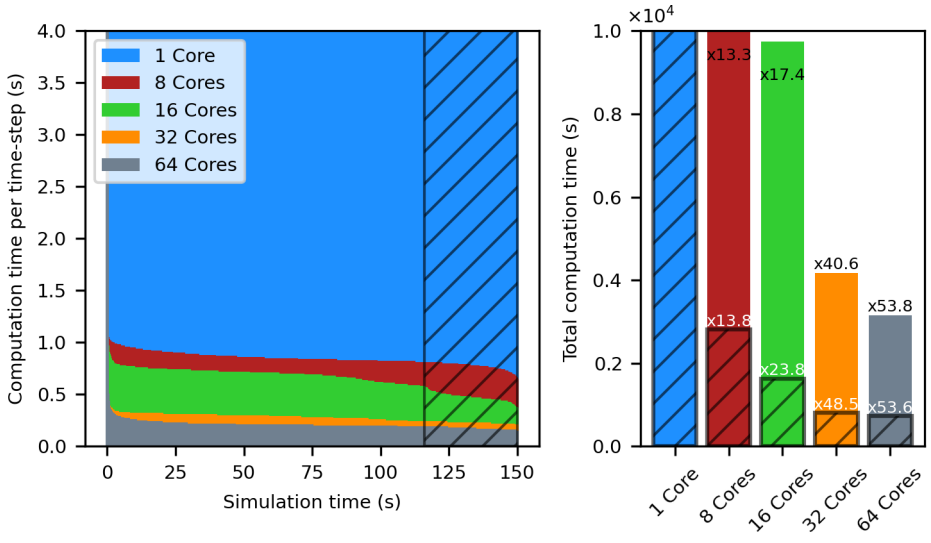


Figure 8.9: Computational time for the Eulerian simulation using 1, 8, 16, 32, and 64 cores, in the case where preserving vorticity structures in the wake is important. The left subfigure shows the computational cost per time-step, while the right subfigure displays the total simulation time, with speedup values annotated in black. The computational time for four periods in the periodic phase is indicated by the hatched region in the left plot and highlighted with a boxed hatch pattern in the right plot, with the corresponding speedup annotated in white.

the computational domain is handled by freely moving particles, simplifying the process significantly.

This advantage becomes even more pronounced in multibody simulations. In the hybrid solver, different regions can be meshed separately and solved in parallel on different CPU cores without significantly increasing the computational effort. In contrast, OpenFOAM requires more demanding meshing, often involving overset mesh techniques that can substantially increase computational costs. Furthermore, in OpenFOAM, additional bodies increase the number of cells, which in turn leads to longer computational times.

For pure Eulerian simulations, increasing the number of CPU cores is necessary to achieve speedup, especially for very large problems. As the problem size grows, the need for using multiple nodes becomes inevitable, which introduces inter-node communication overhead. However, for the hybrid solver, it was demonstrated that even with just 8 CPU cores, efficient performance was achieved, and the simulations were completed with the use of a single GPU.

This suggests that for larger problems, the number of CPUs may not be a limiting factor for speedup in the hybrid solver. Instead, adding extra GPUs would significantly enhance the speedup, making the hybrid approach highly scalable for larger-scale simulations.

8.7. Further speed-up recommendations

The *VPMFoam* solver has already demonstrated great efficiency in terms of both computational cost and accuracy. However, there is potential to scale up its performance even further, especially for large-scale simulations. Below are strategies for achieving further acceleration of the solver:

- **Parallel GPUs for convection and diffusion using CuPy:** Currently, the advection and diffusion processes are executed on a single GPU. By distributing these operations across multiple GPUs, the workload can be shared, enabling parallel processing of particle interactions and diffusion steps. Leveraging CUDA and CuPy for GPU-based operations allows for a significant acceleration by dividing the computational load, particularly as the number of particles increases. This approach would prevent advection and diffusion from becoming bottlenecks, ensuring that the solver can handle larger simulations more efficiently.

- **Different evolution times for particles based on proximity:** Not all particles in the simulation require the same time-step. Particles near the solid body need finer time-steps to accurately capture complex dynamics, while those further away, particularly in the wake region, can evolve with larger time-steps. By assigning different time-steps based on a particle's proximity to the body, the overall computational cost can be significantly reduced, as demonstrated earlier, without compromising the accuracy of the solution. In the current setup, all particles were evolved using the same time-step, which limited the maximum time-step to three times the Eulerian time-step. However, by implementing a technique that allows different time-steps for different particle regions, the time-step for particles in the wake could be increased even further, leading to additional reductions in computational time.

- **Reduce particles number:** Previously, it was mentioned that the total particle count at the end of the hybrid simulation was around 660,000, while the number of cells in the two Eulerian simulations was 159,982 and 718,818. However, unlike the Eulerian approach, where the cells extend across the entire computational domain to accommodate far-field boundary conditions, the Lagrangian particles in the hybrid simulation are free to move and tend to concentrate in regions with significant vorticity.

  Figure 8.10 illustrates the computational elements for both the pure Eulerian and hybrid simulations.

  The Eulerian mesh (shown in black) spans from $-40m$ to $80m$ in the x direction and from $-40m$ to $40m$ in the y direction. In contrast, the vortex particles (shown in orange) only cover a smaller part of the domain, ranging from $0m$ to $80m$ in the x direction and from approximately $-8m$ to $8m$ in the y direction, primarily following the wake pattern. This shows that a higher number of particles is used in the hybrid simulation compared to the resolution of the pure Eulerian simulations.

  This suggests that the particle count in the wake region could be reduced by merging particles to form larger particles without significantly sacrificing accuracy. Re-

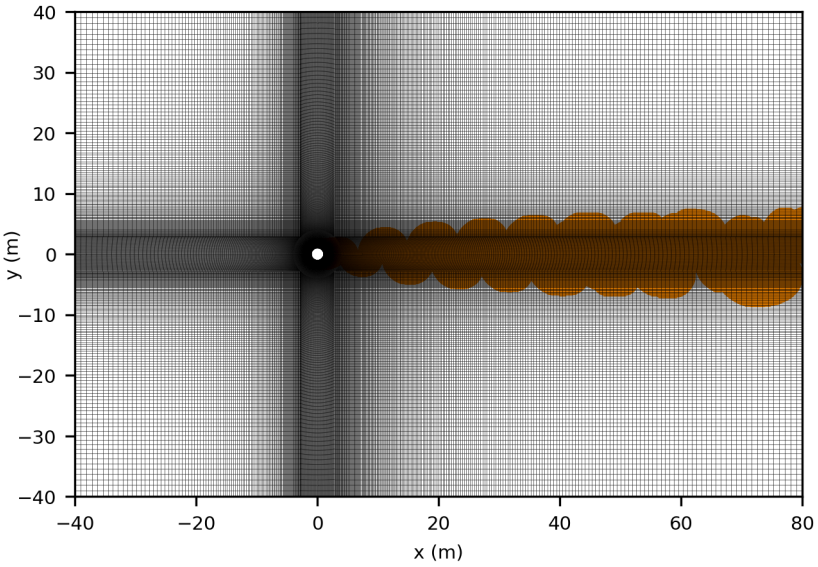


Figure 8.10: Comparison of the computational mesh in the pure Eulerian simulation (black) and the distribution of vortex particles in the hybrid simulation (orange). The Eulerian mesh extends across the entire domain, while the vortex particles are concentrated in a smaller region, primarily within the wake.

ducing the particle count in these areas would lower both memory usage and computational demand, allowing the solver to operate more efficiently while maintaining accuracy. This would also significantly reduce the computational time required for the advection and diffusion processes, which are the most time-consuming parts of the solver, as shown in Figure 8.4, where the computational time increases as the particle count grows.

Work on particle merging has already been explored in the under-review paper by J.D. Siemaszko, R. Pasolari, and A. van Zuijlen, *Accelerating vortex particle methods by downsampling the vorticity field representation*, though it has not yet been implemented in the *VPMFoam*. Additionally, Stock et al. [37] proposed an adaptive VRM method, where particles increase in size while diffusing, allowing for larger particles in the wake region.

Implementing these strategies would allow *VPMFoam* to further improve its scalability and efficiency, particularly for simulations with extensive wake regions and high particle counts. By reducing the number of particles and optimizing the use of computational resources, the solver would be better suited for handling more complex and larger-scale cases.

II

Applied Case Studies

9

Static stall of a NACA0012 at low Reynolds number

In the previous part, the VPMFoam solver was presented and validated with simple cases and geometries. This chapter begins the second part of the dissertation, focusing on applied cases of the solver involving more complex geometries and phenomena. Specifically, this chapter addresses the static stall of a National Advisory Committee for Aeronautics (NACA)0012 airfoil at $Re = 1000$*. First, a brief literature review of static stall is presented, followed by the configuration of the case. The results obtained here are compared with those from literature, demonstrating the accuracy of the hybrid Eulerian-Lagrangian solver in replicating known static stall behaviors. This case not only confirms the capability of hybrid solvers in accurately modeling challenging flows but also paves the way for their increased application in the field of external aerodynamics.*

Parts of this chapter have been published in: **R. Pasolari**, C.J. Ferreira, and A. van Zuijlen, Eulerian-Lagrangian Hybrid Solvers in External Aerodynamics: Modeling and Analysis of Airfoil Stall, Physics of Fluids, vol. 36, no. 7, p. 077135, July 2024, ISSN: 1070-6631. DOI: 10.1063/5.0216634. [104]

9.1. Introduction

Static stall refers to the phenomenon where an airfoil or wing loses lift when the angle of attack (aoa) exceeds a certain critical value, known as the stall angle. Static stall also leads to a significant increase in pressure drag. The primary reason for static stall is the separation of airflow from the surface of the airfoil. This behavior is critical in the study of aerodynamics as it affects the performance and stability of aircraft, wind turbines, and other aerodynamic devices. Unlike dynamic stall, which involves unsteady flow conditions and is characterized by delayed stall onset and hysteresis effects, static stall occurs under steady-state conditions. Understanding static stall is essential for predicting and improving the performance of airfoils. At low Reynolds numbers, static stall is particularly important in applications such as Micro Air Vehicles (MAVs) and small unmanned aerial systems.

The study of static stall has been a focal point in aerodynamic research for decades. Numerous experimental and numerical investigations have been conducted to understand the stall mechanisms and predict the stall angle accurately. For the NACA0012 airfoil, Malhotra et al. [105] studied static stall at a Reynolds number of three million using the $k-\omega$ SST turbulence model. Kurtulus [106] focused on investigating the same airfoil at a lower Reynolds number, specifically Re = 1000, which has extensive applications in MAVs. Using this case, Di Ilio et al. [107] validated the hybrid LBM they developed, and Nguyen et al. [108] validated their low-order Brinkman penalized vortex method and data-driven dynamic decomposition at aoa 20°, 60° and 90°. Experimental studies on static stall at low Reynolds numbers are limited in literature. One notable work is by Molaa et al. [109], but the lowest Reynolds number tested in this study is $Re = 8 \cdot 10^4$.

Here, the hybrid solver is tested in the static stall case of the NACA0012 airfoil at Re = 1000. The airfoil is evaluated at various angles of attack, ranging from the symmetrical configuration of 0° aoa up to 30° aoa, with an increment of 1°. The results obtained here will be compared with the corresponding numerical results from Kurtulus [106], Di Ilio et al. [107], and Liu et al. [110].

9.2. Case configuration and convergence tests

To ensure the accuracy of the simulation, conducting a convergence test is crucial. This involves identifying the optimal mesh resolution, time-step, and particles' resolution (in terms of the particles' spacing h). The convergence test focuses on the airfoil at a 4° angle of attack. The selected mesh is C-shaped, entirely composed of hexahedra, and was generated using Gmsh [86]. Visual representations of the mesh, including detailed views of the leading edge, trailing edge, and top surface, are presented in Figure 9.1. Additionally, a summary of all simulation parameters employed in this study is provided in Table 9.1.

For the spatial convergence, the mesh resolution and the particle spacing change simultaneously, while the time-step is chosen to be small enough that even for the most refined mesh, the Courant number remains under 1.0. The cell size and the particle size are refined by a factor of $r = \sqrt{2}$. However, since an exponential expansion is used in the normal direction of the airfoil, the total number of cells does not increase by a factor of 2 (as expected). The findings of the spatial convergence study are detailed in Table 9.2,

where h, C_l, C_d, and p represent the particle spacing, the instantaneous lift and drag coefficient, and the convergence order, respectively. The highlighted row indicates the case chosen, which appears to be converged.

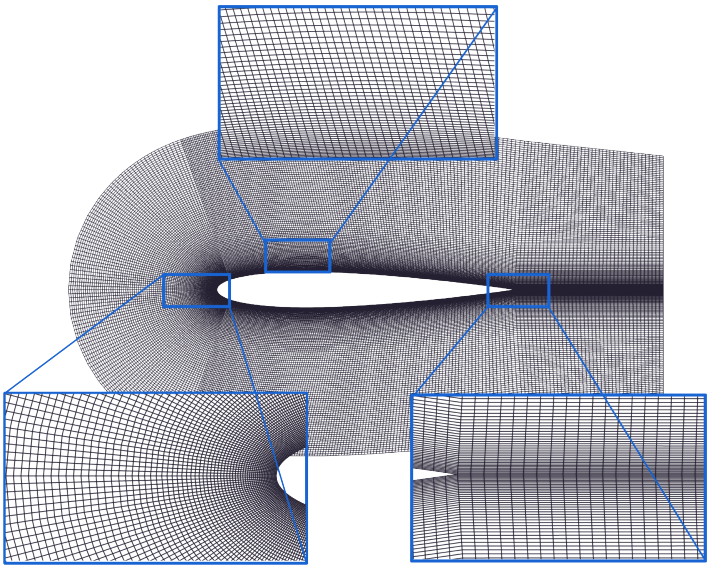


Figure 9.1: The mesh of the NACA0012 airfoil created in Gmsh [86]. Snippets of the leading edge, the trailing edge, and the top surface of the airfoil are also present.

Table 9.1: The simulation parameters used for the case of the flow around a NACA0012 at $Re = 1000$ and angle of attack ranging from 0° to 30°, using *VPMFoam*.

Parameter	**Symbol**	**Value**	**Dimension**
Reynolds number	Re	10^3	–
Freestream velocity	U_{inf}	1.0	m/s
Chord length	C	1.0	m
Simulation time	t_{sim}	100	s
Interpolation domain offset	d_{Bdry}	0.1	m

The relative error is computed as $\frac{Cl_{new} - Cl_{old}}{Cl_{new}} \cdot 100\%$. It can be seen in Table 9.2 that the aerodynamic coefficients converge to a solution, allowing C_l and C_d to be determined as $C_l = 0.1920$ and $C_d = 0.1244$. Therefore, it can be concluded that the fourth level of refinement is sufficient for running the rest of the simulations (the changes in both the lift and drag coefficients are ≤ 1%.). For an additional check, the order of convergence p can be calculated, using the formula in Equation 9.1, where f_{level_i} is the solution at the current refinement level and r is the refinement factor.

$$p = ln\frac{f_{level_{i-1}} - f_{level_{i-2}}}{f_{level_i} - f_{level_{i-1}}} / ln(r) \tag{9.1}$$

Table 9.2: Mesh and particles' spacing convergence study for the case of the flow around a NACA0012 at Re = 1000 and 4° aoa, using *VPMFoam*.

Case	**Cells**	**Time-step**(s)	**h**(m)	$\mathbf{C_l}(\mathbf{\Delta C_l})$	$\mathbf{C_d}(\mathbf{\Delta C_d})$	p
Coarse	3864	$7.5 \cdot 10^{-4}$	0.0424	0.1641	0.1239	–
Refined (level 1)	6766	$7.5 \cdot 10^{-4}$	0.0300	0.1819 (9.8%)	0.1244 (0.4%)	–
Refined (level 2)	11844	$7.5 \cdot 10^{-4}$	0.0212	0.1867 (2.6%)	0.1244 (0.0%)	3.8
Refined (level 3)	20956	$7.5 \cdot 10^{-4}$	0.0150	0.1900 (1.7%)	0.1244 (0.0%)	1.2
Refined (level 4)	36544	$7.5 \cdot 10^{-4}$	0.0106	0.1920 (1.0%)	0.1244 (0.0%)	1.3

From the value of p in Table 9.2, it can be seen that the first set of solutions is outside the asymptotic range, while the next two sets are within the asymptotic range, with the order of convergence between 1.2 and 1.3. Although the discretization schemes used here are second-order, this order of convergence is not achieved due to several factors, such as the dependence of particle resolution on the mesh, grid quality, etc.

Now, having identified the mesh for which the solution can be considered independent, a time-step convergence study can be conducted. As before, a second-order temporal discretization scheme is used. The convergence study results are presented in Table 9.3.

Table 9.3: Time-step convergence study for the case of the flow around a NACA0012 at Re = 1000 and 4° aoa.

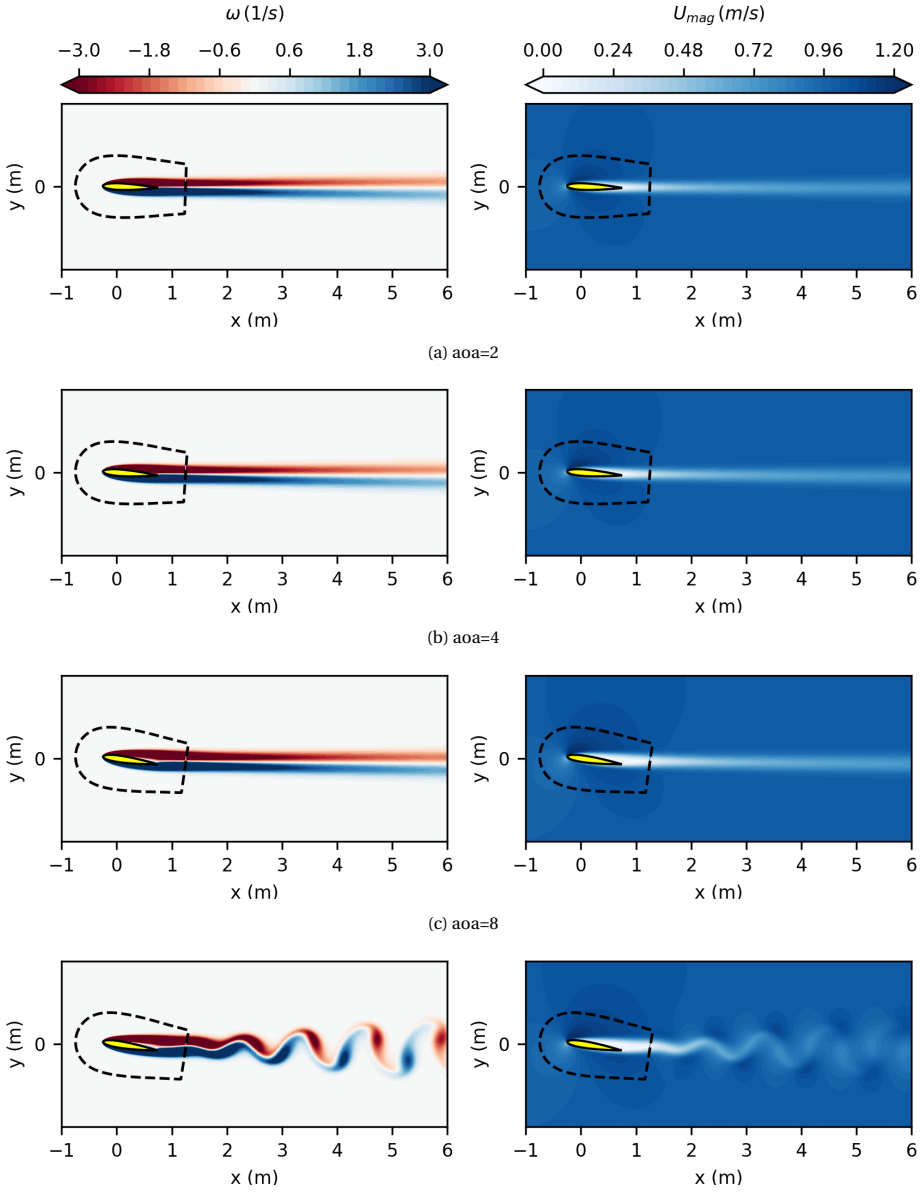
Temporal case	**Spatial case**	**Time-step**(s)	$\mathbf{C_l}(\mathbf{\Delta C_l})$	$\mathbf{C_d}(\mathbf{\Delta C_d})$
Coarse	Refined (level 4)	$1.50 \cdot 10^{-3}$	0.1956	0.1244
Refined (level 1)	Refined (level 4)	$1.06 \cdot 10^{-3}$	0.1935 (-1.1%)	0.1244 (0.0%)
Refined (level 2)	Refined (level 4)	$7.50 \cdot 10^{-4}$	0.1920 (-0.8%)	0.1244 (0.0%)
Refined (level 3)	Refined (level 4)	$5.30 \cdot 10^{-4}$	0.1916 (-0.2%)	0.1244 (0.0%)

The solution appears to converge more quickly for the time-step convergence study. From the results in Table 9.3, it can be observed that the second level of time-step refinement ($\Delta t = 7.5 \cdot 10^{-4}\ s$) results in changes of less than 1% for both aerodynamic coefficients. Hence, the highlighted row indicates the case chosen as converged.

9.3. Results

First, the flow fields are explored. Figure 9.2 showcases the vorticity field and the velocity magnitude field across various aoa. At aoa of 8° or lower, the wake remains steady without any flow instabilities. This observation aligns with the prediction by Di Ilio et al. [107], who suggested that instability begins at an aoa of 8°. The present findings indicate that instability onset occurs between 8° and 9°. Specifically, at 8° (Figure 9.2c), the flow is stable, but at 9° (Figure 9.2d), instability is already evident. Beyond these angles, the wake becomes unsteady, exhibiting a Von Kármán vortex street. This phenomenon also reflects on the aerodynamic forces, leading to oscillations as depicted in Figure 9.3 where the lift and drag coefficients are plotted for seven different aoa. At higher aoa,

such as 25° and 28°, the wake no longer exhibits its sinusoidal periodicity and transitions into a more complex form, as shown in both the contour plots (Figure 9.2) and the aerodynamic coefficients (Figure 9.3). Flow separation, which initiates at lower aoa (for instance, 4°) near the trailing edge, gradually progresses towards the leading edge as the aoa increases.



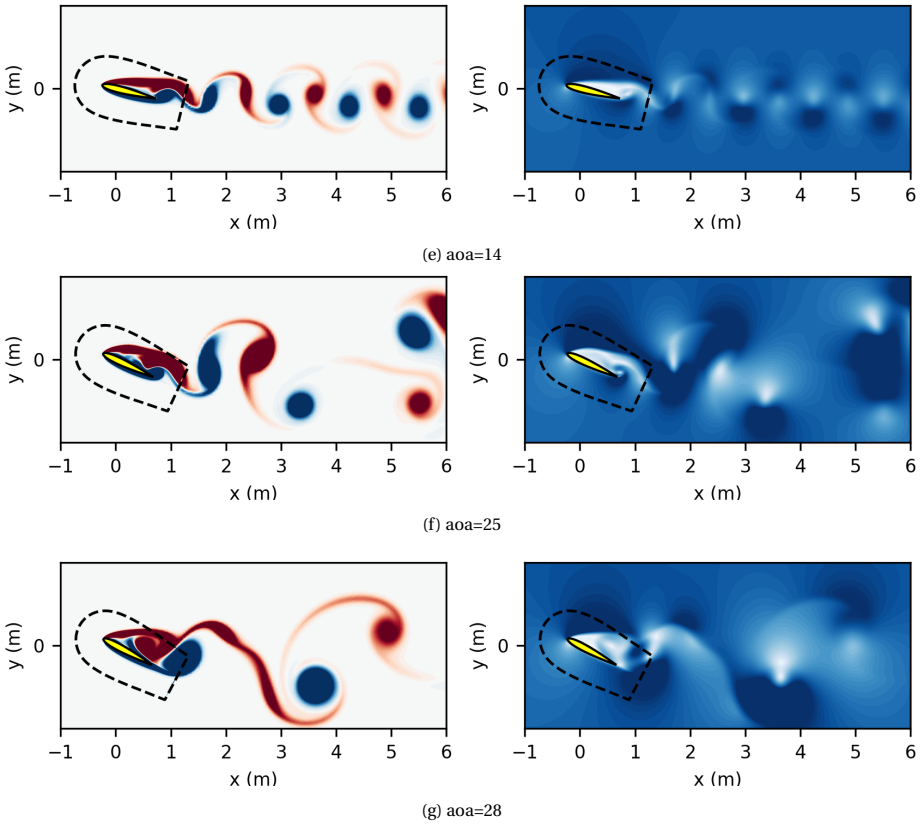
(a) aoa=2

(b) aoa=4

(c) aoa=8

(d) aoa=9

Figure 9.2: *Cont.*



(e) aoa=14

(f) aoa=25

(g) aoa=28

Figure 9.2: Flow field of NACA0012 airfoil at $Re = 1000$ for varying angles of attack, using *VPMFoam*. The left panels illustrate the vorticity field while the right panels display velocity magnitude contours. The shape formed by the black dashed line [– – –] defines the Eulerian mesh region.

In order to assess the aerodynamic performance of the hybrid solver and its capability to predict the static stall of the airfoil, the lift coefficient is plotted over the angle of attack. The plot can be seen in Figure 9.4. It must be mentioned here that a mean value is calculated for the cases where the wake is unsteady, excluding the first 10 seconds of the simulation where the transient phenomena dominate. The results are compared with those from three bibliographical references [106, 107, 110]. It is observed that *VPMFoam* shows great agreement with the references. Specifically, up to 12° aoa, all the solvers predict the lift coefficient similarly. However, there are slight deviations within the acceptable range after this point. It can be seen that up to 22° aoa, *VPMFoam* has a better agreement with the results from Di Ilio et al. [107], while for the rest, the agreement is closer to the results of Kurtulus [106]. The present solver, as well as the results from Kurtulus [106] and Di Ilio et al. [107], predict that the static stall occurs between 25° – 26° aoa, while Liu et al. [110] predict a much higher lift coefficient for 27° aoa, and then the airfoil stalls. The results from Di Ilio et al. [107] differ from the other two references at 29°, while the present solver agrees with them. Finally, the result for 30° aoa has been

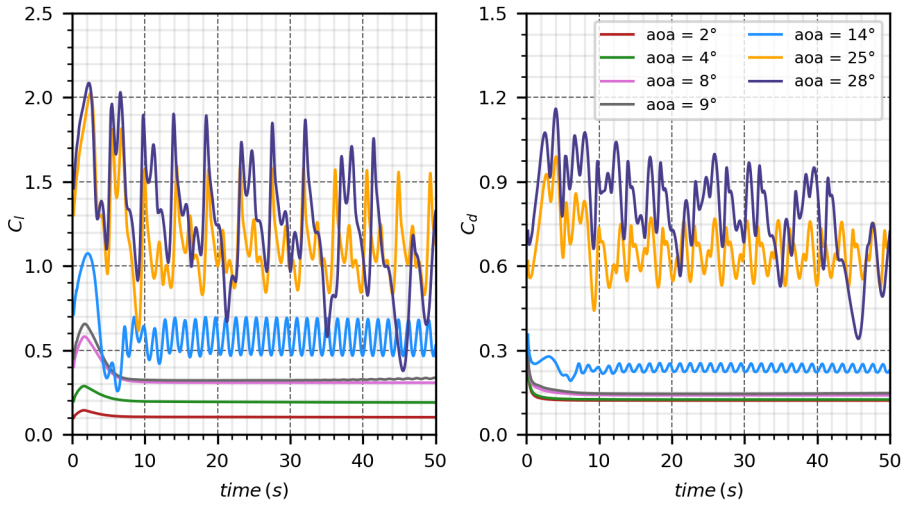


Figure 9.3: Aerodynamic coefficients of NACA0012 airfoil at $Re = 1000$ for varying angles of attack, using *VPMFoam*. The left plot illustrates the lift coefficient over the aoa, while the right plot displays the drag coefficient over the aoa.

included for *VPMFoam*. It can be mentioned here that slight differences in the aerodynamic coefficient in the high aoa regime may be present due to the high unsteadiness of the solution.

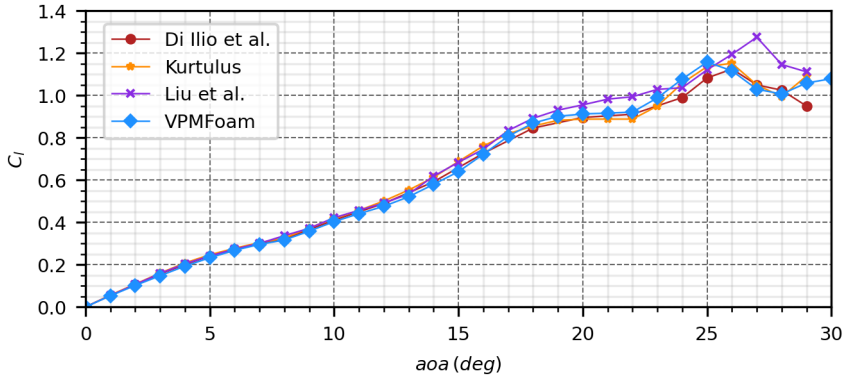


Figure 9.4: Lift coefficient as a function of the aoa. The present results are compared with the corresponding results from Kurtulus [106], Di Ilio et al. [107] and Liu et al. [110].

Figure 9.5 shows the ratio between the lift and drag coefficient for the different aoa. Again, there is a very nice agreement between *VPMFoam* and the references [106, 107].

The present solver is in slightly better agreement with the results from Di Ilio et al. [107], especially for the aoa between 5°−12°, where the results from Kurtulus [106] differ slightly, within acceptable limits again. All the solvers predict similarly that the highest C_l/C_d occurs for aoa around 10° to 12°.

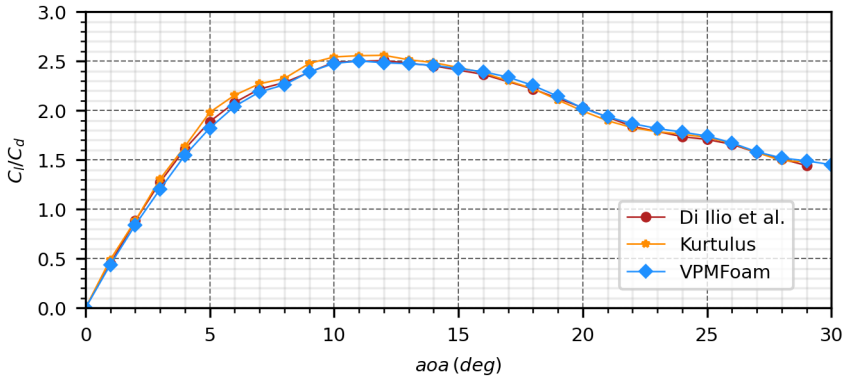


Figure 9.5: Ratio of the lift and drag coefficients as a function of the aoa. The present results are compared with the corresponding results from Kurtulus [106] and Di Ilio et al. [107].

9.4. Conclusions

The results confirm that *VPMFoam* accurately captures the static stall of the NACA0012 airfoil at low Reynolds numbers. The solver successfully models both steady and unsteady flow behaviors, including vortex shedding and stall onset. The close agreement with established results from the literature demonstrates the robustness of the hybrid Eulerian-Lagrangian approach for simulating complex aerodynamic phenomena in external aerodynamics.

10

Dynamic stall of a NACA0012 at low Reynolds number

The previous chapter dealt with the static stall of a NACA0012 airfoil at $Re = 1000$. *In this chapter, another very interesting phenomenon is investigated: dynamic stall. For this case, the same airfoil and Reynolds number are used. First, a brief literature review is presented, followed by the case configuration and the simulation parameters. Then, the results obtained from the simulation are presented and compared with reference data from pure OpenFOAM simulations.*

Parts of this chapter have been published in: **R. Pasolari**, C.J. Ferreira, and A. van Zuijlen, Eulerian-Lagrangian Hybrid Solvers in External Aerodynamics: Modeling and Analysis of Airfoil Stall, Physics of Fluids, vol. 36, no. 7, p. 077135, July 2024, ISSN: 1070-6631. DOI: 10.1063/5.0216634. [104]

10.1. Introduction

A critical phenomenon observed in airfoils is the dynamic stall. Unlike static stall, which is observed in fixed angles of attack (aoa), dynamic stall involves the rapid variation of the angle of attack, introducing additional complexities into the airflow behavior around airfoils. The dynamic stall is characterized by delayed stall onset, a temporary lift overshoot, and complex vortex-shedding patterns that significantly influence the lift and drag experienced by the airfoil. This phenomenon is particularly relevant in wind turbines, both Horizontal Axis Wind Turbines (HAWTs) [111] and Vertical Axis Wind Turbines (VAWTs) [112], helicopters [113], highly maneuverable fighters [114] and MAVs. Wang et al. [115] investigated computationally the dynamic stall of the NACA0012 airfoil at $Re = 10^5$ and compared the CFD results with experimental. They used two different sets of oscillating patterns. Akbari et al. [116] used a vortex method to investigate the dynamic stall for the NACA0012 airfoil at $Re = 10^4$ and different oscillation patterns.

After the static stall case presented in the previous chapter, the solver is applied to investigate the dynamic stall of the NACA0012 at $Re = 1000$, a Reynolds number significant for many applications, such as the MAVs, and which has not been investigated thoroughly in the bibliography. Existing studies primarily focus on higher Reynolds numbers [115, 116], or do not address dynamic stall directly. For example, Kurtulus [117] explored small amplitude oscillations of 1° pitch at frequencies of $1Hz$ and $4Hz$, comparing these to a non-oscillatory case. To support the hybrid simulations, pure OpenFOAM simulations are also conducted to ensure result validation.

10.2. Case configuration and simulation parameters

The airfoil is set to oscillate around its aerodynamic center, located a quarter chord ($0.25C$) from the leading edge. This oscillation is characterized by a mean angle of attack (α_0), an oscillation amplitude (α_{amp}), and a reduced frequency ($k = \frac{f_{ang}}{2}\frac{C}{U_{\text{inf}}}$), where C is the chord length, U_{inf} is the freestream velocity, and f_{ang} is the angular frequency of the oscillation. The angle of attack varies over time as follows:

$$aoa = \alpha_0 - \alpha_{amp} \cdot \sin(f_{ang}\, t) \tag{10.1}$$

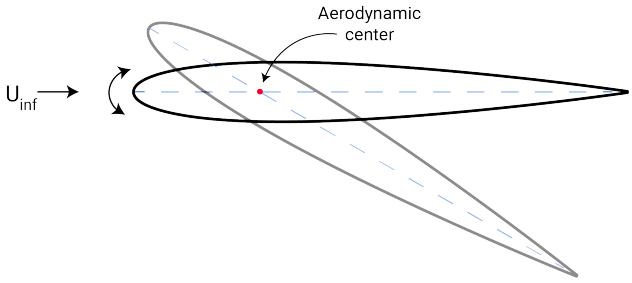


Figure 10.1: The NACA0012 airfoil oscillates from the initial aoa a_0, with an angular frequency f_{ang} and amplitude a_{amp}.

In this analysis, the initial angle of attack is specified as $\alpha_0 = 15°$, with an amplitude

also set at 15° (α_{amp} = 15°). The investigation will incorporate four distinct reduced frequencies: $k = 0.1, 0.2, 0.4, 0.6$.

Experimental data are not present in the bibliography for this Reynolds number, so the results will be compared to those of a pure Eulerian simulation in OpenFOAM. The computational domain used in the pure OpenFOAM simulations is illustrated in Figure 10.2. The oscillatory rotating motion of the airfoil is modeled using the cyclic AMI feature of OpenFOAM [88], which is widely used in many applications for airfoil simulations, like the work of Pan et al. [118]. A mesh and time-step convergence test has also been employed for the OpenFOAM case to have a fair comparison of the converged OpenFOAM and hybrid solver solutions. The OpenFOAM results refer to the converged cases for all the cases presented in this section.

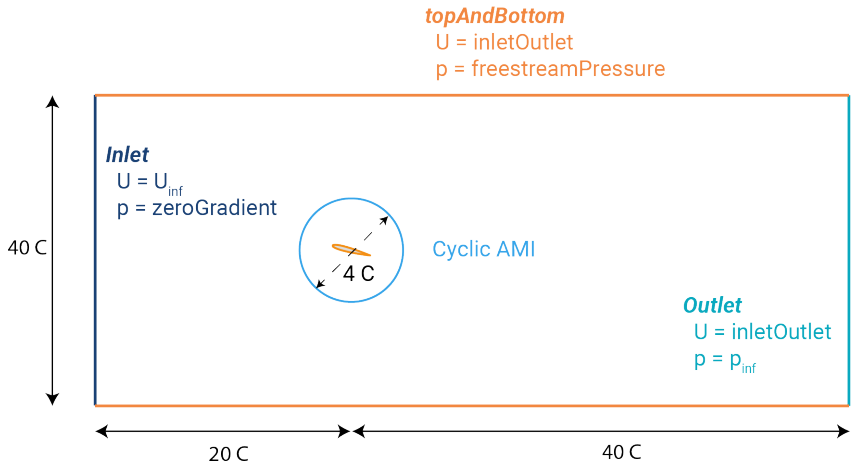


Figure 10.2: The computational domain for the pure OpenFOAM simulation of the dynamic stall of a NACA0012 airfoil.

Dynamic stall is a highly complex phenomenon affected by various parameters, one of which is the mesh. To ensure consistency between the OpenFOAM solution and the solution obtained from the hybrid solver, a different mesh is employed than that used in the static case (see Chapter 9). Specifically, the inner mesh of the full OpenFOAM solution is utilized, which, in OpenFOAM, is the region that rotates, as the complete mesh for the hybrid solution. This mesh is depicted in Figure 10.3. It is a circular mesh with a radius twice the chord length, created using the *snappyHexMesh* tool of OpenFOAM. The mesh features four levels of refinement as it approaches the surface.

10.3. Results

10.3.1. Flow fields

Figure 10.4 illustrates the vorticity fields for different pitching reduced frequencies and angles of attack. On the left, the results obtained from pure OpenFOAM simulations are depicted for arbitrary phases of the motion, while the corresponding results from the hybrid solver are illustrated on the right panel of Figure 10.4. Across all scenarios,

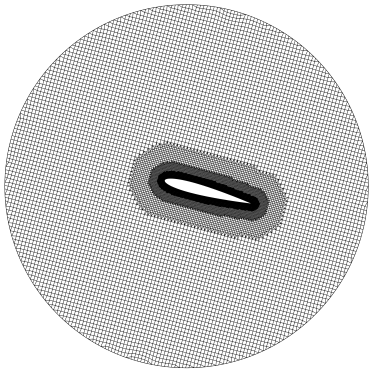
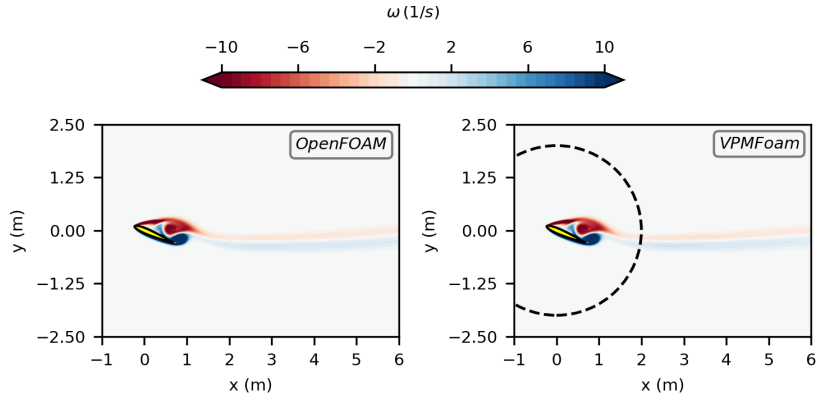


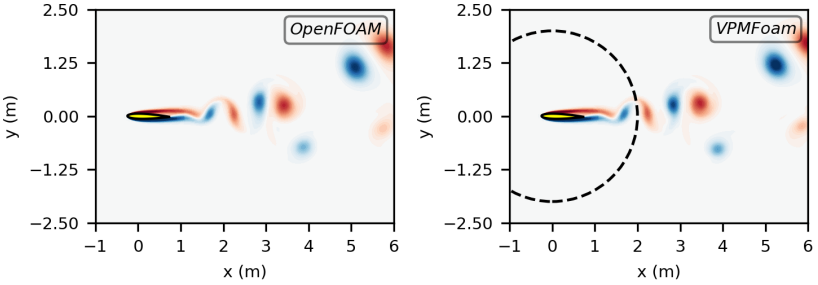
Figure 10.3: The inner mesh of the full OpenFOAM simulation, which is here utilized as the entire mesh for the hybrid simulation.

similar wake structures are observed from both solvers. The only noticeable difference is that in the OpenFOAM case, the wake vortices are somewhat more diffused than those in the hybrid solver. This was expected, as the vortex particles in the hybrid solver ensure very minimal numerical diffusion is added to the flow. The artificial diffusion into the wake is more obvious in the wake structures depicted in Figure 10.5. This is the wake for the case of $k = 0.6$ and aoa = 19.5°, with the airfoil ascending. In this contour, the absolute vorticity is illustrated. It can be seen that the wake vortices in the OpenFOAM case (left) are more diffused than the corresponding vortices in the hybrid solver (right). Even with three levels of refinement in the wake for the OpenFOAM case, and despite the aerodynamic coefficients being converged, the artificial diffusion has an impact on the wake structures.

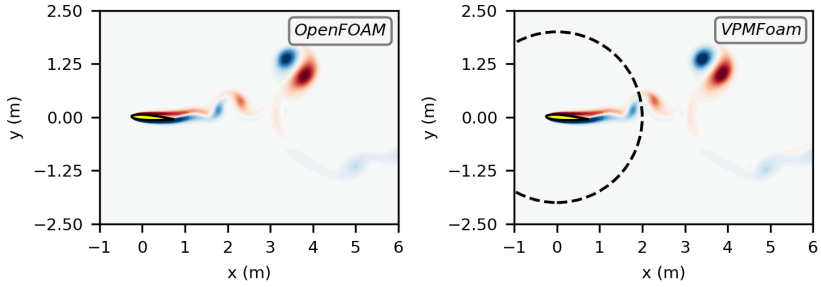


(a) $k = 0.1, aoa = 23° \uparrow$

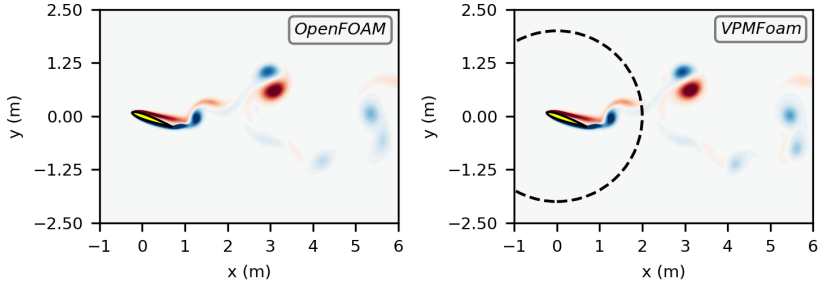
Figure 10.4: *Cont.*



(b) $k = 0.2, aoa = 1.3^\circ \downarrow$



(c) $k = 0.4, aoa = 4^\circ \uparrow$



(d) $k = 0.6, aoa = 19.5^\circ \uparrow$

10

Figure 10.4: Vorticity fields for the dynamic case of the NACA0012 airfoil at $Re = 1000$. The contours represent different pitching reduced frequencies and time instances. On the left panel, the OpenFOAM results are depicted, while on the right panel, the results from the hybrid solver are illustrated. The circle formed by the black dashed line [– – –] defines the Eulerian mesh region in the hybrid case.

10.3.2. Hysteresis loops

Figure 10.6 depicts the hysteresis loops for the four different reduced frequency cases. The initial observation here is that, as expected, the pitching airfoil can reach higher

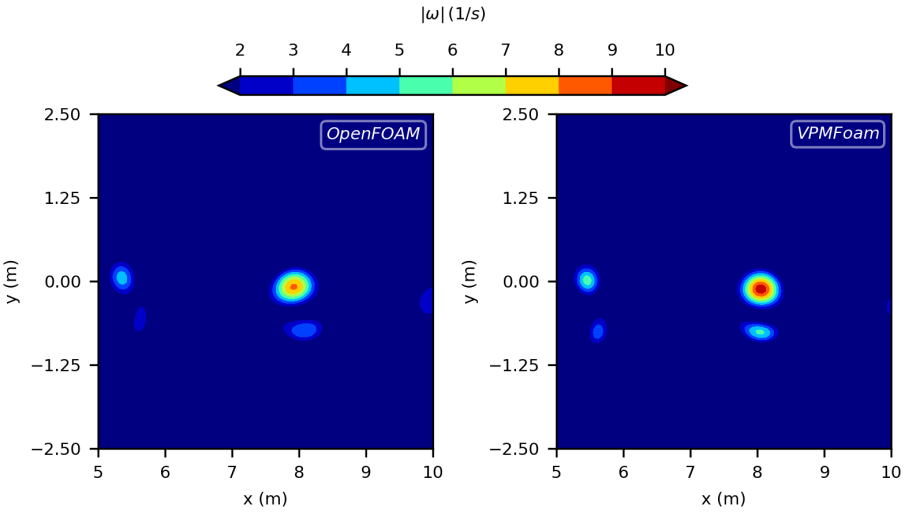
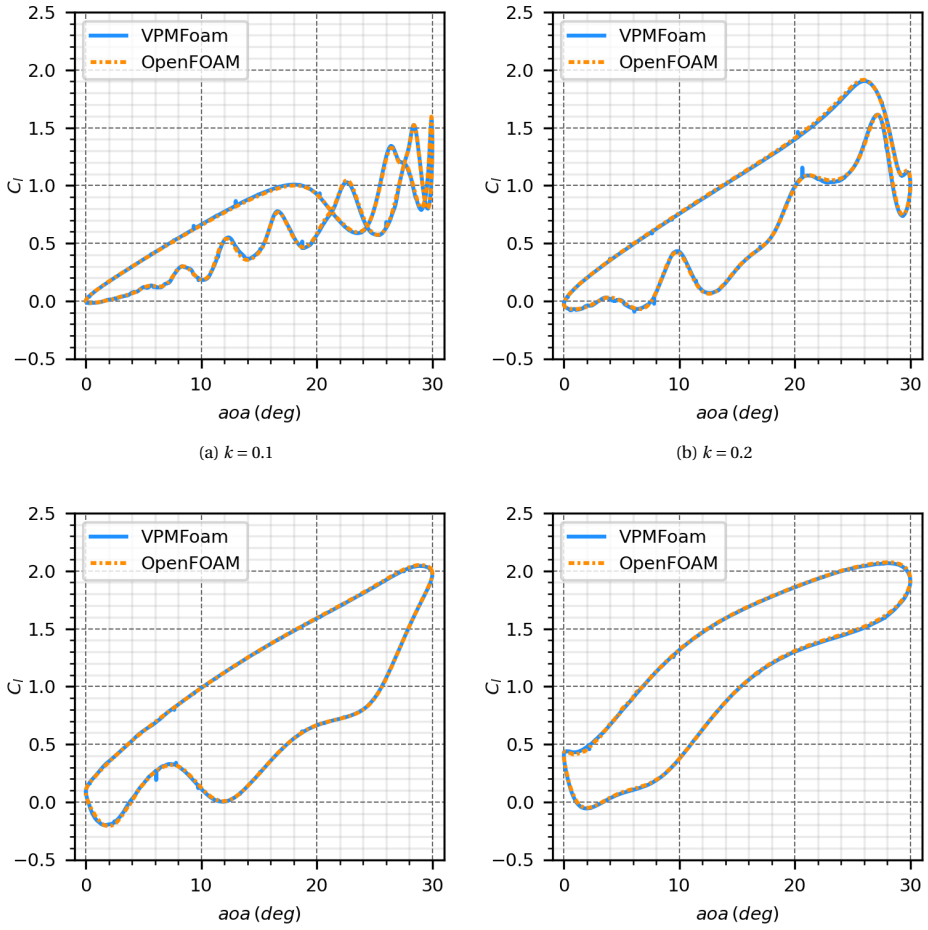


Figure 10.5: The wake for the case of $k = 0.6$, aoa = 19.5°, with the airfoil ascending. On the left is the result from OpenFOAM, while on the right is the result from the hybrid solver. The absolute vorticity is plotted.

angles of attack before stalling compared to the static airfoil. This is particularly evident in the cases of $k = 0.4$ and $k = 0.6$, where the airfoil reaches $29° - 30°$ without stalling, while in the static case, stalling occurs between 25° and 26°. As the frequency increases, the airfoil does not have sufficient time to stall, allowing the lift coefficient to increase beyond the static stall point. In lower reduced frequencies, like the case of $k = 0.1$, a more complex situation arises, as depicted in Figure 10.6a, where the airfoil has enough time to adapt to its state.

Now, when comparing the hysteresis loops obtained from the pure OpenFOAM simulations with those from the hybrid solver, no differences are observed. The hysteresis loops for all the reduced frequencies match exactly, demonstrating that *VPMFoam* can accurately reproduce the results that OpenFOAM produces. Some spikes present in the hysteresis loops of the hybrid solver results are due to the phenomenon that appeared in dynamic mesh simulations (this was studied in Chapter 6 and in the publication [87]).



(a) $k = 0.1$ (b) $k = 0.2$

(c) $k = 0.4$ (d) $k = 0.6$

Figure 10.6: Hysteresis loops for the dynamic stall case of a NACA0012 airfoil at $Re = 1000$, and reduced frequencies $k = 0.1, 0.2, 0.4, 0.6$.

10

10.4. Conclusions

The results show that *VPMFoam* successfully simulates the dynamic stall of the NACA0012 airfoil at $Re = 1000$, matching the accuracy of OpenFOAM. The hybrid solver provides similar wake structures and hysteresis loops, with the key advantage of reduced artificial diffusion in the wake due to its Lagrangian method. This demonstrates that *VPMFoam* is a reliable and accurate alternative for simulating complex unsteady flows, especially when wake fidelity is crucial.

11

Hybrid Darrieus-Savonius turbine using actuator models

The previous two chapters discussed the stall (static and dynamic) of an airfoil. In this chapter, a two-dimensional hybrid Darrieus-Savonius VAWT is explored, using actuator models. The chapter begins with a brief introduction to Darrieus and Savonius wind turbines, followed by their combination into the hybrid Darrieus-Savonius VAWT. The chapter then proceeds to the modeling method used here. Two different configurations of the hybrid solver are presented, illustrating the range of variable choices available in the hybrid context, depending on the desired level of fidelity. The results from the simulations are compared with corresponding results obtained from OpenFOAM.

Parts of this chapter are based on the dissertation: **J. Pan**, Towards hybrid modeling of hybrid VAWT, Delft University of Technology, 2024. [Online]. Available: https://repository.tudelft.nl/record/uuid:30929ef5-705d-4fc6-9251-8d159e6b1139. Accessed: 2024-09-10. [119]

11.1. Introduction

Wind energy, as a field of external aerodynamics, is well-suited for hybrid Eulerian-Lagrangian solvers due to the strong body-vortex interactions involved. Accurate modeling of wakes is crucial, particularly in configurations with multiple wind turbines, where wake interactions play a significant role. This field continues to attract research attention, with new designs and configurations constantly emerging. However, achieving high-fidelity results remains computationally demanding, especially when simulating entire wind farms, which can pose challenges in practical applications.

One of the main classifications of wind turbines is based on the axis of rotation. Wind turbines can be categorized into two types: horizontal axis (HAWT) and vertical axis (VAWT) turbines. In HAWTs, the rotational axis is parallel to the ground and aligned with the wind direction, whereas in VAWTs, the rotational axis is perpendicular to both the ground and the wind direction. Today, most commercial developments focus on HAWTs due to their higher performance. However, over the past two decades, VAWTs have garnered increasing attention for several reasons: they perform better in low or unsteady wind conditions, produce much less noise, do not require a starting mechanism (except for the Darrieus type), and have lower cut-in speeds [120]. These advantages make VAWTs particularly appealing in urban areas where wind conditions are more turbulent, and factors like noise and size are critical. Nevertheless, a detailed analysis of VAWTs is beyond the scope of this dissertation, as this case is used primarily as a validation example for the hybrid solver, demonstrating its potential in the wind energy sector. For further information on VAWTs, the reader can refer to the review by Kumar et al. [120].

Wind turbines, including VAWTs, can also be classified into drag-based and lift-based turbines. Savonius turbines [121] represent the drag-based category, while Darrieus turbines [122] represent the lift-based category. Savonius turbines operate at low cut-in speeds, with their simplest form being two half-cylinders facing in opposite directions, forming an S-shape. However, their main drawback is their relatively poor aerodynamic performance compared to other turbines [120]. On the other hand, Darrieus turbines, which have better aerodynamic performance, are lift-based turbines with airfoils that move in a circular path.

In an effort to combine the advantages of both types of VAWTs, researchers have developed hybrid Darrieus-Savonius turbines. An example of such a hybrid turbine is the PowerNEST [123], shown in Figure 11.1. In hybrid turbines, the two types are often connected to the same shaft to increase the power coefficient, while also incorporating the benefits of Savonius turbines, such as low cut-in speeds and self-starting capabilities. As mentioned earlier, the focus of this chapter is not on the technology of hybrid turbines, but rather on comparing the results with those obtained from pure Eulerian solvers. For more detailed information on hybrid turbines, the reader can refer to the review by Kumar et al. [120] and the doctoral thesis by Pan [119].

In her doctoral thesis, Pan [119] focused on modeling a hybrid Darrieus-Savonius wind turbine using various computational tools. The main objective of this chapter is to compare the results presented in her thesis using OpenFOAM with the corresponding results produced by *VPMFoam*.

Figure 11.1: A hybrid Darrieus-Savonius wind turbine called PowerNEST [123].

11.2. Cases configuration and modeling

The two combined turbines are modeled here through their force fields. Specifically, an Actuator Disk (AD) model is used to represent the Savonius wind turbine, while an Actuator Cylinder (AC) model is used for the Darrieus wind turbine. Pan [119] explored the idealized hybrid VAWT with thrust coefficients $c_{T,AD} = 0.1$ for the actuator disk and $c_{T,AC} = 0.7$ for the actuator cylinder. The thrust coefficient c_T is theoretically related to the power coefficient c_P as follows:

$$c_P = c_T \left(0.5 + 0.5\sqrt{1 - c_T}\right) \tag{11.1}$$

The power coefficient c_P is related to the turbine's torque M, rotational speed Ω, air density ρ, inflow velocity U_{inf}, and the turbine's swept area A, through the following expression:

$$c_P = \frac{\Omega M}{0.5\rho U_{inf}^3 A} \tag{11.2}$$

Two different configurations of the hybrid solver are explored here. The first configuration includes both turbines in the Eulerian domain and models them with actuators, as described earlier. The second configuration includes only the Savonius turbine in the Eulerian domain, while the Darrieus turbine is modeled as point vortices shed from its airfoils. This approach can be further refined by using the actual geometries of the turbines. For instance, the Savonius turbine could be modeled as a separate domain based on its geometry or as an actuator disk. Similarly, the Darrieus turbine could be represented by two independent Eulerian domains, each corresponding to one of its airfoils, or alternatively as point vortices or an actuator cylinder. Of course, various combinations of these models could be employed depending on the specific objectives of the simulation. However, in this chapter, the focus is only on the two configurations mentioned above. The following paragraphs briefly present these two cases, along with the pure Eulerian configuration analyzed by Pan [119].

AD and AC in OpenFOAM

In the pure OpenFOAM case, both the actuator disk and the actuator cylinder are included within the Eulerian domain, as shown in Figure 11.2. The computational domain extends $30R_{AC}$ in the y direction and $50R_{AC}$ in the x direction, where R_{AC} is the radius of the actuator cylinder. As depicted in Figure 11.2, the actuator disk generates a uniform force field in the x direction.

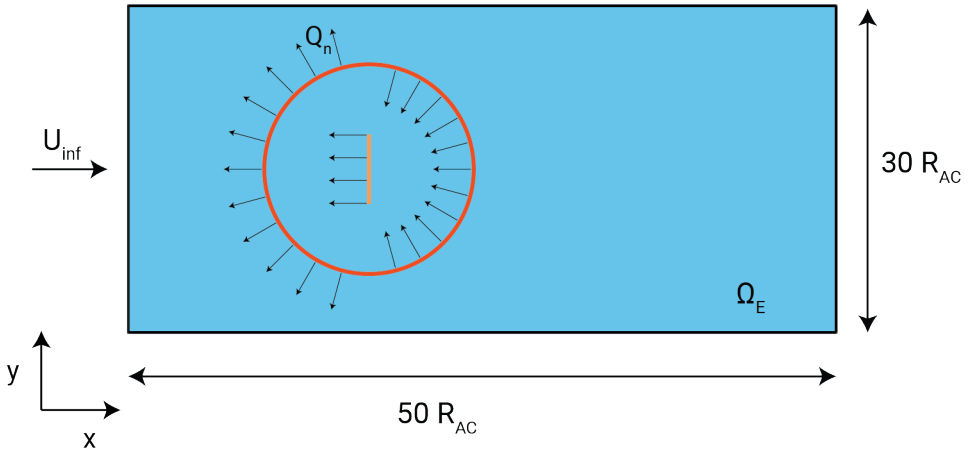


Figure 11.2: The configuration of the AD and the AC in the pure OpenFOAM case. *Note: Scale is illustrative only, not representative of actual proportions.*

Given the value of $c_{T,AD}$, the total thrust of the actuator disk T_{AD} is calculated as:

$$T_{AD} = 0.5 c_{T_{AD}} \rho U_{inf}^2 A_{AD} \tag{11.3}$$

Thus, the force applied to each cell of the actuator disk can be computed as:

$$T_{AD_{cell}} = T_{AD} \frac{V_{AD_{cell}}}{V_{AD}} \tag{11.4}$$

where V denotes volume, with V_{AD} being the total volume of the actuator disk and $V_{AD,cell}$ the volume of each cell in the actuator disk.

For the actuator cylinder, the force field is more complex as it has two components. Assuming a uniform force field normal to the cylinder (denoted as Q_n in Figure 11.2), the total thrust in the x direction is calculated as:

$$T_{AC} = 0.5 c_{T,AC} \rho U_{inf}^2 A_{AC} \tag{11.5}$$

The total volumetric force can then be computed from the thrust force using the formula:

$$Q_{n,V} = \frac{T_{AC}}{\sum_{i=0}^{n_{AC,cell}} \sin\left(\arctan \frac{|x - x_{0,c}|}{|y - y_{0,c}|}\right) V_i} \tag{11.6}$$

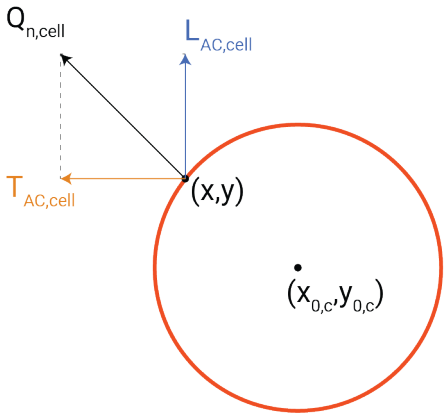


Figure 11.3: Decomposition of the normal force of a cell in the actuator cylinder into the thrust force $T_{AC,cell}$ and the y-direction component $L_{AC,cell}$.

where x and y are the coordinates of the point on the actuator cylinder, and $x_{0,c}$ and $y_{0,c}$ represent the center of the actuator cylinder, as shown in Figure 11.3.
$Q_{n,V}$ is the value of the volumetric force vector $\mathbf{Q_{n,V}}$, which points in the normal direction of the cylinder. For each cell in the actuator cylinder, a direction vector is constructed as follows:

$$\mathbf{d_{cell}} = \begin{bmatrix} \sin\left(\arctan\dfrac{|x-x_{0,c}|}{|y-y_{0,c}|}\right) \\ \cos\left(\arctan\dfrac{|x-x_{0,c}|}{|y-y_{0,c}|}\right) \end{bmatrix} \tag{11.7}$$

To ensure the correct y direction (pointing in the negative direction for cells in the first and third quadrants), the direction vector for these cells is adjusted as follows:

$$\mathbf{d_{cell}} = \begin{cases} \begin{bmatrix} \sin\left(\arctan\dfrac{|x-x_{0,c}|}{|y-y_{0,c}|}\right) \\ -\cos\left(\arctan\dfrac{|x-x_{0,c}|}{|y-y_{0,c}|}\right) \end{bmatrix}, & \text{if } \dfrac{x-x_{0,c}}{y-y_{0,c}} < 0 \\ \begin{bmatrix} \sin\left(\arctan\dfrac{|x-x_{0,c}|}{|y-y_{0,c}|}\right) \\ \cos\left(\arctan\dfrac{|x-x_{0,c}|}{|y-y_{0,c}|}\right) \end{bmatrix}, & \text{otherwise} \end{cases} \tag{11.8}$$

Finally, the force applied to each cell in the two directions, with $T_{AC_{cell}}$ representing the thrust in the x direction and $L_{AC_{cell}}$ the component in the y direction, is calculated as:

$$\begin{bmatrix} T_{AC_{cell}} \\ \\ L_{AC_{cell}} \end{bmatrix} = Q_{n,V} V_{cell} \mathbf{d_{cell}} \tag{11.9}$$

AD and AC in *VPMFoam*

In the second case, the *VPMFoam* is used instead of pure OpenFOAM. Both actuators are once again placed inside the Eulerian domain, which is a circular domain with a diameter of $3.8R_{AC}$, as shown in Figure 11.4. All aspects of the two actuators are identical to the OpenFOAM case presented above.

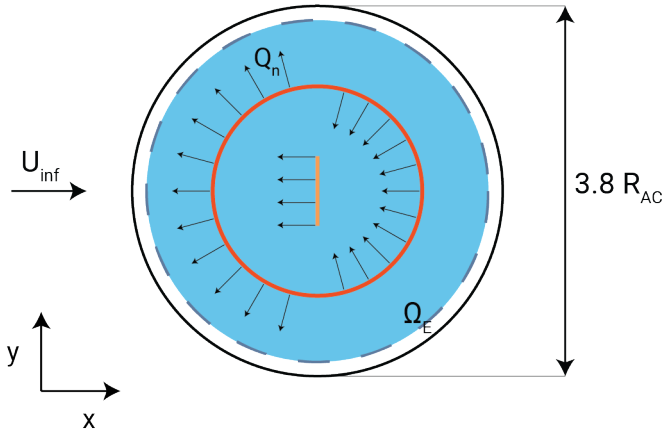


Figure 11.4: The configuration of the AD and the AC in the *VPMFoam* case.

AD and point vortices in *VPMFoam*

In the third and final case, the configuration is slightly different. The *VPMFoam* is again used, but this time only the Savonius turbine is modeled as an actuator. As shown in Figure 11.5, the Eulerian domain is a square with the actuator disk inside, and its dimension is reduced to $1.6R_{AC}$.

The Darrieus turbine is not modeled as an actuator cylinder; instead, point vortices are introduced at the top and bottom of the Eulerian domain, at the locations where vorticity is shed from the turbine. In this idealized case, without considering diffusion or vortex stretching, vorticity can only be introduced by a change in external body forces $\mathbf{f}$:

$$\frac{D\boldsymbol{\omega}}{Dt} = \nabla \times \mathbf{f} \tag{11.10}$$

which, for a 2D case, leads to:

$$\frac{D\boldsymbol{\omega}}{Dt} = \begin{bmatrix} 0 \\ 0 \\ \dfrac{\partial f_y}{\partial x} - \dfrac{\partial f_x}{\partial y} \end{bmatrix} \tag{11.11}$$

Hence, for the actuator cylinder, the only locations where shed vorticity is created are the points where vortices are introduced into the flow. The amount of circulation Γ generated at each time-step of the simulation is calculated using the thrust coefficient $c_{T,AC}$

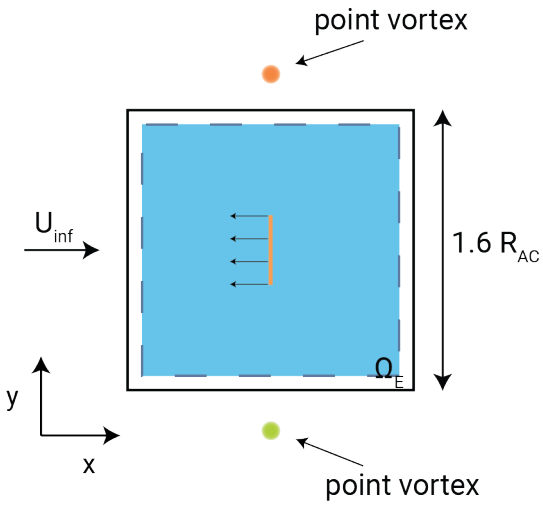


Figure 11.5: The configuration of the AD and the point vortices in the *VPMFoam* case.

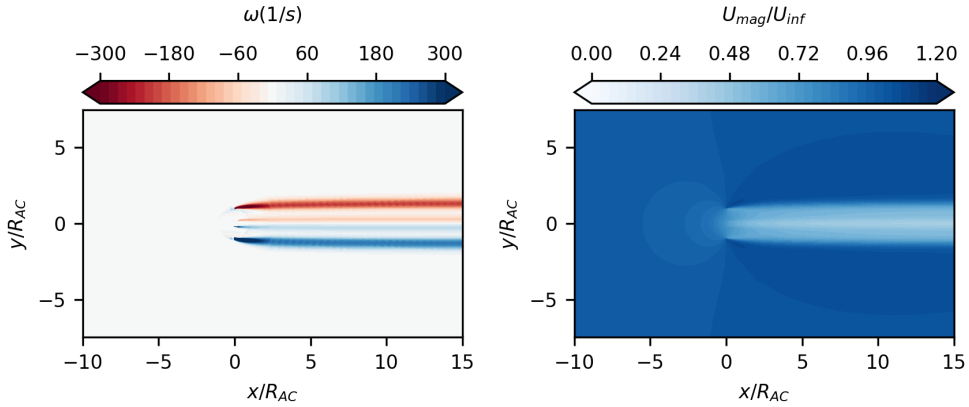
(as previously applied to the actuator cylinder), the freestream velocity U_{inf}, and the time-step size Δt, according to the following formula [124]:

$$\Gamma = 0.5 c_{T,AC} U_{inf}^2 \Delta t \tag{11.12}$$

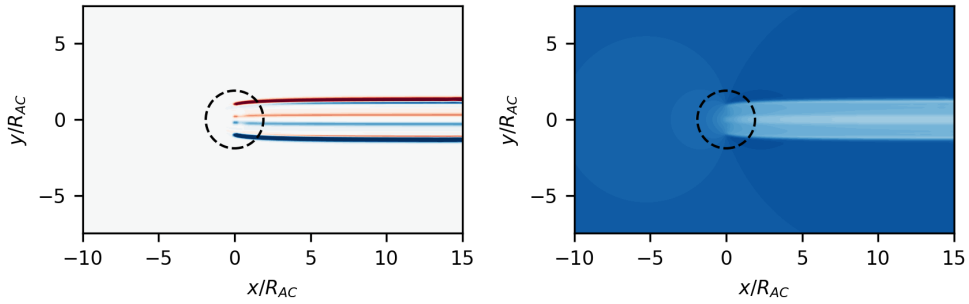
11.3. Results

First, the results of the three cases are compared in terms of the produced velocity and vorticity fields. Figure 11.6 shows the vorticity field on the left and the normalized velocity magnitude field (normalized with U_{inf}) on the right for the three different cases. Comparing the contours, it can be observed that the velocity fields show good agreement across the cases. However, some differences are evident in the vorticity field. Specifically, in the pure Eulerian solver, the effect of artificial diffusion is noticeable, with the shed vorticity being diffused rapidly behind the actuators.

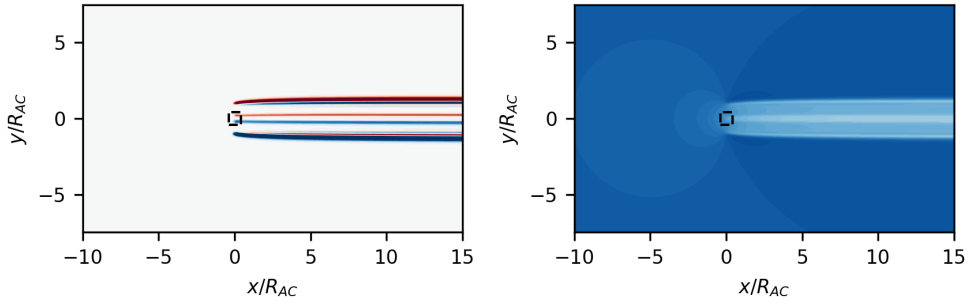
In contrast, the vorticity field in the two hybrid solver cases appears to be more conserved. The locations where vorticity is shed seem to match across all cases. One interesting observation in the hybrid cases, particularly in the case where point vortices are used, is that layers of opposite vorticity are formed in the wake of the Darrieus turbine. This occurs on both sides of the turbine. The cause of this phenomenon is the redistribution method employed in the solver. The redistribution kernel, presented in Figure 2.5, can assign negative values to neighboring particles during the redistribution process to conserve circulation. These negative values, combined with the high advection in this case, result in particles of opposite circulation not having sufficient time to diffuse, causing layers of negative and positive vorticity to form. Despite this, the effect does not seem to significantly impact the results.



(a) AD and AC in OpenFOAM.



(b) AD and AC in *VPMFoam*.



(c) AD and point vortices in *VPMFoam*.

Figure 11.6: Flow fields of the hybrid Darrieus-Savonius turbines using three different modeling methods. The left panels illustrate the vorticity fields while the right panels display normalized velocity magnitude contours.

Another important result to discuss is the velocity profile. Figure 11.7 shows the streamwise velocity profile (in the x direction) at $y = 0$ for the three different cases. The plot also includes two theoretical values for comparison.

To calculate the theoretical values of the velocity deficit at the rotor and in the wake,

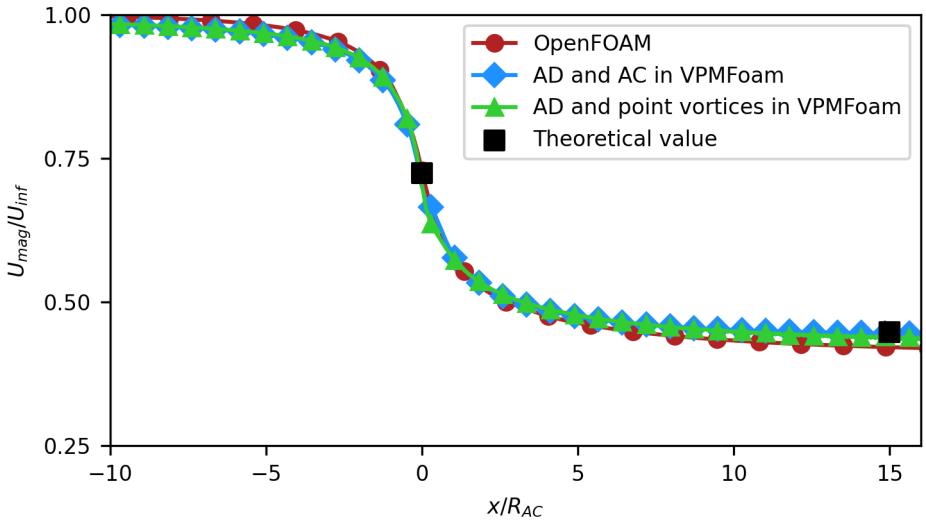


Figure 11.7: Streamwise velocity profile at $y = 0$ for the three different models, along with theoretical values at the center of the rotor and in the wake.

the induction factor α is used. The induction factor is given by:

$$\alpha = 0.5 - 0.5\sqrt{1 - c_T} \tag{11.13}$$

Using this, the theoretical values for the streamwise velocity at the center of the rotor and in the wake are:

$$U_{rotor} = U_{inf}(1-\alpha) \quad , \quad U_{wake} = U_{inf}(1-2\alpha) \tag{11.14}$$

In Figure 11.7, it can be observed that all the solvers predict the velocity at the center of the rotor quite well. For the wake region, the results from all three models are close. However, OpenFOAM predicts a slightly lower velocity due to numerical diffusion, whereas the *VPMFoam* results for both cases align closely with the theoretical values.

11.4. Conclusions

The results presented in this chapter demonstrate the accuracy of *VPMFoam* when applied to the simulation of hybrid Darrieus-Savonius 2D VAWTs using actuator models. Across all cases, *VPMFoam* produced results consistent with those obtained from OpenFOAM, particularly in terms of velocity profiles and wake structures. Moreover, the results highlight the versatility of *VPMFoam*, which enables the use of various modeling approaches, such as actuator disks, actuator cylinders, and point vortices, offering flexibility depending on the level of fidelity required. This adaptability provides significant advantages for optimizing simulations based on available computational resources and accuracy needs.

Future developments could focus on extending this hybrid framework by incorporating actual turbine geometries and advancing to 3D simulations, further expanding the applicability of *VPMFoam* for more complex and realistic turbine configurations.

III

Closing Remarks

12

Conclusions and reflections

This chapter wraps up the key findings presented throughout the dissertation, reflecting on the challenges addressed and the performance of the developed solver in terms of accuracy and efficiency. Additionally, it demonstrates the broader impact of the solver on external aerodynamics, highlighting its contributions to the field.

After validating the hybrid solver in numerous scenarios, including static cases, dynamic cases, and multibody problems, assessing its efficiency, and applying it to more realistic two-dimensional scenarios, it is now time to summarize the key findings from these efforts. The following bullet points gather the main outcomes of this work and present personal reflections on the development and application of the solver.

- A significant advantage of the hybrid solver is its versatility and ease of use in multiple aspects. First, running a case with *VPMFoam* is much simpler since the user does not need to spend time constructing complex meshes. A very simple mesh around the object is sufficient (e.g., cylindrical for a cylinder, C-shaped, cylindrical, or elliptical for an airfoil), without requiring distinct boundary patches; the entire outer boundary can be treated as a single entity, referred to here as the numerical boundary. The rest of the domain is handled by particles, which are initialized in the Eulerian domain on a structured grid and advected downstream. The only parameters that need to be specified are the overlap ratio, λ, and the particle spacing, h. This simplification becomes even more beneficial in multibody cases, where traditional meshing becomes particularly challenging, a point that will be further discussed later in this chapter.

  Furthermore, the hybrid solver offers a high degree of versatility in managing the particles. Particles can be removed based on criteria such as their circulation or location, new particles can be introduced to model vorticity generation, and particles can even be merged, as demonstrated in the works of Stock et al. [37] and Rossi [125].

  Additionally, the solver is flexible with respect to time-step management. The Eulerian part of the solver requires smaller time-steps to accurately resolve the boundary layer and vorticity generation, whereas particles, particularly those far from boundary layers, can evolve with larger time-steps. In the current version of the code, the entire Lagrangian solver evolves with a single time-step. However, this can be modified to allow different sets of particles to evolve at different time-steps, similar to the approach taken by Billuart et al. [65], where particles had two distinct redistribution frequencies.

- OpenFOAM, being open-source, provided the ideal framework for the coupling with the VPM solver due to its flexibility and extensive capabilities. The versatility offered by OpenFOAM makes it highly suitable for integrating with other software and custom-built codes. Various coupling techniques are possible, with one powerful option being *preCICE* [56], which allows for efficient, numerically stable coupling between different solvers by facilitating data exchange between them in partitioned multi-physics simulations.

  Here, the coupling between the VPM and OpenFOAM was achieved by modifying the *pimpleFoam* solver into a class-based structure, named *EulerianPimpleFoam*[1]. This approach provided enhanced flexibility by transforming OpenFOAM from an

[1] The original code was developed by Carlos F. Baptista (personal communication). Significant modifications have been made to this work.

application solver into a modular structure, enabling custom workflows and coupling strategies. Interestingly, newer versions of OpenFOAM (v11 and later) have implemented modular solvers that offer similar flexibility, replacing traditional solvers like *pimpleFoam* with structures such as *incompressibleFluid*. These updates allow for more versatile solver functionality, facilitating multi-physics coupling in a way that mirrors the custom modularity introduced in this work. However, this flexibility was not available in the earlier versions of OpenFOAM, including OpenFOAM v4, which was used by Carlos F. Baptista, and OpenFOAM v9, which was used here. These versions required the custom development of modularity and solver flexibility seen in this dissertation.

By using a Python wrapper, full control of OpenFOAM was integrated into a Python interface, allowing flexible management of intermediate steps such as data imports, exports, and synchronization between solvers. This structure greatly enhances versatility, particularly in hybrid simulations where complex coupling is required.

- The solver demonstrated excellent accuracy in simulating various types of flows, comparable to the precision achieved by pure Eulerian solvers like *pimpleFoam* in OpenFOAM. Specifically, the solver was validated against static cases without solid bodies (such as the Lamb-Oseen vortex and dipole cases), as well as static cases with solid bodies (flow around a cylinder and flow around an airfoil). It also handled dynamic cases, including the traveling cylinder, the rotating cylinder (Magnus effect), and the pitching airfoil. Additionally, the solver was tested in cases incorporating actuator models (such as actuator disk and actuator cylinder) and delivered accuracy comparable with OpenFOAM.

  The solver successfully simulated more complex fluid phenomena, such as the Magnus effect at various rotational speeds, the static stall of an airfoil, and the dynamic stall of an airfoil. This comprehensive validation process establishes confidence in the hybrid solver's ability to handle external aerodynamic simulations, making it a reliable tool for such cases. The only remaining step for completing the 2D validation of the solver is its application to flows in the turbulent regime.

- *VPMFoam* demonstrated excellent accuracy in simulating multibody problems. One key advantage of the solver is its ability to simplify the discretization of the computational domain. Very simple meshes are created for each object in the flow, while the rest of the domain is handled by particles that are free to move. In contrast, for the same problem, a pure Eulerian solver requires much more effort and time in mesh construction. A simpler approach in Eulerian solvers is to use overset mesh techniques, but in OpenFOAM, this feature is only available in the version developed by *ESI-OpenCFD* [48]. Additionally, when the objects are in close proximity, the meshing process becomes increasingly complex because the mesh needs to maintain high resolution around the objects while avoiding mesh distortion or poor cell quality due to overlapping regions. On the other hand, the hybrid solver successfully ran simulations even when the subdomains nearly touched other solid bodies. All cases, even those with very close objects

and significant domain overlap, produced results that were in very good agreement with corresponding pure Eulerian solvers. The hybrid solver offers an elegant and straightforward method for handling multibody problems, where each Eulerian subdomain is treated independently and solved, with particles interconnecting the different regions.

A potential issue in the hybrid solver arises when simulating multibody interactions in which the mesh of one object intrudes into the solid body of another as they move very close to each other. When this occurs, the Eulerian solver within the hybrid framework could incorrectly assign a velocity to the overlapping region, which is physically unrealistic. A potential solution is to introduce constraints on certain parts of the mesh that would force the velocity in those regions to be zero. However, this requires identifying these regions dynamically at each time-step and applying the appropriate constraints. OpenFOAM offers the capability to implement such constraints using the *fvConstraints* feature [126], but this case warrants further exploration. This kind of case does not pose a problem only for hybrid solvers; an issue will most probably arise in pure Eulerian solvers as well. When the objects are in very close proximity and morphing meshes are used, the mesh can become highly skewed. This leads to significant numerical challenges, such as cell distortion and instability, which degrade the accuracy of the simulation.

- An important numerical phenomenon was revealed in hybrid solvers using a redistribution method when simulating moving bodies. In hybrid solvers that do not incorporate surface elements to capture wall vorticity in the Lagrangian solver, such as in the present work and in the work by Billuart et al. [65], a constant shift in the distribution of particles near the solid boundary occurs as the mesh moves through space (see Figure 6.11). The Eulerian mesh, along with the solid body, moves, while the particles are continuously redistributed at specific points on a fixed Lagrangian grid, altering their distribution. These particles, carrying the highest vorticity in the flow, significantly impact the boundary conditions. Consequently, high-frequency oscillations arise in the boundary conditions, affecting the forces acting on the body. While these oscillations do not alter the mean drag coefficient, they increase the amplitude of force oscillations. However, these oscillations can be mitigated either by increasing the particle resolution or by synchronizing the movement of the particle grid with that of the Eulerian mesh.

- When addressing the efficiency of the code, especially in comparison to OpenFOAM, an important distinction must be made. The key question is: “Which part of the domain needs to be resolved with high accuracy?”. This distinction is elaborated in Chapter 8, where the performance evaluation of *VPMFoam* is discussed. It was shown that if the primary objective is to compute aerodynamic coefficients, OpenFOAM can generally run quite efficiently. However, the simulation effort can increase significantly if an accurate representation of the wake is also required, as the number of cells in the wake region needs to be increased substantially. This challenge becomes even more pronounced when simulating multibody cases.

  In contrast, *VPMFoam* exhibited excellent performance in cases where wake preservation is critical, due to the advantage of the Lagrangian solver in reducing artifi-

cial diffusion. Particularly in cases with dense wakes, *VPMFoam* demonstrated a potential speedup of approximately five times, using a time-step for the Lagrangian part that was three times larger than that used for the Eulerian part, and also assuming that the Eulerian portion is parallelized across multiple cores.

Here, it is important to mention that for the pure OpenFOAM simulations, the speedup for the dense wake case was achieved using 64 cores of a single node in DelftBlue [83], which is the total number of cores available on one node. This indicates that such simulations can only be efficiently run on a cluster. On the other hand, *VPMFoam* achieved optimal speedups using just 8 CPU cores and a single GPU, making it feasible to run on local machines such as laptops or personal computers.

As discussed in Chapter 8, this speedup could be further enhanced. In this work, an excessive number of particles was used, covering areas where vorticity is present. While this resulted in a very large number of particles, future improvements aim to reduce this number, which is addressed in the following chapter.

13

Future recommendations

This chapter outlines future recommendations, focusing on potential changes and improvements to the VPMFoam. It discusses areas that warrant further exploration, strategies for increasing code efficiency, possible extensions of the solver, and its potential applications in broader contexts. These recommendations aim to guide future work in refining and expanding the solver's capabilities.

VPMFoam has demonstrated great accuracy and efficiency in the various test cases throughout this dissertation, validating its performance in a range of external aerodynamic applications. However, there is always room for improvement. This chapter presents areas where enhancements can be made, as well as potential applications and further tests that could extend the solver's capabilities. These recommendations are outlined in the bullet points below:

- **Parallelize the Eulerian part**. As discussed in Chapter 8, the Eulerian part of *VPMFoam* currently runs on a single CPU core, without utilizing OpenFOAM's parallel capabilities through MPI. Since OpenFOAM operates within the *VPMFoam* via a Python environment, MPI should be initialized in a way that aligns with OpenFOAM's parallel setup, but through Python. While OpenFOAM already supports this feature, it needs to be enabled for *VPMFoam* as well. Implementing this change would minimize the computational load of the Eulerian part in hybrid simulations due to the small cell count, which would result in a significant impact on the efficiency of the solver.

- **Incorporate turbulence modeling**. OpenFOAM already has robust turbulence modeling capabilities in its Eulerian framework, which can also be utilized in *VPMFoam* for handling external flows at high Reynolds numbers. However, to accurately simulate these flows, turbulence must also be modeled in the Lagrangian component of *VPMFoam*, where particles represent vorticity and flow structures.

  Currently, the Lagrangian particles capture and transport vorticity, but they lack direct turbulence modeling. This presents a limitation in high-Reynolds number flows, where small eddies and turbulent scales are crucial. By introducing Subgrid-scale (SGS) models into the particles, turbulence effects can be better captured, with unresolved eddies modeled similarly to how Large Eddy Simulation (LES) works in the Eulerian framework. This would improve the accuracy of the simulations, particularly in regions far from solid boundaries where Lagrangian particles dominate.

- **Extend the code in 3D**. This dissertation focuses on the 2D version of the solver, but the overall goal is to extend it into three dimensions. The Eulerian part is already compatible with 3D, as OpenFOAM is inherently 3D. Current 2D simulations are achieved by "limiting" the third direction (using 1 cell in the third direction and empty boundary types and boundary conditions). The primary challenge lies in extending the Lagrangian part and the coupling between the two solvers.

  First, vortex stretching (the additional term in 3D, as shown in Equation 2.4) must be incorporated into the evolution step of the Lagrangian solver. Furthermore, functions for calculating induced fields, diffusing, and redistributing, currently implemented for 2D, need to be extended to 3D. While extending these functions is expected to be less difficult, some remarks must be made.

  As mentioned earlier, the calculation of induced fields can be performed serially or in parallel on a CPU, and in parallel on a GPU. However, for the induced velocity (the most computationally demanding part, which must be calculated at each

iteration), FMM libraries are used in 2D. These libraries, developed by Goude et al. [15] and Engblom [16], run in parallel on both CPU and GPU. In 3D, due to the large number of particles and the increased number of calculations, a FMM becomes even more crucial. A suitable library for this is *FMM3D*, an adaptive distributed FMM in 3D, developed by Bull et al. [127] and available as open-source software online at `https://user.it.uu.se/~steng957/freeware.html`. This library is designed for computations involving the scalar potential and thus would require additional development to extend it for use with the Biot-Savart formulation. Nonetheless, adopting FMM3D would represent a valuable extension of the existing FMM code to three-dimensional problems.

In terms of coupling the two solvers, after extending the induced velocity calculation on the Lagrangian side, the boundary conditions calculation will be straightforward. The correction step can be achieved by adapting the method proposed by Billuart et al. [65] and used here, integrating over the faces of a cube instead of a square.

Finally, the number of particles involved in 3D will increase significantly, so efficient particle control and the use of acceleration techniques (with the potential to add more) will be essential for maintaining performance.

- **Run in multiple GPUs**. Currently, the solver runs on a single GPU. The main components leveraging GPU acceleration are the calculation of induced fields (using CUDA) and part of the diffusion process (using CuPy). Expanding these computations to multiple GPUs in a cluster environment would significantly enhance the solvers performance, particularly as the number of particles increases substantially. Efficiently distributing these tasks across multiple GPUs would be crucial in achieving better scalability and reducing computational time in large-scale simulations.

- **Transition the solver in C++**. Currently, *VPMFoam* is built in Python. The main part of the VPM solver is entirely in Python[1]. Over time, critical components have been accelerated using languages like C, C++, and CUDA, as explained in Chapter 2. The communication between the Eulerian (OpenFOAM) and Lagrangian (VPM) components is also managed through Python [1]. Furthermore, OpenFOAM's *pimpleFoam* solver has been wrapped in Python and is controlled via Python interfaces. This Python-based class structure enables seamless coupling with the VPM solver, as outlined in Chapter 12.

  However, with the introduction of modular solvers in OpenFOAM v11 and later, solvers like *pimpleFoam* have been replaced by more flexible and modular alternatives, which inherently facilitate easier coupling between solvers. This presents an opportunity for further improvement.

  A potential future enhancement would be to rewrite the entire hybrid solver, including the VPM and the communication framework, in C++, OpenFOAM's native

[1] The original code was developed by Palha et al. [31] in the context of pHyFlow solver. Significant modifications have been made to this work.

language. By doing this, the VPM solver could be integrated directly into OpenFOAM as a library. This approach would eliminate the need for Python in the workflow and streamline the process for users already familiar with OpenFOAM. Instead of learning to operate a separate *VPMFoam* code, users could run the hybrid solver just like any standard OpenFOAM case. This integration would not only improve performance by minimizing the overhead of cross-language communication but also make the hybrid solver more accessible to a broader range of users who are proficient with OpenFOAM's C++ environment.

- **Explore coupled Dirichlet/Neumann boundary conditions**. To enhance the accuracy of the velocity boundary conditions, it is worth exploring alternatives beyond the Dirichlet conditions currently used in the solver. Presently, Dirichlet boundary conditions for the velocity field across the numerical boundary introduce challenges, as artificial vorticity can reflect off the boundary. This is mitigated by eliminating $2-3$ cells during the correction step. The underlying issue stems from the fact that, when assigning the velocity field based on the particles' induced velocity, the actual velocity field is a continuous distribution, but a single value is assigned at the center of each boundary face. Small corrections are then applied along the boundary to ensure that the continuity equation is satisfied.

  To improve boundary conditions, a more nuanced approach could involve using Dirichlet conditions for the velocity on faces where the flow enters the domain, and Neumann conditions for the velocity on faces where the flow exits. For the pressure field, the opposite could be applied: Neumann conditions where the flow enters, allowing the pressure to be calculated naturally, and Dirichlet conditions where the flow exits, ensuring the pressure is well-defined at the outlet.

  This coupled Dirichlet/Neumann approach would allow for more physical flexibility in the boundary treatment. The velocity would be enforced at the inlet while giving the solver freedom to compute velocity naturally at the outlet. Similarly, by specifying pressure at the outlet and leaving it flexible at the inlet, the solver could maintain the correct balance between velocity and pressure across the boundary. Implementing this could reduce artificial reflections at the boundary and improve the overall accuracy of the flow field.

- **Optimize particle count by merging and adaptive diffusion techniques**. In the simulations presented above, the reduction in the number of particles has primarily been achieved by removing particles based on low circulation or their location, such as particles that have drifted far from the solid object. However, particle count can also be reduced through particle merging, a technique applied in other works, such as those by Stock et al. [37] and Rossi [125]. In this approach, particles that are close to one another can be merged into a larger particle, effectively reducing computational load without sacrificing accuracy in regions where high precision is not needed. A code for particle merging was developed and presented in the under-review paper by J.D. Siemaszko, R. Pasolari, and A. van Zuijlen, *Accelerating vortex particle methods by downsampling the vorticity field representation*, which involves merging particles or regridding a set of particles into a coarser dis-

tribution with larger core sizes. However, this merging technique has not yet been incorporated into the hybrid solver.

Applying this merging technique would be especially beneficial for the hybrid solver, as particles far from the solid object can be consolidated into larger particles, dramatically reducing computational effort, as illustrated in Figure 8.4. This would be a particularly valuable optimization for regions where vorticity is weaker or less critical to the accuracy of the solution.

Additionally, another algorithm that fits well with the hybrid solver is the adaptive diffusion method proposed by Stock et al. [37], where particle sizes adjust dynamically during the diffusion step. This adaptive approach allows particles to grow larger in regions far from solid objects, further enhancing computational efficiency.

The main challenges in implementing these improvements in the current version of the hybrid solver are twofold: first, the FMM code used for accelerating the solver is currently designed to handle particles with identical core sizes (though direct GPU-accelerated calculations can accommodate particles of varying core sizes); second, the diffusion with redistribution step, which currently follows the method proposed by Tutty [36], also requires particles to be equally spaced.

Possible solutions to these challenges include adapting the FMM code to handle particles with varying core sizes or switching to direct calculations, and replacing the current diffusion algorithm with the adaptive approach proposed by Stock et al. [37], which would allow for more flexible particle distributions during diffusion.

- **Expand applications**. *VPMFoam* was validated in this dissertation using simple test cases and later applied to more complex scenarios, such as the airfoil stall. Having established the solver's accuracy and versatility, it can now be applied to more realistic case studies, such as the hybrid Darrieus-Savonius turbine discussed in Chapter 11. This time, however, real geometries of the turbines should be used. A potential configuration would divide the domain into three Eulerian subdomainsone for the Savonius turbine and two identical ones for the blades of the Darrieus turbine. Each region would rotate around the central shaft, with the Savonius mesh rotating about its center and the two Darrieus blades undergoing both rotation and translation. This setup would test the solver in a more realistic and challenging scenario.

  Additionally, while the solver was designed for external aerodynamic cases, it could potentially be adapted for other applications, such as internal flow simulations. One real-world example could be the flow in heat exchangers, where cylindrical objects are placed inside pipes to enhance heat transfer. Similar to the multibody case presented in Chapter 7, internal objects and flow could be simulated using separate Eulerian meshes for the walls, where particles would not be inserted. Alternatively, boundary conditions could be applied using an external panel code to simulate interactions at the walls. This would open up new avenues for using the solver beyond external aerodynamics.

IV

Supplementary Material

A

Mathematics of the Vortex Particle Method

Velocity-Vorticity Formulation of N-S equations

The velocity-pressure formulation of the incompressible N-S equation with constant viscosity is :

$$\underbrace{\frac{\partial \mathbf{u}}{\partial t}}_{\text{time derivative}} + \underbrace{(\mathbf{u}\cdot\nabla)\mathbf{u}}_{\text{non-linear advection}} = -\underbrace{\nabla\left(\frac{p}{\rho}\right)}_{\text{pressure gradient}} + \underbrace{\nu\nabla^2\mathbf{u}}_{\text{viscous term}} + \underbrace{\mathbf{f}}_{\text{body accelarations}} \tag{A.1}$$

where

- $\mathbf{u}$ is the velocity field.
- t is the time.
- ∇ is the nabla operator.
- p is the pressure field.
- ρ is the mass density.
- ν is the kinematic viscosity.
- $\mathbf{f}$ is the body acceleration acting on the fluid (e.g. gravity).

The vorticity field is defined as the curl of the velocity field :

$$\boldsymbol{\omega} = \nabla \times \mathbf{u} \tag{A.2}$$

Taking the curl of the Equation A.1 yields:

$$\nabla \times \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = \nabla \times \left(-\nabla \left(\frac{p}{\rho} \right) + \nu \nabla^2 \mathbf{u} + \mathbf{f} \right) \Longrightarrow$$

$$\Longrightarrow \nabla \times \frac{\partial \mathbf{u}}{\partial t} + \nabla \times (\mathbf{u} \cdot \nabla)\mathbf{u} = \nabla \times \left(-\nabla \left(\frac{p}{\rho} \right) \right) + \nabla \times (\nu \nabla^2 \mathbf{u}) + \nabla \times \mathbf{f} \tag{A.3}$$

Each term can now be analyzed separately to simplify the expression:

- **Time derivative**

$$\nabla \times \frac{\partial \mathbf{u}}{\partial t} = \frac{\partial (\nabla \times \mathbf{u})}{\partial t} = \frac{\partial \boldsymbol{\omega}}{\partial t}$$

- **Non-linear advection**

  Using the vector identities :

$$\nabla \times (\mathbf{u} \cdot \nabla)\mathbf{u} = (\mathbf{u} \cdot \nabla)(\nabla \times \mathbf{u}) - (\nabla \times \mathbf{u}) \cdot \nabla \mathbf{u} =$$

$$(\mathbf{u} \cdot \nabla)\boldsymbol{\omega} - \boldsymbol{\omega} \cdot \nabla \mathbf{u} =$$

$$(\mathbf{u} \cdot \nabla)\boldsymbol{\omega} - (\boldsymbol{\omega} \cdot \nabla)\mathbf{u}$$

- **Pressure gradient**

  The curl of a gradient field is by definition zero, so:

$$\nabla \times \left(-\nabla \left(\frac{p}{\rho} \right) \right) = 0$$

- **Viscous term**

$$\nabla \times (\nu \nabla^2 \mathbf{u}) = \nu \nabla^2 (\nabla \times \mathbf{u}) = \nu \nabla^2 \boldsymbol{\omega}$$

- **Body accelaration**

$$\nabla \times \mathbf{f}$$

Combining all the terms together back to Equation A.3 and rearranging, the velocity-vorticity formulation of an incompressible fluid with constant viscosity in 3D yields:

$$\underbrace{\frac{\partial \boldsymbol{\omega}}{\partial t}}_{\text{time derivative}} + \underbrace{(\mathbf{u} \cdot \nabla)\boldsymbol{\omega}}_{\text{advective term}} = \underbrace{(\boldsymbol{\omega} \cdot \nabla)\mathbf{u}}_{\text{stretching term}} + \underbrace{\nu \nabla^2 \boldsymbol{\omega}}_{\text{viscous term}} + \underbrace{\nabla \times \mathbf{f}}_{\text{external forces term}} \tag{A.4}$$

A new term appeared in the equation, which is the stretching term. This term is associated with the stretching and tilting of the vorticity lines, which is essential in turbulence.

During this process, the vortices break into smaller scales, leading to the energy cascade from the larger to the smaller scales.
Equation A.4, in absence of body forces can be written as :

$$\underbrace{\frac{\partial \boldsymbol{\omega}}{\partial t}}_{\text{time derivative}} + \underbrace{(\mathbf{u}\cdot\nabla)\boldsymbol{\omega}}_{\text{advective term}} = \underbrace{(\boldsymbol{\omega}\cdot\nabla)\mathbf{u}}_{\text{stretching term}} + \underbrace{\nu\nabla^2\boldsymbol{\omega}}_{\text{viscous term}} \tag{A.5}$$

In 2D, the stretching term has all its components zero, and the vorticity field has only one non-zero component, making it a scalar. Therefore, in 2D and without any external forces, Equation A.5 can be written as:

$$\begin{aligned} &\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{u}\cdot\nabla)\boldsymbol{\omega} = \cancelto{0}{(\boldsymbol{\omega}\cdot\nabla)\mathbf{u}} + \nu\nabla^2\boldsymbol{\omega} \implies \\ \implies &\underbrace{\frac{\partial \omega}{\partial t}}_{\text{time derivative}} + \underbrace{(\mathbf{u}\cdot\nabla)\omega}_{\text{advective term}} = \underbrace{\nu\nabla^2\omega}_{\text{viscous term}} \end{aligned} \tag{A.6}$$

Continuous Biot-Savart law in 2D

In 2D, the vorticity field has only one non-zero component, which is perpendicular to the 2D plane xy. So, in 2D the vorticity is considered a scalar :

$$\omega = \omega_z = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{A.7}$$

In an incompressible 2D flow, the velocity field can be expressed in terms of a stream-function ψ such that:

$$u = \frac{\partial \psi}{\partial x} \quad , \quad v = -\frac{\partial \psi}{\partial y} \tag{A.8}$$

Combining Equation A.8 and Equation A.7 yields:

$$\begin{aligned} &\omega = \frac{\partial\left(-\frac{\partial \psi}{\partial x}\right)}{\partial x} - \frac{\partial\left(\frac{\partial \psi}{\partial y}\right)}{\partial y} \implies \\ &\implies \omega = -\nabla^2\psi \end{aligned} \tag{A.9}$$

Equation A.9 is a Poisson equation, and it can be solved using Green's function approach. The Green's function G for the 2D Laplacian is :

$$\nabla^2 G(\mathbf{x},\mathbf{x}') = \delta(\mathbf{x}-\mathbf{x}') \tag{A.10}$$

where δ is the Dirac delta function. In two dimensions :

$$G(\mathbf{x},\mathbf{x}') = \frac{1}{2\pi} ln\| \mathbf{x}-\mathbf{x}'\| \tag{A.11}$$

The streamfunction ψ can be written as a convolution of the Green's function with the vorticity distribution:

$$\begin{aligned} \psi(\mathbf{x}) &= -\int_{\mathbb{R}^2} G(\mathbf{x},\mathbf{x}')\omega(\mathbf{x}')d\mathbf{x}' \implies \\ \implies \psi(\mathbf{x}) &= -\frac{1}{2\pi}\int_{\mathbb{R}^2} ln\| \mathbf{x}-\mathbf{x}'\| \,\omega(\mathbf{x}')d\mathbf{x}' \end{aligned} \tag{A.12}$$

Now, taking into account the Equation A.8, the two components of the velocity can be written as:

$$\begin{aligned} u(\mathbf{x}) &= \frac{\partial\psi(\mathbf{x})}{\partial y} = \frac{\partial}{\partial y}\left(-\frac{1}{2\pi}\int_{\mathbb{R}^2} ln\| \mathbf{x}-\mathbf{x}'\| \,\omega(\mathbf{x}')d\mathbf{x}'\right) \implies \\ \implies u(\mathbf{x}) &= -\frac{1}{2\pi}\int_{\mathbb{R}^2} \omega(\mathbf{x}')\frac{\partial}{\partial y}\left(ln\| \mathbf{x}-\mathbf{x}'\| \right)d\mathbf{x}' \implies \\ \implies u(\mathbf{x}) &= -\frac{1}{2\pi}\int_{\mathbb{R}^2} \omega(\mathbf{x}')\frac{y-y'}{\| \mathbf{x}-\mathbf{x}'\|^2}d\mathbf{x}' \end{aligned} \tag{A.13}$$

and

$$\begin{aligned} v(\mathbf{x}) &= -\frac{\partial\psi(\mathbf{y})}{\partial x} = -\frac{\partial}{\partial x}\left(-\frac{1}{2\pi}\int_{\mathbb{R}^2} ln\| \mathbf{x}-\mathbf{x}'\| \,\omega(\mathbf{x}')d\mathbf{x}'\right) \implies \\ \implies v(\mathbf{x}) &= \frac{1}{2\pi}\int_{\mathbb{R}^2} \omega(\mathbf{x}')\frac{\partial}{\partial x}\left(ln\| \mathbf{x}-\mathbf{x}'\| \right)d\mathbf{x}' \implies \\ \implies v(\mathbf{x}) &= \frac{1}{2\pi}\int_{\mathbb{R}^2} \omega(\mathbf{x}')\frac{x-x'}{\| \mathbf{x}-\mathbf{x}'\|^2}d\mathbf{x}' \end{aligned} \tag{A.14}$$

Combining the components, the velocity field $\mathbf{u} = (u, v)$ at point $\mathbf{x}$ due to a vorticity distribution ω can be expressed as:

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi}\int_{\mathbb{R}^2} \omega(\mathbf{x}')\frac{(\mathbf{x}-\mathbf{x}')^\perp}{\| \mathbf{x}-\mathbf{x}'\|^2}d\mathbf{x}' \tag{A.15}$$

where $(\mathbf{x}-\mathbf{x}')^\perp$ is the perpendicular vector to $\mathbf{x}-\mathbf{x}'$, defined as:

$$(\mathbf{x}-\mathbf{x}')^\perp = (y-y', -(x-x'))$$

Discretization into Vortex Particles

To discretize the vorticity field $\omega(\mathbf{x})$ into vortex particles, the continuous vorticity distribution is approximated by a sum of Dirac delta functions, each representing a discrete vortex particle:

$$\omega(\mathbf{x}) \approx \sum_{i=1}^{N} \Gamma_i \delta(\mathbf{x} - \mathbf{x}_i) \tag{A.16}$$

where:

- Γ_i is the circulation of the i-th vortex particle.
- $\mathbf{x}_i = (x_i, y_i)$ is the position of the i-th vortex particle.

Substituting this discrete vorticity distribution into the continuous Biot-Savart law:

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi} \int_{\mathbb{R}^2} \left(\sum_{i=1}^{N} \Gamma_i \delta(\mathbf{x}' - \mathbf{x}_i) \right) \frac{(\mathbf{x} - \mathbf{x}')^{\perp}}{\| \mathbf{x} - \mathbf{x}' \|^2} \, d\mathbf{x}' \tag{A.17}$$

The Dirac delta function $\delta(\mathbf{x}' - \mathbf{x}_i)$ acts as a filter, picking out the contribution precisely at $\mathbf{x}' = \mathbf{x}_i$. This allows the integral to be simplified by evaluating only at these specific points:

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi} \sum_{i=1}^{N} \Gamma_i \int_{\mathbb{R}^2} \delta(\mathbf{x}' - \mathbf{x}_i) \frac{(\mathbf{x} - \mathbf{x}')^{\perp}}{\| \mathbf{x} - \mathbf{x}' \|^2} \, d\mathbf{x}' \tag{A.18}$$

Since the delta function $\delta(\mathbf{x}' - \mathbf{x}_i)$ is nonzero only at $\mathbf{x}' = \mathbf{x}_i$, the integral evaluates to:

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi} \sum_{i=1}^{N} \Gamma_i \frac{(\mathbf{x} - \mathbf{x}_i)^{\perp}}{\| \mathbf{x} - \mathbf{x}_i \|^2} \tag{A.19}$$

Biot-Savart law with Gaussian kernel

Instead of using the Dirac delta function to represent the vorticity of each vortex particle, a smooth kernel function K is used. The vorticity distribution $\omega(\mathbf{x})$ is approximated by:

$$\omega(\mathbf{x}) \approx \sum_{i=1}^{N} \Gamma_i K(\mathbf{x} - \mathbf{x}_i) \tag{A.20}$$

where K is a kernel function centered at $\mathbf{x}_i$. Substituting this into the continuous Biot-Savart law, yields:

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi} \int_{\mathbb{R}^2} \left(\sum_{i=1}^{N} \Gamma_i K(\mathbf{x}' - \mathbf{x}_i) \right) \frac{(\mathbf{x} - \mathbf{x}')^{\perp}}{\| \mathbf{x} - \mathbf{x}' \|^2} \, d\mathbf{x}' \tag{A.21}$$

Since the kernel function $K(\mathbf{x}' - \mathbf{x}_i)$ is centered at $\mathbf{x}_i$, the integral can be simplified by interchanging the summation and the integral:

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi} \sum_{i=1}^{N} \Gamma_i \int_{\mathbb{R}^2} K(\mathbf{x}' - \mathbf{x}_i) \frac{(\mathbf{x} - \mathbf{x}')^{\perp}}{\| \mathbf{x} - \mathbf{x}' \|^2} \, d\mathbf{x}' \tag{A.22}$$

Now, K is defined as a Gaussian kernel:

$$K(\mathbf{x} - \mathbf{x}_i) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}_i \|^2}{2\sigma^2} \right) \tag{A.23}$$

where σ is the standard deviation of the Gaussian, controlling the spread of the vorticity around each particle. The standard deviation σ represents the radius of the particle as well.

Using the Gaussian kernel, the velocity field induced by the i-th vortex particle becomes:

$$\mathbf{u}_i(\mathbf{x}) = -\frac{\Gamma_i}{2\pi} \int_{\mathbb{R}^2} \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\| \mathbf{x}' - \mathbf{x}_i \|^2}{2\sigma^2}\right) \frac{(\mathbf{x}-\mathbf{x}')^\perp}{\| \mathbf{x}-\mathbf{x}' \|^2} d\mathbf{x}' \tag{A.24}$$

Combining the contributions from all particles, the total velocity field is:

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi} \sum_{i=1}^{N} \Gamma_i \int_{\mathbb{R}^2} \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\| \mathbf{x}' - \mathbf{x}_i \|^2}{2\sigma^2}\right) \frac{(\mathbf{x}-\mathbf{x}')^\perp}{\| \mathbf{x}-\mathbf{x}' \|^2} d\mathbf{x}' \tag{A.25}$$

The integral above can be computed, and the equation for the velocity $\mathbf{u}(\mathbf{x})$ can be written as :

$$\mathbf{u}(\mathbf{x}) = -\frac{1}{2\pi} \sum_{i=1}^{N} \Gamma_i \left(1 - \exp\left(-\frac{\| \mathbf{x}-\mathbf{x}_i \|^2}{2\sigma^2}\right)\right) \frac{(\mathbf{x}-\mathbf{x}_i)^\perp}{\| \mathbf{x}-\mathbf{x}_i \|^2} \tag{A.26}$$

The expression for the vorticity field is:

$$\omega(\mathbf{x}) = \sum_{i=1}^{N} \Gamma_i \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\| \mathbf{x}-\mathbf{x}_i \|^2}{2\sigma^2}\right) \tag{A.27}$$

B

Finite volume discretization

The incompressible N-S equations in 3D can be written as:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla\left(\frac{p}{\rho}\right) + \nabla\cdot(\nu\nabla\mathbf{u}) \tag{B.1}$$

The same equation can be written for a generalized scalar ϕ instead of the vector $\mathbf{u}$, and p instead of the ratio p/ρ for simplicity:

$$\frac{\partial \phi}{\partial t} + (\mathbf{u}\cdot\nabla)\phi = -\nabla p + \nabla\cdot(\nu\nabla\phi) \tag{B.2}$$

In the FVM approach, the equation is solved in its integral form, and so the entire equation is integrated over a small control volume dV. As dV approaches zero, the following internal form must hold for every infinitesimally small volume in the domain.

$$\begin{aligned} &\iiint_V \left(\frac{\partial \phi}{\partial t} + (\mathbf{u}\cdot\nabla)\phi\right) dV = \iiint_V \left(-\nabla p + \nabla\cdot(\nu\nabla\phi)\right) dV \implies \\ \implies &\iiint_V \frac{\partial \phi}{\partial t} dV + \iiint_V \left((\mathbf{u}\cdot\nabla)\phi\right) dV = -\iiint_V \nabla p dV + \iiint_V \left(\nabla\cdot(\nu\nabla\phi)\right) dV \end{aligned} \tag{B.3}$$

Figure B.1 illustrates a 3D cuboid finite volume. This control volume features a central point where variables are computed (collocated grid) and six faces where fluxes are determined. The central node is denoted by the letter P (for polar), with adjacent nodes labeled N (north) at the top, S (south) at the bottom, E (east) on the right, W (west) on the left, and T (front) and B (back) representing the front and back nodes, respectively. The nodes are designated with uppercase letters, each of which is associated with a lowercase letter that identifies the corresponding face of the cell.

Equation B.3 is solved for every finite volume in the computational domain, and each term of the equation is discretized. Here, the discretization process for each term of the equation is presented separately. The schemes mentioned here are written in the way

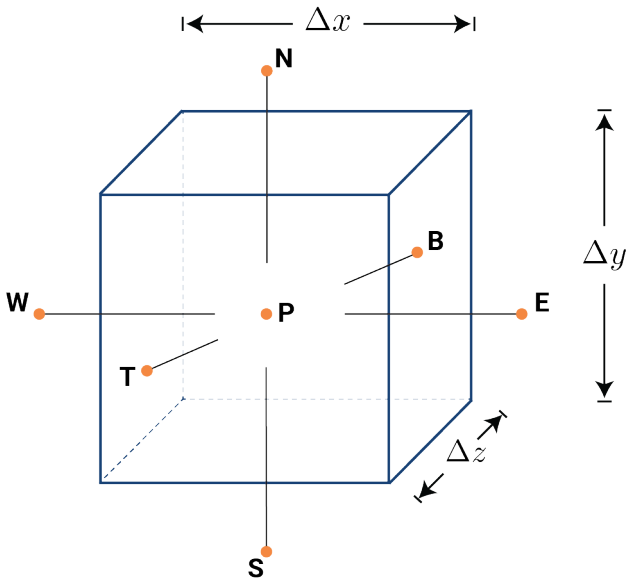


Figure B.1: A cuboid finite volume. The cuboid has a central point (denoted by the letter P), where the flow fields are computed, and six faces where the fluxes are determined. The six central nodes of the adjacent cells are also illustrated, denoted by the letter of their direction: N (north) at the top, S (south) at the bottom, E (east) on the right, W (west) on the left, T (front) and B (back).

that someone can find them in OpenFOAM, but they can also be found in most other CFD software.

- **Temporal term**

$$\iiint_V \frac{\partial \phi}{\partial t} dV = V_P \left(\frac{\partial \phi}{\partial t}\right)_P$$

In OpenFOAM there are different options for discretizing the term $\left(\frac{\partial \phi}{\partial t}\right)_P$ with the most common to be:

  - *steadyState* : sets time derivatives to zero.

$$\left(\frac{\partial \phi}{\partial t}\right)_P = 0$$

  - *Euler* : transient, first order implicit, bounded.

$$\left(\frac{\partial \phi}{\partial t}\right)_P = \frac{\phi_P^{n+1} - \phi_P^n}{\Delta t}$$

– *backward* : transient, second order implicit, potentially unbounded.

$$\left(\frac{\partial \phi}{\partial t}\right)_P = \frac{3\phi_P^{n+1} - 4\phi_P^n + \phi_P^{n-1}}{2\Delta t}$$

Note that n indicates the time level, not the face-orientation index as used elsewhere.

- **Advection term**

$$\iiint_V \big((\mathbf{u}\cdot\nabla)\phi\big)\,dV = \iiint_V (\nabla\cdot\mathbf{u}\phi + \phi\cancelto{0}{\nabla\cdot\mathbf{u}})\,dV = \iiint_V \nabla\cdot\big(\mathbf{u}\phi\big)\,dV \implies$$

$$\implies \iiint_V \nabla\cdot\big(\mathbf{u}\phi\big)\,dV \underset{\text{Theorem}}{\overset{\text{Gauss}}{=}} \oiint_S (\mathbf{u}\cdot\hat{\mathbf{n}})\phi\,dS$$

The surface integral can be transformed into a summation over all the faces around the control volume:

$$\oiint_S (\mathbf{u}\cdot\hat{\mathbf{n}})\phi\,dS = \sum_f (\mathbf{u}_f\cdot\mathbf{S}_f)\phi_f = \sum_f (\mathbf{u}_f\cdot\mathbf{S}_f)\phi_f$$

The term $\mathbf{u}_f\cdot\mathbf{S}_f$ is called the flux through the face f. The velocity $\mathbf{u}_f$ is calculated on the face f using the *linear* interpolation scheme. The most important issue here is the way the advected property ϕ_f is expressed. There are different schemes that can be used to express this property. Some of the most common schemes in OpenFOAM, which are also present in most CFD tools, are:

– *Gauss upwind* : first order bounded.
In the upwind scheme, the value assigned to the property depends on the sign of the flux. For example, for the face n, the flux is $\mathbf{u}_n\cdot\mathbf{S}_n$, and the value of ϕ_n is determined as:

$$\phi_n = \begin{cases} \phi_P & \text{if } \mathbf{u}_n\cdot\mathbf{S}_n > 0 \\ \phi_N & \text{if } \mathbf{u}_n\cdot\mathbf{S}_n < 0 \end{cases}$$

– *Gauss linear*: second order, unbounded.
In the linear scheme, the property is calculated as a linear interpolation between the cell P and the cell adjacent to the face. For example, for the face n, ϕ_n is approximated as:

$$\phi_n = \frac{\phi_P + \phi_N}{2}$$

This averaging assumes that the cells P and N are of equal size and symmetrically arranged. If the grid is non-uniform or the cells differ in size, a proper linear interpolation weighted by the distances to the face from each cell center should be used to preserve accuracy.

- **Gradient term**

$$\iiint_V \nabla p dV \overset{\text{Gauss}}{\underset{\text{Theorem}}{=}} \oiint_S p\hat{\mathbf{n}} dS$$

The surface integral around the volume P can be transformed into a summation over all the faces of the control volume, such that:

$$\oiint_S p\hat{\mathbf{n}} dS = \sum_f p_f \mathbf{S}_f =$$

$$= p_n\mathbf{S}_n + p_s\mathbf{S}_s + p_w\mathbf{S}_w + p_e\mathbf{S}_e + p_b\mathbf{S}_b + p_t\mathbf{S}_t$$

Now the question is how the pressure terms will be calculated, since the values are stored at the cell centers (collocated grid). Different methods to calculate the face values include:

  - *Gauss linear*: second order scheme.
    The face values are calculated as linear interpolations (central differences) between the current cell P and the cell adjacent to the face. For example, the term $p_n\mathbf{S}_n$ yields:

$$p_n\mathbf{S}_n = \frac{p_P + p_N}{2}\mathbf{S}_n$$

  - *Gauss cubic*: third order scheme

There are also schemes that do not make use of the Gauss theorem. For example, the *leastSquares* scheme, which is a second-order scheme, evaluates the gradient term using all the neighboring cells and a least squares finite difference method.

- **Laplacian term**

$$\iiint_V \left(\nabla \cdot (\nu \nabla \phi)\right) dV \overset{\text{Gauss}}{\underset{\text{Theorem}}{=}} \oiint_S \nu \nabla \phi \cdot \hat{\mathbf{n}} dS$$

The integral can be transformed into a summation over the faces around the control volume:

$$\oiint_S \nu \nabla \phi \cdot \hat{\mathbf{n}} dS = \sum_f \nu_f \nabla \phi_f \cdot \mathbf{S}_f =$$

$$= \nu_n \nabla \phi_n \cdot \mathbf{S}_n + \nu_s \nabla \phi_s \cdot \mathbf{S}_s + \nu_w \nabla \phi_w \cdot \mathbf{S}_w + \nu_e \nabla \phi_e \cdot \mathbf{S}_e + \nu_b \nabla \phi_b \cdot \mathbf{S}_b + \nu_t \nabla \phi_t \cdot \mathbf{S}_t$$

Now the values of the gradient and the diffusion coefficient (here the kinematic viscosity ν) should be transferred from the faces to the cell center. For the diffusion coefficient, the scheme used is *linear*, so the coefficient is calculated as an

interpolation between the cell P and the cell adjacent to the face. For example, the value of ν_n is approximated as:

$$\nu_n = \frac{\nu_P + \nu_N}{2}$$

This averaging again assumes that the cells P and N are of equal size and symmetrically arranged. If the grid is non-uniform or the cells differ in size, a proper linear interpolation weighted by the distances to the face from each cell center should be used. The gradient is calculated using the surface normal gradient schemes, denoted as *snGradSchemes* in OpenFOAM. When the face is orthogonal, the gradient is calculated as:

$$\nabla\phi_n = \frac{\phi_N - \phi_P}{|\mathbf{d}|}\hat{\mathbf{n}}_f$$

where $\mathbf{d} = \mathbf{x}_N - \mathbf{x}_P$ is the vector between the centers of the neighboring cells P and N, and $\hat{\mathbf{n}}_f$ is the unit normal vector to the face. This approximation is valid when the face is orthogonal, i.e., when $\mathbf{d}$ is aligned with the face normal.

When the face is not orthogonal, i.e., when the vector connecting the centers of adjacent control volumes is not aligned with the face normal, a non-orthogonal correction must be applied. In OpenFOAM, the surface-normal gradient is typically decomposed into two components. The first is the orthogonal contribution, while the second accounts for the non-orthogonality between the face normal and the vector connecting the neighboring cell centers. OpenFOAM implements several non-orthogonal correction strategies which control the accuracy and stability of the scheme depending on the mesh quality. For more information please refer to the book by Greenshields et al. [46].

These are just a few of the discretization schemes used in OpenFOAM. There are many more schemes available, including those designed for cases where the mesh quality is not as high as the example shown in Figure B.1, which is extremely simple. Readers are encouraged to consult the book by Greenshields et al. [46], the book by F. Moukalled [52], as well as the user guide provided by *The OpenFOAM Foundation* [73].

C

Source code of pimpleFoam solver

The *pimpleFoam* solver in OpenFOAM implements the PIMPLE algorithm, a hybrid of the PISO and SIMPLE algorithms. This hybrid method enables efficient handling of both transient and steady-state simulations by balancing computational cost and accuracy.

The solver begins by including essential header files, followed by entering the main PIMPLE loop. In the first PIMPLE iteration, or whenever *moveMeshOuterCorrectors* is set to true, the mesh is updated. Additionally, quantities such as the finite volume models (*fvModels*) and the fluxes (*phi*) are appropriately corrected to account for changes in the mesh or flow conditions.

The solver then iteratively handles the momentum and pressure equations. This process also includes updating turbulence quantities as necessary, ensuring accurate turbulence modeling during the simulation. The same sequence of corrections is repeated until the specified number of PIMPLE iterations is reached or when a predefined convergence criterion, such as an error threshold, is satisfied.

Thus, the mesh updates and solution advancements occur simultaneously within each PIMPLE loop iteration, ensuring the solution remains consistent with the evolving geometry and flow conditions.

Listing C.1: Source code of the *pimpleFoam* solver in OpenFOAM v9 illustrating the PIMPLE algorithm's implementation.

```
while (pimple.run(runTime))
{
    #include "readDyMControls.H"
    if (LTS)
    {
        #include "setRDeltaT.H"
    }
    else
    {
        #include "CourantNo.H"
```

```
        #include "setDeltaT.H"
    }

    runTime++;

    Info<< "Time = " << runTime.timeName() << nl << endl;

    // --- Pressure-velocity PIMPLE corrector loop
    while (pimple.loop())
    {
        if (pimple.firstPimpleIter() || moveMeshOuterCorrectors)
        {
            fvModels.preUpdateMesh();

            mesh.update();

            if (mesh.changing())
            {
                MRF.update();

                if (correctPhi)
                {
                    #include "correctPhi.H"
                }

                if (checkMeshCourantNo)
                {
                    #include "meshCourantNo.H"
                }
            }
        }

        fvModels.correct();

        #include "UEqn.H"

        // --- Pressure corrector loop
        while (pimple.correct())
        {
            #include "pEqn.H"
        }

        if (pimple.turbCorr())
        {
            laminarTransport.correct();
            turbulence->correct();
        }
    }
    runTime.write();
}
```

D

VPMFoam folder structure

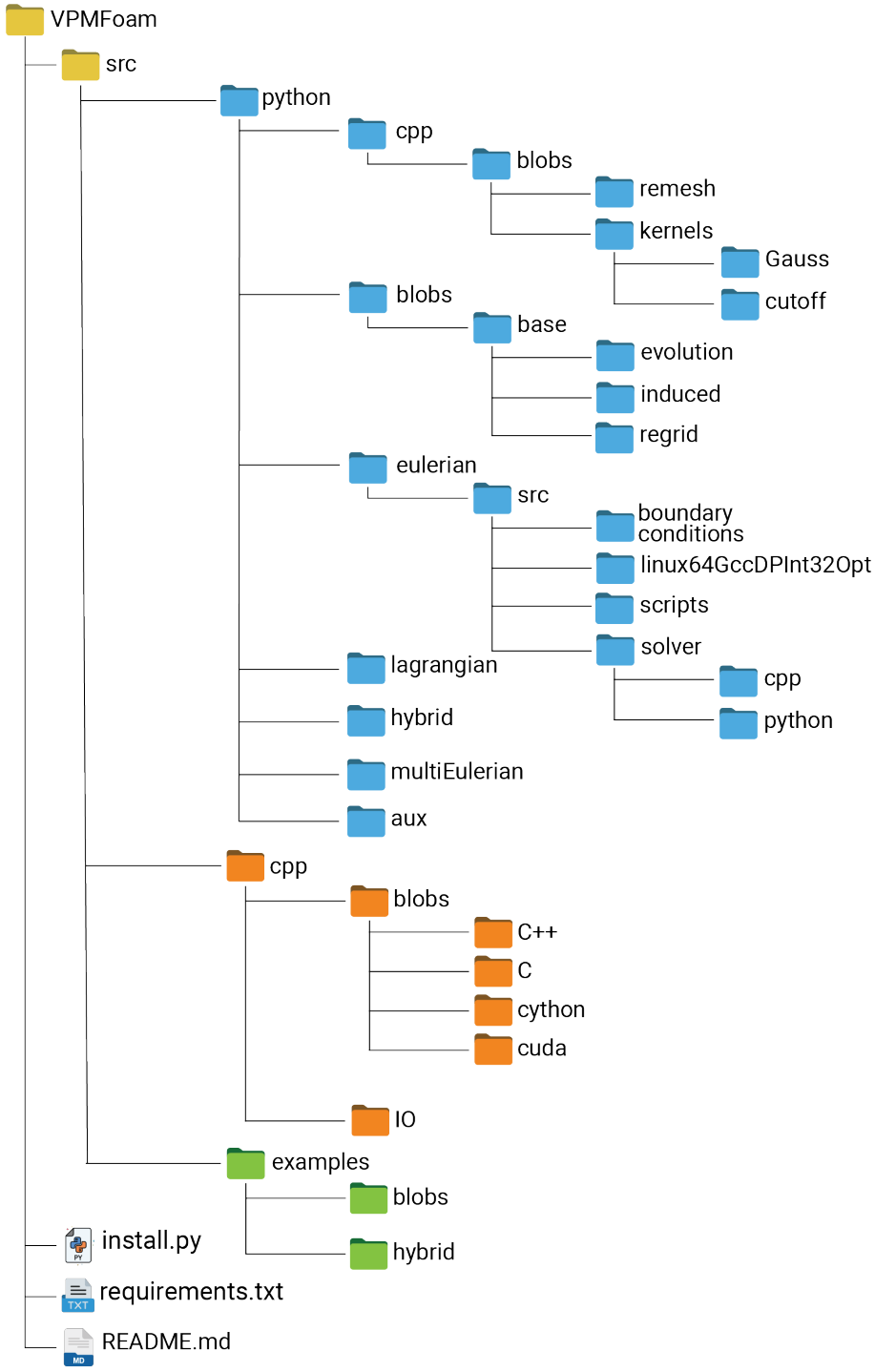


Figure D.1: Folder structure diagram of the hybrid solver *VPMFoam*.

E

Calculation of pressure gradient using vortex particles

To determine the pressure gradient induced by a set of vortex particles, it is useful to first analyze the contribution from a single particle. The pressure gradient at a point $\mathbf{x} = (x, y)$, due to a vortex particle i located at $\mathbf{x}_i = (x_i, y_i)$, can be derived from the N-S equations. In this context, p denotes the pressure normalized by the constant density ρ for simplicity. The relative position vector between the point of interest and the vortex particle is expressed as $\mathbf{r}_i(\mathbf{x}, \mathbf{x}_i(t)) = \big[x - x_i(t), y - y_i(t)\big]$, and so the corresponding magnitude is given by $\|\mathbf{r}_i\| = \sqrt{(x - x_i(t))^2 + (y - y_i(t))^2}$.

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} \implies \\ \implies \nabla p = -\frac{\partial \mathbf{u}}{\partial t} - (\mathbf{u} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \mathbf{u} \end{aligned} \tag{E.1}$$

From this equation, the pressure gradient can be derived if all the terms on the RHS are calculated. Each term will be treated separately in the following steps.

- **Temporal term**

$$\frac{\partial \mathbf{u}(\mathbf{r}_i(t))}{\partial t} \underset{\text{rule}}{\overset{\text{Chain}}{=}} \nabla_{\mathbf{r}} \mathbf{u}(\mathbf{r}(t)) \cdot \frac{d\mathbf{r}_i(t)}{dt} \tag{E.2}$$

An equivalent of $\nabla_{\mathbf{r}}$ can be derived using the following equations:

$$\mathbf{x} = \mathbf{r}_i + \mathbf{x}_i \implies \nabla_{\mathbf{x}} \mathbf{x} = \nabla_{\mathbf{x}} \mathbf{r}_i(t) + \nabla_{\mathbf{x}} \mathbf{x}_i(t) \implies \mathbf{I} = \nabla_{\mathbf{x}} \mathbf{r}_i(t) \implies \nabla_{\mathbf{x}} \mathbf{r}_i(t) = \mathbf{I} \tag{E.3}$$

$$\nabla_{\mathbf{x}} = \frac{\partial}{\partial \mathbf{x}} = \frac{\partial \mathbf{r}_i(t)}{\partial \mathbf{x}} \frac{\partial}{\partial \mathbf{r}_i(t)} + \cancelto{0}{\frac{\partial \mathbf{x}_i(t)}{\partial \mathbf{x}}} \frac{\partial}{\partial \mathbf{x}_i(t)} \implies$$

$$\implies \nabla_{\mathbf{x}} = \nabla_{\mathbf{x}} \mathbf{r}_i(t) \nabla_{\mathbf{r}} \overset{\text{Eq. E.3}}{\implies} \nabla_{\mathbf{x}} = \mathbf{I} \nabla_{\mathbf{r}} \implies \nabla_{\mathbf{x}} = \nabla_{\mathbf{r}} \tag{E.4}$$

Moreover, given that $\mathbf{r}_i(t) = \mathbf{x} - \mathbf{x}_i(t)$:

$$\mathbf{r}_i(t) = \mathbf{x} - \mathbf{x}_i(t) \implies \frac{d\mathbf{r}_i(t)}{dt} = \frac{d\mathbf{x}}{dt} - \frac{d\mathbf{x}_i(t)}{dt} \implies \frac{d\mathbf{r}_i(t)}{dt} = -\frac{d\mathbf{x}_i(t)}{dt} \tag{E.5}$$

Hence, Equation E.2 can be written in Jacobian matrix form as:

$$\frac{\partial \mathbf{u}(\mathbf{r}_i(t))}{\partial t} = -\nabla_{\mathbf{x}} \mathbf{u}(\mathbf{r}_i(t)) \cdot \frac{d\mathbf{x}_t(t)}{dt} \implies$$

$$\implies \frac{\partial \mathbf{u}(\mathbf{r}_i(t))}{\partial t} = - \begin{bmatrix} \frac{\partial u(\mathbf{r}_i(t))}{\partial x} & \frac{\partial u(\mathbf{r}_i(t))}{\partial y} \\ \frac{\partial v(\mathbf{r}_i(t))}{\partial x} & \frac{\partial v(\mathbf{r}_i(t))}{\partial y} \end{bmatrix} \begin{bmatrix} \dot{x}_i(t) \\ \dot{y}_i(t) \end{bmatrix} \tag{E.6}$$

- **Advection term**

  At the end of the calculation of the temporal term, the gradient of the velocity appears. However, since it also appears in the advective term, the calculation will be presented here.

$$\mathbf{u} \cdot \nabla \mathbf{u} = \begin{bmatrix} u(\mathbf{r}(t)) & v(\mathbf{r}_i(t)) \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial u(\mathbf{r}_i(t))}{\partial x} & \frac{\partial u(\mathbf{r}_i(t))}{\partial y} \\ \frac{\partial v(\mathbf{r}_i(t))}{\partial x} & \frac{\partial v(\mathbf{r}_i(t))}{\partial y} \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{\partial u(\mathbf{r}_i(t))}{\partial x} & \frac{\partial u(\mathbf{r}_i(t))}{\partial y} \\ \frac{\partial v(\mathbf{r}_i(t))}{\partial x} & \frac{\partial v(\mathbf{r}_i(t))}{\partial y} \end{bmatrix} \begin{bmatrix} u(\mathbf{r}_i(t)) \\ v(\mathbf{r}_i(t)) \end{bmatrix} \tag{E.7}$$

where $\nabla_{\mathbf{x}} \mathbf{u}(\mathbf{r}_i(t)) = \begin{bmatrix} \frac{\partial u(\mathbf{r}_i(t))}{\partial x} & \frac{\partial u(\mathbf{r}_i(t))}{\partial y} \\ \frac{\partial v(\mathbf{r}_i(t))}{\partial x} & \frac{\partial v(\mathbf{r}_i(t))}{\partial y} \end{bmatrix}$, being the velocity gradient.

The induced velocity of a single particle, as shown in Appendix A, can be expressed as:

$$\mathbf{u}(\mathbf{x}(t)) = -\frac{1}{2\pi}\Gamma_i\left(1-\exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}\right)\right)\frac{(\mathbf{x}-\mathbf{x}_i)^{\perp}}{\|\mathbf{x}-\mathbf{x}_i\|^2} \tag{E.8}$$

and for simplicity, it can be written as:

$$\mathbf{u}(\mathbf{r}_i(t)) = \phi^{\mathbf{u}}(\mathbf{r}_i(t))\zeta(\|\mathbf{r}_i(t)\|^2/2\sigma^2) \tag{E.9}$$

where $\phi^{\mathbf{u}}(\mathbf{r}_i(t)) = -\frac{1}{2\pi}\Gamma_i\frac{\mathbf{r}_i^{\perp}}{\|\mathbf{r}_i(t)\|^2}$ and $\zeta(\|\mathbf{r}_i(t)\|^2/2\sigma^2) = \left(1-\exp\left(-\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right)\right)$

So, the velocity gradient in a matrix form can be written as:

$$\nabla_{\mathbf{x}}\mathbf{u}(\mathbf{r}_i(t)) = \begin{bmatrix} \frac{\partial}{\partial x}\left(\phi^u(\mathbf{r}_i(t))\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right)\right) & \frac{\partial}{\partial y}\left(\phi^u(\mathbf{r}_i(t))\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right)\right) \\ \frac{\partial}{\partial x}\left(\phi^v(\mathbf{r}_i(t))\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right)\right) & \frac{\partial}{\partial y}\left(\phi^v(\mathbf{r}_i(t))\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right)\right) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial\phi^u(\mathbf{r}_i(t))}{\partial x}\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right)+ & \frac{\partial\phi^u(\mathbf{r}_i(t))}{\partial y}\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right)+ \\ \phi^u(\mathbf{r}_i(t))\frac{\partial}{\partial x}\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right) & \phi^u(\mathbf{r}_i(t))\frac{\partial}{\partial y}\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right) \\ \frac{\partial\phi^v(\mathbf{r}_i(t))}{\partial x}\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right)+ & \frac{\partial\phi^v(\mathbf{r}_i(t))}{\partial y}\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right)+ \\ \phi^v(\mathbf{r}_i(t))\frac{\partial}{\partial x}\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right) & \phi^v(\mathbf{r}_i(t))\frac{\partial}{\partial y}\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right) \end{bmatrix} \tag{E.10}$$

Now, the task is to calculate each derivative term in the matrix above:

$$
\begin{aligned}
\frac{\partial \phi^u(\mathbf{r}_i(t))}{\partial x} &= \frac{\partial}{\partial x}\left(\frac{-\Gamma_i}{2\pi}\frac{-(y-y_i)}{(x-x_i)^2+(y-y_i)^2}\right) = -\frac{\Gamma}{\pi}\frac{(y-y_i)(x-x_i)}{\|\mathbf{r}_i(t)\|^4} \\
\frac{\partial \phi^u(\mathbf{r}_i(t))}{\partial y} &= \frac{\partial}{\partial y}\left(\frac{-\Gamma_i}{2\pi}\frac{-(y-y_i)}{(x-x_i)^2+(y-y_i)^2}\right) = -\frac{\Gamma}{\pi}\frac{(y-y_i)^2-1/2\|\mathbf{r}_i(t)\|^2}{\|\mathbf{r}_i(t)\|^4} \\
\frac{\partial \phi^v(\mathbf{r}_i(t))}{\partial x} &= \frac{\partial}{\partial x}\left(\frac{-\Gamma_i}{2\pi}\frac{(x-x_i)}{(x-x_i)^2+(y-y_i)^2}\right) = \frac{\Gamma}{\pi}\frac{(x-x_i)^2-1/2\|\mathbf{r}_i(t)\|^2}{\|\mathbf{r}_i(t)\|^4} \\
\frac{\partial \phi^v(\mathbf{r}_i(t))}{\partial y} &= \frac{\partial}{\partial x}\left(\frac{-\Gamma_i}{2\pi}\frac{(x-x_i)}{(x-x_i)^2+(y-y_i)^2}\right) = \frac{\Gamma}{\pi}\frac{(x-x_i)(y-y_i)}{\|\mathbf{r}_i(t)\|^4} \\
\frac{\partial}{\partial x}\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right) &= \frac{x-x_i}{\sigma^2}\exp\left(-\frac{(x-x_i)^2}{2\sigma^2}\right) \\
\frac{\partial}{\partial y}\zeta\left(\frac{\|\mathbf{r}_i(t)\|^2}{2\sigma^2}\right) &= \frac{y-y_i}{\sigma^2}\exp\left(-\frac{(x-x_i)^2}{2\sigma^2}\right)
\end{aligned}
\tag{E.11}
$$

Once all the terms of the velocity gradient matrix are known, the velocity gradient can be computed by substituting the variables with their respective values.

- **Diffusion term**

  The diffusion term is given by:

$$
\nu\nabla^2\mathbf{u}(\mathbf{r}_i(t)) = \nu\begin{bmatrix}\frac{\partial}{\partial x}\left(\frac{\partial u(\mathbf{r}_i(t))}{\partial x}\right) \\ \frac{\partial}{\partial y}\left(\frac{\partial v(\mathbf{r}_i(t))}{\partial y}\right)\end{bmatrix}
\tag{E.12}
$$

  Knowing the partial derivatives in the parentheses, the diffusion term can be computed as well.

So the final form of the equation is given by:

$$\nabla p(\mathbf{r}_i(t)) = -\frac{\partial \mathbf{u}(\mathbf{r}_i(t))}{\partial t} - (\mathbf{u}(\mathbf{r}_i(t)) \cdot \nabla)\mathbf{u}(\mathbf{r}_i(t)) + \nu\nabla^2\mathbf{u}(\mathbf{r}_i(t))$$

$$\begin{aligned}
\implies \nabla p(\mathbf{r}_i(t)) = &-\left(-\begin{bmatrix} \frac{\partial u(\mathbf{r}_i(t))}{\partial x} & \frac{\partial u(\mathbf{r}_i(t))}{\partial y} \\ \frac{\partial v(\mathbf{r}_i(t))}{\partial x} & \frac{\partial v(\mathbf{r}_i(t))}{\partial y} \end{bmatrix} \begin{bmatrix} \dot{x}_i(t) \\ \dot{y}_i(t) \end{bmatrix} \right) \\
&- \begin{bmatrix} \frac{\partial u(\mathbf{r}_i(t))}{\partial x} & \frac{\partial u(\mathbf{r}_i(t))}{\partial y} \\ \frac{\partial v(\mathbf{r}_i(t))}{\partial x} & \frac{\partial v(\mathbf{r}_i(t))}{\partial y} \end{bmatrix} \begin{bmatrix} u(\mathbf{r}_i(t)) \\ v(\mathbf{r}_i(t)) \end{bmatrix} + \nu \begin{bmatrix} \frac{\partial}{\partial x}\left(\frac{\partial u(\mathbf{r}_i(t))}{\partial x}\right) \\ \frac{\partial}{\partial y}\left(\frac{\partial v(\mathbf{r}_i(t))}{\partial y}\right) \end{bmatrix} \\
\implies \nabla p(\mathbf{r}_i(t)) = &\begin{bmatrix} \frac{\partial u(\mathbf{r}_i(t))}{\partial x} & \frac{\partial u(\mathbf{r}_i(t))}{\partial y} \\ \frac{\partial v(\mathbf{r}_i(t))}{\partial x} & \frac{\partial v(\mathbf{r}_i(t))}{\partial y} \end{bmatrix} \begin{bmatrix} \dot{x}_i(t) - u(\mathbf{r}_i(t)) \\ \dot{y}_i(t) - v(\mathbf{r}_i(t)) \end{bmatrix} + \nu \begin{bmatrix} \frac{\partial}{\partial x}\left(\frac{\partial u(\mathbf{r}_i(t))}{\partial x}\right) \\ \frac{\partial}{\partial y}\left(\frac{\partial v(\mathbf{r}_i(t))}{\partial y}\right) \end{bmatrix}
\end{aligned} \tag{E.13}$$

And so the pressure gradient due to a set of particles can be obtained from summing all the individual contributions:

$$\nabla p(\mathbf{r}(t)) = \sum_{i=1}^{N} \left(\begin{bmatrix} \frac{\partial u(\mathbf{r}_i(t))}{\partial x} & \frac{\partial u(\mathbf{r}_i(t))}{\partial y} \\ \frac{\partial v(\mathbf{r}_i(t))}{\partial x} & \frac{\partial v(\mathbf{r}_i(t))}{\partial y} \end{bmatrix} \begin{bmatrix} \dot{x}_i(t) - u(\mathbf{r}_i(t)) \\ \dot{y}_i(t) - v(\mathbf{r}_i(t)) \end{bmatrix} + \nu \begin{bmatrix} \frac{\partial}{\partial x}\left(\frac{\partial u(\mathbf{r}_i(t))}{\partial x}\right) \\ \frac{\partial}{\partial y}\left(\frac{\partial v(\mathbf{r}_i(t))}{\partial y}\right) \end{bmatrix} \right) \tag{E.14}$$

Bibliography

[1] J. L. M. Poiseuille and W. H. Herschel. *Experimental investigations upon the flow of liquids in tubes of very small diameter*. 1940.

[2] Manuel Colera and Miguel Pérez-Saborid. "An efficient finite differences method for the computation of compressible, subsonic, unsteady flows past airfoils and panels". In: *Journal of Computational Physics* 345 (2017), pp. 596–617. ISSN: 10902716. DOI: 10.1016/j.jcp.2017.05.046.

[3] Michael Alletto. "Comparison of Overset Mesh with Morphing Mesh: Flow Over a Forced Oscillating and Freely Oscillating 2D Cylinder". In: *OpenFOAMő Journal* 2 (2022), pp. 13–30. DOI: 10.51560/ofj.v2.47.

[4] Elia Daniele. "Wind turbine control in computational fluid dynamics with OpenFOAM". In: *Wind Engineering* 41.4 (2017), pp. 213–225. ISSN: 2048402X. DOI: 10.1177/0309524X17709724.

[5] Zhi Cheng, Fue-Sang Lien, Eugene Yee, and Ji Hao Zhang. "Mode transformation and interaction in vortex-induced vibration of laminar flow past a circular cylinder". In: *Physics of Fluids* 34.3 (Mar. 2022), p. 033607. ISSN: 1070-6631. DOI: 10.1063/5.0080722.

[6] Qiming Zhu and Jinhui Yan. "A moving-domain CFD solver in FEniCS with applications to tidal turbine simulations in turbulent flows". In: *Computers and Mathematics with Applications* 81 (2021), pp. 532–546. ISSN: 08981221. DOI: 10.1016/j.camwa.2019.07.034.

[7] Carlos J. Ruiz-Sánchez, Alejandro Martínez-Cava, Miguel Chávez-Módena, and Eusebio Valero. "An adjoint-based drag reduction technique for unsteady flows". In: *Physics of Fluids* 35.7 (July 2023), p. 073603. ISSN: 1070-6631. DOI: 10.1063/5.0153892.

[8] L. C. Hsu, J. Z. Ye, and C. H. Hsu. "Simulation of Flow Past a Cylinder with Adaptive Spectral Element Method". In: *Journal of Mechanics* 33.2 (2017), pp. 235–247. ISSN: 18118216. DOI: 10.1017/jmech.2016.77.

[9] R. Courant, K. Friedrichs, and H. Lewy. "Über die partiellen Differenzengleichungen der mathematischen Physik". In: *Mathematische Annalen* 100.1 (1928), pp. 32–74. DOI: 10.1007/BF01448839.

[10] Daniel Rettenmaier, Daniel Deising, Yun Ouedraogo, Erion Gjonaj, Herbert De Gersem, Dieter Bothe, Cameron Tropea, and Holger Marschall. "Load balanced 2D and 3D adaptive mesh refinement in OpenFOAM". In: *SoftwareX* 10 (2019), p. 100317. ISSN: 2352-7110. DOI: 10.1016/j.softx.2019.100317.

[11] Jingna Pan, Carlos Ferreira, and Alexander van Zuijlen. “Estimation of power performances and flow characteristics for a Savonius rotor by vortex particle method”. In: *Wind Energy* 26.1 (2023), pp. 76–97. DOI: 10.1002/we.2788.

[12] Georges-Henri Cottet and Petros Koumoutsakos. *Vortex Methods: Theory and Practice*. Cambridge University Press, 2000. ISBN: 9780521630249. DOI: 10.1017/CBO9780511526442.

[13] G. S. Winckelmans. “Vortex Methods”. In: *Encyclopedia of Computational Mechanics Second Edition*. John Wiley & Sons, Ltd, 2017, pp. 1–24. ISBN: 9781119176817. DOI: 10.1002/9781119176817.ecm2055.

[14] Chloé Mimeau and Iraj Mortazavi. “A review of vortex methods and their applications: From creation to recent advances”. In: *Fluids* 6 (2 2021). ISSN: 23115521. DOI: 10.3390/fluids6020068.

[15] Anders Goude and Stefan Engblom. “Adaptive Fast Multipole Methods on the GPU”. In: *Journal of Supercomputing* 63.3 (2013), pp. 897–918. ISSN: 0920-8542. DOI: 10.1007/s11227-012-0836-0.

[16] Stefan Engblom. “On well-separated sets and fast multipole methods”. In: *Applied Numerical Mathematics* 61.10 (2011), pp. 1096–1102. ISSN: 01689274. DOI: 10.1016/j.apnum.2011.06.011.

[17] Qi Hu, Nail A. Gumerov, and Ramani Duraiswami. “GPU accelerated fast multipole methods for vortex particle simulation”. In: *Computers and Fluids* 88 (2013), pp. 857–865. ISSN: 00457930. DOI: 10.1016/j.compfluid.2013.08.008.

[18] Jacob S. Calabretta and Robert A. McDonald. “A three dimensional vortex particle-panel method for modeling propulsion-airframe interaction”. In: *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition* January (2010). DOI: 10.2514/6.2010-679.

[19] Philippe Poncet. “Analysis of an immersed boundary method for three-dimensional flows in vorticity formulation”. In: *Journal of Computational Physics* 228.19 (2009), pp. 7268–7288. ISSN: 10902716. DOI: 10.1016/j.jcp.2009.06.023.

[20] Diego Rossinelli and Petros Koumoutsakos. “Vortex methods for incompressible flow simulations on the GPU”. In: *Visual Computer* 24.7-9 (2008), pp. 699–708. ISSN: 01782789. DOI: 10.1007/s00371-008-0250-z.

[21] Carlos Simão Ferreira, Gijs van Kuik, Gerard van Bussel, and Fulvio Scarano. “Visualization by PIV of dynamic stall on a vertical axis wind turbine”. In: *Experiments in Fluids* 46.1 (2009), pp. 97–108. ISSN: 07234864. DOI: 10.1007/s00348-008-0543-z.

[22] R. Pasolari, C. Ferreira, and A. van Zuijlen. “Coupling of OpenFOAM with a Lagrangian vortex particle method for external aerodynamic simulations”. In: *Physics of Fluids* 35.10 (Oct. 2023), p. 107115. ISSN: 1070-6631. DOI: 10.1063/5.0165878.

[23] John D. Anderson. *Fundamentals of Aerodynamics*. 6th. McGraw-Hill Education, 2016. ISBN: 978-1259129919.

[24] Lorena A. Barba. “Vortex Method for Computing High-Reynolds Number Flows: Increased Accuracy with a Fully Mesh-Less Formulation”. Dissertation (Ph.D.) California Institute of Technology, 2004. DOI: 10.7907/TSR5-DE67.

[25] Philippe Chatelain, Alessandro Curioni, Michael Bergdorf, Diego Rossinelli, Wanda Andreoni, and Petros Koumoutsakos. “Billion vortex particle direct numerical simulations of aircraft wakes”. In: *Computer Methods in Applied Mechanics and Engineering* 197.13 (2008), pp. 1296–1304. ISSN: 0045-7825. DOI: 10.1016/j.cma.2007.11.016.

[26] Andrzej Kosior and Henryk Kudela. “Parallel computations on GPU in 3D using the vortex particle method”. In: *Computers & Fluids* 80 (2013). Selected contributions of the 23rd International Conference on Parallel Fluid Dynamics ParCFD2011, pp. 423–428. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2012.01.014.

[27] Diego Rossinelli, Michael Bergdorf, Georges-Henri Cottet, and Petros Koumoutsakos. “GPU accelerated simulations of bluff body flows using vortex particle methods”. In: *Journal of Computational Physics* 229.9 (2010), pp. 3316–3333. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2010.01.004.

[28] Alexandre Joel Chorin. “Numerical Study of Slightly Viscous Flow”. In: *Journal of Fluid Mechanics* 57 (1973), pp. 785–796. DOI: 10.1017/S0022112073002016.

[29] G. Strang. “On the Construction and Comparison of Difference Schemes”. In: *SIAM Journal on Numerical Analysis* 5.3 (1968), pp. 506–517. DOI: https://doi.org/10.1137/0705041.

[30] J. Thomas Beale. “On the Accuracy of Vortex Methods at Large Times”. In: *Computational Fluid Dynamics and Reacting Gas Flows*. New York, NY: Springer New York, 1988, pp. 19–32. ISBN: 978-1-4612-3882-9. DOI: 10.1007/978-1-4612-3882-9_2.

[31] Artur Palha, Lento Manickathan, Carlos Simao Ferreira, and Gerard van Bussel. *A hybrid Eulerian-Lagrangian flow solver*. 2015. arXiv: 1505.03368.

[32] P. Degond and S. Mas-Gallic. “The Weighted Particle Method for Convection-Diffusion Equations. Part 1: The Case of an Isotropic Viscosity”. In: *Mathematics of Computation* 53.188 (1989), pp. 485–507. ISSN: 00255718, 10886842. DOI: 10.2307/2008716.

[33] A Leonard. “Vortex methods for flow simulation”. In: *Journal of Computational Physics* 37.3 (1980), pp. 289–335. ISSN: 0021-9991. DOI: 10.1016/0021-9991(80)90040-6.

[34] Yoshifumi Ogami and Teruaki Akamatsu. “Viscous flow simulation using the discrete vortex modelthe diffusion velocity method”. In: *Computers & Fluids* 19.3 (1991), pp. 433–441. ISSN: 0045-7930. DOI: 10.1016/0045-7930(91)90068-S.

[35] S. Shankar and L.van Dommelen. “A New Diffusion Procedure for Vortex Methods”. In: *Journal of Computational Physics* 127.1 (1996), pp. 88–109. ISSN: 0021-9991. DOI: 10.1006/jcph.1996.0160.

[36] O. R. Tutty. *A Simple Redistribution Vortex Method (with Accurate Body Forces)*. 2010. arXiv: 1009.0166.

[37] Mark J. Stock and Adrin Gharakhani. "Solution-Responsive Particle Size Adaptivity in Lagrangian Vortex Particle Methods". In: vol. Volume 1: Aerospace Engineering Division Joint Track; Computational Fluid Dynamics. Fluids Engineering Division Summer Meeting. Aug. 2021, V001T02A033. DOI: 10.1115/FEDSM2021-65621.

[38] Ole H. Hald. "Convergence of Vortex Methods for Euler's Equations. II". In: *SIAM Journal on Numerical Analysis* 16.5 (1979), pp. 726–755. ISSN: 00361429. DOI: 10.1137/0716055.

[39] Lamb H. *Hydrodynamics, 6th ed.* Cambridge University Press, Cambridge, 1932, pp. 1–8. ISBN: 9780521458689.

[40] Rention Pasolari, Carlos Simão Ferreira, Alexander van Zuijlen, and Carlos Fernando Baptista. "Dynamic Mesh Simulations in OpenFOAM: A Hybrid EulerianLagrangian Approach". In: *Fluids* 9.2 (2024). ISSN: 2311-5521. DOI: 10.3390/fluids9020051.

[41] Leonhard Euler. *Institutiones Calculi Integralis*. Vol. I-III. Academia Imperialis Scientiarum, 1768–1770. URL: https://scholarlycommons.pacific.edu/euler-works/342/ (visited on 05/30/2025).

[42] Jiyuan Tu, Guan-Heng Yeoh, and Chaoqun Liu. "Chapter 5 - CFD Techniques: The Basics". In: *Computational Fluid Dynamics (Third Edition)*. Ed. by Jiyuan Tu, Guan-Heng Yeoh, and Chaoqun Liu. Third Edition. Butterworth-Heinemann, 2018, pp. 155–210. ISBN: 978-0-08-101127-0. DOI: 10.1016/B978-0-08-101127-0.00005-2.

[43] Shiyi Chen and Gary D. Doolen. "Lattice Boltzmann method for fluid flows". In: *Annual Review of Fluid Mechanics* 30.1 (1998), pp. 329–364. DOI: 10.1146/annurev.fluid.30.1.329.

[44] A. Palha and M. Gerritsma. "A mass, energy, enstrophy and vorticity conserving (MEEVC) mimetic spectral element discretization for the 2D incompressible NavierStokes equations". In: *Journal of Computational Physics* 328 (2017), pp. 200–220. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2016.10.009.

[45] Ningyu Zhan, Rongqian Chen, Yancheng You, and Zelun Lin. "Linear lattice Boltzmann flux solver with compact third-order finite volume method for acoustic propagation simulations on three-dimensional hybrid unstructured grids". In: *Computers & Mathematics with Applications* 156 (2024), pp. 103–120. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2023.12.003.

[46] Christopher Greenshields and Henry Weller. *Notes on Computational Fluid Dynamics: General Principles*. Reading, UK: CFD Direct Ltd, 2022. URL: https://doc.cfd.direct/notes/cfd-general-principles/ (visited on 05/30/2025).

[47] The OpenFOAM Foundation. *OpenFOAM: The Open Source CFD Toolbox*. URL: https://www.openfoam.org (visited on 05/30/2025).

[48] ESI-OpenCFD. *ESI-OpenCFD: OpenFOAM*. URL: https://www.openfoam.com (visited on 05/30/2025).

[49] The OpenFOAM Foundation. *OpenFOAM v9*. URL: https://openfoam.org/download/9-ubuntu/ (visited on 05/30/2025).

[50] Resolved Analytics. *Computational Fluid Dynamics (CFD) Software User Survey*. URL: https://www.resolvedanalytics.com/theflux/comparing-cfd-software-part-5-cfd-software-user-survey-results (visited on 05/30/2025).

[51] enlyft. *Companies using OpenFOAM*. URL: https://enlyft.com/tech/products/openfoam (visited on 05/30/2025).

[52] M. Darwish F. Moukalled L. Mangani. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAMő and Matlab*. 1st. Springer Cham, 2015. ISBN: 978-3-319-16873-9.

[53] OpenFOAM Foundation. *PimpleFoam Class Reference*. URL: https://cpp.openfoam.org/v9/classFoam_1_1pimpleLoop.html (visited on 05/30/2025).

[54] K. Anirudh and S. Dhinakaran. "On the onset of vortex shedding past a two-dimensional porous square cylinder". In: *Journal of Wind Engineering and Industrial Aerodynamics* 179 (2018), pp. 200–214. ISSN: 0167-6105. DOI: 10.1016/j.jweia.2018.03.004.

[55] Gaurav Gokhale and Pankaj Dhatrak. "Performance analysis of vertical axis wind turbines by varying tip-speed ratio using open source CFD solver". In: *AIP Conference Proceedings* 2358.1 (July 2021), p. 110005. ISSN: 0094-243X. DOI: 10.1063/5.0057913.

[56] G Chourdakis, K Davis, B Rodenberg, M Schulte, F Simonis, B Uekermann, G Abrams, HJ Bungartz, L Cheung Yau, I Desai, K Eder, R Hertrich, F Lindner, A Rusch, D Sashko, D Schneider, A Totounferoush, D Volland, P Vollmer, and OZ Koseomur. "preCICE v2: A sustainable and user-friendly coupling library [version 2; peer review: 2 approved]". In: *Open Research Europe* 2.51 (2022). DOI: 10.12688/openreseurope.14445.2.

[57] I.P Christiansen. "Numerical simulation of hydrodynamics by the method of point vortices". In: *Journal of Computational Physics* 13.3 (1973), pp. 363–379. ISSN: 0021-9991. DOI: 10.1016/0021-9991(73)90042-9.

[58] Denis-Gabriel Caprace, Grégoire Winckelmans, and Philippe Chatelain. "An immersed lifting and dragging line model for the vortex particle-mesh method". In: *Theoretical and Computational Fluid Dynamics* 34.1-2 (2020), pp. 21–48. DOI: 10.1007/s00162-019-00510-1.

[59] C. Mimeau, G.-H. Cottet, and I. Mortazavi. "Direct numerical simulations of three-dimensional flows past obstacles with a vortex penalization method". In: *Computers & Fluids* 136 (2016), pp. 331–347. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2016.06.020.

[60] G.H. Cottet. "Particle-grid domain decomposition methods for the Navier-Stokes equations in exterior domains". In: *Lecture in Applied Mathematics, American Mathematical Society* (1991). Series N.Y., pp. 110–118.

[61] Goéric Daeninck. “Developments in Hybrid Approaches: Vortex Method with Known Separation Location; vortex method with near-wall Eulerian solver; RANS-LES coupling”. PhD Thesis. Université Catholique de Louvain, 2006.

[62] Yongjie Shi, Guohua Xu, and Peng Wei. “Rotor wake and flow analysis using a coupled Eulerian-Lagrangian method”. In: *Engineering Applications of Computational Fluid Mechanics* 10.1 (2016), pp. 384–402. ISSN: 1997003X. DOI: 10.1080/19942060.2016.1174887.

[63] Mark J. Stock, Adrin Gharakhani, and Christopher P. Stone. “Modeling rotor wakes with a hybrid OVERFLOW-vortex method on a GPU cluster”. In: vol. 1. 28th AIAA Applied Aerodynamics Conference. 2010. DOI: 10.2514/6.2010-4553.

[64] Mark J. Stock and Adrin Gharakhani. “A Hybrid High-Order Vorticity-Based Eulerian and Lagrangian Vortex Particle Method, the 2-D Case”. In: vol. Volume 1: Aerospace Engineering Division Joint Track; Computational Fluid Dynamics. Fluids Engineering Division Summer Meeting. Aug. 2021. DOI: 10.1115/FEDSM2021-65637.

[65] P. Billuart, M. Duponcheel, G. Winckelmans, and P. Chatelain. “A weak coupling between a near-wall Eulerian solver and a Vortex Particle-Mesh method for the efficient simulation of 2D external flows”. In: *Journal of Computational Physics* 473 (2023). ISSN: 10902716. DOI: 10.1016/j.jcp.2022.111726.

[66] George Papadakis and Spyros G. Voutsinas. “A strongly coupled Eulerian Lagrangian method verified in 2D external compressible flows”. In: *Computers & Fluids* 195 (2019), p. 104325. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2019.104325.

[67] Giorgos Papadakis and Spyros G. Voutsinas. “In view of accelerating CFD simulations through coupling with vortex particle approximations”. In: *The Science of Making Torque from Wind 2014 (TORQUE 2014)*. Vol. 524. 1. 2014. DOI: 10.1088/1742-6596/524/1/012126.

[68] J. L. Guermond and H. Z. Lu. “A domain decomposition method for simulating advection dominated, external incompressible viscous flows”. In: *Computers and Fluids* 29.5 (2000), pp. 525–546. ISSN: 00457930. DOI: 10.1016/S0045-7930(99)00017-1.

[69] Serge G. Huberson and Spyros G. Voutsinas. “Particles and grid”. In: *Computers and Fluids* 31.4-7 (2002), pp. 607–625. ISSN: 00457930. DOI: 10.1016/S0045-7930(01)00077-9.

[70] Abhinav Golas, Rahul Narain, Jason Sewall, Pavel Krajcevski, Pradeep Dubey, and Ming Lin. “Large-scale fluid simulation using velocity-vorticity domain decomposition”. In: *ACM Transactions on Graphics* 31.6 (2012), pp. 1–10. ISSN: 07300301. DOI: 10.1145/2366145.2366167.

[71] M. S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. “The FEniCS Project Version 1.5”. In: *Archive of Numerical Software* 3 (2015). DOI: 10.11588/ans.2015.100.20553.

[72] George Papadakis, Vasilis A. Riziotis, and Spyros G. Voutsinas. “A hybrid LagrangianEulerian flow solver applied to elastically mounted cylinders in tandem arrangement”. In: *Journal of Fluids and Structures* 113 (2022), p. 103686. ISSN: 0889-9746. DOI: 10.1016/j.jfluidstructs.2022.103686.

[73] Christopher Greenshields. *OpenFOAM v11 User Guide*. London, UK: The OpenFOAM Foundation, 2023. URL: https://doc.cfd.direct/openfoam/user-guide-v11 (visited on 05/30/2025).

[74] ESI-OpenCFD: OpenFOAM. *OpenFOAM User Guide: Standard Boundary Conditions*. URL: https://www.openfoam.com/documentation/user-guide/a-reference/a.4-standard-boundary-conditions (visited on 05/30/2025).

[75] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[76] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Python Software Foundation. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[77] ISO. *Programming languages — C — Amendment 1: C integrity (ISO/IEC 9899:1990/AMD 1:1995); English version EN 29899:1993/A1:1996 (foreign standard)*. Geneva, Switzerland: International Organization for Standardization (ISO), 1996. URL: https://shop.standards.ie/en-ie/standards/iso-iec-9899-1990-amd-1-1995-592351_saig_iso_iso_1411932/ (visited on 05/30/2025).

[78] Bjarne Stroustrup. *The C++ Programming Language*. Pearson Education India, 2000. URL: https://www.pearson.de/media/muster/toc/toc_9780133522839.pdf (visited on 05/30/2025).

[79] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. “Cython: The Best of Both Worlds”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 31–39. DOI: 10.1109/MCSE.2010.118.

[80] Rohit Chandra, Leo Dagum, David Kohr, Ramesh Menon, Dror Maydan, and Jeff McDonald. *Parallel Programming in OpenMP*. Morgan Kaufmann, 2001. URL: https://www.vitalsource.com/nz/products/parallel-programming-in-openmp-chandra-rohit-menon-ramesh-v9781558606715 (visited on 05/30/2025).

[81] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. *CUDA, Release: 10.2.89*. 2020. URL: https://developer.nvidia.com/cuda-toolkit (visited on 05/30/2025).

[82] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. "CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations". In: *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*. 2017. URL: http://learningsys.org/nips17/assets/papers/paper_16.pdf (visited on 05/30/2025).

[83] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 2)*. 2024. URL: https://www.tudelft.nl/dhpc/ark:/44463/./DelftBluePhase2.

[84] H.J.H. Clercx and C.-H. Bruneau. "The normal and oblique collision of a dipole with a no-slip boundary". In: *Computers & Fluids* 35.3 (2006), pp. 245–279. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2004.11.009.

[85] P. Koumoutsakos and A. Leonard. "High-resolution simulations of the flow around an impulsively started cylinder using vortex methods". In: *Journal of Fluid Mechanics* 296 (1995), pp. 1–38. DOI: 10.1017/S0022112095002059.

[86] Christophe Geuzaine and Jean-François Remacle. "Gmsh: A 3-D Finite Element Mesh Generator with Built-in Pre- and Post-processing Facilities". In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331. DOI: 10.1002/nme.2579.

[87] R. Pasolari, J. Pan, C.J. Ferreira, and A. van Zuijlen. "Flow over traveling and rotating cylinders using a hybrid EulerianLagrangian solver". In: *Computers & Fluids* 279 (2024), p. 106327. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2024.106327.

[88] The OpenFOAM Foundation. *Arbitrary Mesh Interface (AMI)*. URL: https://openfoam.org/release/2-3-0/non-conforming-ami/ (visited on 05/30/2025).

[89] You Wu, Yuting Dai, and Chao Yang. "Time-Delayed Active Control of Stall Flutter for an Airfoil via Camber Morphing". In: *AIAA Journal* 60 (10 2022), pp. 5723–5734. ISSN: 1533385X. DOI: 10.2514/1.J061947.

[90] Yu-long Li, Ren-chuan Zhu, Guo-ping Miao, and Ju Fan. "Simulation of tank sloshing based on OpenFOAM and coupling with ship motions in time domain". In: *Journal of Hydrodynamics, Ser. B* 24.3 (2012), pp. 450–457. ISSN: 1001-6058. DOI: 10.1016/S1001-6058(11)60266-7.

[91] Yichao Chen and Mi-An Xue. "Numerical Simulation of Liquid Sloshing with Different Filling Levels Using OpenFOAM and Experimental Validation". In: *Water* 10.12 (2018). ISSN: 2073-4441. DOI: 10.3390/w10121752.

[92] Eduardo J. Alvarez and Andrew Ning. "High-Fidelity Modeling of Multirotor Aerodynamic Interactions for Aircraft Design". In: *AIAA Journal* 58.10 (2020), pp. 4385–4400. ISSN: 0001-1452. DOI: 10.2514/1.J059178.

[93] Golnesa Karimi-Zindashti and Özgür Kurç. "Flow past rotating cylinders using deterministic vortex method". In: *Ocean Engineering* 291 (2024), p. 116342. ISSN: 0029-8018. DOI: 10.1016/j.oceaneng.2023.116342.

[94] Ming Pingjian and Zhang Wenping. “Numerical Simulation of Low Reynolds Number Fluid-Structure Interaction with Immersed Boundary Method”. In: *Chinese Journal of Aeronautics* 22.5 (2009), pp. 480–485. ISSN: 1000-9361. DOI: 10.1016/S1000-9361(08)60129-6.

[95] Sanjay Mittal and Bhaskar Kumar. “Flow past a rotating cylinder”. In: *Journal of Fluid Mechanics* 476 (2003), pp. 303–334. DOI: 10.1017/S0022112002002938.

[96] R. Pasolari, C. J. Ferreira, and A. van Zuijlen. “Flow around a pair of 2D cylinders using a hybrid Eulerian-Lagrangian solver”. In: *The Science of Making Torque from Wind (TORQUE 2024)*. Vol. 2767. IOP Publishing, 2024, p. 052006. DOI: 10.1088/1742-6596/2767/5/052006.

[97] Maokun Ye, Hamn-Ching Chen, and Arjen Koop. “Verification and validation of CFD simulations of the NTNU BT1 wind turbine”. In: *Journal of Wind Engineering and Industrial Aerodynamics* 234 (2023), p. 105336. ISSN: 0167-6105. DOI: 10.1016/j.jweia.2023.105336.

[98] Jessica M.I. Strickland and Richard J.A.M. Stevens. “Investigating wind farm blockage in a neutral boundary layer using large-eddy simulations”. In: *European Journal of Mechanics - B/Fluids* 95 (2022), pp. 303–314. ISSN: 0997-7546. DOI: 10.1016/j.euromechflu.2022.05.004.

[99] Chengjian He and Jinggen Zhao. “Modeling Rotor Wake Dynamics with Viscous Vortex Particle Method”. In: *AIAA Journal* 47.4 (2009), pp. 902–915. DOI: 10.2514/1.36466.

[100] Ghazale Kavari, Mojtaba Tahani, and Mojtaba Mirhosseini. “Wind shear effect on aerodynamic performance and energy production of horizontal axis wind turbines with developing blade element momentum theory”. In: *Journal of Cleaner Production* 219 (2019), pp. 368–376. ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2019.02.073.

[101] Gracjan M. Skonecki and James M. Buick. “Numerical Study of Flow around Two Circular Cylinders in Tandem, Side-By-Side and Staggered Arrangements”. In: *Fluids* 8.5 (2023). ISSN: 2311-5521. DOI: 10.3390/fluids8050148.

[102] J.R. Meneghini, F. Saltara, C.L.R. Siqueira, and J.A. Ferrari. “Numerical simulation of flow interference between two circular cylinders in tandem and side-by-side arrangements”. In: *Journal of Fluids and Structures* 15.2 (2001), pp. 327–350. ISSN: 0889-9746. DOI: 10.1006/jfls.2000.0343.

[103] François Pellegrini. *SCOTCH User’s Guide 5.1*. 2009. URL: https://hal.science/hal-00410327v1/file/scotch_user5.1.pdf (visited on 05/30/2025).

[104] R. Pasolari, C. J. Ferreira, and A. van Zuijlen. “EulerianLagrangian hybrid solvers in external aerodynamics: Modeling and analysis of airfoil stall”. In: *Physics of Fluids* 36.7 (July 2024), p. 077135. ISSN: 1070-6631. DOI: 10.1063/5.0216634.

[105] Aarjav Malhotra, Arpan Gupta, and Pradeep Kumar. “Study of Static Stall Characteristics of a NACA 0012 Aerofoil Using Turbulence Modeling”. In: *Innovative Design and Development Practices in Aerospace and Automotive Engineering*. Ed. by Ram P. Bajpai and U. Chandrasekhar. Singapore: Springer Nature Singapore, 2017, pp. 369–378. ISBN: 978-981-10-1771-1. DOI: 10.1007/978-981-10-1771-1_39.

[106] Dilek Funda Kurtulus. “On the Unsteady Behavior of the Flow around NACA 0012 Airfoil with Steady External Conditions at Re=1000”. en. In: *International Journal of Micro Air Vehicles* 7.3 (Sept. 2015). Publisher: SAGE Publications Ltd STM, pp. 301–326. ISSN: 1756-8293. DOI: 10.1260/1756-8293.7.3.301.

[107] G. Di Ilio, D. Chiappini, S. Ubertini, G. Bella, and S. Succi. “Fluid flow around NACA 0012 airfoil at low-Reynolds numbers with hybrid lattice Boltzmann method”. In: *Computers & Fluids* 166 (2018), pp. 200–208. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2018.02.014.

[108] Van Duc Nguyen, Viet Dung Duong, Minh Hoang Trinh, Hoang Quan Nguyen, and Dang Thai Son Nguyen. “Low order modeling of dynamic stall using vortex particle method and dynamic mode decomposition”. In: *International Journal of Micro Air Vehicles* 15 (Jan. 2023). Publisher: SAGE Publications Ltd STM. ISSN: 1756-8293. DOI: 10.1177/17568293221147923.

[109] Ayat Abdulhussien Molaa and Mohammed Abdulwhaab Abdulwahid. “Numerical and experimental study of the impact on aerodynamic characteristics of the NACA0012 airfoil”. In: *Open Engineering* 14.1 (2024), p. 20220506. DOI: doi:10.1515/eng-2022-0506.

[110] Yan Liu, Kailun Li, Jiazhong Zhang, Hang Wang, and Liguang Liu. “Numerical bifurcation analysis of static stall of airfoil and dynamic stall under unsteady perturbation”. In: *Communications in Nonlinear Science and Numerical Simulation* 17.8 (2012), pp. 3427–3434. ISSN: 1007-5704. DOI: 10.1016/j.cnsns.2011.12.007.

[111] Chengyong Zhu, Tongguang Wang, and Wei Zhong. “Combined Effect of Rotational Augmentation and Dynamic Stall on a Horizontal Axis Wind Turbine”. In: *Energies* 12.8 (2019). ISSN: 1996-1073. DOI: 10.3390/en12081434.

[112] A-J. Buchner, M.W. Lohry, L. Martinelli, J. Soria, and A.J. Smits. “Dynamic stall in vertical axis wind turbines: Comparing experiments and computations”. In: *Journal of Wind Engineering and Industrial Aerodynamics* 146 (2015), pp. 163–171. ISSN: 0167-6105. DOI: 10.1016/j.jweia.2015.09.001.

[113] K. Mulleners, K. Kindler, and M. Raffel. “Dynamic stall on a fully equipped helicopter model”. In: *Aerospace Science and Technology* 19.1 (2012), pp. 72–76. ISSN: 1270-9638. DOI: 10.1016/j.ast.2011.03.013.

[114] Brandon Jay M. *Dynamic Stall Effects and Applications To High Performance Aircraft*. Tech. rep. 2003. URL: https://ntrs.nasa.gov/citations/./19910012793 (visited on 05/30/2025).

[115] Shengyi Wang, Derek B. Ingham, Lin Ma, Mohamed Pourkashanian, and Zhi Tao. "Numerical investigations on dynamic stall of low Reynolds number flow around oscillating airfoils". In: *Computers & Fluids* 39.9 (2010), pp. 1529–1541. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2010.05.004.

[116] M.H. Akbari and S.J. Price. "Simulation of dynamic stall for a NACA 0012 airfoil using a vortex method". In: *Journal of Fluids and Structures* 17.6 (2003), pp. 855–874. ISSN: 0889-9746. DOI: 10.1016/S0889-9746(03)00018-5.

[117] Dilek Funda Kurtulus. "Unsteady aerodynamics of a pitching NACA 0012 airfoil at low Reynolds number". en. In: *International Journal of Micro Air Vehicles* 11 (2019). Publisher: SAGE Publications Ltd STM, p. 1756829319890609. ISSN: 1756-8293. DOI: 10.1177/1756829319890609.

[118] Jingna Pan, Carlos Ferreira, and Alexander van Zuijlen. "A numerical study on the bladevortex interaction of a two-dimensional DarrieusSavonius combined vertical axis wind turbine". In: *Physics of Fluids* 35.12 (Dec. 2023), p. 125152. ISSN: 1070-6631. DOI: 10.1063/5.0174394.

[119] Jingna Pan. "Towards hybrid modeling of hybrid VAWT". PhD thesis. Delft University of Technology, 2024. URL: https://repository.tudelft.nl/record/uuid:30929ef5-705d-4fc6-9251-8d159e6b1139 (visited on 05/30/2025).

[120] Rakesh Kumar, Kaamran Raahemifar, and Alan S. Fung. "A critical review of vertical axis wind turbines for urban applications". In: *Renewable and Sustainable Energy Reviews* 89 (2018), pp. 281–291. ISSN: 1364-0321. DOI: 10.1016/j.rser.2018.03.033.

[121] Sigurd Johannes Savonius. "Rotor adapted to be driven by wind or flowing water". US1697574A. 1929. URL: https://patents.google.com/patent/US1697574A.

[122] Georges Jean Marie Darrieus. "Turbine having its rotating shaft transverse to the flow of the current". US1835018A. 1931. URL: https://patents.google.com/patent/US1835018A.

[123] IBIS Power. *POWERNEST*. 2024. URL: https://ibispower.eu/powernest/ (visited on 05/30/2025).

[124] Wei Yu. "The wake of an unsteady actuator disk". PhD thesis. Delft University of Technology, 2018. URL: https://repository.tudelft.nl/record/uuid:0e3a2402-585c-41b1-81cf-a35753076dfc (visited on 05/30/2025).

[125] Louis F. Rossi. "Merging Computational Elements in Vortex Simulations". In: *SIAM Journal on Scientific Computing* 18.4 (1997), pp. 1014–1027. DOI: https://doi.org/10.1137/S1064827595285287.

[126] OpenFOAM Foundation. *fvConstraints Class Reference*. URL: https://cpp.openfoam.org/v12/classFoam_1_1fvConstraints.html (visited on 05/30/2025).

[127] Jonathan Bull and Stefan Engblom. "Distributed and Adaptive Fast Multipole Method in Three Dimensions". In: *Communications in Computational Physics* 30.4 (2021), pp. 959–984. ISSN: 1991-7120. DOI: 10.4208/cicp.OA-2020-0072.

List of Abbreviations

Nomenclature

Greek symbols

α	Angle of attack; used as the spin coefficient in Chapter 6 and as the induction factor in Chapter 11
Γ	Circulation (vortex strength)
δ	Dirac delta function; occasionally used to denote distance
Δ	Finite difference
λ	Overlap ratio
ν	Kinematic viscosity
π	Irrational number pi
ρ	Density
σ	Core radius
Σ	Summation
τ	Time constant
ϕ	Generalized scalar
ψ	Stream function
ω	Vorticity (vector form denoted by $\boldsymbol{\omega}$)
Ω	Region; used as the rotational speed in Chapter 11

Latin symbols

A	Amplitude
C	Line path; occasionally used to denote chord length
C_d	Drag coefficient
C_l	Lift coefficient
Co	Courant number
C_P	Power coefficient
C_T	Thrust coefficient

D	Diameter
$\hat{\mathbf{e}}$	Unit vector
E	Error
f	Force (vector form denoted by $\mathbf{f}$); occasionally used to denote frequency
G	Green's function
h	Particle spacing; when written as h_v, denotes the characteristic diffusion length
$\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}$	Unit vectors in the x-, y-, and z-directions
k	Gaussian cutoff parameter; occasionally used to denote the number of substeps or the reduced frequency
K	Gaussian kernel
L	Length; occasionally used to denote horizontal distance
lim	Limit
M	Torque
$\hat{\mathbf{n}}$	Normal unit vector
$\mathcal{O}$	Order
p	Pressure; occasionally used to denote the convergence order
Q	Force
r	Distance (vector form denoted by $\mathbf{r}$); occasionally used to denote the refinement factor
R	Radius
Re	Reynolds number
S	Surface
St	Strouhal number
t	Time
T	Vertical distance; occasionally used to denote the thrust force
$\mathbf{u}, \mathbf{U}$	Velocity vector
u, v	x- and y-components of the velocity vector $\mathbf{u}$
V	Volume
W	Kernel

$\mathbf{x}$	Position vector
x, y, z	x-, y-, and z-components of the position vector $\mathbf{x}$

Subscripts

0	Initial
θ	Circumferential
A	Analytical
amp	Amplitude
ang	Angular
$Bdry$	Boundary
cell(s)	Eulerian mesh cell(s)
C	Center
E	Eulerian
ext	External
f	Face
F	Front
$glob$	Global
H	Hybrid
i, q	Counters
inf, ∞	Freestream condition
int	Interpolation
L	Lagrangian; occasionally used to denote lower
loc	Local
max	Maximum
n	Normal; occasionally used to denote the north face of a cell, and the time level
osc	Oscillation
p	Particle's property
$pop.control$	Population control
r	Radial
R	Rear

red	Redistribution
rot	Rotation
sim	Simulation
U	Upper
v	Vortex
x, y, z	Directions x, y, and z

Curriculum Vitæ

Rention Pasolari

07-04-1996	Born in Këlçyrë, Albania.

Education

2011-2014	Secondary school - Gymnasium and Lyceum Neapoli Lakonias, Greece
2014-2019	Integrated Master's in Mechanical Engineering and Aeronautics University of Patras, Greece
2019-2020	Master of Science in Mechanical Engineering University of Patras, Greece
2020-2024	Doctor of Philosophy in Wind Energy Delft University of Technology, The Netherlands *Thesis:* VPMFoam: A 2D Incompressible Hybrid Eulerian-Lagrangian Solver for External Aerodynamics *Promotors:* Prof. dr. ir. C.J. Simão Ferreira and Dr. ir. M.I. Gerritsma *Copromotor:* Dr. ir. A.H. Van Zuijlen

List of Publications

8. J.D. Siemaszko, **R. Pasolari**, A. van Zuijlen, *Accelerating vortex particle methods by down-sampling the vorticity field representation"*. (*under review*)

7. **R. Pasolari**, C.J. Ferreira, A. van Zuijlen, *Eulerian-Lagrangian Hybrid Solvers in External Aerodynamics: Modeling and Analysis of Airfoil Stall, doi:10.1063/5.0216634*, Physics of Fluids **36**, 077135 (2024). (*in this thesis*)

6. **R. Pasolari**, C.J. Ferreira, A. van Zuijlen, *Flow around a pair of 2D cylinders using a hybrid Eulerian-Lagrangian solver, doi:10.1088/1742-6596/2767/5/052006*, Journal of Physics: Conference Series **2767** (5), 052006 (2024). (*in this thesis*)

5. **R. Pasolari**, J. Pan, C.J. Ferreira, A. van Zuijlen, *Flow over traveling and rotating cylinders using a hybrid EulerianLagrangian solver, doi:10.1016/j.compfluid.2024.106327*, Computers & Fluids, **Volume 279**, 106327, (2024). (*in this thesis*)

4. **R. Pasolari**, C.J. Ferreira, A. van Zuijlen, C.F. Baptista *Dynamic Mesh Simulations in OpenFOAM: A Hybrid EulerianLagrangian Approach, doi:10.3390/fluids9020051*, Fluids **9**, 51 (2024). (*in this thesis*)

3. **R. Pasolari**, C.J. Ferreira, A. van Zuijlen, *Coupling of OpenFOAM with a Lagrangian vortex particle method for external aerodynamic simulations, doi:10.1063/5.0165878*, Physics of Fluids **35**, 107115 (2023). (*in this thesis*)

2. **R. Pasolari**, P.K. Papadopoulos, P. Svarnas, E. Giannakopoulos, I. Kalavrouziotis, S. Georga, C. Krontiras, *Macroscopic modeling of plasma effects on heat and fluid flow in a dielectric barrier discharge based process for biosolid stabilization, doi:10.1063/1.5144385*, AIP Advances **10**, 045021 (2020).

1. P. Svarnas, E. Giannakopoulos, I. Kalavrouziotis, C. Krontiras, S. Georga, **R. Pasolari**, P.K. Papadopoulos, I. Apostolou, D. Chrysochoou, *Sanitary effect of FE-DBD cold plasma in ambient air on sewage biosolids, doi:10.1016/j.scitotenv.2019.135940*, Science of The Total Environment **705**, 135940 (2020).

VPMFoam

A 2D INCOMPRESSIBLE HYBRID EULERIAN-LAGRANGIAN SOLVER FOR EXTERNAL AERODYNAMICS

Aerodynamics is a constantly evolving field, driven by the demands of diverse and rapidly advancing applications: from aviation and space exploration to renewable energy and modern transportation. At the core of aerodynamic research lies Computational Fluid Dynamics (CFD), with two dominant approaches: Eulerian and Lagrangian. Each has its strengths and limitations. Eulerian methods often suffer from artificial diffusion in sparsely meshed regions, mitigated only by significantly increasing computational cost. Lagrangian methods, while advantageous in many respects, continue to face challenges in accurately resolving boundary layers.

This work introduces VPMFoam, a two- dimensional, incompressible hybrid Eulerian–Lagrangian solver that integrates the open-source platform OpenFOAM with a Vortex Particle Method (VPM). By leveraging the strengths of both approaches, VPMFoam reduces artificial diffusion, preserves vorticity structures with high fidelity, and significantly lowers computational requirements. The solver is rigorously verified across multiple 2D benchmark cases, laying a solid foundation for its application in external aerodynamics.

VPMFoam emerges as a promising, efficient, and accurate tool for the future of aerodynamic simulation, built for the evolving needs of the industry.

R E N T I O N P A S O L A R I