UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Parallel Algorithms for Fluid and Rigid Body Interaction

Cristóbal Samaniego Alvarado

Advisor: Guillaume Houzeaux
Co-advisor: Mariano Vázquez

Thesis submitted for the degree of Doctor of Philosophy

Universitat Politècnica de Catalunya
Barcelona, España

November 2015

Acknowledgements

As a personal matter, I prefer to write the acknowledgements in Spanish, my mother tongue.

Quiero empezar por agradecer a mi director de tesis, a Guillaume Houzeaux. Soy muy afortunado al haberle tenido como director. No solo por que es una persona inteligente sino también amable y que disfruta de su trabajo. Gracias por tu paciencia y dedicación.

También quiero agradecer al BSC (Barcelona Supercomputing Center), al departamento del CASE, por haber confiado en mi trabajo. Especialmente a Mariano Vázquez. Fue él quien me conoció primero y en Ecuador. Él vió que podía ayudarles de alguna manera y confió en mí. Nunca lo olvidaré.

A mis compañeros de trabajo y de oficina. Ninguno de ellos, me ha negado jamás ayuda.

A Paola, mi compañera de viaje y de vida. Sin ella, todo esto hubiera sido mucho menos agradable. Sé que todos los logros que he hecho, ella los ha hecho suyos. Me halaga que se sienta tan orgullosa de mí, es una de las razones que más me ha empujado a terminar este trabajo. Yo también me siento muy orgullaso de todo lo que ha logrado y está a punto de lograr. Espero que esto a ella tamién le sirva como me ha servido a mí.

A mi familia, a mis padres y hermanos. Ellos me llevaron hasta aquí. Me apoyaron y me siguieron como si fueran ellos los que estuvieran haciendo este trabajo. A ellos es a los que más extraño ahora que estamos en diferenrtes países y continentes. Esteban, además, estuvo conmigo ayudándome en mi tesis y mis publicaciones sin importar el día, la fecha o la hora. Sabía que podía contar con él siempre. Sé que Augusto, Haydeé o Pedro hubieran hecho lo mismo si su campo de trabajo hubiera sido parecido al mío.

A mis amigos, a los que se fueron ya. Con Natalia, Juan Carlos y Oscar pasamos muy buenos tiempos en Barcelona. A los amigos de acá, de Cataluña. Ellos me han tratado como uno más. Gracias Cristina, Laia, Jordi e Iván. El día que me vaya, voy a extrañar mucho las noches de comida, bebida, de conversaciones, de muy buenas conversaciones.

Summary

This thesis is based on the implementation of a computational system to numerically simulate the interaction between a fluid and an arbitrary number of rigid bodies. This implementation was performed in a distributed memory parallelization context, which makes the process and its description especially challenging. As a consequence, for the sake of descriptive precision and conceptual clarity, a new formal framework using set theory concepts is developed.

The fluid is discretized using a non body-conforming mesh and the boundaries of the bodies are embedded in this mesh. The force that the fluid exerts on a body is determined from the residual of the momentum equations. Conversely, the velocity of the body is imposed as a boundary condition in the fluid. In this context, two new approaches are proposed.

To account for the fact that fluid nodes can become solid nodes and vice versa due to the rigid body movement, we have adopted the FMALE approach, which is based on the idea of a virtual movement of the fluid mesh at each time step. A new method of interpolation is adopted inside the FMALE implementation in order to improve the results.

The physics of the fluid is described by the incompressible Navier-Stokes equations. These equations are stabilized using a variational multiscale finite element method and solved using a fractional step like scheme at the algebraic level. The incompressible Navier-Stokes solver is a parallel solver based on master-worker strategy.

The bodies can have arbitrary shapes and their motions are determined by the Newton-Euler equations. The contacts between bodies are solved using impulses to avoid interpenetrations. The time of impact is determined implementing a dynamic collision detection algorithm. As far as the parallel implementation is concerned, the data of all the bodies are shared by all the subdomains. To track the boundary of the bodies in the fluid mesh, computational geometry tools have been used.

List of publications

- C. Samaniego, G. Houzeaux, E. Samaniego, M. Vázquez, Parallel embedded boundary methods for fluid and rigid-body interaction, Computer Methods in Applied Mechanics and Engineering 290 (2015) 387–419
- E. Casoni, A. Jérusalem, C. Samaniego, B. Eguzkitza, P. Lafortune, D. Tjahjanto, X. Sáez, G. Houzeaux, M. Vázquez, Alya: computational solid mechanics for supercomputers, Archives of Computational Methods in Engineering (2014) 1–20
- H. Owen, G. Houzeaux, C. Samaniego, A. Lesage, M. Vázquez, Recent ship hydrodynamics developments in the parallel two-fluid flow solver alya, Computers & Fluids 80 (2013) 168–177
- G. Houzeaux, H. Owen, B. Eguzkitza, C. Samaniego, R. de la Cruz, H. Calmet, M. Vázquez, M. Ávila, Developments in Parallel, Distributed, Grid and Cloud Computing for Engineering, Vol. volume 31 of Computational Science, Engineering and Technology Series, Saxe-Coburg Publications, 2013, Ch. Chapter 8: A Parallel Incompressible Navier-Stokes Solver: Implementation Issues, pp. 171–201
- H. Owen, G. Houzeaux, C. Samaniego, F. Cucchietti, G. Marin, C. Tripiana, H. Calmet, M. Vázquez, Two fluids level set: High performance simulation and post processing, in: 2012 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC), IEEE, Salt Palace Convention Center, Salt Lake City, UT, 2012, pp. 1559–1568
- G. Houzeaux, C. Samaniego, H. Calmet, R. Aubry, M. Vázquez, P. Rem, Simulation of magnetic fluid applied to plastic sorting, The Open Waste Management Journal 3 (2010) 127–138

Contents

List of Figures

List of Algorithms

1
Introduction

The numerical simulation of the interaction of a fluid and a rigid body in the context of high performance computing is a challenging subject. Efficiency is tightly interlinked with a careful implementation. In this thesis we try to elucidate the data structures and the algorithms that lead to an efficient simulation tool for supercomputers by means of formal definitions, thereby generating a general framework. The implementation of two embedded boundary methods are described within this framework. They are implemented inside the Alya system [3], a parallel multiphysics code. Finally, several numerical examples are used to demonstrate the accuracy and the computational efficiency of the implemented methods.

1.1 Motivation

The detailed modeling of the interaction of a rigid solid with a fluid has been the object of intensive research [8, 9, 10, 11]. However, this is still a challenging subject that entails several difficulties. The problem can become even harder when a high performance computing implementation is sought.

There exist different methods to simulate the interaction between the fluid and a solid in movement. We are mainly interested in techniques developed within the context of the Finite Element Method here. However, it is important to mention other alternatives like those based on Lattice-Boltzman [12] and meshless methods [13, 14, 15].

To put our work into context, the main approaches based on the Finite Element Method are described below and schematized in Figure 1.1. This list is based on the review presented in [9].

- Domain decomposition methods [16]. Due to the actual process followed in this class of methods for fluid-structure interaction, maybe a more appropriate name is domain composition methods as pointed out in [17]. A fluid mesh attached to the body is moving over a fixed fluid mesh. As a consequence, the information between adjacent meshes or subdomains has to be exchanged to obtain a global solution. Several instances of this approach can be mentioned. The Chimera method [18, 19], and HERMESH [20], are examples of partially overlapping domain decomposition as illustrated in Figure 1.1(Top)(Left). The sliding mesh method [21] is another example of domain decomposition; here the subdomains are disjoint and information between them is transmitted across the interfaces, see Figure 1.1(Top)(Right). In the shear-slip mesh update method (SSMUM) [8], a layer of shear-absorbing elements is used to connect a

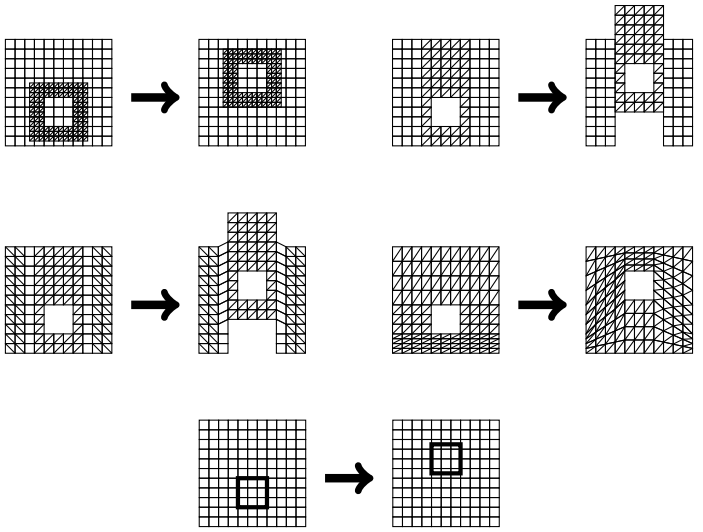

Figure 1.1: Illustration of some methods to simulate flows around moving components. (Top) (Left) Chimera method. (Top) (Right) Sliding mesh method. (Mid.) (Left) SSMUM. (Mid.) (Right) ALE method (Bot.) Embedded boundary mesh.

moving, associated to the body, and non-moving region as illustrated in Figure 1.1(Mid.)(Left).

- The ALE method. The Arbitrary Lagrangian-Eulerian description (ALE) method takes advantage of the features of both (Lagrangian and Eulerian) descriptions to move the fluid mesh in order to adapt it to the changing solid configuration [22]. Figure 1.1(Mid.)(Right) illustrates the movement of the mesh around a body in an ALE implementation. Remeshing is required when the elements in the discretization are too distorted.

- Embedded boundary methods. The fluid is discretized using a non body-conforming mesh and described in an Eulerian frame of reference. The wet boundaries of the bodies are embedded in this mesh and geometrically tracked by means of moving polyhedral surface meshes, see Figure 1.1(Bot.) Examples of this approach are the Immersed Boundary (IB) method [23] and the Fictitious Domain (FD) [24, 25]. Another example relevant to this work is the strategy proposed by Löhner et al. [26], which imposes the velocity of the body directly as a Dirichlet boundary condition on the fluid. There exist other alternatives such as the work developed in [27] that combines concepts from embedded boundary methods and the isogeometric analysis introduced in [28].

- Monolithic approach. A unified formulation is used for both the solid and

fluid. Interaction is taken into account by means of an extra stress tensor appearing in the Navier-Stokes equations [10].

Within this context, the two new schemes proposed in this work can be characterized as based on the embedded boundary concept. They both manage an internal boundary in the fluid domain at each time step to track the solid wet boundary.

The selection of the strategies has been motivated by the search of a computationally efficient parallel implementation. We decided to avoid connecting different meshes, because it implies changing the nodes connectivities, thereby increasing parallel communications and the complexity of the algorithms. Alternatives that can cause severe distortions in some elements were also avoided. In order to tackle these distortions, re-meshing can be used, but this would entail the need of changing nodes connectivities, which would require redistributing the computational load in the mesh partitions. That is why we avoid changes in the topology of the mesh in both of the proposed approaches.

To account for the fact that fluid nodes can become solid nodes and vice versa due to the rigid body movement, we have adopted the FMALE approach [29, 30]. A new interpolation method is adopted inside the FMALE implementation in order to improve the results. Also, to track the wet boundary of the body, computational geometry tools have been used. In general, the two new approaches, in order to be both computationally efficient and accurate, entail the integration of different algorithmic solutions.

In addition, in a simulation of a dynamic rigid body system multiple problems have to be solved. First, the motion of bodies due to the external forces must be determined. Next, when the bodies are in movement, it is necessary to prevent interpenetration between them and to solve the collisions when the bodies are in contact. The simulation framework of dynamic rigid bodies is well-known, see [31, 32], and tries to solve the problems mentioned above in the following consecutive stages:

- Collision Detection.
- Rigid Body Motion.
- Collision Response.

The previous paragraphs can give the reader a hint of the intrinsic complexity associated to obtaining an efficient parallel implementation of the interaction of a fluid and a rigid body. This complexity is reflected in the difficulty of giving an accurate explanation of such implementations. This is why the need of generating a framework that allows for a precise description was felt. A very interesting attempt to create such a framework for the modeling of incompressible flows can be found in [33, 34]. However, in the author's opinion, a new framework better suited for fluid-structure interaction (FSI) was necessary. Thus, a new formal characterization of the data structures needed in a distributed memory environment in terms of set theory concepts is introduced. It must

be said that the parallel framework, although mainly thought for FSI, can be generalized to other applications. In [2], some elements of this framework were used to explain a parallel solver for solid mechanics.

1.2 Objectives

The aim of this thesis is to numerically simulate the interaction of a fluid and a number of rigid bodies considering a distributed memory environment. To achieve this goal, we have to accomplish the objectives mentioned below.

In order to have a precise description of the parallel algorithms to solve the interaction:

- To develop a general framework for the parallel implementation of the interaction between a fluid and the rigid bodies by means of a new formal definition using the set notation. This general framework is intended to elucidate the data structures and algorithms involved in a precise fashion. The main formal definitions are detailed in Chapter 2.

In order to numerically solve the interaction inside the embedded boundary mesh framework:

- To propose two new strategies to accurately solve the interaction of a fluid and a number of rigid bodies inside the embedded boundary mesh framework considering a distributed memory parallelization environment. The description is detailed in Subsection 6.2.2. The validation of both approaches is described in Subsection 7.1.1.

- To adopt a new interpolation method inside the FMALE framework in order to account for the fact that fluid nodes can become solid nodes and vice versa due to the rigid body movement. The FMALE framework is explained in Subsection 6.2.3. The new method of interpolation is studied in Subsection 7.1.2.

- To solve the interactions between the bodies. As all the subdomains simulate the interaction of all the bodies and redundant work is done, the implementation has to be done in such way that each subdomain solves these interactions as fast as possible. The theory is described in Chapter 5. Some examples are shown in Section 7.2.

Finally, in order to implement the interaction to solve real problems:

- To select numerical strategies motivated by the search of a computationally efficient parallel implementation.

1.3 Limitations

We do not know the positions of the bodies inside the mesh that discretizes the problem a priori. Thus, in general, the discretization of a problem entails a fine mesh in order to obtain results that are good enough.

The mesh has to become finer as the Reynolds number increases. To solve turbulent flows, the required mesh could imply a considerable growth in the number of degrees of freedom and alternative numerical methods, that include numerical strategies to simulate flows with high Reynolds numbers, can render better solutions for this kind of problems with coarser meshes. Remeshing can be used, but, as mentioned above in Section 1.1, this would require redistributing the computational load in the mesh partitions.

For all these reasons, in this thesis, the analyses will be focused on laminar and transition flows. In particular, flows with Reynolds numbers until nearly 6000. The discretization of the problems will use meshes of until nearly 30 million elements. Even so, the sizes of the meshes and the time of simulation require a distributed memory environment to solve the problems considered in this work. In this context, our main goal is not to affect the scalability of the Alya system. That is, not to affect the scalability of the fluid solver. An analysis of the scalability of the implementation for the proposed new strategies is described in Subsection 7.1.5.

1.4 Outline of the thesis

The rest of this thesis is organized as follows. Chapter 2 is devoted to explaining the mesh topology structures considering a parallel context. Also, the algorithms to exchange the data structures associated to this mesh are explained inside a parallel finite element and a parallel finite difference implementations. The physics and numerical aspects to solve a fluid and a rigid body are described respectively in Chapters 3 and 4. The general framework of interaction between rigid bodies is explained in Chapter 5. The Chapter 6 describes in detail a general algorithm to solve the interaction between a fluid and a rigid body. It is important to remark that all the algorithms derived from the general algorithm are described considering a parallel implementation and using the algorithms of exchange explained in Chapter 2.

The numerical examples are presented in Chapter 7 in order to validate the methods. Finally, the conclusions of this work are presented in Chapter 8.

2
Parallel context

In a parallel finite element program, the original mesh is partitioned into subdomains. The data that has a direct relationship with the set of nodes of the mesh will be also divided. As a consequence, the data between adjacent subdomains has to be exchanged to preserve the coherency of the data and to obtain the correct solution to the problem.

In order to be precise and avoid ambiguities, some sets are defined to represent the original mesh, first, in a serial context, and then, in a parallel context. To illustrate the concepts, a simple one-dimensional example will be considered.

Then, a formal description of the algorithms to exchange data in a finite element or a finite difference parallel program will be described. A simple iteration of an iterative solver will be considered in order to motivate the definition of the algorithms.

2.1 Finite Element Serial Context

In the context of the finite element method, the continuous domain is discretized into a set of elements $\mathcal{E} = \{e_1, e_2, e_3, ...\}$ and a set of nodes $\mathcal{N} = \{n_1, n_2, n_3, ...\}$. Each node $n \in \mathcal{N}$ is defined by its position inside the domain. And each element $e \in \mathcal{E}$ is defined, for our purposes, by a subset of the set of nodes $e = \{n_1^e, n_2^e, n_3^e, ...\} \subset \mathcal{N}$.

Mesh connectivities

The definition of an element as a subset of nodes relates any node $n \in \mathcal{N}$ with other nodes and elements of the mesh. These relations are called the connectivity of node n and can be characterized by the following definitions:

- ***Element connectivity of*** n. Let $\mathcal{C}_{ele}(n)$ denote the set of elements in $\mathcal{E}$ directly connected to the node n, the gray squares in Figure 2.1. Formally,

$$\mathcal{C}_{ele}(n) = \{e \in \mathcal{E} : n \in e\}.$$

- ***Node connectivity of*** n. Let $\mathcal{C}_{nod}(n)$ denote the set of nodes in $\mathcal{N}$ directly connected to n, the black circles in Figure 2.1. Formally,

$$\mathcal{C}_{nod}(n) = \{m \in \mathcal{N} : \exists e \in \mathcal{C}_{ele}(n), m \in e\} \setminus \{n\}.$$

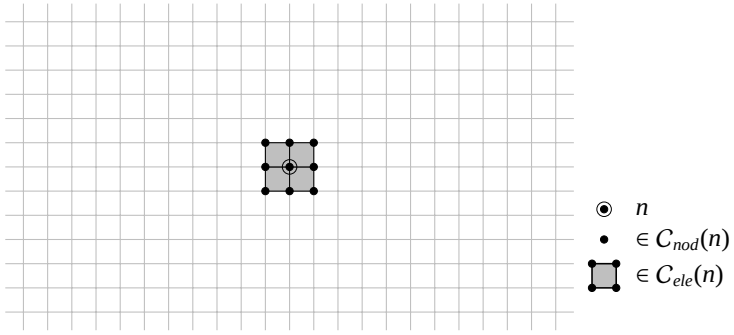

Figure 2.1: Node connectivity.

2.2 Finite Element Parallel Context

In the parallel context of the finite element method, the original mesh is partitioned into subdomains. Each subdomain is defined by subsets of the set of elements $\mathcal{E}$ and the set of nodes $\mathcal{N}$. Let $\mathcal{N}^S$ and $\mathcal{E}^S$ denote the set of nodes and elements of an arbitrary subdomain S respectively. Then, the nodes and elements of the mesh can be grouped by subdomains fulfilling

$$\mathcal{N} = \bigcup_{I=1}^{P} \mathcal{N}^I \text{ and } \mathcal{E} = \bigcup_{I=1}^{P} \mathcal{E}^I,$$

where P is the number of subdomains.

The partition of the mesh is done such that in any subdomain S,

$$\mathcal{N}^S \bigcap \left(\bigcup_{I=1, I\neq S}^{P} \mathcal{N}^I \right) \neq \emptyset$$

and

$$\mathcal{E}^S \bigcap \left(\bigcup_{I=1, I\neq S}^{P} \mathcal{E}^I \right) = \emptyset,$$

i.e., nodes can be shared between subdomains, whereas elements cannot.

The shared nodes are located at the interface between subdomains created by the partition of the mesh. This partition allow us to divide the set of nodes $\mathcal{N}^S$ into two disjoint subsets defined as

- ***The set of interior nodes of*** S. Let

$$\mathcal{N}^S_{int} = \mathcal{N}^S \backslash \left(\bigcup_{I=1, I\neq S}^{P} \mathcal{N}^I \right)$$

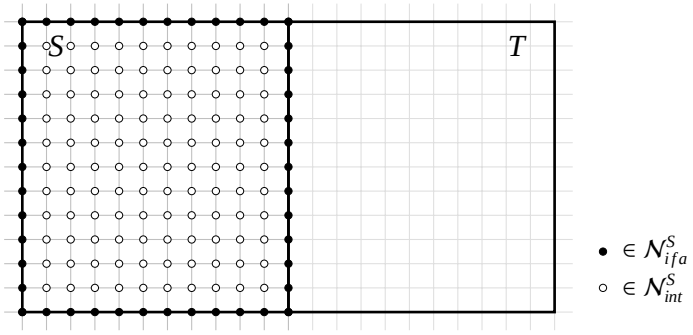

Figure 2.2: Interface and interior nodes of the subdomain S.

denote the set of interior nodes of the subdomain S. These nodes do not belong to the interface; see Figure 2.2, where white circles denote the interior nodes of S.

- ***The set of interface nodes of*** S. Let $\mathcal{N}^S_{ifa} = \mathcal{N}^S \backslash \mathcal{N}^S_{int}$ denote the set of interface nodes of S. These nodes belong to the interface and are shared by different subdomains, including S; see Figure 2.2, where black circles denote the interface nodes of S.

Two arbitrary subdomains S and T that share at least one node at the interface are called as ***adjacent subdomains***, i.e $\mathcal{N}^S_{ifa} \cap \mathcal{N}^T_{ifa} \neq \emptyset$. Consider the partition shown in Figure 2.2. In this particular example, the subdomains S and T are adjacent because they share a set of interface nodes.

Let us define a useful subset of the interface nodes $\mathcal{N}^S_{ifa}$ that will be used in most of the parallel algorithms for fluid and rigid body interaction describe in this thesis:

- ***The set of own interface nodes of*** S. Let $\mathcal{N}^S_{ifa,own}$ denote the own interface nodes of a subdomain S. These *own* nodes are uniquely associated to a subdomain in order to manage communications properly when performing certain operations. The definition of the set of own interface nodes of S states that:

$$\mathcal{N}^S_{ifa,own} \cap \bigcup_{I \neq S, I \text{ is adjacent to } S} \mathcal{N}^I_{ifa,own} = \emptyset.$$

That is, an own interface node of S cannot be own by another subdomain different from S.

Parallel mesh connectivity

In this context, consider a node n in an arbitrary subdomain S that is located at the interface. From the point of view of subdomain S, there are two disjoint sets whose union defines the whole node connectivity of n:

- ***Node connectivity of*** n ***in*** S. Let the set

$$\mathcal{C}_{nod}^{S}(n) = \mathcal{C}_{nod}(n) \cap \mathcal{N}^{S}$$

denote the set of nodes in $\mathcal{N}^S$ directly connected to the node n.

- ***Node connectivity of*** n ***in other subdomains***. Let the set

$$\mathcal{C}_{nod}^{\hat{S}}(n) = \mathcal{C}_{nod}(n) \backslash \mathcal{C}_{nod}^{S}(n)$$

denote the set of nodes in subdomains different from S directly connected to the node n. These nodes will be referred to as *halo nodes of* S, see Section 2.4.

In a similar way, there are two disjoint sets whose union defines the whole element connectivity of n:

- ***Element connectivity of*** n ***in*** S. Let the set

$$\mathcal{C}_{ele}^{S}(n) = \mathcal{C}_{ele}(n) \cap \mathcal{E}^{S}$$

denote the set of elements in $\mathcal{E}^S$ directly connected to the node n.

- ***Element connectivity of*** n ***in other subdomains***. Let the set

$$\mathcal{C}_{ele}^{\hat{S}}(n) = \mathcal{C}_{ele}(n) \backslash \mathcal{C}_{ele}^{S}(n)$$

denote the set of element in subdomains different from S directly connected to the node n. These elements will be referred to as *halo elements of* S, see Section 2.4.

In Figure 2.3, the whole connectivity of the interface node n is divided between the adjacent subdomains S and T.

2.3 Finite Element and Finite Difference Parallel Exchange

In a distributed memory context, a typical parallel implementation of the finite element (FE) method differs from a typical parallel implementation of the finite difference (FD) or the finite volume (FV) method. The difference stems from the way these methods assemble the algebraic systems resulting from the discretizations. On the one hand, in a finite difference code (similarly in a FV code), each process is responsible for a given set of rows of the matrix. In order to complete each row, a subdomain is defined by a subset of the set of nodes of the original mesh and by the set of edges that are directly connected with this

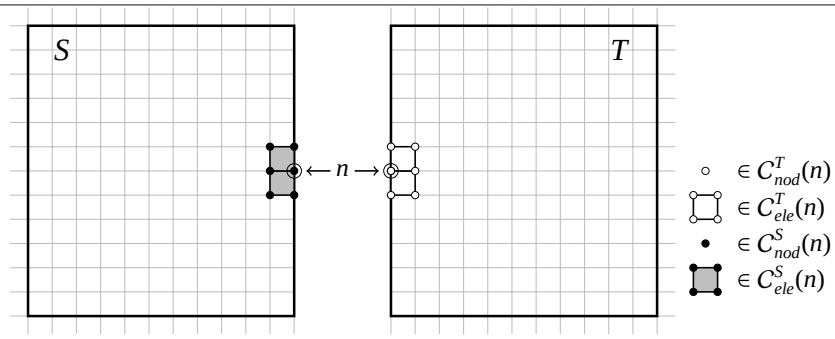

Figure 2.3: Node connectivity in a parallel context.

subset of nodes. Thus, the edges located at the interface between subdomains (cells in a FV code) are duplicated, resulting in an overlap of edges (cells), see Figure 2.4. On the other hand, in a finite element code, a subdomain is defined by a subset of the set of elements of the original mesh and by the set of nodes that belongs to this subset of elements, see also Figure 2.4. Only the nodes located at the interface between subdomains are duplicated and on these nodes, the matrix is assembled locally and only partly on each subdomain. To illustrate this fact, let us take a very simple one-dimensional example. Figure 2.4 shows the partition of the mesh into two subdomains, S and T. In the case of the FD method, edge $n_3 - n_4$ is duplicated. Subdomain S is responsible for the rows of nodes n_1,n_2 and n_3 while subdomain T takes care of nodes n_4 and n_5. In the case of the finite element method, no element is duplicated. But both subdomains will partly be responsible for node n_3. Now let us examine how the parallelization works.

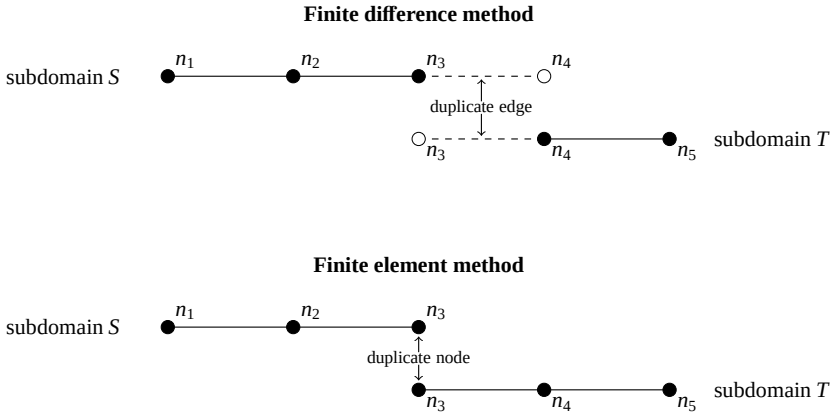

Figure 2.4: Mesh partition for FD and FE.

The numerical solution of a PDE (and consequently the Navier-Stokes equations) consists mainly of two steps. First, the construction of the matrix $\mathbf{A}$ and right-hand side (RHS) $\mathbf{b}$ of the algebraic system $\mathbf{Ax} = \mathbf{b}$. Second, the solution of this system using an iterative solver. As far as the matrix and RHS assemblies are concerned, in the case of the FD and FV methods, each subdomain is able to construct complete rows and RHS thanks to the duplicated edges (cells in a FV code). In the case of the finite element method, only part of the matrix is assembled for the interface nodes. As far as iterative solvers are concerned, the basic operation is the matrix-vector product. Let us consider the matrix-product $\mathbf{y} = \mathbf{Ax}$ and examine the parallelization of this product for the FD and FE methods; see Figure 2.5.

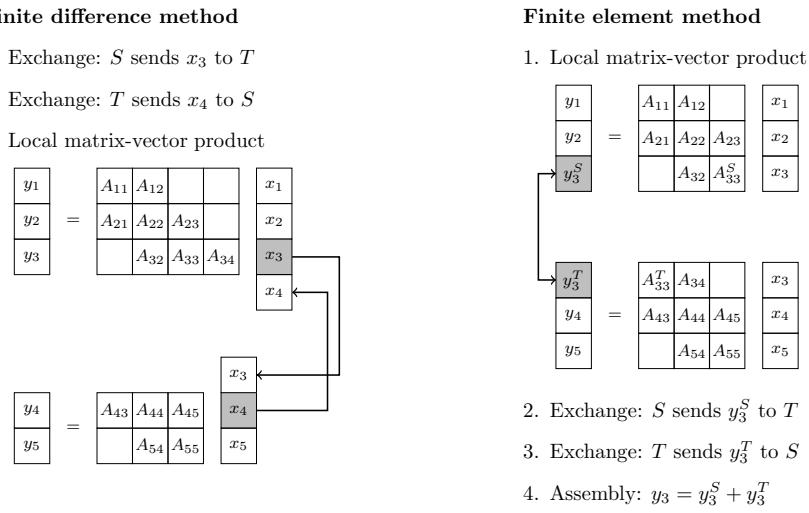

Figure 2.5: Parallel matrix-vector product for FD and FE.

In the FD case, on the one hand, subdomain S is in charge of the whole row of node n_3. Thanks to the duplication of edge $n_3 - n_4$, coefficients A_{33} and A_{34} are complete. On the other hand, subdomain T is in charge of the whole row of node n_4. As before, thanks to the duplication of edge $n_3 - n_4$, coefficients A_{43} and A_{44} are complete. The matrix-vector product can be carried out in parallel as follows:

1. Exchange the data x_3 and x_4 between the subdomains S and T.
2. Perform local matrix-vector product.

In the case of the FE, the coefficients of the matrix come from element integrals. Subdomain S can therefore provide only part of coefficient A_{33}, namely A_{33}^S, while subdomain T provides A_{33}^T. Note that

$$y_3 = A_{32}x_2 + A_{33}x_3 + A_{34}x_4$$

can be rewritten as

$$\begin{aligned} y_3 &= (A_{32}x_2 + A^S_{33}x_3) + (A^T_{33}x_3 + A_{34}x_4) \\ &= y^S_3 + y^T_3. \end{aligned}$$

Then, the matrix-vector product can be carried out in parallel as follows:

1. Perform local matrix-vector product.
2. Exchange the results on the interface node n_3: y^S_3 and y^T_3.
3. Assemble (sum) the local contribution: $y_3 = y^S_3 + y^T_3$.

Considering any arbitrary subdomain Q, the important fact of interest for us is that a priori, the parallelization of a finite element code requires only the local data related with the set of nodes that belongs to Q, the set $\mathcal{N}^Q$.

However, as we will see in further sections, the coupling of the rigid-body and the Navier-Stokes solvers requires that an arbitrary subdomain Q can access the data related with the whole node connectivity of the set of nodes that belongs to Q, the set $\bigcup_{n\in\mathcal{N}^Q} \mathcal{C}_{nod}(n)$, which includes the data related with nodes that belong to other adjacent subdomains. In particular, we need to include data of the interior nodes of all the adjacent subdomains of Q that are directly connected to its interface nodes $\mathcal{N}^Q_{ifa}$. These nodes are the set of halo nodes of Q and are formally defined in Section 2.4. Considering the problem shown in Figure 2.5, the sets $\{n_4\}$ and $\{n_2\}$ are the sets of halo nodes of S and T respectively. Note that the data related with these sets of nodes are already included in their respective subdomains when we are working in the context of the finite difference or finite volume method, see Figure 2.4.

2.4 Halo nodes and Halo elements

From the section 2.2, we can easily deduce that the number of nodes and elements directly connected to an interface node n in a subdomain is smaller than in the original mesh, see Figures 2.1 and 2.3. This lack of topological information can seriously affect the ability of the algorithms that perform the coupling of the rigid-body and the Navier-Stokes solvers (RB-NS coupling) to reach the right results.

As mentioned in Section 2.3 and considering the example shown in Figure 2.4, this means that subdomain S needs to access the data related with node n_4 and subdomain T needs to access the data related with node n_3, the set of halo nodes of S and T respectively. This data is not only geometrical and topological but can also consists of values of some variables.

In a finite element parallel program, we can consider two options in order to implement the RB-NS coupling:

- Implicit implementation. Include the geometrical and topological data related with the halo nodes changing the structure of the local matrices.

In this case, the implementation have to enable rectangular matrices like in the case of the FD method in order to implicit the relation with the halo nodes.

- Explicit implementation. Include the geometrical and topological data related with the halo nodes without changing the structure of the local matrices. In this case, we lose in convergence as the values of the variables related with the halo nodes have to go to the RHS.

In our code, we choose the explicit implementation option in order to preserve the structure of the local matrices. Some geometrical and topological data is added in the subdomain definitions in order to have the same connectivity as in the original mesh for any interface node.

From the point of view of an arbitrary subdomain S, the formal definitions of these new added sets of nodes and elements are given by:

- ***Set of halo nodes of*** S. Let the set

$$\mathcal{N}_{hal}^{S} = \bigcup_{n \in \mathcal{N}_{ifa}^{S}} \mathcal{C}_{nod}^{\hat{S}}(n)$$

  denote the set of halo nodes in S.

- ***Set of halo elements of*** S. Let the set

$$\mathcal{E}_{hal}^{S} = \bigcup_{n \in \mathcal{N}_{ifa}^{S}} \mathcal{C}_{ele}^{\hat{S}}(n)$$

  denote the set of halo elements in S.

Consider again the connectivity of the interface node n in Figure 2.3. Now, if we include the halo nodes and halo elements of the subdomain S, as shown in Figure 2.6, the interface node n in Figure 2.3 or any other interface node in the subdomain S, will have defined its whole connectivity inside S.

2.5 Parallel exchange algorithms

In a finite element program, the most important data structures have a direct relationship with the set of nodes of the mesh. These structures are collections of numerical values, each one identified by an index (or a tuple of indices). In a parallel context, these data structures have to be exchanged between subdomains to preserve the coherency of the data.

In parallel, for any subdomain S, a node in $\mathcal{N}^{S} \cup \mathcal{N}_{hal}^{S}$ is related to its index by:

$$\begin{aligned} index^{S} : \mathcal{N}^{S} \cup \mathcal{N}_{hal}^{S} &\rightarrow \{1, 2, 3, ... | \mathcal{N}^{S} \cup \mathcal{N}_{hal}^{S} |\} \\ n &\mapsto i^{S}. \end{aligned}$$

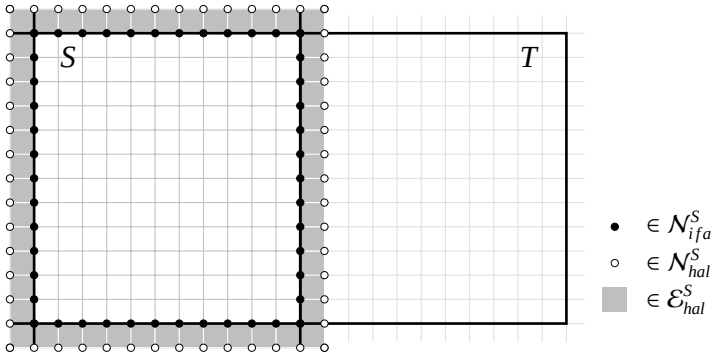

Figure 2.6: Halo nodes and halo element of subdomain S.

For implementation aspects, a subdomain S enumerates consecutively its interior nodes, next its own interface nodes, the rest of its interface nodes, and finally its halo nodes, see Figure 2.7. Thus, any numerical data array *data* of length $|\mathcal{N}^S \cup \mathcal{N}^S_{hal}|$ can be conveniently splitted in four consecutive arrays: $data(1 : |\mathcal{N}^S_{int}|)$, the values related with the interior nodes of S, $data(|\mathcal{N}^S_{int}| + 1 : |\mathcal{N}^S_{int} \cup \mathcal{N}^S_{ifa,own}|)$, the values related with the own interface nodes of S, $data(|\mathcal{N}^S_{int} \cup \mathcal{N}^S_{ifa,own}| + 1 : |\mathcal{N}^S|)$, the values related with the interface nodes that do not own S, and $data(|\mathcal{N}^S| + 1 : |\mathcal{N}^S \cup \mathcal{N}^S_{hal}|)$, the values related with the halo nodes of S. Also, these divisions facilitate the definition of the algorithms written above which allow us to exchange data between subdomains.

2.5.1 Interface node exchange algorithm (INE)

Consider an arbitrary subdomain S. Then, for each adjacent subdomain T of S, the algorithm carries out the exchange of values associated with the subset of interface nodes $\mathcal{N}^S \cap \mathcal{N}^T$. For this purpose, the algorithm needs a common index in S and T as defined below:

$$\begin{aligned} index^{S,T}_{ifa} : \mathcal{N}^S \cap \mathcal{N}^T &\rightarrow \{1, 2, 3, ...|\mathcal{N}^S \cap \mathcal{N}^T|\} \\ n &\mapsto i^{S,T}_{ifa}. \end{aligned}$$

The exchange of data is described in Algorithm 1. Considering the two adjacent subdomains S and T shown in Figure 2.8, this exchange involves the data related with the black nodes shown in Figure 2.8 and can be schematized as illustrated in Figure 2.9.

From the point of view of an arbitrary node $n \in \mathcal{N}^S_{int}$, the Algorithm 1 works as explained next. Let the contributions of a variable x evaluated at node n furnished by S and all its adjacent subdomains A^1, A^2, ..., A^N that share n; that is $n \in \mathcal{N}^{A^1}$, $n \in \mathcal{N}^{A^2}$, ..., $n \in \mathcal{N}^{A^N}$; be denoted by x_S and $x_1, x_2, ..., x_N$ respectively. The Algorithm 1, first exchanges the values $x_1, x_2, ..., x_N$ and x_S

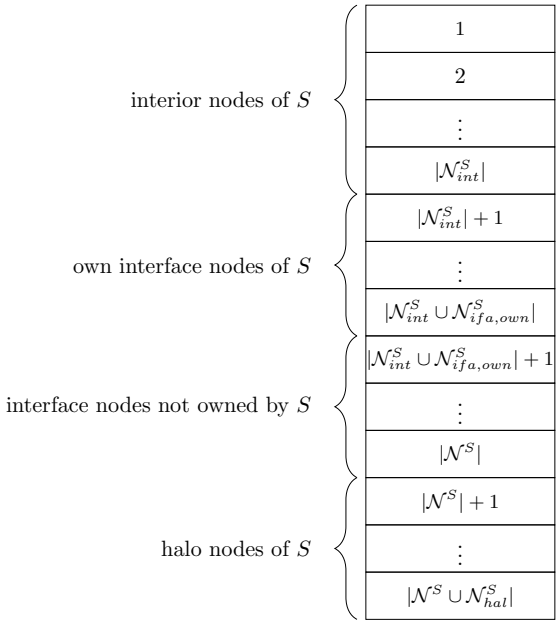

Figure 2.7: Array of data related with the set of nodes of S.

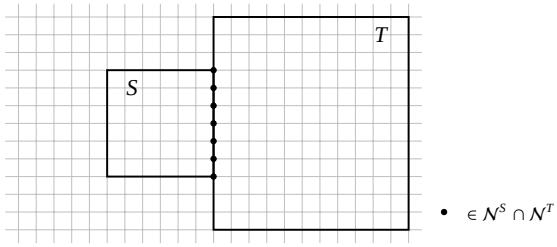

Figure 2.8: Adjacent subdomains S and T.

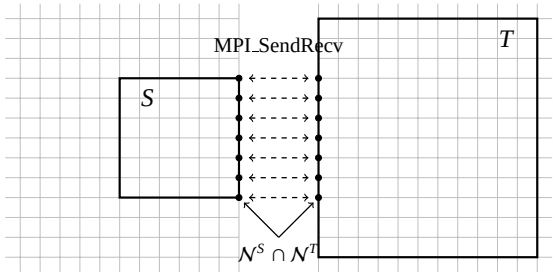

Figure 2.9: Interface nodes parallel exchange.

between the subdomains A^1, A^2, ..., A^N and S. Then, the Algorithm 1 adds the contribution coming from the subdomains A^1, A^2, ..., A^N to get a new value associated to n in S equal to $x_S + \sum_{I=1}^{N} x_I$.

Algorithm 1 Parallel exchange algorithm **INE** for an arbitrary subdomain S

Require: A numeric array *data* with length $|\mathcal{N}^S|$
Ensure: A modified array *data*

▷ Construct sending data arrays
for each adjacent subdomain T of S **do**
  for each node $n \in \mathcal{N}^S \cap \mathcal{N}^T$ **do**
    $i^S \leftarrow index^S(n)$
    $i^{S,T}_{ifa} \leftarrow index^{S,T}(n)$
    Construct the array $data\_send^T(i^{S,T}_{ifa}) \leftarrow data(i^S)$
  end for
end for

▷ Send and receive data arrays
for each adjacent subdomain T of S **do**
  Using MPI_SendRecv, send $data\_send^T$ to T and receive $data\_receive^T$ from T
end for

▷ Assembly
for each adjacent subdomain T of S **do**
  for each node $n \in \mathcal{N}^S \cap \mathcal{N}^T$ **do**
    $i^S \leftarrow index^S(n)$
    $i^{S,T}_{ifa} \leftarrow index^{S,T}(n)$
    $data(i^S) \leftarrow data(i^S) + data\_receive^T(i^{S,T}_{ifa})$
  end for
end for

This algorithm is commonly used in parallel finite element programs to perform the matrix-vector operation during the execution of iterative solvers as illustrated in Section 2.3. In this work, the idea is to reuse this code for the algorithms that perform the fluid and the rigid body interaction.

2.5.2 Halo node exchange algorithm (HNE)

Consider an arbitrary subdomain S. Then, for each adjacent subdomains T of S, the algorithm carries out the exchange of values associated to the subset of halo nodes of S: $\mathcal{N}^S_{hal} \cap \mathcal{N}^T$, and to the subset of halo nodes of T: $\mathcal{N}^S \cap \mathcal{N}^T_{hal}$.

In this case, the algorithm needs two numerical data arrays as common indices instead of only one for S and T: an array to send data to T, see Figure

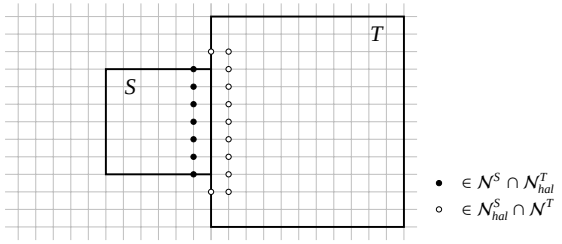

Figure 2.10: Adjacent subdomains S and T.

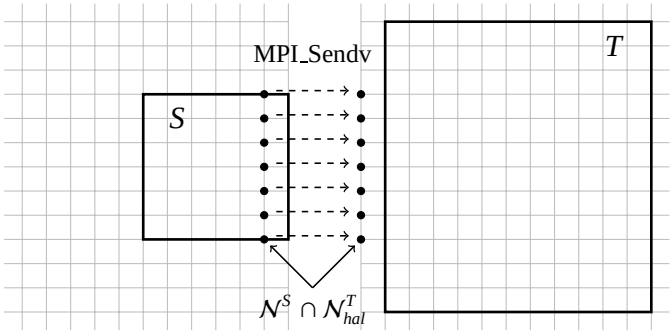

Figure 2.11: Halo nodes parallel exchange. Send data from S to T.

2.11, and another one to receive data from T, see Figure 2.12. From S to T:

$$\begin{aligned} index^{S,T}_{hal} : \mathcal{N}^S \cap \mathcal{N}^T_{hal} &\rightarrow \{1,2,3,...|\mathcal{N}^S \cap \mathcal{N}^T_{hal}|\} \\ n &\mapsto i^{S,T}. \end{aligned}$$

From T to S:

$$\begin{aligned} index^{T,S}_{hal} : \mathcal{N}^T \cap \mathcal{N}^S_{hal} &\rightarrow \{1,2,3,...|\mathcal{N}^T \cap \mathcal{N}^S_{hal}|\} \\ n &\mapsto i^{T,S}. \end{aligned}$$

The exchange is described in Algorithm 2. Considering the two adjacent subdomains S and T shown in Figure 2.10, the exchange of data involves the data related with the black and white nodes shown in Figure 2.10 and can be schematized as illustrated in Figures 2.11 and 2.12. In Figure 2.11 the data is sent from S to T and in Figure 2.12 the data is sent from T to S.

From the point of view of an arbitrary node $n \in \mathcal{N}^S_{hal}$ that is shared with an adjacent subdomain T of S, that is $n \in \mathcal{N}^T$, the Algorithm 2 works as explained next. Let the values associated to n be x_S and x_T for S and T respectively. The Algorithm 2, first, sends the value x_T from T to S and, then, replaces the value associated to n in S to get a new value $x_S = x_T$.

Actually, the relationships between a subset of nodes and a common index for a pair of adjacent subdomains defined above are slightly different in the

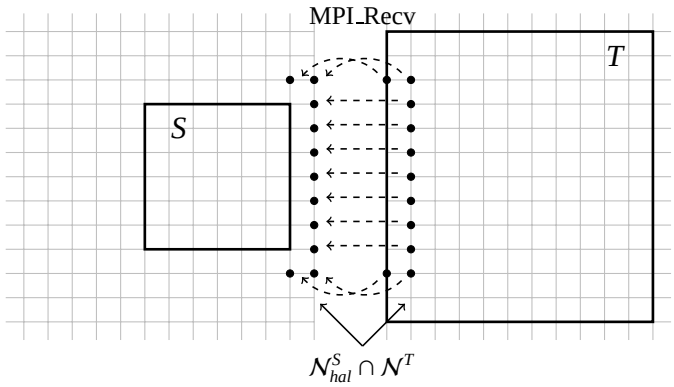

Figure 2.12: Halo nodes parallel exchange. Receive data from T in S.

Algorithm 2 Parallel exchange algorithm **HNE** for an arbitrary subdomain S

Require: A numeric array *data* with length $|\mathcal{N}^S \cup \mathcal{N}^S_{hal}|$
Ensure: A modified array *data*

▷ Construct sending data arrays

for each adjacent subdomain T of S **do**
  for each node $n \in \mathcal{N}^S \cap \mathcal{N}^T_{hal}$ **do**
    $i^S \leftarrow index^S(n)$
    $i^{S,T}_{hal} \leftarrow index^{S,T}_{hal}(n)$
    Construct the array $data\_send^T(i^{S,T}_{hal}) \leftarrow data(i^S)$
  end for
end for

▷ Send and receive data arrays

for each adjacent subdomain T of S **do**
  Using MPI_Send, send $data\_send^T$ to T
  Using MPI_Recv, receive $data\_receive^T$ from T
end for

▷ Data substitution

for each adjacent subdomain T of S **do**
  for each node $n \in \mathcal{N}^T \cap \mathcal{N}^S_{hal}$ **do**
    $i^S \leftarrow index^S(n)$
    $i^{T,S}_{hal} \leftarrow index^{T,S}(n)$
    $data(i^S) \leftarrow data\_receive^T(i^{T,S}_{hal})$
  end for
end for

implementation level. The idea is to avoid to send or to receive redundant data. Thus, the value of a node $n \in \mathcal{N}^S_{hal}$ shared for the adjacent subdomains of S: A^1, A^2, ..., A^N; that is $n \in \mathcal{N}^{A^1}$, $n \in \mathcal{N}^{A^2}$, ..., $n \in \mathcal{N}^{A^N}$; will be sent only for the adjacent subdomain $\mathcal{N}^{A^I}$, where $1 \leq I \geq N$, with the smaller identifier value.

2.5.3 Parallel matrix-vector and dot product

To describe some characteristics of the iterative methods for solving linear systems in a parallel context, consider a simple iteration of an Orthomin(1) method:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha \left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^k \right),$$

where k is the iteration index, $\alpha =< \boldsymbol{r}^k, \boldsymbol{A}\boldsymbol{r}^k > / < \boldsymbol{A}\boldsymbol{r}^k, \boldsymbol{A}\boldsymbol{r}^k >$, and $\boldsymbol{r}^k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^k$.

It is clear that in this simple iteration, there are two matrix-vector products and two dot products operations involved. In a parallel context, these operations require the exchange of data between subdomains. In order to be precise in the implementation, let us formally defined the algorithms to solve a parallel matrix-vector and a parallel dot product operations.

Parallel matrix-vector product

The current implementation of the matrix-vector product uses synchronous communications. Formally, the operation is written in Algorithm 3. Note that each subdomain has to call the parallel exchange algorithm **INE** defined in section 2.5.1 after calculating its local matrix-vector product.

Algorithm 3 The parallel matrix-vector product

```
for each subdomain S do
    for each n ∈ N^S do
        i = index^S(n)
        Initialize y^S(i) = 0
        for each m ∈ {n} ∪ C^S_nod(n) do
            j = index^S(m)
            Construct y^S(i) = y^S(i) + A^S(i,j) * x^S(j)
        end for
    end for
    call INE(y^S)
end for
```

Parallel dot product

The current implementation of a parallel dot product is formally defined in Algorithm 4. At the end, each subdomain has to call the MPI_AllReduce subroutine

after calculating its local dot product.

Algorithm 4 The parallel dot product

for each subdomain S **do**
  Initialize $\alpha = 0$
  for each $n \in \mathcal{N}_{int}^{S} \cup \mathcal{N}_{ifa,own}^{S}$ **do**
    $i = index^{S}(n)$
    $\alpha = \alpha + x^{S}(i) * y^{S}(i)$
  end for
  MPI_AllReduce of α
end for

It is necessary to ensure that only one subdomain calculates α for any arbitrary interface node. For this reason, and as shown in Algorithm 4, any arbitrary subdomain S will take into account only its set of interior nodes N_{int}^{S} and its set of own interface nodes $\mathcal{N}_{ifa,own}^{S}$ defined in Chapter 2.

3
Fluid

This chapter introduces the mathematical and numerical models for a transient and incompressible fluid flow considering the coupling with a rigid solid. In particular, the fluid is described by the Navier-Stokes equations and approximated using the finite element method. The coupling of the fluid with a rigid solid is taken into account by imposing the velocity of the solid surface as a Dirichlet boundary condition in the Navier-Stokes equations.

The discretization of the Navier-Stokes equations will lead to a velocity and pressure coupled algebraic system. The solvers used to find a solution of this algebraic system are described at the end of this chapter.

3.1 The Navier-Stokes equations

The physics of the fluid is described by the incompressible Navier-Stokes equations. Let μ be the viscosity of the fluid, and ρ its density. Let $\boldsymbol{\varepsilon}$ and $\boldsymbol{\sigma}$ be the velocity rate of deformation and the stress tensors respectively, defined as:

$$\begin{aligned} \boldsymbol{\varepsilon}(\boldsymbol{u}) &= \frac{1}{2}\left(\nabla\boldsymbol{u} + \nabla\boldsymbol{u}^t\right) \text{ and} \\ \boldsymbol{\sigma} &= -p\boldsymbol{I} + 2\mu\boldsymbol{\varepsilon}(\boldsymbol{u}). \end{aligned}$$

The problem is stated as follows. Find the velocity $\boldsymbol{u}$ and mechanical pressure p in a domain Ω such that they satisfy in a time interval $(0, T]$:

$$\rho\frac{\partial\boldsymbol{u}}{\partial t} + \rho[(\boldsymbol{u} - \boldsymbol{u}_{\text{msh}})\cdot\nabla]\boldsymbol{u} - \nabla\cdot[2\mu\boldsymbol{\varepsilon}(\boldsymbol{u})] + \nabla p = \rho\boldsymbol{f} \qquad \text{in } \Omega\times(0,T] \quad (3.1)$$

$$\text{and } \nabla\cdot\boldsymbol{u} = 0 \qquad \text{in } \Omega\times(0,T] \quad (3.2)$$

together with initial and boundary conditions.

In the momentum equations, $\boldsymbol{u}_{\text{msh}}$ is the velocity of the fluid particles, which basically enables one to go locally from an Eulerian ($\boldsymbol{u}_{\text{msh}} = \boldsymbol{0}$) to a Lagrangian ($\boldsymbol{u}_{\text{msh}} = \boldsymbol{u}$) description of the fluid motion. The boundary conditions considered in this work are:

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{u}_D \qquad \text{on } \Gamma_D\times(0,T], \\ \boldsymbol{u} &= \boldsymbol{u}_S \qquad \text{on } \Gamma_S\times(0,T], \text{ and} \\ \boldsymbol{\sigma}\cdot\boldsymbol{n} &= \boldsymbol{t} \qquad \text{on } \Gamma_N\times(0,T], \end{aligned}$$

where Γ_D, Γ_S and Γ_N are the boundaries of Ω where Dirichlet, rigid body Dirichlet and Neumann boundary conditions are prescribed respectively, and $\partial\Omega = \overline{\Gamma_D \cup \Gamma_S \cup \Gamma_N}$. Note that the wet boundary of the solid Γ_S, and the associated prescribed solid surface velocity $\boldsymbol{u}_S$ will change in time. They are respectively the boundary and the variable used in the coupling with the rigid body.

In general, in an embedded boundary method, the fluid is discretized using a non body-conforming mesh and described in an Eulerian frame of reference. However, the Navier-Stokes Equations (3.1) and (3.2) are expressed in an Arbitrary Lagrangian-Eulerian (ALE) frame of reference. The reason has to do with the fact that there is a set of nodes in the fluid mesh at the current time step of the simulation that were part of the solid mesh at the previous time step. Then, the undetermined values of the velocities in the fluid for this set of nodes at the previous time step can be obtained considering a hidden movement of the mesh with velocity $\boldsymbol{u}_{\text{msh}}$. This framework is known as the Fixed Mesh ALE (FMALE) method and will be deeply explained in Section 6.2.3.

Now, for sake of simplicity in the numerical description, let us rewrite the Navier-Stokes Equations (3.1) and (3.2) in a more compact form. Then, considering $\boldsymbol{U} := [\boldsymbol{u}, p]^T$, we can define the differential operator $\mathcal{L}(\boldsymbol{U})$ and the force term $\boldsymbol{F}$ as

$$\mathcal{L}(\boldsymbol{U}) := \begin{bmatrix} \rho[(\boldsymbol{u} - \boldsymbol{u}_{\text{msh}}) \cdot \nabla]\boldsymbol{u} - \nabla \cdot [2\mu\boldsymbol{\varepsilon}(\boldsymbol{u})] + \nabla p \\ \nabla \cdot \boldsymbol{u} \end{bmatrix} \text{ and} \tag{3.3}$$

$$\boldsymbol{F} := \begin{bmatrix} \rho\boldsymbol{f} \\ 0 \end{bmatrix}.$$

By introducing also the matrix $\boldsymbol{M} = \text{diag}(\rho\boldsymbol{I}_d, 0)$, where $\boldsymbol{I}_d$ is the identity tensor, the compact form of the incompressible Navier-Stokes equation reads:

$$\boldsymbol{M}\partial_t\boldsymbol{U} + \mathcal{L}(\boldsymbol{U}) = \boldsymbol{F}.$$

3.2 Numerical treatment

The numerical solution of the incompressible Navier-Stokes was implemented inside the Alya system, a parallel computational mechanics code developed at the Barcelona Supercomputing Center (BSC-CNS). The Alya system uses the finite element method as a general tool to find a numerical solution of partial differential equations. In particular and in order to solve an incompressible fluid, the Alya system uses a stabilized finite element method.

3.2.1 Stabilization

The stabilization is based on the Variational MultiScale (VMS) method, see [35]. The formulation is obtained by splitting the unknowns into grid scale

and a subgrid scale components, $\boldsymbol{U} = \boldsymbol{U}_h + \tilde{\boldsymbol{U}}$. This method has been introduced in 1995 and sets a remarkable mathematical basis for understanding and developing stabilization methods [36]. The general form of this stabilization is

$$\text{Galerkin} + \text{Stabilization} = 0.$$

Let $\boldsymbol{V}$ be the test function vector including the velocity and pressure test functions, $\boldsymbol{v}$ and q, respectively, such that $\boldsymbol{V} := [\boldsymbol{v}, q]^T$. Then, the stabilization based on the VMS framework reads:

$$\text{Stabilization} = (\partial_t(\rho\tilde{\boldsymbol{u}}), \boldsymbol{v}) + (\tilde{\boldsymbol{U}}, \mathcal{L}^*(\boldsymbol{V})).$$

For the sake of clarity, subscript h is removed.

3.2.2 Subgrid scale modeling

In addition to the scale splitting technique, the subgrid scale must be modeled. Define the residual $\mathcal{R}$ of the Navier-Stokes system such that $\mathcal{R}(\boldsymbol{U}) = \boldsymbol{F} - \boldsymbol{M}\partial_t\boldsymbol{U} - \mathcal{L}(\boldsymbol{U})$. Then, the expression

$$\tilde{\boldsymbol{U}} = \boldsymbol{\tau}\mathcal{R}(\boldsymbol{U})$$

is considered for the ASGS stabilization, where $\boldsymbol{\tau}$ is approximated as a diagonal matrix $\boldsymbol{\tau} = (\boldsymbol{I}_d\tau_1, \tau_2)$, where τ_1 is the algebraic approximate of the inverse momentum operator, and τ_2 is the algebraic approximate of the inverse continuity operator.

Let us linearize Equation (3.3) by setting the convection velocity to $\boldsymbol{a}$. Then, the values of τ_1 and τ_2 are:

$$\begin{aligned} \tau_1 &= \left(\tfrac{4\mu}{h^2} + 2\rho\tfrac{|\boldsymbol{a}|}{h}\right)^{-1} \text{ and} \\ \tau_2 &= c_1\mu + c_2\rho|\boldsymbol{a}|h, \end{aligned}$$

with $c_1 = 4$ and $c_2 = 2$.

3.2.3 Solution Procedure

The time discretization is based on second order BDF (Backward Differentiation) schemes and the linearization is carried out using the Picard method. At each time step, the linearized velocity-pressure coupled algebraic system must be solved:

$$\begin{bmatrix} \mathbf{A}_{uu} & \mathbf{A}_{up} \\ \mathbf{A}_{pu} & \mathbf{A}_{pp} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_u \\ \mathbf{b}_p \end{bmatrix},$$

where $\mathbf{u}$ and $\mathbf{p}$ are velocity and pressure unknowns. In order to solve efficiently this system on large supercomputers, we consider a split approach, see [37]. That is, we solve for the pressure Schur complement system. In its simplest

form, this method can be understood as a fractional step technique. The advantage of this technique is this it leads to two decoupled algebraic systems: one for the velocity and one for the pressure. The Orthomin(1) method, explained in [38], is used to solve the pressure system. In our work, we only consider the continuity preserving Orthomin(1). Both momentum and continuity are preserved only when convergence of the algorithm is achieved. The continuity preserving Orthomin(1) iteration reads:

1. Solve momentum equation: $\mathbf{A}_{uu}\mathbf{u}^{k+1} = \mathbf{b}_u - \mathbf{A}_{up}\mathbf{p}^k$.
2. Compute Schur complement residual: $\mathbf{r}^k = [\mathbf{b}_p - \mathbf{A}_{pu}\mathbf{u}^{k+1}] - \mathbf{A}_{pp}\mathbf{p}^k$.
3. Solve continuity equation: $\mathbf{Q}\mathbf{z} = \mathbf{r}^k$.
4. Solve momentum equation: $\mathbf{A}_{uu}\mathbf{v} = \mathbf{A}_{up}\mathbf{z}$.
5. Compute $\mathbf{x} = \mathbf{A}_{pp}\mathbf{z} - \mathbf{A}_{pu}\mathbf{v}$.
6. Compute $\alpha =< \mathbf{r}^k, \mathbf{x} > / < \mathbf{x}, \mathbf{x} >$.
7. Update velocity and pressure:
$$\left\{ \begin{array}{lcl} \mathbf{p}^{k+1} & = & \mathbf{p}^k + \alpha\mathbf{z}, \\ \mathbf{u}^{k+2} & = & \mathbf{u}^{k+1} - \alpha\mathbf{v}. \end{array} \right.$$
8. Compute Schur complement residual: $\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha\mathbf{x}$.
9. Solve continuity equation: $\mathbf{Q}\mathbf{z} = \mathbf{r}^{k+1}$.
10. Update velocity and pressure:
$$\left\{ \begin{array}{lcl} \mathbf{p}^{k+2} & = & \mathbf{p}^{k+1} + \mathbf{z}, \\ \mathbf{u}^{k+3} & = & \mathbf{u}^{k+2} + \mathbf{C}(\mathbf{p}^{k+2} - \mathbf{p}^{k+1}). \end{array} \right.$$

The superscript k is the iteration index. The matrix $\mathbf{Q}$ is the preconditioner and $\mathbf{C}$ is a correction matrix that depends on the preconditioner.

3.2.4 Algebraic Solvers

The two algebraic systems resulting from the Orthomin(1) method applied to the pressure Schur complement must be solved. For the momentum equation, the GMRES or BiCGSTAB methods are considered, with symmetric Gauss-Seidel preconditioner. For the pressure system, a Deflated Conjugate Gradient (CG) method [39] with linelet preconditioning when boundary layers are considered [40] has been developed in the framework of PRACE FP7 European Project. The Figure 3.1 compares the convergence of the classical CG with diagonal preconditioning, the deflated CG with diagonal preconditioning and the Deflated CG with linelet preconditioning for a thermal turbulent cavity with boundary layer mesh. This last method exhibits a strong robustness and enables to obtain a much better rate of convergence.

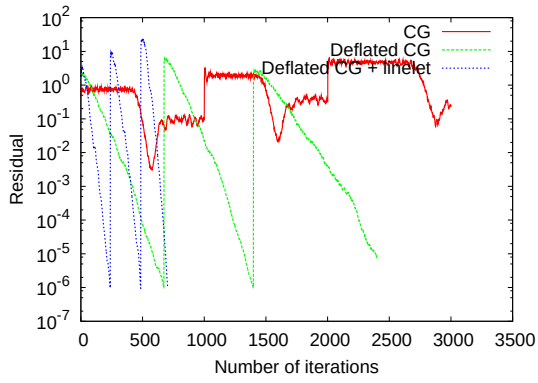

Figure 3.1: Convergence of different solvers.

3.2.5 Parallelization

The parallelization is based on a master-worker strategy for distributed memory supercomputers, using MPI as the message-passing library [4, 37]. The master reads the mesh and performs the division of the mesh into mesh subdomains using METIS (an automatic graph partitioner). Each process will then be in charge of a subdomain. These subdomains are the workers. The workers build the local element matrices and the local right-hand sides, and are in charge of finding the resulting system solution in parallel. In the elementary assembling tasks, no communication is needed between the workers, and the scalability depends only on the load balancing. In the iterative solvers, the scalability depends on the size of the interfaces and on the communication scheduling.

As mentioned previously, the momentum and continuity equations are solved with unsymmetric and symmetric iterative solvers respectively. During the execution of the iterative solvers, two main types of communications are required:

- Global communications via `MPI_AllReduce`, which are used to compute residual norms and scalar products.
- Blocking point-to-point communications via `MPI_Send` and `MPI_Recv`, which are used when sparse matrix-vector products are calculated.

Both types of communication were described in Chapter 2. The global communications corresponds to the parallel exchange Algorithm 4 and the blocking point-to-point communications corresponds to the parallel exchange Algorithm 3.

All solvers need both these types of communication, but, when using complex solvers like the DCG (Deflated Conjugate Gradient Method), additional operations may be required, such as the `MPI_AllGatherv` functions, explained in [39]. When using parallelized sequential solvers in Alya, the solution obtained in parallel is, up to round-off errors, the same as the sequential one all the way

through the computation. This is because the mesh partition is only used for distributing work without altering the actual sequential algorithm in any way. This would not be the case if one considered more complex solvers, like the primal/dual Schur complement solvers, or more complex preconditioners, like linelet or block LU, which are implemented as well. Figure 3.2 is a schematic flowchart for the execution of a simulation using Alya. The tasks that the master process is responsible for are shown on the left side of the Figure 3.2 with a grey background. The master process performs the first steps of the execution, namely reading the file and partitioning the mesh. Afterwards, the master sends the corresponding subdomain information to each worker process; then the master and the workers enter the time and linearization loops, represented as one single loop.

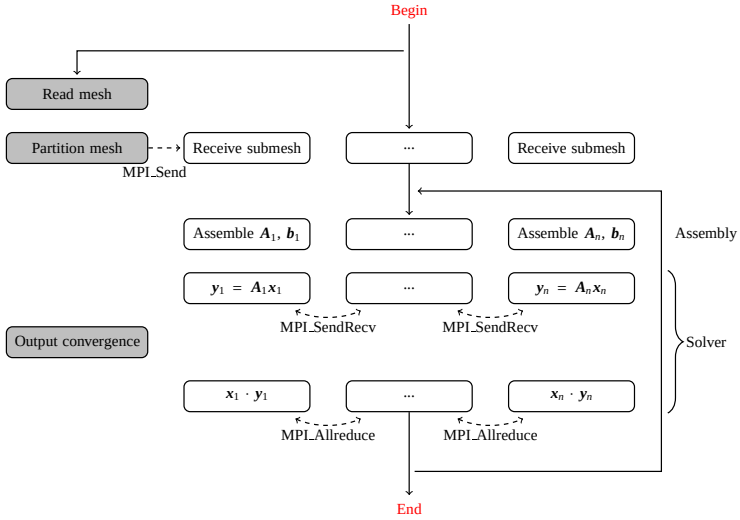

Figure 3.2: Flowchart for Alya execution. The tasks that the master and worker processes are responsible for are shown on figure with a grey and white background respectively.

Fluid simulations have been tested on Blue Waters Supercomputer and Jugene Supercomputer with two viscous Navier-Stokes benchmarks, see Figure 3.3.

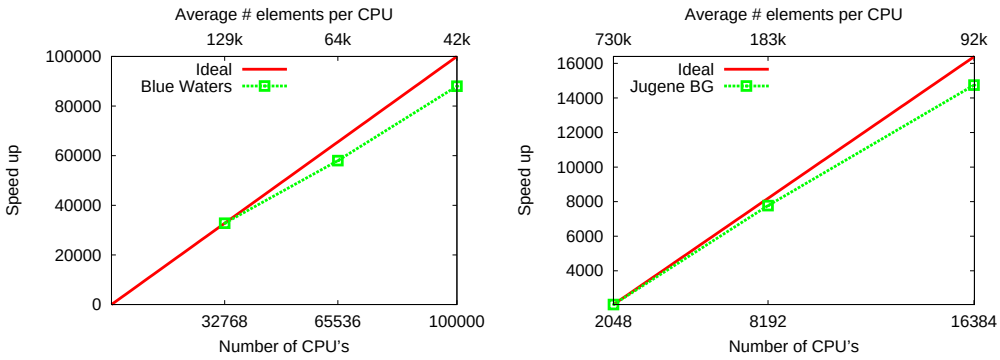

Figure 3.3: Speedup of the incompressible Navier-Stokes solver for solving different physical problems.

4
Rigid Body

In this chapter, once the Newton-Euler equations are introduced, we will explain the numerical scheme that models the movement of a rigid solid given the forces exerted on the body.

4.1 The Newton-Euler equations

The position of an arbitrary point inside a rigid body at a given time t can be defined as

$$\boldsymbol{p}(t) = \boldsymbol{x}(t) + \boldsymbol{r}(t), \tag{4.1}$$

where $\boldsymbol{x}(t)$ is the position of the center of mass of the body and $\boldsymbol{r}(t)$ is the position of $\boldsymbol{p}(t)$ relative to $\boldsymbol{x}(t)$. Considering that

$$\boldsymbol{r}(t) = \boldsymbol{R}(t) \cdot \boldsymbol{r}_0,$$

where $\boldsymbol{R}(t)$ is the rotation of the body about $\boldsymbol{x}(t)$ and $\boldsymbol{r}_0$ is the initial position of $\boldsymbol{p}(t)$ relative to $\boldsymbol{x}(t)$, Equation (4.1) can be rewritten as

$$\boldsymbol{p}(t) = \boldsymbol{x}(t) + \boldsymbol{R}(t) \cdot \boldsymbol{r}_0.$$

Taking into account that the rotation matrices are orthogonal, the velocity of $\boldsymbol{p}(t)$ can be expressed as

$$\begin{aligned} \dot{\boldsymbol{p}}(t) &= \dot{\boldsymbol{x}}(t) + \dot{\boldsymbol{R}}(t) \cdot \boldsymbol{r}_0 \\ &= \boldsymbol{v}(t) + \dot{\boldsymbol{R}}(t) \cdot \boldsymbol{R}^T(t) \cdot \boldsymbol{r}(t), \end{aligned}$$

where $\boldsymbol{v}(t)$ is the linear velocity of the body. The product $\dot{\boldsymbol{R}}(t) \cdot \boldsymbol{R}^T(t)$ defines an antisymmetric tensor:

$$\boldsymbol{W}(t) := \dot{\boldsymbol{R}}(t) \cdot \boldsymbol{R}^T(t) = \begin{bmatrix} 0 & -\omega_3(t) & \omega_2(t) \\ \omega_3(t) & 0 & -\omega_1(t) \\ -\omega_2(t) & \omega_1(t) & 0 \end{bmatrix}, \tag{4.2}$$

where $\omega_1(t), \omega_2(t)$ and $\omega_3(t)$ are the components of the angular velocity vector $\boldsymbol{\omega}(t)$ of the body. The tensor $\boldsymbol{W}(t)$ is called the angular velocity tensor.

The linear acceleration $\boldsymbol{a}(t)$ and angular acceleration $\boldsymbol{\alpha}(t)$ of the body are related with the input force $\boldsymbol{f}_F(t)$ and input torque $\boldsymbol{\tau}_F(t)$ by the Newton-Euler equations:

$$\boldsymbol{f}_F(t) = m\boldsymbol{a}(t) \tag{4.3}$$

and

$$\boldsymbol{\tau}_F(t) = \boldsymbol{I}(t) \cdot \boldsymbol{\alpha}(t) + \boldsymbol{\omega}(t) \times (\boldsymbol{I}(t) \cdot \boldsymbol{\omega}(t)), \tag{4.4}$$

where m is the total mass of the body and $\boldsymbol{I}(t)$ is the inertia tensor. By integrating in time the Equations (4.3) and (4.4), the velocity and the position of the rigid body can be determined.

4.2 The Newton-Euler discretization

Assume we know the force $\boldsymbol{f}_F^{n+1}$ and torque $\boldsymbol{\tau}_F^{n+1}$, exerted by the fluid, at the current time step t^{n+1}. Both will be approximated as described in Chapter 6. Then, the linear acceleration is easily computed by dividing the current force exerted on a rigid body by the total mass of the body

$$\boldsymbol{a}^{n+1} = \frac{\boldsymbol{f}_F^{n+1}}{m}.$$

The superscript $n+1$ refers to the current values of the simulation. The linear velocity and linear displacement of the center of mass can be determined using the Newmark scheme as method of numerical integration. Given the time step Δt of simulation, the Newmark method states that the current linear velocity is equal to

$$\boldsymbol{v}^{n+1} = \boldsymbol{v}^n + \Delta t(1-\gamma)\boldsymbol{a}^n + \Delta t\gamma\boldsymbol{a}^{n+1}$$

and the current linear displacement is

$$\boldsymbol{x}^{n+1} = \boldsymbol{x}^n + \Delta t\boldsymbol{v}^n + \Delta t^2(1/2-\beta)\boldsymbol{a}^n + \Delta t^2\beta\boldsymbol{a}^{n+1},$$

where γ and β are specified coefficients of the integration method, and the superscript n refers to the values from the previous time step of the simulation. The coefficients γ and β are deeply studied in [41].

The angular velocity vector can also be computed using Newmark as method of numerical integration:

$$\boldsymbol{\omega}^{n+1} = \boldsymbol{\omega}^n + \Delta t(1-\gamma)\boldsymbol{\alpha}^n + \Delta t\gamma\boldsymbol{\alpha}^{n+1}.$$

Nevertheless, the implementation of an iterative method is necessary in order to obtain a good approximation of the solution of the nonlinear ordinary differential Equation (4.4), the Euler rotation equation.

4.3 Algorithm of the Euler rotation equation

The rotation of the body around its center of mass can be computed using the relation from Equation (4.2) as shown below:

$$\boldsymbol{R}^{n+1} = \boldsymbol{R}^n + \Delta t \boldsymbol{W}^n \cdot \boldsymbol{R}^n, \tag{4.5}$$

where $\boldsymbol{W}^n$ is the angular velocity tensor obtained from the previous time step. Then, the current components of $\boldsymbol{W}(t)$ are obtained by solving the Euler rotation equation. Thus, the current angular acceleration is equal to

$$\boldsymbol{\alpha}^{n+1} = (\boldsymbol{I}^{-1})^n \cdot \left[\boldsymbol{\tau}_F^{n+1} - \boldsymbol{\omega}^n \times (\boldsymbol{I}^n \cdot \boldsymbol{\omega}^n)\right],$$

and the angular velocity vector using Newmark as method of numerical integration is

$$\boldsymbol{\omega}^{n+1} = \boldsymbol{\omega}^n + \Delta t(1-\gamma)\boldsymbol{\alpha}^n + \Delta t\gamma\boldsymbol{\alpha}^{n+1}.$$

Note that the components of the angular velocity tensor $\boldsymbol{W}(t)$ can be obtained from the angular velocity vector $\boldsymbol{\omega}(t)$.

Note also that the inertia tensor is time dependent, so it is necessary to recalculate their values at each time step. In order to avoid this expensive task, the following relation can be used:

$$\boldsymbol{I}(t) = \boldsymbol{R}(t) \cdot \boldsymbol{J} \cdot \boldsymbol{R}^T(t),$$

where $\boldsymbol{J}$ is the initial inertia tensor of the body. This tensor is a symmetric tensor and is defined by

$$\boldsymbol{I} = \int_{\Omega_S} \rho_S \left(\boldsymbol{p} \cdot \boldsymbol{p}\boldsymbol{I}_d - \boldsymbol{p} \otimes \boldsymbol{p}\right) d\Omega_S, \tag{4.6}$$

where Ω_S is the body domain, ρ_S is the body density, $\boldsymbol{p}$ defines the position of a point in the body, $\boldsymbol{I}_d$ is the identity tensor, and $\otimes$ represent the tensor product.

In the current numerical implementation, bodies are described by their boundaries Γ_S (boundary mesh.) It is therefore convenient to re-express the initial inertia tensor of the body as an integral over its volume into an integral over its surface using the divergence/Gauss theorem, see [42] to a fast computation of other body properties. Then, from Equation (4.6), we have that for

each component of the inertia tensor $\boldsymbol{I}$:

$$
\begin{aligned}
I_{11} &= \frac{1}{3}\rho_S \int_{\Gamma_S} p_2^3 n_2 + p_3^3 n_3 \, d\Gamma_S, \\
I_{22} &= \frac{1}{3}\rho_S \int_{\Gamma_S} p_1^3 n_1 + p_3^3 n_3 \, d\Gamma_S, \\
I_{33} &= \frac{1}{3}\rho_S \int_{\Gamma_S} p_1^3 n_1 + p_2^3 n_2 \, d\Gamma_S, \\
I_{12} &= \frac{1}{4}\rho_S \int_{\Gamma_S} -p_1^2 p_2 n_1 - p_1 p_2^2 n_2 \, d\Gamma_S, \\
I_{13} &= \frac{1}{4}\rho_S \int_{\Gamma_S} -p_1^2 p_3 n_1 - p_1 p_3^2 n_3 \, d\Gamma_S, \text{ and} \\
I_{23} &= \frac{1}{4}\rho_S \int_{\Gamma_S} -p_2^2 p_3 n_2 - p_2 p_3^2 n_3 \, d\Gamma_S,
\end{aligned}
$$

where n_1, n_2, and n_3 are the components of the exterior normal of the body in $\boldsymbol{p}$.

Now, although the rotation matrix can be computed from (4.5), it is highly recommended to implement an iterative method to improve the approximate solution of this non-linear system of equations. An alternative algorithm is described below:

Initialize values: $(\cdot)^{i,n+1} = (\cdot)^n$.
Iterate while ϵ be higher than a given tolerance.

- $\boldsymbol{R}^{i+1,n+1} = \boldsymbol{R}^n + \Delta t \boldsymbol{W}^{i,n+1} \cdot \boldsymbol{R}^{i,n+1}$.
- $(\boldsymbol{I}^{n+1})^{-1} = (\boldsymbol{R}^T)^{i+1,n+1} \cdot \boldsymbol{J}^{-1} \cdot \boldsymbol{R}^{i+1,n+1}$.
- $\boldsymbol{\alpha}^{i+1,n+1} = (\boldsymbol{I}^{n+1})^{-1} \cdot [\boldsymbol{\tau}_F^{n+1} - \boldsymbol{\omega}^{i,n+1} \times (\boldsymbol{I}^{n+1} \cdot \boldsymbol{\omega}^{i,n+1})]$.
- $\boldsymbol{\omega}^{i+1,n+1} = \boldsymbol{\omega}^n + \Delta t(1-\gamma)\boldsymbol{\alpha}^n + \Delta t \gamma \boldsymbol{\alpha}^{i+1,n+1}$.
- $\epsilon = \|\boldsymbol{\omega}^{i+1,n+1} - \boldsymbol{\omega}^{i,n+1}\| / \|\boldsymbol{\omega}^{i+1,n+1}\|$.
- Update values: $(\cdot)^{i,n+1} = (\cdot)^{i+1,n+1}$.

The superscript $i+1$ refers to the values of the current iteration, the superscript i to the values of the previous iteration, ϵ is a norm for the angular velocity vector, and $(\cdot)$ represent all the angular variables.

Numerical errors will appear in the coefficients of $\boldsymbol{R}(t)$ so that the rotation matrix will no longer be precisely an orthogonal matrix. For this reason, at each iteration it is necessary to reorthogonalize $\boldsymbol{R}(t)$, see [43]. To avoid this problem, unit quaternions can be used to represent rotations. However, it is important that the quaternions remain normalized at each iteration. A deeper description of quaternions and general implementation aspects can be found in [44].

To finish, let us summarize the necessary steps to update the the position of the bodies (the coordinates of their boundary meshes.) Then, given the force and torque exerted on a body, do:

- Determine the current linear displacement $\boldsymbol{x}^{i+1}$ using Newmark as method of numerical integration.
- Determine the current rotation matrix $\boldsymbol{R}^{i+1}$ using the iterative algorithm described above.
- Finally, update the position $\boldsymbol{p}$ of each node that defines the boundary mesh of the body using the relation

$$\boldsymbol{p} = \boldsymbol{x}^{i+1} + \boldsymbol{R}^{i+1} \cdot \boldsymbol{r}_0,$$

  where $\boldsymbol{r}_0$ is the initial position of $\boldsymbol{p}$ relative to the center of mass of the body.

5
Rigid Body Interaction

In a simulation of a dynamic rigid body system we deal with multiple problems. First, we have to determine the motion of particles due to external forces. Then, when the particles are in movement, we have to prevent interpenetration between them and solve possible collisions when the bodies are in contact. The simulation framework of dynamic rigid bodies is well-known and tries to solve the problems mentioned above. In this context, we will present the algorithms to describe and solve the collision between bodies.

5.1 General Framework

The geometrical description of all the rigid bodies consists mainly of an STL file describing the outer boundaries of the bodies. Note that a priori, only one STL description is necessary for each type of bodies.

For the sake of simplicity, we will consider the bodies as convex polyhedra. For non-convex bodies a convex decomposition is required.

In our simulation, we are able to solve the interaction between a lot of bodies with different shapes. For this reason, a collision detection module, where the time of collision is estimated, is necessary to avoid a situation where we need to do a lot of corrections to fix penetration between bodies. Also, we have to solve possible collisions when the bodies are in contact. The simulation framework of dynamic rigid bodies, see [32, 45, 46], solves all these problems in the following consecutive stages:

1. Collision Detection.
2. Rigid Body Motion.
3. Collision Response

Now, we will explain how to implement the first and the last stages mentioned above. The rigid body motion was already described in Chapter 4.

5.1.1 Collision detection

Until now we determine the motion of bodies without considering collisions. In this context, the penetrations between bodies are not detected. To avoid this unrealistic situation, we can proceed as described below:

1. We estimate a time of contact between bodies.
2. Then, we move the bodies freely until the estimated time is reached.

To ensure not missing any collision we implemented a *dynamic collision detection* algorithm. In Figure 5.1 we see an example of a missing collision. Notice that no penetration was detected between the two consecutive time steps t_0 and t_1. The algorithm we use to estimate the time of collision is detailed in [47].

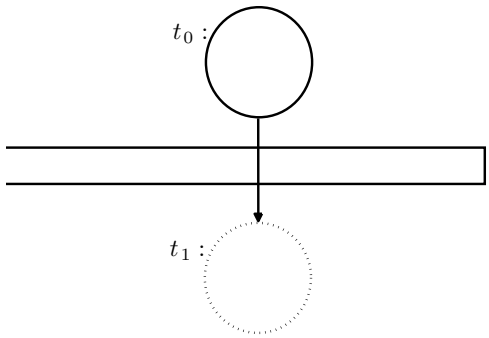

Figure 5.1: Missing collision.

Let us briefly explain the idea. Consider two convex polyhedra A and B, then determine:

- The closest points between the bodies: $\boldsymbol{p}_A$ on body A and $\boldsymbol{p}_B$ on body B.
- The direction $\boldsymbol{d} = \boldsymbol{p}_A - \boldsymbol{p}_B$.
- The minimum distance between bodies $d = \|\boldsymbol{d}\|$.
- The normalized direction $\hat{\boldsymbol{d}} = \boldsymbol{d}/d$.

In Figure 5.2 we see two convex bodies A and B and their closest points. Next, if the last time step reached is t_0, we define:

- $D_A(t)$ as an upper bound for the distance traveled by any point in A along $-\hat{\boldsymbol{d}}$ in the time interval $[t_0, t]$.
- $D_B(t)$ as an upper bound for the distance traveled by any point in B along $\hat{\boldsymbol{d}}$ in the same time interval.

A collision occurs at time $t = t_c$ between the two convex bodies A and B if

$$D_A(t_c) + D_B(t_c) \geq d.$$

This result is derived from the fact that the bodies are convex. Now, consider the total acceleration of any point in the body:

$$\boldsymbol{a}_{total}(t) = \boldsymbol{a}(t) + \boldsymbol{\alpha}(t) \times \boldsymbol{r}(t),$$

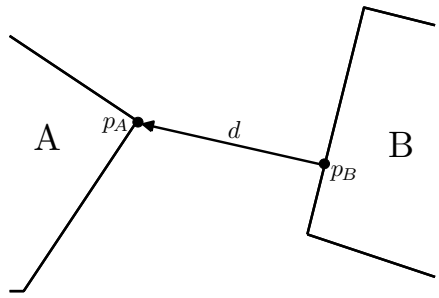

Figure 5.2: Closest points between the bodies A and B.

where $\boldsymbol{r}$ is the position from the center of mass to the point. The acceleration of an arbitrary point in the direction of $\hat{\boldsymbol{d}}$ is

$$\boldsymbol{a}_{total}(t) \cdot \hat{\boldsymbol{d}} = \boldsymbol{a}(t) \cdot \hat{\boldsymbol{d}} + (\boldsymbol{\alpha}(t) \times \boldsymbol{r}(t)) \cdot \hat{\boldsymbol{d}},$$

and fulfills

$$\boldsymbol{a}_{total}(t) \cdot \hat{\boldsymbol{d}} \leq \boldsymbol{a}(t) \cdot \hat{\boldsymbol{d}} + \alpha_{max} r_{max},$$

where r_{max} is the maximum distance of any point in the body from the center of mass and α_{max} is the maximum angular acceleration in the time interval $[t_0, t_c]$. Integrating twice over time the function on the right side of this inequality, a suitable expression for $D_A(t)$ and $D_B(t)$ is obtained. Thus, if we also consider the inequality (5.1.1), we obtain an estimated value for the time of collision.

5.1.2 Collision response

Once the bodies reach the time of collision estimated by the collision detection, we need to identify the bodies in contact and, when it is necessary, calculate new forces in order to avoid interpenetrations. These tasks are carried out by the collision response.

We use an impulse-based method for computing the contact forces. An impulse force is defined as

$$\boldsymbol{J}_S = \lim_{\Delta t \to 0} \int_{t_c}^{t_c + \Delta t} \boldsymbol{f} dt,$$

where t_c is the time of collision and Δt is the period of time of collision. An impulse produces an instantaneous change in the velocity of a body.

For frictionless bodies, the direction of the impulse is determined by the type of contact. For the typical face-vertex contact, the direction of the impulse is

the unit exterior normal of the face of contact. For edge-edge contact it is the unitized cross-product of the edge directions. Thus, we can express the impulse as

$$\boldsymbol{J}_S = j\boldsymbol{n}(t_c),$$

where j is the impulse magnitude and $\boldsymbol{n}(t_c)$ is the unit collision vector.

Now, consider two polyhedra bodies A and B in contact and suppose that the unit collision vector $\boldsymbol{n}(t_c)$ is in body B, see Figure 5.3. The relative velocity of these two bodies is defined as

$$v_{\text{rel}} = \boldsymbol{n} \cdot \left((\boldsymbol{v}_A^- + \boldsymbol{\omega}_A^- \times \boldsymbol{r}_A) - (\boldsymbol{v}_B^- + \boldsymbol{\omega}_B^- \times \boldsymbol{r}_B) \right).$$

If the relative velocity v_{rel} is positive, the bodies are moving apart. But if v_{rel} is negative, the bodies are moving closer together. Then, an impulse force is necessary to change the velocity of the bodies in order to avoid interpenetration.

Take into account that the magnitude j of the impulse is still undetermined. Then, to obtain an expression for j we have to consider the empirical law for frictionless collisions which relates the velocities of the bodies before and after the collision. The empirical law for frictionless collisions states that

$$\boldsymbol{n}(t_c) \cdot \left(\boldsymbol{u}_A^+(t_c) - \boldsymbol{u}_B^+(t_c) \right) = -c\boldsymbol{n}(t_c) \cdot \left(\boldsymbol{u}_A^-(t_c) - \boldsymbol{u}_B^-(t_c) \right), \tag{5.1}$$

where $\boldsymbol{u}_A$ is the total velocity of body A, $\boldsymbol{u}_B$ is the total velocity of body B, c is the restitution coefficient, the superscript $+$ indicates the quantities after the collision and the superscript $-$ the quantities before the collision. When $c = 1$, the collision is perfectly elastic. If $c = 0$ the collision is perfectly inelastic. For a collision that is perfectly elastic, the momentum and kinetic energy is conserved by the empirical law for frictionless collisions.

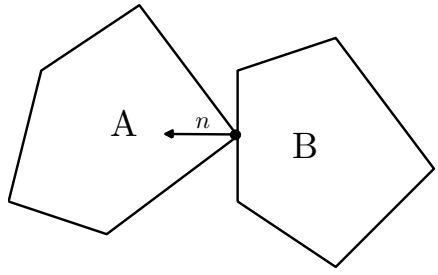

Figure 5.3: Contact between two bodies.

On the other hand, the linear and angular velocities in body A, after the collision, are related with the previous linear and angular velocities through an impulse by equations

$$\boldsymbol{v}_A^+(t_c) = \boldsymbol{v}_A^-(t_c) + \frac{j\boldsymbol{n}(t_c)}{m_A} \tag{5.2}$$

and

$$\boldsymbol{\omega}_A^+(t_c) = \boldsymbol{\omega}_A^-(t_c) + \boldsymbol{I}_A^{-1}(t_c)\left(\boldsymbol{r}_A(t_c) \times j\boldsymbol{n}(t_c)\right), \tag{5.3}$$

where $\boldsymbol{v}_A$ is the linear velocity of body A, $\boldsymbol{w}_A$ is the angular velocity of A, m_A is the mass of A, $\boldsymbol{I}_A^{-1}$ is the inverse of inertia tensor of A, and $\boldsymbol{r}_A$ is a vector defined from the contact point to the center of gravity of A. For body B we must consider the opposite impulse $-\boldsymbol{J}_S$.

Now, considering the total velocity of body A after collision:

$$\boldsymbol{u}_A^+(t_c) = \boldsymbol{v}_A^+(t_c) + \boldsymbol{\omega}_A^+(t_c) \times \boldsymbol{r}_A(t_c)),$$

by Equations (5.2) and (5.3) we obtain that

$$\boldsymbol{u}_A^+(t_c) = \boldsymbol{v}_A^-(t_c) + \frac{j\boldsymbol{n}(t_c)}{m_A} + \left(\boldsymbol{\omega}_A^-(t_c) + \boldsymbol{I}_A^{-1}(t_c)\left(\boldsymbol{r}_A(t_c) \times j\boldsymbol{n}(t_c)\right)\right) \times \boldsymbol{r}_A(t_c) \tag{5.4}$$

A similar expression can be obtained for body B considering the opposite impulse $-\boldsymbol{J}_S$.

Finally, the magnitude j of the impulse can be obtained replacing the equation (5.4) for body A in Equation (5.1), the law for frictionless contacts, and for body B with the opposite impulse $-\boldsymbol{J}_S$. Thus, the magnitude j is equal to

$$j = \frac{-(1+c)\boldsymbol{n} \cdot \left((\boldsymbol{v}_A^- + \boldsymbol{\omega}_A^- \times \boldsymbol{r}_A) - (\boldsymbol{v}_B^- + \boldsymbol{\omega}_B^- \times \boldsymbol{r}_B)\right)}{\frac{1}{m_A} + \frac{1}{m_B} + \boldsymbol{n} \cdot \left(\boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times \boldsymbol{n})\right) \times \boldsymbol{r}_A + \boldsymbol{n} \cdot \left(\boldsymbol{I}_B^{-1}(\boldsymbol{r}_B \times \boldsymbol{n})\right) \times \boldsymbol{r}_B}.$$

An expression for j is also obtained in [32].

5.2 Geometric tools algorithms

Important issues in the collision detection, the collision response, and the fluid and particles interactions are related to the implementation of efficient algorithms to search the minimum distance between a pair of particles or to determine if a node in the mesh is contained inside a particle. These searches can affect the performance of the whole system in a negative way: the time of simulation can grow considerably. To reduce the number of computations of this expensive task, it is necessary to implement different kind of structures to optimize these searches.

5.2.1 Skd-Trees

The skd-trees are binary trees. These structures are bounding volume hierarchies. Each node of these binary trees is a bounding volume for a subset of faces of a particle. In particular, a skd-tree allow us to find the shortest distance between a point and a surface mesh in an efficient way. The details of implementation for building these structures for a particle are described in [48] and outlined below:

1. Create a new node, the root node, see Figure 5.4.
2. Link all the faces of the particle with the root node.
3. Do for each newly created node whenever it has more than one face:
    - Determine the boundary box that contains all the faces linked to the current node.
    - Store the boundary box in the current node.
    - Find the largest dimension of the boundary box, let us denote as d.
    - Distribute the faces into two distinct sets. If n is the current number of faces, each set will have $n/2$ number of elements. The centroids of the boundary boxes of the faces in the first set will have the smallest values on the d coordinate. The second set will have the biggest ones.
    - Unlink the faces for the current node.
    - Create two new nodes, the child nodes of the current node.
    - Link the faces in the first set with the first child node.
    - Link the faces in the second set with the second child node.

The skd-tree construction for a particle is schematized in Figure 5.4. The thin red line indicates that the faces are not linked to the node and their information is not available. However, the information of the boundary box that contains the faces is still available. Only the leaf nodes, the nodes that do not have any children, have a face linked with them.

In [48] it is also described how to use the skd-trees to find the shortest distance between a point and a particle. The idea is to minimize an upper bound for the distance between the point and the particle while we are traversing the binary tree from the root node. We will denote this upper bound as dis. The algorithm is summarized below:

1. Determine the maximum distance between the point and the boundary box of the root node. Let us define this value as dis.
2. Traverse the binary tree from the root node in pre-order and perform the following operations:
    - Determine the minimum distance between the point and the boundary box of the current node. Let us define this value as min.

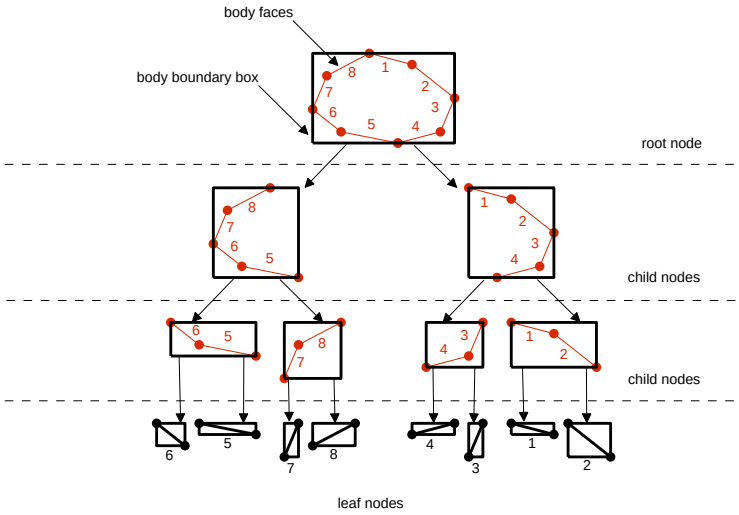

Figure 5.4: The skd-tree construction for a particle. The surface mesh of the body has 8 edges.

- If *min* is smaller than *dis* then:
  - If the current node is a leaf, a node that does not have any children, then there is one face identifier linked to the node. Save it in a candidate list.
  - If the current node is not a leaf then determine the maximum distance between the point and the boundary box of the current node. Let us define this value as *max*. If $max < dis$ then $dis = max$.
- Else, do not traverse the subtree below the current node.

3. Find the distances between the point and all the faces associated in the candidate list. Choose the smallest one.

Improvement in Skd-tree searching

A simple modification is implemented when we use the skd-trees in order to obtain a better performance in the simulation. In practice, we obtain better results if we first visit the nodes whose boundary boxes are closest to the point when we traverse the tree from the root node. The idea is described below:

1. Determine the maximum distance between the point and the boundary box of the root node. Let us define this value as *dis*.

2. Traverse the binary tree from the root node in pre-order and perform the following operations:
   - Determine the minimum distance between the point and the boundary box of the current node. Let us define this value as min.
   - If min is smaller than dis then:
     - If the current node is a leaf, a node that does not have any children, then there is one face linked to the node. Find the distance between the point and this face and save it as new_dis. If $new\_dis < dis$ then $dis = new\_dis$.
     - If the current node is not a leaf then sort their two child nodes such that we visit first the child node whose boundary box is closest to the point.
   - Else, do not traverse the subtree below the current node.
3. dis has the minimum distance between the point and the particle.

These structures was used in different applications [1, 3, 5, 6].

5.2.2 Closest points between particles

In order to determine the time of collision between two particles we first have to find the closest points between them. An algorithm to calculate the distance between two convex bodies is described in [49] and it is summarized below.

- Find the minimum distance d_A between the nodes of A and the particle B.
- Find the minimum distance d_B between the nodes of B and the particle A.
- Find the minimum distance d_E between the edges of A and the edges of B.
- Choose the shortest distance between d_A, d_B and d_E.

All these tasks can be carried out using skd-trees to obtain better computational times of execution.

5.2.3 Bucket sort

The idea is to subdivide the domain along each coordinate to obtain buckets or boxes with the same size in all the coordinates. This grid may have different numbers of boxes in each direction. The boxes will contain elements that are inside or intersect with them. The construction details are shown in [50].

By incorporating this kind of structure in the code, we will able to reduce the size of the search space when we need to carry out some local operations with the elements contained in the boxes.

In particular, this structure is used to store the nodes of the mesh in different boxes. Thus, in order to determine the nodes that are inside a body at each time step of the simulation, we only need to check the nodes inside the boxes that intersect with the boundary box of the body.

Consider the elements and nodes of the mesh shown in Figure 5.5, represented by small black squares and small circles respectively. The body and its respective boundary box are represented by a big red circle and a big red square. The big black squares represent a bucket sort structure, where the nodes of the mesh are stored. Then, in order to find the nodes inside the body, the program has only to consider the nodes in the mesh inside the boxes that intersect with the boundary box of body, the white circles in Figure 5.5.

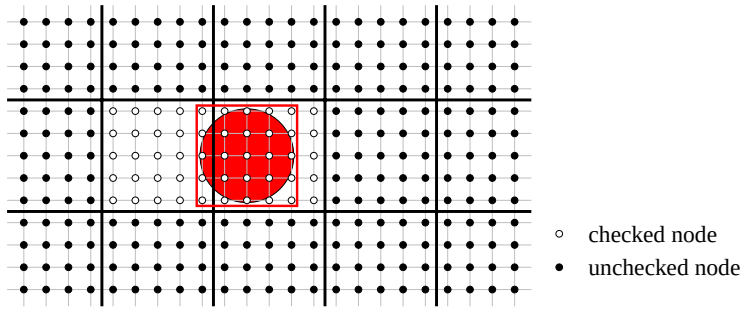

Figure 5.5: Bucket sort structure. In order to find the nodes inside the body, the program has only to consider the nodes represented by white circles, the nodes in the mesh inside the boxes that intersect with the boundary box of body.

6

Rigid body and fluid interaction

In this chapter we describe in detail the mathematical and numerical models to solve the interaction between a fluid and a rigid body. The solid is taken into account on the fluid by imposing the body surface velocity on the fluid as a Dirichlet boundary condition and the motion of the body is determined by the force and torque that the fluid exerts on its surface.

The numerical simulation of a fluid and its interaction with rigid bodies requires the implementation of many different algorithms. In order to present these algorithms in a organized manner we will first describe a general algorithm to solve the fluid and rigid body interaction. Then, we will describe in detail each part of the algorithm. These algorithms will be described considering a parallel context.

6.1 Framework of an embedded boundary mesh method

Let Ω_F and Ω_S be the fluid and solid domain, where Ω_S is the union of all the domains associated to the rigid bodies in the problem. Then, in an embedded boundary mesh method, at the beginning, $\overline{\Omega_F \cup \Omega_S}$ is discretized without any particular regard to the rigid bodies. The movements of the boundaries describes the movements of the solids inside the fluid. Then, at each time step of the simulation, the nodes and elements that are considered as part of the solids will be excluded from the assembly process. Finally, the solids are taken into account on the fluid by imposing the body velocity on the fluid as a Dirichlet boundary condition in an interpolated way.

In particular, at each time step of the simulation, the program identifies the elements in $\mathcal{E}$ whose volumes of intersection with the rigid body domain are big enough to consider them as part of the solid, that is, elements that belong to the set of hole elements $\mathcal{E}_{hol}$, see Figure 6.1. They are then excluded from the finite element assembly process. Let $\hat{\Gamma}_{S,h}$ be the internal boundary mesh generated in the fluid mesh once the hole elements have been excluded. In Figure 6.1 the bold black line represents $\hat{\Gamma}_{S,h}$. Inside this closed line one can find the hole elements represented by gray squares. In a embedded boundary mesh method, the velocity of the rigid solid is imposed on the nodes that define $\hat{\Gamma}_{S,h}$. Let this set be the set of fringe nodes: $\mathcal{N}_{fri}$. The set $\mathcal{N}_{fri}$ allow us to define other important sets of nodes: the set of free $\mathcal{N}_{fre}$ and the set of hole nodes $\mathcal{N}_{hol}$. The set of free nodes belongs to the discretized fluid domain and the set of hole nodes belongs to the discretized solid domain, see Figure 6.2.

Some of the implementation details of the embedded mesh boundary methods described next in this work was published previously in [1, 6].

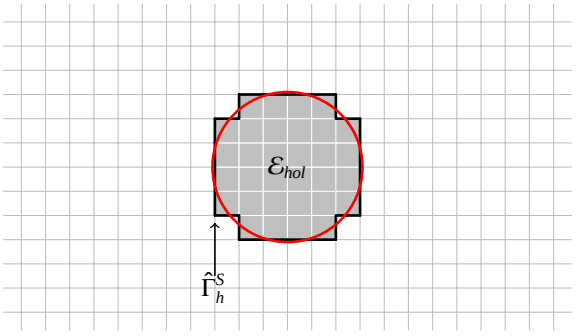

Figure 6.1: Hole elements and $\hat{\Gamma}_{S,h}$ schematization.

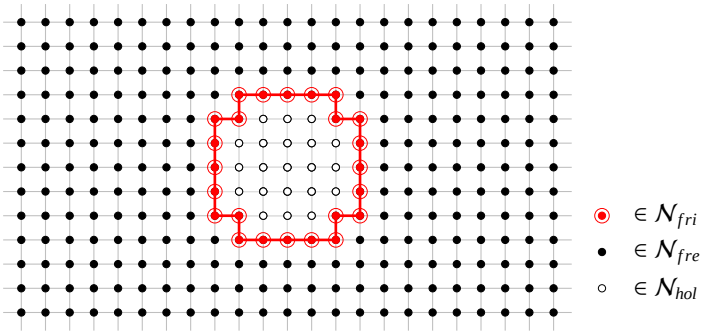

Figure 6.2: Fringe, free and holes nodes.

6.2 Fluid and rigid body interaction algorithm

The numerical schemes to solve the Navier-Stokes (NS) and the Newton-Euler (NE) equations need information from each other to account for the interaction. In order to close the problem, one is left with the variables involved in the coupling between the fluid and the rigid body problems. On the one hand, the variables that the fluid receives from the rigid body are enumerated below:

- The linear velocity $\boldsymbol{v}^{n+1}$ of the rigid body from Equation (4.3).
- The angular velocity $\boldsymbol{w}^{n+1}$ of the rigid body taken from Equation (4.4).
- The definition of the boundary mesh $\hat{\Gamma}_{S,h}^{n+1}$ once the program excludes the hole elements from the fluid discretization at the current time step $n+1$.
- The total velocity $\boldsymbol{u}_S^{n+1}$ to be imposed on $\hat{\Gamma}_{S,h}^{n+1}$. That is, the velocity of the set of fringe nodes $\mathcal{N}_{fri}^{n+1}$ to approximate the rigid body boundary velocity.

On the other hand, the set of variables that the solid requires from the fluid problem is enumerated below:

- The force $\boldsymbol{f}_F^{n+1}$ that the fluid exerts on the rigid solid.
- The torque $\boldsymbol{\tau}_F^{n+1}$ that the fluid exerts on the rigid solid.

Taking into account all the coupling variables described above, a new coupling strategy is briefly described in Algorithm 5.

Algorithm 5 NS-NE Coupling strategy

Initialize the variables
repeat
1. Determine the time step Δt, see Subsection 6.2.4.
2. Solve NE equations to obtain $\boldsymbol{v}^{n+1}$ and $\boldsymbol{w}^{n+1}$, see Chapter 4.
3. Define $\hat{\Gamma}_{S,h}^{n+1}$, which implies to determine $\mathcal{N}_{fri}^{n+1}$, see Subsection 6.2.1.
4. Determine $\boldsymbol{u}^n$ and $\boldsymbol{u}_{\text{msh}}^n$ applying the FMALE method, see Subsection 6.2.3.
5. Embedded approaches. Impose $\boldsymbol{u}_S^{n+1}$ on $\mathcal{N}_{fri}^{n+1}$, see Subsection 6.2.2.
6. Solve the NS equations to obtain $\boldsymbol{u}^{n+1}$ and p^{n+1}, see Chapter 3.
7. Determine $\boldsymbol{f}_F^{n+1}$ and $\boldsymbol{\tau}_F^{n+1}$ from $\boldsymbol{u}^{n+1}$ and p^{n+1}, see Subsection 6.2.5.

until the time of simulation is reached

Note that the NS-NE system is a two-way coupled problem. Therefore, Algorithm 5 consists of a staggered approximation of the coupled solution at each time step, as no coupling loop has been introduced and variables Γ_S^{n+1}, $\boldsymbol{u}_S^{n+1}$, $\boldsymbol{f}_F^{n+1}$ and $\boldsymbol{\tau}_F^{n+1}$ are approximations of the actual values at time step $n+1$. We thus expect the accuracy of the scheme to depend not only on

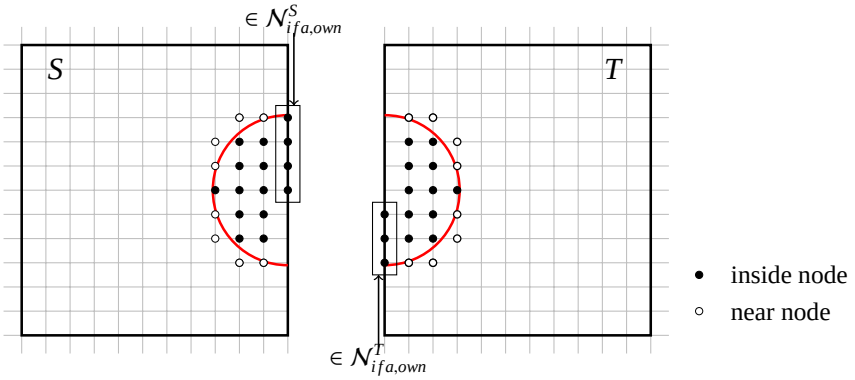

Figure 6.3: Near and inside nodes.

the way the set of coupling variables is defined but also on the time step Δt. Another important issue is the so-called added mass effect [51], which may cause instabilities in many cases. In order to circumvent this problem, it is important to make sure that the ratio between fluid and solid densities is not too close to the unity [52]. In the numerical experiments presented in this thesis, these ratios are rather small than the unity and we have not observed any instability. However, it must be said that subiterations at each time step could be used to achieve a strong coupling, increasing the computational cost, but without the need of significant effort in a parallel implementation.

Let us now describe in detail each step of Algorithm 5.

6.2.1 Algorithms to define an approximated body boundary $\hat{\Gamma}^{n+1}_{S,h}$

Fringe nodes identification algorithm

The idea is simple, but the implementation, specially in a parallel context, is somehow complicated.

First, for each subdomain, we have to identify the nodes inside the body, and then the nodes outside and near the body, see Figure 6.3.

It is important to remark that the round-off errors of the geometric operations such as the projection of a node on the solid and its minimum distance can cause that a node n located at the interface between subdomains be considered as part of the solid for a subdomain and as part of the fluid for another one. In order to avoid these errors, each subdomain S will be the only responsible for identifying the set of inside and near nodes considering the subset of nodes $\mathcal{N}^S_{int} \cup \mathcal{N}^S_{ifa,own}$ as illustrated in Figure 6.4. Thanks to the definition of the set of own interface nodes $\mathcal{N}^S_{ifa,own}$ for an arbitrary subdomain S, see Chapter 2, the program will be able to consider all the nodes in the mesh without consider

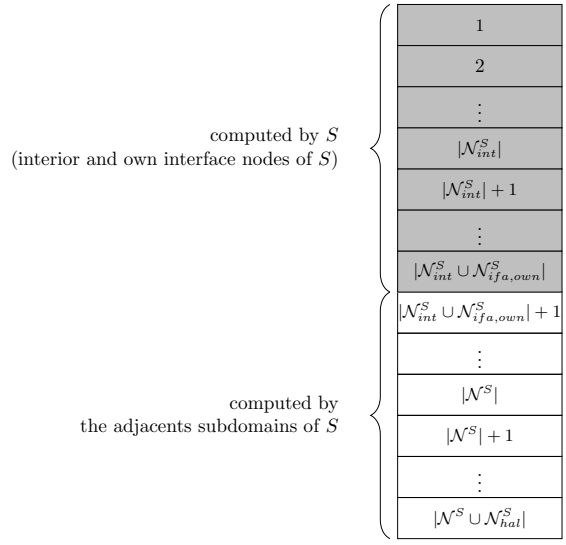

Figure 6.4: Array of data related with the set of nodes of S. The gray zone represents the nodes take into account by S.

the same node twice in a parallel execution.

Precises definitions of the set of inside and the set near nodes are carried out by Algorithms 6 and 7 respectively. As mentioned above, each subdomain S identify the set of inside and the set of near nodes considering only the set $N^S_{int} \cup \mathcal{N}^S_{ifa,own}$. Then, the new data is exchanged between the subdomains at the end of each algorithm using the algorithms of exchange **INE** and **HNE** defined in Subsections 2.5.1 and 2.5.2 respectively.

In Algorithm 6, the bin search uses the bucket sort structure explained in Subsection 5.2.3. This structure allows us to reduce the size of the search for nodes in $\mathcal{N}^S_{int} \cup \mathcal{N}^S_{ifa,own}$ that are inside the body considering only the nodes in the boxes of the bucket sort structure that intersect with the boundary box of the body.

In Algorithm 7 the idea, as illustrated in Figure 6.3, is to obtain a set of fringe nodes closer to the rigid body surface mesh.

Then, considering a node $n \in \mathcal{N}^S \cup N^S_{hal}$, for an index $i^S = index^S(n)$ we have that

$$inside(i^S) = \begin{cases} 1 \text{ when } n \text{ is inside the body} \\ 0 \text{ when } n \text{ is outside the body} \end{cases}$$

after the execution of Algorithm 6, and

$$near(i^S) = \begin{cases} 1 \text{ when } n \text{ is near the body} \\ 0 \text{ when } n \text{ is not near the body} \end{cases}$$

Algorithm 6 Inside nodes identification algorithm for an arbitrary subdomain S

Require: A numeric array *inside* with length $|\mathcal{N}^S \cup N^S_{hal}|$
Ensure: A modified array *inside*
  Initialize $inside(1 : |\mathcal{N}^S \cup \mathcal{N}^S_{hal}|) \leftarrow \mathbf{0}$
  Bin search. Select a reduced candidate list of nodes in $N^S_{int} \cup N^S_{ifa,own}$ using the bucket sort structure, see Subsection 5.2.3.
  for each node $n \in N^S_{int} \cup N^S_{ifa,own}$ in the previous list **do**
    Skd-tree search. Efficiently determine if n is inside the body
    if n is inside **then**
      $i^S = index^S(n)$
      $inside(i^S) = 1$
    end if
  end for
  call **INE**($inside(1 : |\mathcal{N}^S|)$), see Algorithm 1.
  call **HNE**($inside(1 : |\mathcal{N}^S \cup \mathcal{N}^S_{hal}|)$), see algorithm 2

Algorithm 7 Near nodes identification algorithm for an arbitrary subdomain S

Require: A numeric array *near* with length $|\mathcal{N}^S \cup N^S_{hal}|$
Ensure: A modified array *near*
  Initialize $near(1 : |\mathcal{N}^S \cup N^S_{hal}|) \leftarrow \mathbf{0}$
  for each node $n \in \mathcal{N}^S_{int} \cup \mathcal{N}^S_{ifa,own}$ outside the body **do**
    for each node $m \in \mathcal{C}_{nod}(n)$ inside the body **do**
      if n is closer to the surface mesh than m **then**
        $i^S = index^S(n)$
        $near(i^S) = 1$
      end if
    end for
  end for
  call **INE**($near(1 : |\mathcal{N}^S|)$), see Algorithm 1
  call **HNE**($near(1 : |\mathcal{N}^S \cup \mathcal{N}^S_{hal}|)$), see Algorithm 2

after the execution of Algorithm 7.

Finally, once the Algorithms 6 and 7 have been executed, we can determine the fringe nodes as described in Algorithm 8. By definition, a fringe node n has at least one free and hole node in its node connectivity $\mathcal{C}_{nod}(n)$. That is, n is an inside or near node with at least one node outside and not near the body in $\mathcal{C}_{nod}(n)$, a node at the interface between a solid and the fluid.

Algorithm 8 Fringe nodes identification algorithm for an arbitrary subdomain S

Require: A numeric array $fringe$ with length $|\mathcal{N}^S \cup N^S_{hal}|$
Ensure: A modified array $fringe$
  Initialize $fringe(1 : |\mathcal{N}^S \cup \mathcal{N}^S_{hal}|) \leftarrow \mathbf{0}$
  for each node $n \in N^S$ inside or near the body **do**
    if there is at least one node outside and not near the body in $\mathcal{C}_{nod}(n)$ **then**
      $i^S = index^S(n)$
      $fringe(i^S) = 1$
    end if
  end for
  call **HNE**($fringe(1 : |\mathcal{N}^S \cup \mathcal{N}^S_{hal}|)$), see Algorithm 2

As before, considering a node $n \in \mathcal{N}^S \cup N^S_{hal}$, for an index $i^S = index^S(n)$ we have that

$$fringe(i^S) = \begin{cases} 1 \text{ when } n \text{ is a fringe node} \\ 0 \text{ when } n \text{ is not a fringe node} \end{cases}$$

after the execution of Algorithm 8.

Free and hole nodes identification algorithm

The set of free $\mathcal{N}_{fre}$ and hole nodes $\mathcal{N}_{hol}$ are defined as the set of nodes in the mesh that are outside and inside the body respectively, excluding the set of fringe nodes $\mathcal{N}_{fri}$.

Hole and free elements Identification algorithm

The elements that will be considered as a part of the solid, the set of hole elements $\mathcal{E}_{hol}$, see Figure 6.1, can be easily identified from the set of fringe, free and hole nodes. The process is described in Algorithm 9 for an arbitrary subdomain S. As mentioned in this chapter, the set of hole elements $\mathcal{E}_{hol}$ will be excluded from the finite element assembly process.

Finally, the set of free elements is defined as $\mathcal{E}_{fre} = \mathcal{E} \backslash \mathcal{E}_{hol}$.

Algorithm 9 Solid elements identification algorithm for an arbitrary subdomain S

for each element $e \in \mathcal{E}^S \cup \mathcal{E}^S_{hal}$ **do**
  if all the nodes $n \in e$ belong to $\mathcal{N}_{fri}$ or $\mathcal{N}_{hol}$ **then**
    e will be considered as part of the solid, that means e belongs to $\mathcal{E}_{hol}$.
  end if
end for

6.2.2 Embedded approaches

In our implementation, two approaches that allow us to impose the velocity of the rigid body on $\hat{\Gamma}_{S,h}$ are considered: an updated body fitted and a non body fitted strategies. The first approach implements a local r-adaptivity algorithm that moves the nodes in $\mathcal{N}$ close to the rigid body surface in order to adapt their position to that of the body surface mesh. The second approach implements a high order kriging interpolation to impose the velocity of the body on the nodes in $\mathcal{N}$ close to the rigid body surface.

Updated body fitted method (UBF)

The updated body fitted approach implements a local r-adaptivity algorithm that moves the set of fringe nodes $\mathcal{N}_{fri}$ incrementally until the body surface mesh is reached. Then, the program directly imposes the velocity of the rigid body in each fringe node n_{fri} equation as:

$$\boldsymbol{u}_{fri} = \boldsymbol{u}_S(\boldsymbol{x}_{fri}),$$

where $\boldsymbol{u}_{fri}$ is the fringe node velocity, $\boldsymbol{x}_{fri}$ is the spatial coordinates of the fringe node and $\boldsymbol{u}_S(\boldsymbol{x}_{fri})$ is the velocity of the solid at $\boldsymbol{x}_{fri}$.

Actually, in our implementation, the algorithm that defines the movement of the nodes of the fluid mesh involves several sets of nodes besides the set of fringe nodes. The reason is to avoid distorted or inverted elements.

In this context, the algorithm also defines the movement of a group of subsets of the set of free nodes that have a close connectivity with the set of fringe nodes. In order to elucidate what we mean by 'close connectivity', let us introduce some definitions. Define the subset

$$\mathcal{N}^1_{fre} = \bigcup_{n \in \mathcal{N}_{fri}} \mathcal{C}_{nod}(n) \setminus (\mathcal{N}_{fri} \cup \mathcal{N}_{hol})$$

as the set of free nodes at level 1, see Figure 6.5. In an analogous way, a second subset

$$\mathcal{N}^2_{fre} = \bigcup_{n \in \mathcal{N}^1_{fre}} \mathcal{C}_{nod}(n) \setminus (\mathcal{N}^1_{fre} \cup \mathcal{N}_{fri})$$

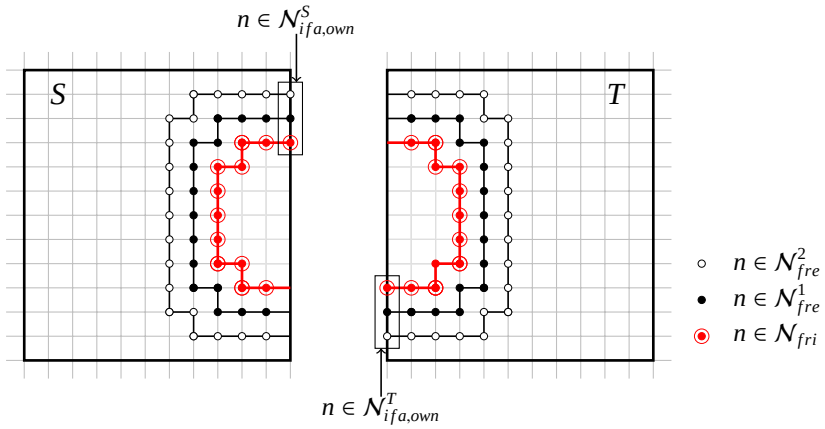

Figure 6.5: Sets of free nodes at different levels. The red concentric circles represent the set $\mathcal{N}_{fri}$. The sets $\mathcal{N}^1_{fre}$ and $\mathcal{N}^2_{fre}$ surround the set of fringe nodes.

will be called the set of free nodes at level 2, see Figure 6.5. In general, the subset

$$\mathcal{N}^l_{fre} = \bigcup_{n \in \mathcal{N}^{l-1}_{fre}} \mathcal{C}_{nod}(n) \Big\backslash \left(\mathcal{N}^{l-1}_{fre} \cup \mathcal{N}^{l-2}_{fre} \right)$$

defines the set of free nodes at level $l \in \mathbb{N} \backslash \{0, 1, 2\}$. Evidently, the smaller the value of l, the closer the connectivity with the set of fringe nodes.

The movement of the nodes of the fluid mesh is incremental and finishes when the set of fringe nodes reaches the body surface mesh. In particular, for each increment in the movement of the set of fringe nodes, there are several increments in the movement of the free nodes that belong to the set $\bigcup_{l \leq level} \mathcal{N}^l_{fre}$ for a given value of *level*. The flow of the whole algorithm is illustrated in Figure 6.6.

In particular, the movement of the free nodes is defined by a Laplacian-like smoothing technique similar to that described in [53, 54]. In these references, a node n is relocated in the centroid $\boldsymbol{c}$ of the nodes directly connected with n: the set of nodes $\mathcal{C}_{nod}(n)$, as described in Algorithm 10. In our approach, we perform some treatment to the set $\mathcal{C}_{nod}(n)$ so that the region defined by these nodes be convex.

On the other hand, the movement of the set of fringe nodes is more complex. The movement for a fringe node n is illustrated in Figure 6.7 and performed as indicated below:

- Determine the centroid $\boldsymbol{c}$ of the set $\mathcal{C}_{nod}(n) \cap \mathcal{N}_{fri}$, see Figure 6.7(Middle.)

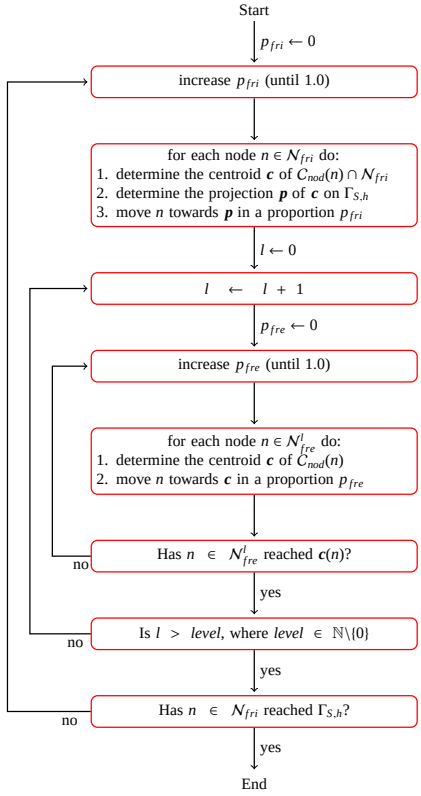

Figure 6.6: A scheme of the algorithm that defines the movement of nodes. The body surface mesh is represented as $\Gamma_{S,h}$. The parameters p_{fri} and p_{fre} are the proportions of the movement of the set of fringe and free nodes respectively. And the value $\boldsymbol{c}$ is the centroid defined by the set of nodes $\mathcal{C}_{nod}(n)$.

- Determine the point of projection $\boldsymbol{p}$ on the body surface mesh of $\boldsymbol{c}$, see Figure 6.7(Middle.)
- Move n towards $\boldsymbol{p}$, see Figure 6.7(Bottom.)

A precise description of the movement of the set of fringe and the set of free nodes at a given level is described next considering a parallel context.

Parallel movement algorithm

Consider now a distributed memory parallelization environment. An increment in the movement of the set of fringe nodes is described in Algorithm 11. And an increment in the movement of a set of free nodes at a given level is described in Algorithm 12. In both algorithms, as before, only one subdomain

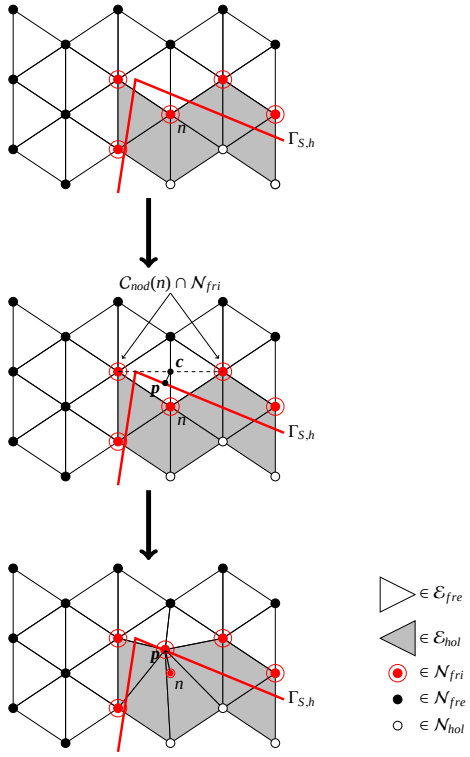

Figure 6.7: The movement of a fringe node n considering only one increment. (Middle) First, we have to determine the centroid c of the set of nodes $\mathcal{C}_{nod}(n) \cap \mathcal{N}_{fri}$. (Bottom) Then, we move the node n towards the projection p of c on the boundary mesh.

defines the movement of an interface node The idea is illustrated in Figure 6.5. The subdomain S only moves the interface nodes in $\mathcal{N}^S_{ifa,own}$ and the subdomain T the interface nodes in $\mathcal{N}^T_{ifa,own}$.

The whole movement is described in Algorithm 10 for a given number of increments and levels of the set of free nodes. From here, the Algorithms1 11 and 12 are called.

Non body fitted embedded method (NBF)

The non body-fitted approach implements a high order kriging interpolation algorithm. The idea is to impose the velocity of the body for each fringe node n_{fri} in an interpolating way. For this purpose, the program first has to consider a convenient subset of the set of free nodes $\mathcal{N}_{fre}$ that have a close connectivity with n_{fri}; denote it as $\mathcal{N}_{sel}(n_{fri})$. Then, the program imposes the velocity of

Algorithm 10 R-local adaptivity algorithm for an arbitrary subdomain S

Require:
1. An array *coordinates* with the positions of the nodes in $\mathcal{N}^S \cup \mathcal{N}^S_{hal}$
2. A value for *level*

Ensure: A modified array *coordinates*

```
proportion_fringe ← 0
repeat
    increase proportion_fringe
    call MOVE_FRINGES(coordinates, proportion_fringe), see Algorithm 11
    l ← 0
    repeat
        l = l + 1
        proportion_free ← 0
        repeat
            increase proportion_free
            call MOVE_FREES(coordinates, l, proportion_free), see Algorithm 12
        until proportion_free = 1
    until l ≥ level
until proportion_fringe = 1
```

Algorithm 11 Fringe nodes movement algorithm **MOVE_FRINGES** for an arbitrary subdomain S

Require:
1. An array *coordinates* with the positions of the nodes in $\mathcal{N}^S \cup \mathcal{N}^S_{hal}$
2. A value for *proportion*

Ensure: A modified array *coordinates*

```
Initialize new_coordinates(1 : N^S ∪ N^S_hal) ← 0
for n ∈ N_fri ∩ (N^S_int ∪ N^S_ifa,own) do
    Determine the centroid c defined by C_nod(n) ∩ N_fri
    Determine the projection p of c on Γ_S,h
    Determine the position in order to move n towards p in a proportion equal to proportion and save it in position
    i^S = index^S(n)
    new_coordinates(i^S) = position
end for
call INE(new_coordinates(1 : |N^S|)), see Algorithm 1
call HNE(new_coordinates(1 : |N^S ∪ N^S_hal|)), see Algorithm 2
coordinates ← new_coordinates
```

Algorithm 12 Free nodes movement algorithm **MOVE_FREES** for an arbitrary subdomain S

Require:
1. An array *coordinates* with the positions of the nodes in $\mathcal{N}^S \cup \mathcal{N}^S_{hal}$
2. A value for *level*
3. A value for *proportion*

Ensure: A modified array *coordinates*
Initialize $new\_coordinates(1 : \mathcal{N}^S \cup \mathcal{N}^S_{hal}) \leftarrow \mathbf{0}$
for $n \in \mathcal{N}^{level}_{fre} \cap (N^S_{int} \cup N^S_{ifa,own})$ **do**
Determine the centroid $\boldsymbol{c}$ defined by $\mathcal{C}_{nod}(n)$
Determine the position in order to move n towards c in a proportion equal to *proportion* and save it in *position*
$i^S = index^S(n)$
$new\_coordinates(i^S) = position$
end for
call **INE**($new\_coordinates(1 : |\mathcal{N}^S|)$), see Algorithm 1
call **HNE**($new\_coordinates(1 : |\mathcal{N}^S \cup \mathcal{N}^S_{hal}|)$), see Algorithm 2
$coordinates \leftarrow new\_coordinates$

the rigid body in the fringe node n_{fri} equation as

$$N_{fri}\boldsymbol{u}_{fri} + \sum_{n_i \in \mathcal{N}_{sel}(n_{fri})} N_i \boldsymbol{u}_i = \boldsymbol{u}_S(\boldsymbol{x}_S),$$

where $\boldsymbol{u}_i$ is the velocity of free node n_i, $\boldsymbol{x}_S$ is the projection point of the fringe node on the surface mesh of the body, and $\boldsymbol{u}_S(\boldsymbol{x}_S)$ is the velocity of the body at $\boldsymbol{x}_S$. N_{fri} and N_i are the interpolation coefficients determined by solving the matrix kriging system.

The whole algorithm can be divided into three consecutive main steps. For each fringe node n in $\mathcal{N}_{fri}$ do:

- The selection of a convenient subset of free nodes that has a close connectivity with n to perform the interpolation: $\mathcal{N}_{sel}(n) \subset \mathcal{N}_{fre}$.
- The assembly of the matrix of the kriging system to interpolate the body surface velocity. In particular, this velocity will correspond to the solid velocity at the projection point p of n on the body surface. The positions of the free nodes in $\mathcal{N}_{sel}(n)$ and p will be used in the assembly.
- The inversion of the matrix of the kriging system by using the LU decomposition method in order to obtain the interpolation coefficients N_{fri} and N_i of Equation 6.2.2.

Parallel element and nodes selection

The interpolation requires to previously select a subset of the set of free nodes $\mathcal{N}_{fre}$ with a close connectivity with a fringe node n. The idea is schematized in Figure 6.8.

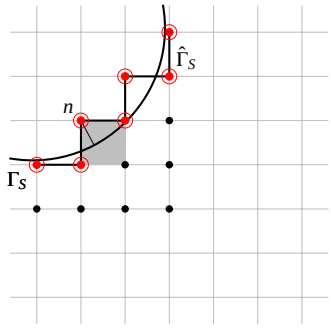

Figure 6.8: Illustration of the selection algorithm. the gray square denotes $e_{sel}(n)$. The red concentric circles denote members of the set of fringe nodes, and the black circles are the free nodes that belong to set $\mathcal{N}_{sel}(n)$.

Considering an arbitrary fringe node n, the definition of the set $\mathcal{N}_{sel}(n)$ can be carried out in an algorithmic fashion as follows:

- Select a convenient element $e_{sel}(n)$ containing n, that is $n \in e_{sel}(n)$. In Figure 6.8, the gray square denotes $e_{sel}(n)$.
- Then, let

$$\mathcal{N}_{sel}(n) = \bigcup_{m \in e_{sel}(n)} \mathcal{C}_{nod}(m) \cap \mathcal{N}_{fre}$$

  be the definition of the set of nodes used to perform the interpolation. In Figure 6.8, the black circles are free nodes that belong to set $\mathcal{N}_{sel}(n)$.

Algorithm 13 describes in detail the selection of nodes considering a parallel context. It can be divided into two parts. In the first part, each fringe node n starts by selecting a convenient element $e_{sel}(n)$. The idea is to select an element $e = \{n_1^e, n_2^e, n_3^e, ...\}$ with n_1^e, n_2^e, n_3^e, ... close to the boundary of the body.

As explained in Subsection 6.2.1, only one subdomain has to execute the geometric operations for any arbitrary node located at the interface between subdomains in order to assure the coherency of the data. For this reason, any arbitrary subdomain S will consider only its set of interior nodes N_{int}^S and its set of own interface nodes $\mathcal{N}_{ifa,own}^S$ to select $e_{sel}(n)$.

At the end of the first part of the Algorithm 13 the data is exchanged between the subdomains using the algorithms of exchange **INE** and **HNE** defined in Subsections 2.5.1 and 2.5.2 respectively.

In the second part, each fringe node n defines the set of nodes $\mathcal{N}_{sel}(n)$ to perform the interpolation taking into account $e_{sel}(n)$.

Algorithm 13 Selection nodes algorithm for an arbitrary subdomain S

▷ Selection of the convenient element $e_{sel}(n)$
Initialize $element(1 : |\mathcal{N}^S \cup \mathcal{N}^S_{hal}|) \leftarrow \mathbf{0}$
for each node $n \in \mathcal{N}_{fri} \cap (\mathcal{N}^S_{int} \cup \mathcal{N}^S_{ifa,own})$ **do**
  for each element $e \in \mathcal{C}_{ele}(n)$ (including $\mathcal{E}^S_{hal}$) **do**
    if the distance from the centroid of e to n is the smallest one **then**
      $i^S \leftarrow index^S(n)$
      $element(i^S) \leftarrow$ the global id of e
    end if
  end for
end for
call **INE**($element(1 : |\mathcal{N}^S|)$), algorithm 1
call **HNE**($element(1 : |\mathcal{N}^S \cup \mathcal{N}^S_{hal}|)$), algorithm 2
▷ Selection of the set of nodes $\mathcal{N}_{sel}(n)$
for each node $n \in \mathcal{N}_{fri} \cap \mathcal{N}^S$ **do**
  for each element $e \in \mathcal{C}_{ele}(n) \cap \mathcal{E}^S$) **do**
    $i^S \leftarrow index^S(n)$
    if the global id of e is equal to $element(i^S)$ **then**
      Select the set of nodes $\mathcal{N}_{sel}(n) = \bigcup_{m \in e_{sel}(n)} \mathcal{C}_{nod}(m) \cap \mathcal{N}_{fre}$ for n
    end if
  end for
end for

Kriging interpolation algorithm

In particular, we use an approximation method known as the universal kriging. The concepts and implementation aspects are detailed in [55].

In the kriging approach, the unknown function is the sum of a mean value $\mu(\boldsymbol{x})$ and an error term $\epsilon(\boldsymbol{x})$:

$$F(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \epsilon(\boldsymbol{x}),$$

where $\boldsymbol{x}$ is the position vector of the unknown function.

The approximation function for $F(\boldsymbol{x})$ is expressed as a linear combination of the data $\{F(\boldsymbol{x}_i)\}_{i=1,n}$:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} N_i(\boldsymbol{x}) F(\boldsymbol{x}_i).$$

The weights N_i are chosen to minimize the squared variance of the error of prediction:

$$\mathrm{Var}\left(F(\boldsymbol{x}) - f(\boldsymbol{x})\right)^2 = \mathrm{Var}\left(F(\boldsymbol{x}) - \sum_{i=1}^{n} N_i(\boldsymbol{x}) F(\boldsymbol{x}_i)\right)^2,$$

subject to the unbiasedness condition. This condition states that the mean of the unknown function is equal to the mean of its approximation:

$$\mu(\boldsymbol{x}) = \sum_{i=1}^{n} N_i(\boldsymbol{x}) \mu(\boldsymbol{x}_i).$$

Our choice for the mean of the unknown function is a polynomial function. Some implementation aspects are taken from [56].

6.2.3 FMALE

As mentioned before, the proposed embedded boundary techniques identify a set of free nodes $\mathcal{N}_{fre}$, a set of fringe nodes $\mathcal{N}_{fri}$, and a set of hole nodes $\mathcal{N}_{hol}$ at each time step of the simulation. Then, only the nodes in $\mathcal{N}_{hol}$ are excluded from the finite element assembly process. Now, consider the nodes in $\mathcal{N}_{fre} \cup \mathcal{N}_{fri}$ at the current time step t^{n+1} that were hole nodes at the previous time step t^n. They are the new fluid nodes of the simulation at t^{n+1}. These nodes were therefore, for practical purposes, nonexistent at the previous time step. Then, one of the practical problems with these new fluid nodes consists in defining the velocities at the previous time step t^n, which are required by the Navier-Stokes equations to compute the time derivatives.

This problem can be solved by considering a hidden motion of the mesh from t^n to t^{n+1}, which can be explained and formulated in the framework of the FMALE method [57]. In this work, new characteristics are adopted inside the FMALE implementation in order to improve the results obtained. These new features are explained below. But first, it is important to know in detail how the FMALE works.

We slightly reinterpret the FMALE algorithm described in [57] here. It consists of the following:

- Move the mesh at the current time step t^{n+1} such that all the new fluid nodes lie on the fluid domain at t^n. This virtual time step being referred to as t^{n*}.
- Then, interpolate the values of the previous velocity onto this new mesh from the solution obtained at t^n.
- Finally, compute a mesh velocity $\boldsymbol{u}_{\text{msh}}$ to be included in Equation (3.1) in order to recover the original mesh at t^{n+1} from t^{n*} and to account for the mesh motion.

In order to illustrate the FMALE approach, let us consider the one-dimensional example shown in Figure 6.9. The dotted lines represent the solid body at t^n, which moves to the right, and depicted with continuous lines at t^{n+1}, see Figure 6.9 (original mesh). At time t^n, the fringe node is node n_3 and at time t^{n+1} we end up with a new free node n_4, and a new fringe node n_5. The procedure is described below:

- Prescribe a displacement for the new fringe node n_5 such that at t^n it falls into the fluid, and move it incrementally together with nodes n_3 and n_4. Nodes n_1 and n_2 are assumed to be sufficiently far to remain fixed. The resulting new mesh at t^{n*} is shown in Figure 6.9 (b).
- The values of the velocities for the moved nodes n_3, n_4 and n_5 are then interpolated from the solution obtained at time t^n. This interpolation is represented by the vertical arrows between Figures 6.9(b) and 6.9(a).
- The mesh velocity is then computed from the positions obtained at time t^{n*} to recover the positions of the nodes on the original mesh t^{n+1}, Figures 6.9(b) and 6.9(c) for nodes n_3, n_4 and n_5. The nodal mesh velocity is simply $\boldsymbol{u}^i_{\text{msh}} = (\boldsymbol{x}_i^{n+1} - \boldsymbol{x}_i^{n*})/\Delta t$. The mesh velocity is represented by horizontal arrows.

A new virtual movement of the mesh inside the FMALE framework

The positions of the nodes at the previous virtual time step t^{n*}, as it is illustrated in Figure 6.9(b), are determined by the r-local adaptivity Algorithm 10.

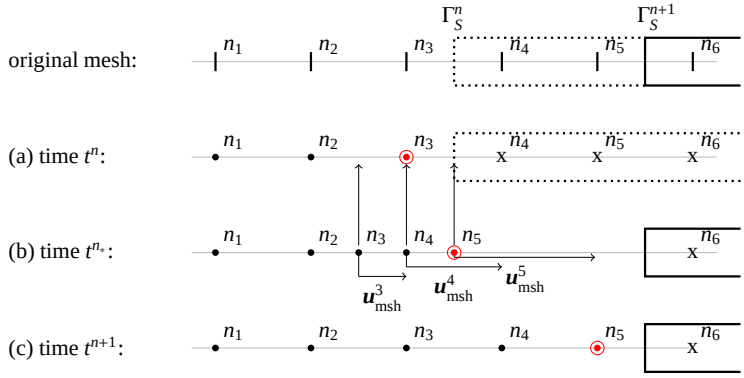

Figure 6.9: Illustration of the FMALE framework. The dotted lines represent the body surface mesh at the previous time step t^n and the continuous lines represent the body surface mesh at the current time step t^{n+1}. The red concentric circles denote members of the set of fringe nodes, black circles members of the set of free nodes, and crosses members of the set of hole nodes. The plots (a) and (c) represent the fluid mesh in two consecutive time steps after remeshing.

Considering the mesh at the current time step t^{n+1}, the idea is to incrementally move the set of fringe nodes $\mathcal{N}_{fri}$ and a subset of the set of free nodes until $\mathcal{N}_{fri}$ reaches the body surface mesh defined at the previous time step t^n.

A new way of interpolation of the velocity at the previous time step inside the FMALE framework

The velocities of the nodes at the previous virtual time step t^{n*} are taken from the velocities of the nodes at the real previous time step t^n, as it is illustrated by the vertical arrows between Figures 6.9(b) and 6.9(c). The values of these velocities are determined by interpolation using the high order kriging method defined in Subsection 6.2.2.

The interpolation, as mentioned in Subsection 6.2.2, requires that we previously select a set of nodes that have a close connectivity with the node whose velocity we need to interpolate. The algorithm to select the nodes is almost the same as the Algorithm 13. Considering the mesh at the previous time step t^n and an arbitrary moved node n at the virtual previous time step t^{n*}, the idea is also to select an convenient element $e_{sel}(n)$ that contains n, that is $n \in e_{sel}(n)$. Then, select a set of nodes defined by

$$\mathcal{N}_{sel}(n) = \bigcup_{m \in e_{sel}(n)} \mathcal{C}_{nod}(m) \cap \left(\mathcal{N}^n_{fri} \cup \mathcal{N}^n_{fre}\right) \tag{6.1}$$

to interpolate the velocity of n at the previous time step.

6.2.4 Time step Δt

The time step is limited by one algorithmic constraint and by some accuracy constraints. The algorithmic constraint comes from the way the FMALE formulation is implemented in order to properly work in a parallel context.

In the FMALE framework, the main purpose is to interpolate the fluid velocity for any arbitrary fluid node n at the previous time step. For this reason, and considering a parallel implementation, the program selects the set of nodes $\mathcal{N}_{sel}(n)$ defined in Equation (6.1), see Subsection6.2.3.

Then, the idea is to determine a time step Δt in such way that the set $\mathcal{N}_{sel}(n) \neq \emptyset$. That is, we have to assure that we have data to interpolate the velocity at the previous time step for n. Thus, consider again the set of nodes $\mathcal{N}_{sel}(n)$ defined in Equation (6.1). This set includes nodes two elements away from n. The idea is illustrated in Figure 6.8, where the nodes in set $\mathcal{N}_{sel}(n)$, the black circles, include nodes two elements away from n. Now, the movement of the mesh in the FMALE framework has a direct relation with the movement of the bodies. Then, in order to avoid that $\mathcal{N}_{sel}(n) = \emptyset$, we require that a rigid body do not cross more than two elements at each time step. Therefore, we define the time step of the NE solver as:

$$\Delta t_{\mathrm{NE}} = 2 \min_{n_{fri} \in \mathcal{N}_{fri}} \left(\frac{h_{fri}}{|\boldsymbol{u}_{fri}|} \right), \tag{6.2}$$

where h_{fri} is the minimum edge length that connects n_{fri} with the set of nodes $\mathcal{C}_{nod}(n_{fri})$ and $\boldsymbol{u}_{fri}$ is the velocity at n_{fri}.

As far as the accuracy constraint is concerned, both the NS and NE equations, as well as the coupling strategy, have different requirements. To control the time accuracy of the NS equations, we use the CFL condition and define

$$\Delta t_{\mathrm{NS}} = \alpha \min_{e_{fre} \in \mathcal{E}_{fre}} \left(\frac{4\mu}{\rho h_{fre}^2} + \frac{2|\boldsymbol{u}_{fre}|}{h_{fre}} \right)^{-1},$$

where α is called the safety factor which, for an unconditionally stable implicit scheme, could take in principle a high range of values, depending on the physics of the problem. A typical range is $[10, 1000]$. One can alternatively prescribe a time step Δt_{p} which does not rely on the mesh but on the physics of the problem.

For the NE equations, a critical time step should be devised as well, depending on the Newmark scheme considered. Note that the one given by Equation (6.2) relies on the mesh size, which would be irrelevant to solve the NE equations without an underlying mesh. However, we do not consider here any additional constraint for the Newmark scheme.

As for the time accuracy due to the coupling, we have no way to explicitly compute it in the general case. Therefore, the time step of the simulation is

computed as

$$\Delta t = \min(\Delta t_{\mathrm{NE}}, \Delta t_{\mathrm{NS}}) \quad \text{or} \quad \Delta t = \min(\Delta t_{\mathrm{NE}}, \Delta t_{\mathrm{p}}). \tag{6.3}$$

6.2.5 The force and torque exerted on the solid surface

In order to close the Newton-Euler equations for the rigid body, we need the force and the torque exerted by the fluid on the rigid body, $\boldsymbol{f}_F$ and $\boldsymbol{\tau}_F$, respectively. Let us first consider the force. Basically, there are two alternatives. Let $\boldsymbol{\sigma} \cdot \boldsymbol{n}$ be the normal stress exerted on the fluid, where $\boldsymbol{n}$ is the exterior normal to the fluid and $\boldsymbol{\sigma} = -p\boldsymbol{I} + 2\mu\varepsilon(\boldsymbol{u})$. The first option consists in integrating the pressure and viscous stresses along the solid boundary:

$$\boldsymbol{f}_F = \int_{\Gamma_S} \boldsymbol{\sigma} \cdot \boldsymbol{n} \, d\Gamma = -\int_{\Gamma_S} \boldsymbol{\sigma} \cdot \boldsymbol{n}_S \, d\Gamma,$$

where $\boldsymbol{n}_S$ is the exterior normal to the solid. The integration of these two stresses over the solid boundary is referred to as numerical force, as it is computed from the numerical solution for velocity and pressure.

The other option consists in considering the algebraic force, computed at the algebraic level. To understand the link between numerical and algebraic forces, let us consider the simple following Poisson equation:

$$\nabla \cdot (k \nabla u) = q,$$

which variational form reads:

$$\int_\Omega k \nabla u \cdot \nabla v \, d\Omega = \int_\Omega q v \, d\Omega + \int_{\Gamma_N} v g \, d\Gamma + \int_{\Gamma_D} v k \nabla u \cdot \boldsymbol{n} \, d\Gamma. \tag{6.4}$$

Γ_N is the part of the boundary Γ where the natural condition g is imposed, and Γ_D is the Dirichlet part of the boundary where the unknown is imposed to $\tilde{u}$, such that $\Gamma = \overline{\Gamma_D \cup \Gamma_N}$. Let N_i be the shape function of node n_i, then the matrix and right-hand side components resulting from the discretization of the variational form (6.4) are given by

$$A_{ij} = \int_\Omega k \nabla N_j \cdot \nabla N_i \, d\Omega \text{ and} \tag{6.5}$$

$$b_i = \int_\Omega q N_i \, d\Omega + \int_{\Gamma_N} N_i g \, d\Gamma. \tag{6.6}$$

In order to impose the Dirichlet condition at the variational level, we require the test function to vanish on Γ_D. At the algebraic level, one option consists in assembling the complete matrix and RHS of the system $\mathbf{A}$ and $\mathbf{b}$, given by

Equations (6.5) and (6.6) respectively, and then to force the solution in the matrix system to be the Dirichlet value. Let $\mathcal{N}_{dir}$ be the set of nodes in the Dirichlet boundary. To impose the Dirichlet condition, one can define:

$$\begin{cases} \tilde{A}_{ij} = \delta_{ij}, & \tilde{b}_i = \tilde{u}_i \quad \forall\, n_i \in \mathcal{N}_{dir} \text{ and} \\ \tilde{A}_{ij} = A_{ij}, & \tilde{b}_i = b_i \quad \text{otherwise,} \end{cases}$$

where δ_{ij} is the Kronecker delta, so that the final system to be solved reads:

$$\tilde{\mathbf{A}}\mathbf{u} = \tilde{\mathbf{b}}.$$

Now, let us go back to Equation (6.4). We find that the variational flux on the Dirichlet boundary can be computed as

$$\int_{\Gamma_D} vk\nabla u \cdot \boldsymbol{n}\, d\Gamma = \int_{\Omega} k\nabla u \cdot \nabla v\, d\Omega - \int_{\Omega} qv\, d\Omega - \int_{\Gamma_N} vg\, d\Gamma.$$

The discrete counterpart of last equation for node $n_i \in \mathcal{N}_{dir}$ is therefore

$$\begin{aligned} \textstyle\sum_j u_j \int_{\Gamma_D} kN_i\nabla N_j \cdot \boldsymbol{n}\, d\Gamma \; = \textstyle\sum_j u_j \int_{\Omega} k\nabla N_j \cdot \nabla N_i\, d\Omega \\ - \int_{\Omega} qN_i\, d\Omega - \int_{\Gamma_N} N_i g\, d\Gamma. \end{aligned}$$

Then, we note that the nodal flux on n_i can be associated to the residual of the equation as

$$\begin{aligned} \mathbf{f}_i \; &= \textstyle\sum_j u_j \int_{\Gamma_D} kN_i\nabla N_j \cdot \boldsymbol{n}\, d\Gamma \\ &= (\mathbf{A}\mathbf{u} - \mathbf{b})\,|_i\,. \end{aligned}$$

We note that in last equation we must consider $\mathbf{A}$ and $\mathbf{b}$, and not $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{b}}$, as these last quantities have been modified in order to impose the Dirichlet boundary condition. Eventually, we have that the total flux on the Dirichlet boundary is

$$f \; = \sum_{n_i \in \mathcal{N}_{dir}} \mathbf{f}_i.$$

By analogy, we can relate the residual of the momentum equations to the force exerted by the fluid on the particle. Considering only the fringe nodes, we find:

$$\boldsymbol{f}_F = \sum_{n_{fri} \in \mathcal{N}_{fri}} (\mathbf{b}_u - \mathbf{A}_{uu}\mathbf{u} - \mathbf{A}_{up}\mathbf{p})\,|_{fri}\,.$$

Note that as in the Poisson equation, one must consider the matrices $\mathbf{A}_{uu}$, $\mathbf{A}_{up}$ and vector $\mathbf{b}_u$ before imposing the Dirichlet boundary condition on the fringe nodes. As far as the algebraic torque is concerned we compute the nodal torque

$$\boldsymbol{\tau}_F = \sum_{n_{fri} \in \mathcal{N}_{fri}} (\mathbf{b}_u - \mathbf{A}_{uu}\mathbf{u} - \mathbf{A}_{up}\mathbf{p}) \mid_{fri} \times \mathbf{r}_{fri}.$$

The advantage of considering the algebraic force rather than the numerical force is now illustrated by a simple example. It consists of a two-dimensional flow over a cylinder at Re = 20. We have performed a mesh convergence for the value of the force using both the numerical and algebraic approximations. Figure 6.10 shows that the algebraic force approximation converges much faster to the asymptotic value than its numerical counterpart.

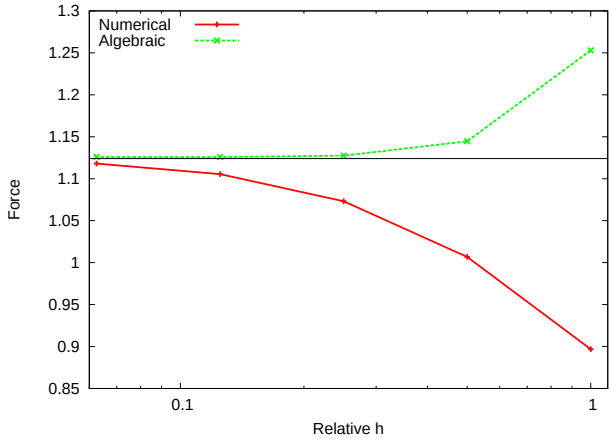

Figure 6.10: Force over a cylinder at Re = 20 using the numerical and algebraic approximations.

Another important advantage when we obtain the force algebraically has to do with its computational cost. The algebraic force consists of one simple matrix-vector product. It is indeed less expensive than computing a boundary integral, especially in a parallel context.

6.3 Mass conservation

To impose the velocity of a particle in the fluid by interpolation is a non-conservative strategy. As is shown in [58], the transmission of Dirichlet condition involves the necessity to ensure the conservation of the mass for each particle in the simulation. Let us consider a single body, the idea is to obtain new velocities $\boldsymbol{u}^*_{fri}$ for the fringe nodes from the values obtained using interpolation $\boldsymbol{u}_{fri}$ by

minimizing

$$\int_{\Gamma_S} \left| \boldsymbol{u}^*_{fri} - \boldsymbol{u}_{fri} \right|^2 d\Gamma_{S,h}$$

under the constraint

$$\int_{\Gamma_S} \boldsymbol{u}^*_{fri} \cdot \boldsymbol{n} d\Gamma_{S,h} = 0,$$

where $\Gamma_{S,h}$ is the wet boundary mesh of the rigid body and $\boldsymbol{n}$ is the normal vector. The restriction is derived in [58] and allows to conserve the mass going through the solid and therefore that of the whole system.

6.4 Summarizing

In order to summarize all the ingredients presented throughout this work, Figure 6.11 presents a flowchart of the general algorithm associated to the UBF and NBF approaches.

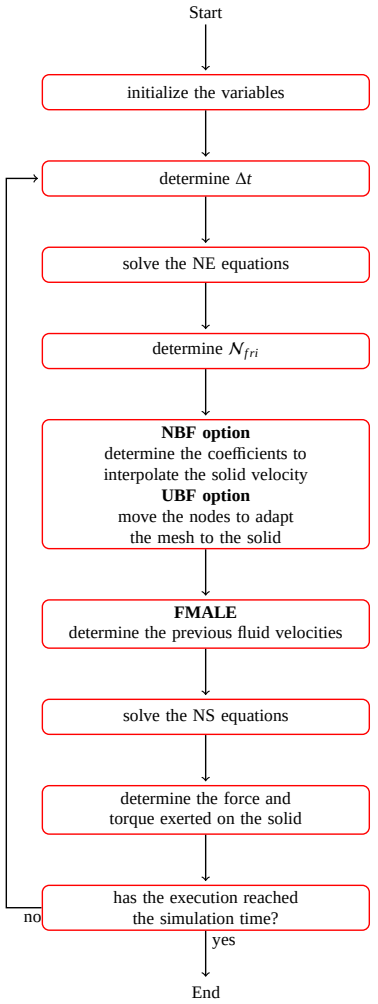

Figure 6.11: Flow chart of the whole process for both methods: UBF and NBF.

7
Numerical Experiments

This chapter will be divided into three parts. In the first part, we will consider examples that include the interaction between a fluid and a solid without take into account the collisions between the bodies. In the third part, on the contrary, we will consider the collisions between the rigid bodies without take into account any fluid. Finally, we will solve a problem that include all types of interactions explained in this thesis.

7.1 Fluid and rigid body interaction

We will first tackle a two-dimensional test case of a fluid and rigid solid interacting. Its main purpose is to determine the correctness of the coding and to study mesh convergence for the approaches explained in the previous sections: UBF and NBF. In particular, for this example, we consider two versions of the non body-fitted approach (NBF): one based on high order kriging interpolation (HNBF) and the other on linear (LNBF) kriging interpolation. The results show that the UBF and HNBF implementations have a much better performance than that of LNBF.

In the second example, we will solve a set of three-dimensional problems where the solutions can be analytically determined. The geometry is common to all of them. A spherical rigid body is immersed within a fluid. The simulation starts with the body at rest. Immediately, the sphere begins to fall downwards. The velocity of the body increases until the net forces acting on the sphere are equal to zero. Then, the body moves with a constant velocity known as terminal velocity. Different Reynolds numbers will be considered in order to compare UBF and HNBF approaches with the analytical solutions. The performance of UBF reaches better results as the Reynolds number increases.

In a third example, we will consider a circular cylinder immersed within a uniform fluid field that oscillates vertically with harmonic motion. The flow velocities, imposed as Dirichlet boundary condition, vary from one numerical experiment to another. The fluid domain and problem characteristics are described in [59]. The idea is to capture the interval of velocities in the fluid where the vortex shedding frequency f_v coincides with the natural frequency of a cylinder-spring system f_c. The characteristic behavior of the problem is the so-called "lock-in" phenomenon. Both, experimental and numerical results have been determined by a number of researchers.

In a fourth example, we will simulate the behavior of two Bileaflet mechanical heart valves. This mechanism consists of a pair of artificial heart valves that replace the native ones when they are malfunctioning. Due to the blood flow (forward and reverse) the two valves are opened and closed. The large

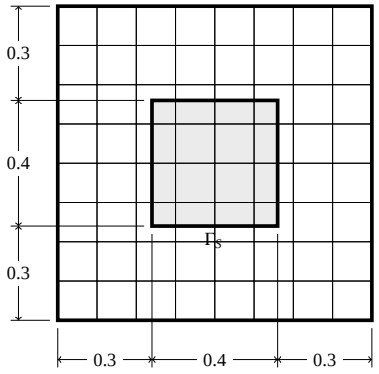

Figure 7.1: Problem domain for the manufactured solution.

acceleration rates that the valves experiment during the opening and closing phases and the maximum Reynolds number reached during the simulation are a challenge for any program that solves the interaction of a fluid with a rigid solid.

Finally, we will compare the parallel performance of the Navier-Stokes solver with and without considering the UBF and NBF algorithms used to simulate a fluid that contains twenty rigid solids with arbitrary shapes falling inside it.

7.1.1 Mesh convergence of a manufactured solution

The manufactured solution technique enables one, among other objectives, to easily carry out a mesh convergence of an implemented algorithm. Let us consider the Navier-Stokes operator $\mathcal{L}_{NS}(\boldsymbol{u}, p)$ represented by the LHS of Equations (3.1) and (3.2). Let $\boldsymbol{u}_{\text{man}}$ and p_{man} be some given target velocity and pressure, with a desired degree of smoothness. The manufactured solution technique consists in solving

$$\mathcal{L}_{NS}(\boldsymbol{u}, p) = \mathcal{L}_{NS}(\boldsymbol{u}_{\text{man}}, p_{\text{man}}),$$

together with $\boldsymbol{u} = \boldsymbol{u}_{\text{man}}$ as a Dirichlet boundary condition on the whole boundary of the computational domain, and $p = p_{\text{man}}$ on a unique node (indeed, when $\Gamma_N = \emptyset$, the pressure is defined up to a constant and thus should be prescribed somewhere.) We consider the following manufactured solution:

$$\begin{aligned} \boldsymbol{u}_{\text{man}} &= [\sin(\pi x - 0.7)\sin(\pi y + 0.2), \cos(\pi x - 0.7)\cos(\pi y + 0.2)] \text{ and} \\ p_{\text{man}} &= \sin(x)\cos(y), \end{aligned}$$

to be sought in the computational domain depicted in Figure 7.1. Note that the manufactured velocity field is divergence free.

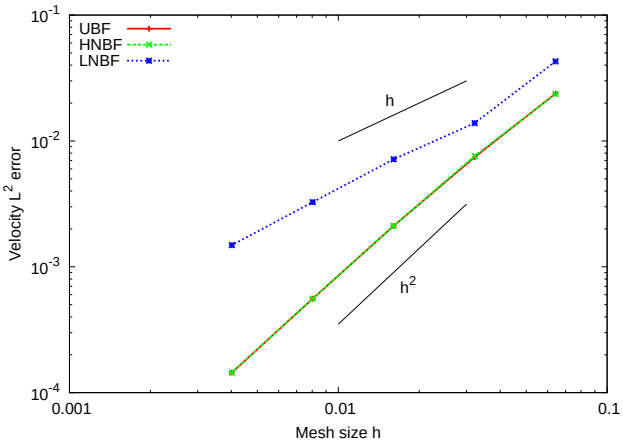

Figure 7.2: Mesh convergence of the velocity field for UBF, LNBF and HNBF.

First, We study the convergence of the solution as the mesh is refined. We compare the L^2 convergence of a manufactured solution, the convergence of the force at the solid boundary as well as that of the mass. To be able to assess this last one, the mass conservation algorithm presented in Section 6.3 was disabled. The mesh convergence is obtained for the UBF and NBF methods using linear and higher order kriging interpolations, as shown in Figure 7.2. In the case of UBF, the solid velocity is imposed to be equal to the value of the manufactured solution on the body surface, where the fringe nodes have been moved to. In the case of the two NBF methods, the solid velocity is interpolated at each fringe node n so that it is equal to the manufactured velocity at the projection point of n on the body surface. We observe that the convergence graphs for UBF and NBF with a high order kriging interpolation (HNBF) are very similar and both methods exhibit a quadratic convergence. It is also clear that the linear interpolation gives a linear mesh convergence.

Next, we show the mesh convergence of the total force exerted by the fluid on the solid and the mass unbalance resulting from the interpolation of the solid velocity in Figure 7.3. The top plot shows that the force converges much faster in the case of UBF and high order NBF than the linear NBF. As far as the mass conservation is concerned, the mass loss resulting from the UBF scheme is much smaller than that found with the other methods (bottom plot.) The order of convergence is neither linear nor clearly quadratic as nodes are not moved onto the body in a coherent way as the mesh size is refined. The mass loss of the linear NBF converges linearly to zero while that of the HNBF converges quadratically. Here the mass is computed as described in [58], using a closed quadrature rule.

Finally, let us study the effect of the mass conservation algorithm described

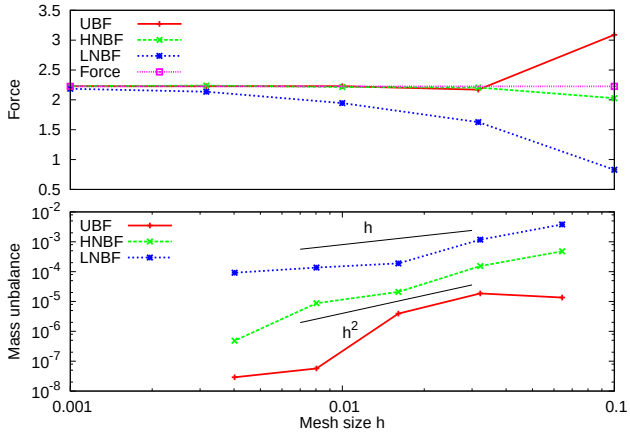

Figure 7.3: (Top) Mesh convergence of the force exerted on the solid for UBF, LNBF and HNBF. (Bot.) Mesh convergence of mass balance for UBF, LNBF and HNBF.

in Section 6.3 on the mesh convergence. Figure 7.4 shows the convergences of the velocity and pressure for the UBF, HNBF and LNBF methods. We observe that both the UBF and HNBF give very similarly results with and without mass conservation. On the contrary, the LNBF without mass conservation does not even converge. Let us remember that when the velocity Dirichlet boundary condition is imposed on the whole boundary just like in the case considered here, then the problem is not-well posed at the continuous level if the mass is not zero. At the numerical level, this fact translates into a non-converging pressure.

7.1.2 Terminal velocities

Stokes flow

Consider a spherical rigid body of radius $r = 1$ and density $\rho_s = 2$ immersed in fluid with density $\rho_f = 1$ and viscosity $\mu = 10$. For low Reynolds numbers, $Re << 1$, where the inertia effects are negligible, as in the problem just stated, Stokes derived a simple equation to obtain the terminal velocity of a sphere:

$$v_s = \frac{2(\rho_s - \rho_f)r^2 g}{9\mu} = -0.222$$

where g is the modulus of the gravity.

The geometry of the fluid domain is a cylinder with height equal to 60 and radius equal to 30. The initial position of the sphere is at 30 times the body radius from the sides of the cylinder and at 40 times the body radius from the

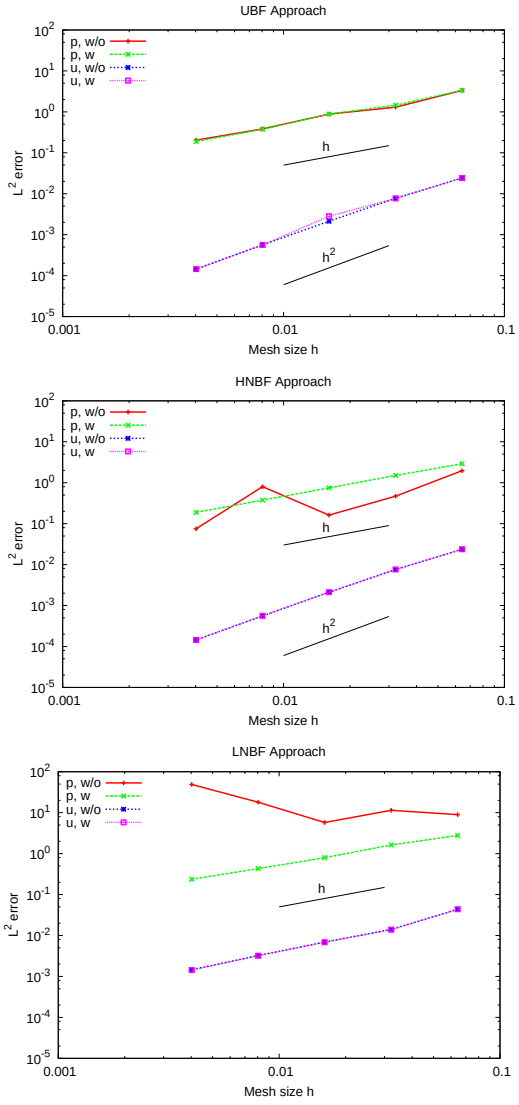

Figure 7.4: Mesh convergence of the velocity and pressure fields with and without mass conservation for (Top) the UBF scheme, (Mid.) the HNBF scheme, and (Bot.) LNBF scheme.

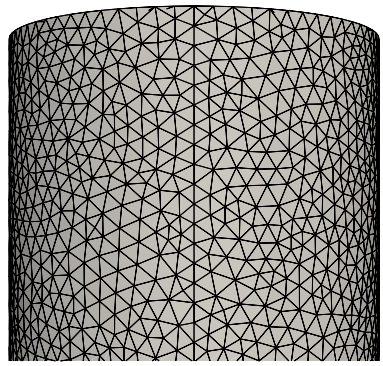

Figure 7.5: Mesh used for the cylindrical fluid domain.

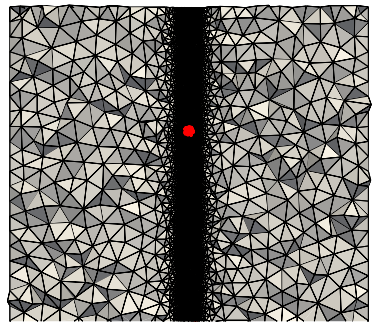

Figure 7.6: Initial position of the sphere in the interior of the mesh.

bottom of the cylinder. The mesh is unstructured, see Figures 7.5, 7.6, and composed of 400.000 tetrahedral elements, see Figure 7.6 where the red volume represents the sphere at the beginning of the simulation.

Figure 7.7 shows the set of fringe nodes $\mathcal{N}_{fri}$ without applying the local r-adaptivity algorithm and Figure 7.8 shows the set $\mathcal{N}_{fri}$ after the algorithm is applied.

In Figure 7.9, the velocity for UBF and HNBF approaches is compared with the analytical solution. Both velocities are almost equal and tend to the analytical solution.

Interpolation inside the FMALE framework

The values of the previous fluid velocities of the mesh are interpolated by the FMALE method, see Subsection 6.2.3. In order to study the influence of this

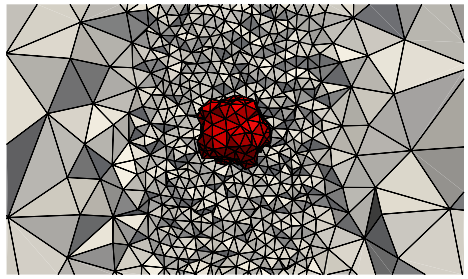

Figure 7.7: Set of fringe nodes before applying the r-local adaptivity algorithm.

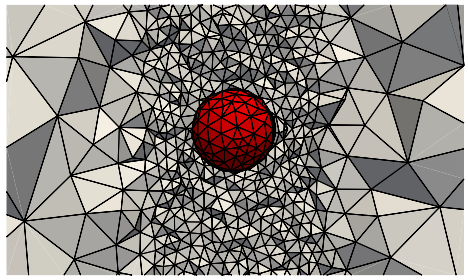

Figure 7.8: Set of fringe nodes after applying the r-local adaptivity algorithm.

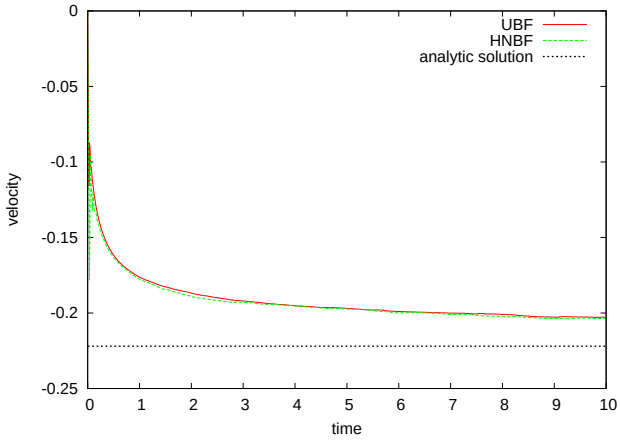

Figure 7.9: Numerical and analytical Stokes terminal velocity for Re = 0.004.

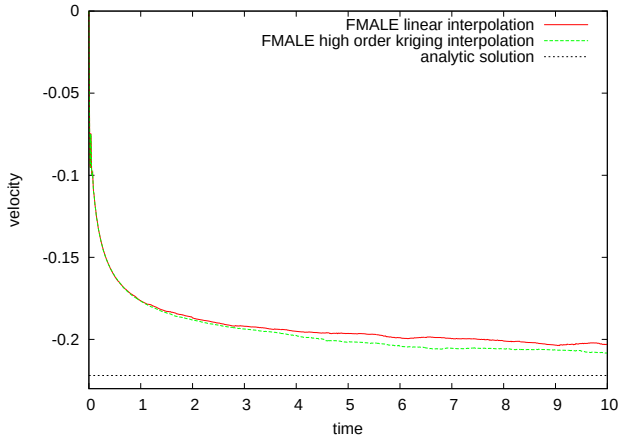

Figure 7.10: Linear and high order interpolation for the FMALE framework.

interpolation, we will compare a high order kriging with a linear interpolation for calculating the previous fluid velocities within the FMALE framework. Figure 7.10 shows that using a high order kriging algorithm produces a better solution than linear approximation.

Moderate Reynolds Numbers

Now, let us consider higher Reynolds numbers to solve the problem stated above. As shown in Figures 7.11 and 7.12, the difference in the velocities obtained with UBF and HNBF approaches becomes larger and larger as the Reynolds number grows. These numerical experiments show the better performance of the UBF with respect to the HNBF scheme when we compare them with the analytical solutions.

Although, only considering an infinity time of simulation and also a cylinder with an infinite height we can obtain a final solution for the simulation. However, we can assure that the UBF at least reach faster the solution than the HNBF scheme. And also, as shown above, the solution is smoother, especially when we consider the acceleration of the body.

We briefly explain how we determine the analytic solution now. As mentioned before, for very low Reynolds numbers a terminal velocity can be easily obtained thanks to the linear relationship between the drag force and the velocity of the rigid body. However, when the inertial effect cannot be neglected, as in the problems shown in Figures 7.11 and 7.12, the relationship is no longer linear and finding the terminal velocity requires an iterative solution. The details can be found in [60]. We will now analyze three further issues: the mesh convergence, the acceleration behavior, and the determination of the time step.

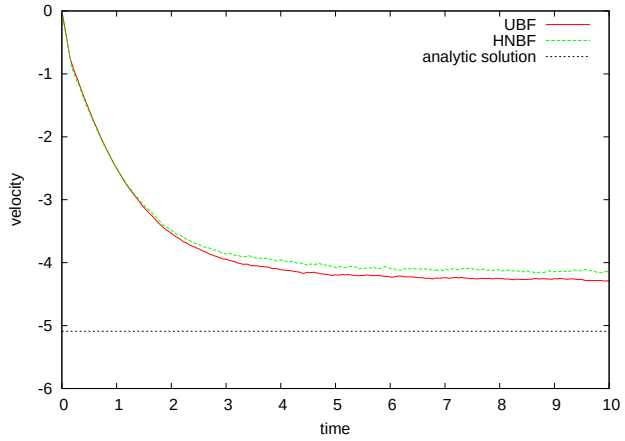

Figure 7.11: Numerical and analytical terminal velocity for Re = 101.

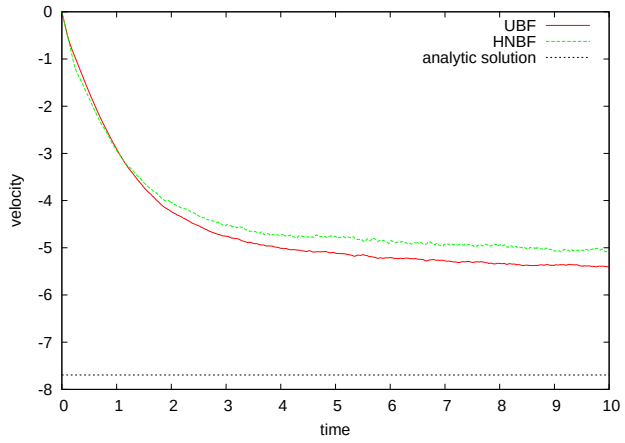

Figure 7.12: Numerical and analytical terminal velocity for Re = 1647.

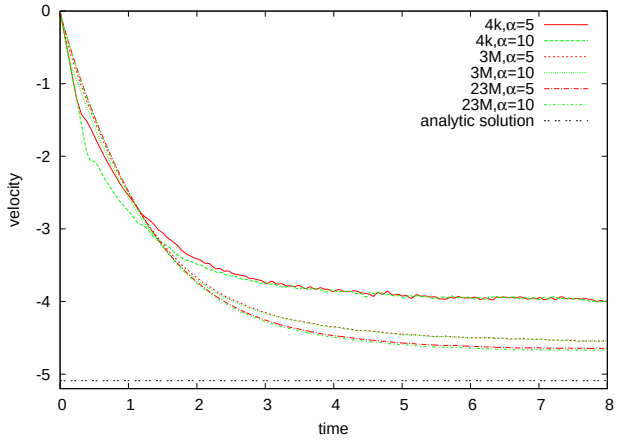

Figure 7.13: Numerical and analytical terminal velocity for Re = 101 using different meshes and safety factors α and considering only the HNBF approach.

Mesh convergence

Now, let us consider only the HNBF approach to carry out the mesh convergence. The reason for this choice is to improve the performance of the HNBF with respect to UBF scheme. In Figures 7.13 and 7.14 the difference between the velocities obtained with the HNBF scheme and the analytic ones becomes shorter and shorter as the mesh is refined. In particular, we use three different meshes of 400000, 3 millions, and 23 millions of elements. Also, the velocities in Figures 7.13 and 7.14 were obtained using different safety factors α.

The solution reached is specially improved for the flow with a Reynolds number of 1647, as shown in Figure 7.14. We start with a difference with respect to the analytic solution of 39.4% to finally obtain a difference of 13.5%. For the flow with a Reynolds number of 101 we have a initial difference of 21.6% and a final one of 7.6%.

Acceleration

We will consider a Reynolds number equal to 3.4, which entails an analytical solution for the terminal velocity equal to 1.8. The reason for this choice is to have a Reynolds number where the velocities reached by both the UBF and HNBF approaches are still very similar. Then, it is interesting to take a closer look at the results for the acceleration and velocity values. In Figure 7.15, in the inside plots, we display a zoom of the accelerations and velocities for the last time steps. The Figures confirm the better performance of the UBF approach, especially concerning the acceleration values.

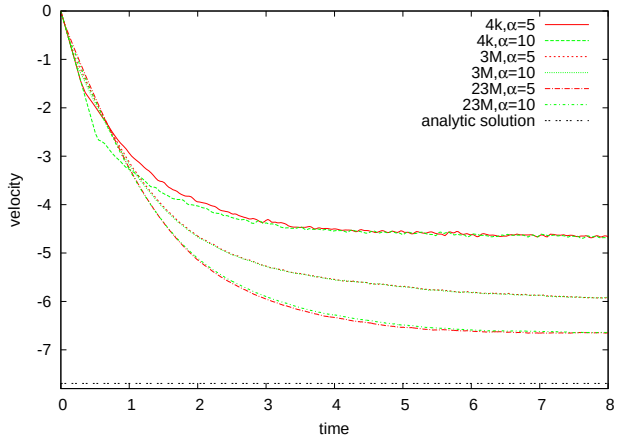

Figure 7.14: Numerical and analytical terminal velocity for Re = 1647 using different meshes and safety factors α and considering only the HNBF approach.

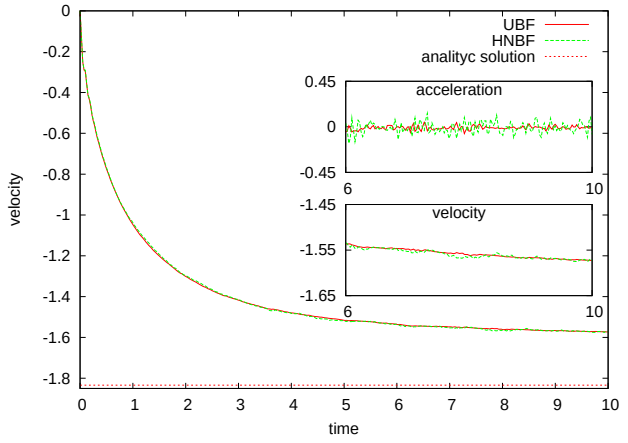

Figure 7.15: Solid acceleration and solid velocity for the UBF and HNBF approaches with Re=3.7.

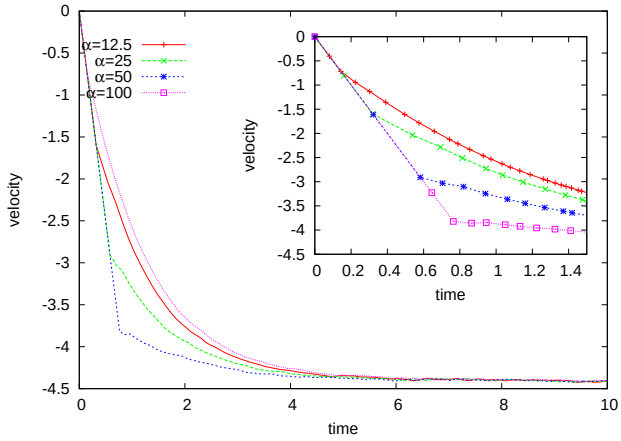

Figure 7.16: Time step analysis using different safety factors for the UBF scheme with Re=101.

Time step analysis

Now, consider again a Reynolds number equal to 101 and the UBF approach implementation. Then, following the methodology in Chapter 6, at each time step a critical time step value has to be estimated to solve the fluid and rigid body coupled problem. Figure 7.16 shows the results of solving the problem stated above with different safety factors α where $\Delta t = \alpha \times \Delta t_{cri}$ and Δt_{cri} is the critical time step for the NS solver. The inside plot in Figure 7.16 shows the times at which the results of the simulations were calculated. For $\alpha = 12.5$ and $\alpha = 25$ the Δt obtained for the NS equations is selected. However, for $\alpha = 50$ and $\alpha = 100$, the time step is limited by Δt_{NE}, defined in Subsection 6.2.4, in order to avoid that the solid steps over more than two elements during a time step. As we can see in the Figure 7.16, this limitation is only activated after the first time step as it is based on the previous time step solution (see the first step of the time loop in Algorithm 5).

We also observe that the terminal velocity is achieved quicker in the case of higher safety factors. This is the reason why the time step is smaller for $\alpha = 50$ and $\alpha = 100$ than for $\alpha = 25$, despite the fact that the safety factor is higher.

7.1.3 Vortex oscillations of a circular cylinder

The problem geometry is displayed in Figure 7.17. The circle represents the solid and its surface mesh is embedded inside the fluid mesh. The fluid has a viscosity $\mu = 0.01\,\mathrm{g\,(cm\,s)^{-1}}$ and a density $\rho = 1.0\,\mathrm{g\,cm^{-3}}$. The motion of the cylinder defines a linear spring-mass system with a stiffness $k = 5.79\,\mathrm{N\,m^{-1}}$ and a damping factor $c = 0.325\,\mathrm{g\,s^{-1}}$. The mass of the cylinder is $m = 2.979\,\mathrm{g}$ with

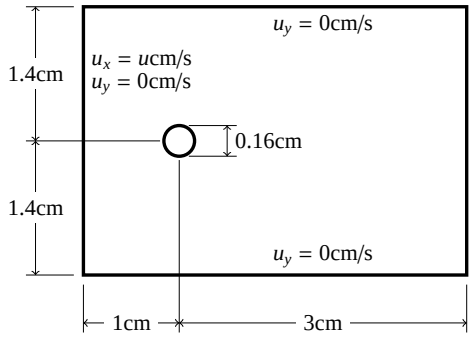

Figure 7.17: Problem domain definition.

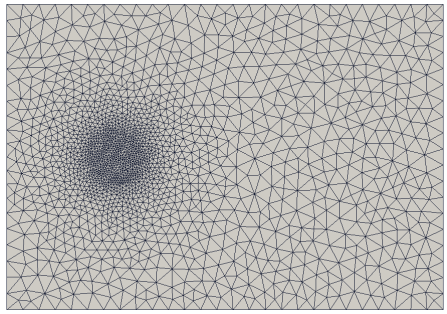

Figure 7.18: Discretization of the problem domain.

a circular section $D = 0.16\,\text{cm}$. The Reynolds number $Re = uD\rho/\mu$ ranges from 90 to 120 by changing the value of the inflow velocity u.

The mesh is unstructured and composed of 10000 triangular elements as shown in Figure 7.18. The time step is prescribed using $\Delta t_\text{p} = 0.001\,s$ in Equation (6.3). The portions of the mesh near the hole are shown in Figures 7.19 and 7.20 for the HNBF and UBF algorithms, respectively, at a given time step for an arbitrary Reynolds number.

The most interesting characteristic of the problem is the so-called "lock-in" phenomenon, which is captured for all the simulations with Reynolds numbers ranging from 90 to 120. The relative amplitudes Y/D, where Y is the displacement of the cylinder with respect to its original position, considering some Reynolds numbers, are shown in Figures 7.21 and 7.22 for the UBF and the HNBF implementations, respectively.

The values of the amplitudes for both algorithms and for all the simulations are shown in Figure 7.23. These values are compared with the experimental results obtained in [61] and the values shown by Dettmer et al. in [59] for

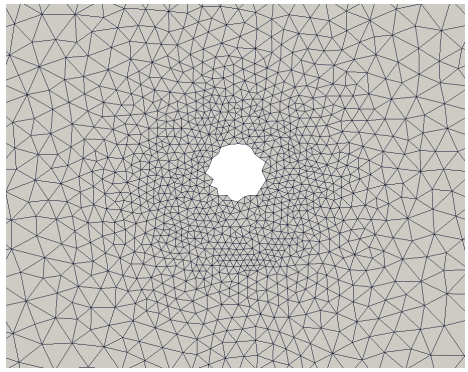

Figure 7.19: Mesh near the hole for the high order kriging interpolation algorithm.

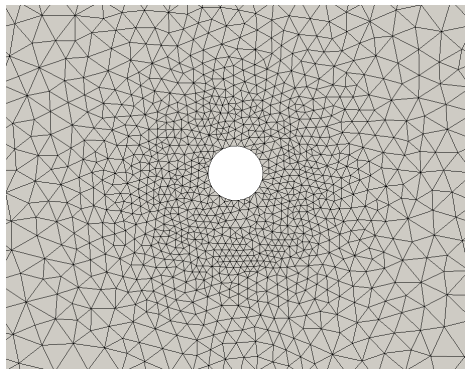

Figure 7.20: Mesh near the hole after applying the local r-adaptivity algorithm.

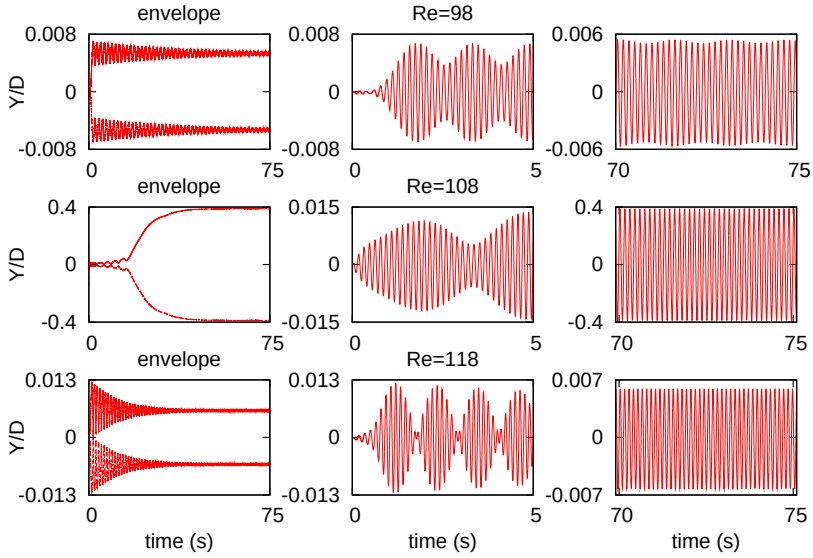

Figure 7.21: Amplitudes of the solid oscillations due to the vortex for the UBF algorithm. (Left) The envelope (curve outlining the extremes) of the amplitudes of the oscillations, created using the Hilbert transform. (Mid.) Initial amplitudes of the oscillations (Right) Final amplitudes of the oscillations.

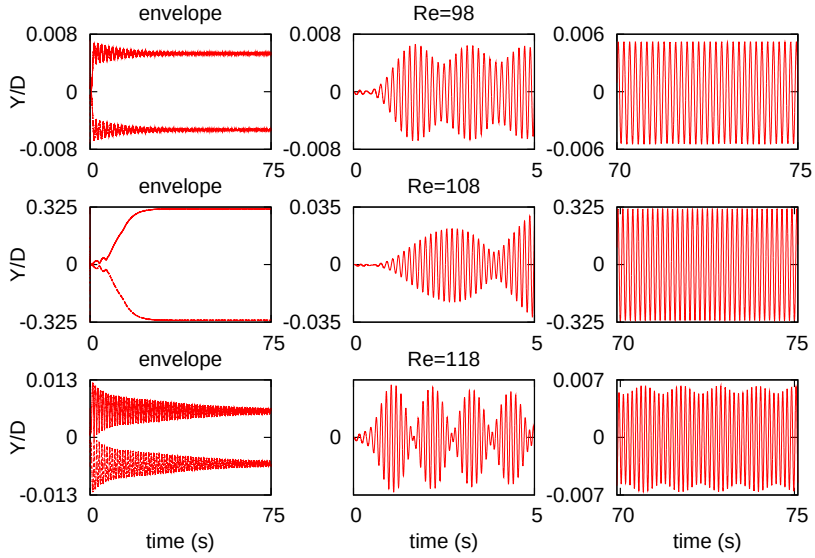

Figure 7.22: Amplitudes of the solid oscillations due to the vortex for the HNBF algorithm. (Left) The envelope (curve outlining the extremes) of the amplitudes of the oscillations, created using the Hilbert transform. (Mid.) Initial amplitudes of the oscillations. (Right) Final amplitudes of the oscillations.

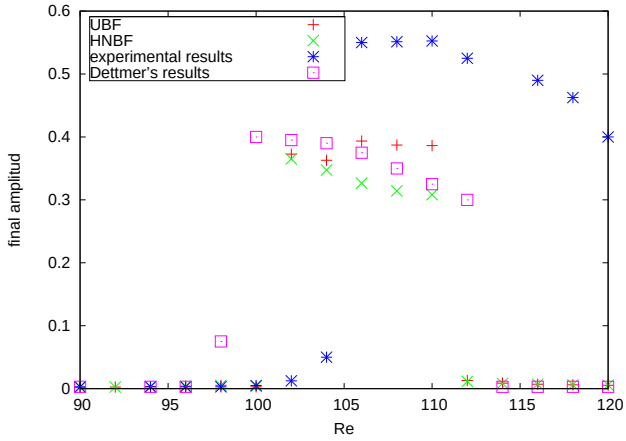

Figure 7.23: Amplitudes reached at the last time step for UBF and HNBF schemes compared to Dettmer's and experimental results.

a mesh of 3574 elements. In general, the amplitudes obtained with the UBF algorithm are larger than the amplitudes obtained with the HNBF algorithm and, what is more important, closer to the experimental results. In addition, the maximum amplitude obtained by Dettmer et al. in [59] is also closer to the maximum amplitude obtained by the UBF algorithm.

The vortex shedding frequency f_v with respect to the natural frequency of the cylinder-spring system f_n are shown in Figures 7.24 and 7.25. The frequencies obtained by both algorithms are very similar to the experimental results obtained in [61] and the frequencies shown Dettmer et al. in [59] for a mesh of 3574 elements.

7.1.4 Two Bileaflet Mechanical Heart Valves

A primary choice of artificial heart valves to replace the native ones when they are malfunctioning is the Bileaflet mechanical heart valves (BMHVs). These prostheses are made of a durable pyrolitic carbon material. The design includes a circular wall (referred as the housing wall) and two semicircular leaflets attached to the circular wall. Due to the blood flow (forward and reverse), and consequently different pressure levels on either side of the valves, the two leaflets are opened and closed.

In our simulation, the real problem was simplified as explained next at the geometrical and physical description of the simulated problem. However, as it is shown later, the results obtained of the simulation reproduce the experimental results obtained in [62].

The whole domain and two zooms near to the valves are shown in Figures 7.26, 7.27, and 7.28 respectively. The movement of the valves is schematized

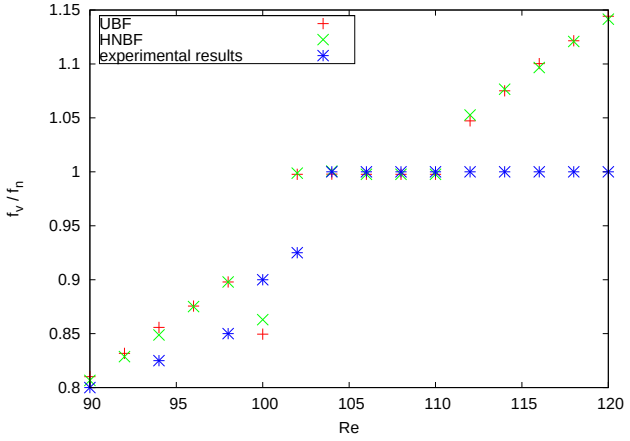

Figure 7.24: Frequencies reached at the last time step for UBF and HNBF schemes compared to experimental results.

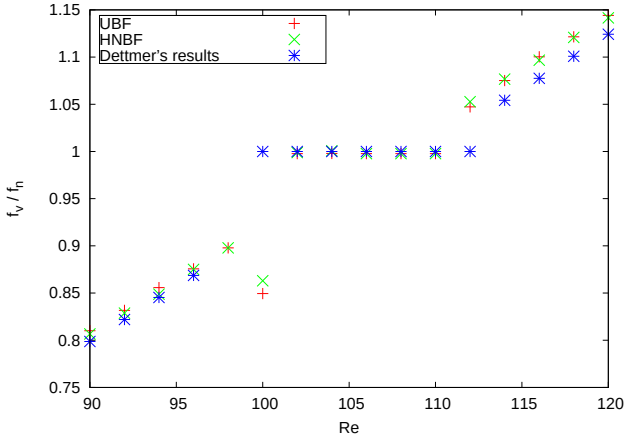

Figure 7.25: Frequencies reached at the last time step for UBF and HNBF schemes compared to Dettmer's results.

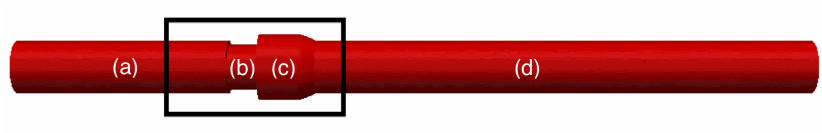

Figure 7.26: Domain of the two bileaflet mechanical heart valves. A zoom is done as shown in the square in Figure 7.27.

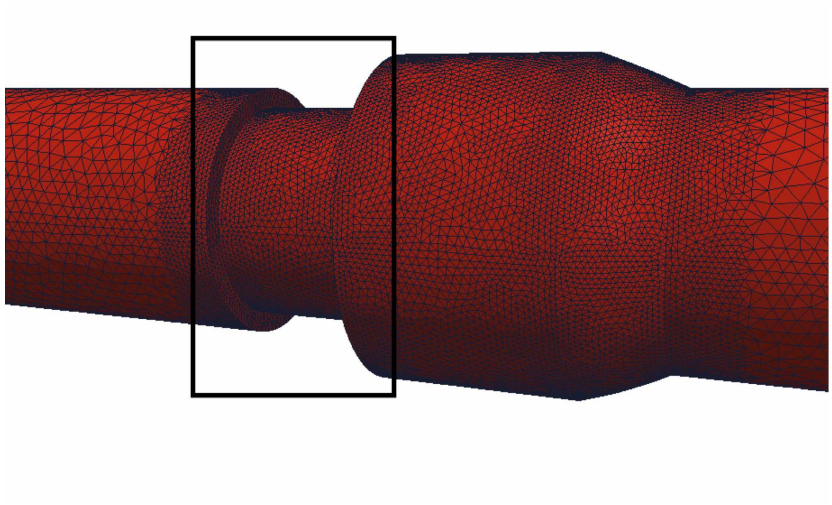

Figure 7.27: Zoom of the whole domain. Another zoom is done as shown in the square in Figure 7.28.

in Figure 7.28. The maximum and minimum angles of aperture of the valves are near 60° and 5° respectively. Both, the ventricular, see Figure 7.26(a), and aortic chamber, see Figure 7.26(d), have a diameter of $D = 25.4$cm. The chamber where the valves are located, see Figure 7.26(b), has a diameter of 21.4cm. The expansion of the aorta chamber to a diameter of 31.74cm, see Figure 7.26(c), represents the aortic sinus root. The total domain length is $16D$ where the ventricular chamber is $4D$ long. The pyrolitic carbon material of the valves has an approximated density of 1750kg/m. More details about the geometrical and physical description of the problem can be found in [63, 62, 64, 65, 66].

A plug flow profile based on the experimental data given in [62] is prescribed at the beginning of the ventricular chamber as the inflow boundary condition. The inflow profile is shown in Figure 7.29.

The Reynolds number of the flow varies from 0, when the valves are closed, to nearly 6000 when the valves are fully open. The complexity of the problem comes from the geometry and the flow that generates the incoming plug flow

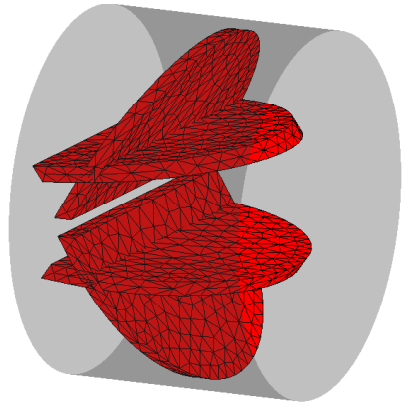

Figure 7.28: Maximum and minimum angles of aperture of the valves.

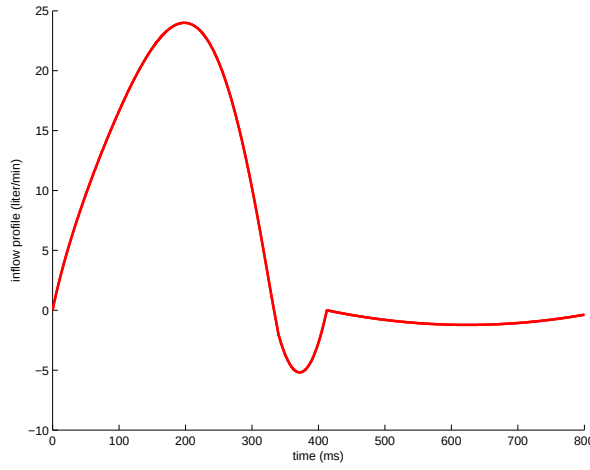

Figure 7.29: Plug inflow boundary profile.

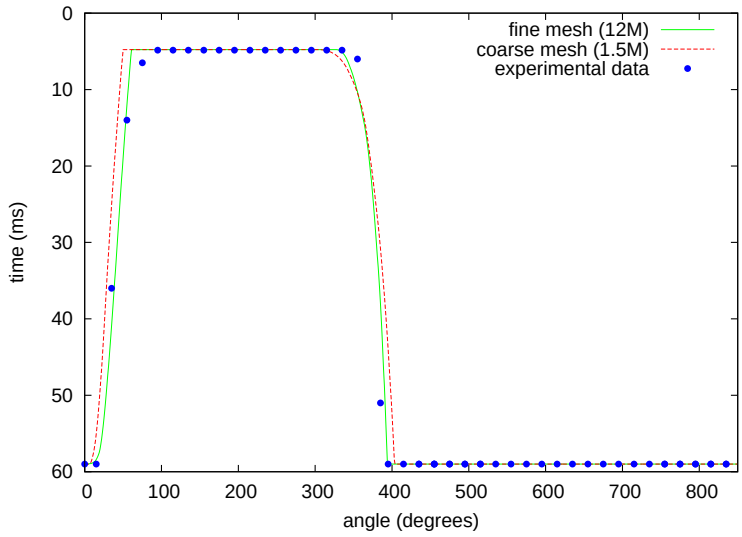

Figure 7.30: Aperture angle of the valves.

profile and interacts with this geometry.

In Figure 7.30, the angular displacement of the valves obtained with a coarse mesh of 1.5 millions of elements and a fine mesh of 12 millions of elements are compared to the experimental results obtained by Dasi et al [62]. The displacement of the valves are captured better with the fine mesh than the coarse mesh. In general, both simulations reproduce the experimental results. Note the large acceleration rates that the valves experiment to reach the minimum and maximum angles of aperture during the opening and closing phases.

The vorticity field at the plane of symmetry can be used to visualize the motion of the leaflets during a complete cycle as shown in Figure 7.31 at different inflow values. The plane of symmetry is perpendicular to the leaflets. During the acceleration phase, see Figure 7.31(a), the flow field remains laminar and is dominated by the vortex shedding from the leaflets and the circular wall of the mechanical heart valves. During the deceleration phase, see Figure 7.31(c),the vorticity field is characterized by the recirculation zone generated near the sinus root walls. Finally, at the closing phase, see Figure 7.31(d), the backflow induces the break down of the vortices and the closing of the leaflets. The numerical simulation reproduces the major features and behavior of the vortex field described in Dasi et al [62], although some difference exist.

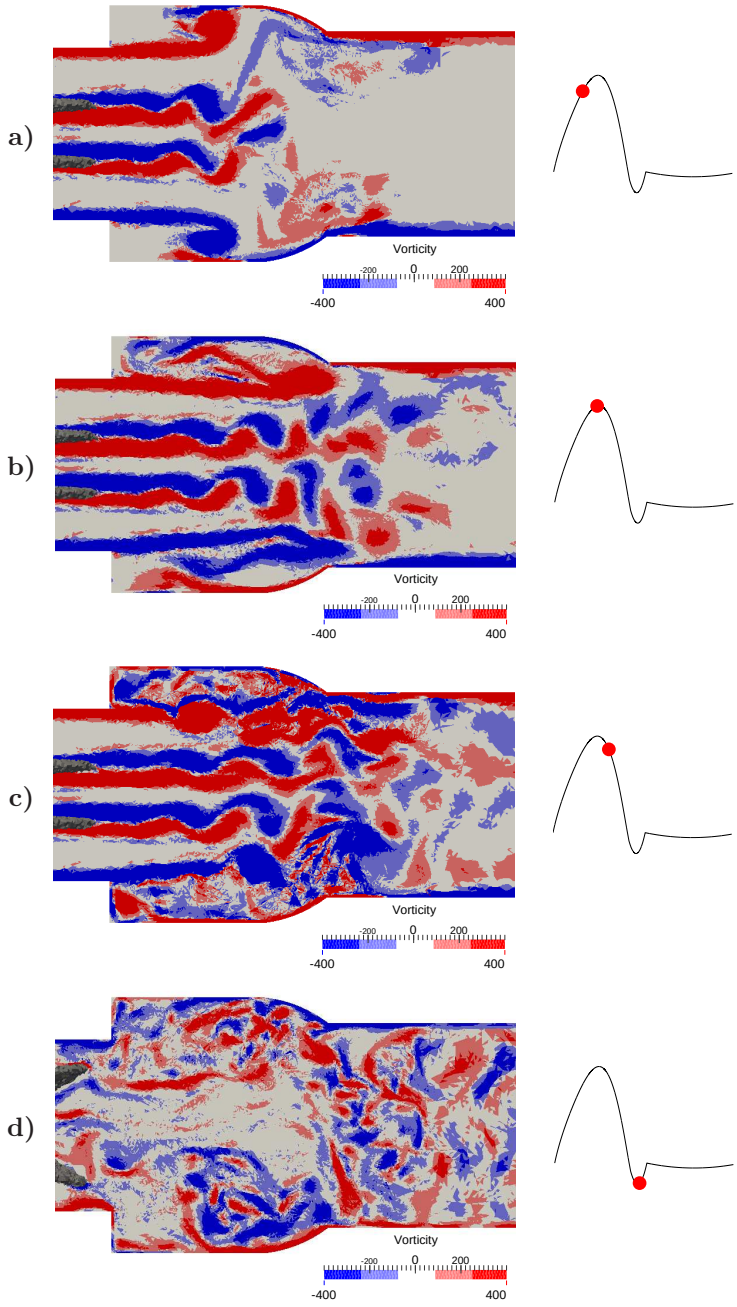

Figure 7.31: Vorticity field at the plane of symmetry at different time steps of the simulation.

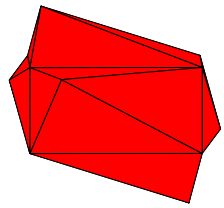

Figure 7.32: One of the solids with arbitrary shape.

7.1.5 Parallel performance of the UBF and NBF algorithms

Some all-to-all communications are necessary at different stages of the UBF and NBF algorithm, for example, to compute the force acting on the solids. With respect to the communications due to the fact that the bodies are stored in all the processors, we can say that they are very few compared to the ones needed for the NS iterative solver. Thus, the scalability of the code is not affected significantly.

In order to analyze the scalability of the implementation of the UBF and NBF approaches, take into account the following problem. There are twenty rigid solids with arbitrary shapes, see Figure 7.32, immersed inside a fluid. The domain of the fluid is a cube of side 100 and the boundary boxes of the rigid bodies are similar to cubes of side 5. The fluid density and viscosity are equal to 1.0 and 0.1 respectively. The solid density is equal to 5.0. The velocity in the fluid is imposed to be equal to zero at the side of the domain and negative one at the top in the z direction.

In a first set of runs, we only considered the NS equation solver implementation, in order to have a reference for the performance behavior of the UBF and NBF schemes. We then considered the UBF algorithm and finally the NBF algorithm.

The mesh uses 24 million elements, running in a range of processors that goes from 64 to 1024 (considering only integer powers of 2). It is important to mention that running in 1024 processors implies that each processor handles 23460 elements on average. This is an efficiency limit in terms of scalability, due to the fact that a small number of elements per processor implies that the weight of the communications in the total processing time becomes significant.

The scalability using the NS equations solver with and without considering the UBF and NBF algorithms is shown in Figure 7.33. As it can be observed, the scalability with respect to the NS equations solver acting alone is not affected significantly. We have intentionally fixed the number of solver iterations in order to compare the scalability of all the methods. For the momentum equations it was fixed to 25; whereas for the pressure equation, it was fixed to 100. These figures are sufficiently high to decrease the residual by several orders of magnitude with respect to the initial residual.

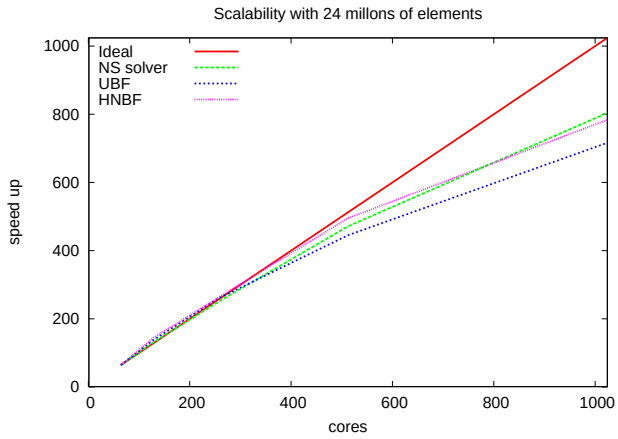

Figure 7.33: The scalability using the NS equations solver with and without considering the UBF and NBF algorithms.

7.2 Rigid bodies interaction

In order to prove the implementation of the general framework to solve the interaction between rigid solids, we consider three numerical examples. First, we focus in the prediction accuracy of the estimation time of collision and the collision resolution for a small group of cubes, less than 100 bodies, falling inside a funnel. Then, we will simulate a group of 10000 spheres falling inside a cube. In both examples we consider only elastic collisions and all the bodies have the same size.Finally, we solve the interaction between a group of 4000 spheres with different sizes and masses considering a high loss of energy in each collision. For a large number of bodies, as the last two examples, the bucket sort algorithm is considered, see Subsection 5.2.3. This algorithm allow us to drastically reduce the number of operations during a simulation.

7.2.1 50 squares falling into a funnel

In Figures 7.34 and 7.35, fifty squares are falling into a funnel. The squares have an initial linear and angular velocity imposed. All the collisions are elastic, that is, there is not a loss of energy in a collision. The number of contacts at the bottom of the funnel is much bigger than the number of contacts at the top.

7.2.2 10000 spheres falling inside a cube

Inside a cube, 100000 spheres fall due to the gravity force as shown in Figures 7.36 and 7.37. The quantity of contacts that the program has to solve is very high. This high frequency of collisions is a challenge for the collision detection

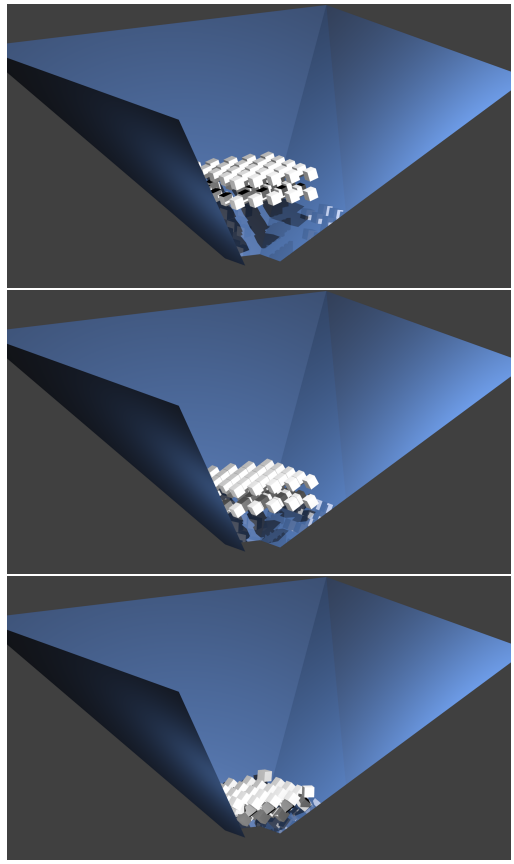

Figure 7.34: Fifty cubes falling into a funnel at the beginning of the simulation.

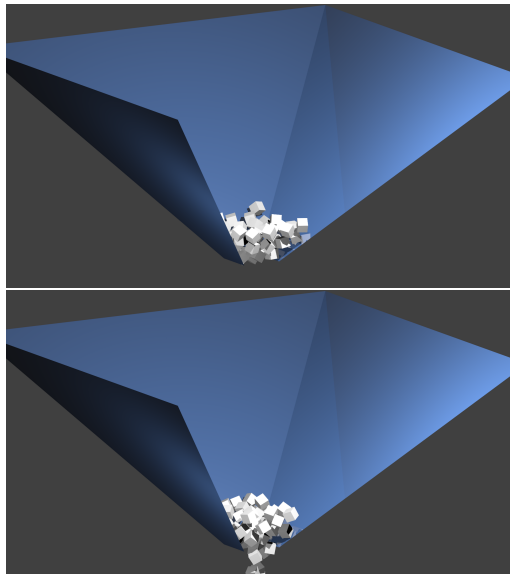

Figure 7.35: Fifty cubes falling into a funnel at the end of the simulation.

and collision response algorithms.

7.2.3 4000 spheres of different sizes crashing against the floor

A group of 4000 spheres with different sizes and masses crash against the floor in Figures 7.38 and 7.39. The spheres fall with a high initial linear velocity. The different masses of the spheres and the high loss of energy of the collisions increase the number of collisions of the whole system and complicate the estimated time of collision between the bodies. It is also complex for the bucket sort algorithm.

7.3 Fluid and rigid bodies interaction (collisions)

In this last section we solve two problems that include the fluid and rigid body interaction as well as the rigid bodies interacting among themselves.

In a first example, we will reproduce the *drafting, kissing and tumbling* phenomenon for two interacting spheres considering impulses to solve the contact between the spheres. The kinematics of the spheres reproduce the positions obtained in other publications. The same scenario will be considered for the second example; however, this time there will be eight spheres interacting with fluid. At the end, separation of spherical bodies in a rectangular microchannel will be simulated. The results are compared to analytic and experimental data.

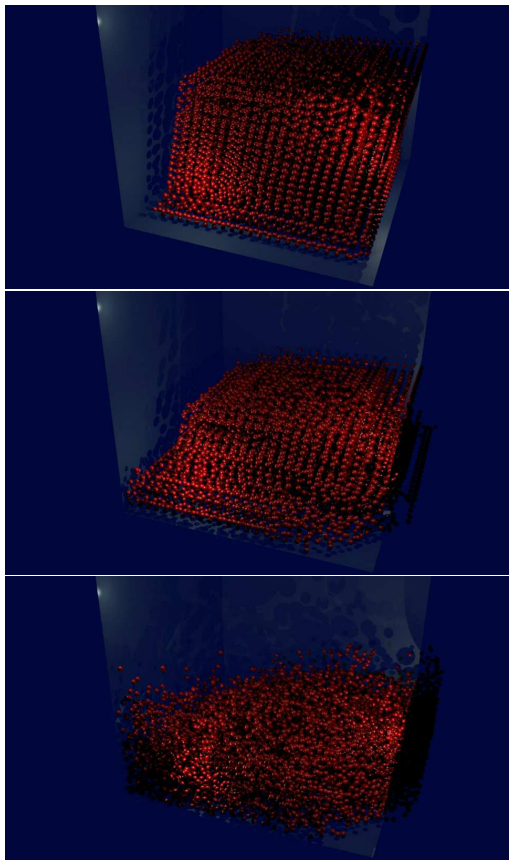

Figure 7.36: 10000 spheres falling inside a square at the beginning of the simulation.

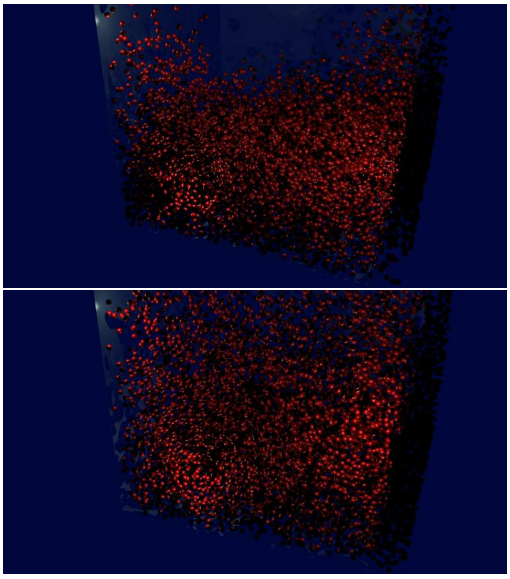

Figure 7.37: 10000 spheres falling inside a square at the end of the simulation.

7.3.1 Drafting, kissing and tumbling for two interacting spheres

Consider two spheres with densities of 1.14g/cm^3 and a radius of 0.083cm falling due to the gravity inside a fluid in a rectangular channel of 1cm $\times$ 1cm $\times$ 4cm. The spheres are located at the positions [0.5cm, 0.5cm, 3.5cm] and [0.5cm, 0.5cm, 3.16cm] inside the channel. The fluid has a density of $1.0g/cm^3$ and a viscosity of 0.01g/(cm s). The problem is discretized using a mesh of 5 millions of elements.

During the simulation, the sphere located at the top will increase its velocity respect to the sphere located at the bottom as a consequence of the lower drag that the top sphere experiments, the *drafting phenomenon*. At some point of the simulation both spheres will be in contact, the *kissing contact*. In a Newtonian fluid, this contact produces an unstable state and as result the particles *tumble*.

In Figure 7.40 we can see that the positions of the spheres at different time steps of the simulation in the z axis. The kinematics reproduces the positions obtained in [7] and [67] as shown in Figure 7.41.

7.3.2 Drafting, kissing and tumbling for more than two interacting spheres

In order to show that our implementation is capable of handling more spheres we consider the same domain and fluid described above now with eight spheres

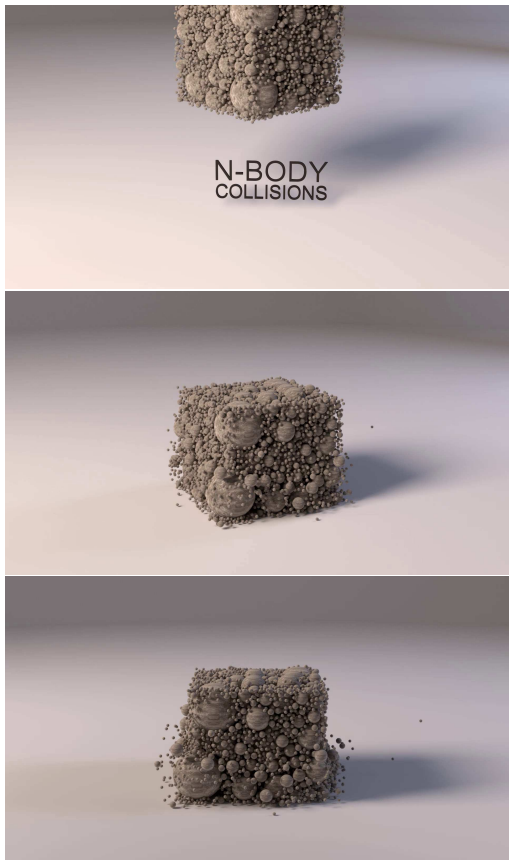

Figure 7.38: 4000 spheres crashing against the floor at the beginning of the simulation.

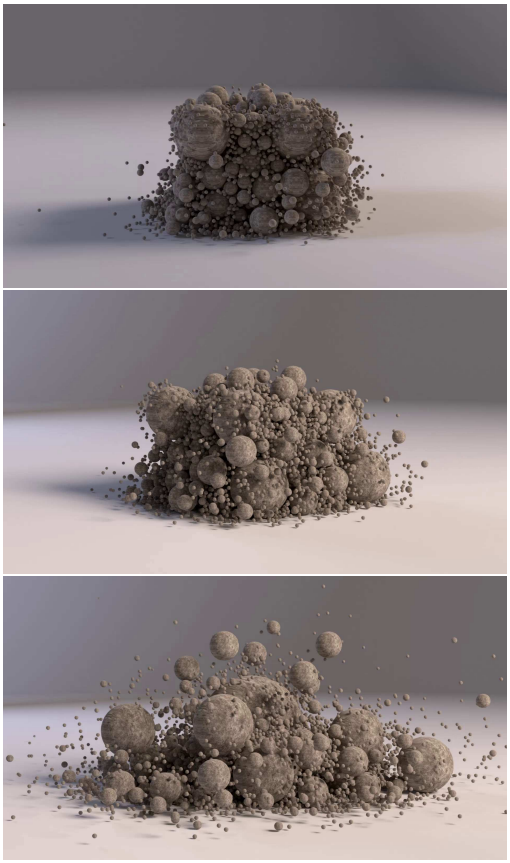

Figure 7.39: 4000 spheres crashing against the floor at the end of the simulation.

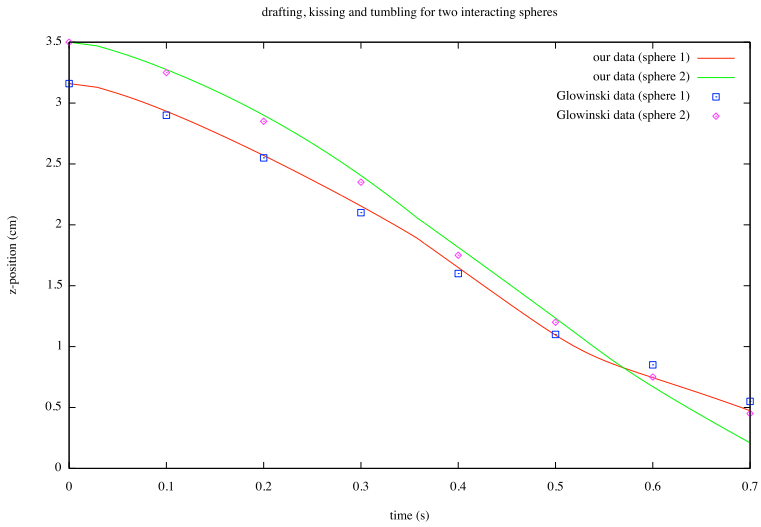

Figure 7.40: Comparison of positions of the spheres at different time steps of the simulation in the z axis obtained in our work and in [7].

inside the fluid. As before, the problem is discretized using a mesh of 5 millions of elements.

The positions of the spheres at different time steps of the simulation are shown in Figure 7.42.

7.3.3 Separation of bodies in square microchannels

It has been demonstrated that the inertial effects of the fluid flow in microchannels are important in many biomedical and environmental applications that include bodies separation, and bio-bodies focusing, see [68, 69].

For squares microchannels, spherical bodies tend to focus on four equilibrium positions considering laminar flows without any external force, see Figure 7.43. Inertial migration towards these four equilibrium positions is due to two lift forces exerted on the surface bodies:

- A "wall effect" force that moves the bodies away from the wall
- A shear gradient force that moves the bodies away from the center of the channel towards the wall.

Consider the square face perpendicular to the primary flow direction centered at $[0, 0]$. Then, the equilibrium positions will be located at $[0, p]$, $[0, -p]$, $[p, 0]$ and $[-p, 0]$, where $0 \geq p \geq h/2$ and h is the size of the square face side as illustrated in Figure 7.44. The focusing increases as the Reynolds number and the distance traveled by the bodies increase.

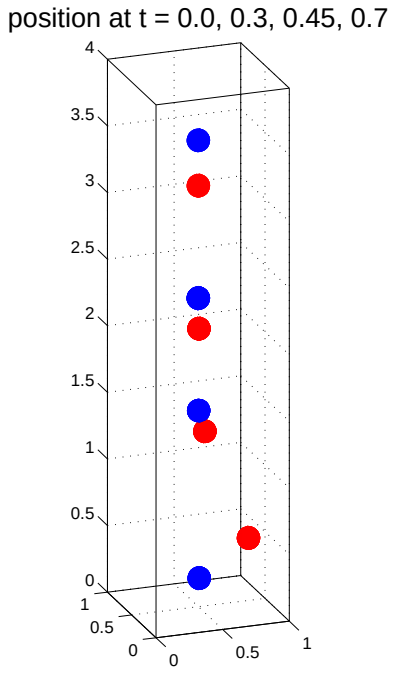

Figure 7.41: Positions of the spheres at different time steps of the simulation.

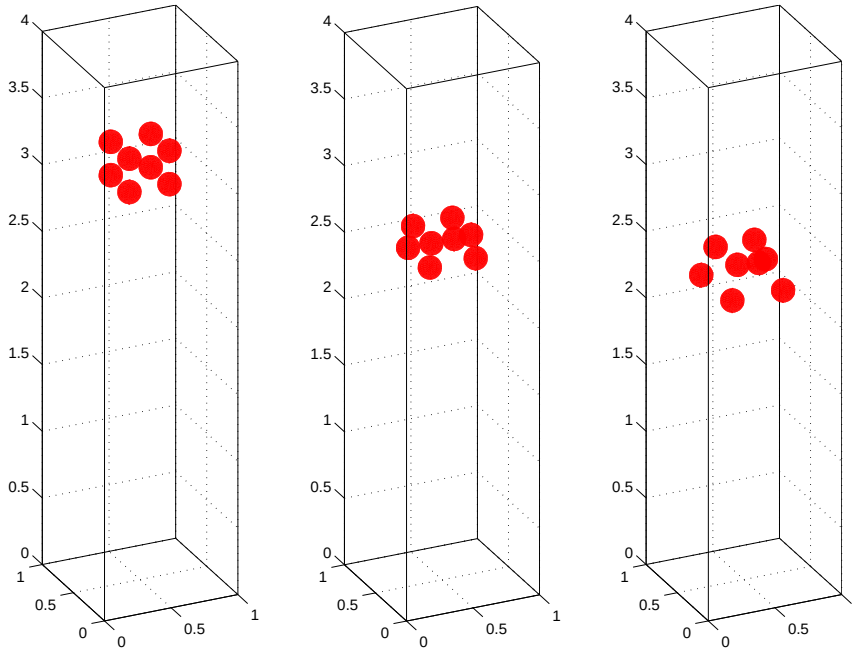

Figure 7.42: Positions of the spheres at the time steps 0, 0.20 and 0.25 of the simulation.

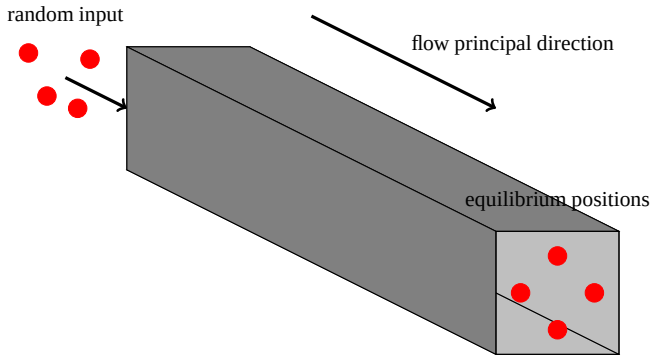

Figure 7.43: Spherical bodies focus at four equilibrium positions in squares microchannels.

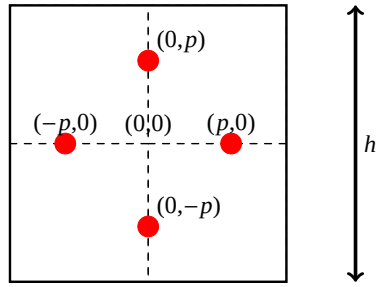

Figure 7.44: Equilibrium positions in the microchannel considering the square face perpendicular to the primary flow direction.

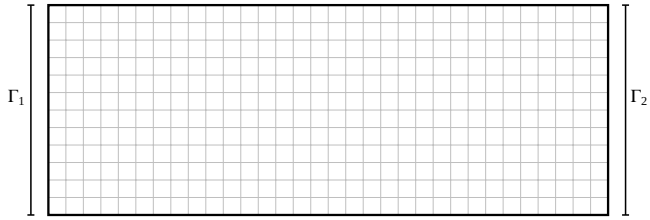

Figure 7.45: Considered periodic boundaries.

In our simulation, there are eight bodies with a diameter of 9μm inside a 50μm wide square channel that contains water. The Reynolds number of the problem is similar to 60. The geometry and physics details of the problem are obtained from [68].

In order to reduce the size of the problem, periodic conditions are considered for the bodies and the fluid. Suppose that the periodic conditions are imposed on boundaries Γ_1 and Γ_2 shown in Figure 7.45, then the velocity on Γ_1 is equal to the velocity on Γ_2. The final length of microchannel will be 200μm.

At the implementation level, the periodic conditions require that any node n that discretizes Γ_1 and Γ_2 has its whole element and node connectivities defined, $\mathcal{C}_{ele}(n)$ and $\mathcal{C}_{nod}(n)$ respectively. That is, any node n that discretizes Γ_1 has to add the connectivity of its corresponding node that discretizes Γ_2 and vice versa. The idea is illustrated in Figure 7.46. In a parallel context, the addition of the whole connectivities for periodic nodes is a complex issue and requires a careful modification of their connectivities.

Also, each body in the simulation has a copy of itself. This copy allow us to properly find the set of fringe nodes and impose the body boundary velocity on the fluid when the body is near or over the boundaries Γ_1 and Γ_2, see Figure 7.47. Thus, we actually have sixteen bodies in our simulation.

The discretization of the square channel uses 500000 hexahedral elements. The periodicity of the bodies across the channel is shown in Figure 7.48 at a given time of simulation. The positions of the bodies at the square face

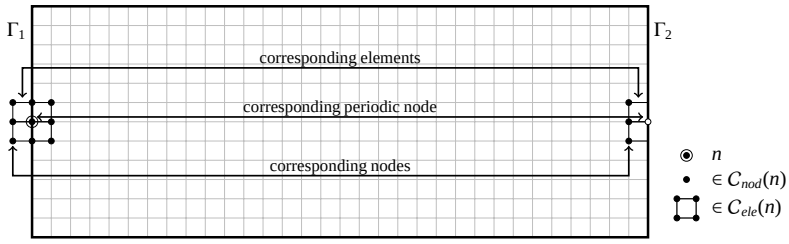

Figure 7.46: Added element and node connectivities for the periodic node n.

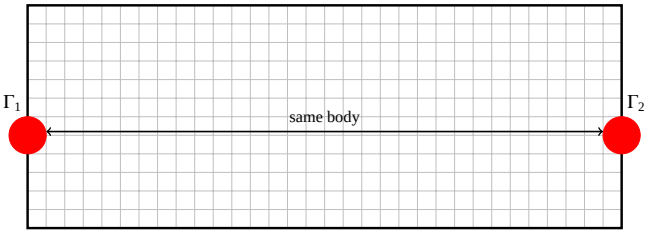

Figure 7.47: Body replication at the periodic boundaries.

of microchannel at the beginning and at the end of simulation are shown in Figure 7.49. The bodies positions are shown together with the four equilibrium positions obtained analytically in [70]. The positions at the begging of the simulation (top) and at the end of the simulation (bottom) are shown in Figure 7.50. All these results are similar to the results obtained in [68].

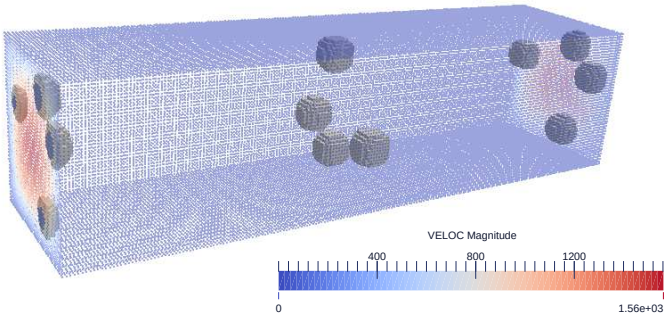

Figure 7.48: Bodies at the periodic boundaries during the simulation.

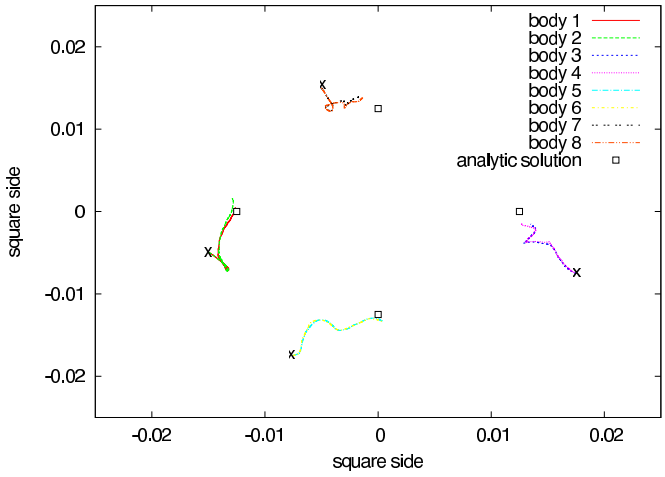

Figure 7.49: Positions of the bodies in the microchannel considering the square face perpendicular to the primary flow direction. The crosses indicate the positions at the beginning.

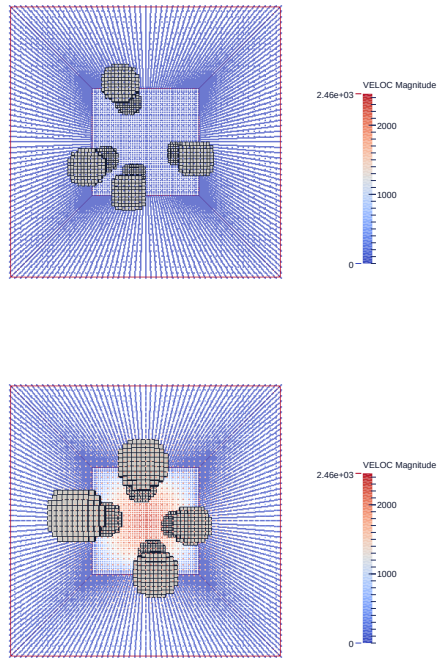

Figure 7.50: Positions of the bodies in the microchannel considering the square face perpendicular to the primary flow direction. (Top) Bodies at beginning of the simulation. (Bot.) Bodies at the end of the simulation.

8
Conclusions and future work

In this thesis, the main aim is to contribute to the numerical simulation of the interaction of a fluid and and a number of rigid bodies considering a distributed memory environment. The interaction is based on the embedded boundary mesh concept. Here, the fluid is discretized using a non body-conforming mesh and the boundaries of the bodies are embedded in this mesh and geometrically tracked by means of moving polyhedral surface meshes.

8.1 Achievements

Within an embedded boundary mesh context, two new approaches to deal with the interaction of a fluid and a rigid body have been presented. They basically differ in the way velocities from the solid are imposed on the fluid interface.

The first approach, an updated body-fitted one (UBF), implies the movement of nodes onto the body surface to conform with its current position at the previous time step. The second, a non body-fitted approach (NBF), uses interpolation to impose velocities on the rigid body surface on the fluid. In both cases, the FMALE framework is considered to deal with the new fluid nodes appearing at each time step. A new method of interpolation within this framework has been implemented. Also, the mass conservation is imposed by solving a minimization problem under a mass conservation constraint.

Both UBF and NBF new approaches have been tested by using numerical experiments and their accuracies have been studied. Regarding convergence, assessed by solving a manufactured solution example, the UBF approach seems to outperform the NBF one. However, the last method remains competitive whenever a high order interpolation is considered. Both methods are also capable of closely reproducing the final velocity of the Stokes problem. In a more complex example, the movement of a rigid body produced by resonance with the frequency of vortices is simulated. Both approaches are able to detect the initiation and describe the development of the body movement. Although it could be said that the UBF approach is more accurate in a general sense, the NBF approach usually gives reasonably accurate results too. In addition, it has to be mentioned that the last one is better in principle when considering computational cost and robustness. We also proof that the scalability of the fluid solver is not affected significantly considering both approaches.

More complex problems have been considered. The simulation of two bileaflet mechanical heart valves, with a complex geometry domain and where the Reynolds number of the flow varies from 0 to nearly 6000. The reproduction of the "drafting, kissing and tumbling" phenomenon that includes the resolution of the collisions between the bodies. And the separation of spherical bodies in a

rectangular microchannel. All the results obtained in these simulations have been compared with the data obtained in other studies.

In the implementation of these two new approaches, we include the solution of the interactions between the bodies. Although, all the subdomains simulate the interaction of all the particles and redundant work is done, the implementation has to be done in such way that each subdomain solves these interactions as fast as possible.

Also, a new framework for the fluid-structure interaction was described in a new formal definition using the set notation and considering a distributed memory environment. This framework can be generalized to other applications and allow us to elucidate the data structures and algorithms involved in a precise fashion.

8.2 Future Lines of Research

There are a lot of possibilities for research in the numerical simulation of the interaction of a fluid and a rigid body within an embedded boundary mesh context. Based on the work presented in this thesis, some suggestions for future research are presented below:

- To study more in depth the parallel behaviour of both approaches in order to improve their execution times.
- To develop numerical strategies (e.g. wall law) in order to be able to simulate turbulent flows.
- To improve the movement of nodes in order to allow the mesh to adapt to the boundary meshes of the bodies considering less levels of free nodes. Thus, the UBF approach will improve its robustness.
- To consider the possibility of remeshing in a distributed memory environment in order to improve the quality of the results for both approaches.

Bibliography

[1] C. Samaniego, G. Houzeaux, E. Samaniego, M. Vázquez, Parallel embedded boundary methods for fluid and rigid-body interaction, Computer Methods in Applied Mechanics and Engineering 290 (2015) 387–419.

[2] E. Casoni, A. Jérusalem, C. Samaniego, B. Eguzkitza, P. Lafortune, D. Tjahjanto, X. Sáez, G. Houzeaux, M. Vázquez, Alya: computational solid mechanics for supercomputers, Archives of Computational Methods in Engineering (2014) 1–20.

[3] H. Owen, G. Houzeaux, C. Samaniego, A. Lesage, M. Vázquez, Recent ship hydrodynamics developments in the parallel two-fluid flow solver alya, Computers & Fluids 80 (2013) 168–177.

[4] G. Houzeaux, H. Owen, B. Eguzkitza, C. Samaniego, R. de la Cruz, H. Calmet, M. Vázquez, M. Ávila, Developments in Parallel, Distributed, Grid and Cloud Computing for Engineering, Vol. volume 31 of Computational Science, Engineering and Technology Series, Saxe-Coburg Publications, 2013, Ch. Chapter 8: A Parallel Incompressible Navier-Stokes Solver: Implementation Issues, pp. 171–201.

[5] H. Owen, G. Houzeaux, C. Samaniego, F. Cucchietti, G. Marin, C. Tripiana, H. Calmet, M. Vázquez, Two fluids level set: High performance simulation and post processing, in: 2012 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC), IEEE, Salt Palace Convention Center, Salt Lake City, UT, 2012, pp. 1559–1568.

[6] G. Houzeaux, C. Samaniego, H. Calmet, R. Aubry, M. Vázquez, P. Rem, Simulation of magnetic fluid applied to plastic sorting, The Open Waste Management Journal 3 (2010) 127–138.

[7] R. Glowinski, T. W. Pan, T. I. Hesla, D. D. Joseph, J. Périaux, A fictitious domain approach to the direct numerical simulation of incompressible viscous flow past moving rigid bodies: Application to particulate flow, Journal of Computational Physics 205 (2001) 363–426.

[8] M. Behr, T. Tezduyar, The shear-slip mesh update method, Computer Methods in Applied Mechanics and Engineering 174 (3) (1999) 261–274.

[9] R. Codina, G. Houzeaux, Implementation aspects of coupled problems in cfd involving time dependent domains, Verification and Validation Methods for Challenging Multiphysics Problems (2006) 99–123.

[10] S. Feghali, E. Hachem, T. Coupez, Monolithic stabilized finite element method for rigid body motions in the incompressible navier-stokes flow: Monolothic sfem for fsi, European Journal of Computational Mechanics/Revue Européenne de Mécanique Numérique 19 (5-7) (2010) 547–573.

[11] C. Farhat, V. K. Lakshminarayan, An ale formulation of embedded boundary methods for tracking boundary layers in turbulent fluid–structure interaction problems, Journal of Computational Physics 263 (2014) 53–70.

[12] D. Owen, C. Leonardi, Y. Feng, An efficient framework for fluid–structure interaction using the lattice boltzmann method and immersed moving boundaries, International Journal for Numerical Methods in Engineering 87 (1-5) (2011) 66–95.

[13] T. Rabczuk, R. Gracie, J.-H. Song, T. Belytschko, Immersed particle method for fluid–structure interaction, International Journal for Numerical Methods in Engineering 81 (1) (2010) 48–71.

[14] S. Idelsohn, E. Onate, F. Del Pin, N. Calvo, Fluid–structure interaction using the particle finite element method, Computer Methods in Applied Mechanics and Engineering 195 (17) (2006) 2100–2123.

[15] F. Habbal, The optimal transportation meshfree method for general fluid flows and strongly coupled fluid-structure interaction problems, Ph.D. thesis, California Institute of Technology (2009).

[16] A. Quarteroni, A. Valli., Domain Decomposition Methods for Partial Differential Equations, Oxford Science, 1999.

[17] J. J. C. W. W. Charlesworth, D. C. Anderson, The domain composition method applied to poisson's equation in two dimensions, International Journal for Numerical Methods in Engineering 37 (1994) 3093–3115.

[18] G. Houzeaux, R. Codina, A chimera method based on a dirichlet/neumann(robin) coupling for the navier-stokes equations, Computer Methods in Applied Mechanics and Engineering 192 (31-32) (2003) 3343–3377.

[19] G. Houzeaux, B. Eguzkitza, R. Aubry, H. Owen, M. Vázquez, A chimera method for the incompressible navier–stokes equations, International Journal for Numerical Methods in Fluids 75 (3) (2014) 155–183.

[20] B. Eguzkitza, Hermesh: a geometrical domain composition method in computational mechanics, Ph.D. thesis, Universitat Politènica de Catalunya. Departament d'Arquitectura de Computadors (2014).

[21] C. A. Rivera, M. Heniche, F. Bertrand, R. Glowinski, P. A. Tanguy, A parallel finite element sliding mesh technique for the simulation of viscous flows in agitated tanks, International Journal for Numerical Methods in Fluids 69 (3) (2012) 653–670.

[22] S. Tanaka, K. Kashiyama, Ale finite element method for fsi problems with free surface using mesh re-generation method based on background mesh, International Journal for Numerical Methods in Fluids 20 (2006) 229–236.

[23] C. Peskin, Flow patterns around heart valves: a numerical method, Journal of Computational Physics 10 (1972) 252–271.

[24] R. Glowinsky, T. Pan, J. Périaux, A fictitious domain method for external incompressible viscous flow modelled by navier-stokes equations, Computer Methods in Applied Mechanics and Engineering 111 (1994) 133–148.

[25] P. A. T. F. Bertrand, F. Thibault, A three-dimensional fictitious domain method for incompressible fluid flow problems, International Journal for Numerical Methods in Fluids 25 (1997) 719–736.

[26] R. Löhner, J. D. Baum, E. Mestreau, et al., Adaptive embedded unstructured grid methods, International Journal for Numerical Methods in Engineering 60 (2004) 641–660.

[27] D. Schillinger, L. Dedè, M. A. Scott, J. A. Evans, M. J. Borden, E. Rank, T. J. Hughes, An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of nurbs, immersed boundary methods, and t-spline {CAD} surfaces, Computer Methods in Applied Mechanics and Engineering 249–252 (0) (2012) 116–150.

[28] T. Hughes, J. Cottrell, Y. Bazilevs, Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement, Computer Methods in Applied Mechanics and Engineering 194 (39–41) (2005) 4135–4195.

[29] R. Codina, G. Houzeaux, H. Coppola-Owen, J. Baiges, The fixed-mesh ale approach for the numerical approximation of flows in moving domains, Journal of Computational Physics 228 (5) (2009) 1591–1611.

[30] J. Baiges, R. Codina, H. Owen, The fixed-mesh ale approach for the numerical simulation of floating solids, International Journal for Numerical Methods in Fluids 67 (8) (2011) 1004–1023.

[31] A. S. Jan Bender, Constraint-based collision and contact handling using impulses, Proceedings of the 19th international conference on computer animation & social agents, Geneva (Switzerland).

[32] D. Baraff, An Introduction to Physically Based Modeling: Rigid Body Simulation II. Nonpenetration Constraints, SIGGRAPH Course Notes, 2001.

[33] T. Heister, A massively parallel finite element framework with application to incompressible flows, Ph.D. thesis, Niedersächsische Staats-und Universitätsbibliothek Göttingen (2011).

[34] W. Bangerth, C. Burstedde, T. Heister, M. Kronbichler, Algorithms and data structures for massively parallel generic adaptive finite element codes, ACM Trans. Math. Softw. 38 (2) (2012) 14:1–14:28.

[35] G. Houzeaux, J. Príncipe, A variational subgrid scale model for transient incompressible flows, International Journal of Computational Fluid Dynamics 22 (3) (2008) 135–152.

[36] T. J. R. Hughes, Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods, Computer Methods in Applied Mechanics and Engineering 127 (1995) 387–401.

[37] G. Houzeaux, M. Vázquez, R. Aubry, J. Cela, A massively parallel fractional step solver for incompressible flows, Journal of Computational Physics 228 (17) (2009) 6316–6332.

[38] G. Houzeaux, R. Aubry, M. Vázquez, Extension of fractional step techniques for incompressible flows: The preconditioned orthomin(1) for the pressure schur complement, Computers & Fluids 44 (2011) 297–313.

[39] R. Löhner, F. Mut, J. Cebral, R. Aubry, G. Houzeaux, Deflated preconditioned conjugate gradient solvers for the pressure-poisson equation: Extensions and improvements, International Journal for Numerical Methods in Engineering 87 (2011) 2–14.

[40] O. Soto, R. Löhner, F. Camelli, A linelet preconditioner for incompressible flow solvers, International Journal of Numerical Methods for Heat & Fluid Flow 13 (1) (2003) 133–147.

[41] T. C. Fung, Numerical dissipation in time-step integration algorithms for structural dynamic analysis, Progress in Structural Engineering and Materials 5 (3) (2003) 167–180.

[42] B. Mirtich, Fast and accurate computation of polyhedral mass properties, J. Graph. Tools 1 (2) (1996) 31–50.

[43] C. Hecker, Physics, Part 4: The Third Dimension, Game Developer Magazine, 1997.

[44] D. Baraff, An Introduction to Physically Based Modeling: Rigid Body Simulation I. Unconstrained Rigid Body Dynamics, SIGGRAPH Course Notes, 2001.

[45] F. Schornbaum, A real–time capable impulse–based collision response algorithm for rigid body dynamics, Master's thesis, Friedrich–Alexander–Universität Erlangen-Nürnberg (2010).

[46] B. V. Mirtich, Impulse–based dynamic simulation of rigid body systems, Ph.D. thesis, University of California at Berkeley (1996).

[47] B. V. Mirtich, Impulse-based dynamic simulation of rigid body systems, Ph.D. thesis, University of California, Berkeley (1996).

[48] A. Khamayseh, A. Kuprat, Deterministic point inclusion methods for computational applications with complex geometry, Computational Science & Discovery 1.

[49] E. Dyllong, W. Luther, W. Otten, An accurate distance-calculation algorithm for convex polyhedra, Reliable Computing 5 (1999) 241–253.

[50] M. W. Heinstein, F. J. Mello, S. W. Attawaya, T. A. Laursen, Contact-impact modeling in explicit transient dynamics, Computer Methods in Applied Mechanics and Engineering 187 (2000) 621–640.

[51] C. Förster, W. A. Wall, E. Ramm, Artificial added mass instabilities in sequential staggered coupling of nonlinear structures and incompressible viscous flows, Computer methods in applied mechanics and engineering 196 (7) (2007) 1278–1293.

[52] P. Causin, J.-F. Gerbeau, F. Nobile, Added-mass effect in the design of partitioned algorithms for fluid–structure problems, Computer methods in applied mechanics and engineering 194 (42) (2005) 4506–4527.

[53] D. A. Field, Laplacian smoothing and delaunay triangulations, Communications in Applied Numerical Methods 4 (6) (1988) 709–712.

[54] S. H. Lo, A new mesh generation scheme for arbitrary planar d omains, International Journal for Numerical Methods in Engineering 21 (8) (1985) 1403–1426.

[55] C. V. Deutsch, Geostatistical Reservoir Modeling, Oxford University Press, 2002.

[56] D. Y. Le Roux, C. A. Lin, A. Staniforth, An accurate interpolating scheme for semi-lagragian advection on an unstructured mesh for ocean modelling, Tellus (1997) 119–138.

[57] G. Houzeaux, R. Codina, Finite element modeling of the lost foam casting process tackling back-pressure effects, International Journal of Heat and Fluid Flow 16 (5) (2005) 573–589.

[58] G. Houzeaux, R. Codina, Transmission conditions with constraints in finite element domain decomposition method for flow problems, Communications in Numerical Methods in Engineering 17 (2001) 179–190.

[59] D. P. W. Dettmer, A computational framework for fluid-rigid body interaction: Finite element formulation and applications, Computer Methods in Applied Mechanics and Engineering 195 (2006) 1633–1666.

[60] P. P. Brown, D. F. Lawler, Sphere drag and settling velocity revisited, Journal of Environmental Engineering 129 (3) (2003) 222–231.

[61] P. Anagnostopoulos, P. Bearman, Response characteristics of a vortex-excited cylinder at low reynolds numbers, Journal of Fluids and Structures 6 (1) (1992) 39–50.

[62] L. P. Dasi, L. Ge, , H. A. Simon, F. Sotiropoulos, A. P. Yoganathan, Vorticity dynamics of a bileaflet mechanical heart valve in an axisymmetric aorta, Physics of Fluids 19, 067105 (2007) 1–17.

[63] B. Min Yun, C. K. Aidun, A. P. Yoganathan, Blood damage through a bileafletmechanicalheart valve: A quantitative computational study using a multiscale suspension flow solver, Journal of Biomechanical Engineering 136, 101009 (2014) 1–17.

[64] I. Borazjani, L. Ge, F. Sotiropoulos, Curvilinear immersed boundary method for simulating fluid structure interaction with complex 3d rigid bodies, Journal of Computational Physics 227 (2008) 7587–7620.

[65] K. Dumont, J. Vierendeels, R. Kaminsky, G. Van Nooten, P. Verdonck, D. BLUESTEIN, Comparison of the hemodynamic and thrombogenic performance of two bileaflet mechanical heart valves using a cfd/fsi model, Journal of Biomechanical Engineering-Transactions of The Asme 129 (4) (2007) 558–565.

[66] H. L. Leo, An in vitro investigation of the flow fields through bileaflet and polymeric prosthetic heart valves, Ph.D. thesis, Georgia Institute of Technology (2005).

[67] N. Sharma, N. A. Patankar, A fast computation technique for the direct numerical simulation of rigid particulate flows, Journal of Computational Physics 205 (2005) 439–457.

[68] D. Di Carlo, D. Irimia, R. G. Tompkins, M. Toner, Continuous inertial focusing, ordering, and separation of particles in microchannels, Proceedings of the National Academy of Sciences 104 (48) (2007) 18892–18897.

[69] D. Di Carlo, Inertial microfluidics, Lab Chip 9 (2009) 3038–3046.

[70] E. S. Asmolov, The inertial lift on a spherical particle in a plane poiseuille flow at large channel reynolds number, Journal of Fluid Mechanics 381 (1999) 63–87.