POLITECNICO
MILANO 1863

DEPARTMENT OF AEROSPACE SCIENCE
AND TECHNOLOGY

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

Development of an aerodynamic panel code and a vortex particle wake with the Fast Multipole Algorithm

Supervisor
Prof. Pierangelo Masarati

Master Thesis by
Enrico Sabatino

Academic Year 2019/2020

Politecnico di Milano:
www.polimi.it

Abstract

In the context of *urban mobility* and electric aircraft technology, recent developments have supported the use of distributed propulsion and/or ducted fans, due to a growing interest in UAV/MAV and PAV. Within this frame of reference, the ability to rapidly assess the performance of these design concepts with sufficient fidelity and with relatively low computational cost is key for the current industry. In this way, this paper presents an implementation of an aerodynamic panel code to be integrated with a structural solver, and a viscous Vortex Particle Method (VPM) which has been demonstrated to be a a valuable tool to assess the aerodynamic performance of complex configurations.

The elements of the vortex particle method are summarized, while presenting the theoretical aspects related to the coupling with a classical unsteady panel solver. In order to speed up calculations and control the method accuracy, the development of an *octree* structure suitable to implement a *Cartesian Fast Multipole Method* (FMM) is presented, together with FMM technical aspects and convergence tests. Preliminary validation results carried on for open hovering helicopter rotors are given, comparing VPM results with wind-tunnel measurements. Finally, introductory investigations for a ducted fan configuration are presented.

Keywords: Vortex Particle Method, Shrouded fan configuration, Helicopter wake, Fast Multipole Algorithm, Octree data structure.

Acknowledgments

First of all, I would like to express all my gratitude to professor P. Masarati, not only for being a very passionate teacher but also for offering me the unique opportunity to join a company at the forefront of cutting edge technology and innovation. Hence, it goes without saying that I wish to acknowledge with gratitude *Leonardo Helicopters* and all the people I have met during this journey.
Specifically, I cannot express enough thanks to Davide and Fabrizio for their continued support and invaluable help. In particular, rather than just my company supervisor, I have considered Fabrizio a mentor, not only for his patient guidance but also for the enthusiastic encouragement and useful critiques: his willingness to give his time so generously has been very much appreciated.

Special thanks to my girlfriend Debora, for all her love, support and patience from the distance, especially during the hardest study times.
Finally, my deepest and sincere gratitude to my family for their continuous and unparalleled support: without them, I would have never had the chance to even start this path. Hence, I dedicate my last academic work to them, together with my master's degree.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

This paper is the result of a six months internship at *Leonardo Helicopters*, where this thesis work was developed. In this chapter, the aim of this project is presented, along with a brief summary of contributions and activities carried out at Leonardo.

As a matter of fact, the company is launching a preliminary investigation on ducted fan propellers, which may be one of the major design concepts for aeronautical urban mobility vehicles. Within this frame of reference, a vortex particle method was developed from scratch as part of this thesis work and then employed on such configurations. As it will be underlined later, it has been previously shown that this method, when coupled with a fast summation algorithm, can provide mid-fidelity accuracy results with a relatively low computational cost. Moreover, it has been proven to be robust enough in simulations involving wake mixing and wake-body interactions.

In Chapter 2, the aerodynamic problem is formulated, summarizing on one hand the principles of potential theory and on the other hand the main models adopted for the simulation code. Furthermore, the modeling methodology of a vortex particle method is also briefly presented. The Fast Multipole Method is applied for accelerating the computation of particle to particle interaction: hence, theoretical aspects and practical considerations upon such algorithms are outlined in Chapter 3.

To integrate such dissertation, a literature review on hierarchical data structure is given in Chapter 4, while pointing out the choices that were made for the present work. Therefore, in Chapter 5 the main architecture and the new features of the code are presented.

Finally, some of the tests that were carried out during the code development are presented in Chapter 6, together with preliminary validation tests, to be further examined in the future.
Hence, in closing, the results of simulations made on a ducted fan configuration are reported and discussed.

1.1 A brief work report

Over the last seven months I joined a *Leonardo Helicopters* developer team. They are in charge of the Leonardo aeroelastic proprietary-software, named *GyroX*.

As an intern, I took part in a recently launched project of code renovation, features extension and performance enhancement of the old software.
My work focused on the aerodynamic library of the main source code (*AeroX*), which was completely rewritten. AeroX is designed to handle three-dimensional unsteady aerodynamic simulations around complex configurations exploiting potential methods, that is currently mainly used for helicopter simulations. It can work as a library linked to the structural solver or as a stand-alone program. In this context the new version of the code will also allow to perform a performance assessment of multibody configurations such as the ducted fan rotor.
As the older version it is written in `Fortran`, but now finally exploiting all the wherewithal of the *object oriented paradigms* (OOP) available in the latest standards.
In this way, I had the opportunity to help designing the new AeroX software architecture while personally taking care of the mathematical and aerodynamic technical aspects of the implementation.

In the first part of my internship I developed the source code of a modern

particle wake that so far was missing from AeroX wakes models. At the same time, I implemented an algorithm for particle fast induction (the so called *Fast Multipole Method*) while designing a hierarchical data structure upon which the algorithm itself is built. Apart form simple tests, two literature test cases were carried on, whose results are reported later in this paper.
In order to help the debugging work and to ease the user experience I designed a graphical output framework to visualize results in the *binary VTK* format. As far as output is concerned I gave a minor contribution to a new *Graphical User Interface* (GUI) written in `Visual Basic`, that is currently under development.

The second period was dedicated to writing all those classes which manage solid bodies, following three different aerodynamic modeling approaches. To do so, an aerodynamic meshing tool were written from scratch. Then, the whole implementation was finally tested with a porting of a vortex lattice wake. Furthermore, aerodynamic semi-empirical corrections and the solution set-up were built.

Over the last month the first parallel implementations and related tests were carried on, while setting up more complex simulations that will be reported at the end of this paper.

Chapter 2

Problem formulation

In this chapter, the theoretical aspects and methodologies used by the aerodynamic solver are reviewed and explained step by step. As the vast majority of helicopter aerodynamic solvers, *AeroX* is built upon a *Boundary Element Method* (BEM). Generally speaking this is a numerical framework for solving boundary-value or initial-value problems formulated by the use of *Boundary Integral Equations*. As will be underlined later, in order to solve the problem, alongside this formulation, only domain boundaries need to be discretized.

While this is undeniably a key advantage of the method, it still suffer form its intrinsic limits, due to the fact that it is only applicable to low speed aerodynamics.

Moreover, although the BEM has enjoyed the reputation of easy meshing complicated geometries, its efficiency in solutions has been a problem for analyzing large-scale models. Generally speaking, it requires $\mathcal{O}(N^2)$ operations to compute the coefficients.

Recently, BEM solvers have been coupled with a *Fast Multipole Method* (FMM), originally developed by Rokhlin and Greengard [23] in the mid-1980s. The FMM can be used to accelerate the solutions of BEM solvers, in order to reduce the CPU time in a FMM-accelerated BEM to $\mathcal{O}(N)$. Some of the early research on the fast multipole BEM in applied mechanics can be found in [22] and [19]. Moreover, a comprehensive review of the fast multipole accelerated BEM and the

research work status up to 2002 can be found in [45], while a more recent textbook covering theoretical and practical aspect is given by Yijun [39].

In the first part of this chapter, the boundary integral basics for the Laplace equation will be reported, while in the second part the Vortex Particle Method is introduced. An in-depth presentation of the FMM will be given in Chapter 3.

2.1 Preamble

The differential equations that are generally used in the solution of problems relevant to low-speed aerodynamics are a simplified version of the governing equations of fluid dynamics, which are based upon conservation principles. As a matter of fact, taking advantage of the constant density hypothesis, one could reduce the problem to the solution of a simpler differential equation, known as the Laplace problem.

By the Helmholtz Decomposition Theorem, any rapidly decaying and sufficiently smooth vector field in three dimensions can be expressed as the sum of an irrotational (curl-free) vector field and a solenoidal (divergence-free) one. Hence, Helmholtz decomposition of the velocity field reads

$$\boldsymbol{U}(\boldsymbol{x},t) = \boldsymbol{U}_\infty(t) + \boldsymbol{u}_\phi(\boldsymbol{x},t) + \boldsymbol{u}_\psi(\boldsymbol{x},t) \tag{2.1}$$

where

- $\boldsymbol{U}_\infty$ is the free stream velocity, in general it can be associated with a part independent from the spatial coordinate $\boldsymbol{x}$
- $\boldsymbol{u}_\phi$ is the irrotational part of the velocity field ($\boldsymbol{\nabla} \times \boldsymbol{u}_\phi = 0$), hence

$$\boldsymbol{u}_\phi = \boldsymbol{\nabla}\phi \tag{2.2}$$

- $\boldsymbol{u}_\psi$ is the solenoidal component of the velocity field ($\boldsymbol{\nabla} \cdot \boldsymbol{u}_\psi = 0$), hence

$$\boldsymbol{u}_\psi = \boldsymbol{\nabla} \times \psi \tag{2.3}$$

2.2 Laplace problem

The incompressibility constraint $\boldsymbol{\nabla} \cdot U = 0$ reduces to $\boldsymbol{\nabla} \cdot \boldsymbol{u}_\phi = 0$. Thus, the Laplace equation for the scalar velocity potential is formulated:

$$\nabla^2 \phi(\boldsymbol{x}, t) = 0 \qquad \boldsymbol{x} \in \Omega \tag{2.4}$$

which clearly must be supplied with the proper boundary conditions. Since this is an elliptic equation, one must provide them on all boundaries. For a submerged body in a fluid, the velocity component normal to the body's surface $\delta\Omega_b$ and to other solid boundaries must be zero $\boldsymbol{U} \cdot \hat{\boldsymbol{n}}|_{\delta\Omega_b} = 0$. It can be written in terms of the velocity potential ϕ as:

$$\hat{\boldsymbol{n}} \cdot \boldsymbol{\nabla}\phi = \frac{\partial \phi}{\partial \hat{n}} = \hat{\boldsymbol{n}} \cdot \boldsymbol{u}_\phi = \hat{\boldsymbol{n}} \cdot (\boldsymbol{u}_b - \boldsymbol{U}_\infty - \boldsymbol{u}_\psi) = 0 \tag{2.5}$$

where $\hat{\boldsymbol{n}}$ is a vector normal to the body's surface and $\boldsymbol{u}_b$ is the velocity of the body. Moreover, the disturbance created by the motion should decay far from the body ($\boldsymbol{x} \to \infty$):

$$\lim_{\boldsymbol{x} \to \infty} (\boldsymbol{\nabla}\phi - \boldsymbol{u}_{rel}) = 0 \tag{2.6}$$

where $\boldsymbol{u}_{rel}$ is the velocity seen by an observer moving with the body. In this frame of reference, the irrotational component of the velocity field is the main unknown that must be sought as a solution of the Laplace problem (2.4).
To do so, a generic approach known as *Morino formulation* [41] is provided, supplied with practical considerations.

The main advantage of this operation is to convert the original differential problem on the overall fluid volume into an integral equation over analytically defined surfaces or (more likely) numerically defined ones, i.e. a BEM frame of

reference.

Exploiting a convolution with the Laplace operator and the Green's function[1] for the Laplace problem over the overall fluid domain Ω, one will easily get the following integral equation. In principle, this process could be extended to a generic differential problem, so that once the Green function for that specific equation is known, one must exploit the boundary conditions only.

$$\begin{aligned} 0 &= \int_{\Omega} G(\boldsymbol{x};\boldsymbol{x_0})\boldsymbol{\nabla}^2\phi(\boldsymbol{x},t) \\ &= \int_{\Omega} \boldsymbol{\nabla}\cdot[G(\boldsymbol{x};\boldsymbol{x_0})\boldsymbol{\nabla}\phi(\boldsymbol{x},t) - \phi(\boldsymbol{x},t)\boldsymbol{\nabla}G(\boldsymbol{x};\boldsymbol{x_0})] + \int_{\Omega} \boldsymbol{\nabla}^2 G(\boldsymbol{x};\boldsymbol{x_0})\phi(\boldsymbol{x},t) \\ &= \oint_{\delta\Omega} G(\boldsymbol{x};\boldsymbol{x_0})\boldsymbol{\nabla}\phi(\boldsymbol{x},t)\cdot\hat{\boldsymbol{n}} - \oint_{\delta\Omega} \phi(\boldsymbol{x},t)\boldsymbol{\nabla}G(\boldsymbol{x};\boldsymbol{x_0})\cdot\hat{\boldsymbol{n}} + \int_{\Omega} \boldsymbol{\nabla}^2 G(\boldsymbol{x};\boldsymbol{x_0})\phi(\boldsymbol{x},t) \end{aligned} \tag{2.8}$$

The boundary $\delta\Omega$ of the domain is the union of the body surface $\delta\Omega_b$, the irrotational part of the wake $\delta\Omega_b$ and the surface at infinity S_∞. The latter has no contribution, because of the asymptotic behavior of the perturbation potential for $\boldsymbol{x} \to \infty$. To be more specific, as outlined above, the wake will be described as a combination of vortex lattices and vortex particles. The former represent the irrotational part of the wake, since they are a solution of the Laplace problem,the latter corresponds to the rotational part of the wake, whose influence determine the component of the velocity field associated with $\boldsymbol{u}_\psi$.

Following the very same original approach and introducing the function $E(\boldsymbol{x_0})$:

[1]The Green's function is defined as the solution of the corresponding adjoint impulse response process of the given original problem. For the present case, it means that G will be the solution of $\nabla^2 G(\boldsymbol{x};\boldsymbol{x}_0) = -\delta(\boldsymbol{x}-\boldsymbol{x}_0)$. Taking advantage of the homogeneity of space, G for the three-dimensional Laplace problem will be:

$$G(\boldsymbol{x},\boldsymbol{x_0}) = \frac{1}{4\pi}\frac{1}{|\boldsymbol{x}-\boldsymbol{x_0}|} \tag{2.7}$$

which is also known as the infinite-space Green function.

$$E(\boldsymbol{x_0}) = \begin{cases} 1 & \boldsymbol{x}_0 \in \Omega \\ 1/2 & \boldsymbol{x}_0 \in \delta\Omega \\ 0 & \boldsymbol{x}_0 \notin \Omega \cup \delta\Omega \end{cases} \tag{2.9}$$

such that

$$\int_\Omega \boldsymbol{\nabla}^2 G(\boldsymbol{x}; \boldsymbol{x_0})\phi(\boldsymbol{x}) = -\int_\Omega \boldsymbol{\nabla}^2 \delta(\boldsymbol{x}-\boldsymbol{x}_0)\phi(\boldsymbol{x}) = -E(\boldsymbol{x}_0)\phi(\boldsymbol{x}_0) \tag{2.10}$$

equation 2.8 becomes:

$$E(\boldsymbol{x}_0)\phi(\boldsymbol{x}_0) = -\oint_{\delta\Omega} G(\boldsymbol{x}; \boldsymbol{x_0})\frac{\partial\phi(\boldsymbol{x},t)}{\partial\hat{\boldsymbol{n}}} + \oint_{\delta\Omega} \phi(\boldsymbol{x},t)\frac{\partial G(\boldsymbol{x},\boldsymbol{x}_0)}{\partial\hat{\boldsymbol{n}}} \tag{2.11}$$

The latter is the conventional *Boundary Integral Equation* used in BEM solver, valid for a Laplace problem. If the Laplace equation is provided with a forcing term f (i.e. a Poisson equation), an additional integral term must be introduced. In this context we are interested in a homogeneous equation, so we will concentrate on the above-mentioned formulation.

Equation 2.11 can be rearranged through a sum of a source distribution σ and a doublet distribution μ placed on the boundaries $\delta\Omega$.
As for the source terms, it is defined by the following relation:

$$\sigma(\boldsymbol{x},t) = \hat{\boldsymbol{n}}(\boldsymbol{x},t) \cdot \boldsymbol{\nabla}\phi(\boldsymbol{x},t) \tag{2.12}$$

Moreover, the gradient of the Green's function with respect to the space variable $\boldsymbol{x}$ measures the potential induced in $\boldsymbol{x}_0$ by a unit-value point doublet placed in $\boldsymbol{x}$:

$$\boldsymbol{\nabla} G(\boldsymbol{x}; \boldsymbol{x}_0) = -\frac{1}{4\pi}\frac{\boldsymbol{x}_0 - \boldsymbol{x}}{|\boldsymbol{x}_0 - \boldsymbol{x}|^3} \tag{2.13}$$

With this in mind, equation 2.11 can be expressed explicitly as a function of σ and μ:

$$E(\boldsymbol{x}_0)\phi(\boldsymbol{x}_0,t) = -\frac{1}{4\pi}\oint_{\delta\Omega}\frac{1}{|\boldsymbol{x}_0-\boldsymbol{x}|}\sigma(\boldsymbol{x},t) + \frac{1}{4\pi}\oint_{\delta\Omega}\hat{\boldsymbol{n}}\cdot\frac{\boldsymbol{x}_0-\boldsymbol{x}}{|\boldsymbol{x}_0-\boldsymbol{x}|^3}\mu(\boldsymbol{x},t) \quad (2.14)$$

In view of Equation 2.14, it is possible to establish a fairly general approach to the solution of incompressible potential flow problems. As a matter of fact, the solution of $\nabla^2\phi = 0$ can be obtained by distributing elementary solutions (sources and doublets) on the problem boundaries. These elementary solutions automatically fulfill the boundary condition of Equation 2.6 by providing velocity fields that decay as $\boldsymbol{x} \to \infty$. However, at the point were $\boldsymbol{x} = \boldsymbol{x}_0$ the velocity becomes singular, therefore the basic elements are called singular solutions. For practical applications, basic solutions must be provided with a smoothing radius.

Hence, the solution of a fluid dynamic problem is now reduced of finding the appropriate singularity element distribution over some known boundaries, so that the boundary condition (2.5) will be fulfilled.
To uniquely define the solution of this problem, specifying the boundary conditions is not enough: first, a specific combination of sources and doublets distribution should be made, then the amount of circulation around the surface of the body should be fixed by means of some criterion. These considerations deal mainly with the modeling of the wake but also with the fixing of the wake shedding lines (i.e. their initial orientation and geometry). After the releasing locus is fixed, the three-dimensional equivalent of the Kutta condition must be implemented.

Explicit formulations for point or surface singularities can be found in Katz and Plotkin (2001) [33].

The boundary condition for Equation 2.4 can directly specify a zero normal velocity component $\partial\phi/\partial n = 0$ on the surface $\delta\Omega_b$, in which case the formulation will be a *Neumann problem*. On the other hand, the potential ϕ can be directly

specified on the boundary , so that indirectly the zero normal flow condition will be met: this approach constitutes a *Dirichlet problem*.

2.2.1 Dirichlet boundary condition

For an enclosed boundary, such as the one described by the body, if the condition expressed in Equation 2.5 holds, then the potential inside the body (without internal singularities) will not change ([35]), i.e.

$$\phi \mid_{inner} = \text{const} \tag{2.15}$$

that constant is usually set to zero.

Hence, the potential has to be specified anywhere on $\delta\Omega_b$: for this purposes, Equation 2.14 will come in useful by distributing the singularity elements on the surface, while placing the point $\boldsymbol{x}_0$ inside the surface.
Before doing so, one must retrieve the source distribution. Given the far-field velocity, the motion of the bodies and the rotational part of the velocity field $\boldsymbol{u}_\psi$, the source intensity distribution can be easily get by means of its definition. As for $\boldsymbol{u}_\psi$, it is part of the unknowns of the problem, but in this context is a given datum, since that field is the result of the rotational part of the problem, solved at the previous iteration. In other words, the two velocity fields ($\boldsymbol{u}_\phi$ and $\boldsymbol{u}_\psi$) can be solved alternatively using the known field to compute the other.

In this way, the intensity of the source distribution reads:

$$\sigma(\boldsymbol{x}, t) = \hat{\boldsymbol{n}} \cdot \boldsymbol{\nabla}\phi = \hat{\boldsymbol{n}} \cdot (\boldsymbol{u}_b - \boldsymbol{U}_\infty - \boldsymbol{u}_\psi - \boldsymbol{u}_\phi^w) \tag{2.16}$$

where $\boldsymbol{u}_\phi^w$ is the velocity contribution coming from the whole vortex lattice wake. To define the problem uniquely, the wake doublet distribution (no sources contribution are present, i.e. the wake is always modeled as a thin sheet of doublets) should be known or related to the unknown doublets of the body (here it comes in

to play the Kutta condition and the releasing algorithm). Here, $\boldsymbol{u}_\phi^w$ stands for the velocity contribution of all those inductive elements released in the field, linked with the irrotational part of the velocity field.

At this point, the doublet distribution must be sought as the solution of a numerical system.
To simplify the notation, two convolution kernels $\mathcal{S}$ and $\mathcal{G}$ are introduced, respectively for the source and doublet contributions. Hence, the expression of the potential 2.14, when condition 2.15 is applied on the body, will read

$$\left(\left[E\delta + \mathcal{G} + \mathcal{G}^{Kutta}\right] \star \mu\right)(\boldsymbol{x}, t) = (\mathcal{S} \star \mu)(\boldsymbol{x}, t) \tag{2.17}$$

where the kernel $\mathcal{G}^{Kutta}$ gives the potential contribution of the wake at the current timestep (i.e. an implicit contribution), written as a function of the unknowns on the body, by means of the Kutta condition. The explicit contribution of the wake (i.e. the one coming from the previous timesteps) is taken into account with the term $\boldsymbol{u}_\phi^w$ that builds the right-hand side of the equation.
This integral equation can be solved with a collocation method. First, the surface of the body $\delta\Omega_b$ is divided into two-dimensional panels S_k and the equation 2.17 is recast, extending the integrals over the surface of each panel element (generally a quadrilateral or a triangle). Then, the resulting equation is split and rearranged as a linear system of equations, by evaluating it in suitable collocation points.

The choice of the singularity distribution on the panels deserve a more detailed aside. The general practice is to consider a low-order approach (especially for aeroelastic problems), introducing constant intensity distribution for the panels, both for the sources and for the doublets. With this choice, flat panels can be built. This is the path followed by this thesis work.
On the other hand, higher-order methods must take into account not only higher derivatives of surface singularity density, but also higher geometric derivatives. In this manner, a so called *higher order panels method* can be implemented, [28].

The accuracy benefits of such methods are enhanced when the mesh is refined with curved panels. In this way, the governing equations will be the same, but the expressions of induced velocities become more complex to handle, requiring numerical integration on the induced panel. Specifically, such expressions will depend on the geometric derivatives of the body and the singularity distribution derivatives. To save computational time, multipole expansions based on those singularities can be used too.

In this context, constant distributions will be used, i.e. $\mu(\boldsymbol{x},t)|_{S_k}= \mu_k(t)$ and $\sigma(\boldsymbol{x},t)|_{S_k}= \sigma_k(t)$. Hence, in light of what has been said above, the following equation will hold:

$$\begin{aligned} E_i\phi_i &= \sum_{k=1}^{N}\left[-\int_{S_k}\frac{1}{4\pi}\frac{1}{\boldsymbol{x}_i-\boldsymbol{x}_k}\right]\sigma_k - \sum_{k=1}^{N}\left[-\int_{S_k}\frac{1}{4\pi}\frac{\boldsymbol{x}_i-\boldsymbol{x}_k}{|\boldsymbol{x}_i-\boldsymbol{x}_k|^3}\cdot\hat{\boldsymbol{n}}\right]\mu_k \\ &= \sum_{k=1}^{N} S_{ik}\sigma_k - \sum_{k=1}^{N} G_{ik}\mu_k \qquad\qquad i=1:N \end{aligned} \tag{2.18}$$

where k runs over all the N body panels and S_{ik} and G_{ik} are the aerodynamic influence coefficients of the k-th panel on the i-th panel, associated with source and doublet distributions with unit-valued intensity. Finally, exploiting the Kutta condition, Equation 2.17 can be arranged into a linear system of N equations in N unknowns:

$$\sum_{k=1}^{N}(E_i\delta_{ik} + D_{ik} + D_{ik}^{Kutta})\mu_k = \sum_{k=1}^{N} S_{ik}\sigma_k \tag{2.19}$$

which, together with the discrete counterpart of Equation 2.16, closes he problem.

The formulation outlined in this section is implemented in the code, and it is suitable for modeling thick bodies, e.g. wings or fuselages. Generally speaking it requires higher computer time compared to a *lifting surface* approach, that will be described below, in the next section. However, generally speaking, the main bottleneck of the whole method is the induction performed by the wake.

Specifically, the computational cost increases with the number of vortical elements released in the field at each timestep.

2.2.2 Neumann boundary condition

In this case it is required that $\partial\phi/\partial n$ will be specified on the solid boundary $\delta\Omega_b$ (Equation 2.5), hence

$$\hat{\boldsymbol{n}} \cdot \boldsymbol{U} = \hat{\boldsymbol{n}} \cdot \boldsymbol{u}_b \tag{2.20}$$

Substituting the velocity decomposition introduced at the beginning (Equation 2.1), one will get:

$$\hat{\boldsymbol{n}} \cdot \boldsymbol{u}_\phi = \hat{\boldsymbol{n}} \cdot (\boldsymbol{u}_b - \boldsymbol{U}_\infty - \boldsymbol{u}_\psi - \boldsymbol{u}_\phi^w) \tag{2.21}$$

where $\boldsymbol{u}_\phi$ contains both the contributions from the body and from the wake doublets linked to the body by the Kutta condition, while $\boldsymbol{u}_\phi^w$ is the velocity contribution coming from released vortical wake elements. Exploiting this velocity formulation, the sources contributions are nullified, so that a zero-thickness body can be modeled. As previously done, the body surface is divided into N panels S_k and each panel will be assigned an unknown constant-strength distribution of doublets.

Remark. The velocity contribution given by a constant-strength doublet distribution on a panel is equivalent to a vortex ring (proof in [33]). Hence, the contribution coming from this term can be indifferently computed with a doublet or with a vortex ring (e.g. the right-hand side of Equation 2.21)

In our code, we use doublet induction for the body and vortex rings for the vortex lattice wake. In the latter case, the induction influence coefficient can be computed by means of the Biot-Savart law.

Writing the expression 2.21 in an abstract form and exploiting the domain decomposition we will get

$$\hat{\boldsymbol{n}}_i \cdot \sum_{k=1}^{N} \boldsymbol{\Lambda}_{ik}\mu_k = \hat{\boldsymbol{n}}_i \cdot (\boldsymbol{u}_{b,i} - \boldsymbol{U}_\infty - \boldsymbol{u}_{\psi,i} - \boldsymbol{u}^w_{\phi,i}) \qquad i = 1 : N \tag{2.22}$$

where $\boldsymbol{\Lambda}_{ik}$ is the velocity influence coefficient upon panel i generated by panel k.

2.2.3 Aerodynamic forces

In this section, the algorithm involved in the aerodynamic force computations is outlined. It goes without saying that a numerical potential scheme experiences the so called *D'Alambert paradox*, hence there is no way to evaluate profile or drag resistance except by means of semi-empirical corrections.

Thick bodies

The mean pressure acting on a panel k of a body modeled with sources and doublets can be calculated using the unsteady Bernoulli equation:

$$C_{p,k} = \frac{p_k - p_\infty}{\frac{1}{2}\rho|\boldsymbol{U}_\infty|^2} = 1 - \left(\frac{|\boldsymbol{U}_k|}{|\boldsymbol{U}_\infty|}\right)^2 - \frac{2}{|\boldsymbol{U}_\infty|^2}\frac{\partial \phi_k}{\partial t} \tag{2.23}$$

where $C_{p,k}$ is the pressure coefficient and $\boldsymbol{U}_k$ is the total induced and free stream velocity related to panel k; ϕ_k is the potential induced on the panel centroid by all bodies and wakes in the field. For forces computation purposes, the potential contributions coming form wakes and bodies is neglected, hence the self-induced potential only will be considered.

In light of the Dirichlet boundary condition and of the definition of the doublet singularity, which can be seen as the discontinuity in the potential between upper and lower side of the panel, the following will hold:

$$\phi_k = \Delta\phi_k + \phi_k^- = \mu_k \tag{2.24}$$

or, due to the equivalence between a doublet panel singularity and a vortex ring, μ_k is equivalent to the total circulation on the vortex ring itself. Hence, the unsteady term in Equation 2.23 can be rewritten in terms of the doublet intensity on each panel k.
Moreover, the velocity $\boldsymbol{U}_k$ calculated on the panel centroid reads:

$$\boldsymbol{U}_k = \boldsymbol{U}_\infty + \boldsymbol{u}_{\phi,k} + \boldsymbol{u}_{\psi,k} \tag{2.25}$$

Once the pressure coefficient is calculated, the aerodynamic actions can be computed easily with the following:

$$\begin{aligned} \boldsymbol{F}_{aero} &= -\frac{1}{2}\rho|\boldsymbol{U}_\infty + \boldsymbol{u}_{b,k}|^2 \sum_{k=1}^{N} C_{p,k}\boldsymbol{S}_k \cdot \hat{\boldsymbol{n}}_k \\ \boldsymbol{M}_{aero} &= -\frac{1}{2}\rho|\boldsymbol{U}_\infty + \boldsymbol{u}_{b,k}|^2 \sum_{k=1}^{N} C_{p,k}\boldsymbol{S}_k \cdot \hat{\boldsymbol{n}}_k \times (\boldsymbol{O}_{ref} - \boldsymbol{O}_k) \end{aligned} \tag{2.26}$$

where $\boldsymbol{S}_k$ is directed as the panel k normal vector and its magnitude equals the panel k surface area, and O_{ref} is a suitable reduction pole, while O_k is the panel centroid.

Lifting surface

For bodies modeled with lifting surfaces the aerodynamic forces and moments are easily obtained by means of the Kutta-Joukowski theorem, which gives the expression of the aerodynamic force on a vortical element as:

$$\boldsymbol{F}_{aero,k} = \rho\boldsymbol{U}(\boldsymbol{O}_k) \times \Gamma_k \boldsymbol{\Delta l}_k \tag{2.27}$$

where $\boldsymbol{\Delta l}$ is the length of the vortical edge and $\boldsymbol{U}(\boldsymbol{O}_k)$ is the total velocity computed on $\boldsymbol{O}_k$, the edge middle point. Inside $\boldsymbol{U}(\boldsymbol{O}_k)$ there is the free stream velocity and the induced velocity contribution form other bodies and wakes. Γ_k is the vortical element circulation intensity.

Once the aerodynamic force is known, the torque given by that force will be trivially

$$\boldsymbol{M}_{aero,k} = \rho(\boldsymbol{O}_k - \boldsymbol{O}_{ref}) \times \boldsymbol{F}_{aero,k} \tag{2.28}$$

A further contribution to the aerodynamic force comes form the unsteady comportment of the vortical circulation, which adds a force contribution proportional to the time evolution of the circulation itself:

$$\boldsymbol{F}^{us}_{aero,k} = \rho \frac{d\Gamma_k}{dt} \hat{\boldsymbol{n}}_k \tag{2.29}$$

the time derivative can be approximated by means of a finite difference of arbitrary order.

2.3 Vortex particle method

2.3.1 Introduction

One promising mid-fidelity approach for modeling complex wake interactions is the viscous *Vortex Particle Method* (VPM). Multiple vortex panel or filaments methods have been proposed for the modeling of free wakes, but experience in this field reveals that these methods are sometimes ill-conditioned for the modeling of complex dynamics, such as wakes mixing, body-wake interactions and viscous effects. Specifically, the latter often require a careful parameter tuning to get reliable results. One major source of instabilities is the requirement of preserving connectivity, which make classical vortex lattice wakes unable to model complex flow fields.

On the other hand, the vortex particle method discretizes the vorticity field into free particles, solving the vorticity form of Navier-Stokes equations in a Lagrangian framework. In the past, mid/high-fidelity simulations have been attained through VPM at a relatively low computational cost. For instance, Calabretta [10] used particles to model the interaction between an actuator-disk propeller and a paneled wing. He et al [27] [64] [26] implemented a VPM scheme to study the interactions between coaxial rotors, coupling a lifting-line wing with an hybrid VPM-CFD solver to characterize rotor-on-fuselage interaction of a full rotor-craft. Among other applications, VPM has also been used to model rotor-craft forward flight [5], [55].
A complete mathematical formulation of vortex particles method is presented by Winckelmans (1989) [61], while a Cottet and P.D.Koumoitsakos (2001) [13] presented a literature review on particles schemes proposed during the years.

2.3.2 Vorticity equations

From the Stokes' assumptions for a Newtonian, incompressible fluid with constant viscosity, the conservation of momentum is governed by the well-known Navier-

Stokes equations:

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{u} = -\frac{1}{\rho}\nabla p + \nu \boldsymbol{\nabla}^2 \boldsymbol{u} \tag{2.30}$$

where $\boldsymbol{u}(\boldsymbol{x}, t)$ is the velocity field, $p(\boldsymbol{x}, t)$ is the pressure field, ρ the fluid density and ν the kinematic viscosity of the fluid.
Using the identity $(\boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{u} = \frac{1}{2}\boldsymbol{\nabla}(\boldsymbol{u} \cdot \boldsymbol{u}) + (\boldsymbol{\nabla} \times \boldsymbol{u}) \times \boldsymbol{u}$, this relation can also be written as:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{\omega} \times \boldsymbol{u} = -\boldsymbol{\nabla}\left(\frac{p}{\rho} + \frac{|\boldsymbol{u}|^2}{2}\right) + \nu \boldsymbol{\nabla}^2 \boldsymbol{u} \tag{2.31}$$

where $\boldsymbol{\omega}(\boldsymbol{x}, t) = \boldsymbol{\nabla} \times \boldsymbol{u}(\boldsymbol{x}, t)$ is the vorticity field. Taking the curl of Equation 2.31, the momentum equation is then transformed into its vorticity form

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{\nabla} \times (\boldsymbol{\omega} \times \boldsymbol{u}) = \nu \boldsymbol{\nabla}^2 \boldsymbol{\omega} \tag{2.32}$$

With some manipulations and from the continuity equation for an incompressible fluid ($\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0$), both $\boldsymbol{u}$ and $\boldsymbol{\omega}$ are found to be divergence-free fields, so that Equation 2.32 becomes

$$\frac{D\boldsymbol{\omega}}{Dt} = (\boldsymbol{\omega} \cdot \boldsymbol{\nabla})\boldsymbol{u} + \nu \boldsymbol{\nabla}^2 \boldsymbol{\omega} \tag{2.33}$$

which describes the Lagrangian evolution of the vorticity field, a non-linear transport equation that can be solved using a particle method.

In this section,we are interested in determining the vortex particle induced velocity $\boldsymbol{u}_\psi$ induced by the vorticity field $\boldsymbol{\omega}$. So both scalar potential and uniform components will be ignored, in order to develop an expression for the vector potential field $\boldsymbol{\psi}$.

Starting from the vorticity field definition

$$\begin{aligned} \boldsymbol{\omega} &= \boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times \boldsymbol{\psi}) \\ &= \boldsymbol{\nabla}(\boldsymbol{\nabla} \cdot \boldsymbol{\psi}) - \boldsymbol{\nabla}^2 \boldsymbol{\psi} \end{aligned} \tag{2.34}$$

The first term will be nullified due to a gauge transform, so that finally a Poisson equation for $\boldsymbol{\psi}$ is obtained:

$$\boldsymbol{\nabla}^2 \boldsymbol{\psi}(\boldsymbol{x}, t) = -\boldsymbol{\omega}(\boldsymbol{x}, t) \tag{2.35}$$

2.3.3 Singular vortex particles

In the singular vortex particle method, also called method of *point vortices* or *vortex sticks* or *vortons* [61], the vorticity field is discretized through a point-wise representation, the so called *particle representation of vorticity field*:

$$\begin{aligned} \boldsymbol{\omega}(\boldsymbol{x}, t) &= \sum_p \boldsymbol{\omega}^p vol^p \delta(\boldsymbol{x} - \boldsymbol{x}^p(t)) \\ &= \sum_p \boldsymbol{\alpha}^p(t) \delta(\boldsymbol{x} - \boldsymbol{x}^p(t)) \end{aligned} \tag{2.36}$$

where $\boldsymbol{\omega}^p$ is the vorticity associated to the p-th particle of volume vol^p and $\boldsymbol{\alpha}^p = \int_{vol_p} \boldsymbol{\omega}^p dV$ is its vectorial circulation or *vectorial vortex strength*.

We may now substitute Equation 2.36 into Equation 2.35 while solving Poisson equation with the Green's function. Hence, recalling that the Green's function for $-\boldsymbol{\nabla}^2$ in unbounded domain is $G(\boldsymbol{x}) = 1/(4\pi|\boldsymbol{x}|)$, we will get

$$\begin{aligned} \boldsymbol{\psi}(\boldsymbol{x}, t) &= (G \star \boldsymbol{\omega})(\boldsymbol{x}, t) \\ &= \sum_p G(\boldsymbol{x} - \boldsymbol{x}^p(t)) \\ &= \frac{1}{4\pi} \sum_p \frac{\boldsymbol{\alpha}^p(t)}{|\boldsymbol{x} - \boldsymbol{x}^p(t)|} \end{aligned} \tag{2.37}$$

where $\star$ is the convolution operator. Then, the rotational part of the velocity field will be easily obtained as

$$\begin{aligned}\boldsymbol{u}(\boldsymbol{x},t) &= \boldsymbol{\nabla}\times\boldsymbol{\psi}(\boldsymbol{x},t)\\ &= \sum_p \left[\, \boldsymbol{\nabla}(G(\boldsymbol{x}-\boldsymbol{x}^p(t)) + G(\boldsymbol{x}-\boldsymbol{x}^p(t))\overbrace{\boldsymbol{\nabla}\times\boldsymbol{\alpha}^p(t)}^{=0} \,\right]\\ &= -\frac{1}{4\pi}\sum_p \frac{1}{|\boldsymbol{x}-\boldsymbol{x}^p(t)|^3}(\boldsymbol{x}-\boldsymbol{x}^p(t))\times\boldsymbol{\alpha}^p(t)\end{aligned} \tag{2.38}$$

Hence, we will get the following expression

$$\boldsymbol{u}(\boldsymbol{x},t) = \sum_p \boldsymbol{K}(\boldsymbol{x}-\boldsymbol{x}^p(t))\times\boldsymbol{\alpha}^p(t) \tag{2.39}$$

where the kernel $\boldsymbol{K}$ is the Biot-Savart kernel for a three-dimensional vortex

$$\boldsymbol{K}(\boldsymbol{x}-\boldsymbol{x}^p(t)) = -\frac{1}{4\pi}\frac{\boldsymbol{x}-\boldsymbol{x}^p(t)}{|\boldsymbol{x}-\boldsymbol{x}^p(t)|^3} \tag{2.40}$$

Evaluating Equation 2.39 directly per each particle in the field require a computational effort proportional to $\mathcal{O}(N^2)$, where N is the total number of particles.

Governing equations

Substituting Equation 2.36 into Equation 2.33, the governing equations for the evolution of vorticity-governed flow in a Lagrangian framework are obtained. The following formulation can be found in the literature under the name *Classical Scheme*:

$$\begin{cases}\dfrac{d}{dt}\,\boldsymbol{x}^p(t) = \boldsymbol{u}^p(\boldsymbol{x}^p(t),t)\\[2ex] \dfrac{d}{dt}\,\boldsymbol{\alpha}^p(t) = (\boldsymbol{\alpha}^p(t)\cdot\boldsymbol{\nabla})\,\boldsymbol{u}^p(\boldsymbol{x}^p(t),t)\end{cases} \tag{2.41}$$

In addition to that, there are two more numerical schemes that can be found in literature, each of them having different numerical properties. The code developed in this theses implemented the so called *Transport scheme*, which has been demonstrated to conserve the total vorticity. More specific mathematical aspects can be found in [61].
Moreover, the viscous diffusion $\nu\boldsymbol{\nabla}^2\boldsymbol{\omega}$ can be recovered by means of a proper numerical scheme (§ 2.3.5).

2.3.4 Regularized vortex particles

In this section, the regularized version of the method of vortex particles is considered. More specifically, the vorticity field is now represented as follows:

$$\boldsymbol{\omega}_\sigma(\boldsymbol{x},t) = \zeta_\sigma(\boldsymbol{x}) \star \boldsymbol{\omega}(\boldsymbol{x},t) = \sum_p \boldsymbol{\alpha}^p(t)\zeta_\sigma(\boldsymbol{x}-\boldsymbol{x}^p(t)) \tag{2.42}$$

where ζ_σ is an appropriate regularization function (i.e. an approximation to the δ-function) which is usually taken as radially symmetric while σ is a smoothing radius (i.e. a cutoff length core size) :

$$\zeta_\sigma(\boldsymbol{x}) = \frac{1}{\sigma^3}\zeta_\sigma\left(\frac{|\boldsymbol{x}|}{\sigma}\right) \tag{2.43}$$

An asymptotic behavior for $|\boldsymbol{x}|\to\infty$ that mimics the Biot-Savart law is granted through the following

$$4\pi\int_0^\infty \zeta(\rho)\rho^2 d\rho = 1 \tag{2.44}$$

Hence, the function ζ_σ defines the vorticity distribution within the core of the vortex particle. This means that a particle can be seen as a vectorial element built upon a position vector, an intensity vector and a cut-off length. By doing so,

the numerical scheme moves from singular particles (linked to the vorticity field through a δ-function) to regularized particles (linked through the ζ_σ-function), sometimes termed *blobs*.

Equation 2.39 can be easily extended to the present case

$$\boldsymbol{u}(\boldsymbol{x},t) = \sum_p \boldsymbol{K}_\sigma(\boldsymbol{x} - \boldsymbol{x}^p(t)) \times \boldsymbol{\alpha}^p(t) \tag{2.45}$$

Hence, the regularized form of the Biot-Savart kernel can be expressed as

$$\boldsymbol{K}_\sigma(\boldsymbol{x}) = -\frac{q_\sigma(\boldsymbol{x})}{|\boldsymbol{x}|^3}\boldsymbol{x} \tag{2.46}$$

where

$$q_\sigma(\boldsymbol{x}) = q\left(\frac{|\boldsymbol{x}|}{\sigma}\right) = q(\rho) = \int_0^\rho \zeta(t)t^2 dt \tag{2.47}$$

Moreover, the following holds:

$$\frac{q'(\rho)}{\rho^2} = \zeta(\rho) \tag{2.48}$$

In this way, the regularized kernel is equivalent to the Biot-Savart one at far distances with respect to the smoothing radius σ. On the other hand, as $\boldsymbol{x} \to 0$ the induced velocity tends to zero.

Desingularization

Vortex particles methods convergence has been first investigated by Beale [6]. As pointed out by Winklemans ([61], [60]) the solution converges to the three-dimensional vorticity equation for a finite time if the number of particles grows and the cut-off radius σ decreases (i.e. $\sigma/d > 1$, where d is a measure of neighbor particles mean distance).

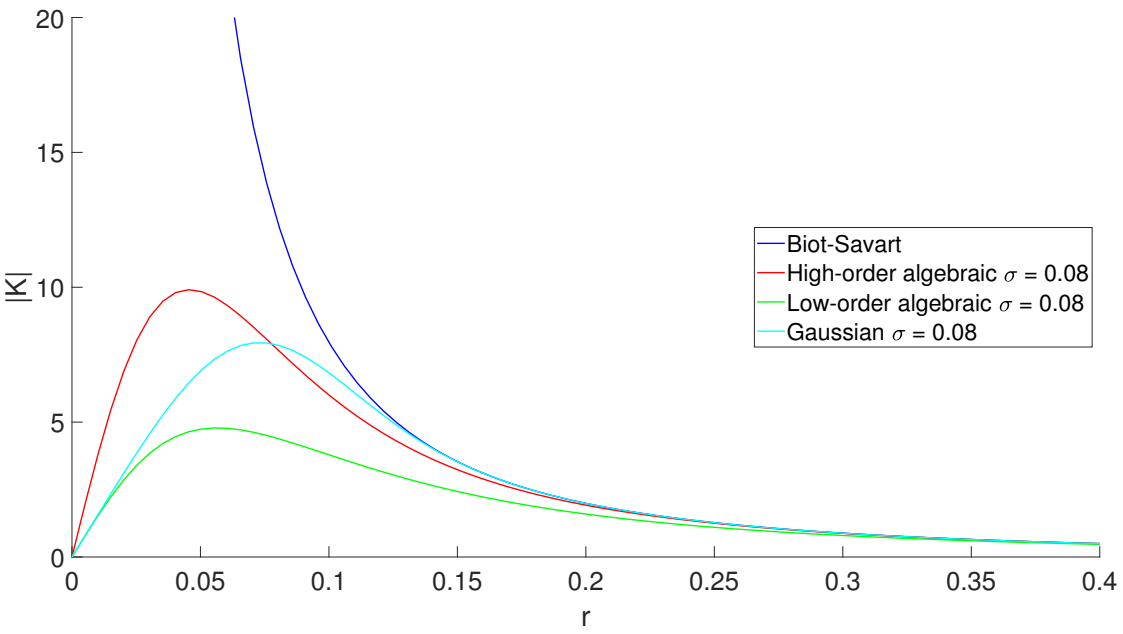

Figure 2.1: Comparison of the singular Biot-Savart kernel (2.40) and non-singular kernels with fixed smoothing radius

The error norm for the vorticity is composed of two terms: the first one is $\mathcal{O}(\sigma^r)$, while the second goes as $\mathcal{O}(\sigma(h/\sigma)^m)$. The exponent m is related to the number of derivatives of $\zeta(\rho)$. For the majorities of practical cases, m is large so that it is essential that cores do overlap for the second error terms to vanish. On the other hand, the exponent r is related to the moment properties of $\zeta(\rho)$: it must first satisfy the normalization constraint 2.44, together with

$$\begin{aligned} &\int_0^\infty \zeta(\rho)\rho^{2+s}d\rho = 0 \qquad 2 \leq s \leq r-1 \qquad \text{(s even)} \\ &\int_0^\infty |\zeta(\rho)|\rho^{2+r}d\rho < \infty \end{aligned} \tag{2.49}$$

This means that non-singlular particles and singular ones shares the very same moments: total vorticity, linear impulse and so on until $r-1$.

Among various alternatives, the most used and cited in literature are the following, [61] (3D case) :

- **Gaussian smoothing**

$$\zeta(\rho) = \frac{e^{-\rho^2/2}}{(2\pi)^{3/2}} \quad \rightarrow \quad q(\rho) = \frac{1}{4\pi}\left(\text{erf}\left(\frac{\rho}{2^{1/2}}\right) - \left(\frac{2}{\pi}\right)^{1/2} \rho e^{-\rho^2/2}\right) \quad (2.50)$$

which corresponds to $m = \infty$ and $r = 2$

- **Low order algebraic smoothing**

$$\zeta(\rho) = \frac{3}{4\pi}\frac{1}{(1+\rho^2)^{5/2}} \quad \rightarrow \quad \boldsymbol{K}_\sigma(\boldsymbol{x},\boldsymbol{y}) = -\frac{1}{4\pi}\frac{\boldsymbol{x}-\boldsymbol{y}}{(|\boldsymbol{x}-\boldsymbol{y}|^2+\sigma^2)^{3/2}} \quad (2.51)$$

which corresponds to $m = \infty$ and $r = 0$

- **High order algebraic smoothing**

$$\zeta(\rho) = \frac{15}{8\pi}\frac{1}{(1+\rho^2)^{7/2}} \quad \rightarrow \quad \boldsymbol{K}_\sigma(\boldsymbol{x},\boldsymbol{y}) = -\frac{1}{4\pi}\frac{|\boldsymbol{x}-\boldsymbol{y}|^2+\frac{5}{2}\sigma^2}{(|\boldsymbol{x}-\boldsymbol{y}|^2+\sigma^2)^{5/2}} \quad (2.52)$$

which corresponds to $m = \infty$ and $r = 2$

In order to obtain an expression for the induced velocity for each smoothing function it is necessary to exploit Equation 2.46 through the function $q(\rho)$ provided by the relation 2.48.
In Figure 2.1 the induced velocity provided by those kernles defined above is plotted and compared with the singular evolution of the Biot-Savart kernel.

The low-order algebraic kernel, (proposed by Rosenhead in the context of vortex filaments) is believed to provide convergence in all cases practical interest. However, it has been warned that the kernel does not satisfy the second inequality reported in 2.49 required in the classical proof of convergence [61]. Moreover, even if it does lead to convergence, the convergence rate is expected to be lower than that of the transcendental kernel based on the Gaussian core function (the kernel

is constructed form a low-order core function). To address these issues , Winklemans and Leonard [60] introduced the high-order algebraic kernel specified above (Equation 2.52).

First of all, this kernel satisfies all the requirements in the classical convergence analysis of vortex methods: moreover, it is expected to be more numerically convenient than those kernels involving transcendental functions.
Second, it has convergence properties equal to those of the Gaussian smoothing and as for vortex particles this smoothing is a case for which closed form expressions for all quadratic diagnostics can be obtained (kinetic energy, elicity and enstrophy, [61]). Other than that, in order to implement a fast induction algorithm, an explicit recurrence relation suitable to compute Taylor coefficients of the kernel is required (ref Chapter 3), which is also available for the high-order kernel (as it is for the low-order one).
Hence, bot algebraic kernels are implemented in the code. A convergence comparison based on numerical analysis of these two kernel is reported in [59].

2.3.5 Viscous diffusion modeling

Despite the weaknesses of a vortex particle scheme (technical aspects can be found in [61]), a key point that that draws the attention on this kind of methods is the ability to model the viscous diffusion term ($\nu\boldsymbol{\nabla}^2\boldsymbol{\omega}$). Neglecting the diffusion effect in specific applications could lead to unreliable results.
In this way, when using vortical edges methods which do not model diffusion at all, one must carry on a careful tuning of empirical viscus coefficients, which are directly related to the induction stability when vortical elements approach each other. Furthermore, in the three-dimensional case the energy transfer from higher to lower scales could lead to complex arrangements of vortical lines[2], increasing

[2]In the context of the vorticity form of the Navier-Stokes equations this effect is directly linked to the *vortex stretching* term, present in the three-dimensional case only ($(\boldsymbol{\omega}\cdot\boldsymbol{\nabla})\boldsymbol{u}$)

complexity and possibly leading to instabilities.

Within this frame of reference, the importance of the Laplacian term arises, since this is the one that smooths out and redistribute particles vorticity strengths, resulting in a energy dissipating effect.
In the leftover part of this section, two method for modeling the Laplacian term are reported. The second one is actually implemented in the code.

Viscous splitting methods

This class of methods inherits the idea to split the viscous contribution from the original equations (2.53): in the first phase the non-viscous equations are solved, i.e. the particle are convected while the vorticity evolves according to the vortex stretching term. Then, the vorticity is corrected with the Laplacian term, adequately approximated:

$$\begin{cases} \dfrac{D\boldsymbol{\omega}}{Dt} = (\boldsymbol{\omega}\cdot\boldsymbol{\nabla})\boldsymbol{u} & ① \\ \dfrac{\partial\boldsymbol{\omega}}{\partial t} = \nu\boldsymbol{\nabla}^2\boldsymbol{\omega} & ② \end{cases} \tag{2.53}$$

There are many ways one can represent the second equation in 2.53. One of the most famous is the so called *Random walk method* proposed by Chorin in 1973 [12]. In this context, particles are subjected to a Brownian-like motion in order to mimic a viscous effect. The inviscid equation is solved and then the particle positions are evolved in time according to the following

$$\frac{d}{dt}\boldsymbol{x}^p(t) = \boldsymbol{u}_\sigma(\boldsymbol{x}^p(t), t) + \boldsymbol{W}^p \tag{2.54}$$

where $\boldsymbol{W}^p$ models a Wiener process with prescribed parameters.

A modern class of methods that opened the way to PSE schemes are the *Resampling* methods, developed by Raviart [51]. Within this framework, the viscous contribution modifies the particle strength. The diffusion equation is solved though

a diffusion kernel built with intial conditions given by particles strength at time t_n.

Particle strength exchange scheme (PSE)

Soon after, starting from the idea of redistributing particles intensities rather than position, a class of new methods based on the integral approximation of the diffusion term, rather than the vortex intensity redistribution has been developed. This is the scheme which is implemented in the code. In 1983 Degond and Mas-Gallic [14] showed that the Laplacian form $\nu\boldsymbol{\nabla}^2\boldsymbol{\omega}$ can be approximated as an integral operator, which is more suitable for a particle method:

$$\nu\boldsymbol{\nabla}^2\boldsymbol{\omega} \approx \frac{2\nu}{\sigma^2}\int(\boldsymbol{\omega}(\boldsymbol{y}) - \boldsymbol{\omega}(\boldsymbol{x}))\ \eta_\sigma(\boldsymbol{x}-\boldsymbol{y})d\boldsymbol{y} \tag{2.55}$$

with ζ a smoothing function of radial symmetry, σ the cut-off radius and

$$\eta_\sigma(\boldsymbol{x}) = \frac{1}{\sigma^3}\left(\frac{|\boldsymbol{x}|}{\sigma}\right) \qquad \eta(\rho) = \frac{\zeta'(\rho)}{\rho} \tag{2.56}$$

Discretizing space into particles of volume vol_p and position $\boldsymbol{x}^p$, the integral in Equation 2.55 can be approximated by means of a quadrature formula. Then, evaluating the viscous diffusion term at a particle position $\boldsymbol{x}^p$ we will get the following:

$$\nu\boldsymbol{\nabla}^2\boldsymbol{\omega}(\boldsymbol{x}^p) \approx \frac{2\nu}{\sigma^2}\sum_q(vol_q\boldsymbol{\omega}_q - vol_q\boldsymbol{\omega}_p)\ \eta_\sigma(\boldsymbol{x}^p - \boldsymbol{x}^q) \tag{2.57}$$

Therefore, multiplying Equation 2.57 by vol_p and applying the vortex particle discretization , viscous diffusion effects are then approximated as

$$\nu\boldsymbol{\nabla}^2\boldsymbol{\omega}(\boldsymbol{x}^p) \approx \frac{2\nu}{\sigma^2}\sum_q(vol_p\boldsymbol{\alpha}_q - vol_q\boldsymbol{\alpha}_p)\ \eta_\sigma(\boldsymbol{x}^p - \boldsymbol{x}^q) \tag{2.58}$$

Winckelmans provides some examples of kernel functions with spherical symmetry, some of them are listed in the following:

$$\begin{aligned}
\zeta(\rho) &= \frac{e^{-\rho^2/2}}{(2\pi)^{3/2}} \quad \rightarrow \quad \eta(\rho) = \frac{e^{-\rho^2/2}}{(2\pi)^{3/2}} && \text{(Gaussian)} \\
\zeta(\rho) &= \frac{3}{4\pi}\frac{1}{(1+\rho^2)^{5/2}} \quad \rightarrow \quad \eta(\rho) = \frac{15}{4\pi}\frac{1}{(1+\rho^2)^{7/2}} && \text{(Low-order algebraic)} \\
\zeta(\rho) &= \frac{15}{8\pi}\frac{1}{(1+\rho^2)^{7/2}} \quad \rightarrow \quad \eta(\rho) = \frac{105}{8\pi}\frac{1}{(1+\rho^2)^{9/2}} && \text{(High-order algebraic)}
\end{aligned} \tag{2.59}$$

2.3.6 Particles injection

A critical part of the whole method is the injection of particles in the field, starting from a releasing object of arbitrary characteristics. There are many contributions on this topic, depending on the application, but the main constraint is the conservation of the vorticity momentum of order zero and one:

$$\begin{aligned}
\boldsymbol{\alpha}^p(t) &= \int_V \boldsymbol{\omega}(\boldsymbol{x},t)\, dV \\
\boldsymbol{\alpha}^p(t) \times \boldsymbol{x}^p(t) &= \int_V \boldsymbol{\omega}(\boldsymbol{x},t) \times \boldsymbol{x}^p\, dV
\end{aligned} \tag{2.60}$$

If the particles are released from a panel interface this means that one must gather the vorticity distributed on vortical edges into discrete point. For instance Hu (2015) [29] places integration volumes in the middle of longitudinal segments perpendicular to the releasing wing, while S.G.Voutsinas (2005) [58] puts new particles onto the vortex lattice nodes.

A very interesting application can be found in [47]: the authors developed a vorticity release strategy at solid surface of three-dimensional bodies. A number of nascent vortices are introduced in the flow field according to the diffusion and the convection of the vorticity near the solid surface. In this way, they attempted to simulate unsteady flow around three dimensional bluff bodies with a vortex particle method.

Chapter 3

Fast methods for particle simulations

As stated in the previous chapter, evaluating the sums reported in Equation 2.39 by direct summation could be very resource demanding, especially when an aerodynamic code has to be complemented with a structural solver.
As the number of particles grows, performing a complete simulation could be unfeasible, specifically form an industrial point of view, since the time required to perform these operation is excessively large. As a matter of fact, the code described in the present thesis is supposed to be exploited especially during design phases, in which a substantial number of runs must be executed.

This problem, termed the *N-body problem*, is well-known across all fields of research, where the computation of pairwise interactions is required (for instance, in celestial motion, electrically charged particles and molecular dynamics). Generally speaking, its computational complexity has an asymptotic behavior, requiring $\mathcal{O}(N^2)$ operations for each time step, where N is the number of particles.
As a matter of fact, the computation of velocity at an arbitrary position is carried out by summing all particles contribution, and the velocity at each particle is required to advance in time the simulation.

During the last decades, in order to address time (but also accuracy) require-

ments, alternative approaches to the direct interactions between well-separated particles have been proposed.

3.1 Treecodes algorithms

The first viable fast algorithms for N body problems combined two main ideas:

- Approximating the effect (in its general meaning) of a group of distant objects, termed *clusters* (charges, masses, blobs) by their first moments.
- Organizing the space in a hierarchical fashion to get a measure of distance for these approximations. In other words, a nested subdivision of space is used to construct the object clusters.

In our context ,the objects are particles and the interaction corresponds to the induction phase. In this way, a particle-cluster interaction is used to rapidly compute the influence of a particle cluster on a single target particle, reducing the amount of computational effort. This is accomplished by approximating the cumulative effect of the particles in the cluster on the target particle with a simplified expression. The latter can be evaluated for multiple target particles with an operation count independent on the number of particles within the cluster.
Moreover, the particle-cluster interactions are only performed for particles and clusters which are separated from each other, i.e. it is performed if the distance from the particle is such that $\mathcal{D}/\mathcal{L} > \epsilon$, where $\mathcal{D}$ in the cluster average dimension, $\mathcal{L}$ is the distance and ϵ is a fixed accuracy parameter.

Thus, this approximation is termed *far-field approximation.*
Moreover, the nested subdivision, which is used to build the clusters, is the tool that generates particle-cluster interactions in which the particle is far from the cluster, relative to the cluster's size. Two early examples of such algorithms are due to Appel [2] and Barnes and Hut [4], who implemented this approach for astrophysics simulations.

The combination of all these ingredients lead to an algorithm whose asymptotic operation count is $\mathcal{O}(N \log N)$.

In the past decades, many alternatives were exploited: the one presented by Lindsay is worth being reported [38]. Lindsay's treecode was used to simulate the roll-up of a circular-disk vortex sheet into a vortex ring. For each particle-cluster interaction, the order of approximation is chosen adaptability, and a run-time choice is made between Taylor approximation and direct summation based on empirical estimates of the necessary CPU time. If the approximation is slower with respect to the direct summation and the current cell is not at the bottom of the tree, then the code descends to the next layer of the tree, recursively calling the compute-velocity routine. The rationale for descending the tree is that the children cells have smaller radii and fewer particles, so it is more likely that the accuracy criterion will be satisfied.

Kernel expansion

Lindsay's results related to the approximation for a particle-cluster interaction are summarized here but they are key for the *fast multipole method* too.

Once the tree is build, the velocity induced at point $\boldsymbol{x}_0$ by the vorticity within a cluster c can be evaluated as

$$\boldsymbol{u}(\boldsymbol{x}_0) = \sum_{j \in c} \boldsymbol{K}_\delta(\boldsymbol{x}_0, \boldsymbol{y}_j) \times \boldsymbol{\alpha}_j \tag{3.1}$$

where $\boldsymbol{y}_j$ are particles inside that cluster.

Clearly, summing over all clusters and using this expression directly results in a computational cost that is $\mathcal{O}(N^2)$. The cost of the computation can be reduced to $\mathcal{O}(N \log N)$ by expanding $\boldsymbol{K}_\delta(\boldsymbol{x}_0, \boldsymbol{y}_j)$ in a Taylor series in the second argument about the center of the source cluster $\boldsymbol{y}_c$.

By using multi-index notation[1]:

$$\begin{aligned}\sum_{j\,\in\,c} \boldsymbol{K}_\delta(\boldsymbol{x}_0, \boldsymbol{y}_j) \times \boldsymbol{\alpha}_j &= \sum_{j\,\in\,c} \boldsymbol{K}_\delta(\boldsymbol{x}_0, \boldsymbol{y}_c + (\boldsymbol{y}_j - \boldsymbol{y}_c)) \;\times\; \boldsymbol{\alpha}_j \\ &= \sum_{\boldsymbol{k}}^{\infty} \frac{1}{\boldsymbol{k}!} D_y^{\boldsymbol{k}} \boldsymbol{K}_\delta(\boldsymbol{x}_0, \boldsymbol{y}_c) \;\times\; \left(\sum_{j\,\in\,c} (\boldsymbol{y}_j - \boldsymbol{y}_c)^{\boldsymbol{k}} \boldsymbol{\alpha}_j \right) \qquad (3.2) \\ &= \sum_{\boldsymbol{k}}^{\infty} \boldsymbol{a}_{\boldsymbol{k}}(\boldsymbol{x}_0, \boldsymbol{y}_c) \;\times\; \boldsymbol{m}_{\boldsymbol{k}}(c)\end{aligned}$$

where

$$\boldsymbol{m}_{\boldsymbol{k}}(c) = \sum_{j\,\in\,c} (\boldsymbol{y}_j - \boldsymbol{y}_c)^{\boldsymbol{k}} \boldsymbol{\alpha}_j \qquad (3.3)$$

The $\boldsymbol{a}_{\boldsymbol{k}}(\boldsymbol{x}_0, \boldsymbol{y}_c)$ are the Taylor coefficients of the non-singular kernel with respect to $\boldsymbol{y}$ about $\boldsymbol{y} = \boldsymbol{y}_c$, and they are independent of the particles $\boldsymbol{y}_j$ in the cluster c.

On the other hand, the $\boldsymbol{m}_{\boldsymbol{k}}(c)$ describe the distribution of particles in cluster c and so they are termed *particle moments*; moreover, they do not depend on the target point $\boldsymbol{x}_0$. Thus, once the particle moments are computed for one particle-cluster interaction, they can be stored and used for subsequent particle-cluster interactions with different target particles.

Clearly, the coefficients must be approximated by truncating the infinite series in Equation 3.2, yielding:

[1] $\boldsymbol{k} = (k_1, k_2, k_3)$ is the third order multi-index, where the subscripts 1-3 refer to the Cartesian directions. Moreover, when $k_i > 0$ the following will hold:

- $\boldsymbol{k!} = k_1! k_2! k_3!$
- $\boldsymbol{x}^{\boldsymbol{k}} = x_1^{k_1} x_2^{k_2} x_3^{k_3}$
- $D_y^{\boldsymbol{k}} = \partial / \partial y_1^{k_1} \partial / \partial y_2^{k_2} \partial / \partial y_3^{k_3}$
- $|\boldsymbol{k}| = k_1 + k_2 + k_3$

$$\boldsymbol{u}(\boldsymbol{x}_0) \sim \sum_{|\boldsymbol{k}|<p} \boldsymbol{a}_{\boldsymbol{k}}(\boldsymbol{x}_0, \boldsymbol{y}_c) \ \times \ \boldsymbol{m}_{\boldsymbol{k}}(c) \tag{3.4}$$

This truncation is referred to as a pth *order expansion* for a particle cluster interaction. The error analysis can be found in [38].

Recurrence relations

For the algorithm to be computationally efficient, it is mandatory to rapidly compute the $\boldsymbol{a}_{\boldsymbol{k}}(\boldsymbol{x}_0, \boldsymbol{y}_c)$ defined in Equation 3.2. To do so, Lindsay suggests a method for doing this, based on a polynomial representation of the Plummer potential and the Rosenhead-Moore kernel, [38]. As he pointed out at the end of his paper, the whole method can be applied to a variety of other kernels; the main prerequisite is that the Taylor coefficients of the kernel should satisfy a recurrence relation: as a matter of fact he applied this approach to problems involving screened electrostatics and in general power-law interactions in molecular dynamics.

It is worth noticing further recent developments proposed by Wee et al [59], in which convergence characteristics of the Rosenhead-Moore kernel are questioned, presenting a recurrence relation for the Taylor coefficients of an higher order kernel, the Winckelmans-Leonard one (Equation 2.52).

In this way, both kernel are implemented in our code, leaving the choice to the user. To get an insight on the relations and to have further technicalities on their convergence studies , one my refer to their paper.

3.2 The Fast Multipole Method (FMM)

An even more efficient implementation of the concept presented in the previous section is the *Fast Multipole Method* (FMM) introduced for the first time by L. Greengard and V. Rokhlin (1987) [23].

The critical new idea introduced here is the *local expansion*. This mathematical tool allows groups of distant objects to interact with groups of *targets*.

Selected as one of the top ten algorithms of the XX century, FMM reduces the original N-body problem $\mathcal{O}(N^2)$ to a linear problem $\mathcal{O}(N)$, when N is high enough. Moreover, the error can be controlled through an appropriate choice of the order of expansion. Indeed, one key difference between treecodes and FMM is the method of achieving a desired accuracy in the approximations. While treecodes ensure an arbitrary accuracy by restricting the acceptable distances for group-to-target interactions, FMM acts on the series expansions and chooses a proper truncation for specified accuracy.
As a matter of fact, FMM approximates the induction by far-field point sources through an expansion performed with spherical harmonic functions, the so called *multipole expansion*. In this way, particle-cluster interactions are approximated to any specified tolerance.
As outlined above, they also introduced cluster-cluster interactions by expanding the far-field approximation into a local near-field expansion for rapid evaluation at multiple target points.

All of these expansions and approximations are appropriate when the interaction kernel is harmonic, such as the Newtonian potential of gravitational and electrostatic interactions, but they are unsuitable for non-harmonic kernels, such as the Rosenhead-Moore and the Winckelmans-Leonard ones. This is because the method relies on the harmonicity of the kernel to ensure convergence.

An expansion using Cartesian Taylor series can be used to overcome this constraint: this idea was first proposed by Zhao[63], who used Taylor series for computations with the Newtonian potential in three dimensions. The main purpose

was to generalize the original two-dimensional complex series expansion to the three-dimensional setting.

In this context, a Cartesian multipole expansion method is implemented: the steps required to built the method are the same one would use for the classical implementation exploiting harmonic expansions; the only difference is related to the operator kernels expressions. We will report those needed for a Cartesian implementation.

3.2.1 Hierarchical subdivisions

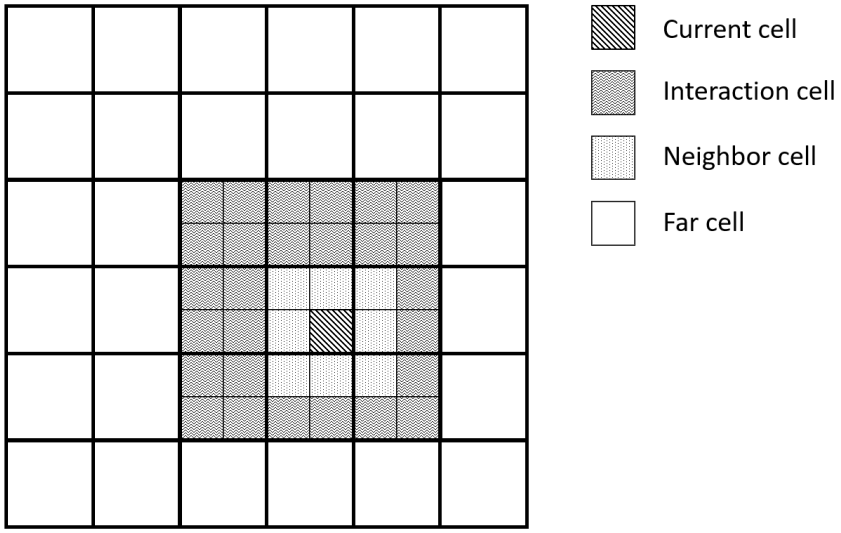

Figure 3.1: Nomenclature convention of the cells for cell C at level l; 2D case, 2 layers

Before going into more detail, a brief presentation on the data structure needed to exploit the method will be given. As sketched above, FMM groups objects hierarchically to apply expansions: in this way, it is necessary to subdivide the computational domain in smaller and smaller regions, like a treecode structure.

Therefore, at the highest level, at the *layer* 0, the whole computational domain is enclosed. Then, successive layers are added in a recursive manner subdividing

the volume into 8 cells or boxes, which are constrained to have the same volume. The layer *l+1* will hold the so called *children* of all the cells in the upper layer l.

Moreover, objects will be stored in the lowest level in the corresponding cell, where the object geometrically belongs to. This are the basics of very interesting data structure that is described in detail in Chapter 4.

In the following, some definitions are given. In order to ease the visualization, a 2D structure is presented in Figure 3.1. Of course, the following definitions are valid also for the 3D case.

- A cell having no child cells that holds at least one object is termed *leaf*. In this context, an object is considered to be within a cell if the center of the element is inside that cell. If the structure is *full* (i.e. every internal cell has eight children and the structure is built until a predefined level) then leaves can only exists at the lower level.
- Two cells placed at level l are said to be *adjacent cells at level l* or *neighbors*, if they have at least one common vertex.
- Two cells are said to be *well separated at level l* if they are not adjacent at level l but their parent cells are adjacent at level l - 1.
  The list of all the well-separated cells at a level l from a cells C is called the *interaction list* of C.
- Given a cell, if its parent cell is not a neighbor to the parent of cell C, then this cell is termed *far cell* of C.

3.2.2 FMM Algorithm

In this section, some results of the mathematical treatment are summarized, for more details and technical aspects one may refer to Liu [39] and Brown[7] while in [17] a practical Cartesian FMM implementation is available: their multipole transnational operator are reported for convenience. Here, the terms *cell*, *cluster* and *node* may be used interchangeably.

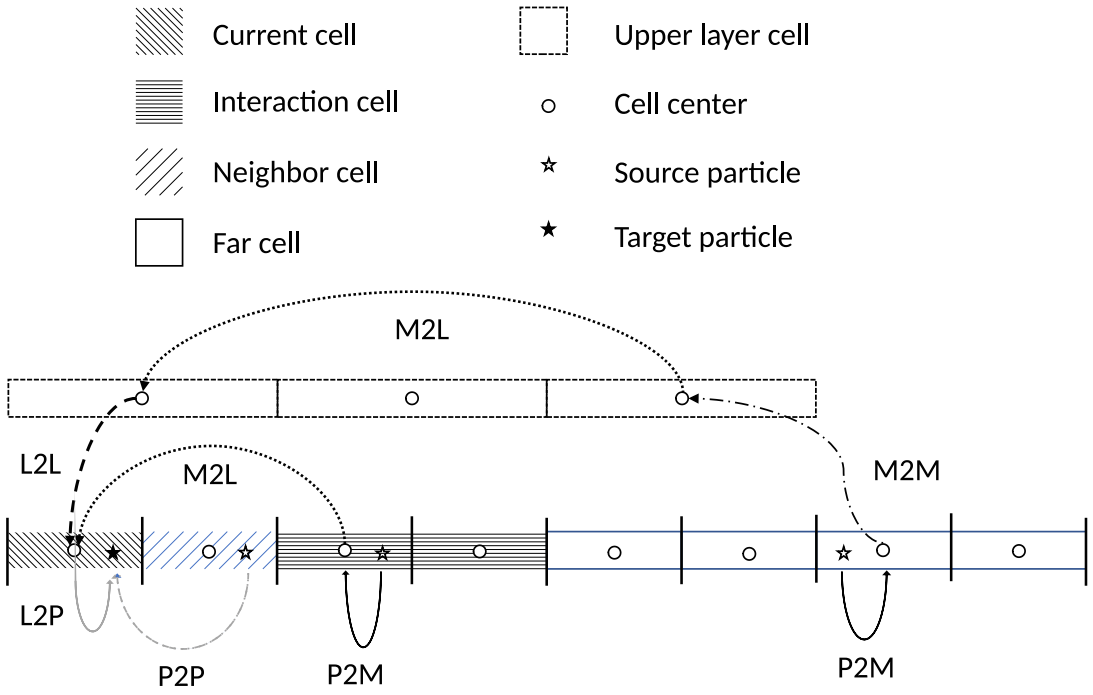

Figure 3.2: FMM algorithm 1D visual representation

The results on the Taylor expansions summarized in the previous section will be still used here, starting from Equation 3.4.

To achieve the $\mathcal{O}(N)$ cost of the FMM calculation the so called *local multipole coefficients* ($\boldsymbol{\mathcal{L}}$) must be computed for each cluster. This is done by differentiating Equation 3.4 with respect to $\boldsymbol{x}_0$ (i.e. by calculating the spatial derivatives of $\boldsymbol{u}$ at $\boldsymbol{x}_0$), which results in

$$\boldsymbol{\mathcal{L}}_k(\boldsymbol{x}_0) = (-1)^{\boldsymbol{k}} \sum_{|\boldsymbol{n}|\leq \boldsymbol{k}} \frac{\boldsymbol{k}! + \boldsymbol{n}!}{\boldsymbol{k}!\boldsymbol{n}!} \quad \boldsymbol{a}_{\boldsymbol{k}+\boldsymbol{n}}(\boldsymbol{x}_0, \boldsymbol{y}_c) \times \boldsymbol{m}_{\boldsymbol{n}}(c) \tag{3.5}$$

Moreover, the velocities derivatives computed above can be shifted to some other nearby point $\boldsymbol{x}$ by expanding Equation 3.5 as a Taylor series expansion about $\boldsymbol{x}_0$ so that

$$\boldsymbol{\mathcal{L}}_k(\boldsymbol{x}) = \sum_{|\boldsymbol{n}|\leq \boldsymbol{k}} \binom{\boldsymbol{n}}{\boldsymbol{k}} (\boldsymbol{x} - \boldsymbol{x}_0)^{\boldsymbol{n}-\boldsymbol{k}} \ \boldsymbol{\mathcal{L}}_{\boldsymbol{n}}(\boldsymbol{x}_0) \tag{3.6}$$

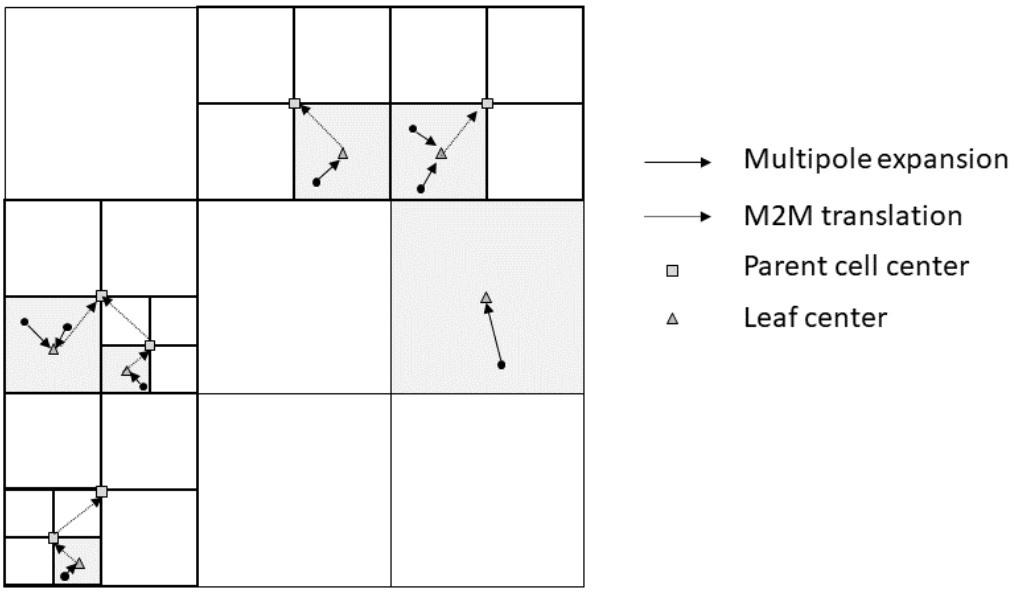

Figure 3.3: FMM upward pass step of the algorithm

3.2.3 Coefficients computation

In Figure 3.2 the main operations involved in the FMM algorithm are outlined for a 1D case, which is easier to visualize, but this diagram extends easily to three dimensions. In this section, each step will be described in details.

Initialization

First, the octree must be created, or, if already present, it must be cleaned from previous computations. It can be built as a full octree (in that case each cell would have 8 children) until the maximum layer is reached. If so, the particles could then be added at the lowest layer. The second option is to fix the maximum number of particles per cell. In this case, the tree would be built recursively, automatically refining those volumes where the particle density is higher.

Upward pass

This stage is further subdivided into two steps. In Figure 3.3 a 2D example is represented.

- First, moments on all leaves must be computed according to Equation 3.3. In other words, the vorticity of the particles is aggregated into the multipole expansions at the center of all cells (P2M operation).

- Then, the rest of this phase involves an upward sweep to the octree: multipole expansions are further clustered by translating the center of each expansion to a larger cell and adding their contributions at that level (M2M operation), according to the following relation:

$$\boldsymbol{m_k}(c) = \sum_{\hat{c} \in c} \sum_{|\boldsymbol{n}| \leq \boldsymbol{k}} \binom{\boldsymbol{k}}{\boldsymbol{n}} (\boldsymbol{x}_{\hat{c}} - \boldsymbol{x}_c)^{\boldsymbol{k}-\boldsymbol{n}} \ \boldsymbol{m_n}(\hat{c}) \tag{3.7}$$

  where c refers to the parent cluster, while with the symbol $\hat{\cdot}$ we denote child clusters. In this way, $\boldsymbol{x}_{\hat{c}}$ denotes the child cluster's center and $\boldsymbol{x}_c$ denotes the parent's center. Moreover, the symbol $\in$ when applied to cluster labels, discriminates if a cell is a child of the parent or not.

Downward pass

The downward pass involves a sweep back down the tree. This sweep will generate and refine the velocity filed on each level of the tree. In this stage, the *local expansion coefficients* on all cells will be computed, starting from the second level, tracing the tree structure downward to all the leaves.
On each layer l, the local expansion associated with cell c is the sum from two contributions:

- The first one comes from the interaction list of cell c and it is termed *middle-field component*. This contribution is evaluated through the so called M2L

translation, applying Equation 3.5 to each of the clusters, located at layer l, within cell c's interaction list.

- The second one is the so called *far field contribution*: on each layer l the far field component is *inherited* from cell c's parent cluster, located at the upper layer. This step is known as L2L translation from the parent center. This operations consists on shifting the expansion point from the parent center to that of c, by applying Equation 3.6.

Both contributions are summed together and saved in the cell c, then they are translated to cell c's children on the subsequent level l - 1 of the octree, except if cell c is on the lowest level of the octree. This process of inheritance and expansion evaluation is performed on all clusters on a given level of the octree before descending into the next one and repeating the entire process. To speed up calculations, all computations are performed only on those cells whose children nodes contains particles.

Velocity evaluation on particles

Once the downward sweep is completed, the velocity on each particle p inside cell c is evaluated adding two contributions:

- The first contribution is build upon the local expansion coefficients just computed, using the following relation (L2P procedure):

$$\boldsymbol{u}_l(\boldsymbol{x}_0) = \sum_{|\boldsymbol{k}|=0}^{p} (\boldsymbol{x}_0 - \boldsymbol{x}_c)^{\boldsymbol{k}} \boldsymbol{\mathcal{L}}_c \tag{3.8}$$

- The second contribution on p located in $\boldsymbol{x}_0$ comes from the direct induction by all those particles inside cell c and all those particles located in the cell c's neighbor clusters (P2P procedure).

$$\boldsymbol{u}_d(\boldsymbol{x}_0) = \sum_{j} \boldsymbol{K}_\delta(\boldsymbol{x}_0, \boldsymbol{y}_j) \times \boldsymbol{\alpha}_j \tag{3.9}$$

Chapter 4

Hierarchical data structures

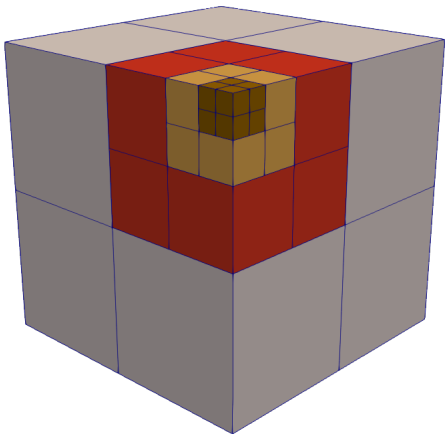

Figure 4.1: A simple four layers octree visual representation

In the previous chapter, the need of a hierarchical data structure suitable to implement the fast multipole method has been stressed.
In this chapter, a brief literature review of these kind of data types (termed *Octree* or *Quadtree*) will be given, alongside possible creation, storage and manipulation strategies [53]. Then, the choices made for our application will be pointed out.

4.1 Preliminaries

An octree is a recursive, spacial isotropic partitioning data structure commonly used in computer graphics to optimize collision detection, neighbor search and frustum culling (a selection process meant to prevent objects not visible on the screen to be rendered real-time).
An octree hierarchically subdivides a finite 3D volume into eight octants, termed *nodes* in the context of data structure or *cells* in the context of geometry (Figure 4.1). The starting volume (the so called *root*) encloses the entire domain of interest. Generally speaking there is no need to force the root cell size to have the shape of a cube, nor to subdivide it into cubic cells. However, following the latter approach, one would slightly reduce memory consumption, since the cell size can be stored as just one value per node instead of three.
Following the same reasoning, a quadtree is the octree equivalent in two dimensional space: in this case it is meant to subdivide a two-dimensional space in four quadrants.

For now on, we will focus on octrees only, since are the ones needed to represent a 3D physical space.

An octree is built by recursively subdividing space into eight cells. Every cell is subdivided by three axis-aligned planes, which are placed in the middle of the parent node. Thus, each node can have up to eight children. Generally speaking every cell must have some sort of dictionary to save objects (in an abstract sense: they could be points or any geometric shape).
All those cells in the last layer which contains an object are called *leaves* of the tree. In our case, they will hold all the particles until the next simulation step. The octree creation strategy depends on the specific purpose the octree is designed for. Generally, there are two neat strategies:

- *Maximum items per cell*: The space is recursively subdivided until the remaining number of objects in each cell is equal or below an assigned threshold.

- *Maximum three dept*: The tree is build until a pre-defined level is reached. In this case the tree is called *full octree*.

  In a full or complete octree, every internal node has eight children and all leaf nodes have the very same tree depth D. Thus, it is equivalent to a regular 3D grid given by $2^D \times 2^D \times 2^D$. The total number of leaf nodes is 8^D, whereas the total number of nodes can be easily calculated as

$$N_{nodes} = \sum_{i=o}^{D} 8^i = \frac{8^{D+1} - 1}{7} \tag{4.1}$$

  Thanks to the regularity of this object, one could store only a pointer to the object inside every cell. Clearly, if most of the nodes are empty, memory savings due to the small node structure are lost by the huge amount of nodes that are to be allocated. Hence, the memory required to store the octree raises exponentially. For instance, a full tree with 10 layers consists of more than 1.2 billion nodes.

The octree implemented for GyroX can be built with both strategies, since it is planned to exploit it for other application beyond particle inductions, for instance *collision detection* calculations.

4.1.1 Node data allocation

Each node must be provided with a structure to access data associated with it whenever needed. As for the implementation presented in this work, each node is linked with a dictionary structure that can host different data types both primitives or user defined, Figure 4.2. Moreover, activity labels are used to assess if a specific node contains a specific data type.

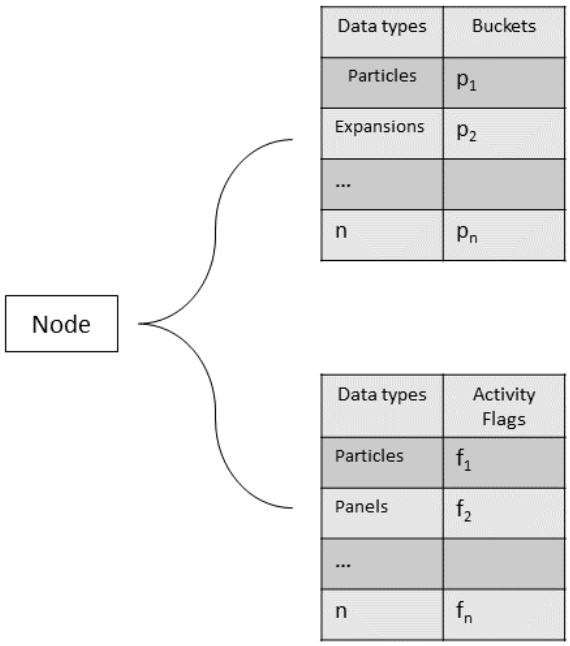

Figure 4.2: Node data structures architecture

4.2 Pointer-based node representations

A pointer-based node representation has been one of the early approaches in octree generation, [16]. In the following, three major alternatives are reported. In all cases, every node can store a pointer to the parent node to accelerate bottom-up traversals.

Parent-child representation

Generally speaking the simplest implementation consists of eight pointers to each of the eight child nodes. The main advantage is the ability to support nodes allocation when strictly needed only.

Block representation

A second option is to store a single pointer to a block of eigh children, instead of eight pointers. Even if a considerable amount of memory could be saved, on de-

mand child allocation is not possible anymore, all eight children must be allocated.

Sibling-child representation

In this last scenario, each node stores a pointer to its first child and to the next child of its parent. This option uses on average a fourth of the memory compared to the parent-child representation, but requires more pointer dereferencing to access a given node, since direct access is lost.

4.3 Pointer-less representations

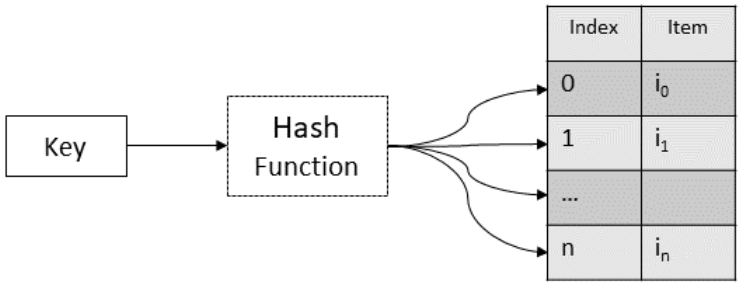

Figure 4.3: Hash map data structure: each key is mapped to an index by means of a function. Each index corresponds to a pointer stored in bucket

A more compact representation has been proposed for the first time by Gargantini [20], who originally developed it for quadtrees.

Linear or *hashed* trees combine the advantage of pointer based and pointer-less representation, replacing pointers by indexes manipulation.

In the general scenario, nodes are stored in a hash table or dictionary-like structure, which allow direct access to *any* node or any whole layer of nodes. Following this approach, each node is assigned a *key*, then a suitable hash function is used to identify the node inside the table. This key may represent the position of the node in the hierachical structure or the geometry of the associated cell in

order to correctly identify the cell itself. The hash function must be chosen with care, since it must satisfy two basic properties:

- It should be fast to compute
- It should strongly minimize duplication of output addresses (named *collisions*)

A hash table potentially provides $\mathcal{O}(1)$ access time, removal, insertion and search to any of its elements, but this is guaranteed as long as there are no collisions in the hash table (termed *perfect hashing*).
For further details on this topic one may check [34].

In efficient schemes, the key cumulates both significations: it is used to identify the node but it is also interpreted as an address in the hash table. This double aim is very advantageous: as a matter of fact, it allows at the same time to identify the children of a node by the octant orientation for traversal algorithm and to directly access a node, with no recursions at all, for fast search procedures.

There are several efficient strategies for generating such keys [57], the most useful and cited being Morton codes [42].
As a matter of fact, Morton encoding is equivalent to linearly index
multi-dimensional data, preserving at the same time *data locality* upon multiple levels, which is key to superior memory performance. In other words, morton encoding is a mapping from a multidimensional space to one dimension. In this way neighbor search would be faster since memory addresses are closer in memory. The way this is accomplished is by storing data using space-filling curves, known also as Z-order curves [57]. The state-of-the-art in linear octree generation is to use a hash table with Morton codes.
An interesting development has been given by Hasbestan et Al., [25] in the field of adaptive mesh refinement. They built a tree storing leaves in a special kind of binary tree in place of the hash table.

4.3.1 Morton encoding

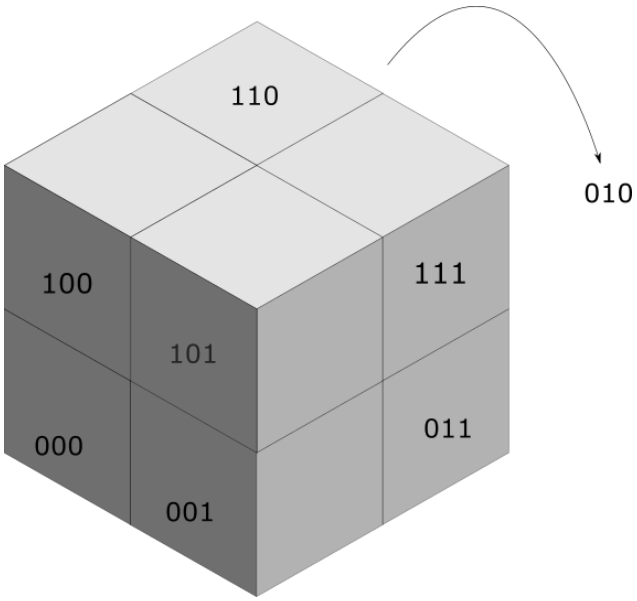

Figure 4.4: Example of an octree with morton keys of each node. The code of the hidden block is specified outside of the cube

The morton key k_n of a node n can be obtained either recursively from the tree hierarchy or from the geometry of the associated cube, thanks to an interleaving procedure that will be discussed later.
Following the first approach each octant get a 3-bit number between 0 and 7 assigned, depending on the specfic node's relative position to it's parent center (Figure 4.4).
The key of any child node n will be the concatenation of the octant numbers of all the nodes from the root down to the node in question.

Layer extraction

It is possible to automatically get the node's layer knowing only its locational code. To do so one must set to 1 the key of the root node, or, in other words, a 1 bit must be used as a flag to mark the end of the sequence. Without such flag, it would not be possible to distinguish between certain nodes, e.g. `000` and `000000`

or `001` and `000001`.

So, using 1 bit for the root node and 3 bits for each tree level, a 32-bit morton code can represent at maximum 10 layers, while a 64-bit morton code can hold up to 21 layers. Given k_n, its depth can be computed as $d = \frac{1}{3}\log_2(k_n)$.
It is always recommended to avoid using explicitly a logarithm for real time calculation: depending on the language adopted there can be many alternatives using bit scanning intrinsics.
For instance, it suffices to retrieve the most significant bit out of the key, and divide the result by 3, once the 1 flag as been removed. To do so without any looping, a bit masking procedure has been used in our implementation.

Interleaving

An interleave sequence is a mathematical object obtained by merging two sequences via a so called *in shuffle*. As outlined above, the key can also be computed from the depth d of node n and the position of its center (x, y, z), or the relative position of the cell inside the given layer, [54], [3]. This feature is really useful since it can be exploited for at least the following purposes:

- In a full tree, it speeds up dramatically tree insertion procedures of point objects.
- It can be exploited to compute a list of both neighbor and *interaction list* (see Chapter 3) of a given node.

To convert a certain set of integer coordinates (*tree coordinates*) to a Morton code, first decimal values must be converted into binary and then interleave the bits of each coordinate:

1. (x,y,z) = (3,9,1) = (0011,1001,0001)
2. Interleaving bits :010000001111 = 1039th cell along the Z-curve

Similarly, a decoding procedure exits to convert a morton code into integer coordinates. More technicalities can be found in [3] and a `C++` encoding/decoding library is available in [30].

4.3.2 Manipulation

As for AeroX aerodynamic simulations, a full octree built with morton codes is allocated. Even if more memory is required, the advantages of this approach with respect to cells manipulation are unquestionable.

Especially for a full octree, the characteristics of a pointer-less tree can be exploited to manipulate neighbor and interaction list search, while a whole layer can be extracted with just a $\mathcal{O}(N)$ instruction. Layer extraction is very important for a FMM algorithm:as a matter of fact, the upward and downward swipes that must be performed in order to calculate coefficients involve a cycle onto each cell in the layer. In our case, an array of cells representing the Z-curve is allocated, so that in order to access a layer of cells, one must just return a pointer corresponding to the beginning of the layer in the array.

In principle, for each cell one can compute run-time both the neighbors and the interaction list. This, in itself, is not an expensive operation thanks to morton code bit-wise manipulations. However, since such searches must be performed for each cell hosting at least one particle, a data structure storing morton codes of both neighbors and cells in the interaction list is allocated at the beginning of the simulation, consuming in this way a small amount of extra memory.

Neighbors

Given a cell c it is natural to retrieve a list of neighbors keys. First the morton code of c must be decoded to retrieve the cell's relative position in its layer. Then, those coordinates must be incremented by one unit for each direction. Clearly, a check must be performed in order to verify if c is on the boundary of the tree or not. Once neighbors' coordinates are available, they will be encoded to get their

addresses in the tree.

Interaction list

Given a cell$c$ the procedure to calculate the cell interaction list is the following:

1. Get cell c parent address clearing the first three bits starting from the right from cell c morton code.

2. Compute the parent's neighbors with the procedure described above.

3. For each neighbor, get its children morton codes concatenating the bit representation from 0 to 7.

Chapter 5

Code description

5.1 Preliminaries

In this chapter, the structure of the solution algorithm implemented in the new AeroX release will be presented.
As briefly reported in the introduction, AeroX is a multibody unsteady aerodynamics potential solver, which can be run as a stand-alone program or it can be liked to a structural solver to provide aerodynamic forcing terms. In addition to that, steady 2D viscous effects can be recovered by means of empirical aerodynamic tables, known as C81 tables.

Within this section, the aerodynamic mesher is introduced. Moreover, few mentions are made on the parallel implementation of the source code.
Further on, aerodynamic objects are introduced, while giving an overview of how the program manage their methods and data within the simulation. At the end of the chapter, the simulation workflow is summarized and details on the implementation are given.

Aerodynamic mesher

AeroX can be used as a stand alone aerodynamic mesher tool to mainly discretize wings and blades. It is possible to generate thick wings or lifting surface meshes,

starting from a given mean-line locus. As for the former case, the user input consists of:

- a list of airfoils locii
- a span-wise airfoil distribution
- a reference line geometry in the three-dimensional space
- the kind of panel discretization and the total number of elements both for the span-wise and chord-wise directions.

The same applies if one asks for an axysimmetric mesh built with a prescribed aerodynamic profile (e.g. the shroud simulated in this work).
Once the mesh is built, it can e saved as an AeroX mesh or as standard `Nastran` mesh. At the same time a mesh generated with `Nastran` can be imported in AeroX to handle more complex geometries as fuselages, hubs and so on.

External libraries

Numerical operations (for instance matrix factorization and matrix-vector multiplication) are performed using the parallel implementation of the `Math Kernel Library` (Intel MKL) to speed up calculations.
Nevertheless, as it will be described in the next section, AeroX supports solid bodies unknowns decomposition to reduce to dimensions of linear problems, i.e. a solid body can be split into multiple parts, that will be solved independently from each other. For instance, a four bladed rotor can be split into four linear systems, or an aircraft can be decomposed into lifting surfaces and fuselage to be solved separately. The mutual aerodynamic influence of each part consists in the mutual induced velocity, which can be taken into account or not depending on the user input, so that further control on simulation time and accuracy is available. However, further investigations are required in order to assess, given the modern computing resources, if and how a single larger system performs batter (in terms of simulation time) compared to multiple smaller systems.

Parallel architecture

The software architecture has been arranged to support a future implementation of an hybrid `MPI + OpenMP` approach. As a matter of fact, one of the major requirements was to make a worthwhile implementation both for the execution on a single machine, but also for cluster computing.

Due to the limited time available, the code was tested in multi-threaded environment only with the aid of the OpenMP library. The following steps were involved:

- Linear system coefficients computation
- Linear system factorization
- Inducing procedures, from any kind of object to any kind of target object
- Multipole coefficients computation
- Forces retrieval
- Position and velocity update, both for bodies and wakes

In this frame of reference, a future study of parallel octrees will be undoubtedly useful to enhance performance.

5.2 Aerodynamic objects

Exploiting an object oriented strategy, AeroX itself is built as a class that interacts with aerodynamic object like bodies, body parts and wakes, as well as an input interpreter and an output manager (Figure 5.1). The solid objects can be grouped in the so called *bodies*, so that each body is made up of a set of *body parts*.

Each body is linked to a given set of wakes, depending on the user input, through an object termed *Wake Interface*, which is supposed to specify the releas-

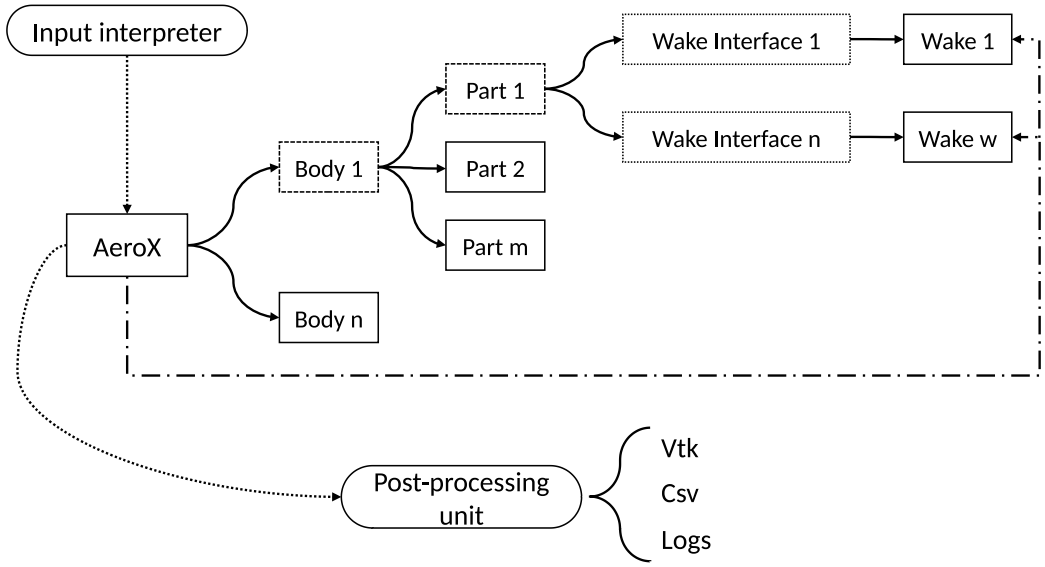

Figure 5.1: AeroX objects hierarchy conceptual diagram

ing law of vorticity, and to share such vorticity with the attached wake[1].

The quantities associated with a wake interface are the unknowns of the problem and so they take part to into the linear system. Once wakes and bodies are defined, the user is also asked to input a so called *Inductivity Table* that defines the overall induction rules. This means that one can easily enable or disable the induction of an object A upon an object B (body or wake) and vice-versa. The inductive properties of each body directly affects the solution cycle (Figure 5.4). Hence, depending on the problem, one can make a sensitivity assessment on the induction. This has been proven to be very useful to speed up calculations at the first stages of a new simulation. Moreover, given an arbitrary table, each body is

[1]A wake interface is an abstract object. It can be instantiated either as a standard panel interface or as a wake buffer releasing particles, or any kind of wake object (e.g. a *constant vorticity contour* (CVC) wake). For instance, a wake interface linked to a wing will prescribe the Kutta condition. In a way, the wake interface can be considered an exchange buffer meant to make a body part and its wake independent form each other.

given one of the following attributes:

- *Non-isolated*: at least one object except from the current body is inducing upon the current body. Hence, the non-isolated body will be solved inside a multibody loop.

- *Isolated*: The current body is not a target of any induction by any object in the field. This body will be solved at each time-step outside the multibody loop.

- *Isolated and fixed*: The current body is isolated but also fixed, i.e. it is not moving or deforming for the entire simulation. This body will be solved only once at the first time-step. For instance, as a first order approximation, a fuselage can be modeled as an isolated and fixed object, so that its aerodynamic influence on the rotor will be only computed at the beginning of the simulation.

5.2.1 Bodies

An AeroX body encapsulates a set of *body parts*, Figure 5.1. A part is an abstract class: a different part type can be cast in order to represent a different aerodynamic model (lifting lines, lifting surfaces or 3D bodies (i.e. thick panel surfaces)).

As stated in the previous section, each body is organized in parts in order to split the computational effort upon separated systems. Moreover, each body which is defined as non-isolated takes part in a multibody cycle (§ 5.4). Inside such loop, the objects configuration is frozen and the parts solution is called iteratively, updating the unknowns and hence the body mutual induction until convergence.

As long as the body part is rigid, the matrix gathering influence coefficient is computed only once, at the beginning of the simulation. Then the influence of the wake interface is added to the above-mentioned matrix and the system is factored. This is done at each time-step if the body is non-isolated (i.e. the wake shape evolves each time-step), otherwise the factorization is performed only once.

Clearly, if the body under exam is also deformable (i.e. its displacements are the results of an elastic solver) the influence coefficient matrix will be computed at hence factored each time-step.

Motion objects

Each body must be linked with at least with one part. Every body is provided with a local reference frame and, if specified, it can be associated with a so called *motion object*, which is created to manage rigid displacement.
In the input file, an arbitrary number of moving terns can be specified, or in other words, a set of motion laws. By doing so, every time the body movement is called, each point P will be updated building the motion chain dependency. Additionally, kinematic velocity is computed by means of a specified finite differences scheme. After that, geometric quantities associated with panels are updated (normal vector, centroids, geometric surface area and so on).

Motion objects can be also associated with body parts, so that complex prescribed motions can be easily achieved. For instance, an helicopter rotor is built as a single body, and each blade is an independent body part, so that one could assign rotation to the whole rotor, while prescribing for example a *cyclic stick* motion.

Aerodynamic inductive elements

Starting from a mesh object, body points are allocated. Then, by means of the mesh connectivity table the following two inductive entities are generated:

- *Panels*. For a lifting surface model, panels host a constant doublet distribution while for thick bodies panels host both a constant doublet and a constant source distribution. In those cases, they are employed as inductive elements suitable to compute induced velocity and/or induced potential (§ 2.2.3).

- *Vortical Edges* store the appropriate circulation depending on the surrounding panels. For the lifting line and surface models, they are employed to compute aerodynamic forces on the body (§ 2.2.3).

Both are allocated for any kind of part type, with the exception of the lifting line model, in which only edges are present. Since a mesh for a thick body could in principle be very large, edges connections are reconstructed by means of a hash table, that stores the connectivity of all edges, which are recovered by means of a hash function that returns the edge address.

5.2.2 Wakes

As far as this thesis work is concerned, vortex lattice wakes and particles were implemented and tested, nonetheless other kind of wakes can be easily added to the source code. Each wake can be linked by the user to any of the specified part through a wake interface, that was briefly introduced above.
In this section, both kind of wakes are briefly described, underlining critical aspects that were accounted for during the implementation.

5.2.3 Vortex lattice wake

The inductive elements of a vortex lattice wake are vortical edges shaping a connected frame.

Points and the inductive data structure (intensities, connectivity and cut-off radii) are managed inside a data structure resembling a *circular list*. This means that all the resources needed are allocated at the beginning of the simulation: then, once the buffer is full, the new released quantities will override the oldest elements present in the field. The user is asked to input the magnitude of this buffer, depending on the case study.
Within this frame of reference, the stack is filled sequentially: when it is filled completely, new elements are added without the need to move data in memory.

One one hand this precaution can save memory, on the other hand it reduces real-time pointers dereferencing.

Even though vortex lattice wakes has been used for years with profit, an intrinsic weakness is related to the fact that this wake misses a viscous model, that for practical applications is essential.
To address this point, the induction expression for vortical edges takes into account a cut-off radius, that smooths out the induced velocity for points close to the edge itself. Hence, this allows to define a *empirical viscous model* based on the release age of the inductive element. In this manner, the cut-off radius will evolve separately for each edge according to a semi-empirical law of growth, so that it changes the way the induction affects elements in the vicinity of the wake. A well known viscous model is the one obtained from the Lamb-Oseen vortex model [35]. The latter is a solution to the one-dimensional *laminar* Navier-Sokes equations, i.e. an axisymmetric solution for the swirl velocity with the assumption that the axial and radial velocities are zero:

$$U_\theta = \frac{\Gamma}{2\pi r}\left[1 - \exp\left(-\frac{r^2}{4\nu t}\right)\right] \tag{5.1}$$

where Γ is the vortex circulation, r the radial distance and ν the kinematic viscosity of the fluid. The viscous core radius is defined as the radial location where the swirl velocity is maximum:

$$r_c(t) = \sqrt{4\alpha\nu t} \tag{5.2}$$

in which $\alpha = 1.25643$ is termed the Oseen parameter. Therefore, within this frame of reference, the following expression that evolves the core vortex radius in time is implemented:

$$r_c^i = \sqrt{4\eta\alpha\nu r_{c(i-1)}^2 t^i} \tag{5.3}$$

where t_i is the simulation time elapsed from the release step of the element, and η is an empirical parameter used to tune the rate of diffusion.

Remark. it is worth noticing that where a vortex lattice wake needs an empirical viscous model, the particles one already embeds viscosity in its system equations. This bring a remarkable difference in practical computation, since sometimes the vortex lattice wake viscous model needs to be tuned to retrieve reliable results.

5.2.4 Particles wake

Particles wakes can be allocated either as free wakes or as bounded wakes. In the former case one can simulate a fluid dynamic object such as a vortex sheet or vortex ring alone. In this case, that wake will be associated with a FMM solver and an octree. In the latter case the wake will be bounded to the solution of a body, and the FMM management duty will be given to the *AeroX* class, since both objects must be treated inside a standard simulation. This means that the body will be solved and then the vorticity will be released into the wake according to a specified criterion, depending on the wake interface.

Hybrid wakes (i.e. starting with a vortex lattice buffer) can also be instantiated, taking advantage of OOP inheritance features.

Multiple wakes management

As mentioned before, AeroX aerodynamic objects have been designed to be independent, so that one can manipulate their characteristics unilaterally (such as the ability to induce or not upon another object). As for particles wakes, if the vorticity transport equations are going to be solved without the FMM (i.e. in direct manner) this is still the case.

Unfortunately, due to the intrinsic nature of the FMM itself, when multiple particles wakes are present, they must be considered as a single entity. As a matter of fact, all particles must participate in multipole coefficients computation. Dif-

ferently, it would not be possible to consider the influence on a group of particles onto the other. Moreover, making and managing more the one octree would not be feasible.

To address this issue, the solver collects all the particles wakes, linking them to the FMM manager and octree. Hence, at each time-step the multipole coefficients are computed taking into account all particles; moreover, the induction is also performed by the FMM manager.
Each wake still stores a pointer to the list of its particles, using it for position update, velocity update and graphical output.
As for hybrid wakes, the induction performed by the vortex lattice and the exchange buffer is managed by the wake itself, while the particles management is still given to the FMM solver.

FMM architecture

As mentioned before, the FMM algorithm is applied only on those cells whose children host particles (those cells are then termed *active cells*[2]).
Thus, it is quite natural to compute particles self induced velocity since each particle will surely be inside a cell whose multipole expansions has been computed. Hence, in this scenario, the self-induced velocity will be easily computed applying the induction algorithm explained in Chapter 3.

Clearly, it can happen that the induced velocity on a point not belonging to a cell where multipole data has been calculated is requested (e.g. a point belonging to a different wake or on a solid body).

On the other hand, in order to compute the approximated quantities also on those points, local expansions coefficients are needed, so that far and middle field

[2]Each cell can host up to n different data types, which can be optionally specified at the beginning of the simulation. Every time a datum is stored in the tree, the cell is flagged as active for that specific data type, and so its parents (in a recursive manner), exploiting the $\mathcal{O}(1)$ access time provided by a hash search.

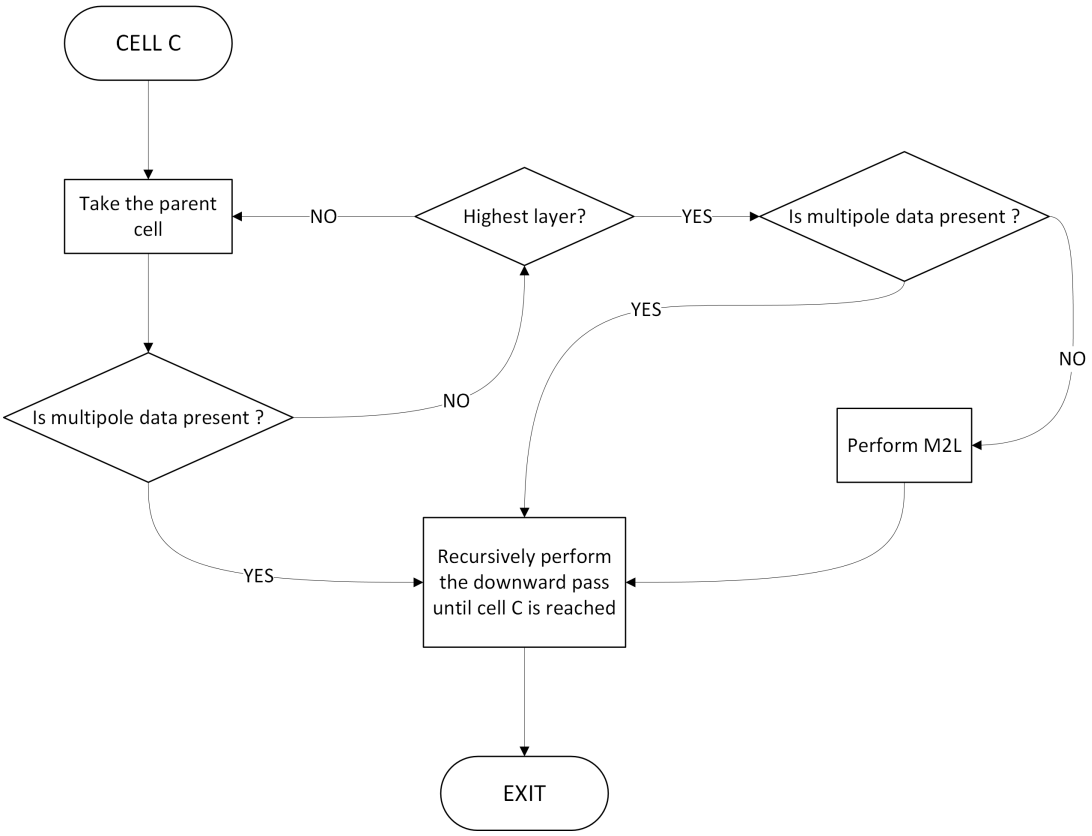

Figure 5.2: Empty cell multipole data computation workflow

contributions can be taken into account. As a remark, multipole coefficients are not calculated in all cells for obvious reasons: clearly, there are cases in which the vast majority of cells could be occupied by particles. However, it would be useless and inefficient to pursue such path, since the algorithm must be adopted for a variety of cases.

In order to counter this need, a simple algorithm was developed, and its simplified form is illustrated in Figure 5.2.
Each time a point is located in a cell with no multipole data available, two main actions are initialized, one after the other. For details on cells manipulation, one may refer to Chapter 4.

- A recursive search is made by sweeping the tree upwards starting from the cell were the target point is located

- Once an active parent cell is found[3], a downward sweep through the tree is performed, until the target cell is reached. In this manner, local expansion coefficients are efficiently pushed where needed. As a matter of fact, only the minimum number of operations is performed (i.e. the multipole downward pass is restricted to the path traveled by the upward search).
  If the highest layer is reached (the second layer in this context), without finding any active cell, first local expansions coefficients are computed, and then the actual downward sweep is initialized.

The actual implementation takes also into account that multipole data is never deallocated from memory nor reset, to save computational time. Hence, further checks are made in order to assess if the data stored in the cell can be used in the current timestep, or if it is just a deprecated data that must be calculated again.

5.3 Sections

AeroX take advantage of a structured mesh, typical of an wing mesh, to build a class, termed *Aerodynamic Section*, whenever the user asks for it. In this way, a section object is a class encapsulated in wings or blades. A group of sections is built starting from a wing span-wise distribution of panels.
The section consists of:

- a local reference frame
- a link to all the panels enclosed in the section
- a data structure for force retrieval
- the section trailing and leading edge locus

[3]Each time the tree is asked to find an active cell, it will perform a search according to the specific data type of the object that is asking this search. In this context, a cell is active if a multipole datum is allocated inside it.

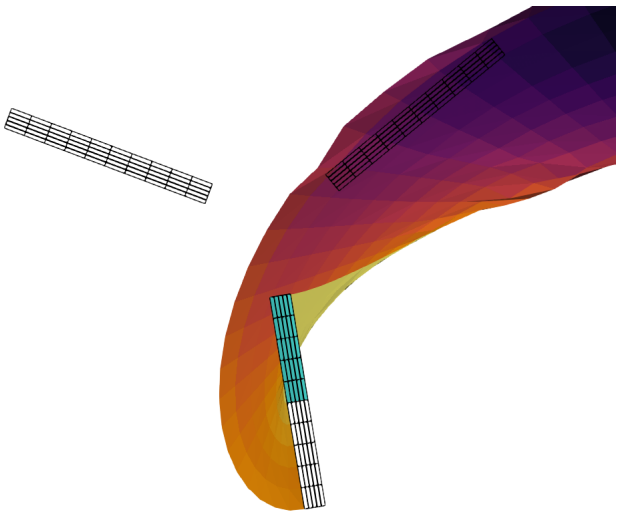

Figure 5.3: Inverted flow regime in a retreating blade (three-bladed rotor), top view. Cyan sections are disabled in this configuration. Flow coming from the left.

- a link to the section's releasing objects in the wake
- a C81 table, created form an input file or interpolated from two different input tables.

While the code automatically checks for span-wise panel stripes, the user is requested to provide a set of span-wise non-dimensional nodes where aerodynamic properties of a 2D section should be placed. Aerodynamic properties are given, as mentioned before, by C81 empirical tables. Then, the program will automatically interpolate the given tables whenever a cross section is placed in a position between two different aerodynamic table nodes.
Aerodynamic sections are a tool used mainly for the following purposes:

- Wing sections can be used to retrieve and eventually correct the 2D aerodynamic behavior of a portion of the wing or blade. Hence, 2D viscous effects can be introduced in the potential solver. This is done by means of the above-mentioned tables, which are text files that list drag, lift and pitching moment coefficients of a given airfoil as a function of the angle of attack and Mach number, usually obtained from wind-tunnel measurements. If this

correction is enabled, the steady actions projected in wing axes are corrected based on the empirical C81 coefficients.

- Non-lifting sections can be created by specifying bluff body-like C81 tables. If this happens, all the panels associated to those sections will still take part in the solution process, but no wake will be released from the section's trailing edge locus. In this manner, the non-lifting behavior can be modeled, preventing that portion of the wing from releasing vorticity. At the same time, the drag effect can be taken into account by means of the aerodynamic empirical tables, as described in the previous point.

- Sections can enable or disable panels induction and nullify vorticity released in the wake.

  For instance, sections are flagged as disabled whenever a retreating blade of a rotor in a forward flight maneuver experiences an inverted flow regime. This can happen when the kinematic velocity associated with the rotation is smaller compared to the free stream velocity (Figure 5.3). When this happens, the portion of the wake circulation (releasing from the involved sections) is nullified, since that body chunk is not in a lifting condition. This is done until the blade section moves to a region where the flow is not inverted. Moreover, all the panels associated with disabled sections are prevented from inducing on any object until section activity is restored. However, their drag effect won't be neglected.

  This process can be also triggered when the local section aerodynamic angle of attack exceeds given bounds, so that stalled conditions can be somehow modeled.

- Sections comes in handy for post-processing purposes, while providing an input object to the structural solver. Pressure coefficient distributions can be easily plotted and force retrieval can provide aerodynamic forcing terms in section axis, wing axis or in the absolute frame of reference.

5.4 Solution cycle

In this section, a global overview of the aerodynamic solution algorithm is presented (Figure 5.4).

After all objects has been correctly initialized from the input manager, the solution cycle is triggered. It consists of multiple internal steps and subroutines, which can be split in a part related with the solution of linear systems and in a second one related with induction, release and objects update.

Outer loop

Each time the multibody loop reaches convergence, the time step is advanced and every wake in the field releases new vortical elements based on the solution computed at the previous time step (*wake release step*).
Clearly, those elements depends on the type of the wake under analysis. For instance, vortical edges will be created for a vortex lattice wake; vortical filaments will be allocated for a constant vorticity contour wake.
As for the particle wake, the release phase is subordinated on the type of wake interface the wake is linked to. If a panel interface is present, vortical edges are released and stored inside an exchange buffer, which can perform the induction and move like any other object in the fluid field. As we will see later on, the particle transformation is not managed in this phase.
If the particle wake is hybrid (i.e. vortex lattice and particles coexists, and particles are generated form wake edges after a buffer is filled), then the release duty is given to the vortex lattice component of the wake.

Once the release phase is completed, the Inductivity Table (§ 5.2) is examined, and according to the induction rules, velocity induction is performed upon each wake, potentially from any other object in the field (*Wake induction phase*).

In the *Wake evolution phase*, wakes position is integrated in time according to the time integration scheme selected. Wake interfaces are also updated in this step.

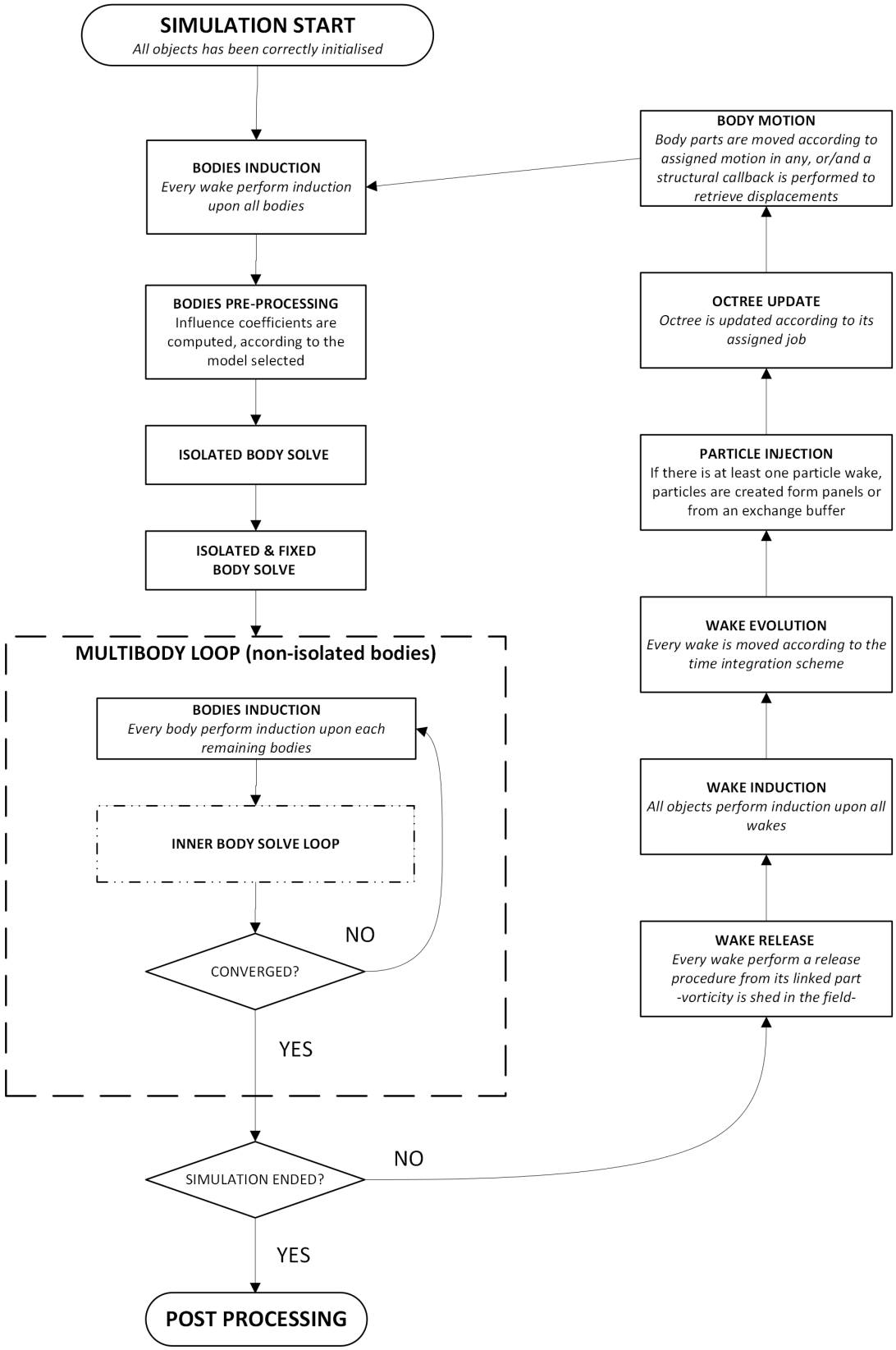

Figure 5.4: AeroX main simulation workflow

If particle wakes are present, the *Particle injection phase* is initialized. How the release is performed depends again on the wake interface. For instance, if the wake is hybrid or if it is a full particle wake with a panel interface, particles are initialized according to the scheme described in [58]. The particle injection step was separated from the the wake release due to the FMM algorithm. As a matter of fact, if the particles velocity field is going to be solved with a direct approach, there is no need to split these two operations: vorticity can be released and transformed into particles simultaneously. On the other hand, using an octree structure, new wake elements are generated, every fluid object is updated, hence particles are ready to be released. Finally, the octree can be updated. Without doing so, one would be compelled to update the tree twice a timestep, prior to the wake release phase, in order to compute the induction on bodies and wakes, and then after new particles are released, since in the meantime the wake were evolved. Clearly, updating the octree more the once a timestep would not be feasible, since it is the most expensive operation (in terms of computational time) of the whole FMM method.

Therefore, it comes the *Octree update phase*. This step involves three stages:

- The tree is emptied from particles and the leaves list is cleaned
- All the particles, whose position were just integrated in time, are then inserted again in the hierarchical structure. The phase is relatively fast if the tree is a full tree and particles are added in the last layer only.
- Finally, multipole coefficients are computed with the procedure described in § 3.2.3: This is the most expensive stage.

Once the octree is fully updated, all the bodies are moved according to the assigned rigid kinematics if any, or structural displacements are applied after the structural solver is called (*Body motion phase*).

Body loop

Once that all bodies motions and displacements are applied, an induction routine is called, so that all bodies becomes the target of an induction process from all wakes.

The next step is the one related with the computation of the linear system's coefficients (*Body pre-processing*), according to the body type and of the body inductive attributes (§ 5.2.1).
When this step is completed, the solution process for *isolated* and *isolated and fixed* bodies is initiated.

If *non-isolated* bodies are present, the execution flows into the *multibody loop*. Inside the multibody loop, each non-isolated body is solved iteratively. If the convergence condition is not satisfied, then an induction routine is called, so that the body mutual induction is computed with the updated values of the unknowns. Hence, the body solution subroutine is called once again. The process is repeated until convergence. In order to assess the multibody convergence, the norm of the solution difference between iteration k and iteration $k-1$ is computed.
This is done for each body inside the multibody loop. Then, the resulting arrays are normalized with respect to the highest unknown value found in the body. Finally, the maximum value among all the bodies is compared with a given accuracy threshold.

5.4.1 Body solve loop

The inner body-solve loop declared in Figure 5.4 is an additional cycle meant to manage the solution of the whole body. Specifically, the body-solve loop cycles onto each body part calling the algorithm sketched in Figure 5.5.

The body part solution loop builds the system right-hand side. In turn, the right-hand side expression depends on the type of the aerodynamic model selected for the current part (Chapter 2). Then, the LU decomposition of the system matrix is used to solve the linear system. Once this task is completed, the part self-induced

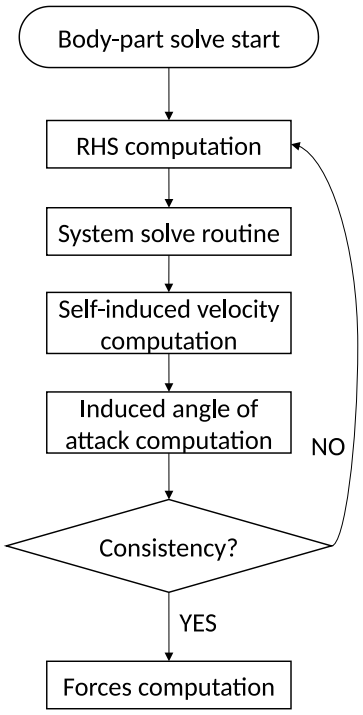

Figure 5.5: AeroX body part solve loop

velocity is calculated. Once again, the way this task is accomplished depends on the aerodynamic model. The lifting surface model exploits the influence coefficients already calculated for the system assembly, while the thick body model builds the self induced velocity differentiating the potential on the body surface.

At this point, if sections are present, the induced aerodynamic angle of attack is computed for each section. This value is used to assess sections activity congruence, by comparing that angle of attack with a given activity threshold (§ 5.3). If, prior to the system solution, a section was active, its status must be active after the new unknowns were computed. If this not happens, the section is disabled and the algorithm is restarted with a new iteration. As soon as the part is found to be congruent, forces and moments are computed.

Chapter 6

Test cases

In this chapter, a set of interesting case studies were selected and reported. In the first section, specific tests concerning the FMM implementation are outlined, and the results are shown. Then, in the second part of this chapter, simulations involving the aerodynamic solution cycle are discussed. Finally, the last part of the work covers a preliminary survey of the code performance simulating a ducted rotor.

6.1 Induction tests

In this section, the particle wake alone is examined and the results of two literature test on particle induction will be reported. There are many interesting studies on this topic: first of all, how the multipole order affects the result, but also the influence of the tree itself. Second, the parallel scalability of the whole method. As far as the current work is concerned, two literature test were selected in order to assess the implementation of the FMM inside the code.

The first one involves the calculation of the induced velocity by a Gaussian vorticity distribution, which is very convenient for this kind of test. As a matter of fact, first it provides an analytic expression for the exact induced velocity, and second the Gaussian distribution rapidly decays at infinity, so that a representative finite volume can be easily discretized.

As far as the second test is concerned, it involves the evolution of a vortex sheet, i.e. a material surface across which the tangential component of the fluid velocity experiences a jump discontinuity.

6.1.1 Gaussian vorticity distribution

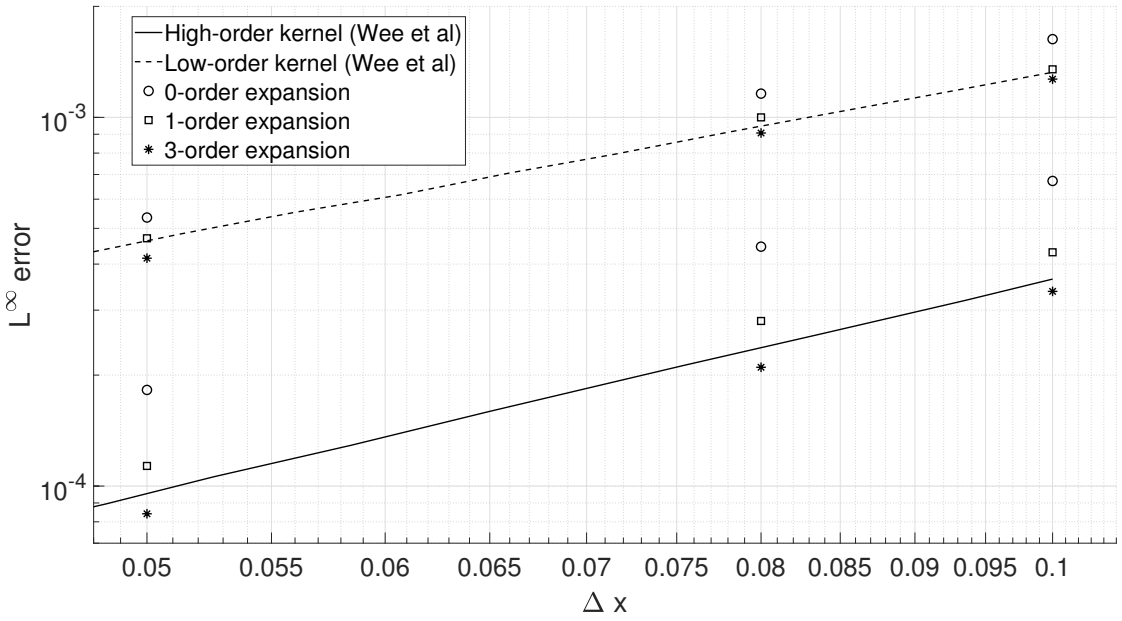

Figure 6.1: L^∞ error of u_3 versus Δx

This test has been performed by Wee et Al. [59] to assess the convergence rate of two algebraic kernels, that were discussed in Chapter 2. It is important to stress that that two well separated kinds of error arise in this context. First of all, there is the so called *desingularization* error, which is strictly related with the kernel being used. Trivially, it measure the difference between the singular kernel and the algebraic one. Then, other sources of error are the discretization error and the one related to the multipole algorithm. As stressed in the same paper, the error due to desingularization becomes a meaningful estimate of the overall error for the case where the error due to desingularization is small.

An in-depth analysis on this topic goes beyond the practical implementation reported in this thesis, bu this case study has successfully been used to test if the

algorithm were correctly implemented.

The main relations are reported here for convenience. Given the following Gaussian vorticity distribution,

$$\boldsymbol{\omega}(\boldsymbol{x}) = \frac{\hat{\boldsymbol{e}}_2}{(2\pi)^{3/2}} e^{-|\boldsymbol{x}|^2/2} \tag{6.1}$$

the exact induced velocity field using a singular kernel reads:

$$\boldsymbol{u}(\boldsymbol{x}) = (\boldsymbol{K} \star \boldsymbol{\omega})(\boldsymbol{x}) = -\frac{\boldsymbol{x} \times \hat{\boldsymbol{e}}_2}{4\pi|\boldsymbol{x}|^3} \left(\text{erf}\left(\frac{|\boldsymbol{x}|}{\sqrt{2}}\right) - \sqrt{\frac{2}{\pi}}\, |\boldsymbol{x}| e^{-|\boldsymbol{x}|^2/2} \right) \tag{6.2}$$

As pointed out by Wee et al., exploiting azimuthal symmetry of the vorticity field, a meaningful estimate of the error should be found along the line $x_2 = 0$, $x_3 = 0$. Furthermore, they found that the maximum error actually occurs near $x_1 = 1$. Moreover, among three components of the velocity field, i.e. u_1, u_2 and u_3, only u_3 is non-trivial, hence we will follow the same approach: the third component only will be reported here.

Numerical discretization of the vorticity field reported in Equation 6.1 is made only for $-4 \leq x_i \leq 4$ which provides a reasonable cut-off distance. Within the domain of discretization, a uniform grid of particles is generated, whose vorticity strength has the following shape:

$$\boldsymbol{\alpha}(\boldsymbol{x}) = \sum_p \boldsymbol{\omega}(\boldsymbol{x}_p) \Delta x^3 \delta(\boldsymbol{x} - \boldsymbol{x}_p) \tag{6.3}$$

where the index p runs over the grid points. The overall convergence rate is estimated by refining Δx (keeping fixed the ratio $\sigma/\Delta x = 2$), while performing

$$\boldsymbol{u}(\boldsymbol{x}) = (\boldsymbol{K}_\sigma \star \boldsymbol{\alpha})(\boldsymbol{x}) \tag{6.4}$$

with $\boldsymbol{K}_\sigma$ being an appropriate non-singular kernel. The computation is performed with the FMM algorithm discussed in Chapter 3.

In the work by Wee et al calculations were performed from $\Delta x = 10^{-1}$ to $\Delta x = 2.0 \cdot 10^{-3}$, using the original tree code algorithm developed by Lindsay [38]. Details on the precision related parameters for the tree code can be found in [59]. For the purposes of this work, induction tests were carried on for $\Delta x = 10^{-1}, 8.0 \cdot 10^{-2}$ and $5.0 \cdot 10^{-2}$, the latter case corresponds to more than 4.17 million particles. Wee results has been digitalised from the original image and reported into the same plot for our range of interest.
In Figure 6.1 the results of the investigation are shown, where the error is plotted against Δx. The low-order kernel referenced in Figure 6.1 is also known as Rosenhead-Moore kernel, while the high-order one is sometimes termed Winckelmans-Leonard kernel. Both kernel expressions can be found in Chapter 2, respectively in Equation 2.51 and 2.52.

The results, obtained with a 5-layers full octree shows a good agreement with the ones computed with a different approximation method. As expected, the absolute velocity error is much smaller with the high-order kernel than with the low-order one. Moreover, increasing the multipole order of expansion the error norm decreases, preserving the same trend reported by Wee et al.

6.1.2 Vortex sheet evolution

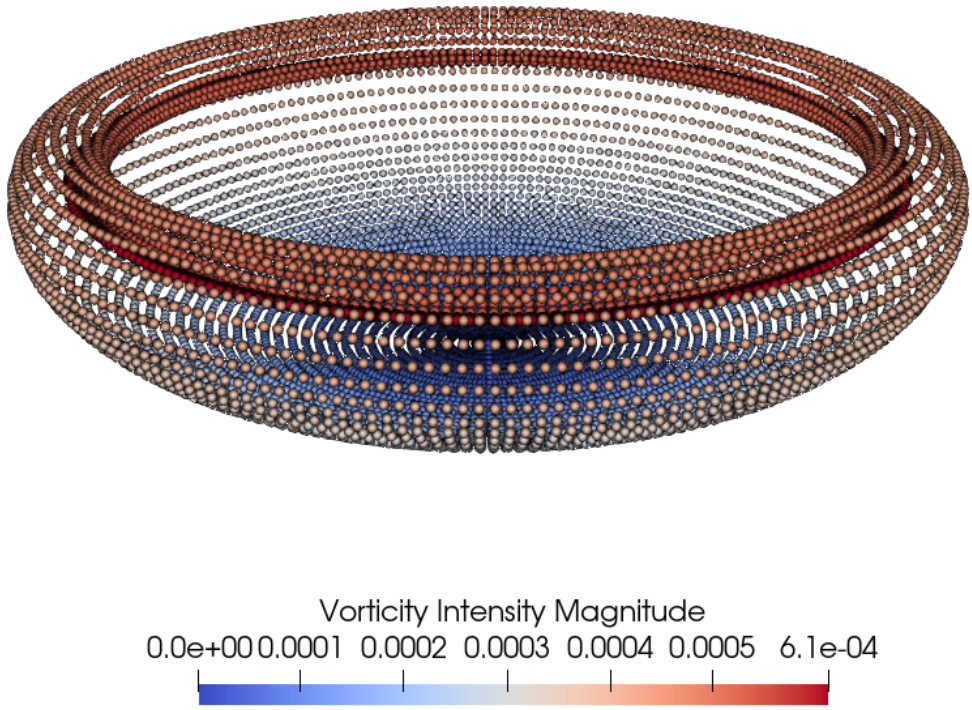

Figure 6.2: Rolling up vortex sheet, 5 layers full octree, 6-order expansions. $\sigma = 0.1$, $t = 1$, $dt = 0.05$

In this section, the results of a particles evolution simulation will be presented. On one hand, the induction provided by the fast multipole algorithm was tested once again; on the other hand this was a benchmark to assess the functionality of the time evolution management of the whole particles wake and octree.

This case study was presented by Lindsay [37] in his PhD thesis, in which he discussed a tree-code algorithm implementation for the rapid computation of three-dimensional vortex sheet motion.
A *vortex sheet* is a material surface in the fluid across which the tangential component of the fluid velocity experiences a jump discontinuity. Away from the surface, the fluid is assumed to be irrotational: however, since the velocity has such discon-

tinuity, the vorticity must be a δ-function there. In his work, Lindsay studied the evolution of such structures to model the formation process and the interaction of vortex rings.

Parametrization

Following a Lagrangian parametrization presented by Caflish [9] and Kaneda [32], the sheet's position is denoted by $\boldsymbol{y}(\lambda_1, \lambda_2, t)$, where λ_1 and λ_2 are Lagrangian parameters.
λ_1 measures the circulation across the vortex lines and $0 \leq \lambda_2 \leq 2\pi$ parametrizes along the vortex lines. In other words, each λ_1 value corresponds to a vortex line. The latter is then discretized in λ_2 with a grid that is uniform with respect to arc-length, for $t = 0$.

Following Chorin and Bernard's desingularization procedure [11], one will get the following expression for the induced velocity, as a function of our Lagrangian parameters.

$$\boldsymbol{u}(\boldsymbol{x}, t) = \iint \boldsymbol{K}_\sigma(\boldsymbol{x}, \boldsymbol{y}(\lambda_1, \lambda_2, t)) \times \frac{\partial \boldsymbol{y}}{\partial \lambda_2}(\lambda_1, \lambda_2, t) d\lambda_1 d\lambda_2 \tag{6.5}$$

$\boldsymbol{K}_\sigma$ is a non-singular kernel. In this context the Rosenhead-Moore kernel (Equation 2.51) will be used, since that is the one exploited by Lindsay in his paper.
Now, the Biot-Savart integral 6.5 is then discretized in λ_1 and then in λ_2 with the trapezoid rule. Moreover, the $\frac{\partial \boldsymbol{y}}{\partial \lambda_2}$ term in the integrand is approximated with a 2nd order centered difference. Of course, this results in the very same system of ordinary differential equations reported in Chapter 2:

$$\frac{d\boldsymbol{x}_i}{dt} = \sum_{j=1}^{N} \boldsymbol{K}_\delta(\boldsymbol{x}_i, \boldsymbol{x}_j) \times \boldsymbol{\alpha}_j \tag{6.6}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are points on the sheet and

$$\boldsymbol{\alpha}_j = D_{\lambda_2} \Delta\lambda_1 \Delta\lambda_2 \tag{6.7}$$

is the product of the finite difference D_{λ_2} along a vortex line and the integration weights.

The initial condition is a flat circular disk of unit radius with circulation distribution $\lambda_1 = \sqrt{1 - r^2}$. In order to have a smooth distribution, the following change of variables could be employed:

$$\lambda_1 = \cos\alpha \qquad 0 \leq \alpha \leq \pi/2 \tag{6.8}$$

The core vortex cut-off radius is fixed and set to $\sigma = 0.1$. Each vortex line is discretized with $128(1 + r)$ particles where r is the radius of the vortex line, rounding the number of particles up to the nearest multiple of 8. In this manner, the total number of particles discretizing the sheet is 13444.

Time evolution

The original test this thesis is referring to were carried on with direct velocity induction and a 4th order Runge-Kutta method. As for the latter, it will be hardly used for complete aero-structural simulations. Thus, an explicit Adam-Bashforth scheme up to the 7th order has been implemented and tested here. However, most of the time, due to the overall complexity of a full simulation, an explicit Euler scheme will be used. However, Euler stepping precision is not enough for satisfactory results for this kind of test.

Different orders of the Adam-Bashforth scheme were tested, but in all cases the end of the simulation was fixed at $t = 1$. As a matter of fact, as the sheet evolves, the vortex lines can individually stretch and eventually separate from each other. This causes a loss of discretization resolution which must be overcame by inserting new particles along the lines and by inserting new lines. Lindsay suggest particle insertion algorithm [37], but since this goes beyond the scope of the our code all simulations were stopped prior to reaching critical conditions. However, as it will be underlined later, our results will be compared with the exact induction computation performed by Lindsay, that were carried on until $t = 1$ without particles insertion.

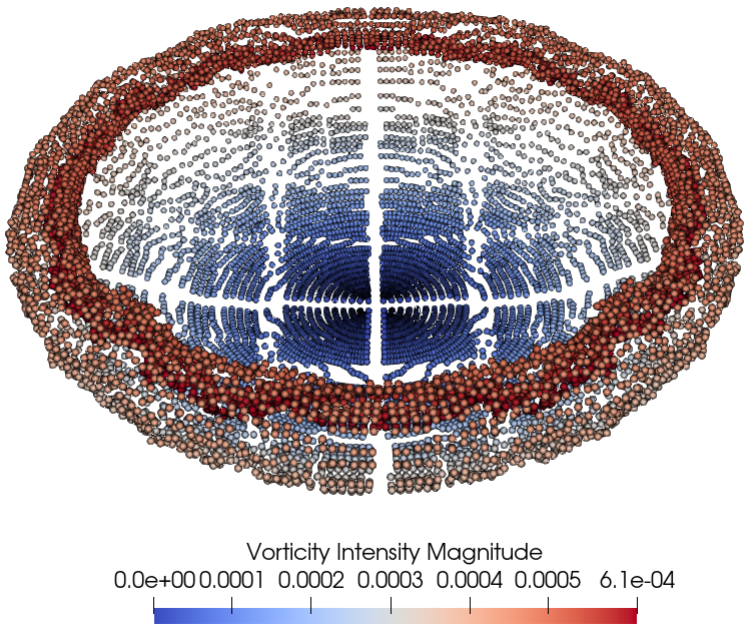

Figure 6.3: Rolling up vortex sheet, 5 layers full octree, 0-order expansions. $\sigma = 0.1$, $t = 1$,

Results

Figure 6.2 presents a 3D rendering of the vortex sheet at time $t = 1$, computed with a 5 layers full octree and 6° order multipole expansions. For reference, the same case with a zero order expansion is shown in Figure 6.3. The borders between two adjacent octree cells are clearly visible.

FMM order	error
2	$2.13 \cdot 10^{-2}$
3	$1.22 \cdot 10^{-2}$
4	$2.20 \cdot 10^{-3}$
5	$3.98 \cdot 10^{-4}$
6	$8.14 \cdot 10^{-5}$

Table 6.1: Maximum point position difference refining FMM order expansions $t = 1$, $\sigma = 0.10$

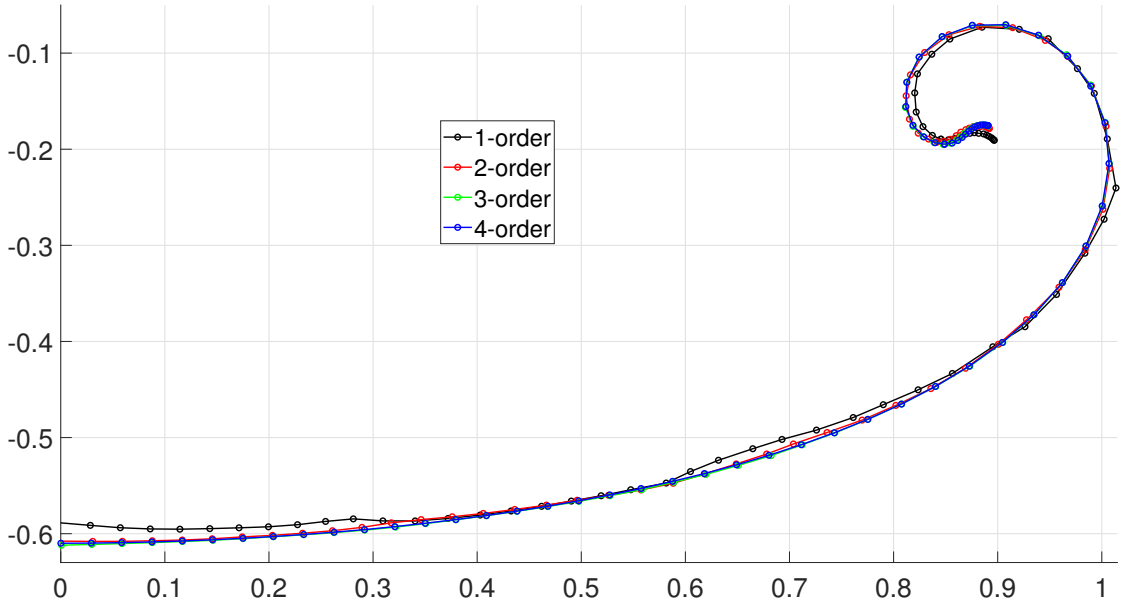

Figure 6.4: Vortex sheet section view at $t = 1$, $dt = 0.05$: comparison increasing the order of expansions

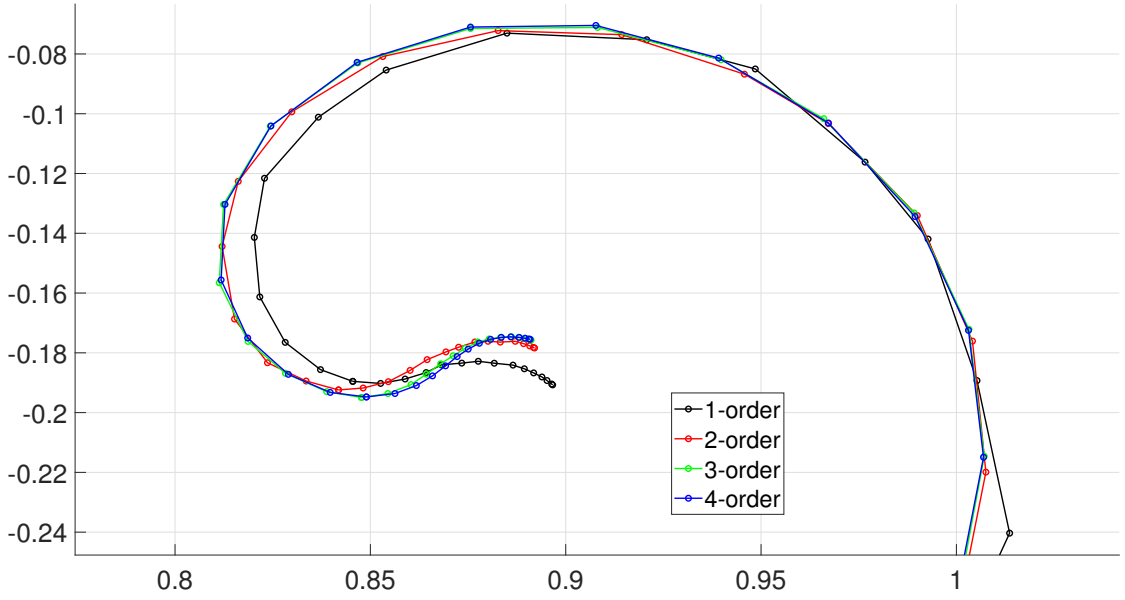

Figure 6.5: Vortex sheet section view at $t = 1$, $dt = 0.05$: comparison increasing the order of expansions - magnification in the vicinity of the tip region

In Figure 6.4 and Figure 6.5 a cross-sectional view of half of the sheet is given. The particle position is enhanced by a marker, and the locus is obtained with a simple interpolation. While the max depth of the tree is kept constant, the sheet shape converges increasing the degree of multipole expansion.
In Table 6.1 the particle position difference were computed for runs made with consecutive FMM multipole expansion degree, according to the following:

$$e_i = max_j \left\| \boldsymbol{x}_j|_{\text{order}_i} - \boldsymbol{x}_j|_{\text{order}_{i+1}} \right\| \tag{6.9}$$

A sensitivity assessment onto the time integration scheme is presented in Figure 6.6. The octree max depth, the degree of multipole expansion and the time step are kept constant while the time integration scheme was changed.
Moreover, the time integration scheme convergence was also tested and the results are reported in Table 6.2. Those were obtained with the 4th order Adam - Bashforth scheme, changing the time step and keeping all the other parameters fixed (5 layers and 6th order multipole expansions). The error values are obtained with the same expression reported in Equation 6.9

$\Delta t[s]$	e(Δt)
0.2	$1.72 \cdot 10^{-3}$
0.1	$9.82 \cdot 10^{-4}$
0.05	$1.02 \cdot 10^{-4}$
0.025	$7.21 \cdot 10^{-6}$

Table 6.2: Maximum point position difference refining the time step $t = 1$, $\sigma = 0.10$ 4th order Adam-Bashforth scheme

Finally, the sheet solution at time t=1 computed with the 4th order Adam-Bashforth scheme and a 4th order multipole expansion is compared with the Lindsay's sheet. The latter was computed with an all-pairs induction process and the 4th order Runge-Kutta numerical scheme.

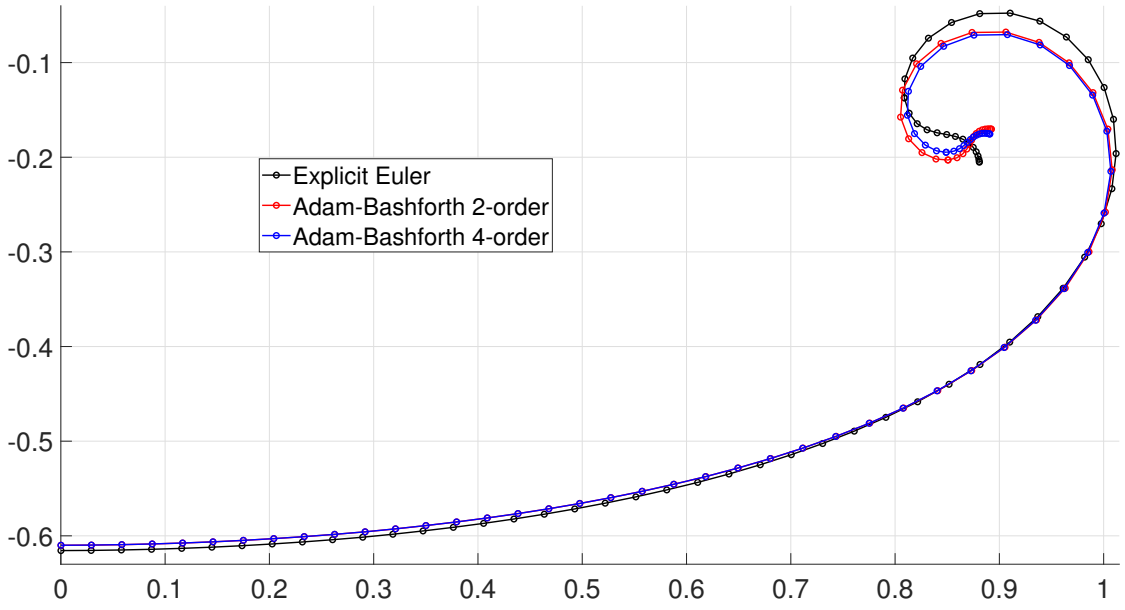

Figure 6.6: Vortex sheet section view at $t = 1$, $dt = 0.05$: comparison with different time integration schemes available

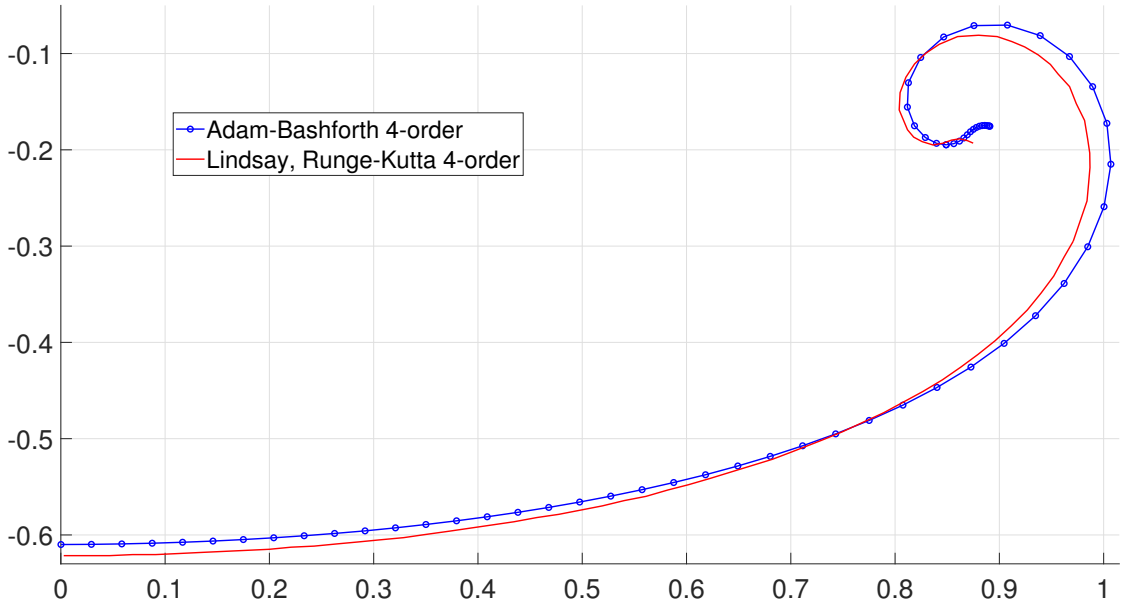

Figure 6.7: Vortex sheet section view at $t = 1$: comparison between 4° order Runge-Kutta scheme and the 4° order Adam-Bashforth one

The Lindsay's particle cross-sectional view displayed in Figure 6.7 was obtained by digitalizing the original image reported in the paper [37]. Unfortunately, the exact particles position was not reported.
In light of all the tests shown above, the differences visible in Figure 6.7 may be linked first to the different time integration scheme. On the other hand, there could have been some minor differences during the vorticity discretization procedure.

6.2 Validations

In the previous section, specific test were performed on the particle wake and on the FMM algorithm, so that the velocity induction on the wake on itself and the time evolution algorithm performances could be assessed.
In this section, more complex runs were made so that the whole solver was involved.

First of all, the new version of AeroX was compared with the older one though a hovering rotor test. The old code computed a lifting surface solution while with new version of the software a full panel solution was obtained (i.e. blades are modeled as thick bodies).
In the second part of this paragraph, the performance of the aerodynamic solver is directly compared with experimental data available in open literature, looking at integral quantities such as forces and pressure coefficients.

6.2.1 Older release comparison

In this section, the results of a preliminary validation test on a three-bladed hovering rotor are reported. The body cycle for a middle-level complexity case is tested, comparing both the C_T time history and the wake shape with the oldest release of the code, modeling the blades as thick bodies. The old release was capable to model rotors with *lifting lines* or *lifting surfaces* only. However, it is known from experience that a lifting surface solution exhibits a similar behavior with respect to a solution exploiting thick bodies. Hence, such comparison is considered fair enough, both from the wake shape standpoint and form integral quantities such as the thrust coefficient. Clearly, a vortex lattice wake was used for both cases, with the very same characteristics.

One major critical aspect when one wants to represent the thickness of a wing or blade, is how the discretization is performed near the leading edge. As a matter of fact, in the vicinity of that region large gradients in the pressure coefficient distribution can occur: the way this gradient is captured is trivially related to the calculated numerical force. Sometimes it is preferred to enhance the number of panels in that region only, for example by means of a Multhopp approach [43]. In this context, since a comparison with a lifting surface model was made, a uniform spacing for both chord-wise and span-wise direction is chosen. Calculations were performed with both codes changing the mesh parameters according to Table 6.3. All the global parameters were kept equal for both implementations.

Each revolution was discretized with 50 azimuth angles, and the computations were carried on for 11 revolutions, resulting in 550 steps. On the eleventh revolution, the wake released during the first blade rotation is deleted, in order to pertain a wake closer to convergence.

The convergence time history for the thrust coefficient is reported in Figure 6.8. Solid lines are related with the thick body solution computed with the new release, while dashed ones report the results obtained with the lifting surface model. As meshes are refined, solutions appear to converge to the same value. Interestingly

Blade mesh	Lifting surf.	Panel method
Mesh A	*chord* 5	*chord* 10
	span 10	*span* 10
Mesh B	*chord* 20	*chord* 40
	span 10	*span* 10
Mesh C	*chord* 30	*chord* 60
	span 20	*span* 20

Table 6.3: Number of panels for the chord-wise and span-wise direction for each tested mesh.

enough, it is clear that a solution built with the lifting surface model underestimates the thrust coefficient, while the opposite happens for the thick body model. With the finest mesh tested, the difference in C_T was less than 4%, showing a very good agreement.

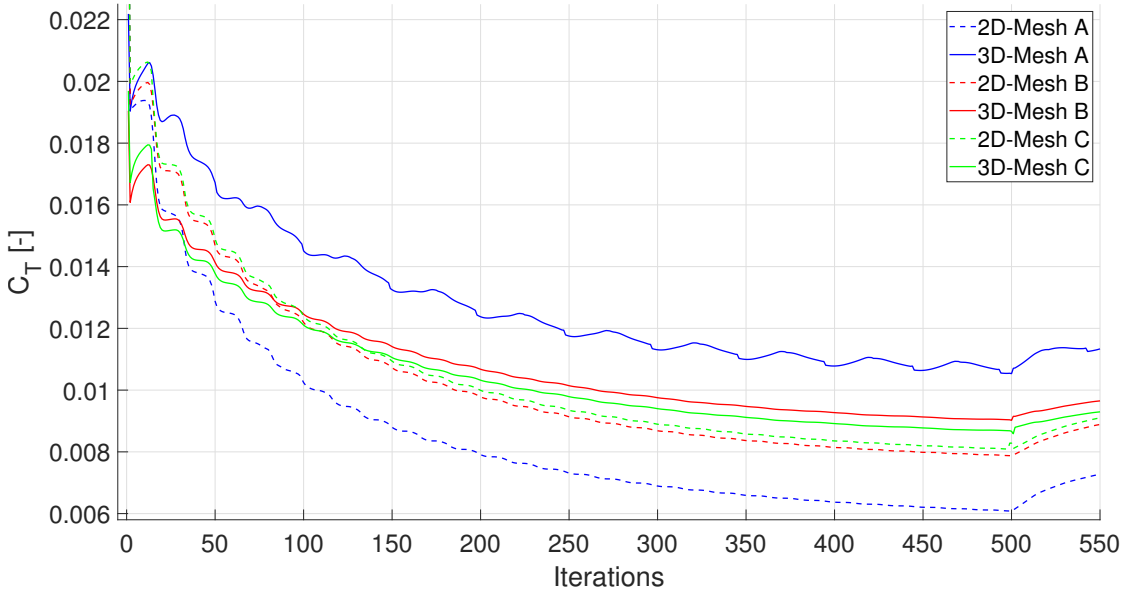

Figure 6.8: C_T convergence time history. Mesh details are referenced in Table 6.3. Dashed lines are referred to the old release, while continuous lines report data obtained with the new solver. 2D label refers to the lifting surface model, while 3D refers to a panel method

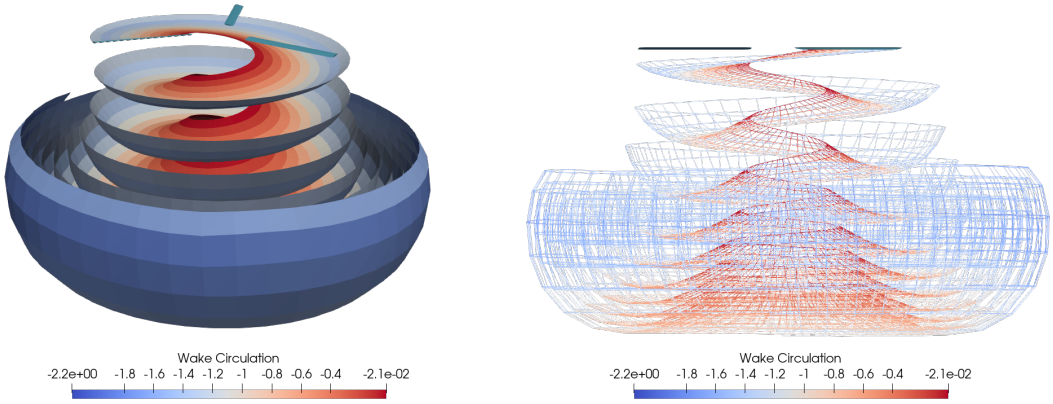

Figure 6.9: Vortex lattice wake visualizations, exhibiting the classical hovering shape. For clearness, one wake only is visualized

In Figure 6.9, a 3D and 2D rendering of the wake released by one blade at the final timestep are shown. As expected, the shape is neat with no sign of instabilities, proving the correct implementation of the vortex lattice diffusion model.

6.2.2 AIAA DLR-F4 workshop

A validation test was performed using the well-known DLR-F4 wing-body configuration, which is representative of transport aircraft designed for transonic flight. The results obtained with AeroX will be compared with those reported by the first AIAA CFD drag prediction workshop [36]. The latter was designed specifically to assess the state-of-the-art of computational fluid dynamics methods for force and moment prediction.

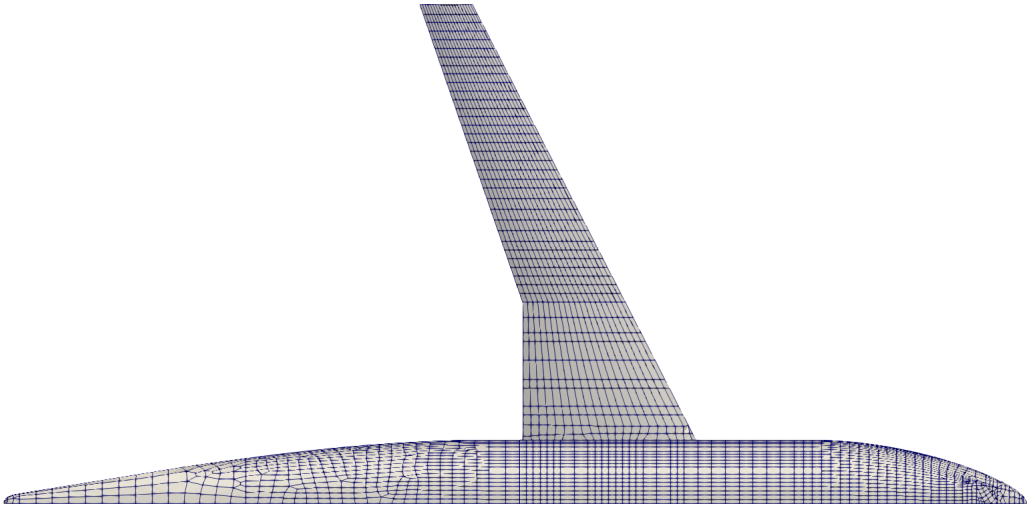

Figure 6.10: Half DLR-F4 aircraft model panel mesh. Top view

Mesh and simulation parameters

The airframe surface visualized in Figure 6.10 was generated with the aid of an external software with the total number of elements counting 9908 panels. The model was subdivided into three pieces: two wings and the fuselage. Hence, the whole body was solved with three smaller linear systems. Moreover, the fuselage wake was not taken into account and wing particle wakes only were released in this test.

In this way, the fuselage linear system was built and factored only once at the beginning of the simulation, hence reducing the computational effort. Moreover, the hybrid particle wake was used with a buffer of four panels (Figure 6.11), and

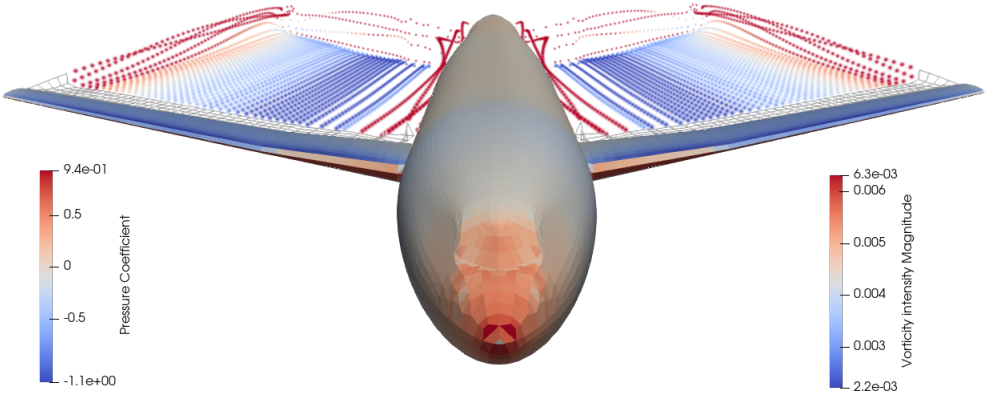

Figure 6.11: DLR-F4 particle wake development. Pressure coefficient distribution on the aircraft is shown

the particle cut-off radius length was 1/8 of the mean wing chord, taken in the stream-wise direction (0.1454 m).

The results shown here were obtained setting Mach number Ma = 0.60. The Reynolds number reported in the workshop for this case was $\mathrm{Re} = 3.0 \cdot 10^6$.
The timestep was $1.224 \cdot 10^{-4}$, so that the wake panels area was comparable to the body panels located in the proximity of the trailing edges of the wing. The simulation was carried on until the wake was convected downstream covering a distance proportional to 15 times the mean chord, so that a steady condition could eventually be reached.

Results

In Figure 6.11 a 3D rendering of the model and the attached wake is presented; the snapshot was taken in the early stages of the simulation and the starting vortex is visible, convected away by the free-stream, while the wake edges roll up as expected, at least for this speed order of magnitude. Clearly, as the free-stream increases the roll-up effect will decrease since the convective characteristic time will shorten. In those circumstances, the wake could be prescribed with a reduced loss in precision, but with a consistent reduction of computational time.

AIAA label	y/b
003	0.24
015	0.52
033	0.85

Table 6.4: DLR-F4 wing cross-sections position.

In Figure 6.13 the C_p distribution plotted for three cross-sectional positions is presented, comparing AeroX results with the ones obtained with DRA, NLR and ONERA codes ($\alpha = 0$). Table 6.4 reports the wing cross section labels and the corresponding span-wise position.

Further runs were made changing the angle of attack. Hence C_L, C_D and C_M are reported respectively in Figure 6.14 and 6.12, for α = -2, 0, 1.35 and 4 degrees. The lift contribution was accurate but, as expected, drag results are very poor, due to the intrinsic nature of a potential code. The resistance contribution result would be enhanced by using empirical tables to correct viscous steady forces, as outlined in Chapter 5.

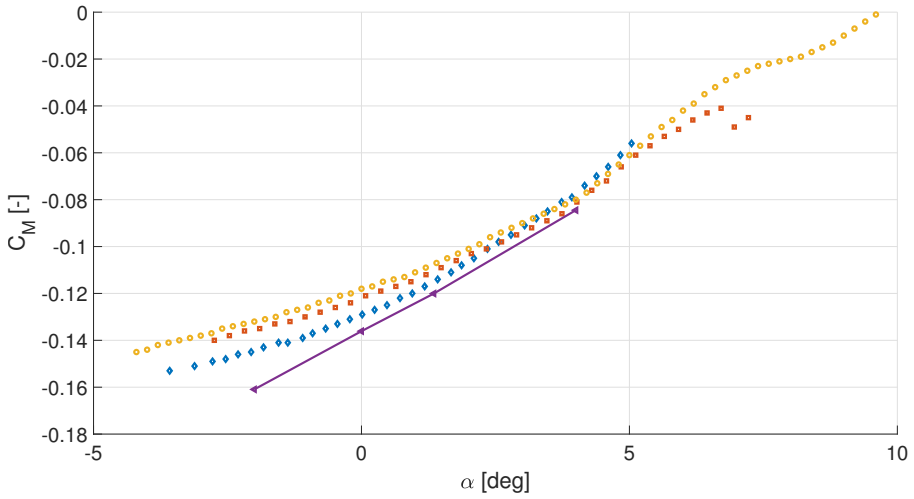

Figure 6.12: C_M coefficients of DLR-F4 as a function of the angle of attack α

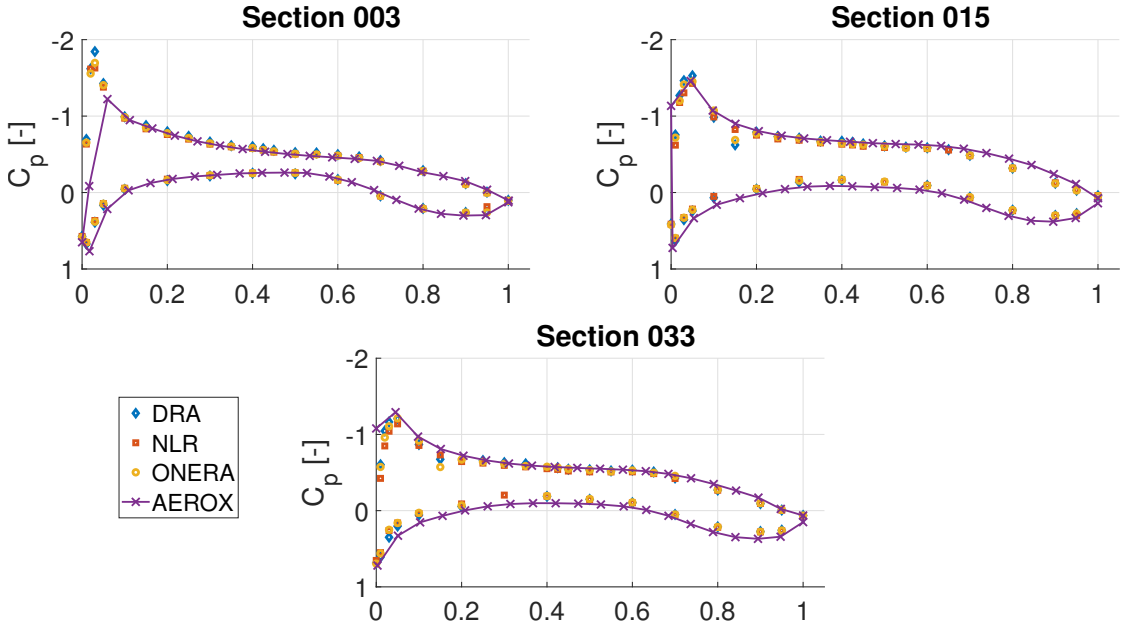

Figure 6.13: C_p cross-sectional distribution for three different wing sections, tabulated in Table 6.4

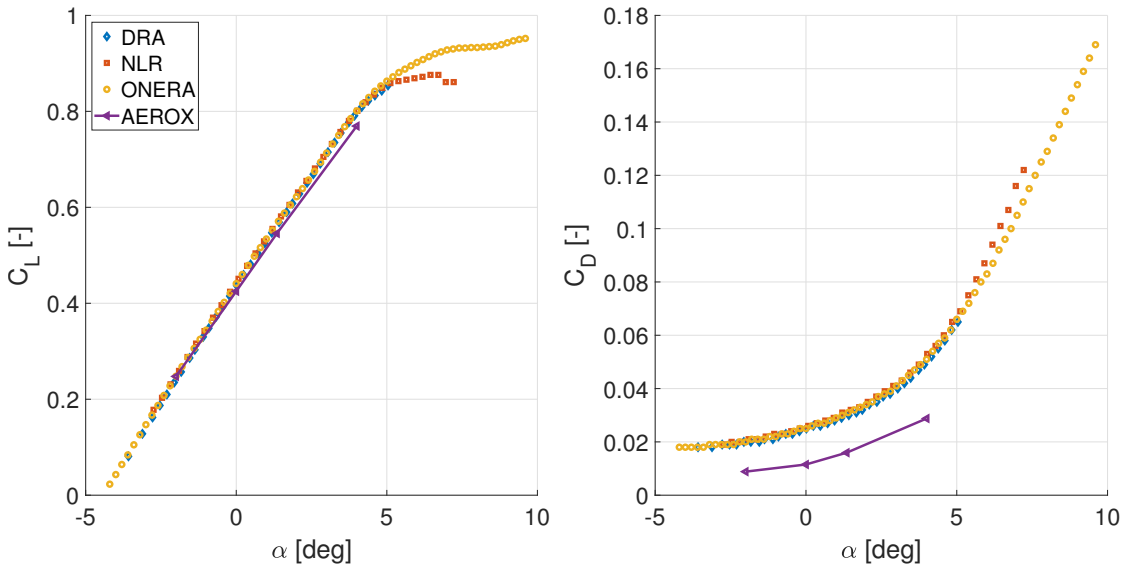

Figure 6.14: C_L and C_D coefficients of DLR-F4 as a function of the angle of attack α

6.2.3 Wind-tunnel hovering rotors

In this section two separated tests are presented.
Both involves the simulation of a rotor in hovering modeled with thick blades, a configuration that sometimes can be critical from the stability standpoint, when dealing with panel methods. As a matter of fact, since the free stream velocity is zero, at the first stages of the computation aerodynamic elements will linger in the vicinity of the blades, and generally speaking this can compromise the results. Indeed, helicopter and tiltrotor aerodynamics is challenging because the flow field generated by the rotor is very complex and unsteady.
For both cases, the wind-tunnel models are not equipped with a swashplate, hence no cyclic control is made on the blades, so a constant collective pitch is applied.

A preliminary mesh sensitivity assessment is made for both cases, varying the number of panels along the chord and along the span of the blade and comparing the thrust coefficient. A good compromise is found with 50 panels along the chord and 25/30 panels along the span. Results with the finest mesh only are shown. Each test was carried on both with the vortex lattice wake and with the particles one. As the vortex lattice wake is concerned, the wake viscous model was tuned to get reliable results. On the other hand, the particle wake did not require any attempt, thanks to the intrinsic viscous term in the equations. The only critical parameter to be set is the particle core vortex radius.
Usually, core vortex radii are set to be on the order of magnitude of 10% of the chord or 1% of the rotor diameter [49]. The latter approach was chosen for both tests.

Rotor configuration R1

The first rotor to be tested corresponds to a wind-tunnel model used by Gibertini et al [21] to assess aerodynamic performances of blades equipped with active control devices, specifically Gurney flaps. Hence, we will compare the four blade rotor model without such devices, in the configuration used as a reference condition.

The blades has a constant 90 mm chord and a constant NACA0012 section, with a 8° linear twist. The results reported in Figure 6.15 were obtained with a panel mesh made with the built-in mesher, using 50 panels along the chord and 25 uniform placed panels along the blade span.

The computation was performed setting the collective pitch angle to 4, 8 and 12 degrees. The rotational speet of the rotor was set to 1600 RPM, which corresponds to $M_{tip} = 0.54$. Results are reported in Figure 6.15, together with the experimental measurements.

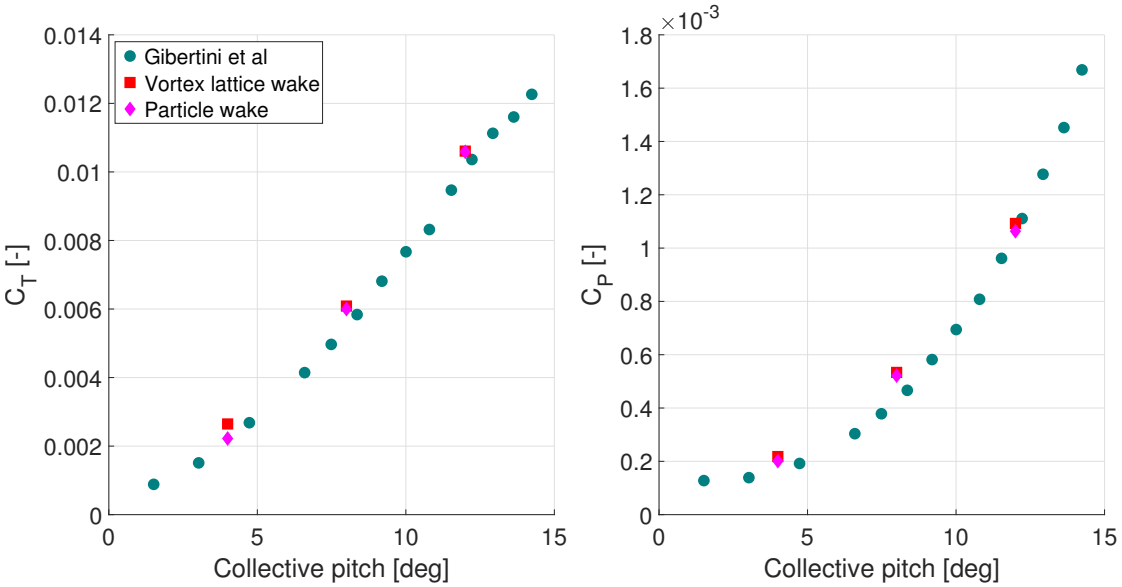

Figure 6.15: Computed C_T and C_P compared with the experimental measurements provided by [21]. Both results obtained with a vortex lattice and a particle wake are shown

Rotor configuration R2

In this section, numerical results will be compared with the experimental campaign described by Droandi in [15] and lead in the Dipartimento di Scienze e Tecnologie Aerospaziali of Politecnico di Milano. In his paper, Droandi presented two rotor blade sets, the first one being a fairly simple helicopter blade while the second one belongs to the innovative typology of high performance tiltwing tiltrotor aircraft.

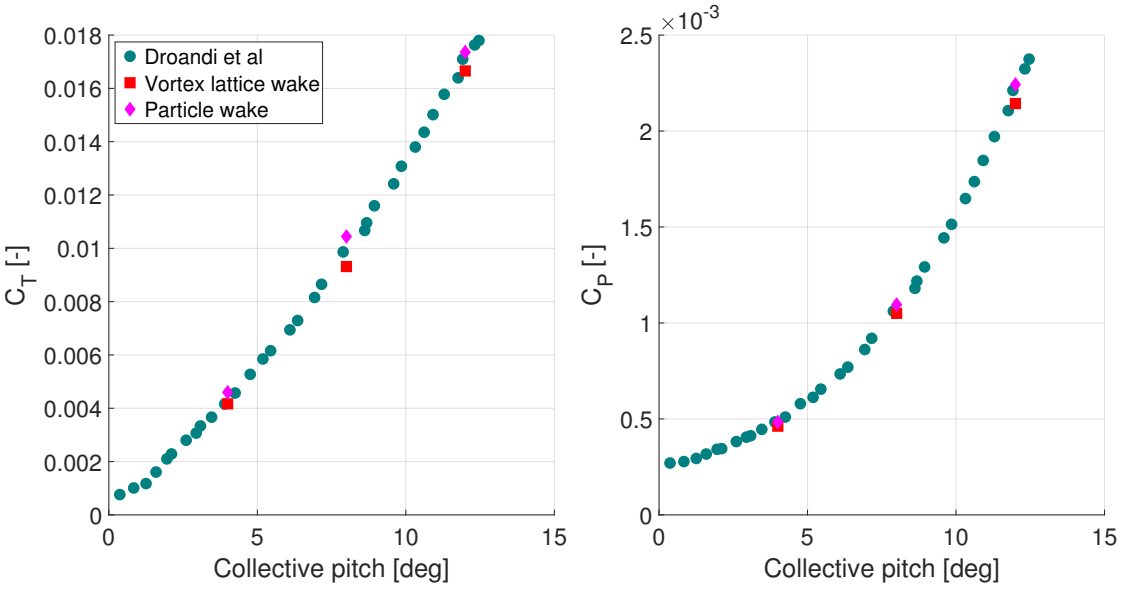

Figure 6.16: Computed C_T and C_P compared with the experimental measurements provided by [15]. Both results obtained with a vortex lattice and a particle wake are shown

The results of the latter configuration will be reported here. The four bladed rotor has swept blades with nonlinear twist and chord distributions; the external rotor radius is 0.925 m (this set-up is called B2 by Droandi). In our simulation the nacelle present in the experimental test rig was not modeled. Moreover, the mach number at the tip wing was $M_{tip} = 0.32$.

Geometrical parameters can be found in details in [15]: the paneled blade was generated with the new built-in mesher. The results reported here were obtained with 50 panels along the chord and 25 panels along the span.

Specifically, due to the complex wing geometry, 12 panels equally distributed along the span were placed from 0 to 75% of the blade, while 13 panels were prescribed from 75% up to the tip of the blade. This meshing strategy has been found to better discretize the circulation on the wing tip, were larger gradients are expected.

The computation was performed for three collective pitch angles, 4, 8 and 12 degrees: C_T and C_P are reported in Figure 6.16 as a function of the collective pitch angle. For each revolution, 50 azimuth angles were discretized , and the

computation was carried on for 8 complete revolutions.
Non-dimensional coefficients reported in Figure 6.16 were obtained as the mean value calculated for the last 150 azimuth angles.

The typical convergence history for this test is reported in Figure 6.17, which shows a relatively fast convergence rate once two complete revolutions were completed.

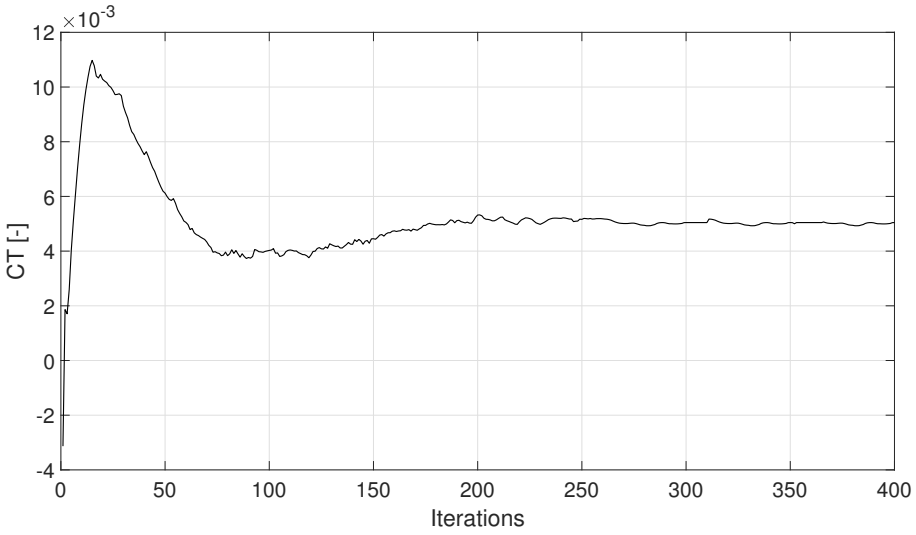

Figure 6.17: CT convergence history - R2, 4° collective pitch

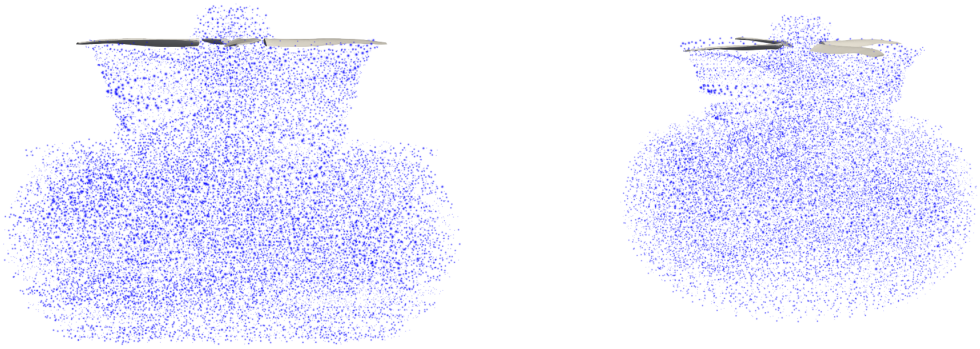

Figure 6.18: 2D projection and isometric view of the particle wake development for the R2 rotor in hover

In Figure 6.18 the particle rotor wake is represented in a graphical render. As we can see, the so called *fountain effect* is clearly visible, even though it is relatively circumscribed.
This phenomenon can occur in hovering rotors and it is a numerical effect that sometimes is related to a relatively low angle of attack in the proximity of the blade root, which results in an upward motion of the aerodynamic elements generated near the rotor hub (due to a reduced suction effect). Indeed, this is not related to particles only, since it can happen also when dealing with low diffusive vortex lattice wakes. The main drawback of the fountain effect is related to the rate of convergence of the numerical scheme. As a matter of fact, under certain circumstances, the starting vortex may roll up on itself without being pushed downwards, while wake elements will linger above the blades, resulting in potentially non-negligible oscillations of integral quantities, leading to a very slow convergence rate or even divergence.

To counter or alleviate the fountain effect, one may prescribe the wake behavior as the rotor completes the first one or two revolutions. To do so, it suffices to overlap the particle velocities with an artificial downward velocity field, whose intensity gradually decays (i.e. through an exponential function).
Alternatively, the *actuator disk theory* can be exploited to formulate an expression for the intensity of the starting velocity field. At each timestep, until one complete revolutions is completed, the velocity field to be added is proportional to the thrust coefficient obtained at the previous timestep, according to the following expression:

$$|\boldsymbol{U}_{ad,s}| = \Omega R \sqrt{\frac{C_T^{s-1}}{2}} \tag{6.10}$$

where Ω is the rotor rotational speed an the subscript 'ad' refers to the actuator disk theory.
However, sometimes those precautions are not enough. In that case one may resort to inserting a paneled hub. If the timestep is small enough, the hub's sources will

prevent particles form escaping the suction region.

Despite being present, the fountain effect did not compromise the results in this test case, and the convergence curve was smooth enough. A possible explanation is related to the particles vorticity intensity decay due to the viscous term. Without the viscous contribution, the induction effect of the particles lingering above the rotor would have been much higher, resulting in sharpened oscillations.

6.3 Ducted fan

The ducted fan, or shrouded propeller concept, was first examined experimentally by Stipa [56] in the 1930s. An overview of most early studies, before 1962, was presented by Sacks et Al. (1962) [52], while reviews on most recent studies can be found in the dissertation of Pereira [50] and Akturk [1], although those research were focused on UAV applications.
Research on large-scale ducted fans for propulsion purposes mostly emerged between the 1950s and 1970s.
The ducted fan is a propeller inside a circular duct. Specifically, the duct is usually considered as an *annular wing* [18]. As confirmed by many early-stage experiments (for instance [24]), the duct can bring superior static performance over its free propeller counterpart given the same power, at least in a specific speed range. The performance gains are mainly attributed to the combined effect of suction on the curved duct inlet lip and larger static pressure around the diffuser outlet. A theoretical analysis for an ideal hover case, carried on with the momentum theory [50], shows that the thrust improvement is proportional to $\sqrt[3]{2E_r}$ were E_r is the duct expansion ratio. Hence, as long as the expansion ratio is greater than 1/2 a performance enhancement could be expected. In goes without saying that as E_r grows, the adverse pressure gradient in the duct also increases, which may lead to flow separation, that would inevitably deteriorate performances.

Recently, Leonardo is developing a numerical study on a specific configuration in partnership with the *University of Glasgow*, that is performing RANS calculations for a specific configuration, studied by Grunwald in the 1960s by means of wind-tunnel experiments [24]. Hence, the simulations performed with AeroX are based upon the same case study, so that a direct comparison between experimental and new CFD results will be made.

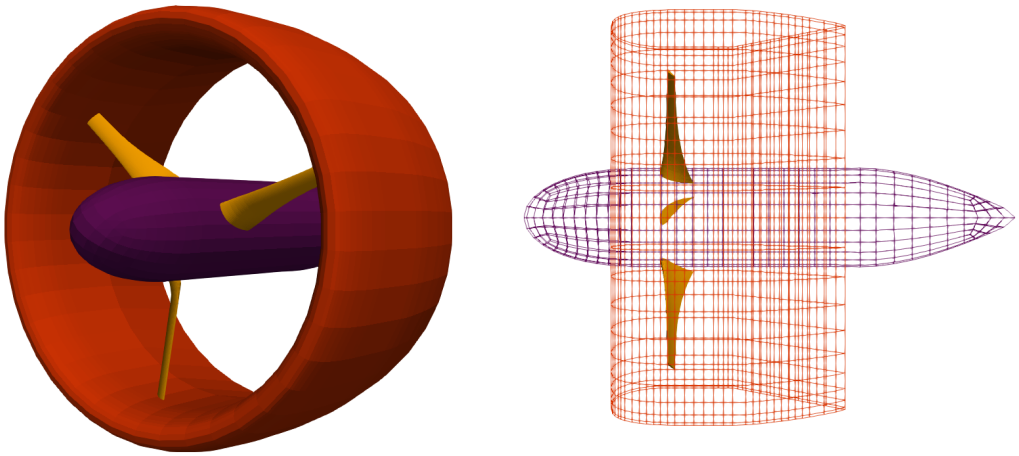

Figure 6.19: Grunwald model AeroX mesh

6.3.1 Axial flight configuration

In this section, the results of a brief investigation on the ducted fan model studied by Grunwald [24] are reported, comparing them with those obtained with the CFD investigation cited above. The numerical tool used to assess the ducted fan performances was HMB3, developed in the University of Glasgow. The Helicopter Multi Block CFD solver is based on the control volume method, and here it performed URANS calculations on the model geometry. Unfortunately, specific details of the turbulence model adopted were not available.

In the original work, the whole powered model (supported by a shielded sting support) was tested with a constant propeller rotational speed of $n = 8000$ rpm, that corresponds to Mach ~ 0.470 on the blade tip. Measurements were taken at various wind tunnel speeds required to cover the desired range of advance ratio $\mu = V/nD$, where V is the free-stream velocity and D is the propeller diameter. At each selected advance ratio, the angle of attack was varied inside a range that could potentially include steady level flight, climb and descent for a tilt-duct VTOL configuration.

The results were nondimensionalized using the free steam dynamic pressure. However, for low advance ratios, data have been nondimensionalized by means of

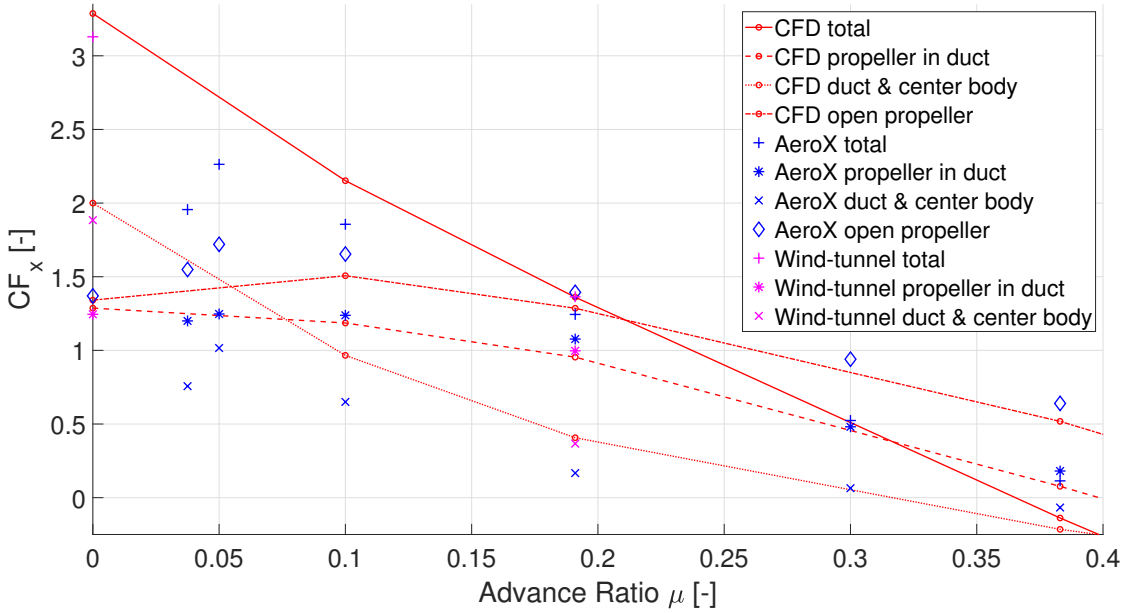

Figure 6.20: Grunwald shoruded propeller results from AeroX, HMB3 and wind-tunnel experiments. AeroX results are obtained with particles wakes

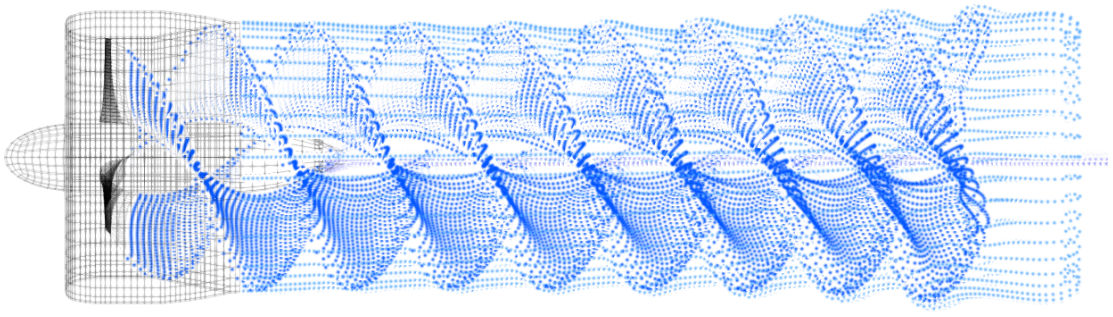

Figure 6.21: Wake development behind the shrouded rotor. Advance ratio $\mu = 0.191$

the static thrust in hovering.

In this paper, results obtained from an axial configuration (i.e. AoA = 0° referring to Grunwald's frame of reference) will be shown and discussed. As a matter of fact, at the time of writing this thesis, integral quantities from CFD applied at high angle of attack configurations were not available. However, calculations performed increasing the angle of attack have shown the need of a rubust *non-interference method* to prevent particles from being convected inside the shroud. As an alternative, the timestep could be further decreased, so that the time integration step would feel the effect of panel sources on the bodies. This latter approach, however, was neither convenient nor feasible, in light of the purposes of this study.

This problem was tackled creating three bodies: the rotor, the shroud and the center body. The whole configuration was tested by means of a moltibody solution (Chapter 5).

Hence, results obtained with AoA = 0 and increasing the advance ratio, are reported in Figure 6.20. The same notation adopted in [24] is used: hence, the CF_x is the longitudinal force coefficient, defined as F_x/qS where in this case F_x corresponds to the thrust force, q is the dynamic pressure and S is the projected shroud area (shroud chord by the exit diameter).

CFD results were obtained only for a limited set of advanced ratios: those points are highlighted in the plot with red circles, and interpolated with solid and dashed lines to capture the trend.
CF_x calculations are made according to three different configuration: the open propeller case, i.e. the isolated fan; the shroud and the center body case; the rotor influenced by the presence of the duct and finally the full multibody case. All those configurations are then labeled with a different marker. Unfortunately, the open propeller experimental measurements were not available.

As the open propeller case is concerned, AeroX results show a good agreement with the CFD computation, although the former slightly overestimates the thrust. However, the blade-angle setting at 75% of the radius was 24°, which may be too

high for a potential solver without specific corrections.
Focusing the region where $\mu \geq 0.05$, more pronounced differences are clearly visible for the duct and the center body case. As a matter of fact, the absolute value of the force coefficient predicted by AeroX is smaller compared to the CFD simulation.
The same configuration was tested also with the vortex lattice wake: by conveniently tuning the vorticity diffusion we managed to replicate the vortex particle wake results, however without getting any closer to the CFD results. Unfortunately, aerodynamic C81 tables of the shroud profile were not available: it might be worthwhile to test the viscous correction on this case.
Examining this point is key, since the difference in the total thrust for the whole model is probably linked to this issue.

As for the low advance ratio region, our implementation gave very poor results, as it is clearly visible form the plot. The isolated propeller converged in hover conditions, but the method was unstable when the whole configuration was considered. Specifically, as soon as the particles reached the exit of the duct, they also started to be summoned back upstream. This event caused eventually a divergence condition.
On the other hand, the vortex lattice wake could converge to a result by setting up a very high viscous coefficient in the wake model. Unfortunately, results obtained in this manner were absolutely unreliable.

As reported by Grunwald, those conditions correspond to a low Reynolds number regime for the duct, possibly causing a separated flow regime. It is possible that a low Reynolds condition on the duct pushes the panel method to its intrinsic limits, leading to instabilities, but further investigations should be carried on.
Hence, this case is further form being clearly understood, therefore more tests should be made. To address this needs, the following steps could be:

- Making the whole method more robust. One way to pursue this may be prescribing the shroud wake interface, preventing it to behave in an unexpected

way.

- Changing the particle releasing strategy. This aspect is covered in the last chapter.

- Setting up a non-penetration algorithm to prevent particles ingestion inside the shroud. The octree hierarchical structure could be used to pursue this goal.

Finally, in Figure 6.21 the wake development behind the model is visualized. In this case, the advance ratio was high enough to provide reasonable results, compared with the CFD survey.

Chapter 7

Future developments

In the previous chapters, the capabilities of the new aerodynamic solver was presented, together with its present limits and weaknesses. Specifically, the vortex particle method convergenge condition could be approached increasing the number of particles in the field, with only a proportional increase of computational effort thanks to the fast summation algorithm.
Future developments will focus on diversified aspects on the implementation.

First of all, the aerodynamic solver will be fully coupled with the structural one, which is currently under development. Hence, the capabilities of the vortex particles method could be tested and exploited in the more complex aeroelastic frame of reference.

Second, the fast multipole scheme could be further improved, acting from the parallel computation standpoint. As a matter of fact, the parallel approach implemented during this thesis work was neither complete nor optimized (as the multipole calculus is concerned). As a matter of fact, the multipole coefficients computation only was parallelized by means of the OpenMP library. Hence, the activity will follow a twofold path.

7.0.1 Parallel enhancement and GPU porting

One one hand, the parallel architecture will be complemented by means of the MPI package, which can be used to speed-up bottle neck processes such as matrix computations and induction operations by means of an hybrid approach with the OpenMP library.

On the other hand, one promising filed of research related with the solution of N-body problems by means of a numerical approach is the exploitation of GPU architectures. In this frame of reference, the steps of the multipole algorithm described in Chapter 3 could be transformed into separated GPU kernels, sending raw data to the chip, while preserving the OOP framework on the CPU. There are many interesting works on this topic in open literature.

For instance, Nyland et al (2007) [46] presented an approach for acceleration of the all-pairs computation on GPUs for the case of gravitational potential of N masses, which can be an interesting starting benchmark. On those basis, Yokota and Barba [62] addressed the more involved task of implementing the $\mathcal{O}(N \log N)$ treecode and the $\mathcal{O}(N)$ fast multipole method. Their results showed that the cross-over point where fast N-body algorithm become advantageous over direct, all-pairs calculations is in the orther of 10^3 for the CPU and in the order of 10^4 for the GPU.

However, about 15x speedup is obtained from the fast algorithm on the GPU for a million particles. For N=10^7 the fast algorithms provide 150x speedup over direct methods on the GPU.

Another interesting contribution is given Burtscher and Pingali (2011) [8], where they described a CUDA implementation of the classical Barnes-Hut [4] N-body algorithm entirely run on a GPU architecture, with an irregular tree-based data structure and six GPU kernels. The CUDA implementation was 74 times faster compared to an optimized (but serial) implementation running on a CPU, when manipulating 5 million bodies. Remaining within the GPU framework, another interesting development is related with octrees creation an and management. Even

though octrees searches are mainly sequential processes and so not suited for GPU implementation, several strategies have been proposed for GPU octree data structures, especially related for optimized search [40].

7.0.2 Particles releasing strategies

In the present implementation, particles can be released only from a vortex lattice interface. That interface could be the last stripe of panels in a vortex lattice wake or it could consist of a wake buffer directly attached to the body. Hence, the vorticity intensity concentrated into particles is the result of the integration of linear vortical edges.

This approach could be a limiting factor when one is trying to solve aerodynamic problems where the releasing point do not consist in a marked line, or when a releasing line locus could not be easily given. In this way, it might be worth assessing the performance of different vortex element introduction methods.

For instance, Kamemoto [31] (1994), proposed a scheme in which the continuous distribution of vorticity on the body surface is replaced with a number of vortex elements and the vortical field is naturally represented by their convective motions. Hence, Nakanishi and Kamemoto [44] applied successfully this scheme to the simulation of the flow around a sphere and showed its applicability to the flow around three-dimensional bodies.

A more recent contribution on this topic is given by Ojima and Kamemoto (2000) [48]. They presented a new scheme for the vorticity creation at a solid surface of a three-dimensional body, in which a number of nascent vortices are introduced in the flow field according to the diffusion and the convection of the vorticity near the solid surface.

Bibliography

[1] Ali Akturk and Cengiz Camci. "Double Ducted Fan (DDF) as a novel ducted fan inlet lip separation control device". English (US). In: *International Powered Lift Conference 2010*. Dec. 2010, pp. 148–170.

[2] Andrew W. Appel. "An Efficient Program for Many-Body Simulation". In: *SIAM Journal on Scientific and Statistical Computing* 6.1 (1985), pp. 85–103.

[3] Jeroen Baert, Ares Lagae, and Philip Dutré. "Out-of-core Construction Of Sparse Voxel Octrees". In: *Proceedings of the 5th High-Performance Graphics Conference*. HPG '13. Anaheim, California: ACM, 2013, pp. 27–32.

[4] Josh Barnes and Piet Hut. "A hierarchical O(N log N) force-calculation algorithm". In: *Nature* 324.6096 (1986), pp. 446–449.

[5] Luke Sterling Battey. "A hybrid Navier Stokes/vortex particle wake methodology for modeling helicopter rotors in forward flight and maneuvers". MA thesis. Georgia Institute of Technology, 2018.

[6] J. Thomas Beale and Andrew Majda. "Vortex Methods. I: Convergence in Three Dimensions". In: *Mathematics of Computation* 39.159 (1982), pp. 1–27.

[7] Richard E. Brown and Andrew J. Line. "Efficient High-Resolution Wake Modeling Using the Vorticity Transport Equation". In: *AIAA Journal* 43.7 (2005), pp. 1434–1443.

[8] Martin Burtscher and Keshav Pingali. "An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-Body Algorithm". In: *GPU Computing Gems Emerald Edition* (Dec. 2011).

[9] Russel E Caflisch. “Mathematical analysis of vortex dynamics”. In: *Mathematical aspects of vortex dynamics* (1988), pp. 1–24.

[10] Jacob Calabretta. “A Three Dimensional Vortex Particle-Panel Code For Modeling Propeller-Airframe Interaction”. MA thesis. California Polytechnic State University, San Luis Obispo, California, June 2010.

[11] Alexandre Joel Chorin and Peter S Bernard. “Discretization of a vortex sheet, with an example of roll-up”. In: *Journal of Computational Physics* 13.3 (1973), pp. 423–429.

[12] Alexandre Joel Chorin et al. “Numerical study of slightly viscous flow”. In: *J. Fluid Mech* 57.4 (1973), pp. 785–796.

[13] G-H Cottet and P D Koumoutsakos. “Vortex Methods: Theory and Practice”. In: *Measurement Science and Technology* 12.3 (2001), pp. 354–354.

[14] Pierre 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.

[15] Giovanni Droandi et al. “Wind-Tunnel Rotor Model for Hover and Forward Flight Tests”. In: Sept. 2013.

[16] Adam Drozdek. *Data structures and algorithms in C++*. Cengage Learning, 2013.

[17] DustGroup. *Dust - Open source repository*. URL: `https://gitlab.com/dust_group/dust`.

[18] Herman S Fletcher. *Experimental investigation of lift, drag, and pitching moment of five annular airfoils*. Tech. rep. National Aeronautics and Space Administration, 1957.

[19] Yuhong Fu et al. “A fast solution method for three-dimensional many-particle problems of linear elasticity”. In: *International Journal for Numerical Methods in Engineering* 42.7 (1998), pp. 1215–1229.

[20] Irene Gargantini. “An effective way to represent quadtrees”. In: *Commun. ACM* 25 (Dec. 1982), pp. 905–910.

[21] Giuseppe Gibertini et al. “Experimental investigation of a helicopter rotor with Gurney flaps”. In: *The Aeronautical Journal* 121.1236 (2017), 191–212.

[22] J.E. Gómez and H. Powert. “A multipole direct and indirect BEM for 2D cavity flow at low Reynolds number”. In: *Engineering Analysis with Boundary Elements* 19.1 (1997). High Performance Computing, pp. 17 –31.

[23] Leslie Greengard and Vladimir Rokhlin. *A fast algorithm for particle simulations.* Tech. rep. Yale University New Haven, 1986.

[24] Kalman J. Grunwald, Kenneth W. Goodson, and United States. *Aerodynamic loads on an isolated shrouded-propeller configuration of angles of attack from -10 degrees to 110 degrees.* NASA technical note. Washington, D.C.: National Aeronautics and Space Administration, 1962, 29 p.

[25] Jaber J. Hasbestan and Inanc Senocak. “Binarized-octree generation for Cartesian adaptive mesh refinement around immersed geometries”. In: *Journal of Computational Physics* 368 (2018), pp. 179 –195.

[26] Chengjian He and Nischint Rajmohan. “Modeling the Aerodynamic Interaction of Multiple Rotor Vehicles and Compound Rotorcraft with Viscous Vortex Particle Method”. In: *AHS 72nd Annual Forum* (May 2016).

[27] Chengjian He and Jinggen Zhao. “Modeling Rotor Wake Dynamics with Viscous Vortex Particle Method”. In: *AIAA Journal* 47.4 (2009), pp. 902–915.

[28] John L Hess et al. “A higher order panel method for three-dimensional potential flow”. In: *7th Australasian Conference on Hydraulics and Fluid Mechanics 1980: Preprints of Papers*. Institution of Engineers, Australia. 1980, p. 517.

[29] Hao Hu et al. “Hybrid Vortex Method for the Aerodynamic Analysis of Wind Turbine”. In: *International Journal of Aerospace Engineering* 2015 (Mar. 2015), pp. 1–8.

[30] Baert Jeroen. *Libmorton: C++ Morton Encoding/Decoding Library.* 2018. URL: `https://github.com/Forceflow/libmorton`.

[31] Kyoji Kamemoto. “Development of the Vortex MEthods for grid-free Lagrangian Direct Numerical Simulation”. In: *JSME/KSME Fluids engineering Conference* (1994), pp. 542–547.

[32] Yukio Kaneda. "A representation of the motion of a vortex sheet in a three-dimensional flow". In: *Physics of Fluids A: Fluid Dynamics* 2.3 (1990), pp. 458–461.

[33] Joseph Katz and Allen Plotkin. *Low-Speed Aerodynamics*. 2nd ed. Cambridge Aerospace Series. Cambridge University Press, 2001.

[34] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., 1998.

[35] Horace Lamb. *Hydrodynamics*. Cambridge University Press, 1932.

[36] D Levy et al. "Summary of data from the first AIAA CFD drag prediction workshop". In: *40th AIAA Aerospace Sciences Meeting & Exhibit*. 2002, p. 841.

[37] Keith Lindsay. "A three-dimensional Cartesian tree-code and applications to vortex sheet roll-up". PhD thesis. University of Michigan, Jan. 1997.

[38] Keith Lindsay and Robert Krasny. "A Particle Method and Adaptive Treecode for Vortex Sheet Motion in Three-Dimensional Flow". In: *Journal of Computational Physics* 172.2 (2001), pp. 879 –907.

[39] Yijun Liu. *Fast Multipole Boundary Element Method: Theory and Applications in Engineering*. Cambridge University Press, 2009.

[40] Daniel Madeira et al. "GPU Octrees and Optimized Search". In: 2011.

[41] Luigi Morino and Ching-Chiang Kuo. "Subsonic Potential Aerodynamics for Complex Configurations : A General Theory". In: *AIAA Journal* 12.2 (1974), pp. 191–197.

[42] Guy Morton. "A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing". In: *Technical Report* (Jan. 1966).

[43] H Multhopp. *Methods for calculating the lift distribution of wings (subsonic lifting-surface theory)*. Aeronautical Research Council London, 1950.

[44] Y Nakanishi and K Kamemoto. "Numerical simulation of flow around a sphere with vortex blobs". In: *Computational Wind Engineering 1*. Elsevier, 1993, pp. 363–369.

[45] N Nishimura. "Fast multipole accelerated boundary integral equation methods". In: *Applied Mechanics Reviews* 55 (July 2002), p. 299.

[46] Lars Nyland, M. Harris, and Jan Prins. “Fast N-body simulation with CUDA”. In: *GPU Gem, Vol. 3* (Jan. 2009), pp. 677–695.

[47] Akira Ojima and Kyoji Kamemoto. “Numerical Simulation of Unsteady Flow around Three Dimensional Bluff Bodies by an Advanced Vortex Method”. In: *JSME International Journal Series B* 43.2 (2000), pp. 127–135.

[48] Akira Ojima and Kyoji Kamemoto. “Numerical simulation of unsteady flow around three dimensional bluff bodies by an advanced vortex method”. In: *JSME International Journal Series B Fluids and Thermal Engineering* 43.2 (2000), pp. 127–135.

[49] Raghuveera Padakannaya. “The vortex lattice method for the rotor-vortex interaction problem”. In: (1974).

[50] Jason L Pereira. “Hover and wind-tunnel testing of shrouded rotors for improved micro air vehicle design”. PhD thesis. University of Maryland, College Park, 2008.

[51] Pierre-Arnaud Raviart. “An analysis of particle methods”. In: *Numerical methods in fluid dynamics*. Springer, 1985, pp. 243–324.

[52] AH Sacks and JA Burnell. “Ducted propellers—a critical review of the state of the art”. In: *Progress in Aeronautical sciences*. Vol. 3. Elsevier, 1962, pp. 85–135.

[53] H. Samet. “The Design and Analysis of Spatial Data Structure”. In: *Addison Wesley, 1990* 50255 (Jan. 1990).

[54] Günther Schrack. “Finding Neighbors of Equal Size in Linear Quadtrees and Octrees in Constant Time”. In: *CVGIP: Image Underst.* 55.3 (May 1991), 221–230.

[55] 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 (Jan. 2016), pp. 386–404.

[56] L Stipa. *Experiments with intubed propellers*. Tech. rep. National Aeronautics and Space Administration, 1932.

[57] Leo Stocco and Guenther Schrack. “On Spatial Orders and Location Codes”. In: *IEEE Transactions on Computers* 58 (Jan. 2008), pp. 424–432.

[58] Spyros Voutsinas. “Vortex methods in aeronautics: How to make things work”. In: *International Journal of Computational Fluid Dynamics* 20 (Jan. 2006), pp. 3–18.

[59] Daehyun Wee et al. “Convergence Characteristics and Computational Cost of Two Algebraic Kernels in Vortex Methods with a Tree-Code Algorithm”. In: *SIAM J. Scientific Computing* 31 (Jan. 2009), pp. 2510–2527.

[60] G S Winckelmans and A Leonard. “Contributions to vortex particle methods for the computation of three-dimensional incompressible unsteady flows”. In: *Journal of Computational Physics; (United States)* 109:2 (Dec. 1993). ISSN: 0021-9991.

[61] Gregoire Stephane Winckelmans. “Topics in vortex methods for the computation of three- and two-dimensional incompressible unsteady flows”. PhD thesis. California Institute of Technology, 1989.

[62] Rio Yokota and Lorena Barba. “Treecode and Fast Multipole Method for N-Body Simulation with CUDA”. In: *GPU Computing Gems Emerald Edition* (Oct. 2010).

[63] Feng Zhao. *An O(N) Algorithm for Three-dimensional N-body Simulations*. Tech. rep. Massachusetts Institute of Technology, 1987.

[64] Jinggen Zhao and Chengjian He. “A Finite State Dynamic Wake Model Enhanced with Vortex Particle Method–Derived Modeling Parameters for Coaxial Rotor Simulation and Analysis”. In: *Journal of the American Helicopter Society* 61 (Apr. 2016).