Professur für
Thermofluiddynamik

BACHELOR THESIS

Exploration of Smoothed Particle Hydrodynamics Techniques for the Simulation of Floating Bodies

Autor:
Álvaro Tomás Gil

Matrikel-No:
03700060

Betreuer:
Dr. Camilo Fernando Silva Garzon

August 13, 2018

Professur für Thermofluiddynamik
Prof. Wolfgang Polifke, Ph. D.

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst zu haben. Ich habe keine anderen Quellen und Hilfsmittel als die angegebenen verwendet.

Ort, Datum	Álvaro Tomás Gil

Contents

1 Introduction

The development of any reliable scientific simulation requires physically sound assumptions in order to ensure that the results provided by such simulation tool are as reliable as possible. Otherwise, the utility of the implemented simulation is hindered by the fact that the reality that it recreates does not correspond with the real case under study. Such is the case of implementations of Smoothed Particle Hydrodynamics (SPH) techniques. Although these simulation methods can provide extremely aesthetically pleasing reproductions of fluid movement, it is of considerable significance for the engineer employing these methods to ensure that these recreations are as plausible as possible.

In the case of the simulation of a free fluid surface, one can obtain immensely realistic results when correctly employing a Lagrangian scheme such as SPH, partly because in Lagrangian flow fields, the fluid is described by observing the individual motion of a fluid parcel in space and time, thus allowing for a more precise definition of the fluid free surface [2]. This is specially advantageous when compared with Eulerian approaches such as the implementation of the Shallow Water Equations, which require the computation of the free surface elevation of the fluid for every point of the computational domain [22].

In this manner, the objective of this study is to implement a 2D SPH simulation of a free fluid surface interacting with a floating body, and to develop this simulation taking into account the significance of the verisimilitude of the results. The plausibility of the obtained results is observed mainly by analyzing the structure of every component of the simulation structure in contrast with the relevant sources, and by validating the methods employed with a standard case scenario.

This report continues by presenting an introduction to the basic principles of SPH and to the basic structure of any SPH Solver, and by further describing and discussing the form that each of the components of the simulation has adopted. Within these descriptions of the elements of the simulation, a special depth will be added to the presentation of the methods of inclusion of the rigid body within the simulation. Later, the results of the simulated cases will be presented and compared. These same results will later be validated by analyzing the resolution of a specific case scenario involving a floating body, in order to end with an evaluation of the study as a whole.

2 Introduction to SPH

SPH simulations describe the fluid domain in a Lagrangian manner, such that they intend to describe a continuous scalar or vector field by a set of discrete entities carrying the properties of the field with them. Bearing this in mind, the value of the corresponding continuous field at a given point is described by an interpolation of the entities surrounding such location [18]. In this way, the discrete variables appearing at each entity are "smoothed out" throughout the spatial domain, using an interpolating kernel function $W(\vec{r}_i - \vec{r}_j, h)$ which can be integrated along the domain Ω in order to obtain a certain quantity A at $\vec{r}$.

$$A(\vec{r}) = \int_{\Omega} A(\vec{r}')W(\vec{r} - \vec{r}', h)d\vec{r}' \tag{2.1}$$

Where h is the smoothing length or support radius of the kernel function, which determines how large the range of influence between discrete entities is. However, Eq. (2.1) can be further simplified taking into account that the contributions to $A(\vec{r})$ are performed by discrete entities at concrete locations in the domain [18]. Thus, a quantity A at $\vec{r}_i$ is computed as a function of the known quantities A_j of the neighboring discrete entities.

$$A(\vec{r}_i) = \sum_j \frac{m_j}{\rho_j} A_j W_{ij}(\vec{r}_i - \vec{r}_j, h) \tag{2.2}$$

Note that as can be seen in Eq. (2.2), the role of the chosen kernel function is to interpolate the contribution of each discrete entity j to a certain location $\vec{r}_i$ based on the relation of the distance between them to the support radius h. In this way, it is reasonable to employ kernel functions similar to the Gaussian Kernel, but with a compact support, such that their values vanish after a certain distance [17].

In SPH, the discrete entities describing the continuous domain are defined by dividing the fluid into particles, each having a corresponding mass, pressure, velocity and position. Bearing this in mind, one can compute the density of a particle i in the following manner.

$$\rho(\vec{r}_i) = \rho_i = \sum_j m_j W_{ij}(\vec{r}_i - \vec{r}_j, h) \tag{2.3}$$

Once the fundamental principle of interpolation in SPH has been described, it follows to describe the basic equations governing the motion of each particle. While as in an Eulerian formulation, one would describe the fluid employing the continuity equation describing the conservation of mass as well as the Navier-Stokes equations to ensure conservation of momentum, the implementation of a particle-based approach simplifies these equations substantially [20]. Since the number of particles is controlled throughout the simulation, the continuity equation can generally be omitted in SPH applications. Moreover, due to the employed Lagrangian scheme, the Navier-Stokes equations are transformed into an equation

describing the variation of the particle's linear momentum as a sum of the forces acting on it [20].

$$m_i \frac{\partial \vec{v}_i}{\partial t} = \vec{F}_i^v + \vec{F}_i^s + \vec{F}_i^g + \vec{F}_i^p \tag{2.4}$$

Where in this specific form, the forces assumed to act on the body are those due to viscosity, pressure, surface tension, and gravity. As is reasonable, each of these forces will depend on the density, pressure, position and velocity of the particle i, as well as of such properties of every neighboring particle j. Note that the inclusion of a rigid body in the simulation will further affect the form of these forces.

Bearing in mind the general computation of the particle motion, one can go on to list the generic components making up an SPH simulative method. An essential element in any SPH simulation is the Nearest Neighbor Searching Algorithm. As any physical property of a fluid particle in the simulation is intrinsically related to the particles within influence range of itself, it is crucial to adequately and efficiently determine which are the particles which can be labelled as influencing neighbours. This algorithm can greatly influence the computational cost of the simulation, and is considered the most time-consuming part of SPH simulations [13]. Another repeating element of SPH simulations is the calculation of advective forces, i.e. forces not related with pressure. The models selected defining these forces can greatly affect the quality of the reproduced fluid behaviour. Moreover, the calculation of particle pressures and pressure forces is a step of the SPH simulation for which multiple different alternatives appear, presenting as well a large effect on the results of the simulation. Lastly, in order to properly enclose the fluid particles within the spatial domain under study, one has to make sure that the spatial boundaries are imposed and handled accordingly.

In addition to these generic building blocks of a SPH-based algorithm, the incorporation of a rigid body into the case study is a step which bears special significance in relation to this particular study.

3 Nearest Neighbor Search Algorithm

In the calculation of any system variable of a particle i, one has to find the j neighboring particles which may be within a search area of radius h. There exist several algorithms to perform this task, of which two variants were implemented in this study. The main principle of these two variants is to divide the spatial domain into a uniform grid. Once this division is performed and each particle is associated to a grid cell, each of the relevant neighboring cells of each cell are queried and the particles within them stored as potential neighbors. Uniform grid schemes are most efficient for SPH algorithms with constant kernel support radius, as the construction of a uniform grid costs $O(n)$ while the querying of a specific object implies $O(1)$, where n is the number of cells in the grid [13]. This is specially advantageous when compared with hierarchical data structures such as kd-trees, which are built in $O(nlog(n))$ and accessed with cost $O(log(n))$, thus being used mostly for variable kernel support simulations [16].

The selection of an appropriate grid cell size is crucial when determining the performance of the method employed. While as by choosing a cell size smaller than the kernel support radius, one may obtain a set of relevant neighboring cells to query which better approximates a sphere of radius h, the number of cells to query is increased and thus the performance is reduced [13]. On the other hand, implementing a uniform grid with cell size larger than the kernel support length would imply that less cells are queried, but a larger number of potential neighboring particles have to be stored. Note that a part of these stored potential neighbors will be outside of the influence radius of the respective particle, but their influence will disappear once the kernel function is employed. Thus, an optimal grid cell size is the kernel function support radius h [13]. With such a cell size, only 9 (in 2D) or 27 (in 3D) neighboring cells have to be queried.

The main difference between the two implemented uniform grid sort methods is the way in which each cell is labelled, and consequently to which cell index each particle is related. Based on this, one can develop an array of particle locations that is sorted based on the cell index. In this way, instead of each cell storing references to all of its particles, the method is improved by allowing the cell to store only the index of its first particle in the sorted array [16].

The basic grid sorting method simply sorts each grid cell based on the coordinates of their center with regards to the bottom left corner of the domain.

$$n = floor(\frac{x}{h}) + (\frac{l_x}{h}) floor(\frac{y}{h}) \tag{3.1}$$

In Eq. (3.1), one can see how the calculation of the cell index in the basic sorting scheme is performed, where l_x is the length of the domain in X direction, h is the support length of the kernel function, and (x, y) are the coordinates of the cell center. A disadvantage of this basic sorting method is that although particles of the same cell are close in memory, particles in neighboring cells are not necessarily near in the sorted array. In order to improve this aspect,

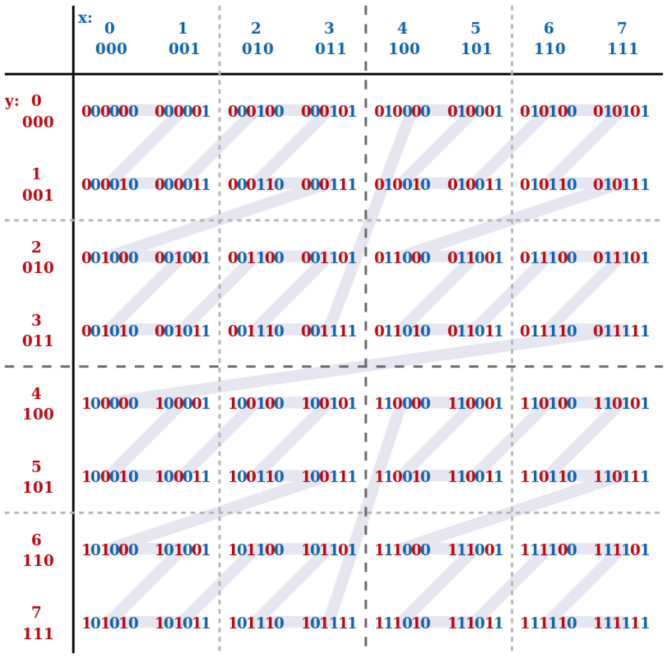

Figure 3.1: Four-level Z-order Curve [4]

the spatial locality of the particles can be mapped into the memory with a Z-curve sorting method [16].

The Z-curve sorting method employs a space-filling Z-order curve in order to label each cell based on 2-dimensional blocks of 4 cells. In order to understand this method, it is essential to describe the nature of a Z-order curve. A Z-order curve is described by firstly generating a discrete set of uniformly ordered points, (such as the centers of each cell of the uniform grid) and assigning normalized coordinates to each of these points, such that each point is described by two non-negative integer coordinates. Bearing this in mind, the binary representations of the coordinates of each point are interleaved, in order to generate the binary form of the point order, or in this case, the cell index [4]. This procedure can be visualized in Fig. 3.1. Then, the Z-curve sorting method employs this sort of spatial ordering to index the cells of the uniform grid.

Although the latter method preserves spatial locality and improves the algorithm's performance for the processing and querying of particle neighbors [16], after implementing both presented methods in the current study, a comparison based on the computation speed revealed the basic grid sorting method to be more favorable. This difference in performance can be seen in Fig. 3.2, where the implemented basic grid sorting method allowed for lower computation times for large numbers of particles. Another important factor that tipped the scales in favor of the basic grid sorting method is that it allowed for a larger flexibility of domain sizes, since employing indexing based on a continuous Z-curve required that the domain had an equal number of cells in X as in Y, and for this number of cells to be an exponent of 2.

Even though the presented grid sorting methods were considered adequate for this study, it is worth noting that in these schemes, the memory consumption scales with the size of the computational domain, where other methods such as spatial hashing or compact hash-

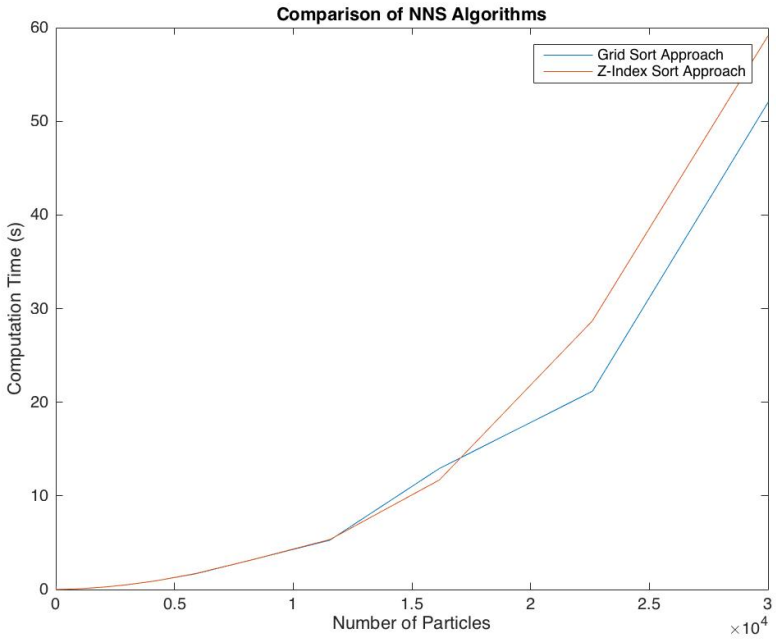

Figure 3.2: Comparison between implemented Neighbor Searching Algorithms

ing may be more preferable in the case of infinite domains [16]. The basic principle of these hashing methods is the mapping of any position $\vec{r} = (x, y)$ (in this case any cell center) to a hash table of size q as is shown in Eq. (3.2) [21]. In this equation, p_1 and p_2 are large prime numbers, and d corresponds to the cell size.

$$c = mod\Big[xor\{floor(\frac{x}{d}) * p_1, xor(floor(\frac{y}{d}) * p_2)\}, q\Big] \tag{3.2}$$

In order to properly implement this scheme avoiding hash collisions, where two different grid cells are assigned to the same hash cell, it is desirable to employ large hash table sizes [21]. According to [18] the performance of this method is optimized by using a cell size d equal to the smoothing length h and by employing a hash table size equal to the prime number closest to $2n$, n being the number of particles in the simulation. Once a variety of NNS algorithms have been presented and the choice between such algorithms has been discussed, it follows to describe the employed kernel function.

4 Interpolating Kernel Function

The choice of the interpolating kernel function to be implemented in any SPH simulation can profoundly affect the speed, accuracy and stability of the system. According to [17], such a function should approximate a Gaussian but with a finite support radius h. Nevertheless, there is no general consensus with regards to an optimal kernel function, as each function signifies a specific trade-off between accuracy and computational cost [16]. This specific implementation has employed a cubic spline kernel function, described by [17] as commonly used and based on Schoenberg M_n splines.

$$W_{ij}(\vec{r},h) = \frac{1}{\pi h^3}\begin{cases} 1-\frac{3}{2}\left(\frac{\|\vec{r}\|}{h}\right)^2+\frac{3}{4}\left(\frac{\|\vec{r}\|}{h}\right)^3 & 0\le\|\vec{r}\|\le h \\ \frac{1}{4}\left(2-\frac{\|\vec{r}\|}{h}\right)^3 & h\le\|\vec{r}\|\le 2h \\ 0 & \|\vec{r}\|>2h \end{cases} \tag{4.1}$$

$$\nabla W_{ij}(\vec{r},h) = \frac{9}{4\pi h^5}\begin{cases} \left(\frac{\|\vec{r}\|}{h}-\frac{4}{3}\right)\vec{r} & 0\le\|\vec{r}\|\le h \\ -\frac{1}{3}\left(2-\frac{\|\vec{r}\|}{h}\right)^2\frac{h}{\|\vec{r}\|}\vec{r} & h\le\|\vec{r}\|\le 2h \\ 0 & \|\vec{r}\|>2h \end{cases} \tag{4.2}$$

Note that the gradient of this kernel function is also employed throughout the implementation, as will be shown later. While as Eq. (4.1) and Eq. (4.2) present the expressions for the cubic spline kernel function and its gradient, Fig. 4.1 shows the evolution of both with $\frac{\|\vec{r}\|}{h}$.

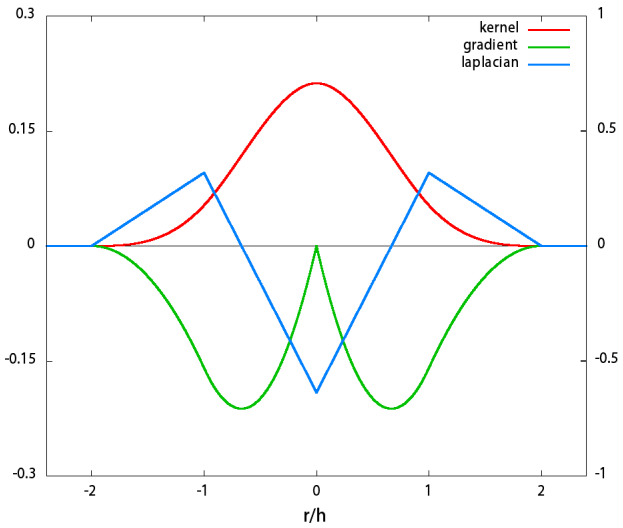

Figure 4.1: Cubic Spline Function and Gradient [3]

5 Advective Forces

After describing the utilized nearest neighbor search algorithms as well as the implemented interpolating function, it is necessary to present the used models for the non-pressure forces, these being the forces associated to viscosity, surface tension, and gravity.

5.1 Viscosity

The viscosity force model implemented in this study is the one suggested by [17]. This viscosity model is said to conserve linear and angular momentum as it depends on relative speeds of particle speeds [11]. Note that this is an artificial viscosity model introduced in order allow for shock phenomena, which although bearing no relation to real viscosities, maintains the continuity of viscous stress accurately and can be used to model real viscous phenomena [17].

$$\vec{F}_i^{\nu} = -\sum_j m_j \Pi_{ij} \nabla W_{ij}(\vec{r}_i - \vec{r}_j, h) \tag{5.1}$$

$$\Pi_{ij} = -\nu\Big(\frac{min(\vec{r_{ij}}\vec{v_{ij}}^T, 0)}{\|\vec{r_{ij}}\|^2 + \varepsilon h^2}\Big) \tag{5.2}$$

$$\nu = \frac{2\alpha h c_s}{\rho_i + \rho_j} \tag{5.3}$$

This artificial viscosity model is presented in Eqs. (5.1),(5.2) and (5.3), where α should remain within 0.08 and 0.5 and $\varepsilon = 0.01$, as this last constant is introduced to avoid singularities once $\|\vec{r_{ij}}\| = 0$ [8].

5.2 Surface Tension

Many surface tension models employed in SPH today are based on the usage of a color value c_i which is applied to all of the particles of a same phase, such that this color value is interpolated following Eq. (5.4) [20].

$$c(\vec{r}_i) = c_i = \sum_j \frac{m_j c_j}{\rho_j} W_{ij}(\vec{r}_i - \vec{r}_j, h) \tag{5.4}$$

With this color value, the surface tension is calculated taking into account the surface normal $\vec{n} = \nabla c$, as shown in Eq. (5.5). As the calculation of $\nabla^2 c$ is error-prone close to the free fluid surface where the particles have fewer neighbors and the calculation of the second derivative is vulnerable to particle disorder [8], this method was not implemented in the current study.

$$\vec{F}_i^s = -\kappa \nabla^2 c \frac{\vec{n}}{\|\vec{n}\|} \tag{5.5}$$

The current simulation is based on the surface tension model presented by [8], where a microscopic view of the fluid is assumed, in which surface tension arises due to attractive forces between molecules. These attractive forces are scaled by means of the kernel function. Moreover, as visible in Eq. (5.6), this model avoids the usage of kernel function gradients, in order to improve the stability and efficiency of the approach [8]. In the implemented model, $\kappa = 0.8$ and $\vec{r_{ij}} = \vec{r}_i - \vec{r}_j$.

$$\vec{F}_i^s = -\kappa \sum_j m_j W_{ij}(\vec{r}_i - \vec{r}_j, h)\vec{r_{ij}} \tag{5.6}$$

5.3 Gravity

The calculation of the gravity force is relatively simple as it assumes a constant gravitational acceleration.

$$\vec{F}_i^g = m_i \vec{g} \tag{5.7}$$

6 Pressure and Pressure Force

Once the density and advective forces of each particle have been obtained, the calculation of the particle pressures and pressure forces can be performed in many different ways, basically depending on which equations are employed and the method used to resolve such equations. The selection of a specific method depends not only on the performance and maximum time interval that such specific method allows for, but also on the extent to which incompressibility can be imposed on the fluid. The enforcement of compressibility can be seen as an essential cause for the verisimilitude of the simulation, this imposition gaining significance as the number of particles in the simulation increases [16]. However, the calculation of particle states with low compressibility can be seen as the most expensive step in any SPH simulation, since either the computation per time step is efficient at the cost of small time steps, or larger time steps occur in which the computation is expensive [16]. In general lines, four different categories can be outlined for SPH pressure solvers imposing incompressibility: non-iterative equation of state (EOS) solvers , EOS solvers with splitting, iterative EOS solvers, and solvers based on a Poisson pressure equation. This study focuses on two different methods: a non-iterative Tait EOS solver, and a pressure projection IISPH solver.

6.1 Non-Iterative Tait EOS Solver

In a non-iterative EOS solver, particle pressures are calculated by directly introducing the particle densities into an equation of state. This equation of state typically includes a set of stiffness terms such as the speed of sound c_s defining the compressibility of the fluid, and takes into account the deviation of the particle's density from the fluid's corresponding rest density ρ_0. In this implementation, Tait's equation of state is used, as it allows for low density oscillations when compared to other equations of state relevant to SPH [8]. This EOS is presented in Eq. (6.1).

$$p_i = \frac{\rho_0 c_s^2}{\gamma}\left(\left(\frac{\rho_i}{\rho_0}\right)^\gamma - 1\right) + p_0 \tag{6.1}$$

If $\gamma = 7$ and $\rho_0 = 1000$ as corresponds to water, one can see that the role of introducing fluid stiffness is performed by c_s. This specific implementation calculates the required speed of sound similarly to [8]. In order to do so, it is assumed that the relative variations in density are proportional to M^2, M being the fluid's Mach number. With such an assumption one can obtain the following relation.

$$\eta = \frac{\Delta\rho}{\rho_0} \sim \frac{\|\vec{v}_f\|^2}{c_s^2} \tag{6.2}$$

Here, η is the maximum allowable relative density oscillation, which is imposed, and $\vec{v}_f$ is a characteristic fluid velocity. If $\|\vec{v}_f\|$ is predicted based on the simulation, f.e. in the case of a falling fluid column of height H, $\|\vec{v}_f\| \sim \sqrt{2gH}$, one can thus determine an appropriate stiffness term for the equation.

Moreover, one can see that Eq. (6.1) includes a constant term which is independent from the particle's density. This is the background pressure, here introduced according to [19]. This background pressure is introduced to avoid generalized negative pressures, which lead to clumping of particles and nonphysical formation of voids. However, this background pressure should be kept at a minimum value to avoid high frequency errors in the solution [19]. Usually, this constant term has the form $p_0 = c_0^2 \rho_0$.

Although any implementation of a non-iterative EOS solver is less efficient than its iterative counterpart [16], [19] emphasizes that the larger time step allowed by ISPH schemes is counteracted by an increased CPU cost of the resolution of the Poisson pressure equation. Moreover, due to the ease with which explicit schemes such as an EOS solver can be parallelized, some cases may exist in which a non-iterative EOS solver may work more efficiently [19].

$$\vec{F}_i^p = -m_i \nabla p_i = -m_i \sum_j m_j \frac{p_j}{\rho_j} \nabla W_{ij}(\vec{r}_i - \vec{r}_j, h) \tag{6.3}$$

With regards to the calculation of the pressure force, it is important to note that the usage of an equation of the form displayed in Eq. (6.3) is not optimal, since this expression does not conserve linear nor angular momenta exactly [17]. This is due to the fact that the pressure force exerted by particle i on particle j is not equal and opposite to the force exerted by the latter on the former, as can be seen in Eq.(6.4). Note that $p_i \neq p_j$ and $\nabla_i W_{ij} = -\nabla_j W_{ij}$.

$$\frac{m_i m_j p_j}{\rho_i \rho_j} \nabla_i W_{ij} \neq -\frac{m_i m_j p_i}{\rho_i \rho_j} \nabla_j W_{ij} \tag{6.4}$$

In order to solve this issue, [17] proposed a symmetric expression making use of a Lagrangian. This expression has been used in the current implementation and is expressed in Eq. (6.5).

$$\vec{F}_i^p = -m_i \sum_j m_j \Big(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \Big) \nabla W_{ij} \tag{6.5}$$

6.2 Pressure Projection IISPH Solver

The pressure projection implicit incompressible SPH (IISPH) solver implemented in this study followed the description by [15]. This solver can be included in the category of solvers based on a Poisson pressure equation. The development of this specific solver structure appeared due to the fact that incompressible SPH solvers are considered impractical within the context of computer graphics, as the performance of such schemes does not scale properly with the simulation domain. Thus, the IISPH presented by [15] discretizes the Poisson pressure equation in a way which improves the convergence of the solver and the stability of the temporal integration.

In order to solve for the particle pressures, this method iteratively solves a linear system of equations in which the particle pressures are the unknowns. To develop such a system, the continuity equation $\frac{D\rho}{Dt} = -\rho\nabla\vec{v}$ is discretized utilizing the SPH expression for the velocity divergence. This discretization is described in Eq. (6.6), where $\vec{v_{ij}} = \vec{v_i} - \vec{v_j}$ [15].

$$\frac{\rho_i(t+\Delta t) - \rho_i(t)}{\Delta t} = \sum_j m_j \vec{v_{ij}}(t+\Delta t)\nabla W_{ij}(t) \tag{6.6}$$

The dependency of the particle velocities $\vec{v_{ij}}$ on the forces acting on the particles can be expressed imposing the conservation of linear momentum. However, the corresponding sum of forces is performed following the splitting concept, where intermediate velocities and positions are predicted using all advective (non-pressure) forces, as can be seen in Eq. (6.7) [16]. These intermediate predicted velocities are known as advective velocities $\vec{v_i^{adv}}$.

$$\vec{v_i}(t+\Delta t) = \vec{v_i}(t) + \Delta t\frac{\vec{F_i^p} + \vec{F_i^{adv}}}{m_i} = \vec{v_i^{adv}} + \Delta t\frac{\vec{F_i^p}}{m_i} \tag{6.7}$$

Furthermore, the intermediate densities can also be calculated by introducing the splitting concept employed in Eq. (6.7) in Eq. (6.6). This yields the following expression.

$$\rho_i^{adv} = \rho_i(t) + \Delta t\sum_j m_j \vec{v_{ij}^{adv}}\nabla W_{ij}(t) \tag{6.8}$$

As the ultimate goal of this development is to obtain a set of equations to solve the particle pressures, one can modify Eq. (6.6) by imposing that $\rho_i(t+\Delta t) = \rho_0$ (where ρ_0 is the fluid's rest density), and by introducing the dependency between the particle velocities and the forces acting on such particles [15]. This is equivalent to expressing the non-advective terms in (6.6) as a function of pressure forces and the advective terms as a function of ρ_i^{adv}.

$$\Delta t^2\sum_j m_j\Big(\frac{\vec{F_i^p}}{m_i} - \frac{\vec{F_j^p}}{m_j}\Big)\nabla W_{ij}(t) = \rho_0 - \rho_i^{adv} \tag{6.9}$$

In this way, one obtains Eq. (6.9), which expresses a linear system of equations due to the fact that the pressure forces acting on the particles depend linearly on the pressures associated to these particles. Note that as the particle advective densities have already been calculated once the point of the pressure calculation has been reached, the independent term of this system of equations will always be $\rho_0 - \rho_i^{adv}$. This last equation is considered as the Poisson pressure equation employed in the system, while as Eq. (6.8) corresponds to the prediction step of the projection scheme [15].

In order to iteratively solve the presented system of equations using a Relaxed Jacobi scheme, the pressure force acting on each particle is separated into the contributions caused by the particle's own pressure ($\vec{d_{ii}}$) and the contributions caused by the particle's neighbor's pressures ($\vec{d_{ij}}$). Thus, starting from the expression for the pressure force shown in [17] and described in Eq. (6.5), one obtains the following expression.

$$\Delta t^2 \frac{\vec{F}_i^p}{m_i} = -\Delta t^2 \sum_j m_j \Big(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \Big) \nabla W_{ij} = \Big(-\Delta t^2 \sum_j \frac{m_j}{\rho_i^2} \nabla W_{ij}(t) \Big) p_i + \sum_j -\Delta t^2 \frac{m_j}{\rho_j^2} \nabla W_{ij}(t) p_j$$
$$= \vec{d}_{ii} p_i + \sum_j \vec{d}_{ij} p_j \tag{6.10}$$

With this, the implemented equation to be solved following a Relaxed Jacobi scheme is described in Eq. (6.11), where the subindex k corresponds to the neighbors of each of the neighbors j of particle i (such that $k \neq i$), l describes the iteration index, and ω represents the relaxation factor. Moreover, a_{ii} is the diagonal value of the coefficient matrix, expressed in Eq. (6.12).

$$p_i^{l+1} = (1-\omega) p_i^l + \frac{\omega}{a_{ii}} \Big(\rho_0 - \rho_i^{adv} - \sum_j m_j \Big(\sum_j \vec{d}_{ij} p_j^l - \vec{d}_{jj} p_j^l - \sum_{k \neq i} \vec{d}_{jk} p_k^l \Big) \nabla W_{ij}(t) \Big) \tag{6.11}$$

$$a_{ii} = \sum_j m_j (\vec{d}_{ii} - \vec{d}_{ji}) \nabla W_{ij}(t) \tag{6.12}$$

Then, a simulation step with an IISPH solver goes as follows: densities are calculated with Eq. (2.3), advective forces are calculated by means of the equations in Section 5, and then advective velocities and densities are obtained with Eq. (6.7) and Eq. (6.8), respectively. Later, the components $\vec{d}_{ii}$, $\vec{d}_{ij}$ and a_{ii} are calculated and introduced into Eq. (6.11), which is solved iteratively following a Relaxed Jacobi scheme. Once all particle pressures have been computed, the particle pressure forces are computed using Eq. (6.10), which is equivalent to employing Eq. (6.5), and thus the overall particle velocity is computed using Eq. (6.7).

With regards to the resolution of the linear equations using a relaxed Jacobi scheme, it follows to describe the choice for the convergence criterion of such iterative resolution. The convergence criterion of the resolution is based on the calculation of the average relative density deviation from the fluid rest density. Simply put, the iterative resolution is to stop once $\frac{\rho^l_{average} - \rho_0}{\rho_0} < \eta$, where η is fixed by the user [15]. This convergence criterion is more representative of the overall fluid volume than when observing the maximum relative density deviation, as this maximum value is expected to vary much more throughout the simulation, thus causing notable variations in the fluid volume if chosen for the convergence criterion [15].

When compared to other iterative SPH solvers such as the iterative EOS solver with splitting predictive-corrective incompressible SPH (PCISPH) or the pressure projection-based incompressible SPH, this specific solver requires much less iterations per simulation step in order to calculate particle pressures, and is capable of imposing much smaller compression ratios in the fluid than these other incompressible SPH solvers [15]. Nevertheless, the implementation of an IISPH solver in this study was not successful, and although much was learned about the specific method, the simulated cases employed only the EOS solver presented in Section 6.1.

7 Integration

Once all of the relevant forces exerted on each particle have been calculated, one has to integrate the equation representing the conservation of linear momentum (Eq. (2.4)) in order to obtain the velocity and position of each particle. While as employing a simple explicit Euler scheme would imply a relatively unstable resolution, bounding the time step to low values, a more stable alternative integration method is commonly used in SPH: the Leap-Frog Method. This implicit Euler method obtains its name from the fact that the velocities leap over the positions, and vice versa [18].

$$\frac{dx}{dt} = v \tag{7.1}$$

$$\frac{dv}{dt} = F(x) \tag{7.2}$$

In (7.1), one can see a generic expression of the typical set of equations to integrate in SPH. If one is to do so employing a Leap-Frog scheme with a time step h, one would follow the technique shown in (7.3). This method is second order accurate with respect to h, an advantage compared with first order schemes such as the typical explicit Euler integration [7]. In Fig. 7.1, one can see a representation of the Leap-Frog scheme, where the horizontal line represents time, and the different jumps in velocity and position are displayed.

$$x_{n+1} = x_n + h v_{n+1/2} \tag{7.3}$$

$$v_{n+3/2} = v_{n+1/2} + hF(x_{n+1}) \tag{7.4}$$

Once the particle velocities have been integrated, an XSPH correction is performed. XSPH prevents particle interpenetration and thus avoids the formation of particle clusters [7]. Basically, this transformation brings the particle velocity closer to the average velocity of its neighborhood, thus reducing particle disorder and allowing more stable results [17]. Although this

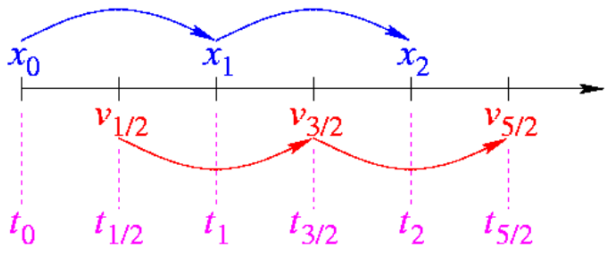

Figure 7.1: Visual representation of the Leap-Frog mechanism [1]

correction conserves angular and linear momenta, energy is not completely conserved [17]. This correction is shown in Eq. (7.5), where ε is the correction factor, ranging from 0 to 1.

$$\vec{v}_i \leftarrow \vec{v}_i + \varepsilon \sum_j \frac{2m_j(\vec{v}_j - \vec{v}_i)}{\rho_i + \rho_j} W_{ij} \tag{7.5}$$

Another implemented measure relevant to the integration is the use of the Courant-Friedrich-Levy (CFL) condition, which in SPH states that a particle must not move more than a certain part of its influence radius h during one time step [13]. This is expressed in Eq. (7.6), where the specification of λ modifies what is the maximum part of h that a particle is allowed to travel [16]. In order to better take into account rigid body behavior, rigid body particle velocities are also taken into account when imposing this condition [6]. In most SPH implementations, $\lambda = 0.4$ [13].

$$\Delta t \leq \lambda \frac{h}{\|\vec{v}_{max}\|} \tag{7.6}$$

8 Boundary Handling

In order to implement an SPH simulation on any sort of finite domain, it is essential to ensure that the number of particles in the domain is controlled. To do so, one can employ fluid sources and sinks, as well as a proper set of spatial boundaries which allow the domain to remain finite. Since this specific implementation lacks any sort of fluid particle generation or elimination, the control on the number of particles within the spatial domain under study is imposed by a set of spatial boundaries at the ends of the domain. These boundaries are imposed in two ways: one involving a set of boundary particles impeding fluid penetration, and one involving a simple reflective wall which is activated once a certain geometric condition is fulfilled.

Boundary handling in SPH is very commonly performed using distance-based penalty methods, which commonly require large penalty forces restricting the time step [17]. These large penalty forces can be avoided by decreasing their stiffness. However, this may lead to boundary penetration [14]. Moreover, these schemes often cause particles to stick near the boundaries due to the absence of neighboring particles at the boundaries [6].

Another common approach is to introduce frozen and ghost particles. Frozen particles are fluid particles which are constrained to fixed positions, while as ghost particles are particles generated dynamically across the boundary. According to [19], ghost particles are used to model the solid domain as a set of fictitious solid particles, to which all fluid fields such as velocity and pressure are extended in order to correctly enforce the desired boundary conditions. More specifically, each ghost particle is impregnated with the same viscosity, pressure, mass and density of its fluid counterpart, whereas the component of the fluid particles' velocity normal to the boundary is inverted [11]. This procedure is described in Fig. 8.1. In order to replicate the relevant fluid fields, ghost particles are associated to specific interpolation points within the fluid [19]. These approaches allow the relevant field variables to be approximated adequately, allowing for continuous pressure gradients, as the particles are sampled all across the boundary [6]. In order to ensure non-penetration, several layers of ghost and/or frozen particles should be defined. Nevertheless, as ghost particles are generated on the fly, they are hard to generate for complex boundaries [14]. This implementation avoids the usage of ghost and frozen particles by employing a simpler scheme which works adequately.

The imposition of spatial boundaries by means of a set of boundary particles in this study is performed based on [14] and [6]. In this technique, the rigid-fluid coupling employs a set of boundary particles b which contribute to the density of neighboring fluid particles. This allows for more continuous density gradients near the boundaries [6]. These boundary particles are sampled along the boundary surface. Then, one can say that each boundary particle b represents a volume V_b at the boundary surface, which follows Eq. (8.1).

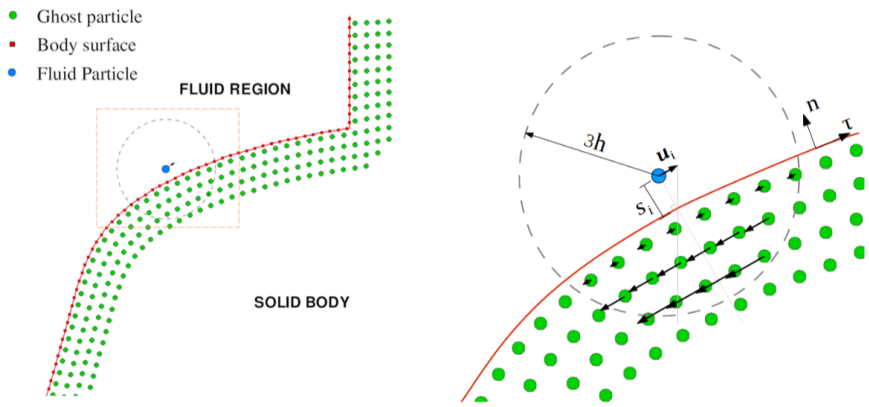

Figure 8.1: Boundary Handling using Ghost Particles. Left: discretization of boundary surface with ghost particles. Right: local mirroring of the velocity field for a generic fluid particle [19]

$$V_b = \frac{m_b}{\rho_b} = \frac{m_b}{\sum_k m_k W_{bk}} = \frac{m_b}{m_b \sum_k W_{bk}} = \frac{1}{\delta_b} \tag{8.1}$$

Note that k denotes the boundary particle neighbors of particle b, and that it is assumed that all boundary particles have the same mass. Bearing this equation in mind, one can see that boundary particle volumes will decrease for densely sampled areas, and increase for scarcely sampled areas [6]. If one were now to employ Eq. (8.2) in order compute the density of a fluid particle, one would obtain relatively large contributions from overly sampled areas of the boundary, as homogeneous sampling of boundary particles is not ensured. Thus, it is necessary to include the boundary particle volume in the contribution to the fluid particle density. This contribution is described in Eq. (8.3) [6].

$$\rho_i = \sum_j m_j W_{ij} + \sum_b m_b W_{ib} \tag{8.2}$$

$$\rho_i = \sum_j m_j W_{ij} + \sum_b \rho_0 V_b W_{ib} = \sum_j m_j W_{ij} + \sum_b \Phi_b(\rho_0) W_{ib} \tag{8.3}$$

With this last formulation, the contribution of boundary particles to fluid particle densities is independent on the sampling homogeneity. Basically, in a homogeneously sampled case, $\Phi_b(\rho_0)$ increases the boundary particle contributions by the amount of the ratio of boundary particle volume to fluid particle volume [6]. Moreover, due to the shape of generally employed kernel functions, the contribution of a second layer of boundary particles will be much smaller than that of the first layer, which allows the usage of thin-shell boundaries with this specific method.

Furthermore, this method used an additional tool for avoiding boundary penetration and clustering of particles near the boundary: the introduction of a pressure force exerted by the boundary on the fluid. The expression for this force is based on the fact that in practice, the pressure force from a fluid to a rigid body does not influence the neighboring fluid parts [6]. Thus, the pressure force exerted by the boundary on the fluid will only depend on the pressure of the fluid part, as boundary particles do not have an individual pressure. Moreover, as

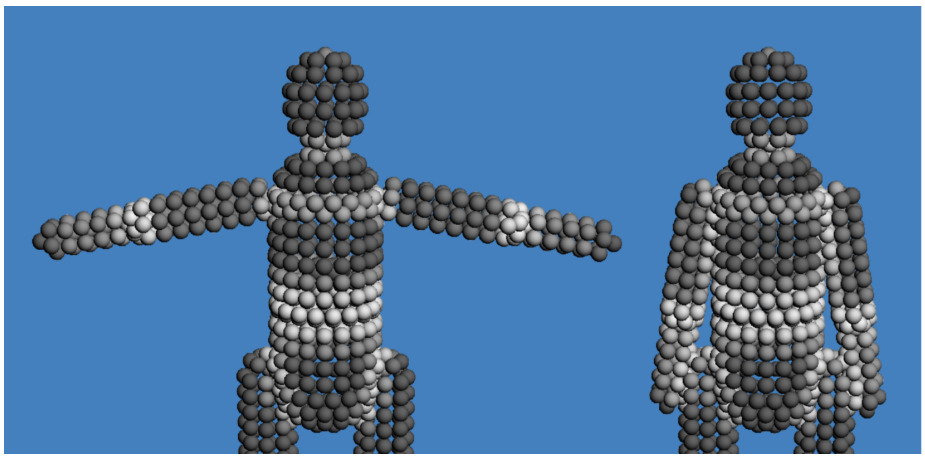

Figure 8.2: Color-coded depiction of boundary particle contributions to fluid particle density. Black and white particles represent large and small contributions, respectively. [6]

the fluid particle pressure will increase near the boundary, so will the magnitude of this force, avoiding sticking artifacts without the need for additional penalty forces or position corrections [15]. This pressure force is expressed in Eq. (8.4). The pressure forces exerted by boundary particles on fluid particles are taken into account when computing the sum of forces acting on each particle in order to calculate particle velocities. This method of boundary handling is also advantageous as far as it allows for the imposition of boundary movement, as for example in the introduction of a wave generator, and it also allows for a smooth introduction of two-way rigid-fluid coupling in the implementation. Moreover, the usage of particle sampling of rigid surfaces allows for different boundary shapes.

$$\vec{F}^{p}_{i \leftarrow b} = -m_i \Phi_b(\rho_{0i}) \frac{p_i}{\rho_i^2} \nabla W_{ib} \tag{8.4}$$

This implementation also achieves the imposition of spatial boundaries of the domain by establishing reflective boundaries at the edges of the region under study. Basically, once the velocities and positions of fluid particles have been calculated at the end of the time step, the obtained particle positions are analyzed in order to find particles outside of the desired domain. Bearing in mind these particles, a classification is done in order to determine through which boundary did each particle leave the domain. Then, after calculating the time required for each particle to exit the domain, a reflection of each particle's velocity is performed, and their new position is updated according to how much time was left within the time step at the point in which the particle left the domain. In this way, for the imposition of a reflective boundary at $y = y_{max}$, the following is employed.

$$v_x = \varepsilon v_x^0; \tag{8.5}$$

$$v_y = -\varepsilon v_y^0; \tag{8.6}$$

$$r_x = r_x^0 + v_x^0\Big(\frac{y_{max} - r_y^0}{v_y^0}\Big)(1-\varepsilon) + \varepsilon v_x^0 \Delta t; \tag{8.7}$$

$$r_y = y_{max} - \varepsilon v_y^0\Big(\Delta t - \frac{y_{max} - r_y^0}{v_y^0}\Big); \tag{8.8}$$

In (8.5), v_x^0, v_y^0, r_x^0 and r_y^0 all correspond to the velocities and positions of the particles at the beginning of the current time step, ε represents the damping degree of the wall boundary, and Δt is the time jump of the iteration. This scheme is implemented analogously in the three remaining boundaries of the 2D domain. Nevertheless, in cases in which a particle is very close to a corner, or has a high velocity, the newly obtained corrected position may still be outside the domain, as the particle bounces on a boundary only to exit through another. Thus, the imposition of such a boundary is performed iteratively, correcting the position of particles that require it until there is not enough time remaining within Δt for any particle to leave the domain again.

9 Introduction of a Rigid Body

Based on how the rigid domain boundaries were implemented in the simulation, the transition to the introduction a rigid body is relatively simple. In this way, as described by [6], the rigid body is introduced as a closed boundary surface for which the boundary particle movement is based on rigid body mechanics. In the previous method described in Section 8, boundary particles (in this case describing the rigid body) contribute to the fluid particles' densities and exert a pressure force on the neighboring fluid particles. Now, this method is further developed by introducing a friction force exerted by boundary particles on neighboring fluid particles, and by taking into account the forces exerted on each boundary particle, based on the fact that these are equal and opposite to the forces exerted by such boundary particle on neighboring fluid particles [6].

With regards to this newly introduced friction force between interacting fluid and boundary particles, one must note that it is generated by utilizing the model of artificial laminar viscosity presented in [17] and described in Section 5.1, which is easily employed once the boundary particle velocities are known. Based on the form expressed in Eq. (5.1), the following equation is utilized.

$$\vec{F^{fr}_{i\leftarrow b}} = -m_i \Phi_b(\rho_{0i}) \Pi_{ib} \nabla W_{ib} \tag{9.1}$$

In (9.1), the expression utilized for Π_{ib} is the same as Eq. (5.2). However, the expression for ν within this expression varies, since boundary particles do not have an assigned density. This new form for the viscous factor ν is described in (9.2), where once again h is the kernel function interpolating radius, c_s is the fluid speed of sound, and the new term σ describes the fiction coefficient between fluid and rigid body [6].

$$\nu = \frac{\sigma h c_s}{2\rho_i} \tag{9.2}$$

In order to ensure the conservation of linear and angular momenta while introducing these forces between boundary and fluid particles, it is necessary to ensure the symmetry of these interactions. Thus, once the forces exerted by boundary particles on fluid particles are known, the forces affecting boundary particles can directly be obtained by applying the following.

$$\vec{F}^{p}_{i\leftarrow b} = -\vec{F}^{p}_{b\leftarrow i} \tag{9.3}$$

$$\vec{F}^{fr}_{i\leftarrow b} = -\vec{F}^{fr}_{b\leftarrow i} \tag{9.4}$$

An important advantage that this specific structure presents is that the variety of body shapes which can be implemented is only limited by how difficult such a shape is to generate and to sample [6]. Moreover, the symmetric nature of these boundary forces enables a

straight-forward collection of forces acting on each boundary particle and thus on the rigid body as a whole.

Once the forces exerted on each boundary particle are stored, they are utilized after the calculation of fluid particle positions in order to compute the rigid body dynamics. As the dynamic state of the rigid body in this two-dimensional implementation is described by its position, velocity, orientation angle, and angular velocity, each of these variables are to be computed in the update of the rigid body motion. In Eq. (9.5), one can see how the translational dynamics of the rigid body are updated, where $\vec{v_G}$ and $\vec{r_G}$ correspond to the velocity and position of the body's center of gravity, and the force sum is performed for each boundary particle b and for each of its fluid particle neighbors j. Moreover, Eq. (9.7) displays how the rotational dynamics of the body are updated, taking into account the sum of the torques of each boundary force with respect to the center of gravity of the body. In this two-dimensional implementation, both $\vec{\omega}$ and $\vec{\theta}$ can be described by scalar values.

$$\vec{v_G}(t+\Delta t) = \vec{v_G}(t) + \frac{\Delta t}{m}\Big(m\vec{g} + \sum_b \sum_j (\vec{F}^{fr}_{b\leftarrow j} + \vec{F}^{p}_{b\leftarrow j})\Big) \tag{9.5}$$

$$\vec{r_G}(t+\Delta t) = \vec{r_G}(t) + \Delta t\, \vec{v_G}(t+\Delta t) \tag{9.6}$$

$$\vec{\omega_G}(t+\Delta t) = \vec{\omega_G}(t) + \frac{\Delta t}{I_G}\Big(\sum_b \sum_j (\vec{r_b} - \vec{r_G}) \times (\vec{F}^{fr}_{b\leftarrow j} + \vec{F}^{p}_{b\leftarrow j})\Big) \tag{9.7}$$

$$\vec{\theta_G}(t+\Delta t) = \vec{\theta_G}(t) + \Delta t \vec{\omega_G}(t+\Delta t) \tag{9.8}$$

Lastly, the velocities and positions of all rigid body particles are updated as shown in Eq. (9.9), where $R(\theta_G)$ is a rotation matrix used to describe rotation about the Z-axis in this two-dimensional implementation.

$$\vec{v_b} = \vec{v_G} + (\vec{r_b} - \vec{r_G}) \times \vec{\omega_G} \tag{9.9}$$

$$\vec{r_b} = \vec{r_G} + R(\theta_G)(\vec{r_b} - \vec{r_G}) \tag{9.10}$$

10 Simulated Cases

10.1 Structure of the Implementation

The current study takes into account the presented alternatives for each of the components in the simulation in order to implement a SPH solver which displays physically sensible fluid behavior. Then, the main structure of the implementation can be outlined as follows. First, the system is initialized by establishing important characteristics of the simulation such as the particle number, the initial fluid and rigid body conditions, and the boundary location and shape. Once this is performed, the iterative part of the simulation begins, and the designed SPH solver is employed. This SPH solver begins by performing the particle neighbor search, and by calculating relative positions and velocities between neighboring particles. Bearing this in mind, it is possible to calculate the relevant values of the required kernel functions and kernel function gradients, and with thus also possible to obtain the density contribution of boundary particles $\Phi_b(\rho_{0i})$. Then, the SPH solver continues on to calculate the fluid particle densities and with this to determine the advective forces acting on each particle. Note that during the calculation of these advective forces, any forces exerted on rigid body boundary particles are also stored. As the implemented solver is based on the non-iterative EOS SPH solver presented in Section 6.1, the solver can directly employ Tait's equation of state to calculate the fluid particle pressures. After that, pressure forces are calculated not only for fluid particles but for rigid particles as well, and the velocities and positions of fluid particles are obtained following the previously presented Leap-Frog scheme. These fluid velocities are XSPH corrected and thus the SPH solver execution ends by updating the rigid body dynamics. Once the SPH solver routine is over, the last step in the iteration is to impose the reflective domain boundaries and calculate the new time step based on the CFL condition. This iterative procedure is performed until the end of the simulation time. Then, the desired variables and their evolution can be visualized and saved.

In this implementation the interaction between different rigid body geometries and fluid configurations is simulated. In order to do so, two initialization processes are required: one to initialize the fluid free surface and another to allow the floating body to reach somewhat steady conditions. Then, if three rigid body geometries are simulated, one requires one case to initialize the fluid surface and three cases to introduce each floating body into the fluid. After the initialization of both the fluid surface and the rigid body is performed, one can study the floating dynamics of each rigid body by generating a perturbation in the free fluid surface and analyzing its effect on the rigid body's motion.

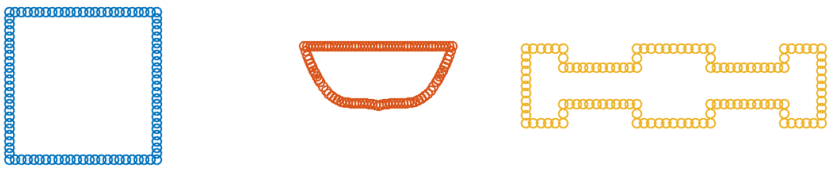

Figure 10.1: Particle sampling of the three simulated geometries. Left: Block, Middle: Generic monohull, Right: Generic multihull

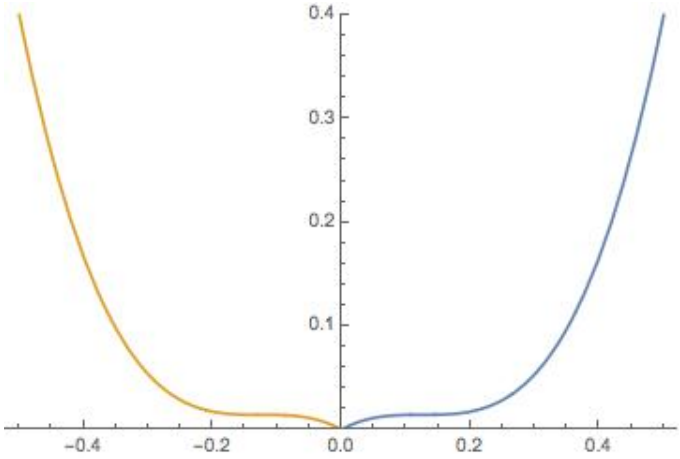

Figure 10.2: Plot representing the geometric duplication of (10.1)

10.2 Definition of Rigid Body Geometries and Characteristics

The rigid body two-dimensional geometries employed represent a cubic prism and two depictions of transverse cuts of generic boat hulls. The first shape choice is typical in many floating body explorations, whereas the latter two were chosen in order to compare the behavior of a monohull boat with that of a multihull when hit by side waves. In Fig. 10.1, one can visualize the particle sampling representing these geometries, where the same characteristic length l was applied to all three of them.

While as the block and the generic multihull geometries are relatively easy to generate due to the fact that these geometries are made up of rectangles, the generic monohull geometry was developed using a pair of cubic functions, to try to simulate the typical curvature that a monohull cross-section has. In this way, the generic monohull geometry employed a symmetric duplication of Eq. (10.1), where b corresponds to the hull beam and d to the hull draft. In order to maintain a steady relation between these two parameters, $b = l$ and $d = 0.4l$, where l is the specified characteristic length defining the hull. A closer look into the functions defining this geometry is shown in Fig. 10.2.

$$y = \frac{128d}{7b^3}\left(x - \frac{b}{8}\right)^3 + \frac{d}{28}; \tag{10.1}$$

Table 10.1: Table describing the specific geometric properties of the simulated rigid bodies

Geometric Properties of Simulated Rigid Bodies ($l = 1.0253m$)		
Geometry Type	Area (m^2)	Moment of Inertia ($kg * m^2$)
Block	0.2628	2.7628
Monohull	0.0826	0.0870
Multihull	0.1971	0.1613

Furthermore, once the relevant geometries have been defined, it is necessary to determine the characteristic properties of the rigid body. With regards to the center of gravity of each shape, while as for the block and the multihull the geometric centroid was chosen, in the case of the monohull a series of complex equations were avoided by establishing that the center of gravity is in the symmetry line separated $d/3$ from the lowest point of the hull. Moreover, to calculate the rigid body masses, it was assumed that these were proportional to the area contained within the cross-section, in order to impose a certain density ratio with respect to the fluid. In the simulated cases, this density ratio was set to 0.24. Finally, in order to calculate the moment of inertia with respect to the center of gravity I_G, common formulas were used for the block and multihull geometries, while as the monohull required double integration of its defining functions. The results of these calculations are expressed in Table 10.1.

10.3 Initialization of the Free Fluid Surface

The initialization of the free fluid surface was performed in a way which allowed for a simple definition of the initial particle configuration, but as a consequence required a long simulation time for the fluid to reach the state of a stable surface. In this case, the simulation time is 3 seconds. This initialization process was developed starting with a lattice of fluid particles stacked in the middle of the domain, in which the initial spacing between particles was specified. Then, the simulated case described the fall of the fluid mass due to gravity and its impact on the imposed boundary edges. During all of the simulated cases, the employed fluid was water, with $\rho_0 = 1000kg/m^3$. With regards to the employed spatial domain, it is worth noting that a square domain was utilized at all times, as it allowed for a simpler implementation of the Nearest Neighbor Searching algorithm. This square domain was limited by a contour of tightly packed boundary particles, as described in Section 8. This specific simulation did not involve any two-way fluid-rigid coupling, as the only rigid surfaces present in the domain are those of the domain limits. Thus, no rigid body dynamics were updated nor were any fluid-to-boundary forces stored.

This simulated case allowed the first contact with the developed solver, as it shed light on the performance of the implementation, and on what configuration could be considered as optimum. It was decided to employ a time step of $\Delta t = 0.001s$ and to limit the number of fluid particles to $n = 2000$, as this configuration ensured an apt combination between accuracy and stability. In order to allow for a constant fluid depth independent of the introduced number of particles, the domain size was implemented as dependant on the number of particles. Moreover, in order to ensure that the initial density of the stacked fluid was close to the rest density of the fluid, the particle mass was also defined as depending on the number of

Table 10.2: Relevant simulation parameters for the initialization of the free fluid surface

Number of Particles	2000
Domain Size (m)	4.1012
Particle Mass (kg)	1.0890
Particle Radius (m)	0.01
Support Radius h (m)	0.08
Initial Lattice Spacing (m)	0.05

fluid particles in the simulation. These relations were developed based on several trials and errors performed with different simulated cases. For the simulated case, the relevant system characteristics are described in Table 10.2.

With regards to the initial state of the fluid fields, it is important to note that while as the velocity of all fluid particles was initially null in order to allow for gravitational velocities, the pressure of all particles was set to $p = 101325Pa$. Also, it is worth mentioning that the employed value for the initial particle spacing was adjusted in order to impose an initial density close to the fluid's rest density, thus avoiding exaggerated compressions or expansions. A set of relevant snapshots of the simulation are presented in Fig. 10.3.

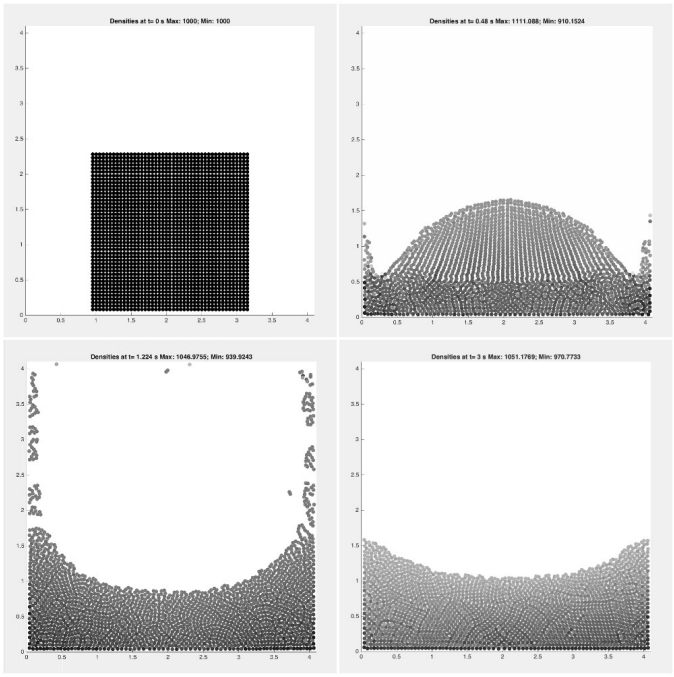

Figure 10.3: Snapshots of the density contours of the initialization of the free surface. Left up: $t = 0.000s$, Right up: $t = 0.480s$, Left down: $t = 1.224s$, Right down: $t = 3.000s$

Once the simulation is performed, one can observe that the resulting set of density contours shows a slightly irregular or j̈umpyëvolution. This may be due to the fact that the contour color scale is referenced to the maximum and minimum density values, such that as these two values oscillate, so does the color of the whole contour. However, it is also expected for weakly

compressible schemes such as the implemented EOS solver to present some of these small artifacts during the simulation [16]. With regards to the density oscillations, Fig. 10.4 shows the temporal evolution of all particle densities for the case in which $n = 2000$, as well as a curve displaying the mean fluid density. Taking into account that at no point of the simulation does the mean density exit the range $[950, 1050]\,kg/m^3$, one could say that the maximum relative deviation of the mean particle density is less than 5%. Finally, it is worth mentioning that the greater the number of particles in this specific initialization, the more continuous did the fluid surface appear during the simulation.

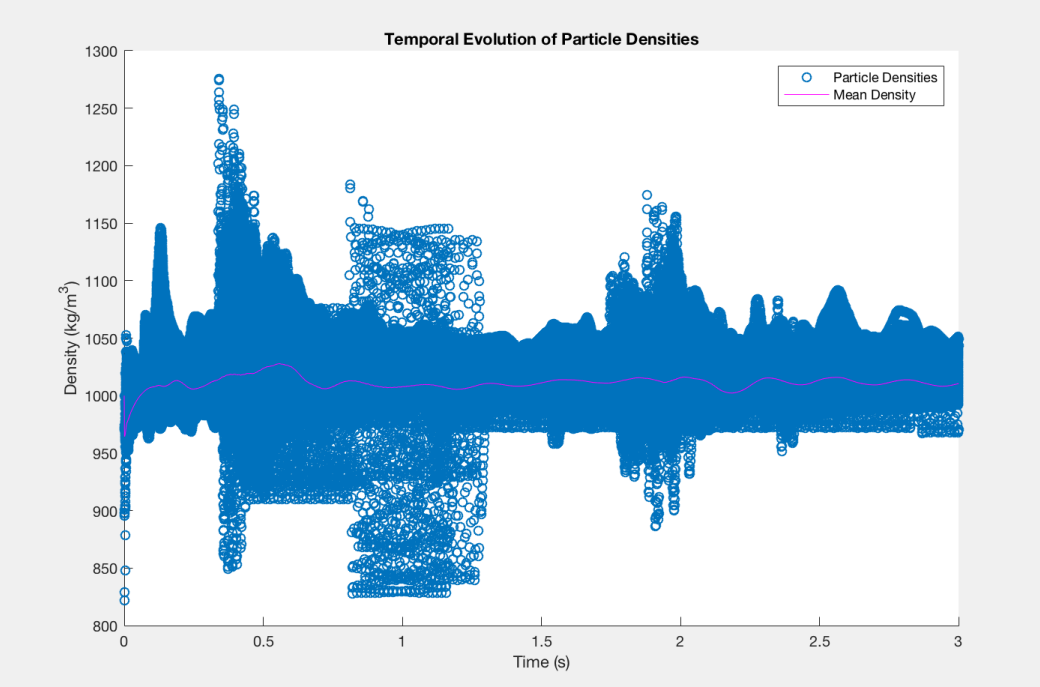

Figure 10.4: Temporal evolution of particle densities during the initialization of the free fluid surface

10.4 Rigid Body Initialization

Similarly to the initialization of the fluid free surface, the initialization regarding the introduction of the floating rigid body required an easy initial set-up, due to the fact that the rigid body is not initially submerged, albeit an extensive simulation time of 2 seconds. In this initialization, each of the simulated geometries fell with an initial vertical velocity into the fluid, thus causing a set of perturbations in the fluid surface. This simulation was carried out until the rigid body was considered to be at somewhat steady conditions. The properties relevant to the rigid body dynamics defined in Table 10.1 were employed, with an initial rigid body velocity of $\vec{v_G}^0 = [0, -10]\,m/s$. Moreover, as this simulation is based on the recently presented initialization of the fluid, the relevant parameters exposed in 10.2 also hold. Some snapshots of the rigid-fluid interaction are presented in Fig. 10.5.

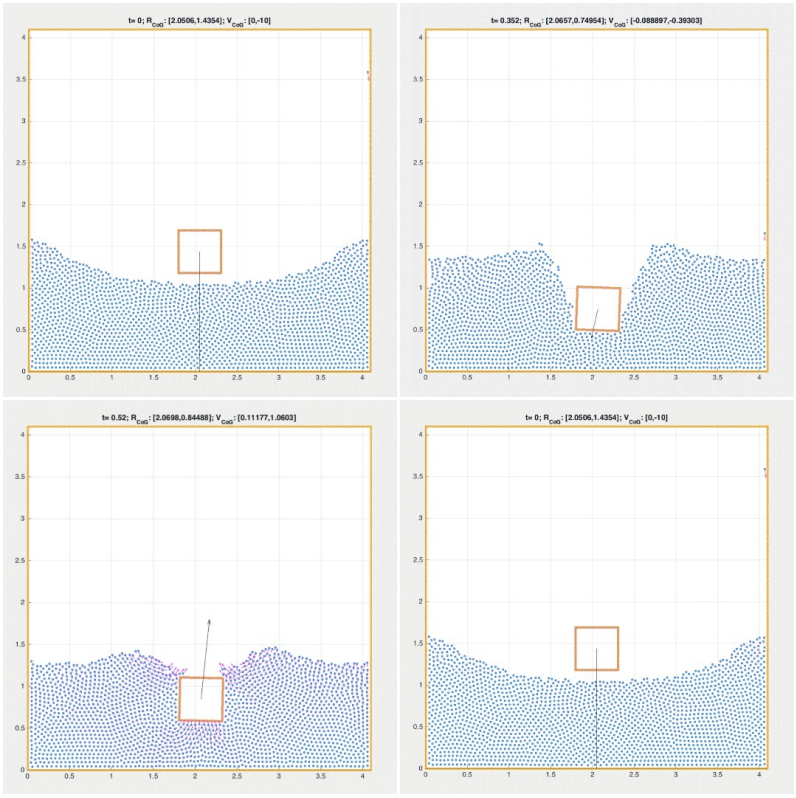

Figure 10.5: Snapshots of the initialization regarding the block geometry. Left up: $t = 0.000s$, Right up: $t = 0.352s$, Left down: $t = 0.520s$, Right down: $t = 1.996s$

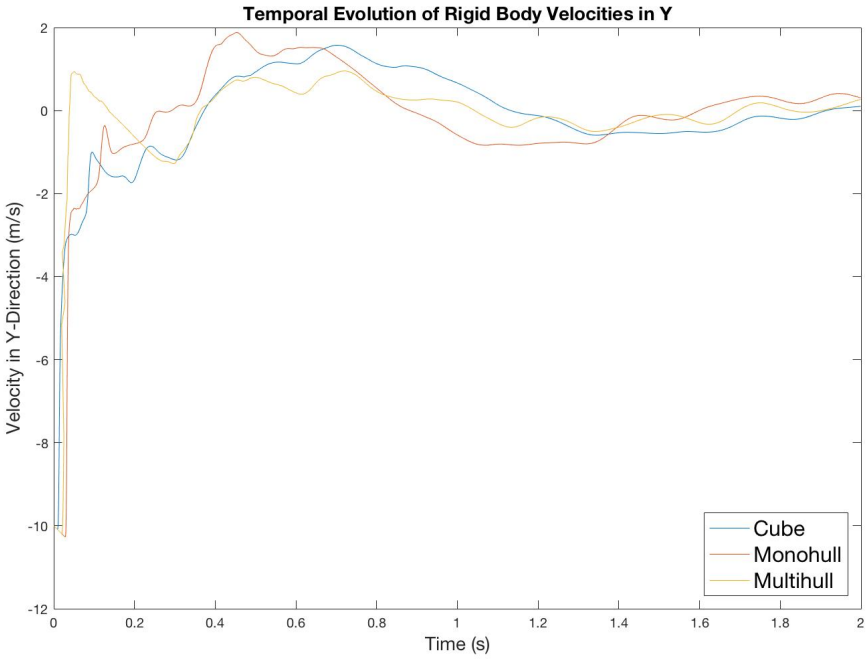

Figure 10.6: Temporal evolution of rigid body velocities during the body initialization

When observing the temporal evolution of the vertical velocities of each rigid body, one can see that once the body impacts the fluid, there is a sudden increase in vertical velocity, making the rigid body adopt even positive vertical velocities. As is expected, this upwards motion is later counteracted and the rigid body vertical velocities enter into a sort of damped oscillatory state. With regards to the initial jump in vertical velocity, one can see in Fig. 10.6 that the increase in velocity is the most visible in the case of the multihull geometry. This may be due to the fact that with the same characteristic length l, this geometry is defined with a much larger beam (see Fig. 10.1), causing the solid body to accelerate upwards much more than the other two geometries. However, this multihull geometry will decrease its vertical velocity and enter this sort of damped oscillatory state earlier than the other two geometries, whereas the velocities for both the cube and the monohull continue increasing but not as steeply, to enter the oscillatory state later. Due to this more drastic variation in velocities, one could say that the multihull geometry would reach its static position earlier, as is visible in Fig. 10.6.

This last hypothesis is proven to be right when observing the evolution of the rigid body positions in Y, as displayed in Fig. 10.8. Here one can see that that the amplitude of the oscillations in elevation corresponding to the multihull geometry is much smaller, which may lead to an earlier achievement of hydrostatic equilibrium. Another interesting feature regarding the elevation of the multihull is its sudden increase in elevation once the surface is reached, a jump in position which is not visible in the two other geometries.

A last remark regarding this simulated case should be made with regards to the evolution of rigid body orientations. One can see in this case that the monohull, having the smallest

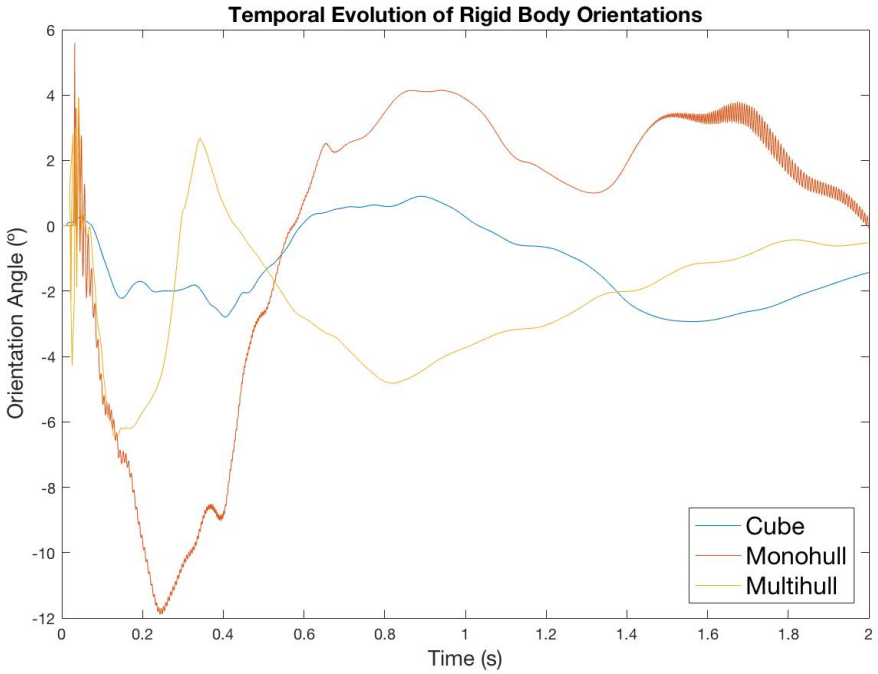

Figure 10.7: Temporal evolution of rigid body orientation angles during the body initialization

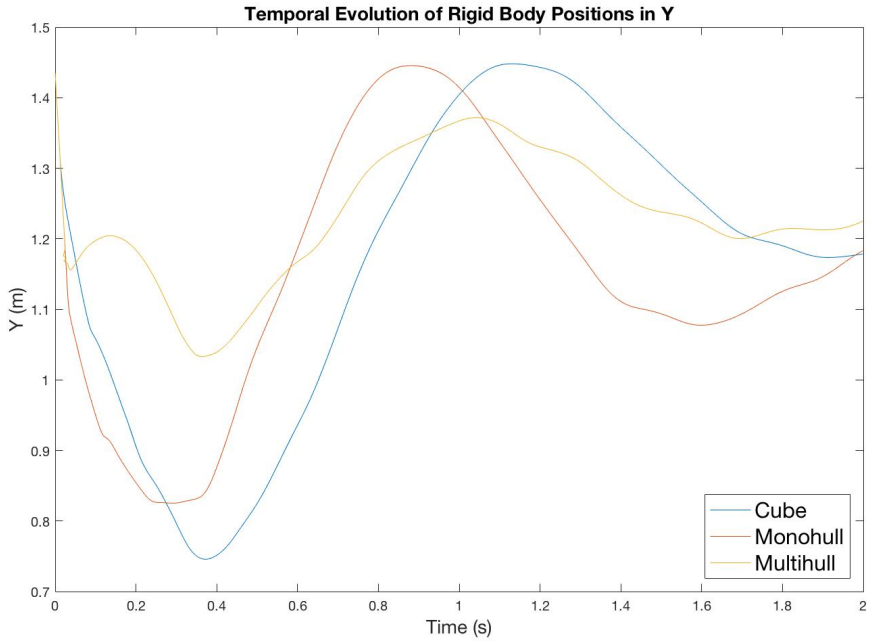

Figure 10.8: Temporal evolution of rigid body elevations during the body initialization

moment of inertia of the three geometries, experiences the largest variations in orientation of the three geometries. However, as visible in Fig. 10.7, these oscillations are not too significant,

as can be expected for a rigid body falling perpendicular to a free surface of water. It is considered that the horizontal dynamics of the presented rigid bodies are not significant for this specific case.

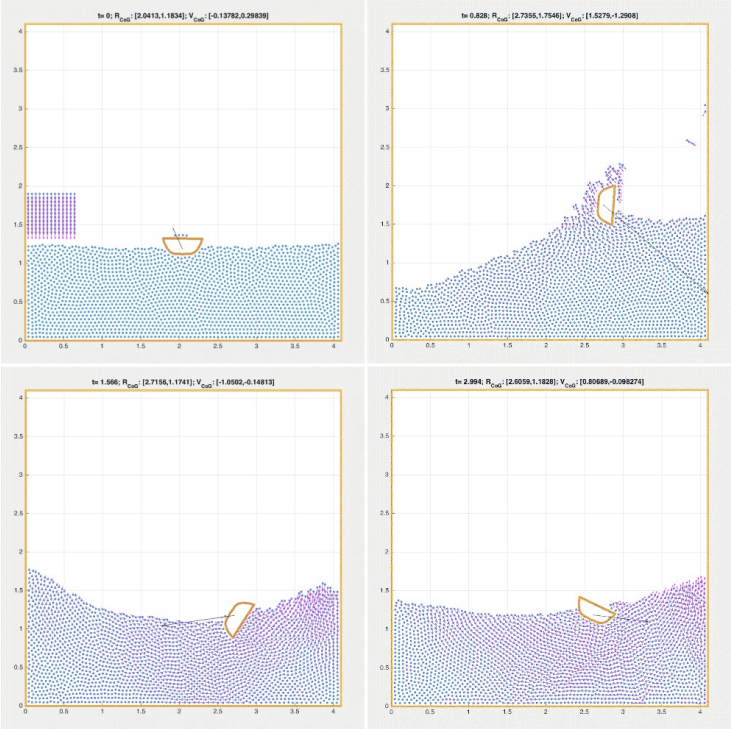

Figure 10.9: Snapshots of the incidence of a fluid surface perturbation on a generic monohull geometry. Left up: $t = 0.000s$, Right up: $t = 0.828$, Left down: $t = 1.566s$, Right down: $t = 2.994s$

10.5 Interaction between Floating Body and Fluid Surface with Perturbed Fluid Surface

Once the free surface has been initialized and the rigid body is properly introduced within the free surface, this implementation continues on to simulate the effect of a perturbation moving along the free surface inciding on the rigid body. In this case, the surface perturbation was introduced by adding an additional mass of fluid at one specific end of the domain, and allowing it to fall with an initial vertical velocity. Different perturbation magnitudes were generated depending on the number of particles introduced in this additional fluid part. Here, two different perturbation magnitudes were simulated: a moderate one with 150 additional particles, and an extreme one with 300. Each of these were simulated with the three geometries presented, based on the performed initializations. A set of snapshots of the simulated case can be seen in Fig. 10.9, where the monohull rigid finds itself colliding with a perturbation formed by 150 added particles.

When analyzing the temporal evolution of rigid body orientation angles in Fig. 10.10, one can see that for both the moderate and the extreme perturbations, the evolution is relatively

similar. Once again, due to the fact that the monohull geometry has the smallest moment of inertia, it is possible to observe that its orientation angle oscillation is much more significant than for the other two geometries.

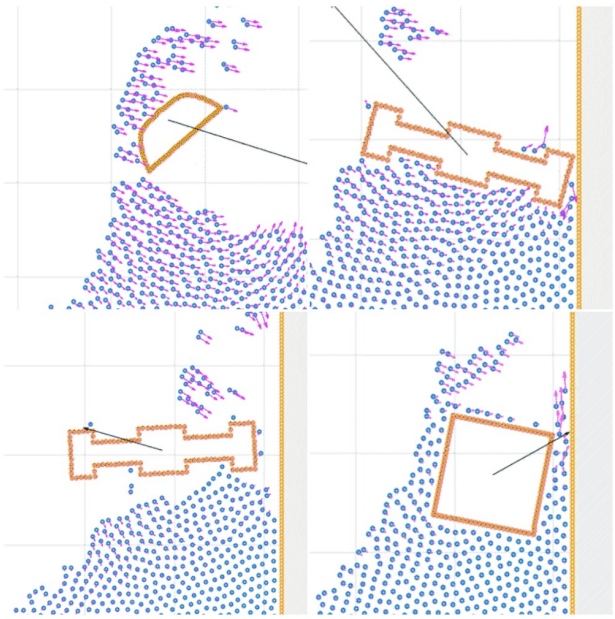

Figure 10.11: Edge effects affecting the orientation angle evolution

However, in the case of the perturbation caused by 300 particles, one can see a series of kinks in the orientation angle evolutions corresponding to the monohull and the multihull, at around $t = 1s$. After examining the animation displaying the free surface evolution when interacting with the rigid body, it is possible to conclude that these discontinuities appear due to, on the one hand, the collision of the rigid body with the boundary (Right up in Fig. 10.11), and on the other hand, to the separation of the rigid body from the fluid free surface (Left up and left down in Fig. 10.11). It is also worth noting that in the case of the block geometry, although no comparable kinks appear, there is still a point in the simulation in which the rigid body compresses a vertical layer of fluid particles against the right (East) boundary (Right down in Fig. 10.11), thus showing that boundary effects are still significant for this simulated case. A last interesting remark regarding the evolution of rigid body orientation angles is that while as both the monohull and the multihull geometries are able to correctly stabilize their orientation after a certain amount of time, one can not say the same for the

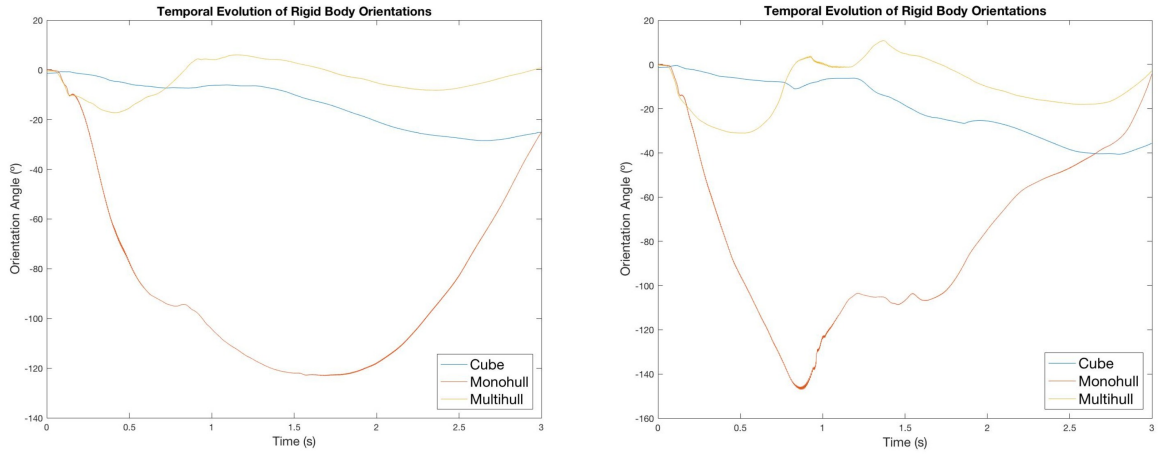

Figure 10.10: Temporal evolutions of rigid body orientation angles. Left: Moderate perturbation, Right: Extreme perturbation

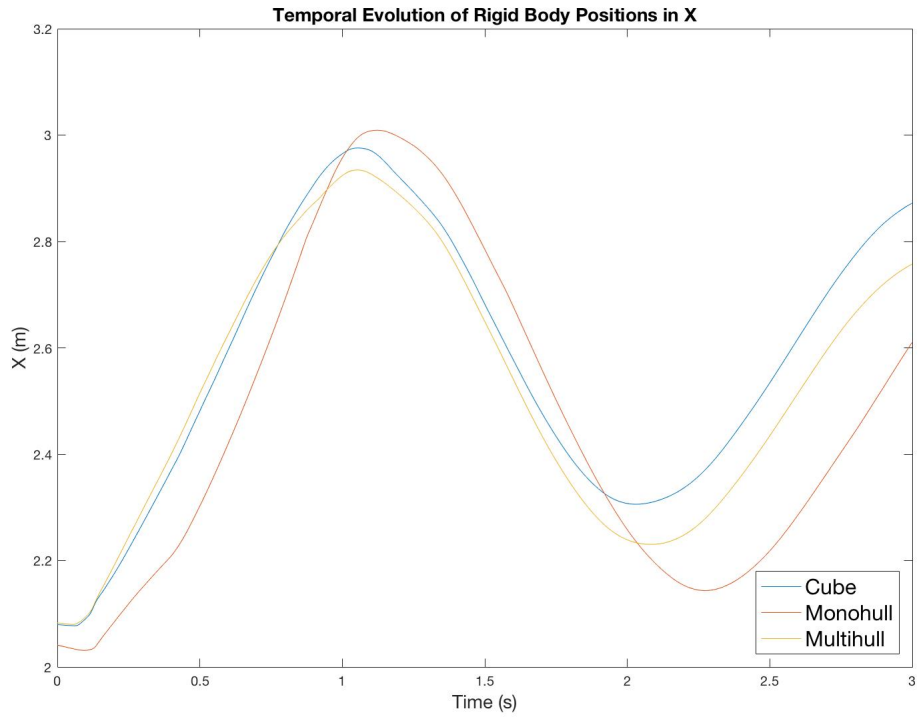

Figure 10.12: Temporal evolutions of rigid body positions in X for a moderate perturbation

floating cube. However, longer simulation times would be required to say so with complete certainty.

With regards to the evolution of the position in X of the rigid bodies in this simulated case, it is pleasing to see that the obtained results are close to a damped oscillatory motion, such that the initial perturbation causes a set of oscillations of decreasing amplitude. This is visible in Fig. 10.12.

Moreover, one can see a certain correspondence between this position evolution and the evolution of rigid body velocities in X; the velocity evolution shows a similar type of oscillatory motion, with a certain offset, as can be expected. However, the velocities portrayed in Fig. 10.13 display a large amount of discontinuities, especially before $t = 1.5s$. As these discontinuities are much more significant for the perturbation with 300 particles than for the more moderate perturbation, one could hypothesize that they appear due to boundary effects as the ones shown in Fig. 10.11, which are also more significant with the extreme perturbation. Other possible causes for these discontinuities include the density artifacts discussed in Section 10.3. Additionally, it is also worth noting that the evolution of the elevation of the simulated rigid body dynamics can also be seen as following a damped oscillatory trend.

Based on these results, one could affirm that the multihull body has the most stable geometry, not only due to the fact that it shows a less significant variation in its orientation angle when hit by a free surface perturbation, but also due to the fact that it returns to its initial stable orientation with greater ease than the other remaining geometries. With regards to the rigid cube, it is possible to affirm that it displays a smaller degree of orientation oscillation,

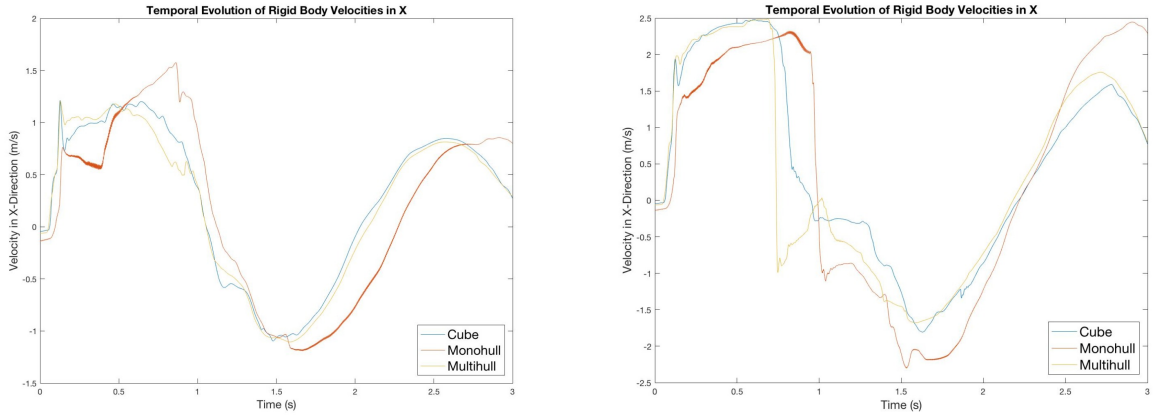

Figure 10.13: Temporal evolutions of rigid body velocities in X. Left: Moderate perturbation, Right: Extreme perturbation

albeit it does not stabilize itself once the perturbation fades. Lastly, the monohull geometry can be associated with the worst performance due to its large variations in orientation angle (reaching the point of capsizing), although it is able to restore its initial stable orientation.

11 Validation

Once the implemented SPH structure has been described and discussed, and its capability of simulating the interaction of a free fluid surface with a set of different rigid bodies has been proven, it follows to show to what extent the results obtained from such an implementation match those of a real physical case scenario. In order to do so, the validation test proposed in [9], based on [12], is performed. This validation test intends to simulate a case of rigid-fluid interaction in which the effect on the rigid body dynamics of a wave travelling along the free fluid surface is analyzed. While as in [12] the results of an implementation without any sort of SPH basis are compared with a set of experimental results, [9] compares that same experimental data with the results obtained when implementing the given case with the open-source SPH solver *DualSPHysics*.

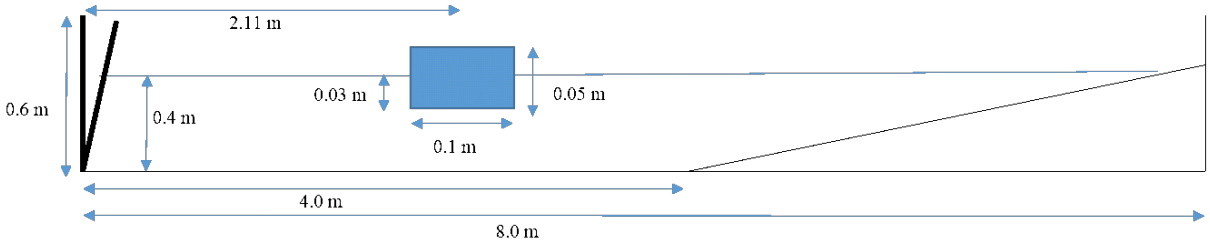

Figure 11.1: Configuration of the validation test presented in [9]

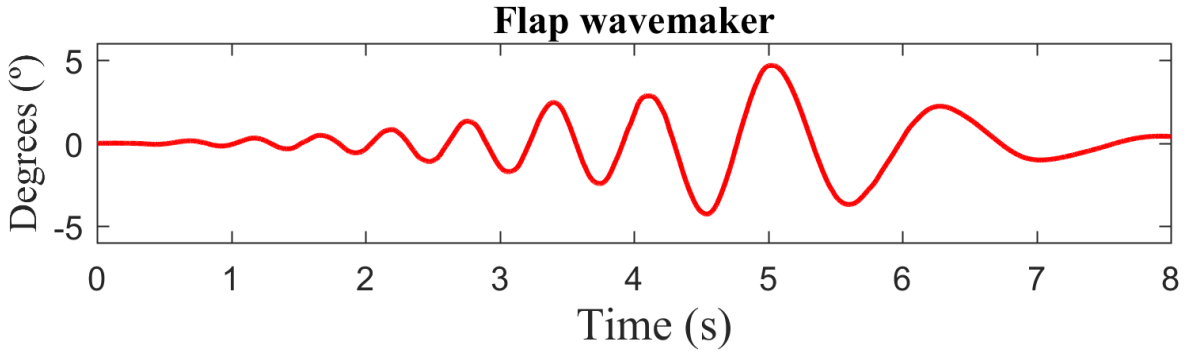

Figure 11.2: Wave generator oscillation [9]

The validation test is proposed following Fig. 11.1. Here, a floating rectangular rigid body is placed at an initial separation from the left wall and at a certain elevation, and a wave is generated by the wave generator placed at this same left wall. Moreover, a ramp is introduced at the opposite side of the domain. The wave generator works as an oscillating wall rotating

about the bottom left corner of the domain, and its oscillation follows Fig. 11.2. With regards to the results obtained in [9], one could say that the experimental results in [12] are approached adequately as visible in Fig. 11.3, and thus that a similitude between the results of this implementation and that of [9] would prove the validity of the here implemented SPH solver.

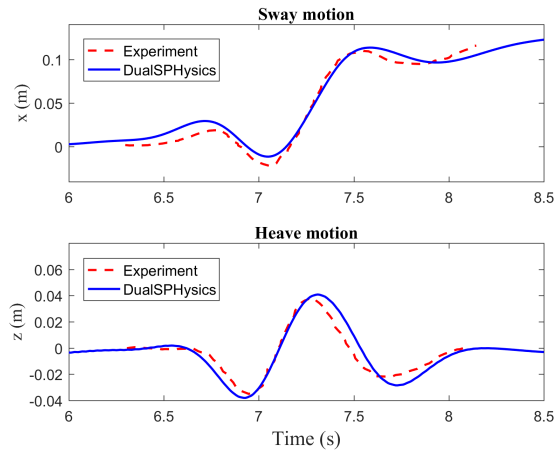

Figure 11.3: Comparison of results from [9] with experimental results

In order to develop such initial configuration, the developed SPH solver required the initialization of not only the free fluid surface but also the adequate introduction of the rigid body. This was performed similarly to Section 10. For the fluid surface initialization, the fluid particle lattice was spread out more evenly in order to reduce the required initialization time, and the fluid particle radius was defined as a function of the selected particle count in order to allow for an initial depth of $0.4m$. Thus, with 2000 particles, the fluid particle radius was $0.0071m$, and a particle mass of $0.4273kg$ was employed to approximate the initial lattice density to the fluid's rest density. Moreover, the rigid body initialization was performed by dropping the specified rectangular geometry from a point separated from the wall generator as specified but at a slightly larger elevation.

The results of this validation test show that the here implemented SPH solver is far from representing the experimental reality. This can be seen when comparing the sway and heave motion (motion in X and Y, respectively) of the floating body for both the solver employed in [9] and the one developed in the current study. In Fig. 11.4, one can see the huge difference in obtained results, as for the small time range for which the results of *DualSPHysics* are given, the shape of both curves is completely different. Note that as with the utilized initialization processes the imposition of both a precise fluid depth and an exact rigid body location is difficult, one could understand the appearance of an offset in the values for the temporal evolution of the presented sway and heave motions. Thus, one has to focus on the shapes of the curves in order to compare the fluid behavior properly.

First, one should point out that the measurements obtained with the current solver are performed once the motion of the wave generator begins, which is programmed to occur once both the initialization of the fluid surface, requiring $5s$, and the introduction of the floating prism, taking up $4s$, are performed. However, one can see that these initializations do not allow for static conditions in the rigid body motion, as the rigid body is moving at the start of the wave generation. This already presents a large difference with the results presented in [9], which although not present for the start of the wave generation, are considered to be those of a static fluid surface.

Another major difference between the resulting curves is in the period of the oscillations in the rigid body's motion. This divergence is easily visible in Fig. 11.5, showing a comparison of the roll angle evolutions of the simulated rigid body. While as the motion defined by

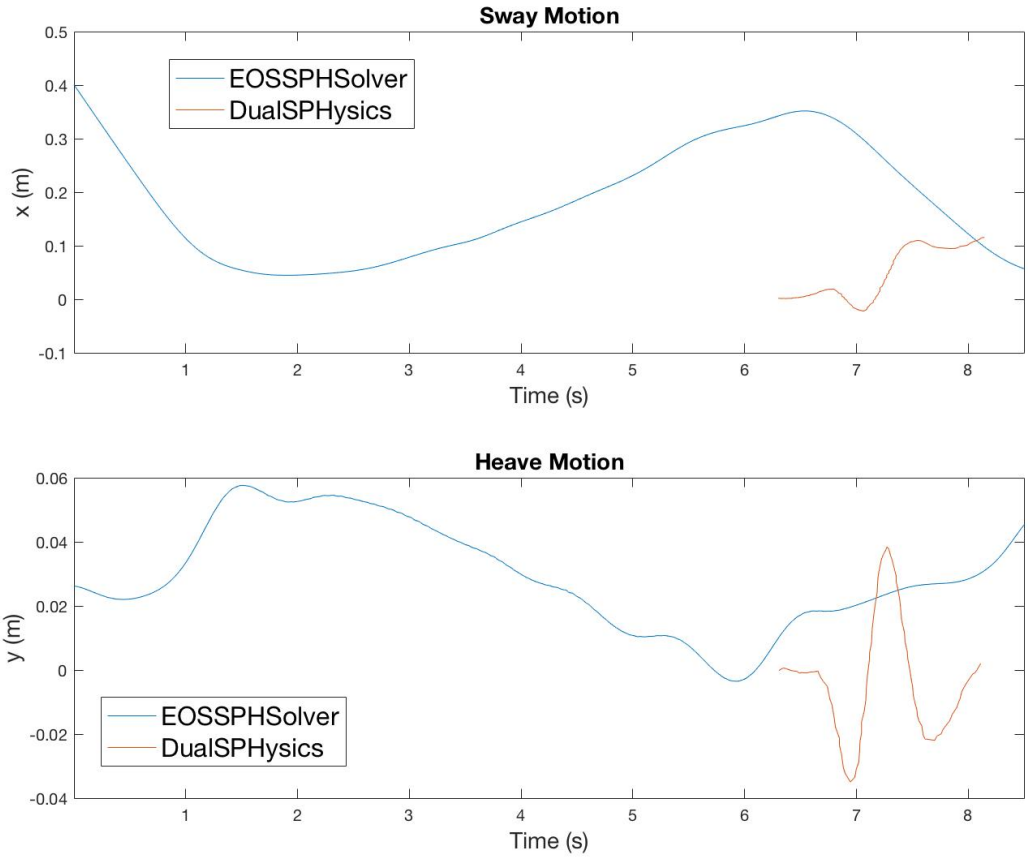

Figure 11.4: Comparison of sway and heave motion of the floating rigid body simulated in the current validation test

DualSPHysics is fast, with short characteristic periods, it is clear that the rigid body dynamics shown by the EOS solver here implemented are much slower. This makes one think of a divergence in the employed viscosity models. By observing the user's manual for the solver employed in [9], one can see that the same viscous model based on artificial viscosity is employed (see Section 5.1), but an additional model for laminar viscosity and sub-particle scale turbulence is applied [5]. As an in-depth description of this additional model would be out of the scope of this study, it will only be said that this additional term in the momentum equation is intended to take into account turbulent effects in the SPH simulation [5].

A further delineation of possible causes for this difference in results requires the description of the governing equations employed in *DualSPHysics*. While as this solver employs the same equation of state as described in Section 6.1, it also implements a version of the continuity equation, applying a so-called *delta-SPH* formulation. This formulation introduces a diffusive term in the continuity equation to reduce density oscillations [5].

In this particular exploration of SPH-based techniques for the simulation of floating bodies, one should also explore possible differences in the models employed when modelling rigid boundaries. *DualSPHysics* employs a Dynamic Boundary Condition, which defines bound-

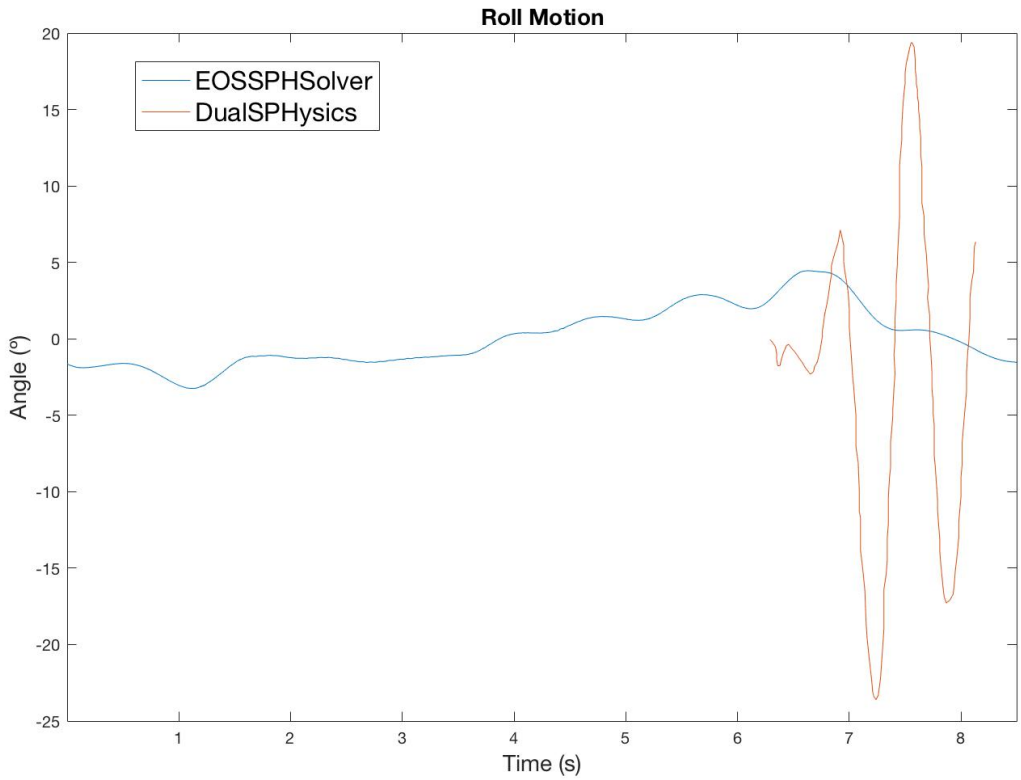

Figure 11.5: Comparison of roll angle evolutions of the floating rigid body simulated in the current validation test

ary particles that satisfy the same equations as fluid particles, but follow a desired, user-specified motion [10]. These boundary particles affect fluid particles by modifying their density once they are within the boundary's particle influence range, resulting in this way in a repulsive force acting on the fluid particle due to the pressure term on the momentum equation [10]. Moreover, similarly to the here employed method, *DualSPHysics* defines the solid floating body as a closed boundary surface, for which all of the boundary particles defining the body receive a force equal and opposite to the force exerted on neighboring fluid particles [5].

Thus, it can be concluded that the implemented SPH solver does not represent the experimental reality as would be desirable. This can be seen when comparing the results of the simulation of a specific case scenario regarding floating body dynamics with those of a professionally developed SPH solver which agrees with experimental results, as is *DualSPHysics*. By observing the structure of this reliable solver, one can see that while as some of the core concepts in the solver structure are shared with the here implemented solver, *DualSPHysics* travels the extra mile in order to include a set of additional models which approximate its results to the hydrodynamic reality.

12 Evaluation of the Study

The current study has sought for an adequate and physically sound way of implementing an SPH-based simulation of a floating rigid body. In order to do so, the form of each component of the implementation has been analyzed and its possible alternatives have been discussed. Moreover, after simulating a set of cases to observe such rigid-fluid interaction, the validity of this application has been tested with a specific case scenario, for which experimental data was available. After doing so, it follows to discuss the flaws and possible improvements that this study presents.

An issue which has significantly affected the quality and quantity of the obtained results is the performance of the implementation. As as said in [13], the Nearest Neighbor Search algorithm is one of the most time-consuming components of a typical SPH solver. Thus it is relevant to investigate to what extent does this faulty performance appear due to the NNSA. Although the neighbor search is performed only once for every iteration, not as in other types of SPH solvers, as for example certain variations of PCISPH [16], it seems that this subroutine accounts for a 13.6% of the time consumption of the SPH solver routine. This subroutine of the SPH solver is second in time consumption to the calculation of the advective forces relevant to each fluid particle, which itself accounts for 13.7% of the overall SPH solver time. On the other hand, one could join the required time to calculate the relevant values of both kernel function and kernel function gradient and obtain that this accounts for 18.4% of the SPH solver time usage.

Nevertheless, the SPH solver accounts for 35.4% of the overall simulation time. After taking a closer look, it appears that the generation of animations and plots relevant to the study, such as the density and pressure contours, or the illustration of the evolution of particle densities (see Fig. 10.4), is quite expensive. One could argue that storing the relevant values for these graphics and displaying or generating them in a separate routine would decrease the required computation time.

Another factor influencing the performance of the simulation is the limitation applied on the time step. It has already been said that the employed Tait EOS SPH solver is not the most efficient alternative that one can employ [16], which points to a possible notable decrease in simulation time if the implemented IISPH scheme were to function properly. In any case, it is definitely so that the required $\sim 6s$ to run an iteration of $\Delta t = 0.001s$ with the current EOS SPH solver is not optimum.

With regards to the coding style, it is worth noting that the usage of MATLAB was taken into account by employing array operations instead of *for* loops as much as possible, as can be seen throughout the script. This however sometimes required the usage of commands such as *find*, which tend to be time-consuming.

The significance of the performance in this specific study can be seen in the limitations which it imposes on the results of the simulation. A larger particle count in the simulated

cases would have easily improved the verisimilitude of the fluid behavior. Also, it would have allowed for larger domains with the same fluid depth and shape sampling, thus avoiding the edge effects discussed in Section 10.5. However, further increasing the particle count would have implied an excessively time-consuming simulation.

Had the main objective of this study been to perform a rigorous comparison between different hull shapes when hit by a wave, the domain size would have been required to be much larger, in order to simulate more physically sound conditions. Moreover, such a study would have required much more extensive fluid surface and rigid body initializations, as to allow for completely static conditions to occur. That is, to generate the wave once the rigid object is already floating in its corresponding hydrostatic equilibrium. One cannot see such static conditions after the initializations performed in the current study, as the performance of the implementation acts as a limiting factor on the simulation time. This reduction on the possible initialization time has also been seen to affect the results of the validation test performed in Section 11. With regards to the generation of free surface perturbations, it is clear that employing a wave generator similar to the one used in Section 11 is a much more elegant alternative than the introduction of additional fluid particles at an extreme of the domain. Not only does its elegance make it more preferable, but also the fact that it is much easier to relate to real physical applications, as it produces a perturbation closer to a common wave.

With regards to the work conditions of the current study, it is worth saying that one's independence has been a double-edged sword. While as there have been times in which the constant supervision of an expert in SPH would have resolved many of the obstacles encountered during the development of this implementation, the possibility for self-organization as well as the flexibility that it allows have even increased one's interest on the topic. It is worth propounding whether a work dynamic in which supervision were only present to solve any occurring doubts whilst still allowing for one's complete self-organization would have been better. In any case, this autonomous development has been an opportunity to develop a set of skills which are advantageous to the engineer.

13 Conclusions

In order to adequately perform an exploration of SPH techniques for the simulation of floating bodies, one should analyze the form that each of the components of the simulation should take in order to ensure that the results of the implementation are physically sound. With regards to the method employed to search for neighboring particles, a basic grid-sorting NNS algorithm is employed, as it provides efficient execution with a simple implementation. Moreover, the interpolating kernel function, the bread and butter of any SPH implementation, is chosen to be a cubic spline kernel, as it is simple and commonly used. With regards to the models used to calculate non-pressure forces, these are selected due to their conservation of linear and angular momenta and the avoidance of the calculation of the Laplacian of the kernel function. A large part of this exploration goes into presenting two methods of calculating particle pressures: a simple EOS solver which although not being incredibly efficient nor ensuring a large degree of incompressibility of the fluid, is very simple to implement, and a much more complex IISPH solver which, through the iterative resolution of a discretized version of the continuity equation allows for an almost incompressible fluid state. Albeit the larger efficiency of the latter method, its implementation is not achieved, and the former EOS scheme is employed.

After presenting a set of measures introduced to improve the stability of the integration of the fluid particle dynamics, the issue of handling the simulation's boundaries is tackled. Although many commonly used methods are presented, a simpler alternative is chosen to define these boundaries: to employ a sampling of boundary particles which contribute to the fluid's density and exert a pressure force on any neighboring fluid particles. This scheme is complemented by the imposition of reflective boundaries at the edges of the domain. Moreover, this scheme presents itself as a solid foundation for the introduction of a two-way rigid-fluid coupling, as one can define any required floating body as a closed boundary surface, which exerts an additional friction force on the fluid and is subjected to typical rigid body mechanics.

Once the employed structure of the solver has been presented, the set of simulated rigid-fluid interactions are presented. To employ three distinct rigid body geometries (a cube, a generic monohull, and a generic multihull), one initialization process is required to develop the free fluid surface, and another three initialization processes are needed to introduce each of these geometries into the free fluid surface. After these initialization processes are performed, the reaction of each of these geometries to a perturbation on the fluid surface is examined, and albeit the influence of the wall boundary at the edges of the domain, it is possible to conclude that the best performing geometry is that of the generic multihull.

In order to observe if the developed solver presents physically sound results, a specific case scenario analyzing the dynamics of a floating prism is simulated, and the results are compared with that of a reference SPH solver. Here it is seen that the implementation does not represent

the corresponding hydrodynamic reality, which may be caused due to several insufficiently realistic configurations in the developed solver. Furthermore, the developed solver is evaluated in terms of its performance, and the course of the study discussed.

By exploring and analyzing the possible forms that each of the components in the structure of a SPH solver can take, and selecting what is thought of as the best configuration in order to obtain solid results, one is taking a small step in the direction of developing a simulation tool that is helpful in the analysis of any hydrodynamic scenario. While as the here implemented SPH solver is far from being a respectable engineering tool, its development has helped not only in the understanding of a relatively new approach to Hydrodynamics, but also in acquiring the skills and notions required to develop any reliable CFD simulator.

Bibliography

[1] Cse885: Special topics-stellar modeling. section 4: Computational astrophysics. lecture 3: Ordinary differential equations. http://coffee.ncat.edu:8080/Flurchick/Lectures/StellarModeling/Section4/Lecture4-3.html.

[2] Lagrangian and eulerian specification of the flow field from wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Lagrangian_and_Eulerian_specification_of_the_flow_field.

[3] Sph-pukiwiki. http://www.slis.tsukuba.ac.jp/fujisawa.makoto.fu/cgi-bin/wiki/index.php?SPH

[4] Z-order curve from wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Z-order_curve.

[5] Users guide for dualsphysics code. 2016. http://www.dual.sphysics.org/index.php/sphysics-project/.

[6] SOLENTHALER B. AKINCI G. TESCHNER M. AKINCI N., IHMSEN M. Versatile rigid-fluid coupling for incompressible sph. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 30:72:1–72:8, 2012.

[7] ERTEKIN B. Fluid simulation using smoothed particle hydrodynamics. *MSc Computer Animation and Visual Effects, Bournemouth University*, 2015.

[8] TESCHNER M. BECKER M. Weakly compressible sph for free surface flows. *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 209–217, 2007.

[9] COLAGROSSI A. CRESPO A. Test 12: 2-d wave interaction with floating body - spheric. http://spheric-sph.org/tests/test-12.

[10] DALRYMPLE R.A. CRESPO A.J.C., GOMEZ-GESTEIRA M. Boundary conditions generated by dynamic particles in sph methods. *Computers, Materials and Continua*, 5:173–184, 2007.

[11] CANI M. DESBRUN M. Smoothed particles: A new paradigm for animating highly deformable bodies. *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, 2010.

[12] PERIC M. XING-KAEDING Y. HADZIC I., HENNIG J. Computation of flow-induced motion of floating bodies. *Applied Mathematical Modelling*, 29:1196–1210, 2005.

[13] BECKER M. TESCHNER M. IHMSEN M., AKINCI N. A parallel sph implementation on multi-core cpus. *Computer Graphics Forum*, 30:99–112, 2011.

[14] GISSLER M. TESCHNER M. IHMSEN M., AKINCI N. Boundary handling and adaptive time-stepping for pcisph. *Proceedings VRIPHYS*, page 79–88, 2010.

[15] SOLENTHALER B. HORVATH C. TESCHNER M. IHMSEN M., CORNELIS J. Implicit incompressible sph. *IEEE Transactions on Visualization and Computer Graphics*, 2013.

[16] SOLENTHALER B. KOLB A. TESCHNER M. IHMSEN M., ORTHMANN J. Sph fluids in computer graphics. *EUROGRAPHICS 2014 STAR*, 2014.

[17] MONAGHAN J. Smoothed particle hydrodynamics. *Re- ports on Progress in Physics*, 68:1703–1759, 2005.

[18] KELAGER M. Lagrangian fluid dynamics using smoothed particle hydrodynamics. *Department of Computer Science, University of Copenhagen*, 2006.

[19] ANTUONO M. COLICCHIO G. GRAZIANI G. MARRONE S., COLAGROSSI A. An accurate sph modeling of viscous flows around bodies at low and moderate reynolds numbers. *Journal of Computational Physics*, 2013.

[20] GROSS M. MUELLER M., CHARYPAR D. Particle-based fluid simulation for interactive applications. *Eurographics/SIGGRAPH Symposium on Computer Animation*, 2003.

[21] MUELLER M. POMERANETS D. GROSS M. TESCHNER M., HEIDELBERGER B. Optimized spatial hashing for collision detection of deformable objects. *Proceedings of Vision, Modeling, Visualization*, page 47–54, 2003.

[22] TOMÁS Á. Dynamics of a floating cube. *Computational Thermo-Fluid Dynamics, Technische Universitaet Muenchen*, 2017.