Real-time Fluid Simulations under Smoothed Particle Hydrodynamics for Coupled Kinematic Modelling in Robotic Applications

by

Gabriel Camporredondo Díaz

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in

Electrical Engineering, Electronics and Automation

Universidad Carlos III de Madrid

Advisors:

Dr. Ramón Barber Castaño

Dr. Mathieu Legrand

Dra. Lourdes Muñoz Gómez

Tutor:

Dr. Ramón Barber Castaño

January 2021

Acknowledgements

I thank God for providing me with all I needed to conclude this work. My especial thanks to my late Mom memory as she was my greatest mentor in life. I deeply thank Adri, my wife, for being so considerate and supporting throughout the years of my studies and work. I thank my Dad for been an unconditional companion and my Grandmother who encourage me all the time.

Dear Ramón, Mathieu, and Lulu, it is a tremendous privilege to meet you. Thank you for patiently share with me your experience and advice. I will never forget your time spent in long chats and large email replies. Lulu, thank you for never say no to this work. Mat, thank you for your patience and for being so neat in each of your feedbacks, you made me understand how fluids need to be handle with care. And Ramón, my advisor and tutor, thank you for accepting me as your doctoral student and for spending hours listening to my thoughts and polishing them for the sake of this work. Thank you for your care, not only to this research but also to any of my personal concerns during this long but rewarding time. Great respect to each of you.

Also, I would like to thank to the "Carlos tercero" University. Thank you for having a doctoral program of excellence and letting me pursue my dream as part of it. It is an honor to be a graduate student at UC3M.

Published Content

Article

G. Camporredondo, R. Barber, M. Legrand, L. Muñoz, **"Kinematic Controller for Liquid Pouring between Vessels Modelled with Smoothed Particle Hydrodynamics"**, *MDPI Appl. Sci. Basel*, vol.23, no. 9, pp. 5007, 2019. doi: 10.3390/app9235007

- This work is fully included and its content is reported in Section 4.3.
- The author's role in this work is focused on the designing, implementation and experimentation of the methodology proposed as well as leader in the writing of the full paper.

Conference

G. Camporredondo, R. Barber, M. Legrand, L. Muñoz, "**Assembling a GPU Physics Simulator for Scientific Training"**, *INTED2020 Proceedings*, pp. 2641-2650, Mar. 2020. doi: 10.21125/inted.2020.0791

- This work is partially included and its content is reported in Chapters 3 and 4.
- The author's role in this work is focused on the designing and implementation of the methodology proposed as well as leader in the writing of the full paper.

Conference

G. Camporredondo, L. Muñoz, R. Barber, M. Legrand, "**Course Content for Learning GPU Programming"**, *ICERI2020 Proceedings*, pp. 2641-2650, Nov. 2020. doi: http://dx.doi.org/10.21125/inted.2020.0791

- This work is partially included and its content is reported in Chapters 3.
- The author's role in this work is focused on the designing and implementation of the methodology proposed as well as leader in the writing of the full paper.

Abstract

Although solids and fluids can be conceived as continuum media, applications of solid and fluid dynamics differ greatly from each other in their theoretical models and their physical behavior. That is why the computer simulators of each turn to be very disparate and case-oriented.

The aim of this research work, captured in this thesis book, is to find a fluid dynamics model that can be implemented in near real-time with GPU processing and that can be adapted to typically large scales found in robotic devices in action with fluid media. More specifically, the objective is to develop these fast fluid simulations, comprising different solid body dynamics, to find a viable time kinematic solution for robotics. The tested cases are: i) the case of a fluid in a closed channel flowing across a cylinder, ii) the case of a fluid flowing across a controlled profile, and iii), the case of a free surface fluid control during pouring. The implementation of the former cases settles the formulations and constraints to the latter applications. The results will allow the reader not only to sustain the implemented models but also to break down the software implementation concepts for better comprehension.

A fast GPU-based fluid dynamics simulation is detailed in the main implementation. The results show that it can be used in real-time to allow robotics to perform a blind pouring task with a conventional controller and no special sensing systems nor knowledge-driven prediction models would be necessary.

Keywords: physics engines, smoothed particle hydrodynamics, GPU fast processing, real-time robot kinematics, computational fluid dynamics

Resumen

Aunque los sólidos y los fluidos pueden concebirse como medios continuos, las aplicaciones de la dinámica de sólidos y fluidos difieren mucho entre sí en sus modelos teóricos y su comportamiento físico. Es por eso que los simuladores por computadora de cada uno son muy dispares y están orientados al caso de su aplicación.

El objetivo de este trabajo de investigación, capturado en este libro de tesis, es encontrar un modelo de dinámica de fluidos que se pueda implementar cercano al tiempo real con procesamiento GPU y que se pueda adaptar a escalas típicamente grandes que se encuentran en dispositivos robóticos en acción con medios fluidos. Específicamente, el propósito es desarrollar estas simulaciones de fluidos rápidos, que comprenden diferentes dinámicas de cuerpos sólidos, para encontrar una solución cinemática viable para robótica. Los casos probados son: i) el caso de un fluido en canal cerrado que fluye a través de un cilindro, ii) el caso de un fluido que fluye a través de un alabe controlado, y iii), el caso del control de un fluido de superficie libre durante el vertido. La implementación de estos primeros casos establece las formulaciones y limitaciones de aplicaciones futuras. Los resultados permitirán al lector no solo sostener los modelos implementados sino también desglosar los conceptos de la implementación en software para una mejor comprensión.

En la implementación principal se consigue una simulación rápida de dinámica de fluidos basada en GPU. Los resultados muestran que esta implementación se puede utilizar en tiempo real para permitir que la robótica realice una tarea de vertido ciego con un controlador convencional sin que sea necesario algún sistema de sensado especial ni algún modelo predictivo basados en el conocimiento.

Palabras clave: simulador físico, hidrodinámica suavizada de partículas, procesamiento rápido con GPU, cinemática robótica en tiempo real, dinámica de fluidos computacional

Content

Chapter 5

List of figures and tables

Supplementary material

Video [4.1] KVS effect.
URL Access

Video [4.2] Control of profile rotation by input reference.
URL Access

Video [4.3] Slosh elevation motion for SIM.
URL Access

Video [4.4] Full kinematics performance under constant tilting velocity.
URL Access

Video [4.5] Lateral excitation of the slosh.
URL Access

Video [4.6] Full kinematics performance test of AFS with SIM.
URL Access

Video [4.7] CFS pouring not meeting h_q condition.
URL Access

Chapter 1

Introduction

1.1. Thesis motivation

The motivation of this work is based in the fact that currently insufficient model simulation is involved when robotics manages fluids in general. Furthermore, no model descriptor implementations are available to describe robot kinematics and fluid dynamics as an entity in the same domain of occurrence. Modern complexity models need to be developed to bring this to light leveraged by high-end computational technologies.

Fast computing based on graphical processing units (GPU) is now very popular and can be found in supercomputers, computers and mobile devices, enabling them to process high demanding data loads like those found in physics engines. Motivated by this progress and meanwhile the GPU parallel processing becomes the right arm of the physics engines, simulations of robotic kinematics should not be left behind to explore this potential in new applications like in those where robotic agents interact with fluids in free surface or underwater.

Motivated by GPU potential, robot kinematic and fluid dynamics can be coupled together allowing the users to interact in real-time with coupled simulations and apply possible control algorithms.

1.2. Main objective and challenges

The main objective of this work is, thus, ***to find a fluid dynamics model that can be implemented with GPU processing for studying robotic devices in fluid media***. For this objective, the following individual goals have been proposed for the development of the thesis:

- To study the possibilities and underline the limitations of one modern fluid simulation: in particular, the smoothed particle hydrodynamics (SPH) model with fast graphical unit processing (GPU).

- To develop a fluid-solid interaction model: a solid-fluid real-time interaction simulator for testing possible robotic kinematic control models, developing these fast fluid simulations and comprising different solid body dynamics to find a viable time kinematic solution for robotics.

- To find a fluid dynamics model implemented with GPU processing that allow fluid-robot model descriptors to be designed and conducted in further work with tested accuracy and well-known dependencies, having applications to validate the simulator model implementation.

- To implement a software simulator that allows coupling with kinematic structures for fluid-robot interaction.

Extensive research on the field of computational fluid dynamics (CFD) has been performed during the last decade using GPU parallel processing, especially in the field of computer graphics and scientific simulations. Yet, applications have not reached full potential in adding these fluid models into the robotic modelling to facilitate fluid-robot interaction. So, based on this, two main challenges are regarded:

- An initial challenge comes from the computational cost, where complexity of most fluid models is a relevant drawback for fast or real-time simulations. Although many fluid model implementations are available these days, low computational cost and accurate models are to be found and adapted to the robot time and space domains to enable kinematic control.

- A secondary challenge consists in building a simulator for motion kinematics that process data in real-time with no batch or offline preprocessing. In computer graphics, the fluid model works around simplification and visual effect, and no real physical data is engaged while the motion remains "realistic" to the viewers. In scientific simulations, though, kinematic requires to match physics reality. So, any motion performance will require special attention to find the best programming tools to implement real-time kinematic models of fluid-robot interaction. Data needs to be traceable in time in order to be able to compare results with literature data.

1.3. Contributions

The primary outcome of this work is to provide the resources to facilitate the development of a simulation software tool to couple mechanical components with fluid models, adding a contribution to the state-of-art in the field of fluid-robot control applications.

Robot kinematics involved in this work has no demanding computational cost requirements, but when this is coupled with fluid dynamics applications, this requirement turn to be of high-demand. Thus, the GPU is a base requirement for such applications. A secondary outcome of this work is to provide test cases of computational fluid dynamics coupled with solid kinematics using an interactive real-time feedback model based on GPU to match or enhance time response and performance of the equivalent stationary simulated cases.

In particular, a liquid pouring case is treated in detail to support the intended contribution. Here, a coupled fluid-solid model has been designed and implemented to finally provide a fine control on the liquid surface to avoid sloshing. ***This work has been finally published in MDPI Applied Science. Also, the study of the GPU programming that is developed in this thesis work allowed to contribute with two articles in related conferences. These published content can be found in INTED and ICERI conferences***.

1.4. Thesis structure and outline

This thesis work, as mentioned earlier, is the conjunction and application of two computational simulators; one, the fluid dynamics simulator and, the other, the robotic (solid) kinematics simulator. The *State-of-art*, captured in Chapter 2, Section 2.1, studies the evolution of the two types of simulators, into what research and software industry have converged and unified with the name of physics engines with GPU processing and its latest applications in the field of robotics. Section 2.2 covers the fluid dynamics theory fundamentals of the models to develop in the following chapters.

In Chapter 3, *SPH model and proposed implementations*, the fluid dynamics model used in this work is extensively described. This description allows the reader to differentiate the model from the particularities of its posterior GPU implementation. Section 3.1 defines the constitutive equations that covers the fluid dynamics model employed. Section 3.2 provides the details of the CFD model: the smoothed particle hydrodynamics, including its principle, advantages and limitations. Sections 3.3 and 3.4 deal with the specific treatment of the boundary conditions and the chosen algorithm for time integration. Section 3.5 describes the particular implementations of the GPU processing structures (or Kernels) to solve the SPH formulation used in this work. Final implementations, in Section 3.6, will allow to find the software specifics that embrace all the model concepts into a final simulator.

Chapter 4, *Coupling robotics and fluids*, describes the three cases tested in this work with real-time implementation and control design. Figure 1.1. shows the methodology of the proposed simulator; initially domain (geometric) model specifications are provided, then, these are fed as input into a coupled/uncoupled fluid dynamics and robot kinematics processing engine, and finally an expected design and control settings are obtained. A redesign feedback is use to redefine robot kinematics and/or fluid dynamics on the fly by allowing the user to interact in real-time with the simulation process.

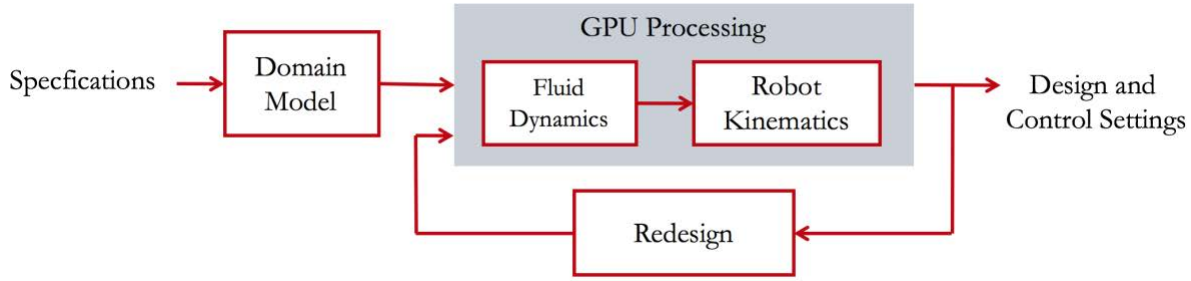


Figure 1.1. Robot kinematics and fluid dynamics block diagram for a closed-loop design system.

In Chapter 4, Section 4.1 presents the benchmark case used for static solids validation of fluid motion. This case tests the well-studied development of the von Karman vortex street (KVS) formation in a closed channel and compares results with abundant literature data. In Section 4.2, a flapping underwater airfoil -or hydrofoil- (NACA profile) is simulated in a closed channel flow. The profile is first modeled in free rotation and later as a controlled agent. The positive results found in this simulation allow to develop a third and final case of a coupled fluid-solid simulation. The third simulation case, in Section 4.3, describes a pouring control model in free surface and the resulting kinematics using closed loop control approach. A resulting final time response of the pouring action is provided.

Results and discussion for each individual case are included at the end of each case Section of Chapter 4. The discussion offers a detailed debate on the overall research throughput. Nonetheless, a special interest is placed in the pouring case results as it provides the performance of the implemented simulator including partial results of the former cases and their fluid and solid models.

Conclusions of this thesis are drawn in Chapter 5, that also offers some lines of further work to follow and deepen the research work to develop new coupling models to validate physics for a robot-fluid ecosystem.

Chapter 2

State of the art

A motion simulator models kinematics of objects to obtain numerical results that can describe their behavior in real physical world. This type of simulators add knowledge to different applications such as, robotics, skill training, scientific exploration, visualization, mechanical prototyping, and medical diagnosis, to name a few. Although the applications are extensive, to match real experiment data, the simulation cases are simplified. This is especially true for real-time applications or in objects with complex geometries, where massive data is processed and the physical modelling adds relevant computational cost. Such is the case of the simulation of computational fluid dynamics.

On the other hand, the interaction in 3D virtual scenarios of computer simulators, that are employed in game applications, provided live user experience featuring the user to guide and redefine the game behavior on the fly. A key feature not only for games engines but also for scientific purposes in physics engines. In many ways, the computer game industry leads the state of the art of computer interaction and continues to evolve quickly due to the improvement of fast parallel computing using graphic processing units (GPU). This GPU technology allows to process intensive data loads even in today standard computers and mobile devices. There are some limitations on regard to GPU usage though; like the memory block exchange speed [1]. However, while this technology continues expanding, the possibilities in using it in computational physics engines are increasing widely. The GPU capabilities in simulators are the massive data processing, the remote and mobile usage, and the real-time graphical user interaction. These

capabilities are precisely the required computing features of this work application: a solid-fluid real-time interaction simulator for testing robotic kinematic control models.

In the next Section 2.1, a brief description of the background of interactive simulators that make use of GPU computing to solve physical models are described, just as in the aim of this work. In the Section 2.2, theoretical and numerical computational fluid dynamics (CFD) methods typically used in fluid simulators with fast computing are described, dividing them in those with eulerian and those with lagrangian formulations.

2.1. Physics engines

The term used to describe the computer software employed to simulate dynamics in physics has been commonly accepted as "physics engines". To avoid misunderstanding, the reference terms used in this work will be physics engines or physics simulators indistinctly, such as those employed within the robot kinematics described in [2]. Many physics engines have been developed in the field of computer graphics, leading to contribute in the engineering experimentation, design and research. Such is the case of the computer aided design (CAD) software, where computational fluid dynamics methods, like the finite element method (FEM), have been used to simulate the stress regions of designed geometries that are subject to mechanical loads [3].

CAD-based physics engines have been the foundation of the real-time physics simulators. The "real-time" term in simulation usually refers to the property of conducting simulation tests on a time dependent scale or timeline that allows to run the simulation seamlessly along the user interaction. Physics game engines are the result of the concurrency of CAD, CFD and GPU technologies for real-time and interactive systems. Actually, the game industry is one of the most advanced in applying physics where many state-of-the-art CFD methods are implemented [4, 5]. Nonetheless, the target of game engines is the user perception experienced throughout the development of a scene reality and not the motion kinematics aligned to real world physics. Such target can be found in the engineering software architectures. Blender [6] is an enhanced engineering simulator framework where motion kinematics and CFD implementations for fluid and solids allow for game development as well for engineering simulations. This type of hybrid

system is formed by a framework core and a set of code layers that are bonded together in a timeline so motion kinematics for solids and fluids can be tested in a 3D scene. [6] and other similar commercial systems allow the user to carry out physics simulations with simple parameter configuration or by adding programming (scripting) functionality.

In robotics, the kinetic and dynamic simulations have a wide range of computer tools that can be found depending on the application. Despite of how heterogeneous robotics applications are, it is commonly simulated in the numerical computer software [7] for its motion kinematics and dynamics study. Other simulations are coded from the ground up though. That is the case of robotics applications like ours where robotics handle fluid volumes for pouring liquid. Such application has been studied from different approaches. One approach consisting in tracking the flow trajectory by computer vision [8]. CFD is used as action constrains but it is an uncoupled model yet, where fluid and solids kinematics work separately. Another model-free approach in [9] uses deep learning techniques based on computer vision to perform liquid pouring. Davis in [10] provides a commonsense study of liquid pouring where the study suggests the development of different theories that will integrate a reasoning to apply artificial intelligence algorithms. And last, the model-based approaches [11, 12] where pouring kinematics is driven by a fluid model, including the free surface model or the volume model. As the pouring task suggests, a free of error control performance is required, thus, as the approach is fully deterministic, it is ideal to follow a model-based one for such applications. In this work, the intended pouring case is based on this approach that includes the selection of a CFD method and a free surface model to fully couple with the pouring kinematics.

Whether is a game engine, an engineering simulator or a hybrid framework combination of both, the application of physics kinematics is a requirement today. Such requirement includes at least the usage of a physics engine where solid objects can react to physical collisions between each other and between them and the boundaries of the space domain where they are allocated. This space domain can be a *n*-dimensional euclidean representation, most of the time a hexahedron or a rectangle space for 3D or 2D respectively. In most of the available physics engines there is a force-based scheme where each object body in the scene has a composed *n*-dimensional force. This force is composed by the forces of interest. It may be gravity added with a collision acceleration for solid body objects [13], or pressure gradient and viscous shear for fluids [14]. Because of these body interactions the composed force may change in each discrete time step of the simulation as well as the body position. Thus, this scheme requires a discrete

integrator that translate driving force into body position and orientation. The study in [15] provides an evaluation of real-time physics simulators systems for many aspects such as the integration scheme performance. As suggested in [15] and extensively developed in [16], the collision force methods require to be as robust as needed in a physics-based simulator. From primitive geometry collisions to bounding volume collisions, there are many force compositions that can be chosen to fit a desired performance in simulation.

Apart from all-in-one simulation solutions and thanks to software portability, there are individual software components, library packages and interfaces that have been developed to provide simulation functionality to core systems, and many of them, to plug in to their own code for rapid development. For instance, Bullet [17] and ReactPhysics3D [18] are two C++ libraries for real-time collision detection of rigid objects including time integration solvers.

Another important feature of physics engines is the real-time performance. The tasks in a physics engine include the scene graphics creation, the physics primitives and collision detection and the user interaction, all running seamlessly in time. The scene creation or the space domain is usually developed under a 3D computer graphics standard for most of the physics engines. OpenGL [19] has been the most used application programming interface (API) for scene graphics creation in cross-platforms but there are others that have same functionalities [20]. Typically, no concerns are related to this task and programmers delegate the computer graphics programming to this type of libraries because of their robustness in dealing with graphical hardware.

The user interaction task is not of a minor relevance. As mentioned earlier, the user interaction defines the real-time feature which allows the user to call events in a discrete time running simulation. Such events may come from different computer input sources and will trigger programmed or on demand functionalities. Different libraries and APIs [21-23] are available depending on the selected operation system and programming language. But for a better selection, as timing is crucial in real-time applications, an option with timing control is preferred. Timing control will allow to synchronize the time step that the simulation data refresh on the screen and the time step for the integration of forces and the position update of the rigid bodies. In this work, an API with time control is selected and described.

The geometry elements or primitives that constitute a solid body can be selected based on the type and complexity of the solid body to simulate. In CAD, triangles are usually the common primitives for rigid and flexible bodies because they can be grouped in meshes to form simple and complex geometries. Fluid zones, processed with CFD methods, are typically constructed with meshes or even particles as their primitives. Those two primitive elements can be used to describe the fluid dynamics from two fundamental frames of reference. One, the lagrangian frame of reference and, two, the eulerian frame of reference. These two frames of reference on which CFD models and ultimately physics engines are developed, will be described in next Section 2.2.

2.2. Computational fluid dynamics

As mentioned earlier, the aim of this work is to find a fluid dynamics model that can be implemented with GPU processing for studying robotic devices in fluid media with a nearly real-time capacity. Such challenge may force to select a fast model to avoid computational cost issues, at a certain expense of precision. GPU technology has quickly improved in fields like computer graphical rendering, and, to these days this is no longer a relevant issue in this field for solving models based on SPH [14, 24, 25] in comparison with other fluid methods with the same spatial resolution. This is an important reason to select a SPH based method in this work. Both Navier-Stokes equations and SPH formulation are based on the conservation equations that are described in the following sections.

2.2.1. Theoretical models

In CFD [26], the lagrangian description of the conservation equations defines fluid entities or particles that carry their own fluid properties like pressure, temperature, density or momentum. Along time, such properties can change and tracking motion of these fluid entities will describe the fluid motion itself. Therefore, conservations of mass, momentum and energy apply to each individual fluid particle.

The other description is the eulerian one. This description allows to find the evolution of the fluid properties at every point in space and time. Thus, this is a field descriptor. Such field properties of fluid depend on an individual location in space and in time. The following Sections provides the notations of the lagrangian and eulerian descriptions specifically for mass and momentum. Conservation of energy equation is not included here for clarity purpose for the following content that will focus only in isothermal fluids.

2.2.1.1. Eulerian formulations

If the viscosity (inviscid flow) and the external forces, like gravity, are neglected, the conservation equations in fluid mechanics have the following eulerian notation in Equations (2.1) for mass and Equation (2.2) for momentum for incompressible fluid:

$$\frac{\partial \rho}{\partial t} + v^{\beta} \frac{\partial \rho}{\partial x^{\beta}} = -\rho \frac{\partial v^{\beta}}{\partial x^{\beta}} \tag{2.1}$$

$$\frac{\partial v^{\beta}}{\partial t} + v^{\alpha} \frac{\partial v^{\beta}}{\partial x^{\alpha}} = -\frac{1}{\rho} \frac{\partial p}{\partial x^{\beta}} + \tau^{\alpha,\beta} + f^{\beta} \tag{2.2}$$

ρ, t, v and x are density, time, velocity and position. The superscripts β and α denote the generalized coordinate space directions. The first term to the left in Equations (2.1) and (2.2) refers to the fluid acceleration, while the second refers to the convective acceleration. The negative term to the right defines, by convention, the direction of the mass or flow. $\tau^{\alpha,\beta}$ is the viscous stress and f^{β} the volumetric force.

2.2.1.2. Lagrangian formulations

The governing equations of an isothermal fluid in the lagrangian formulation are Equation (2.3) for the continuity equation, and Equation (2.4) for the momentum equation.

$$\frac{D\rho}{Dt} = -\rho \frac{\partial v^{\beta}}{\partial x^{\beta}} \tag{2.3}$$

$$\frac{D\rho v^{\beta}}{Dt} = -\frac{\partial p}{\partial x^{\beta}} + \tau^{\alpha,\beta} + f^{\beta} \tag{2.4}$$

Notice the right term from the eulerian and lagrangian equations that change by applying the material derivative relation $\frac{D\varphi^{\beta}}{Dt} = \frac{\partial \varphi^{\beta}}{\partial t} + v^{\alpha}\frac{\partial \varphi^{\beta}}{\partial x^{\alpha}}$, where φ is a conservative quantity. It is worth to notice that the lagrangian formulation presents the advantage of eliminating the non-linear term of the convective acceleration.

Water is assumed as an incompressible Newtonian fluid, and this is the type of fluid of interest for robotics applications like in this work. The elemental fluid forces involved in such type of fluid are the viscous stress and gravity. Many other forces may arise as well, like the surface tension, but those can either be included or not depending on the fluid model of interest. So, if the former terms are added to the momentum formulation in Equation (2.4) and rearranging ρ, Equation (2.5) is obtained.

$$\rho\frac{Dv^{\beta}}{Dt} = -\frac{\partial p}{\partial x^{\beta}} + \mu\frac{\partial^2 v}{\partial x^{2(\beta)}} + f^{\boldsymbol{\beta}} \tag{2.5}$$

$$\rho\frac{D\boldsymbol{v}}{Dt} = -\boldsymbol{\nabla} p + \mu\boldsymbol{\nabla}^2\boldsymbol{v} + \rho\boldsymbol{g} \tag{2.6}$$

Equation (2.5) is rewritten in Equation (2.6) by substituting the partial derivatives by their equivalent gradient (∇) and laplacian (∇^2) operators described in Equations (2.7) and (2.8).

$$\nabla p \equiv \left[\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}, \frac{\partial p}{\partial z}\right] \tag{2.7}$$

$$\nabla^2\boldsymbol{v} \equiv \begin{bmatrix} \frac{\partial^2 v_x}{\partial x^2} + \frac{\partial^2 v_x}{\partial y^2} + \frac{\partial^2 v_x}{\partial z^2} \\ \frac{\partial^2 v_y}{\partial x^2} + \frac{\partial^2 v_y}{\partial y^2} + \frac{\partial^2 v_y}{\partial z^2} \\ \frac{\partial^2 v_z}{\partial x^2} + \frac{\partial^2 v_z}{\partial y^2} + \frac{\partial^2 v_z}{\partial z^2} \end{bmatrix} \tag{2.8}$$

Equation (2.6) includes three terms to the right: the pressure term $-\nabla p$, the viscosity term $\mu\nabla^2 v$, and the gravity term ρg. The viscosity term is given by the dynamic viscosity, μ, and the laplacian operator of velocity $\nabla^2 v$.

2.2.2. Numerical models

Based on the former theoretical formulations, two numerical solvers for NSE can be classified. One class based in meshes (finite volume method), and the other based in particles (mesh-free or mesh-less methods) [27]. In the following Sections, the two are described to allow the reader a broad background of the selected numerical model employed in this work for the robotics application.

In the space domain (dimension scale), a full tracking of the fluid properties that act within and/or around the solids is required; usually to track the force vectors or the velocity field of the fluid. By dismissing the group of eulerian methods that use the GPU acceleration [28, 29] because of its remeshing computational cost, a lagrangian method has to be selected to the application of this work.

Nevertheless, in the following two sections some of the numerical methods within the two formulations of their frame of reference are described to highlight the advantages of lagrangian formulation in the context of complex and moving solid bodies.

2.2.2.1. Numerical methods based on eulerian formulations

As mentioned earlier, the eulerian formulation provides a field description of the fluid. In many cases, this field is represented by a grid spatially fixed in space and no motion of the grid nodes is used to improve computational cost of remeshing. The mass, momentum and energy are computed across the grid on the fluid volume boundaries. The advantage of the eulerian grid computation is that no deformation is presented to the simulated fluid (or matter for the general case), and thus, numerical issues are minimized.

Although, apparently simple, constitutive equations are not linear and continuity and momentum equations are coupled. When the flow is laminar, the computational cost is not so important, but the requirements grow quickly as Reynolds number, Re, increases. In order to handle turbulence effects, these models need more and more computational nodes (in proportion of $Re^{9/4}$) and smaller time steps. In that sense, Direct Numerical Simulation (DNS) can describe,

in a deterministic form, all the turbulence scales for accurate results, at the expense of a tremendous computational cost.

Aiming at reducing the requirement in terms of calculation nodes (mesh size), the Reynolds-average Navier-Stokes method (RANS) is an example of a numerical eulerian method used to model turbulent flows [30]. It proposes an approximation of NSE to have a relative small computational cost compared with the direct numerical simulation (DNS) where NSE are numerically solved including the complete range of spatial and temporal scales of turbulence. Such computational advantage in RANS is found by permitting a solution of NSE decomposing the variables in a mean part and a fluctuating part. The mean quantities remain and the fluctuating quantities vanishes except for the viscous turbulent tensor which is modelled. For steady flows, RANS equations do not contain time derivatives and the solutions is even less costly.

Large eddy simulation (LES) is a numerical eulerian method too [31]. It is well used to describe a good fraction of the turbulence in flows. It uses a low filter approach to ignore the small length scales of turbulence that generates high computational cost. It can produce as much information as required in the small scale of turbulence allowing to reduce the induced error by passing only large eddy formations. From this attribute its name. In contrast with RANS, LES is more accurate and numerically expensive. However, nowadays with novel computer performance, more useful to describe detail in the turbulent regime. For such differences, the applications range between both models and its selection depends upon the case of study.

Another eulerian mesh-based numerical method is the volume of fluid method (VoF) [32]. VoF is a modelling method for free surface tracking. It is very useful for incompressible multi-phase simulations of fluids, specially, to model air-liquid interfaces. However, the more detail in free surface deformation is demanded, the more complex and finer the mesh should be. This produces a high computational cost and for this reason real-time applications could be out of its scope.

2.2.2.2. Numerical methods based on lagrangian formulations

Numerical method based on lagrangian formulation can be set on a grid (mesh-based) or on independent nodes or particles (free-mesh method). Finite element model (FEM) is an example of a mesh-based method, but its applications to fluids is limited. Its potential lies in simulating stress regions in mechanical parts during dynamic tests.

Lattice-Boltzmann method (LBM) is another example of lagrangian formulation method commonly used in CFD applications [33] using particles in a free-mesh method. LBM does not solve NSE, instead, it applies the mesoscopic kinetic equations on particles to obey the macroscopic behavior of a fluid, in other words, the NSE behavior. The main advantages of LBM to be considered towards the point of view of this work are the direct arithmetic calculation, the ideal disposition for parallel processing and the ability to simulate complex solid geometries and multi-phase flows. Two important weaknesses are that it is based on squared or cubic grids (lattice grids) and it is difficult to simulate curved grids, and that it is numerically not stable for small viscosity values. In robotics and specially in this work, some very useful applications are the free surface simulations. In LBM, free surface applications can be adapted by relabeling and reconstructing incomplete lattice cells to fulfil mass and density distributions [34]. Such tasks generate moderate overhead to the algorithm implementation and discourages to be used indistinctly between closed channel and free surface simulations.

There are other lagrangian numerical methods to mention from their importance in different fluid applications, like the moving particle semi-implicit method or the discrete lagrangian method. However, to keep the list consistent with this thesis work, the last lagrangian method to mention is the SPH method. As mentioned earlier, this is the selected method in this work. It was first introduced by Gingold and Monaghan [35] and Lucy [36] in 1977 for astrophysics problems. The principle, formulation and implementation will be provided in depth in Chapter 3, however, our choice is based in three key strengths.

- First, the ease to parallelize the implementation using adopted Kernels and parallel-based particle search algorithms. A very useful advantage for real-time and interactive application.
- Second, the minimum or null requirements to implement the SPH method when switching between free surface and closed channel applications. In distinction to eulerian methods, lagrangian one do not need to track in time the free surface nor to mesh at each time step

close to this interface. The importance of not having to compute a mesh allows to process conveniently faster for real-time simulation. This is also true when a solid moves into the fluid.

- And third, since SPH depends on NSE, fluid-solid interface forces, like the adhesion force or others, can be included to the method with relative ease.

One important limitation in SPH is the formulation of boundary conditions. In general, boundary handling algorithms are not totally aligned to physics. Another limitation is that incompressibility needs to be imposed. Such limitations would not limit the computational performance though, and for robotics applications the dimensions allow to disregard the impact of boundary conditions to some extent. Finally, weakly compressibility approach is commonly adopted in SPH for being very efficient computationally and for providing accurate numerical results.

2.2.3. CPU vs GPU implementations in CFD

Graphics cards were introduced back in the 80's, but the GPU term became popular in the late 90's. Graphics cards are made of GPUs, video memory and peripherals, and historically the first to use them extensively was the game industry while other sectors started to find its potential later when the GPU evolved from only a video processing device to a general purpose intensive processing device. Such more recent definition of the GPU is defined by the parallel architecture that constitutes its concept where data can be transferred and processed in parallel operations. Contrary to the conventional central processing unit (CPU) where parallel processing is intrinsically related to the software implementation. However, as far as the algorithm implementation is parallelizable, the GPU architecture will enhance the processing time over the CPU computation. If the algorithm implementation is purely sequential (meaning that each operation in the algorithm depends form its previous one), despite the great potential of the GPU, both, the CPU and the GPU will offer similar performance. This underlines the parallel characteristic of the algorithm to be implemented by the GPU to allow parallel shortcuts and time cost enhancements. It is also important to mention that CPU technology join parallel programming libraries like OpenMP [37] which also provide fast processing through parallelized algorithms. There are also libraries that combine CPU and GPU processing providing the best

from both worlds, fast sequential and fast parallel computations. This processing mode is also known as heterogeneous computing or general purpose computing on GPU (GPGPU).

In CFD methods, the common denominator is the intensive processing of data, mainly because of the computation of discretized partial differential equations that constitute theoretical formulations. For this reason, GPU processing is a perfect fit for CFD when the implementations can be parallelized. Such is the case of LBM which was initially tested on GPU (Nvidia GeForce FX 5900 Ultra with 256 MB of video memory) by Li et *al.* [38] where the speed was 8 to 15 times faster than in the CPU (Intel Pentium IV 2.56GHz with 1GB of RDRAM) computation. In more recent investigations and with recent GPU technology, Marsh [39] found the speed much faster for the LBM with a 50 to 75 factor of improvement, having an Intel 2.5 GHz Core 2 Duo CPU vs a NVIDIA state-of-the-art GPU.

Initial research using GPU for a fluid particle method, not SPH per se, can be found in [40] from Kipfer et *al.* in 2004. There, the main topics of a lagrangian discrete method is provided: particle emission, sorting, collision and rendering, to name a few. Kolb and Cuntz in [41] describe the SPH method using the GPU and contribute with the parallel data structures to be used by the GPU. They provide initial SPH simulations with fluid filling a cup, and the results are compared in term of number of particles vs frames per second. Other visible pioneers were Harada et *al.* [42] who formally describe SPH tasks in parallel structures to be processed by the GPU. One interesting contribution from [42] was the parallelized neighbor search algorithm that was fully processed on the GPU.

In conclusion, the dominance of the GPU over the CPU programming in the SPH implementation is the possibility of the CPU to delegate parallel tasks to the GPU decreasing the computational overhead and the direct bounding to the video memory.

From these contributions, in a short period of time, many improvements have been made to parallel SPH [14, 24, 43, 44]. Specially because of the interest in the target application of visualization and graphics where time performance is critical. Hoetzlein and Höllerer in [45] provide a performance study of different SPH implementations comparing their efficiency by number of particles, frame per seconds and GPU hardware.

Scientific computing applications where modelling and simulation is involved is less visible than computer graphics in SPH, but equally important. SPH simulation cases like the dam brake study for free surface flows [46] or the Karman vortex street effect in closed channel flows [47] have effectively simulated and studied. Many other science applications can be found as well, like the study of snow avalanche with SPH [48]. In robotics, the SPH method has not been developed as an application field yet, however, the interest to simulate fluid volumes behavior in moving reservoirs is a common application [49] very interesting to migrate to one or two degrees of freedom motion kinematic in an automatic liquid pouring controller.

In fact, the liquid pouring simulation is the final test case implemented in this research work which in different model properties and implementation details contributes to this state of the art in the field of applied robotics.

Chapter 3

SPH model and proposed implementation

In this Chapter, the SPH fluid method is developed, giving to the reader a clear description of this fundamentals. The selected variants and possible issues risen in such concepts are adapted to the final application within this work, and a clear explanation of the separation of conceptualized theory and its implementation structure is provided. Section 3.1 describes the constitutive equations that formulates the Navier-Stokes lagrangians equation in fluid flows. Section 3.2 provides the basic idea of the SPH method by using interpolations and also provides the relation between spatial resolution and the length of influence of such interpolants. Section 3.3 deals with the spatial domain characteristics and the boundary conditions used in this work. Section 3.4 describes the time integration schemes typically used in the SPH method and the selected one for this work. Section 3.5 offers the detailed GPU implementations of each parallel task. Finally, Section 3.6 provides the final thoughts and software structures that constitute the real-time system proposed in this thesis objectives and contributions.

3.1. Constitutive equations

As mentioned earlier, the lagrangian description of matter is based on particles that are able to transit in space, each of them carrying the properties of the matter such as, for instance, the pressure and density of the fluid matter. Such description allows to track conceived fluid volumes

in domains defined by boundaries that may change over time without much concern about domain constrains like it would in the eulerian description. This advantage is key to study fluids with moving solids in real-time simulations. In the next section, the lagrangian formulation of the NSE is presented, giving continuation to the theoretical model described in Section 2.2.1.2 that will provide the fundamentals of the selected SPH method.

Next sections are dedicated to the particular lagrangian formulation of the equations of conservation; mass, momentum and energy. However, this thesis is restricted to isothermal and incompressible flows so the energy equation is not required to be developed further.

3.1.1. Continuity formulation

The continuity equation represents the mass conservation. Then, the lagrangian formulation for individual particles is Equation (3.1):

$$\frac{Dm}{Dt} = 0 \tag{3.1}$$

In SPH, each simulated fluid particle is given a mass m_i, so that the mass conservation is intrinsically granted. However, incompressibility condition has to be treated with care (Section 3.1.3).

3.1.2. Navier-stokes lagrangian equations

Assuming that the fluid can be described by a group of discrete particles, the momentum equation applied to such particles will be,

$$\rho_i \frac{D\boldsymbol{v_i}}{Dt} = -\nabla p + \mu\nabla^2\boldsymbol{v} + \rho_i\boldsymbol{g} \tag{3.2}$$

This lagrangian formulation in Equation (3.2) has the advantage of cancelling non-linear terms denoted in the eulerian formulation of the momentum Equations (2.2) also described as the convective term along the three dimensions (x, y, z):

$$v^{\alpha}\frac{\partial v^{\beta}}{\partial v^{\alpha}} \equiv v \cdot \nabla v = \begin{pmatrix} v_x\frac{\partial v_x}{\partial x} + v_y\frac{\partial v_x}{\partial y} + v_z\frac{\partial v_x}{\partial z} \\ v_x\frac{\partial v_y}{\partial x} + v_y\frac{\partial v_y}{\partial y} + v_z\frac{\partial v_y}{\partial z} \\ v_x\frac{\partial v_z}{\partial x} + v_y\frac{\partial v_z}{\partial y} + v_z\frac{\partial v_z}{\partial z} \end{pmatrix} \tag{3.3}$$

Such term involves calculation of partial derivatives that increase the computation cost for the numerical model. This cancellation become especially relevant for real-time implementations using lagrangian formulation.

If, density ρ_i is isolated in Equation (3.2), a clear acceleration notation of particle i can be described as,

$$a_i = -\frac{1}{\rho_i}\nabla \boldsymbol{p} + \frac{\mu}{\rho_i}\nabla^2 \boldsymbol{v} + g \tag{3.4}$$

Equation (3.4) provides a direct form to calculate the particle acceleration, but pressure needs to be found before. The following Section provides the details to solve the pressure variable.

3.1.3. State equations for weakly compressible fluids: pressure-density formulations

For incompressible fluids, the Poisson equation can be derived from continuity equation, leading to: $\Delta p = \boldsymbol{\nabla} \cdot (\rho \boldsymbol{g} - (\boldsymbol{v} \cdot \boldsymbol{\nabla})\boldsymbol{v})$.

However, the integration of the Poisson equation is time costly due to the double integration of the pressure that requires a stable numerical integration scheme where the solution needs to converge with minimal error. That implies a fine spatial discretization requiring a large number of particles with a high computational cost [50].

This is the reason why, instead of solving the Poisson equation, a weakly compressible state equation is widely used in the SPH methodology [51]. It simply relates pressure p and densit ρ with the equation of state of an isothermal ideal gas denoted in Equation (3.5) and used by Müller et *al.* [24].

$$p - p_0 = k_p(\rho - \rho_0) \tag{3.5}$$

ρ_0 is the rest density of the fluid (for water 1000 kg/m^3). p_0 is the reference pressure. $k_p \approx {c_s}^2$, being c_s the speed of sound. After Morris [52], …*["The sound speed must be chosen carefully to ensure both an efficient and accurate solution of a given problem. The value of c must be large enough that the behavior of the corresponding quasi-incompressible fluid is sufficiently close to that of the real fluid, yet it should not be so large as to make the time step prohibitively small. Using a scale analysis, Monaghan [3]*(external reference) *argued that, for density to vary by at most 1%, the Mach number of the flow should be 0.1 or less"]*… That means c_s should be at least 10 times larger than the velocity scale of the flow, $c_s = 10U_0$ or $k_p = 100U_0^2$. As this is a very restricting condition for k_p, Morris in [52] found less restrictive value by considering a balance of forces in the momentum equation, leading to ${c_s}^2 = k_p = U_0^2/\delta$, when pressure forces dominate over viscous or body forces. Where $\delta = \Delta\rho/\rho_0$ is the allowable relative density fluctuation (fraction of compressibility). As $\delta < 3\%$ is an acceptable value according to Morris [52].

Equation (3.5) of state is widely used in closed channel with good results [52]. However, in free surface simulations at low Reynolds number, Tait's equation [53], in Equation (3.6), is more appropriate because it enforces incompressibility [43].

$$p = p_0\left(\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right) \tag{3.6}$$

Where γ is an index constant defined to a value of 7. Notice that in Equation (3.5) $\gamma = 1$. High values of γ make the fluid more incompressible due to the exponent but any error in density computation leads to large inaccuracies in the pressure field. Thus, conditioning the computational time step to be very small. In Sections 4.1 and 4.2 cases, Equation (3.5) of state is applied, while in Section 4.3 case, Equation (3.6) of state is applied.

There are pros and cons when dealing with equivalent solid-fluid coupling models. One important issue is the volume transfer in free surface. SPH method is well known from its

compressibility nature so fluid incompressibility needs to be imposed to the method itself. This need of incompressibility in several free surface applications, like the final third case in this work (Section 4.3), has led to develop different methods to improve incompressibility, such as the weak-compressibility method [43] or the enforced-incompressibility method [52] for SPH.

As mentioned, incompressibility in free surface fluids is directly enforced by solving a time-consuming derivative Poisson equation. This negatively affects real-time applications like the one presented in this work, so, with a real-time performance approach in mind, the Tait's equation is used instead, just as in [53]. In Figure 3.1, two vessels filled up with fluid particles using different pressure equation are shown using the simulator of this work. To the top-left, using the ideal gas equation (compressible fluid), and to the top-right using the Tait equation (weakly-compressible fluid). To the bottom, a number of particles vs. liquid depth graph is displayed. It is clear how far the ideal gas equation implementation is from the ideal incompressibility where filled height h proportionally increases as a linear function of the number of particles within the vessel. Tait's equation provides a much closer performance to the ideal incompressibility.

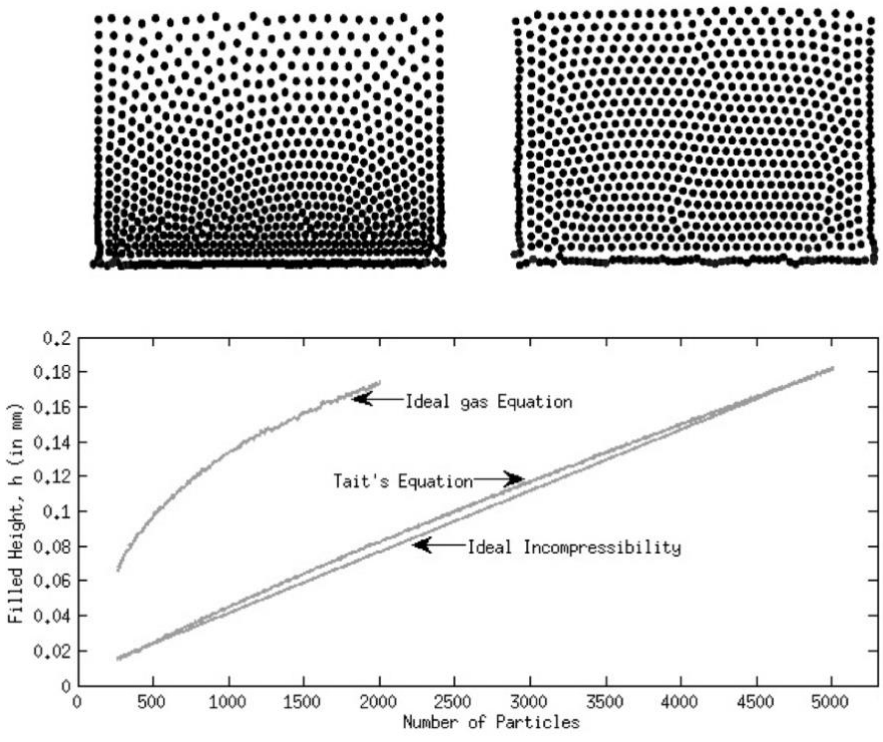


Figure 3.1. (Top) Hydrostatic equilibrium simulation, (left) with ideal gas pressure equation, and (right) with Tait's equation. (Bottom) Graph comparison between (a), (b) and the ideal incompressibility.

The importance of using Tait's equations allow to maintain liquid height quasi-linear with respect to volume height (h). Recall from Equation (4.21) that this coupling model equation depends on $h = A$, so it is fundamental to minimize compressibility.

An issue widely discussed in SPH modeling is the physical inconsistency of particle densities where low pressures (in vortices for instance) generate voids (or agglomerations) in the domain. This particle density inhomogeneities is a non-physical effect in nearly incompressible SPH that has been discussed in the literature. This is the consequence of choosing an inappropriate equation of state or a too small value of k_p. In Figure 3.2, the evidence of this issue is provided in preliminary work of this thesis (Section 4.1 case).

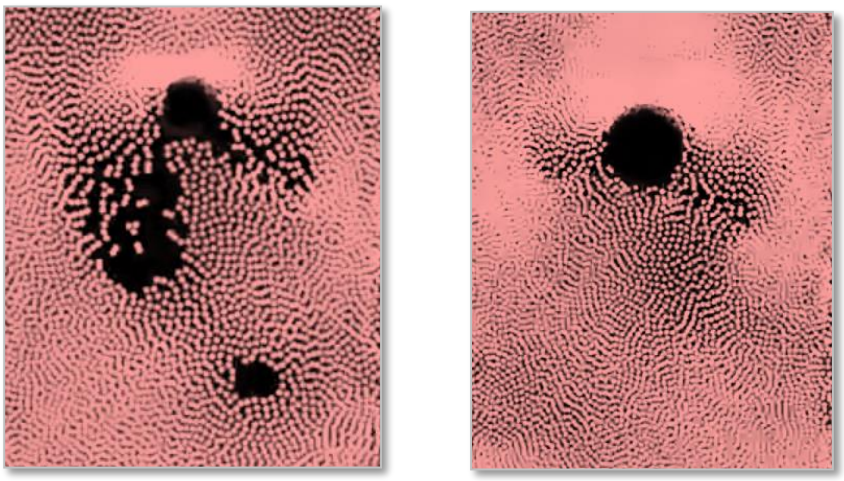


Figure 3.2. (Left) Formation of voids during KVS effect.
(Right) Running simulation with p_0 adjustment.

Morris in [54], followed by Marrone [47], minimizes the formation of voids by adding a background pressure p_0. Such reference pressure p_0 is recommended to be $p_0 = 3\rho_0 U_{max}^2$.

3.2. SPH formulation

SPH was first developed by Lucy (1977), and Gingold and Monaghan (1977) to solve astrophysical problems in 3D spaces. Its possibilities to be applied in a wide range of problems led to multiple improvements to the original method. The essential idea of SPH is the integral

representation of field functions. The concept comes from the integral representation of a function in a volume Ω, $f(\mathbf{r}) = \int_{\Omega} \left(f(\mathrm{r}')\delta(\mathbf{r} - \mathbf{r}', h)\right) \mathrm{dr}'$, where, an integral approximation of f is given by replacing the Dirac delta function by a smoothing function W. Equation (3.7) is this integral representation and the generalized form of the SPH method that describes the force in a given particle at position $\mathbf{r}$. f denotes the function to approximate at position vector $\mathbf{r}$. ρ is the fluid density, and W denotes a weighted function from where the "smoothed" term comes from. W is the result of generalizing a trivial delta function with a characteristic width of two times h, also known as the smoothing length in the SPH formulation. Section 3.2.2 provides an illustrated description of this smoothing length. W also allows to obtain a continuous field of flow properties.

$$f(\mathbf{r}) = \int_{\Omega} \frac{f(\mathrm{r}')}{\rho(\mathrm{r}')} W(\mathbf{r} - \mathbf{r}', h) \mathrm{dr}' \tag{3.7}$$

Equation (3.8) is the discrete version of the SPH interpolation in Equation (3.7) where i ranges over all particles within the smoothing length.

$$f(\mathbf{r}) \approx \sum_i \frac{m_i}{\rho_i} f(\mathbf{r}_i) W(\mathbf{r} - \mathbf{r}_i, h) \tag{3.8}$$

3.2.1. Governing equations: gradient and laplacian interpolants

One of the greatest advantages of the SPH method consists in using the analytical derivative W instead of evaluating the derivatives of the fluid properties, saving calculation time.

For density ρ, required by Equation (3.4), SPH interpolant Equation (3.8) is applied by simply particularizing $f = \rho$ to have,

$$\rho(\mathrm{r}) \approx \sum_i m_i \frac{\rho_i}{\rho_i} W(\mathrm{r} - \mathrm{r}_i, h) \approx \sum_i m_i W(\mathrm{r} - \mathrm{r}_i, h) \tag{3.9}$$

Equation (3.4) requires the first derivative of the pressure, then, if using the definition of the SPH interpolant and deriving f, a gradient interpolant can be expressed as in Equation (3.10).

$$\nabla f(\mathrm{r}) \approx \sum_i \frac{m_i}{\rho_i} f(\mathrm{r}_i) \nabla_i W(\mathrm{r} - \mathrm{r}_i, h) \tag{3.10}$$

Equation (3.10) has a gradient notation where $\nabla \equiv \partial / \partial \mathrm{r}$. From the first term of the right hand of Equation (3.4), the pressure term, $-\frac{1}{\boldsymbol{\rho_i}} \nabla \boldsymbol{p}$, is the gradient term that is developed using SPH gradient interpolant in Equation (3.10). The mathematical details on how, making $f = p$, turns Equation (3.10) into Equation (3.11) is fully developed in Appendix A.1.

$$\frac{\nabla p(\mathrm{r})}{\rho_i} \approx \sum_i m_i \left[\frac{p(\mathrm{r}_i)}{\rho(\mathrm{r}_i)^2} + \frac{p(\mathrm{r})}{\rho(\mathrm{r})^2} \right] \nabla_i W(\mathrm{r} - \mathrm{r}_i, h) \tag{3.11}$$

For the laplacian operator, needed in Equation (3.4) for the middle term of the right hand side, the viscous stress term, $\frac{\mu}{\boldsymbol{\rho_i}} \nabla^2 \boldsymbol{v}$, is derived from SPH formulation in [24], resulting in Equation (3.12),

$$\nabla^2 f(\mathrm{r}) \approx \sum_i \frac{m_i}{\rho_i} f(\mathrm{r}_i) \nabla_i{}^2 W(\mathrm{r} - \mathrm{r}_i, h) \tag{3.12}$$

This last equation is applied to the laplacian and developed, such as it can be found in Appendix A.2, yielding to Equation (3.13),

$$\nabla^2 v(\mathrm{r}) \approx \sum_i m_i \left[\frac{v(\mathrm{r}_i) - v(\mathrm{r})}{\rho} \right] \nabla_i{}^2 W(\mathrm{r} - \mathrm{r}_i, h) \tag{3.13}$$

In summary, Equations (3.9), (3.11), and (3.13) complete the three terms needed to obtain the acceleration of particle i from Equation (3.4).

3.2.2. The smoothed function

The SPH method obtains smoothed approximations of forces, that is, it estimates by interpolation a particle force from the sum of forces of the nearest-length surrounding particles (by two times the smoothing length h), see Figure 3.3. Typically, W from Equation (3.8) is a distributive and smoothed function, also known as kernel in SPH terminology, where the nearest points have more influence in the interpolated function than the most remote ones.

So, basically this is a sampling function of spatial neighborhood where particles get continuous flow properties based in the surrounding set of particles. Different W smoothed functions are described in literature and have the form of the gaussian, cubic spline, quintic spline and other types of distribution functions with compact support. The kernel must comply with the general criteria of normalization of a δ-function, this is:

$$\int_{\Omega} \mathrm{W}(\mathrm{r}-\mathrm{r_i},\mathrm{h})\mathrm{dr_i} = 1 \tag{3.14}$$

In order to have the area under the curve equaled to one for the 2D case or the sphere volume equaled to one for the 3D case as described in Equation (3.14).

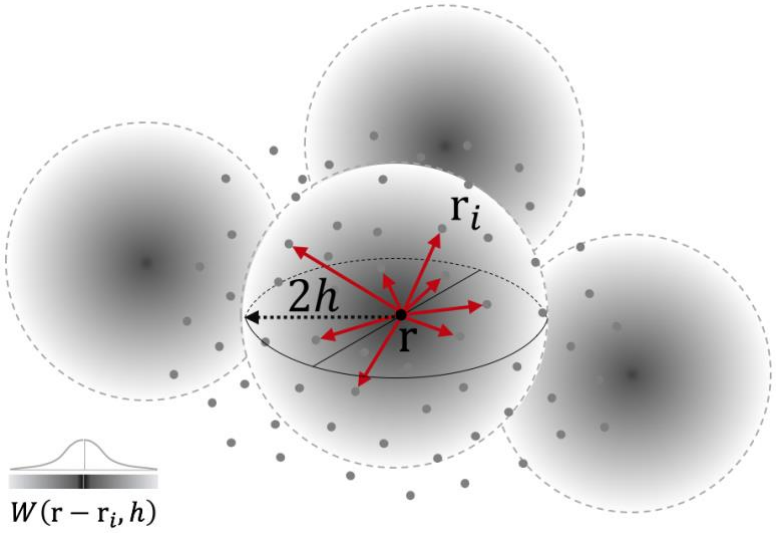


Figure 3.3. Example of the used kernel distribution function W.

In terms of computational cost, a gaussian-formed kernel like $W(\mathrm{r}-\mathrm{r_i},h) = \frac{1}{h^3\pi^{3/2}} e^{-((\mathbf{r}-\mathbf{r_i})/h)^2}$ of Equation (3.11) has a complexity of $O(NN_h) \approx O(N^2)$, being N the number of particles during simulation and N_h the number of particles within the range of h. The issue with a single gaussian kernel is that it evaluates along $\mathbf{r}$ giving a compact function when $\mathbf{r}$ approaches to infinity, far beyond h x 2 range. h is the radious of the supporting kernel, and $2h$ is the range where kernel function should act through. So, the force contribution of the particle that are far beyond $2h$ are considered negligible. For this reason, in literature many kernel functions have been proposed, but particularly, functions with limiting values.

Typically, the cubic spline function is recommended in literature [50] because it provides limited response, compact support in the range of h, and handy computational cost. Cubic spline

fits well for incompressible fluids as mentioned in [55] and supported by experimental data in [56], despite of the non-smoothed form of its laplacian required for the viscosity, see Figure 3.4.

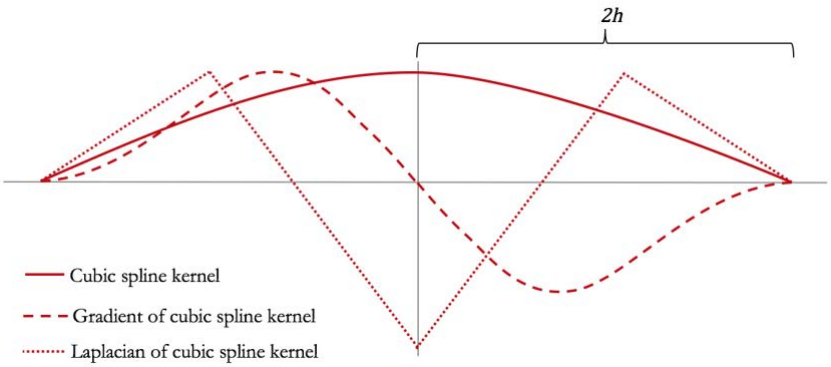


Figure 3.4. 2D cubic spline function, its gradient and laplacian.

$$W(\mathrm{r}-\mathrm{r}_i,h)=\alpha_D\begin{cases}1-\frac{3}{2}q^2+\frac{3}{4}q^3 & 0\le q\le 1\\ \frac{1}{4}(2-q)^3 & 1\le q\le 2\\ 0 & q\ge 2\end{cases} \tag{3.15}$$

Equation (3.15) defines the cubic spline function that is used in the implementations of this work, where $q=\frac{|\mathrm{r}-\mathrm{r}_i|}{h}$; $\mathrm{r}-\mathrm{r}_i$ is the distance between particle of interest $\mathbf{r}$ and particle $\mathbf{r}_i$, and α_D is a dimensional factor which is $\frac{1}{\pi h^3}$ for the 3D case.

This cubic spline kernel is recommended based on the computer processing performance of a state-of-the-art GPU device. It is expected to have more complex kernels implementations where no computational cost is compromised in other real-time applications.

3.2.3. Spatial resolution and kernel size

In this work, the kernel size is selected based on the domain dimensions and the number of particles to simulate. For the same number of particles, the larger the kernel size is, more computational cost is involved, but, a small kernel size would not smooth well enough the fluid properties if a minimum number of particles is not included within the kernel. That is why, a rule of thumb has gain popularity [14, 57] in the SPH method to select the kernel size (smoothing length) based on the spatial dimensions and the maximum number of particles. For this, the input constant values are a selected normalized domain volume V_f (width * height * depth) and a

maximum number of particles N_{max}. The normalization is required by OpenGL data types. Having such inputs, the rule [14, 57] uses the following relations expressed in Equations (3.16), (3.17) and (3.18) to find the smoothing length.

$$V_p = V_f / N_{max} \tag{3.16}$$

$$d_{rest} = 0.87 \sqrt[dim]{V_p} \tag{3.17}$$

The rest distance d_{rest} in the distance between particles in hydrostatic equilibrium, where dim is the number of the space dimension of the simulator, and 0.87 is a *"magic"* number found experimentally and recommended in [57]. It has been demonstrated by Moulinec at *el.* [25] that the number of neighbor particles to include within the kernel can be limited to 32, (Liu in [27] propose 57 particles) regardless to the total number of particles in the domain. This number provides good fluid effect but more importantly it improves execution time by limiting the number of particles to compute. It has been shown that more particles within the kernel do not improve accuracy. Following this rule, the smoothing distance (h) is set to,

$$h = 2\, d_{rest} \tag{3.18}$$

An important issue is the size of the smoothing length h and how this affects computation cost when set constant throughout the entire simulation. Since fluid density may vary along the domain, blank voxels may occur. In that case, if h is too small, no forces act in a particle computation. On the contrary, if h is too large, local variation of fluid properties (*e.g.* vortexes) are smoothed away. This is why, some authors change h dynamically depending on local densities, and others, instead of using spherical kernels, use ellipsoidal kernels to handle force effect on a particular direction. For incompressible fluids, there is no density variation to handle. In consequence, h will be constant in this work.

3.3. Boundary conditions

Within the simulated domain, boundary conditions need to be defined. Those boundaries are the space limits of the simulation domain. Such limits include entry and exit of fluid, walls, and other possible types.

In Section 3.3.1, the spatial domain including the two configurations developed in Chapter 4. Section 3.3.2 provides the details on the type of domain walls the SPH method frequently implements and the one used in this work. Section 3.3.3 provides initial setup of the fluid, and how particles enter and exit the domain.

3.3.1. Spatial domain

The spatial domain consists in a parallelepiped limited by 6 faces. Depending on the simulation type, closed channel or free surface, the volume can be arranged in custom dimensions. For the former, illustrated in left diagram of Figure 3.5, domain is bounded by 4 lateral walls and one inlet and one outlet on the flow direction. For the latter, the domain faces are only sized to accommodate the solid objects within. Particles only interact with walls of the objects that become boundary conditions too.

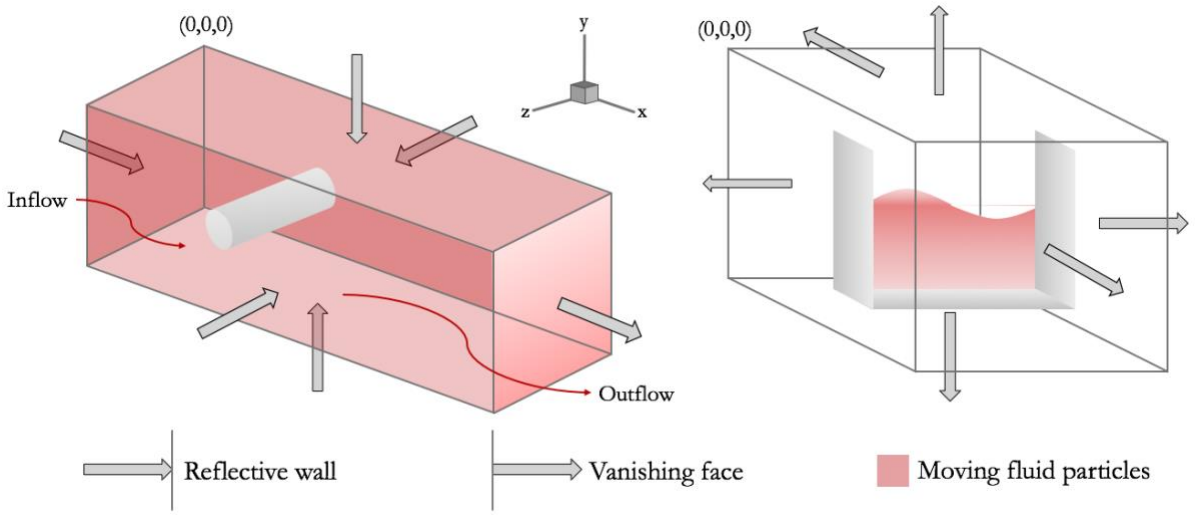


Figure 3.5. (Left) The infilled domain for closed channel simulations. (Right) Cleared domain for free surface simulations.

Any particle that crosses the domain boundary vanishes. Such condition is illustrated in Figure 3.5 as vanishing face. In the following Section, the details on these types of walls are described.

In the simulations of this work, triangles are the geometry primitives of any solids, the domain boundaries and solid bodies. The selection of triangles as primitives allows to compute collision forces of fluid particles with a very straightforward query of the triangle normal vector and the particle trajectory vector. This method, regardless of its pure repulsive nature, is fast and convenient for a real-time computation in robotic kinematic control.

3.3.2. Walls (ghost particles method and reflective walls)

As presented in Section 3.2.2, the kernel is computed to estimate the total net force of each particle. However, the solid boundaries of solid bodies and the domain truncate the kernel in locations near boundary limits. This is a common issue in SPH named as particle inconsistency. Some, like Chen at *al.* [58] came up with a partial boundary correction using Taylor expansion series.

Others [59] treat boundary conditions with an extended wall of fixed particles (or ghost particles) providing a damped nature to the boundary. And others [60] use a normalization formulation of a constant pressure to take care of boundaries. Neither of them solve the issue by a physical approach, but allow to minimize the issue to some extend depending on their SPH application. In our case, this particle inconsistency is not always important because of the application dimensions. However some of the boundary treatments are described in the following paragraphs.

The two most common type of wall boundaries for collision schemes are the ghost particles and the reflective walls:

- Ghost particles: This type of boundary is one of the most common found in literature [61]. This scheme efficiently solves the spurious numerical artifact occurring at the solid-fluid boundary. The model samples solids into fixed particles and adds ghost particles into the surface of those solids. The ghost particles provide a kernel integrity so the computation of the SPH interpolation equations happens with a complete number of particles each time step. These particles are always quiescent. The main disadvantage for

real-time simulation is the extra processing of surface sampling of solids. To the left in Figure 3.6, the ghost particles are illustrated in a solid boundary.

- Reflective walls: In this type of boundary, the solid primitives are triangles so the collision scheme is based on triangle-particle collisions. This is the collision scheme implemented in this work, and its first advantage is that no resampling of solids is required as in the ghost boundaries. The reflective particles require a scheme of collision detection and later a collision reaction. The detection is based on the intersection of the triangle normal and the particle position vector just as in [16]. Details on the implementation of the detection is provided later in Section 3.5.3. After detection, a reaction is implemented by reflecting the collided particle with respect to its original trajectory vector. So, the collided particle simply mirrors its trajectory with respect to the triangle's normal. A simple but time efficient perfect elastic reflection model. To the right in Figure 3.6, the reflective particles are illustrated in a solid boundary.

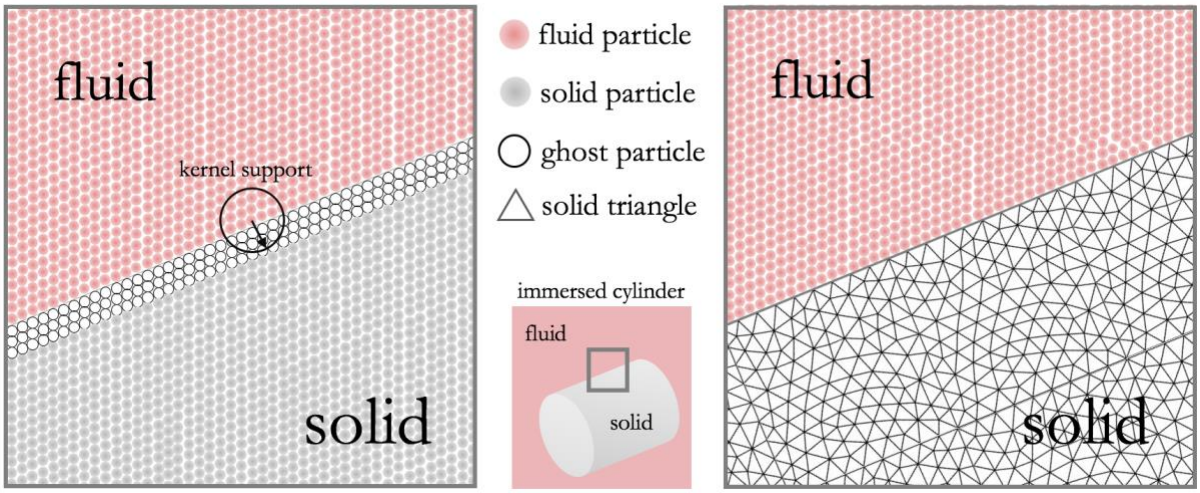


Figure 3.6. (Left) Boundary conditions based on ghost particles of the immersed cylinder diagram. (Right) Boundary conditions based on reflective particles of the immersed cylinder diagram.

A reflective wall collision scheme disregards other important physical properties of a physical collision. Specially the non-slip condition which defines a zero velocity at the fluid-solid boundary in a viscous fluid. The main disadvantage is that repulsive walls create artificial barriers where particles cannot touch solids. Other physical properties of fluids are available at the fluid-solid boundary that can be tackled in the implementation. However, for real-time simulations the computational cost is privileged, so these reflective walls become an interesting option.

3.3.3. Velocity inlet and outlet

The two cases presented in the previous section are the two setups for the tested cases that will be presented in Chapter 4. For the free surface case, no expected velocity criteria for fluid particle entry and exit is required. This simulation initiates by arbitrarily locating particles inside the vessel. The particles reorganize themselves when the simulation starts due to pressure force balance, until reaching rest condition. This moment is considered the initial condition of a free surface simulation.

For closed channel simulations, the domain is initially empty of particles, so, a filling phase is required to fill up the domain. This phase is implemented as follows:

Particles are injected with constant velocity at the inlet boundary. After fluid particles travel along the domain, delimited by repulsive walls, they find the distant vanishing face. Opposite to a reflective wall, a vanishing face allows particles to leave the domain and clear them from the system. The steady flow condition is reached when the number of particles in the input equals the number of particles at the output. To retain the number of particles constant, the entry of particles should compensate the exit. For such effect, in the 2D case, a line of particles is generated at a constant rate of appearance. For the 3D case, a y-z plane of particles is generated instead. See Figure 3.7 for reference of the 2D case. Such line or plane is placed right at $x = 0$ and initiates its travel with a constant x-velocity.

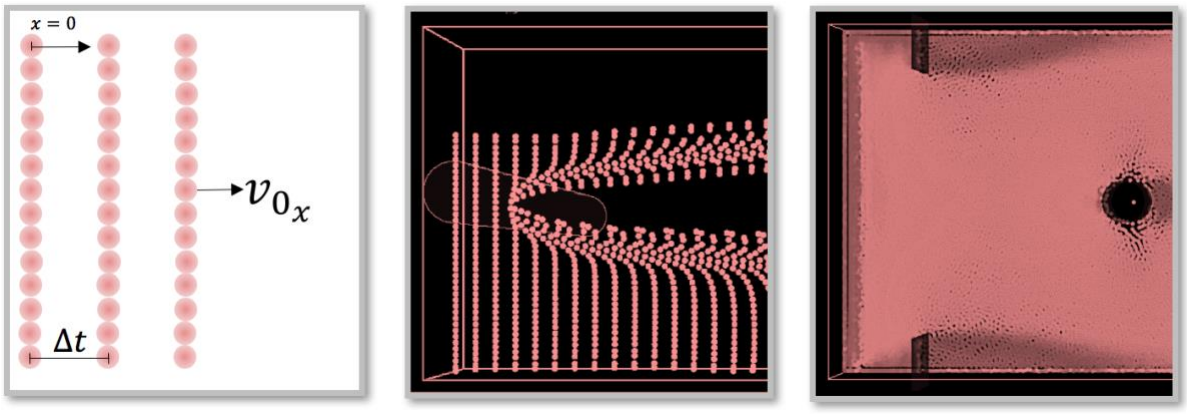


Figure 3.7. (Left) the upstream line of fluid particles. (Center) The upstream line during running simulation grossly exhibits the issue of artificial entry distribution. (Right) The corrected inflow of particles using stimuli plates.

The main issue in the previous described inflow setup is that the artificial distribution of injected particles will not converge quickly enough to a uniform density field. Meantime the uniform density is reached, a long travel of fluid particles may take place increasing the domain size, and the computational cost significantly.

To remediate this issue and reach uniform density fast enough. An inlet of particles is stimulated adding disorder in the initial region. Such region includes a pair of stimuli plates (see right side of Figure 3.7) that promote the particle interaction to create an uniform field. After particles pass through such plates, the fluid forces start acting sooner and the filling process of the domain achieve steady flow condition faster. In Section 4.1.2, practical implementations include the domain dimensioning including the stimuli plates distance that defines the initial region in Section 4.1 and 4.2 cases.

3.3.4. Conclusions and recommendations

It is important that the spatial domain and boundary conditions can be implemented with no compromised computational cost for real-time simulations. Reflective walls are efficient in such effect, allowing the processing load to be related more to fluid force computation than to boundary issues related to the kernel integrity.

The velocity inlet of particles is of great importance when closed channel simulation requires stable and physical flow conditions. However, it has been noticed that the only available approaches of flow inlet are unphysical and have different constrains that drive away the usable domain space from the entry and/or have long waiting delays for filling the domain with particles. The recommendation is, if no real-time is required, to avoid modelling the fluid inlet and setup the domain with an initial fluid distribution. But, as in this work requirement, when dealing with real-time processing an inlet model is required and the domain space needs to be characterized in two regions. The inlet region where flow have not reach uniform density and no simulation should carry on, and the useful region where flow conditions are met and the simulation can run as intended. The fluid outlet is of less concern for most of the closed channel simulations though, so no issues should be found.

3.4. Time integration

A real-time interactive simulator based on SPH requires a temporal domain to fit user interaction events. Section 3.4.1 provides the temporal domain to initiate and setup the simulation framework where force integration timing and user interaction work seamlessly for interaction purposes.

As part of a SPH method a time integration of forces scheme driven by a time step is required. Section 3.4.2 provides the integration scheme selected to work in this work. The time step criteria is also discussed in Section 3.4.3.

3.4.1. Temporal domain and initial conditions

As it will be described in Section 3.6.1, the simulator is a real-time framework that must control execution time. The user events, the window handler and rendering is operated by the OpenGL-GLUT library. Using this library, a timer callback is configured to run every update loop. In Figure 3.8, an update loop defines the period of time to run every task required to render a new system update, including solid and fluid calculations. The purpose of this update loop is to process and pass all simulated data, including SPH, to the renderer library which will update data on the screen with a graphical frame rate.

During the update loop, a time step δt is required as part of the integration scheme. Actually, the integration time can match the update loop considering that the OpenGL-GLUT task would not cause timing conflict with the update loop. In next Section, the selected integration scheme is provided. To avoid compromising integration stability, δt has to be small. On the other hand, δt cannot be too small to guarantee real-time performance. The smallest value of δt is 2 milliseconds as the basis to all simulated cases. Such value allows to run multiple calls of the update loop before sending to the screen by the OpenGL-GLUT rendering. In Section 3.6.1, a detailed description of the involved libraries will be provided to explain the issues

of this real-time and interactive simulator. As well as in Appendix B.1, the details on OpenGL-GLUT usage are presented in regard to the rendering process that improves the data transfer between system memory and video memory to shorten rendering times.

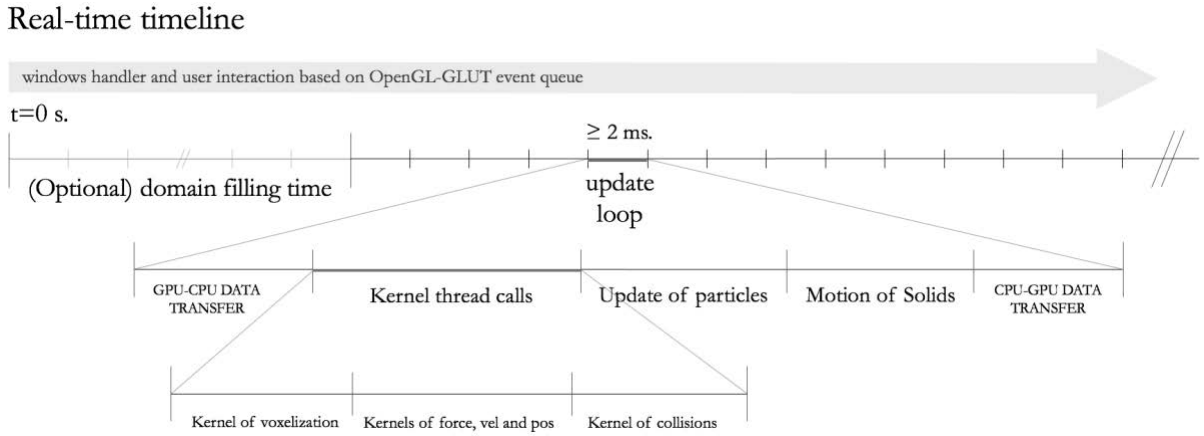


Figure 3.8. Timeline of the real-time simulation.

Previous approaches, described in Section 3.3.3, use an empty domain with fixed solids that is filled with moving particles initialized with constant velocity at the beginning of the simulation to reach the desired flow condition. Such approaches require a period of time for filling in the domain. Notice the optional domain filling time in Figure 3.8. However, for the experiment meant to be simulated in this work (Sections 4.1 case), it can be conceived the opposite simulation setup with a fixed fluid and movable solids. This allows to skip the inlet and outlet conditions, and particles are just placed within the domain with any distribution. This also improves the waiting time required to reach initial flow conditions.

3.4.2. Force integration

Equation (3.4) is a summation of the involved forces for particle i that needs to be transformed into a new position for that particle each time step. This acceleration represented as a force vector is processed by numerical integration as a velocity vector following by a position vector. This integration is done by using the second order leapfrog explicit integration method. Its formulation is in Equation (3.19) and (3.20) using the terms for acceleration a, velocity v and position r, all of these variables with respect to time.

$$\mathrm{r}_{n+1} = \mathrm{r}_n + \delta t \, v_{n+1/2} \tag{3.19}$$

$$v_{n+3/2} = v_{n+1/2} + \delta t \, a_{n+1} \tag{3.20}$$

Leapfrog method has been widely used in discrete motion by SPH implementations [62]. Its main benefit is that it processes one step integration in two half steps, one half step for position and one half step for velocity. Of course, the step count n is an integer so (3.19) and (3.20) can be rewrite as,

$$\mathrm{r}_{n+1} = \mathrm{r}_n + v_n \delta t + \frac{1}{2} a_n \delta t^2 \tag{3.21}$$

$$v_{n+1} = v_n + \frac{1}{2} (a_n + a_{n+1}) \delta t \tag{3.22}$$

Being δt a constant time step through all the simulation process, as defined earlier at 2 milliseconds. Another important benefit of this integration method is the time reversal invariant advantage. This is, to keep Newton's equations of motion invariant, leapfrog method supposes the position and velocity trajectories reversible due to the symmetry of its formulation. See Equation (3.21) and (3.22), and start with $r_0 = X$ and $v_0 = V$ and determine r_1 and v_1. Then start the time reversed trajectory with $r_0^x = r_1$ and $v_0^x = -v_1$. The steps of Equations (3.21) and (3.22) in the reverse trajectory correspond precisely to the steps in the forward trajectory since it will find that $r_1^x = X(= x_0)$ and $v_1^x = -V(= -v_0)$.

3.4.3. Time step selection criteria

Fluid data update is computed based on the integration time step δt. This time step, used in this work for force integration, is constant throughout the entire simulation and defines the limit of the update loop period of time. Other integration methods may use a variable time step schemes to increase performance using a selective method for active and inactive particles [62]. Others, to optimize the real-time performance with a constant time step, use adaptive schemes that adjust the time step based on the data motion within the scene [63].

For a constant time step, it is a good practice to base this definition on the Courant-Friedrich-Lewy (CFL) condition [64], which, in plain words, defines the smoothing length as the maximum length a particle can travel in one time step. Other criteria can be used like in [52], but for simplicity the CFL condition remains dominant. In Equation (3.23), the time step associated to CFL relates with the smoothing length h and the characteristic speed c_s. An index λ between 0.25 and 1 is been used to constrain CFL condition in literature [50, 52]. Our choice of λ is 1 giving the maximum time step value coverage by CFL.

$$\delta t \leq \lambda \frac{h}{c_s} \tag{3.23}$$

The characteristic speed c_s is the maximum velocity a particle can achieve during simulation. This value, usually referred in SPH literature as the sound velocity in fluid media, is approximated to the pressure constant from the Equation of state (3.5) as follows: $c_s = \sqrt{k_p}$. Parametrization values selected in this work may be found different for the three simulation cases in Chapter 4.

3.5. Parallel computing in SPH

OpenCL is a parallel computing software library for the GPU device, developed by Kronos group [65] as an open source and free of use copyright framework. Until now, OpenCL has been updated and maintained continuously, giving support to both GPU and CPU parallel processing code. This dual benefit is known in parallel computing as heterogeneous computing, where parallel coding does not discriminate the processing device, being this a CPU, a GPU or an arrangement of both. This is the main justification of using OpenCL library in this work as a parallel processing library so further work might be implemented regardless of the processing device of interest.

As mentioned earlier in this thesis work, fluid-robot simulation is pretended to work in real-time for feedback redesign. So, it is implied that real-time is defined as how the user interact with the system, that is, that during the simulation time, the user may dynamically conduct some events that could alter the fluid-robot model behavior but observing real-time responses along the interaction. The interaction might also be implemented as a redesign loop where the

experiment can be designed during the simulation time. As described in Figure 1.1, the user interaction may design behaviors of fluid dynamics and/or robotic kinematics on the fly.

The SPH method imposes a numerical algorithm consisting in processing the three terms of Equation (3.4) to obtain the particle acceleration, and from this, by numerical integration, the speed and position of a given particle in the system. This process is repeated for all fluid particles and for each time step in the simulation. This process has quadratic complexity, and the number of operations in the smoothed function grows exponentially upon the number of particles.

These processing requirements make the SPH method a high computational cost implementation for a real-time application. So, to enhance this performance, many optimization techniques have been developed [14, 24, 25]. From these techniques, the basic concepts have been selected to define the programming code structures that were implemented with parallel processing using OpenCL similarly to what is found in [57]. These OpenCL structures are called GPU Kernels in terms of GPU processing. Do not confuse with the term SPH kernel, previously used as the smoothed function W in SPH. To differentiate them and to keep reference consistency in this work, kernel refers to the smoothed function and Kernel refers to GPU structures.

Every time step, a computer thread runs the Kernels for each particle. In next sections, the four GPU Kernels that were implemented in this work are described; the Kernel of voxelization (Section 3.5.1), the Kernel of force integration (Section 3.5.2), and the Kernel of collisions (Section 3.5.3).

The benefit of working with OpenCL Kernels is that those three parallel calls are processed at minimum execution time within the GPU, so, particle positions are processed much faster and CPU load is reduced allowing to process the user interaction inputs task within the main system thread. This Kernel sequence is the minimum sequence needed for a particle position to be computed, other Kernels could be introduced to process custom simulations of fluids. Interesting task of how to parallelize the SPH method has been widely described by [14, 24, 25], so, no further detail will be address on how the implementation of SPH is conducted. Instead, it is focused in providing the simulation workflow (Section 3.6.1) and the real-time processing achieved by the implementation with the selected programming libraries.

3.5.1. Kernel of voxelization

The task of this Kernel is to compute the blocks of data containing the nearest neighbor particles within each kernel so this block of neighbor data can pass to the following Kernels. This task is done by dividing the simulation domain into fictitious cubic sections (voxels) for 3D simulations or squared sections for 2D of the size of two times the length of interaction h of our smoothed function W in Equation (3.7). This is also known as spatial uniform grid in SPH terms. Each particle interacts only with the particles within its voxel and from the next adjacent voxels for a group of eight voxels for the 3D domain and a group of four voxels for the 2D domain. The voxelization only happens where particles exist. See Figure 3.9 for reference.

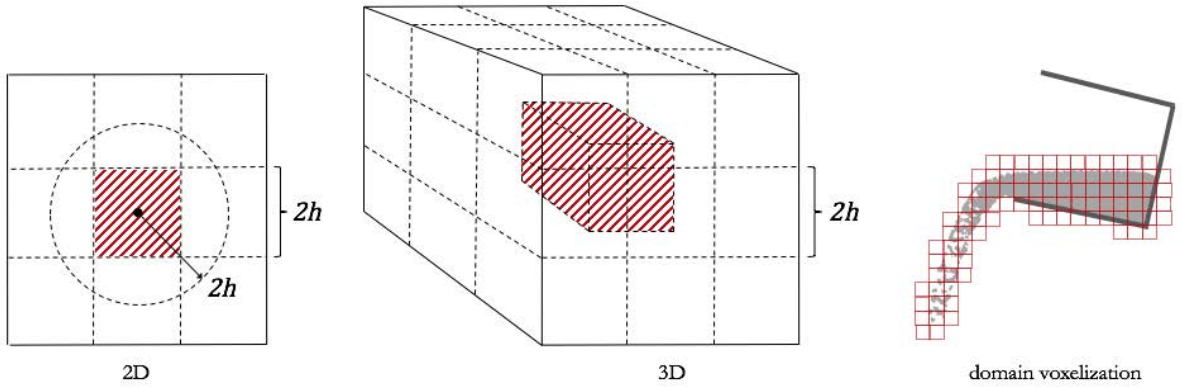


Figure 3.9. (Left) the group of voxels for 2D domain.
(Center) the group of voxels for 3D domain. (Right) 2D voxelization of the fluid domain.

When a particle is located at its voxel center, the range of interaction of W is contained in that voxel and in the part of the other surrounding voxels. But this is not always the case, in many cases the particle will be located far from its voxel center, in such way that W covers two voxels in full; and this is the reason of using voxels grouped by 3^d voxels; being d the domain dimensions). See left diagram in Figure 3.9.

Particle data needs to be initialized with position, velocity, and acceleration for each particle in the system. Depending on the type of simulation, one can add initial velocity or acceleration to the entry particles. In this work, the entry and exit of particles and their initialization are addressed in Section 4.1.3 where the simulation case involves the study of inflow and outflow fluid. In C++ code, it is common to use one four float <**float4**> buffer to store all particles

position. Another identical buffer for velocity and other for acceleration. The four float particle data stores the three float coordinates of the euclidian $[x, y, z]$ space and one float $[w]$ for the voxel ID. This data type is also native for OpenCL 1.0.

To identify particles within each voxel a hash table requires to be computed. A hash table maps a 2D or 3D matrix position elements into a 1D no repeated list value vector. For such task, a different hash functions can be used for the 2D case and for the 3D case. Equation (3.25) shows the hash function for 2D, where $[x, y]$ are the coordinates of each particle, $voxel\_spacing$ is the h of W, and, $voxel\_delta$ in Equation (3.24) is the number of voxels in the x-dimension.

$$voxel\_delta = (domain\_max - domain\_min)/voxel\_spacing \tag{3.24}$$

$$hash\_2d(p) = floor(x/voxel\_spacing) + (floor(y/voxel\_spacing) * voxel\_delta) \tag{3.25}$$

Equation (3.27) shows the hash function $hash\_3d$ for the 3D domain used in this work. $pos(p)$ required by Equation (3.26) is the position coordinates of particle p in $[x, y, z]$ dimensions, $domain\_min$ is minimum coordinates of the domain, usually set to zeros, $voxel\_delta$ is the dimensions of the voxel, $voxel\_id$ defines the index of each voxel in the 3D space, dim_vox is the number of cells in each dimension.

$$voxel\_id(p) = (pos(p) - domain\_min) * voxel\_delta \tag{3.26}$$

$$hash\_3d(p) = (((voxel\_id.z * dim\_vox.y) + voxel\_id.y) * dim\_vox.x) + voxel\_id.x \tag{3.27}$$

Recall that the domain is normalized and these two hash functions are not. So, it is mandatory for these functions to pass particle position and domain dimensions in proper integer values. In SPH literature, this hashing process is indicated as uniform grid generation. Here, a linear voxel index is used as the hash indexing, however, to avoid flow inconsistencies, other voxel indexing can be made like the z-order indexing [66] first described by Morton G. in [67].

Now that the hash value for each particle has been calculated, a sorting process requires to order the hashed particles according to the voxel where they belong. Actually, this process has become a performance differentiating feature in SPH implementations. Some SPH implementations use atomic operations and others a sorting algorithm for ordering particles [68] for the sake of the speed. Since sorting particles is one bottleneck task when dealing with a real-time process in SPH, a parallel implementation of sorting is always desired. Radix sorting has becoming a popular SPH sorting algorithm due to its fast parallel implementation [69] and also

because of the available GPU implementations from GPU manufactures [70, 71]. Speed efficiency is achieved in N particles $>10^4$ for GPU-based radix sort implementations [72]. The counting sorting algorithm [73] is even better in performance than radix sort for larger numbers of particles $>10^8$. However, the free surface applications where this work is meant to contribute requires a number greater than 5k particles. Thus, bitonic or radix sorting are within the application range [74] [pp. 95-96]. A GPU based bitonic sorting algorithm has been used in this work right from an OpenCL implementation in [75]. This implementation relies on a bitonic merge sorting algorithm with a total complexity of $O[\mathrm{N} \cdot log^2(\mathrm{N}) + log(\mathrm{N})]$ which is lower than the brute force sorting case of $O[\mathrm{N}^2]$ and that has been improved by parallel GPU processing by avoiding data copy operations [76].

By sorting the hashed particles, a sorted list of all particles in each voxel is now accessible by their spatial position. Then, a final list is built with the start of all voxels in the sorted list. The way to accomplish this is by comparing the voxel id of the current particle against to the voxel id of the previous particle in the sorted list. If the voxel index changes, the voxel start value is stored in the particle index of the final list. In [68], Figure 3, a 2D domain example provides an illustration of the algorithm. This algorithm, including the hashing, the sorting and the final listing, is required to be computed each time step that the particle position is updated.

3.5.2. Kernel of force integration

At this point a new re-indexed particles list is ready to be relocated accordingly to the fluid force model described by the momentum Equation (3.4). As described in Section 3.4.2, the selected integration method used is the leapfrog. This has the benefit of time reversal that is especially useful for back and forth running simulations, and, also for live recording interactive simulations. In this work, it is desirable to have a reversible kinematic simulation for real-time experiment redesign. Another benefit of using leapfrog method is the conservation of angular momentum and area preservation (symplectic). Being such important benefits in terms of physics robustness, further details can be found in [77].

This Kernel is very straightforward to implement. It basically consists in implementing first Equation (3.22) and, from the resulting velocity, Equation (3.21) to find the new position of each

particle. From Equation (3.4), notice the gravity vector force. This force is included depending on the simulation case. For closed channel simulations, this is not included, but for free surface cases is demanded and it is typically the most dominant force of all when dealing with fluid in the laminar regime as in this work simulations.

3.5.3. Kernel of collisions

The idea of having a system where fluid interact with robot mechanisms, requires to apply a motion handling scheme where the fluid (a particle based system), get in physical contact with two elements; one, the static surroundings or domain, and two, with the moving robot agent. These two elements are built with the same solid primitive: the triangle. Static triangles for the domain and moving tringles for the robotic agent tested in this work.

Robot mechanism and domain borders are solid parts that are build out of triangles for simplification in this work. So, all what collide are triangles, defined by three vertices, against particles. Particle-particle *collision* is driven by the SPH forces (density, pressure, viscosity), triangle-particle collision is handle in this Kernel of collisions, and triangle-triangle collision (collisions between solid bodies) is not in the interest of this work. Remember that particles have an implicit mass (in density formulation) which helps to compute SPH method, however the geometry of particles is null, in the sense that this is only defined by a point in the 3D space.

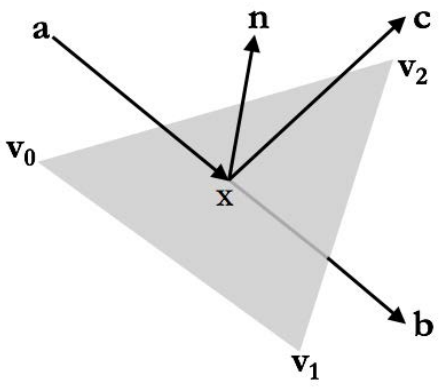


Figure 3.10. Particle-triangle collision detection scheme.

Figure 3.10 defines the trajectory for a particle vector **x** that travels from position **a** to position **b**. If a solid triangle happens to be across this trajectory, a collision test against the triangle with vertices v_0, v_1, and v_2 is performed considering the condition in Equation (3.28), where **n** is the normal vector of the triangle. If such condition is true then a collided particle is identified.

$$(\mathrm{x} - \mathrm{v}_0) \cdot \left((\mathrm{v}_2 - \mathrm{v}_0) \times \mathrm{n}\right) > 0 \tag{3.28}$$

Although the number of triangles that represent solid objects and all six domain walls are less in proportion to the number of particles, one of the issues here is to test all triangles for each particle within the Kernel during each time step, making this very computationally expensive. For this reason, this Kernel only tests collisions on each time step with not all the triangles in the domain, but only with those included in the set of voxels of the particle of interest (27 voxels for the 3D domain). In this fashion, the execution time is optimized.

In our case, the reaction to a collision is a trajectory change (inversion or reflection) of the particle where acceleration and velocity magnitudes of the colliding particle never gets affected but only by its trajectory angle. In such way that, when a collided particle is detected, this particle will change its trajectory with opposite angle with respect to the collision angle, estimated with the normal vector like in Figure 3.10 the final vector **c**. The weighted force or acceleration of particle i in Equation (3.4) is then updated in sign by the collision algorithm.

The datatype for triangles in C++ is not native, then, a triangle class is added to the system. In this implementation, the Triangle class is the structure containing the triangle data which includes an array of three **float4** variables to store the three vertices of the triangle, and one **float4** variable to store the triangle normal vector. All vectors in the simulation scene are referenced from the simulation domain origin that is in the left-top-front location.

There are many issues involved in the effect of physical collisions, and many primitive elements to assume in a complex simulation environment, like those described in [16]. For instance, the elastic and/or inelastic energy and the friction, and therefore, further work may include these ones for different application cases. There is another issue to take care in collision: the time of collision. While collisions of particles against fixed triangles required certain time step

to compute, the collisions of particles against moving triangles required a smaller time step to avoid particles to penetrate triangles (tunneling effect).

Another issue involved in collision is the sliding effect. In a pure dissipative collision approach particles sliding on a tilted solid wall is impossible to have, while it is a valid from the physical perspective. This could be handle by adding a contact model of partial friction where particles can rest in solid surfaces by making zero their velocity when this velocity approaches to a minimum threshold.

3.6. Final implementations

The software implementation of the simulator engine needs to take care of two parts. One, related to the logic process of the simulator including the usage of the code, libraries and frameworks. This involves the parallel and sequential blocks that run in synchrony. The other part to take care of is the real-time processing, in other words, the time rate needed to preserve the same overall time of the simulation with the real physics experiment time. In control kinematics, the output usually is expressed as a discrete time response to be used by any robotic actuator; that is why the time rate specification is important.

The idea of breaking apart the SPH fundamentals in the previous Sections of this Chapter 3 is to provide the mathematical background so, when the following implementation Section was described, a clear understanding of the simulator parts can be obtained. The main objective of this work is not the simulator implementation itself, but the robot-fluid coupling modelling and its implementation. Nonetheless, the simulator requires to be properly described and the following two subsections helps for such goal.

3.6.1. Simulator structure

The simulator structure is divided in two parts; from Figure 3.11, to the left, the update loop controlled by CPU processing running each simulation update cycle, and to the right, the parallel

tasks or Kernels processed by the GPU. In each function of the CPU a sequential task calls the GPU Kernels passing the kinematic data of particles and triangles for parallel processing. After initializing the *N*-size **float4** particle arrays (position, velocity and force), a nearest neighbor search algorithm is computed using the Kernel of voxelization described in Section 3.5.1. Being *N* the number of particles. OpenCL, as the parallel processing library, computes in threads of execution each particle property. Advanced parallel programming allows to optimize the local and global GPU memory transfers for multithreaded tasks [1].

The following block in Figure 3.11 calculates fluid density. This density needs to be precomputed first to allow the following function block to compute each individual particle force of pressure and viscosity, and accumulate them with the gravity vector (in the free surface case) in a weighted force vector. Next CPU task passes this force vector of each particle and the GPU for double integration in time so a new particle position can be found. The resulting new position vector finally is computed by the Kernel of collision. As described in Section 3.3.3, the Kernel of particle-triangle collision computes the collision occurrence and change the new position into the normal-inverted vector when collision occurs. As mentioned earlier, the solid bodies and the domain contain colliding triangles to be processed as purely reflective. But there are vanishing walls in the domain built by colliding triangles. In this special case, if a colliding particle happens to collide against a domain vanishing wall, a marker will be included in its force vector and the particle will be deleted in the updating/adding/deleting CPU task.

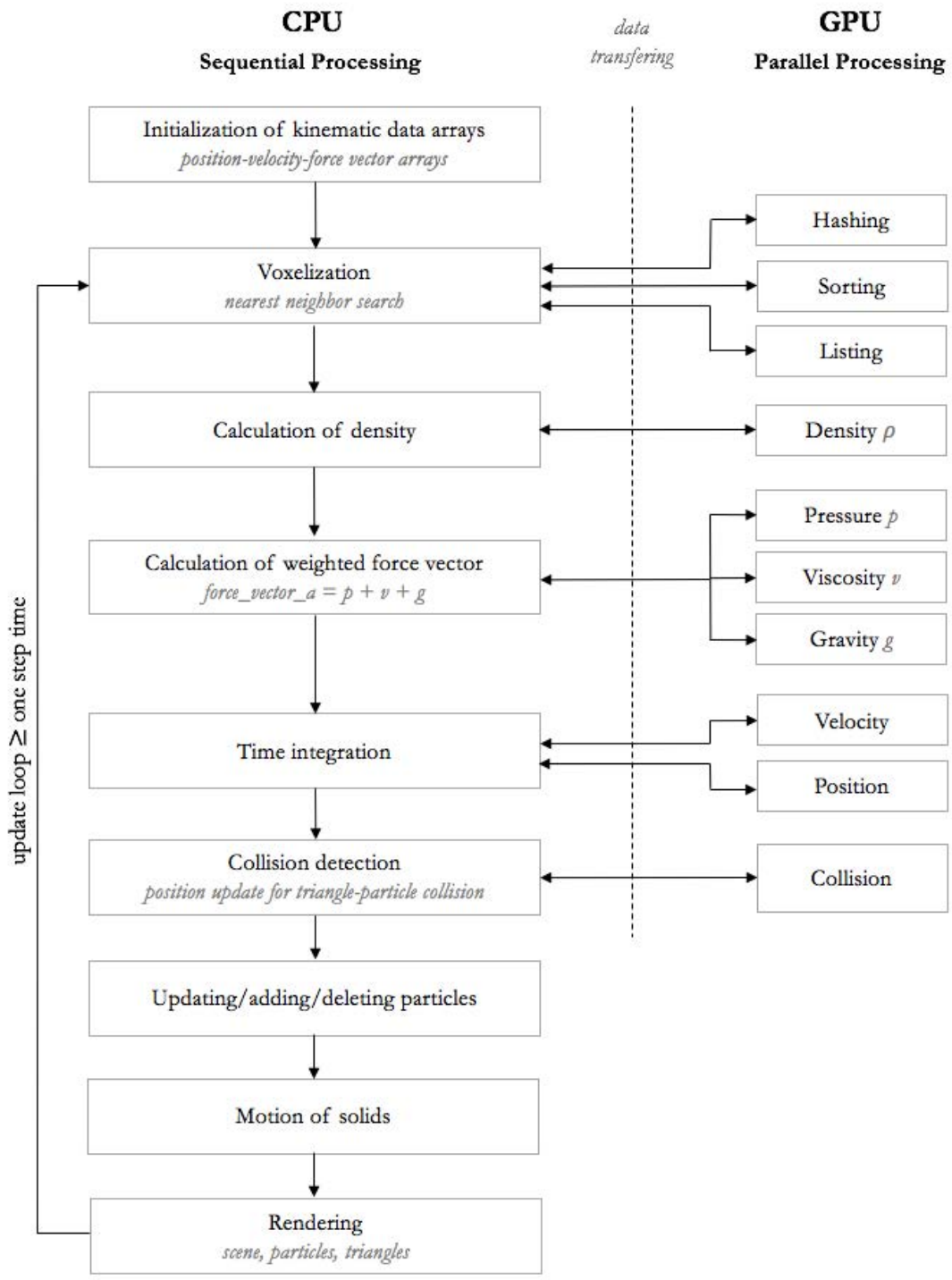


Figure 3.11. One time step CPU-GPU cycle implementation of SPH fluid motion. (Left) The sequential CPU routine tasks. (Right) the GPU parallel Kernels.

The quantity of motion of a solid includes the motion effect of its kinematics and the motion effect due to the fluid mass interacting on its boundaries. In many cases, those two motion forces are opposite and, depending on a desired control action, a resulting weighted force

vector will determine the motion of the solid and the fluid mass. In Chapter 4, the addressed cases offer two possibilities; the motion of solids due to the fluid mass and the opposite. In Figure 3.11, the CPU task of rigid body motion in the scene is always a function of free translation and rotation of triangles following certain rules of real-time processing that will be mentioned in next section.

Rendering, the last CPU task to compute in the update loop, is a time-consuming task for the GPU. However, OpenGL, as the drawing library in this work, allows the concurrency of the threads (multithreading), meaning that the rendering can be processed in parallel with other CPU tasks or GPU threads without blocking the GPU when rendering. This feature of OpenGL is extremely powerful in terms of the real-time performance, and it has gained to OpenGL its popularity.

3.6.2. Real-time processing

An unwanted issue in real-time simulation is the penetration of particles into triangles. Usually expressed in literature as the tunneling effect. This issue appears when the computation of collisions fails and particles penetrate triangles. This effect especially occurs with moving triangles in time steps that does not produce large enough distance to contain the motion step of triangles. Figure 3.12 illustrates the penetration effect; where **a** is the minimum distance between particles in rest (hydrostatic equilibrium), **b** the distance between times t_0 and t_1 of the same moving triangle within a valid range, and, **c** the same as **b** but in an invalid distance range. Notice that hydrostatic equilibrium is only possible in a free surface case; for the closed channel case, the pressure force between particles can increase to the point of reducing the space between particles to very low limits and then not allowing to meet the CFL condition.

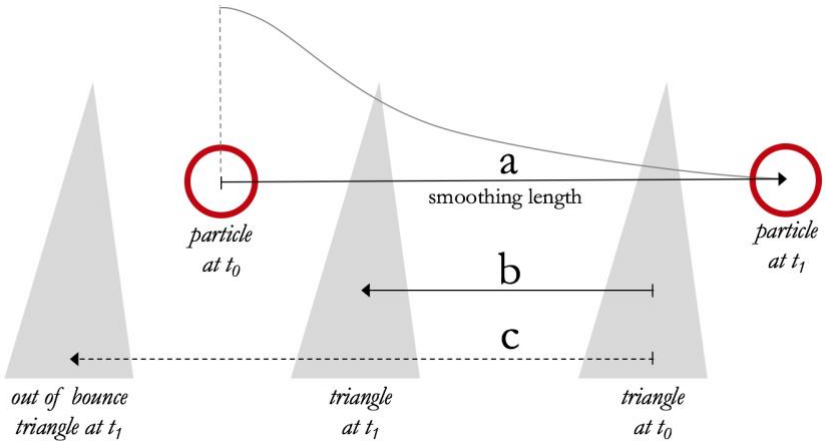


Figure 3.12. Spacing between particles to meet CFL condition.

Despite that the subject of this work is not related to the code programming, there is always need to provide the programming base reference of the implementations. In actual object programming, it is practical to adopt a dedicated library as an update loop structure. In our case, GLUT (OpenGL Utility Toolkit) [21] is employed as a dedicated C++ library for the software update loop structure. GLUT is cross-platform and provides full coverage for the window handles, the input events and the OpenGL context. It also provides the programming logic to run a real-time simulation loop with less effort and implication of programming low level window handles and input callbacks. Latest releases of GLUT have been developed under the name of freeGLUT [21].

3.6.3. Final formulation

To provide the sequence of the equations that were implemented through the list of tasks in Figure 3.11, the following Table 3.1 is provided. In the Section of Abbreviations. nomenclature and notation, one can find all the included variables in Table 3.1 as well as the mathematical notation.

No.	Equation
3.4	$a_i = -\frac{1}{\boldsymbol{\rho}_i}\nabla \boldsymbol{p} + \frac{\mu}{\boldsymbol{\rho}_i}\nabla^2 \boldsymbol{v} + g$
3.5	$p - p_0 = k_p(\rho - \rho_0)$

3.6	$p = p_0\left(\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right)$
3.11	$\frac{\nabla p(\mathrm{r})}{\rho_i} \approx \sum_i m_i \left[\frac{p(\mathrm{r}_i)}{\rho(\mathrm{r}_i)^2} + \frac{p(\mathrm{r})}{\rho(\mathrm{r})^2}\right] \nabla_i W(\mathrm{r} - \mathrm{r}_i, h)$
3.13	$\nabla^2 v(\mathrm{r}) \approx \sum_i m_i \left[\frac{v(\mathrm{r}_i) - v(\mathrm{r})}{\rho}\right] {\nabla_i}^2 W(\mathrm{r} - \mathrm{r}_i, h)$
3.15	$W(\mathrm{r} - \mathrm{r}_i, h) = \alpha_D \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & 0 \le q \le 1 \\ \frac{1}{4}(2-q)^3 & 1 \le q \le 2 \\ 0 & q \ge 2 \end{cases}$
3.16	$V_p = V_f \,/\, N_{max}$
3.17	$d_{rest} = 0.87 \sqrt[dim]{V_p}$
3.18	$h = 2\, d_{rest}$
3.21	$\mathrm{r}_{n+1} = \mathrm{r}_n + v_n \delta t + \frac{1}{2} a_n \delta t^2$
3.22	$v_{n+1} = v_n + \frac{1}{2}(a_n + a_{n+1})\delta t$
3.23	$\delta t \le \lambda \frac{h}{c_s}$
3.24	$voxel\_delta = (domain\_max - domain\_min)/voxel\_spacing$
3.25	$hash\_2d(p) = floor(x/voxel\_spacing) + (floor(y/voxel\_spacing) * voxel\_delta)$
3.28	$(\mathbf{x} - \mathbf{v_0}) \cdot \left((\mathbf{v_2} - \mathbf{v_0}) \times \mathbf{n}\right) > 0$

Table 3.1. Employed equations in SPH implementation.

So far, Chapter 3 provides the theoretical basis of the fluid model and its implementation details. This fundamentals define the required fluid dynamics block in Figure 1.1. The following Chapter 4 will provide the solid (robot) kinematics of three main cases that operate together with the fluid dynamics model, described in this Chapter, to define the real-time and interactive experimentation.

Chapter 4

Coupling robotics and fluids

In Chapter 3, full detail has been provided on how to implement a real-time and interactive SPH framework for a concrete fluid dynamics model. Now, in Chapter 4, this simulation foundation and framework is used to set the first motion kinematics primitives that will allow to test a benchmark case for validation purposes, a rotational profile case for kinematics study, and a liquid pouring case for free surface modelling and control.

4.1. Benchmark case

In CFD, a classic case of study is the downstream vortex analysis in a flow through a static cylinder. This case allows to investigate the behavior of flow in the laminar and turbulent regimes. The Reynolds number (Re), which is the ratio of inertial forces over viscous forces, helps to define the influence of these two types of forces for a given flow condition.

To deliver in this work a validation model of SPH vs. real physics of the flow, the flow around a cylinder case is presented, which is a very well-studied case, not only by experimentation, but also by simulation using the SPH method [78]. This is the main reason for choosing this kind of flow.

4.1.1. Flow around a Cylinder

In this Section, the benchmark case fundamentals are presented in order to define a comparison reference. The general flow topologies for different ranges of Re are presented to compare the reference topologies vs the simulated ones in next sections. The flow separation in theory based on the drag coefficient curve is also presented. Here, since further experiments (Sections 4.2 and 4.3) are considered purely laminar, the requirement to the SPH flow is to find the transition from pure laminar to flow separation regions and provide to these cases a proper limit of flow regimes. And, finally, the Karman Vortex Street (KVS) effect as a reference guide, that is defined by characteristic vortex spacing and frequency, needs to be simulated to identify the equivalence fluid model.

4.1.1.1. General flow topologies depending on Reynolds number

Equation (4.1) describes Re in terms of fluid density ρ_i, fluid velocity v_i, cylinder diameter d, and dynamic fluid viscosity μ. This is the theoretical function of Re which will be used in further validation of theoretical vs simulated data.

$$Re = \frac{\rho_i v_i d}{\mu} \tag{4.1}$$

Figure 4.1 shows the Re intervals by going from low values (laminar) to high values (turbulent) in real flows. In diagram 1, from top to bottom, the pure laminar region behavior of flow is shown having $Re < 5$. In diagram 2, the vortex separation region behavior of flow is shown for $5 < Re < 40$. In diagram 3, there is a laminar separated periodic behavior of the flow known as the Karman Vortex Street effect. This effect is a repeating pattern of swirling vortices caused only at the range of $40 < Re <$ 250K-300K, where the wake behind the cylinder is still laminar. Re can go up to 200K-300K and continue to show similar KVS effects but with a turbulent wake (but boundary layer is still laminar in the separation point). Transition between laminar to turbulent boundary layer regime goes from around 200-350K and so on.

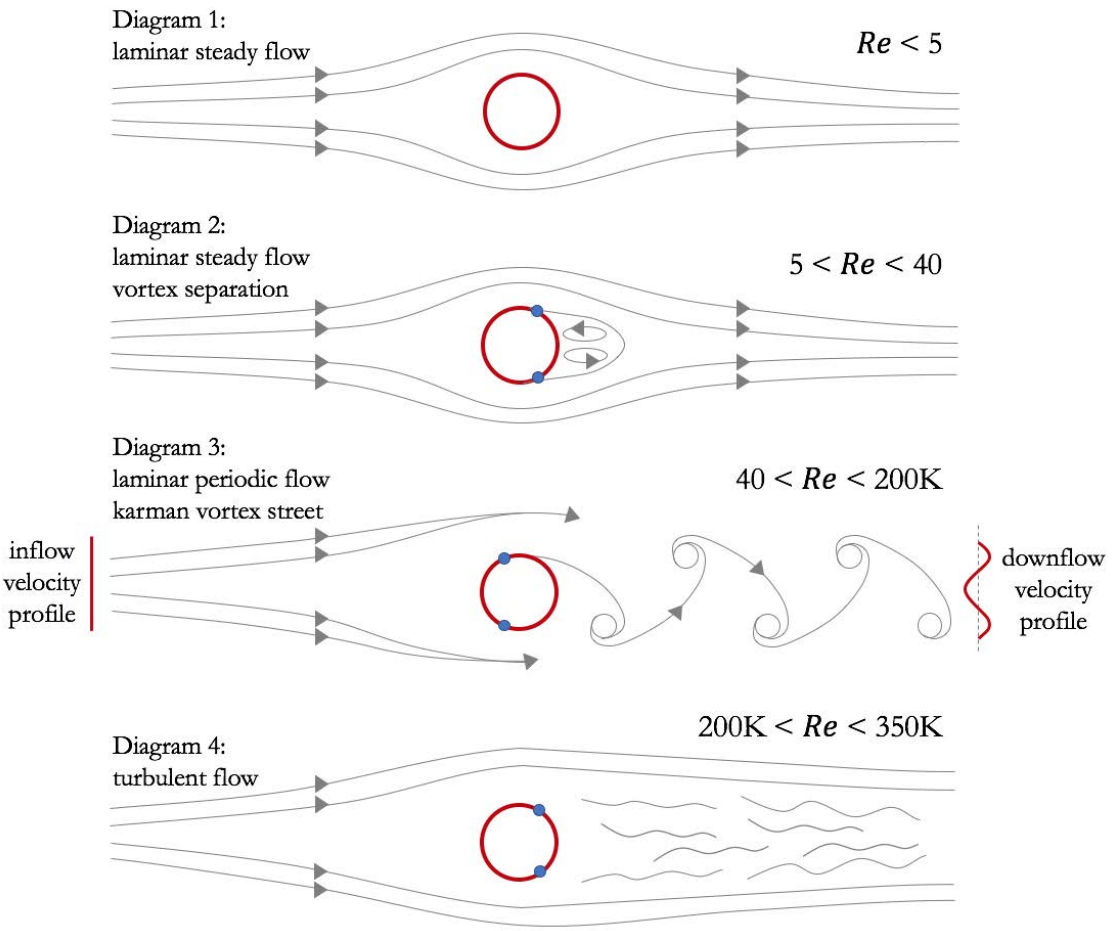


Figure 4.1. Reynolds number effect on flow around a cylinder. Blue dots represent flow separation point.

KVS effect is supposed to happen in our SPH simulation so fluid would be modeled within the laminar boundary layer regime (diagram 1-3), but with boundary layer separation and vortex street in the wake (diagram 3), as sketched in Figure 4.1. No turbulence model is used in this SPH work, but the focus is set on the ability of the methodology to capture complex phenomenon, such as flow separation and KVS.

4.1.1.2. Flow separation and drag coefficient

The drag coefficient is a dimensionless value defined to quantify the resistance of an object in motion in a fluid domain. Drag coefficient C_d equation used for a cylinder is written in Equation (4.2), where F^x denotes the force of a cross-section of the closed channel in x-

direction (streamwise), ρ_i is the initial or inflow density, v_i is the inflow velocity, and d the cylinder diameter. i denotes the inflow condition.

$$C_d = \frac{F^x}{\frac{\pi}{4}\rho_i {v_i}^2 d^2} \tag{4.2}$$

Figure 4.1 shows the range of *Re* where KVS effect is comprised; in this range dragging forces become higher for the cylinder. C_d and Re are typically related in literature as the model curve for the 2D cylinder case, here illustrated in Figure 4.2. This relation between C_d and Re should comply with our SPH simulations in order to validate the correctness of the numerical model and to proceed with further model couplings.

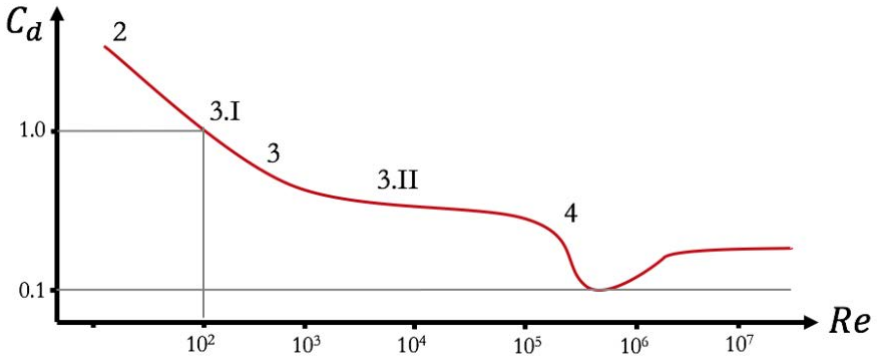


Figure 4.2. C_d vs. Re graph. Region 2 defines steady flow, region 3.I defines KVS with laminar wake, region 3.II defines laminar separation with turbulent wake, and region 4 defines turbulent boundary layer separation.

It could be interesting to fully simulate all regions of Re vs. C_d graph with SPH. Regions 3 and 4 are possible regions to implement in SPH, but Region 4, the region of turbulence, is not straightforward to simulate under standard SPH methods, as mentioned by Bauer and Springel in [79] and Adami et *al.* in [80], where at large values of Reynolds artificial viscosity vanish and simulations produce purely noisy particle motion.

4.1.1.3. Von Karman Vortex street characteristics

The KVS effect, illustrated in Diagram 3 of Figure 4.1, is characterized by vortex shedding. This is a sequence of vortexes generated by unsteady boundary layer separation when steady flow hits a bluff body. The main characteristic of this vortex shedding is the frequency of the vortex generation. The relation that describes the vortex frequency is given by the Strouhal number S_t. Equation (4.3) denotes S_t as a function of the vortex frequency $freq$, the cylinder (or circle) diameter d and the initial flow velocity v_i.

$$S_t = \frac{freq \cdot d}{v_i} \tag{4.3}$$

It has been experimented that the vortex frequency depends lineally with the initial flow velocity v_i. As Re equally depends on v_i, it has been estimated that S_t depends on Re [81]. For this reason, Re and S_t have been related by the graph presented in Figure 4.3. In this graph, two curves are presented, one for a smooth solid cylinder and one for the rough solid cylinder. Both only diverge in the extreme high values of Re, so both can find a constant mean value of S_t of 0.2 where, for a large interval of Re $(5x10^2 < Re < 10^5)$, this value set the operation range of the vortex shedding.

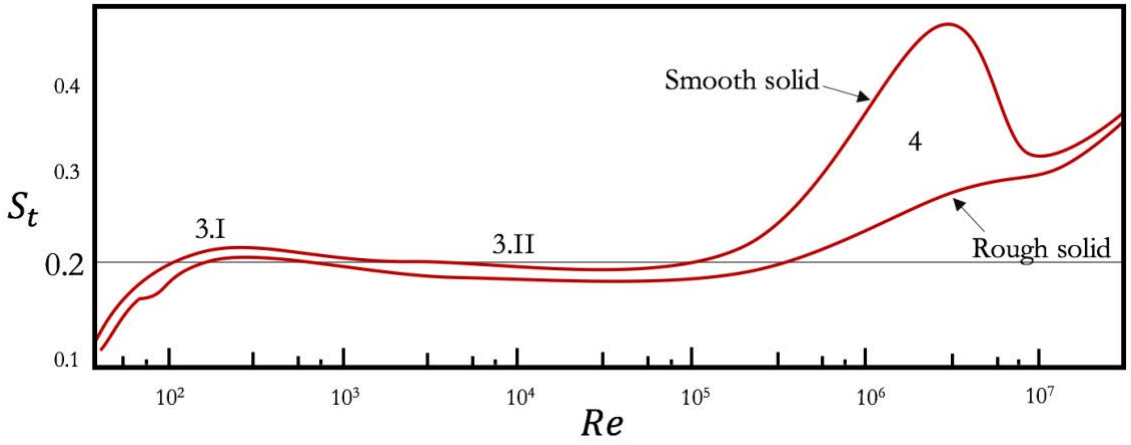


Figure 4.3. S_t vs. Re graph. Mean value of S_t at 0.2 defines the KVS effect interval of Re.

One important application of S_t is to find the vortex frequency of the fluid so the natural frequency of the solids hit by the flow should not match together. If this occurs, resonance of the entire system can be achieved and unwanted effects may happen. But for this fluid simulations,

the importance of S_t is to match the KVS interval in the S_t vs. Re graph so the resulting numerical fluid behavior is similar to the experimental observations.

4.1.2. Practical implementations

Typically, the flow-around-a-cylinder simulations are made in 2D space for symmetry reason, also offering an easier visual interpretation. Even though the SPH model is programmed in a 3D space, it has been restricted to 2D for the same reason using a third zero-dimension for particle positions, velocities and forces. Although data structures remain in 3D format.

The schematic setup of the 3D domain for the benchmark case is illustrated in Figure 4.4. It is a parallelepiped channel where 2D flow enters as a line of particles from inlet U and exits through outlet U'. The outlet U' is defined as a vanishing wall. U is the entry port of particles as described in Section 3.3.3. This entry of particles is configured with certain particle velocity in order to fill up the domain. Flow behavior is studied from the entry stimuli plates located at L_{S_i} and to the exit stimuli plates located at L_{S_o}.

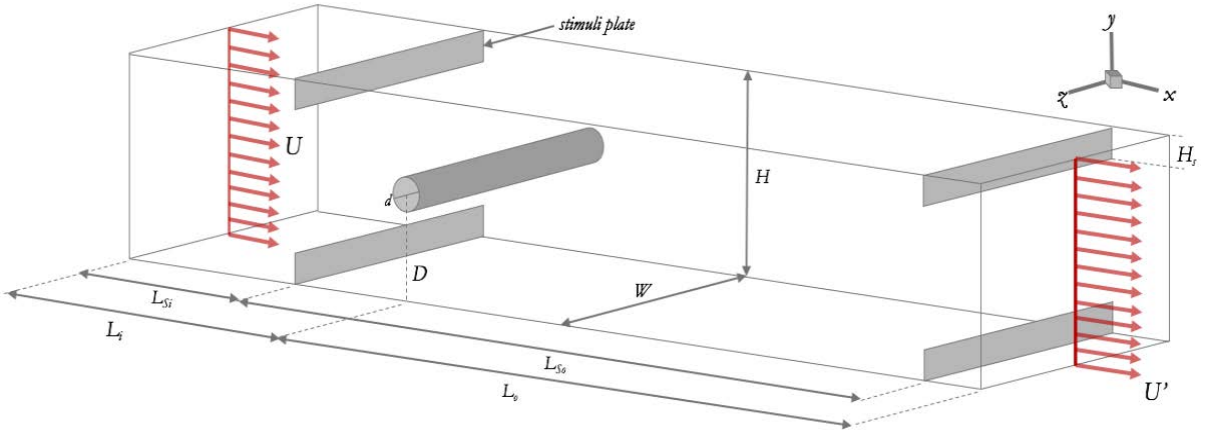


Figure 4.4. 3D domain setup for benchmark case.

Table 4.1 shows four groups of parameters that setup the simulation domain. First group includes the fluid constants; these constants are used to compute the Kernel of force integration (see Section 3.5.2) that obtains particle velocities and positions by leapfrog integration scheme. Second group includes the three forces in momentum Equation (3.4); note that gravity is not

included in this case. In Section 4.3 case, the gravity term is included as a volumetric force. The third group is the domain specifications that defines the domain dimensions and scale. The simulation scale needs to set a floating normalized computation scheme so particle velocities and accelerations converge numerically.

Parameter	**Value**
Fluid Constants	
Density (ρ)	1000 (kg/m^3)
Dynamic viscosity (μ)	1.05x10^{-3} (N·s/m^2)
Inflow port position (U)	Line up with Y-axis at x=0
Particle initial velocity at L_{S_i} (v_i)	0.022 (m/s)
Particle mass (m)	0.5 (gr)
NSE forces used in Equation (3.4)	
Gravity force ($\boldsymbol{g}$)	Not used
Pressure force (∇p)	Used
Viscosity force ($\nabla^2 \boldsymbol{v}$)	Used
Domain Specifications	
Domain dimensions (L x H x W)	0.83x0.279x0.056 (m)
Simulation scale (*e.g.* screen dimension x simulation scale = dimension in meters)	0.0111818
Domain volume (V_d)	0.01296 (m^3)
Cylinder diameter (d)	From 0.05 (m) to 0.06 (m)
Cylinder position (D x L_i)	0.13977x0.313 (m)
Stimuli size (W x H_s)	0.056x0.053 (m)
Entry stimuli position (L_{S_i})	0.078 (m)
Exit stimuli position (L_{S_o})	0.78 (m)
Simulation specifications	
Time step (δt)	0.002 (s)
Maximum number of particles	50,000
Particle space motion	2D (acceleration in z-axis is zero)

Table 4.1. Fluid constants, NSE forces, domain dimensions and simulation specifications.

In Figure 4.4, four stimuli plates are placed within the domain, two next to the entry of particles and two reaching the end of the domain. These first two entry stimuli are placed to set a proper fluid-like entry of particles. Particles enter the domain in line formation for a convenient computational arrangement, however in this formation fluid forces are not acting yet. So, in order

to present a more likely fluid behavior at the beginning of the domain, these first two stimuli plates, located at L_{S_i}, cause particles to hit between each other and fill faster the domain, before reaching the vicinity of the cylinder. In Section 3.3.3, full details have been provided to describe the purposes of having entry plates to reach a uniform density field.

4.1.3. SPH results

In the following three subsections, the results of simulating the benchmark case are presented. First, the flow pattern found with the KVS effect is presented. Second, the velocity profiles during running simulations are presented at different domain heights. And third, two flow simulations are offered with two different Re.

4.1.3.1. Flow pattern

The simulation time is divided into three phases; phase 1, the domain filling, described in Section 3.3.3; phase 2, the adjustment; and phase 3, the proper simulation. The filling is the initial preparation phase that goes from the time of the entry of particles to the moment of time when the domain saturates (filled up with particles). The adjustment is the following preparation phase of reducing the inflow velocity (inflow particle velocity v_i) so the incoming number of particles equals the outgoing number of particles, having a stable number of particles of 50,000 approximately for 2D for the sample simulation presented in Figure 4.5. And finally, the simulation stage itself where vortex shedding develops along the flow direction.

Figure 4.5 illustrates frames of the running simulator in 2D for each of the stages. In blue the cylinder and the stimuli plates, in yellow the domain boundary lines, in magenta the particle flow, and in green the metering sections. From second 2 to 57 the particle filling stage, from second 57 to 90 the velocity adjustment stage where inflow equals outflow, and from second 96 to 132 and beyond the KVS effect. Notice that between the simulation frame at 96 second and at 108.8 seconds a full KVS cycle is presented. Then, a period of time of 12.8 seconds and a KVS frequency of 0.078 hertz. The metering sections are x-planes that cuts the flow to find the

average pressure and velocity of the flow. Please refer to Video [4.1], to illustrate the KVS effect in a running timed simulation.

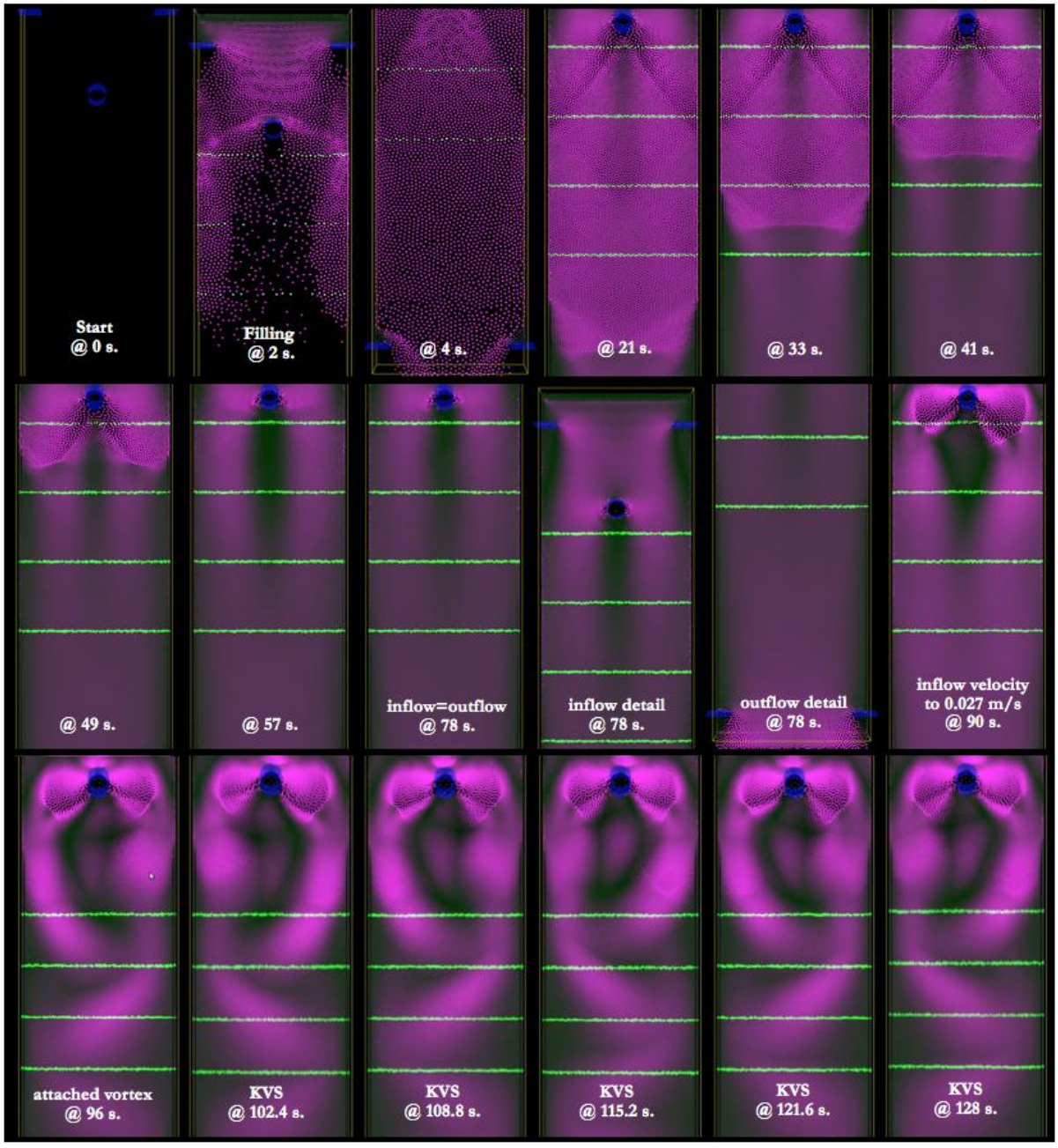


Figure 4.5. KVS effect during running simulations.

4.1.3.2. Velocity profiles

To investigate and validate the convection of the vortexes during KVS and how these vortexes dissipate as a function of distance, simulations have been performed to find crests of

maximum velocities at four different distances of measurement. In Figure 4.5, four horizontal green lines shows the capturing heights used to meter velocities along four sections of the domain space. The first capturing line is the domain from top to bottom in Figure 4.6 and so on for the following lines. Each line captures velocity data of particles nine times every 2.13 seconds.

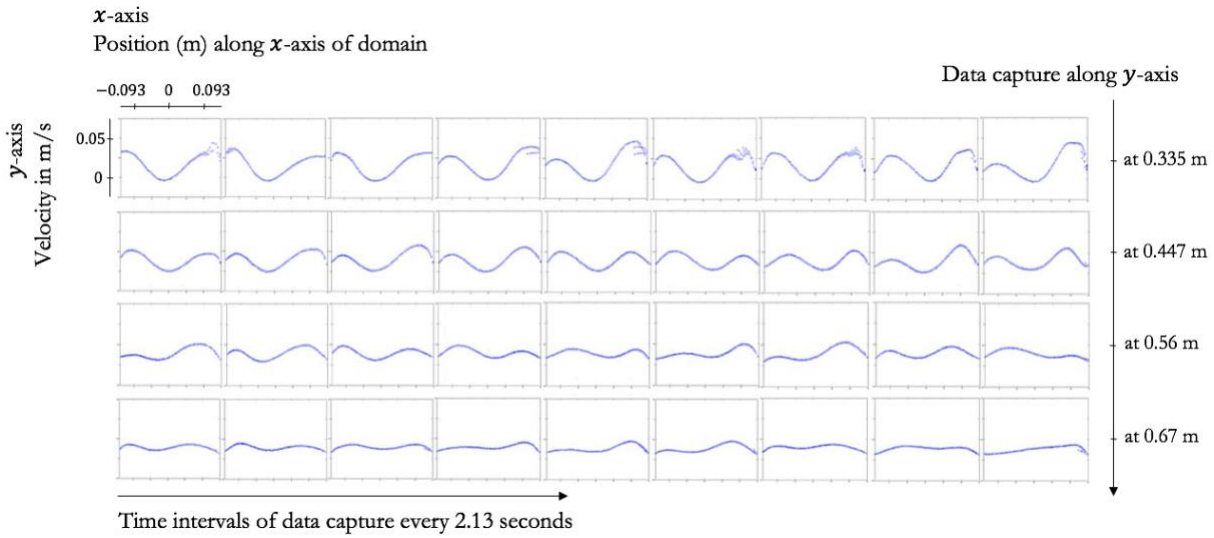


Figure 4.6. In-line particle velocities along x-axis at different domain distances during KVS.

Opposite to simple inspection of Figure 4.5, results from Figure 4.6 allows to find with more precision the frequency of the vortex shedding by finding the maximum velocity values. From Figure 4.6, between first sample data (top-left) to the sixth sample date (in the same line) a full KVS cycle is shown. A value of 2.13 seconds multiplied by 6 data captures gives a 12.78 seconds of period and a value of 0.078 hertz for the KVS frequency.

Using the parameters in Table 4.1, a Re value of 1,047 is found using Equation 4.1. This is not a low value of Re yet for a full laminar KVS effect, but valid in the transition regime of KVS according to [82]. Such validation allows to proceed with other closed channel simulations.

Figure 4.7 illustrates the findings of the KVS effect in Region 3 of the C_d vs. Re graph (Figure 4.2). The vortex separation initiates in $Re = 2.5\text{x}10^2$ and full KVS can be appreciated at $Re = 1{,}047$ aprox.

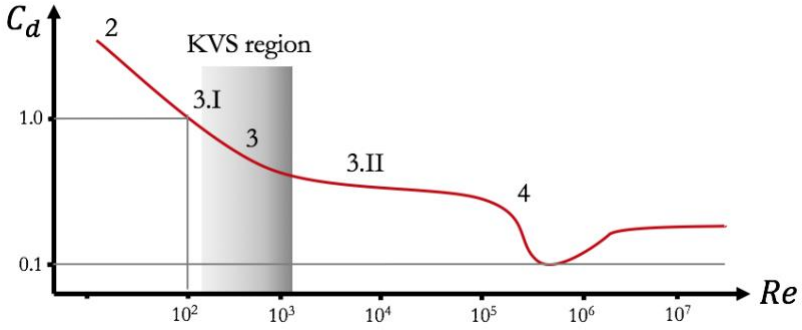


Figure 4.7. KVS effect findings in the C_d vs. Re graph.

4.1.3.3. Reynolds number effect

As mentioned earlier, Re is an indicator of how laminar or turbulent a given flow is. Moreover, it can be set to figure out a parameter range where certain flow effect is expected, like the KVS effect. In this work, the intension is to provide evidence on which is the analogous parametrization that our SPH implementation can employ to match a conventional KVS experiment of fluids in typical clear water flow conditions where robotic mechanisms may act.

In simulations, illustrated in Figure 4.5, it is shown that, at Table 4.1 conditions, KVS effect is achieved approximately at ~50,000 particles. Re value is ~1,047. In Figure 4.8, from left to right, the first two simulations show a half cycle of KVS effect with a smaller cylinder diameter than the next two. The latter two simulations show the same noticeable KVS effect than the former two even by increasing Re number. These results demonstrate theoretical value ranges of Re shown in Figure 4.1.

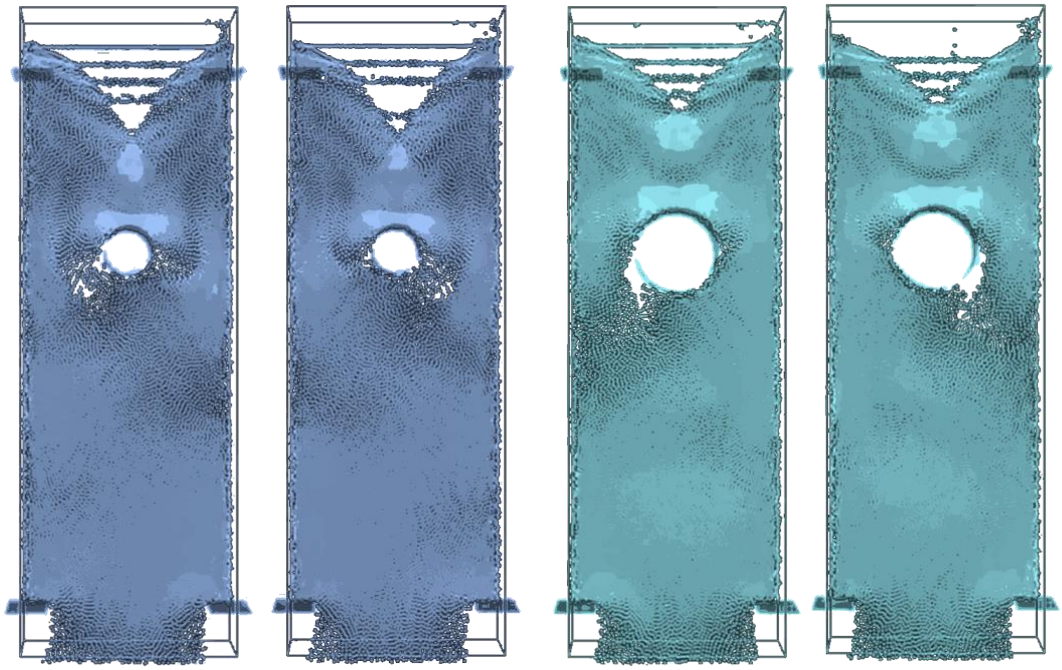


Figure 4.8. KVS effect during running simulations with different cylinder diameter.

The running simulation in Figure 4.8 shows that KVS effect can be achieved at low number of particles with our SPH implementation. This is a promising result for real-time experimental purposes using a user interaction scenario. Now, it has been physically tested fluids with proper flow and vorticity timing, not only from the visual perspective but also from its own theoretical physics. To be clear from this point and so on, since complexity of fluid physics is extensive, our work only studies fluids from its SPH motion dynamics based on parameters evaluated from NSE forces. Different parametrization would derive into different results though.

4.1.4. Conclusions

As far as the case of this Section 4.1, the flow domain has been especially designed and implemented successfully to simulate a flow passing a cylinder. This aim includes the domain dimension setup and filling, the implementation of the inlet and outlet of particles, and the KVS effect finding with different cylinder sizes.

Another remarkable data obtained with the simulations carry out in this section is the Strouhal number value at which KVS is noticeable. From Figure 4.3, this value is around 0.2

along 10^2 to 105 of Re values. By using Equation (4.3), the vortex frequency findings from Figure 4.6 (0.078 hertz), the cylinder diameter (0.05) and the initial flow velocity v_i (0.022 m/s), a Strouhal number value of 0.177 is calculated. This value confirms the proximity to the 0.2 value where KVS effect is expected.

The conducted simulations show promising results in the number of particles and time step that have been used. Marrone et *al.* [47] had similar results in the development of the initial KVS effect (vortex wake) right behind the cylinder face but with $Re = 150$. Ours had a theoretical value of $Re = 200$ during vortex wake and a simulated value of $Re = 1{,}047$ during full development of the KVS effect. Not to mention that, in [47], the ghost technique is used to handle particle-solid collisions, allowing to have better no-slip boundary conditions. Contrary to our simulations where a reflective wall technique has been used and no-slip condition is simulated. As a conclusion, the interval found of Re seems to match theoretical values properly and fluid vorticity works within the expected limit of the laminar to transitional regimes.

4.2. Profile case

Hydrodynamics of profiles is a very well-studied subject for both navigation (hydrofoils or flaps) and propulsion (propellers) both in CFD and experimentally. Yet, it is an open subject in underwater locomotion simulation to develop comparative models [83]. Flaps and propellers are one of the basic means of motion in underwater domains, so, the purpose of this Section 4.2 is to test rotation motion of a typical profile for underwater domain where fluid forces generate a proportional torque that can be controlled to follow an angular reference. In the following Section 4.2.1, the basic theory involved in a rigid body profile dynamics for angular motion is described.

4.2.1. Profile Rotation

Simulation of rigid body dynamics involves all the concepts of motion equations, from position/orientation to angular momentum. That being said, in next sections, the definitions and

the implementation logic of these concepts are described to form the proposed motion system. In particular, using the case of an underwater profile that rotates in a closed channel without involving free surface gravity.

Propulsion or navigation profile model are designed by airfoil profiles. These profiles are classified in NACA series based on their application [84]. Our choice is a general-purpose airfoil (typically used for horizontal tails in navigation), the NACA-0006 model. The first two digits (00) in this model represent an airfoil with symmetrical faces and the last two digits (06) the percentage of thickness to chord. The 3D model was obtained from a 2D NACA profile of 400 datasets. As mentioned in Section 2.3.4, the rigid body primitives are triangles so the 3D profile model is shaped by 800 triangles. Figure 4.9 shows the NACA-0006 model in 2D and 3D space.

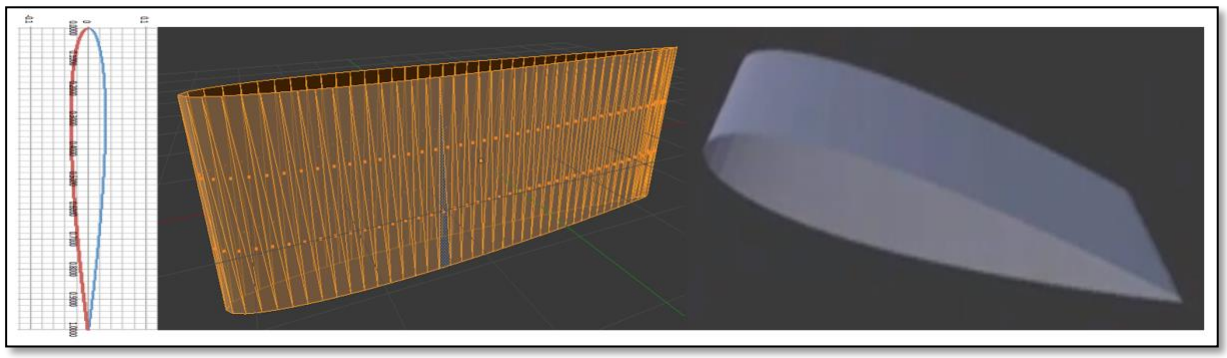


Figure 4.9. 2D and 3D NACA-0006 model designed and employed as rigid body.

The study of flow for different profile models is not going to be investigated, but only the fluid effect and the forces to which this selected NACA airfoil profile interacts with. Future work could easily follow the testing profiles (NACA-00XX Series) for underwater propulsion applications like in [85]. For sizing the profile according to the space domain, a chord length (line) has been adapted to a value of 151.25 mm. Thus, a profile width of 4.5 mm for the NACA-0006 model.

4.2.1.1. Rotation in the space

From a 3D motion point of view, simulation of solid bodies can be modelled equally to the simulation of particles because solid bodies are made from triangle vertices in the 3D space. However, particles do not have orientation but only location in the domain, while a solid body of

n vertices is described by its position and orientation. Equation (4.4) describes the composition of a vertex p_o of a rigid body by a function of time.

$$\mathbf{p}(t) = \mathrm{R}(t)\mathbf{p}_0 + \mathbf{x}(t) \tag{4.4}$$

$$\mathrm{R}(t) = \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix} \tag{4.5}$$

$\mathbf{p}(t)$ is the location and orientation of $\mathbf{p}_0$, where $\mathrm{R}(t)$ the matrix of rotation defined in Equation (4.5) and $\mathbf{x}(t)$ the linear traslation vector of $\mathbf{p}_0$ with respect to the origin. For the profile rotation case, the translation motion of the profile is not of interest, so $\mathbf{x}(t)$ is cancelled out, and only the changes of the profile spin are processed by computing the rotation motion. This matrix changes accordingly to internal or external forces such as collisions, inertia, and control actions.

It is always important to notice that object position and orientation in 3D space can be described locally (object space coordinates) or global/absolute (scenery space coordinates). In the implementation of this work, the computation by default is global and only local if specified. So, when an object is away from the global coordinates this is located it into the global center and its new matrix of rotation $\mathrm{R}(t)$ is processed. For this reason, Equation (4.4) includes the translation variable $\mathbf{x}(t)$ for the same rotation purpose.

Linear velocity $\dot{\mathbf{x}}(t) = \mathbf{v}(t)$ describes how the position of a vertex changes over time due to translation. In the profile rotation case, again, the rotational motion is of interest, so linear velocity is not computed as mentioned before. Then, the only motion dynamics left of the solid body is a rotation degree of freedom (DoF). In many cases a given angle of rotation requires to be reached or maintained as a signal reference in a control task. This can be achieved by applying an angular velocity $\boldsymbol{\omega}(t)$ to the solid body vertexes. Figure 4.10 shows the composition vectors $\mathbf{a}$ and $\mathbf{b}$ of a particular vertex defined by vector $\mathbf{p}(t)$, and its angular velocity given by $\boldsymbol{\omega}(t)$ in one axis. $\boldsymbol{\omega}(t)$, the spinning velocity in revolutions/time, is a vector velocity while $\mathrm{R}(t)$ is a matrix of rotation that defines the orientation of the vertex so it is needed to relate these two variables into a $\dot{\mathrm{R}}(t)$ variable.

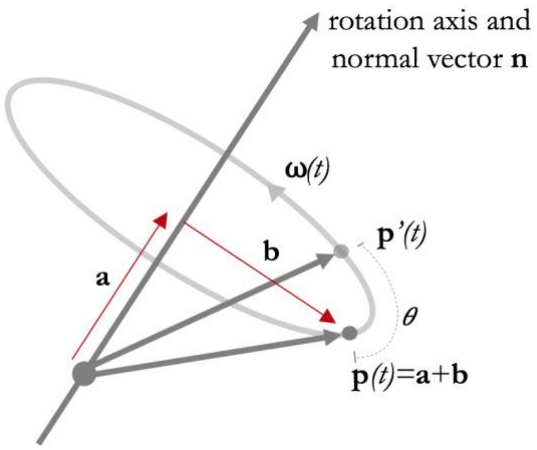


Figure 4.10. Vector of rotation $\mathbf{r}(t)$ and angular velocity $\boldsymbol{\omega}(t)$ relation.

A matrix of rotation velocity $\dot{\mathrm{R}}(t)$ is obtained due to an angular velocity $\boldsymbol{\omega}(t)$ by Equation (4.6) relation from [13]. The symbol $*$ does not define product but a matrix-vector operation expressed in Equation (4.7) is

$$\dot{\mathrm{R}}(t) = \boldsymbol{\omega}(t) * \mathrm{R}(t) \tag{4.6}$$

$$\dot{\mathrm{R}}(t) = \left(\boldsymbol{\omega}(t) \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \quad \boldsymbol{\omega}(t) \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} \quad \boldsymbol{\omega}(t) \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right) \tag{4.7}$$

If motion dynamics involves forces like in this case, there is need to compute mass. It is common to assume a body volume made of many particles, each of these particles can be set to be a vertex of the rigid body geometry. So, the total mass M of a body is the summation of the all our vertices masses m_i as $M = \sum_{i=1}^{N} m_i$.

The center of mass in symmetrical profiles may vary upon their geometry, but all have the same definition shown in Equation (4.8); where M is the object mass, m the vertex mass and, $\mathbf{p}_0$, the location vector where the center of mass in the object become zero.

$$\frac{\sum m_i \mathbf{p}_{0_i}}{M} = 0 \tag{4.8}$$

Neither linear momentum nor linear velocity take place in this profile simulation case, however, angular momentum does. Angular momentum $L_m(t)$ is important because, as Equation (4.9) describes, a tensor of inertia $I(t)$ is needed to know how mass is distributed

around the axis of rotation. Profiles may or may not be shaped symmetrically so the angular momentum would change depending on the selection of the center of rotation.

$$L_m(t) = I(t)\,\boldsymbol{\omega}(t) \tag{4.9}$$

Linear momentum is defined by the product of the object body mass with the linear velocity. The object mass is a scalar so linear momentum is simple to compute. However, the angular momentum, which is of our interest, requires to define a matrix term of the inertia tensor.

Tensor of inertia calculation is not trivial to compute, since, during rotation motion, the tensor of inertia evolves according to the axis of rotation. The way to computationally handle this issue is by using object space coordinates for the calculation instead of using the scenery space coordinate as usual. This assumption sets a constant value to the inertial tension, and allow us to compute it once and for all the simulation time steps. Of course, this lowers the simulation accuracy in terms of the small rotational velocity changes that take place during neglected angular momentum changes. Parallel work may go through the study of rigid body shapes and their moments of inertia in SPH, but for this work the motion of rigid bodies is attached to basic symmetrical rotational motion.

The constant value for the tension of inertia was approximated using the case of a solid cylinder section and the theorem of perpendicular axis. This defines in 2D that, if a planar object has rotational symmetry such that I_x and I_y are equaled, then $I_z = 2I_x = 2I_y$, having an approximated inertial tensor for the profile of $I_z = \frac{m \cdot {pr_w}^2}{2}$ and $I_x = I_y = \frac{m \cdot {pr_w}^2}{4}$, where m is the profile mass and pr_w is the profile width (or profile thickness) in y direction. Simulation scale is applied to radius length so inertia tensor is expressed consistently to the domain.

A more accurate definition of the inertia tensor can be found in [13] where $I(t) = \mathrm{R}(t) I_{body} \mathrm{R}^T(t)$. In such case, I_{body} is constant during the simulation but $I(t)$ is not, it changes as the matrix orientation $\mathrm{R}(t)$ changes each time step providing a more proper angular momentum calculation but with a larger and sequential processing cost.

4.2.1.2. State equations of motion

Now that all needed motion dynamics concepts have been described in section 4.2.1.1, the state equations of motions for profile case can be described for fluid and solid coupling. For a rigid body, $Y(t)$ is defined in Equation (4.10) as a set of the state variables that defines the orientation and torque applied to the profile.

$$Y(t) = \begin{pmatrix} \theta(t) \\ \tau(t) \end{pmatrix} \begin{matrix} \rightarrow \ angular\ position \\ \rightarrow \ torque \end{matrix} \tag{4.10}$$

It cannot be ignored, that in a 2D closed channel the Kutta-Joukowski theorem allows to find theoretically the force that experience a solid body in a uniform flow. This exerted force f, expressed in Equation (4.11), is the product of the fluid density ρ, the upstream (initial) flow velocity $\vec{v}$, and the line integral of the velocity field or circulation Γ.

$$\mathrm{f} = \rho \vec{v} \Gamma \tag{4.11}$$

Equation (4.11) allows to validate the conservation of motion based on circulation. But for this work, as a time and space discrete model of the fluid is provided by SPH, a solid-fluid coupling model based on the net force is used. Besides that Equation (4.11) only works for 2D scenes while a net force model would work for 3D too.

The mass of the rigid body profile and its inertia tensor are assumed constants throughout the simulation. Each time step, the calculation problem is centered in obtaining the angular position of the profile from the fluid force. For continuous systems, fluid flowing around the body generates a local net force f on each infinitesimal region of the body. This is an exception in particle-based fluids, where fluid is not in continuous contact with the body due to space discretization but only at certain discrete time steps and at certain particle-to-body collision range. Anyhow, a resulting force $\vec{\mathbf{f}}$ is obtained by vector addition of all particles forces $\vec{\mathbf{f}}_i$ during collision detection. See central diagram in Figure 4.11 for better illustration.

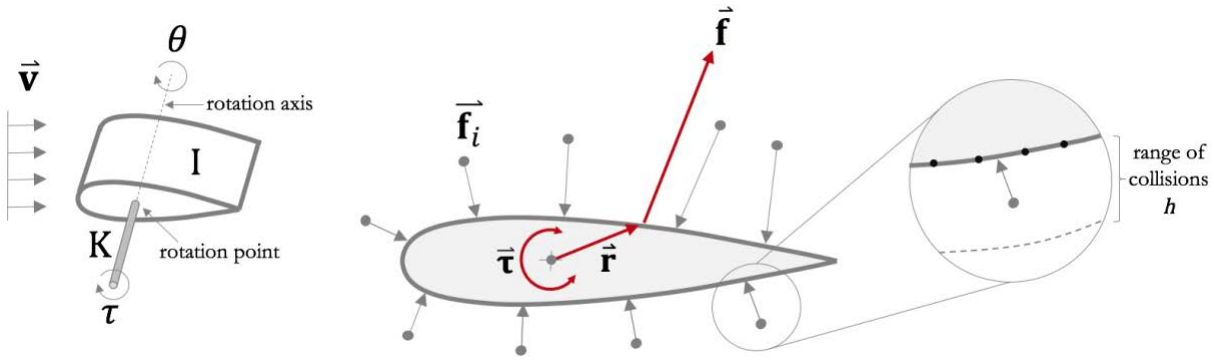


Figure 4.11 Resulting force $\vec{f}$ and torque $\vec{\tau}$ from particle collision.

The resulting net force $\vec{\mathbf{f}}$ is converted into a resulting torque $\vec{\boldsymbol{\tau}}$ by cross product with the distance $\vec{\mathbf{r}}$ of the point of rotation, as in $\vec{\boldsymbol{\tau}} = \vec{\mathbf{f}} \times \vec{\mathbf{r}}$, and converted to angular acceleration $\boldsymbol{\alpha}(t)$ by Equation (4.12). Notice that I is required to compute angular acceleration. I is precomputed in the initialization block for a constant moment of inertia as well as the initial vectors of orientation (matrix of rotation) and angular velocity and acceleration.

$$\boldsymbol{\alpha} = \vec{\boldsymbol{\tau}}/_{I} \tag{4.12}$$

Now, from acceleration, the final angle of rotation $\theta(t)$ can be found by the Euler integration method defined in Equation (4.13) for angular velocity and in Equation (4.14) for angular position. Notice the scalar indication of variables; this comes from vectors magnitudes. By final angle, the next time step angle is $\theta_o = \theta(t)$ and $\theta_f = \theta(t + \Delta t)$.

$$\omega(t + \Delta t) = \omega(t) + \alpha(t)\Delta t \tag{4.13}$$

$$\theta(t + \Delta t) = \theta(t) + \omega(t)\,\Delta t \tag{4.14}$$

Recall that the collision approach is purely dissipative where no spring nor damper elements actuate in the collision region. By using Equations (4.13) and (4.14), the angular acceleration is been approximated to be constant each time step that is only valid for small rotational steps.

θ would be needed to be translated to a rotational matrix $\mathrm{R}(t)$ if 3D flow modeling is involved. However, since the airfoil rotation is developed under one axis for this model case, instead of using unit quaternion operations to find the new $\mathrm{R}(t)$, this matrix can be simplified to

a rotation vector and computed using Rodrigues formulae defined in Equation (4.15). That way it can be modelled under the angle θ rather than in quaternions where normalization must be processed each computation of R to avoid numerical drift. In Rodrigues formulae, $\mathbf{x}'$ is the rotated version of vector $\mathbf{x}$, $\mathbf{n}$ is the normal vector of the rotation plane, and θ the angle of rotation, see Figure 4.10 for reference. $\mathbf{x}'$is used to situate the new position of that solid vertex in the 3D space.

$$\mathbf{x}' = \mathbf{x} + (\sin\theta)(\mathbf{n} \times \mathbf{x}) + (1 - cos\theta)\big(\mathbf{n} \times (\mathbf{n} \times \boldsymbol{x})\big) \tag{4.15}$$

To the right in Figure 4.11, a collision range is illustrated. Only the particles within this region are tested for collision with the solid. There is no need to process a particle search process since the process used in Section 3.5.1 for closest neighbor search, where the collision particle voxels are listed in a proximity order and ready to be iterated, can be employed.

4.2.2. Profile rotation control

Coupling forces could be used to orient a profile for free spinning, but for this work a controlled profile requires to use the resulting torque for navigation or vortex motion. For instance, to control the profile with a reference angle. That reference angle could be provided by a cruise control in a navigation application. This is the test case for profile rotation control developed in this work.

The profile must be conceived as an attached mechanism of rotation, may be a shaft on a bearing support, so there will be a spring element and occasionally a damping element in any real profile rotation system. To provide a generic mechanical model that fits this profile motion case of rotation, a second order system has been adopted given by the greatest order of $\theta(t)$. The diagram to the left in Figure 4.11 describes the motion elements of the employed mechanical system. Equation (4.16) provides the canonical form of such second order system (SOS). Where K is the torsional spring element (stiffness) and I the moment of inertia or inertia tensor, both represent the coefficient in the SOS. Notice that the point of rotation is right at the center of gravity. As mentioned earlier, collision with particles is purely dissipative so no viscous effect in the boundary layer is observed and, then, no friction (damping) element is offered. $\tau(t)$ is the

calculated torque, described in Figure 4.11 and previously mentioned, using the net force that is the summation of all forces of collided particles.

$$I\frac{d^2\theta(t)}{dt^2} + K\theta(t) = \tau(t) \tag{4.16}$$

Equation (4.16) defines the plant function in a closed loop feedback system. SOS of the form of Equation (4.16) in principle has an undamped transient time response ($\zeta = 0$). This means that at the minimum torque stimuli, the plant will oscillate indefinitely. For such unstable behavior, a control gain is required to be applied. In Section 4.2.2.1 this control gain is designed and explained.

4.2.2.1. Proportional control

Figure 4.12 shows the closed loop feedback block diagram for PID control with proportional filtering. θ_0 defines the angle reference where the system should reach in and maintain as a target angle. An undamped system has two conjugate pole roots on the imaginary axis of the Real-Imaginary plane, so, in order to add real and negative values to this transfer function a controller should provide a dominant pole with a real part. This is frequently done by using a proportional, integral and derivative (PID) controller gain. PID control reduces the steady state error of the undamped nature and adds complex roots to translate the response from undamped to underdamped. In Section 4.3, a PID controller is processed for the free surface flow case.

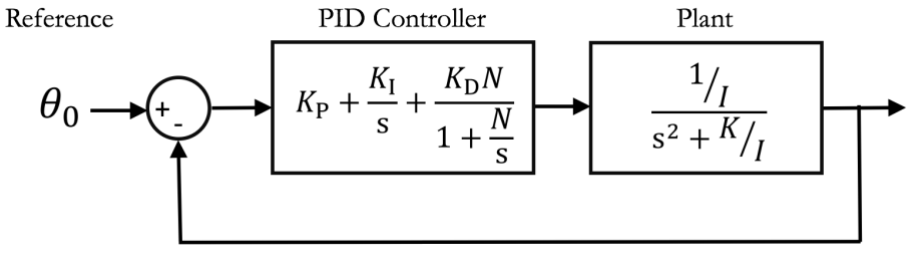
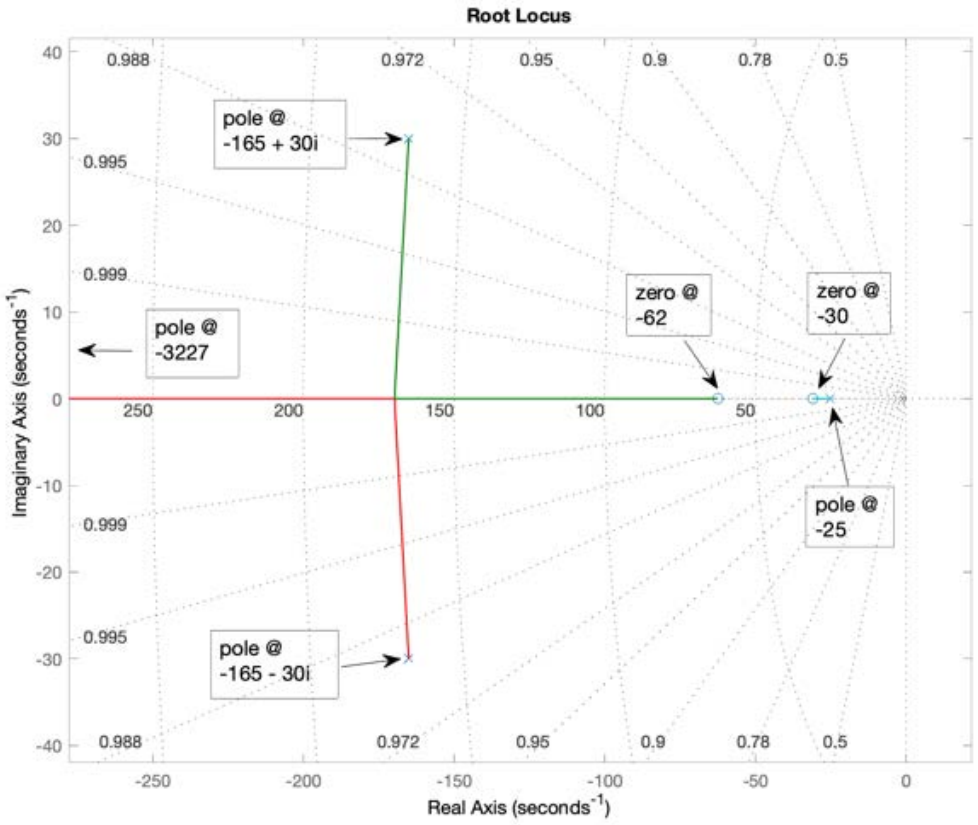


Figure 4.12. Proportional closed loop control of a profile rotation SOS with undamped plant response.

K_P, K_I, K_D and N in Figure 4.12, are the PID constants of the PID controller and N the filter coefficient, and the proposed values for constant values of the plant in Figure 4.12 come

from the testing values used by Woods et *al.* in [86]: $I = 1.73x10^{-4}\ \mathrm{kg \cdot m^2}$ and $K = 0.3832\ \mathrm{N \cdot m/rad}$. It has been used the Matlab® Control System Toolbox to tune the PID constants that turn the undamped oscillatory respond of the plant into an overdamped respond. Figure 4.13 provides the Matlab® result of tuning PID including the root values in the complex plane and the transient time respond.



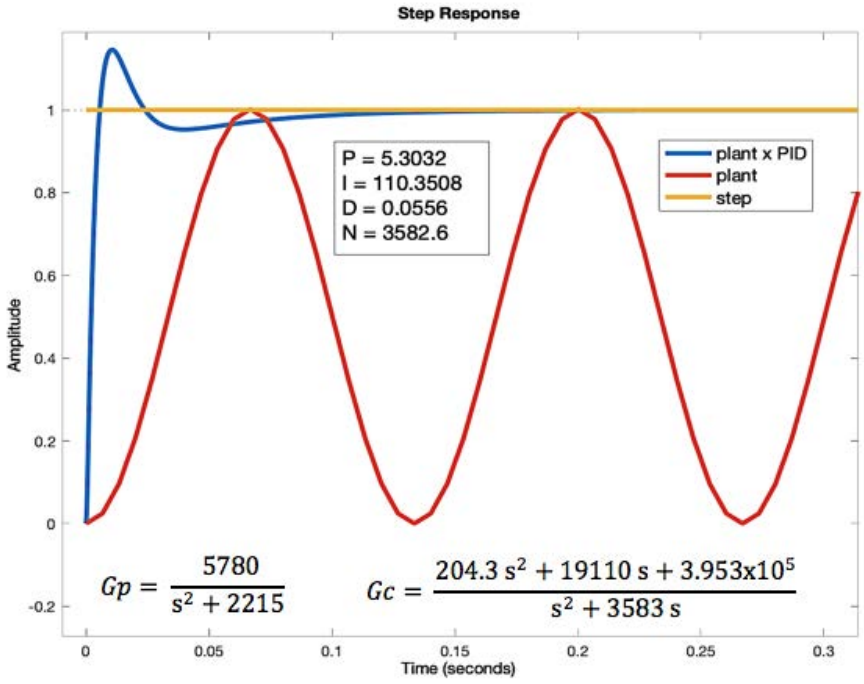


Figure 4.13. (Top) Poles and zeros in the complex plane. (Bottom) Time respond of the PID control system.

The requested respond is overdamped with little overshoot. So far, a usable respond for an undamped rotational system of the profile. In next section, the discussion of this results are provided.

4.2.2.2. Results and discussion

It is clear how PID can solve the undamped respond. But, so far, a pure rotational mechanic system has been addressed. The system needs to play its role in the requested fluid-solid coupling system. As mentioned earlier, the task of the profile is to control the rotation to follow a reference angle. Fluid in closed channel flows exactly in the same direction than in the benchmark case. In Video [4.2], the running simulation illustrates the time performance of the controlled profile. In that test, the user defines the reference angle to 180 degrees, and the torque produced by the particles net force. It is clear the efficiency of the applied control to the profile. In the following Figure 4.14, a sequence of three sample image from Video [4.2] is presented. The sequence shows the effect of the control action avoiding undamped oscillation. The running simulation includes 12,000 fluid particles, real-time calculation of the angular step, angular velocity, torque and reference angle to follow.

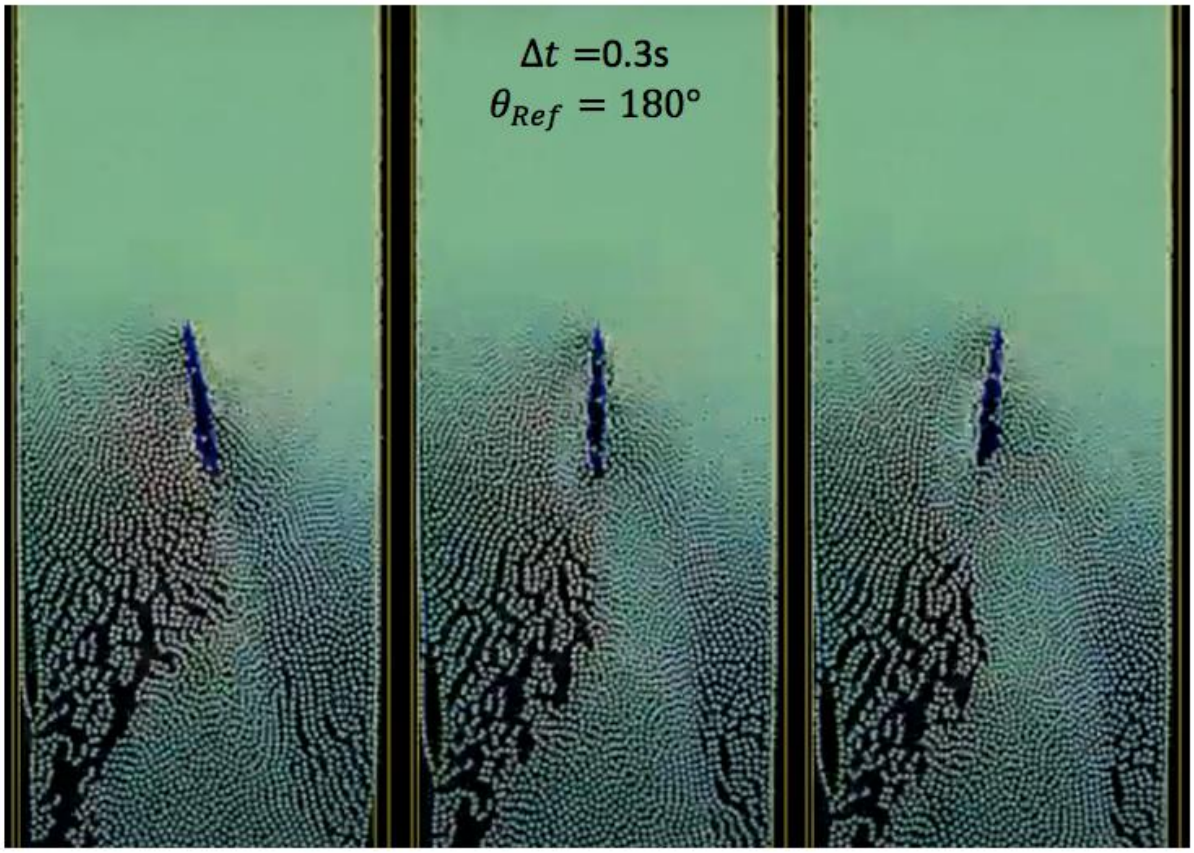


Figure 4.14. Three sample image sequence of running simulation of the controlled profile using a reference angle of 180°.

A constant upstream velocity of the flow is provided over time, however, a time performance test for the suggested controller could be a variation of the upstream velocity to find the flow velocity limit where the profile to cannot follow the reference angle anymore by the effect of a maximum torque. The parametrization of the plant, the controller and the particle engine suggests that with fair modification of the any of these values, the system will remain valid for control purposes.

4.3. Pouring case

SPH model has been widely used to simulate dam brake cases [87] and other free surface flows [88]. As well as in interactive simulations where solid body collision forces act to shape free surface water volumes [89]. Now that closed channel flow has been tested and proved useful for

the two initial cases, the goal in this section case is to simulate a free surface fluid with pouring kinematics control where the final pouring curvature of fluid flow can be modelled and estimate for the application of liquid transfer between vessels. A mathematical coupled model of the pouring kinematics is provided and a new approach of tilting is introduced.

By considering a liquid in hydrostatic equilibrium with gravity-orthogonal free surface, a discharge of this fluid in open space can be analogously modelled as a free fall of solid body motion dynamics for stationary conditions [87]. However, this is valid if outflow velocity remains in a laminar regime range. Case (a), in Figure 4.15, illustrates the effect of free fall fluid motion with predictable curvature over time. Vessel 1, the pouring vessel, has one DoF of rotation on the z-axis, while vessel 2, the catching vessel, has one translation DoF along x-axis.

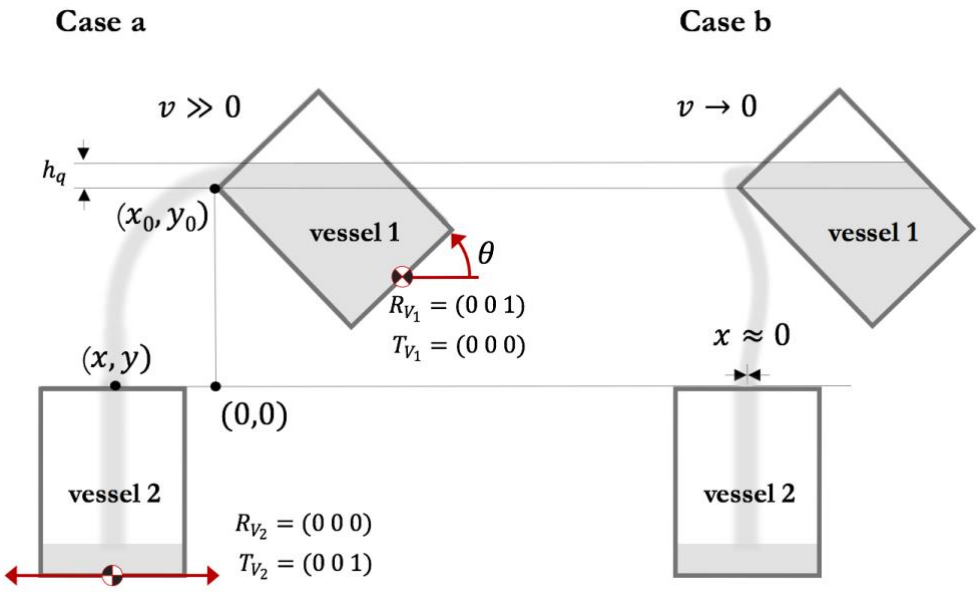


Figure 4.15. (Left) Case a: Expected laminar flow with free fall trajectory slope. (Right) Case b: Unexpected no-slope turbulent fall trajectory.

In contrast, unpredictable fluid curvature (trajectory) will happen in a free surface undergoing motion that is not orthogonal to the gravity vector. Case (b) in Figure 4.15 illustrates such curvature. Even when initial conditions in hydrostatic equilibrium are met, turbulent outflow may occur during pouring at particularly low flow velocity, v, values. The most typical causes of turbulence at low v's are the border pattern or shape of the vessel and the acting surface tension force.

Solid body and fluid motion models can merged together and controlled with a closed loop model to effortless transfer liquid volume from one vessel to the other at a constant outflow velocity using free fall parabolic trajectories [90] (Figure 4.15, Case a). However, maintaining constant outflow requires a dedicated control especially in the initial phase of pouring where flow is inherently non-stationary. Another aspect to consider is the liquid adhesion to the vessel crest and walls. This issue can be avoided by using a sharp spilling edge and/or guaranteeing a marginal velocity at the outlet crest or edge. In this work, the selection of the vessel type is not meant to be restricted at least from its edge form but only by its interior volume, so, the control action will be focused on the outflow velocity and not in the vessel edge type. This outflow velocity is controlled by maintaining a constant liquid elevation over the vessel's crest, which is especially relevant during the initial surface oscillation (slosh). The interior volume of the vessels, here tested, have planar vertical and semi-vertical walls parallel to the y-plane. Such type of vessels are common in domestic/unconstrained scenarios. Also, the vessel edge is considered not to add significant level suppression on the outflow curvature, a dominant parabolic based discharge (sharp crested) over a coefficient based discharge (broad crested).

4.3.1. Free surface fluid in free fall

The relevant parameters of pouring liquid in free surface fluids are the pouring velocity and the height of the pouring liquid over the vessel edge (h_q in Figure 4.15). The faster the pouring is, the faster SPH needs to estimate fluid data at a shorter timestep. But also, too low h_q will compromise the outflow trajectory due to fluid viscosity [91]. There is also a subcritical regime to take care of when height increases drastically in no-linear slopes and then excite possible hydraulic jumps [92]. Thus, a minimal value of h_q must be studied based on the minimal expression of the fluid in the SPH lagrangian method; and, this is, the particle itself.

Particle have a range of influence. That range is the smoothing length in SPH, h in Equation (3.7), defined by a spatial range where only neighbor particles within that range influence the given particle's dynamics. By defining the minimum size of h_q equaled to the size of the smoothing length h, the particles are allowed to carry, along the outflow, the added forces of its nearby particles. This diminishes the sources of turbulence and provides added forces to

the outflow every time step of the fluid outflow. In the simulation of this work, a scaled smoothing length of 10 mm is used, and a radius of interaction of 5mm as the minimum value for h_q. This values are set constant throughout the entire simulation.

Out of vessel 1, the characteristic dimension of the liquid jet ($\sim h_q$) is generally small so that neither boundary conditions nor fluid intrinsic forces has great contribution in open space free fall. As mentioned earlier, a sharp crested parabolic discharge. Under these assumption fluid flow of particles will describe a quadratic trajectory defined by Equations (4.17) and (4.18).

$$\mathbf{y}(t) = \mathbf{y_0}(t) - \frac{1}{2}\mathbf{g}t^2 \tag{4.17}$$

$$\mathbf{x}(t) = \mathbf{x_0}(t) + \mathbf{v_{o_x}}t \tag{4.18}$$

In Equation (4.17), $\mathbf{g}$ is the gravity vector force [0 9.81 0], the initial flow velocity $\mathbf{v_{o_y}}$ in y-axis equaled to zero, the initial flow position $\mathbf{y_0}$ that gradually changes in time, and vessel 2, the catching vessel, at $\mathbf{y} = 0$ position. For coordinate axis reference please refer to Figure 4.15. Time t_y can be calculated at the time the fluid particles enter vessel 2. Assuming a null horizontal acceleration ($\mathbf{a_x} = 0$), Equation (4.18) is reduced to $\mathbf{x_{catch}} = \mathbf{x}(\mathbf{y} = 0) = \mathbf{v_q}t_y$; where $\mathbf{v_{o_x}} = \mathbf{v_q}$ is the bulk velocity right at the vessel outlet. This velocity can be determined analytically as $v_q = \frac{2}{3}C_d\sqrt{2\mathbf{g}h_q}$ for an open channel flow, where C_d is an empirical discharge coefficient (~0.6 approx.). However, $\mathbf{v_q}$ is rather obtained by averaging SPH particle velocities in the outflow plane at each timestep. So $\mathbf{x_{catch}}$, in Equation (4.18), is computed to calculate the ending x-axis position at which vessel 2 will be translated to.

This pouring model based on free fall of particles, is valid for a pouring stage with constant outflow, in this work named as the constant flow stage (CFS) for further reference. However, the initial pouring stage imposes an increasing outflow at the vessel edge, and, thus, an incremental acceleration in the x-axis of the flow. So, here is why it is required to consider with much care an initial pouring stage. The approaching flow stage (AFS) is described as the initial pouring stage in this work. AFS happens when vessel 1 initiates the tilting to reach the window level h_q, see Figure 4.16. From this moment, CFS takes place to conclude the pouring task. In next Section 4.3.2, the modelling and control solution of AFS is described.

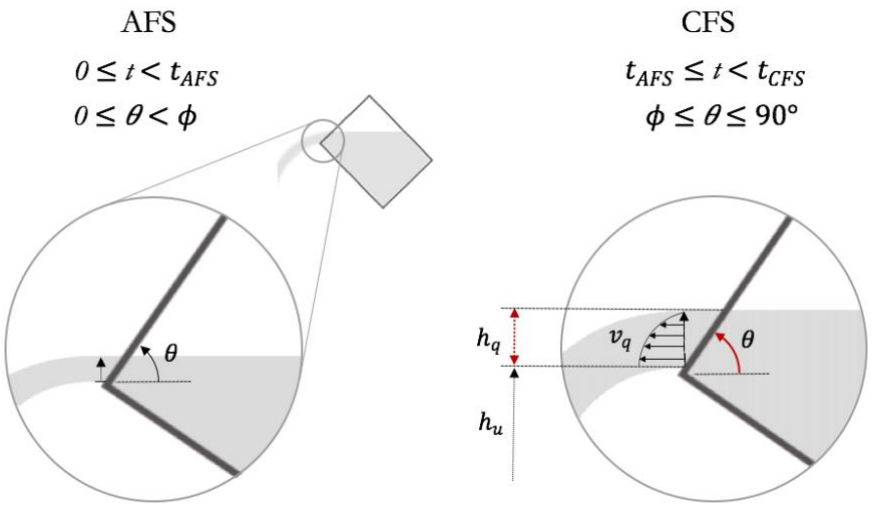


Figure 4.16. Stages of pouring. (Left) AFS going from the initial time to t_{AFS}. (Right) The subsequent CFS going from time t_{AFS} to t_{CFS}.

CFS aims to pour the largest amount of liquid volume because constant outflow can be processed with less discrete error than in an outflow with variable outflow velocity, like in AFS. A volume limit to pour is set so the largest amount of liquid to pour remained in CFS rather than in AFS. To achieve this, a window level h_q should be as short as possible, and just to clear the SPH smoothing length tolerance mentioned before. Tilting vessel 1 can be conducted by controlling the angle θ of vessel 1 as a proportional function of the linear decrease of h_q in time (Δh_q).

$$\theta(t) = arctan\left(\frac{2}{B}(h_q(t) + h_e)\right) \tag{4.19}$$

Equation (4.19), the trigonometric relation of the angle and dimensions of vessel 1, can be used to reach θ. However, this will not offer a linear step on the wanted center of rotation, so, instead, as Figure 4.17 illustrates, a Δh_q is first measured at certain timestep during CFS, then this height is assumed as the magnitude of a vertical vector (vector $\mathbf{b}$) with origin coordinates on the pouring edge of vessel 1. Then find vector $\mathbf{c}$ from vector difference of $\mathbf{a}$ and $\mathbf{b}$. Vector $\mathbf{a}$ is the vector from the center of rotation to the pouring edge in vessel 1. Finally, an angle θ' between the vectors $\mathbf{a}$ and $\mathbf{c}$ is calculated with the scalar product, and vessel 1 tilts at that angle in constant angular steps.

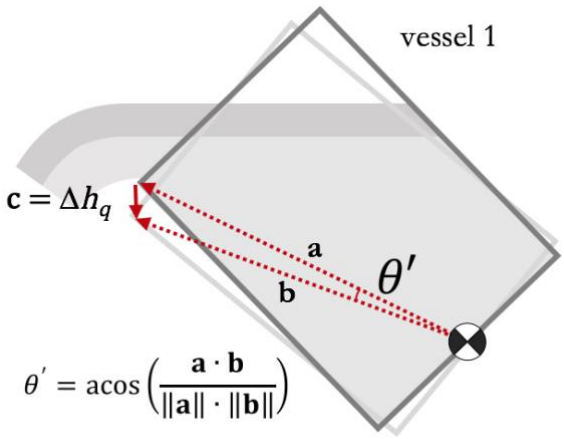


Figure 4.17. Vertical step of tilting in CFS.

In simulations of this work, the pouring of the full liquid volume is developed. No pouring of partial volumes is intended to be developed. So, as the liquid volume ceases, h_q will drop down. The free fall curvature has no flaws over the course of the CFS end. For that ending flow, there has been tried different tilting functions to handle the angle of pouring, not only as a function of h_q but also as a function of time with relatively the same positive results. Bauer in [92] implement the free fall motion of fluids as a measuring device for free surface flow. In further work, partial pouring will require to drive the outflow velocity slowdown and the free surface back elevation derive from tilting back action.

During CFS, the catching motion of vessel 2 will be simplified to translate the vessel on x-axis component of the particle flow curvature. It is not required to offer a special treatment to the catching task, but only to limit large translation steps during filling up in order to prevent sloshing and eventual spill. In the simulations of this work, the update of solids positions take place each particle's position update (timestep of 0.002 seconds). A fairly small discrete spatial step for a smoothed motion of vessel 2.

In Figure 4.18, a simile of the outflow trajectory is presented between the analytical solution expressed in Equations (4.17) and (4.18) (red dots) and the result of the SPH particle flow (grey color particles). As seen in Figure 4.18, analytical curves are fairly followed by the outflow particles at certain v. The fall of particles at x-axis position along all the y-axis positions has been validated. This allows to confirm a quadratic outflow curvature of the SPH method in free surface using our simulation context, and also to confirm how to determine and track an outflow target during CFS. For all the following test, the following 2D vessel's dimensions are

used: width=height=0.191 meters. If not indicated different, please assume a liquid level height of h=0.1591 meters. The scales in x and y-axis in Figure 4.18 are designed in Matlab® and only adapted to illustrate dimensioning.

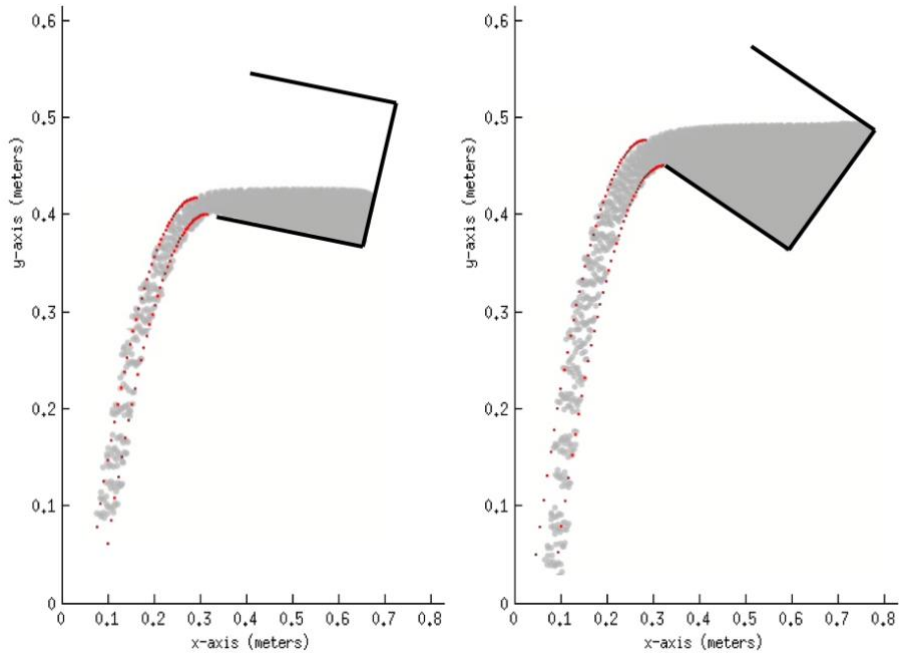


Figure 4.18. SPH outflows (grey particles) vs. analytic curvatures (red dots). (Left) a narrow outflow. (Right) a broad outflow.

4.3.2. Fluid-solid coupling model

For high values of v_q, the outflow follows a free-falling motion, but for low values of v_q the flow may be attached to the external vessel walls, causing the fluid to drip through these walls, case b in Figure 4.15 illustrates this effect. Other factors that may cause this effect is the shape of the vessel tip (Coanda effect) and the air-fluid boundary conditions. However, most of these factors can be prevented by guaranteeing a minimum value of v_q at the vessel edge.

To overcome this issue of dripping and considering that a proper pouring task should transfer the entire volume with no fluid leakage, inspired by human intuitive pouring action, a Slosh Induction Method (SIM) is proposed during AFS. SIM consists in generating an anti-symmetrical oscillation of the fluid free surface such that the tilting angle θ of vessel 1 produces an equal angle θ_1 of the free surface at certain liquid level condition. This condition defines the

usage of SIM and is set when the liquid level h_e is equal or lower than the window level h_q at an arbitrary angle of $\theta = 20°$. Figure 4.19 diagram illustrates this SIM condition. Video [4.3] provides the simulation of the free surface oscillation.

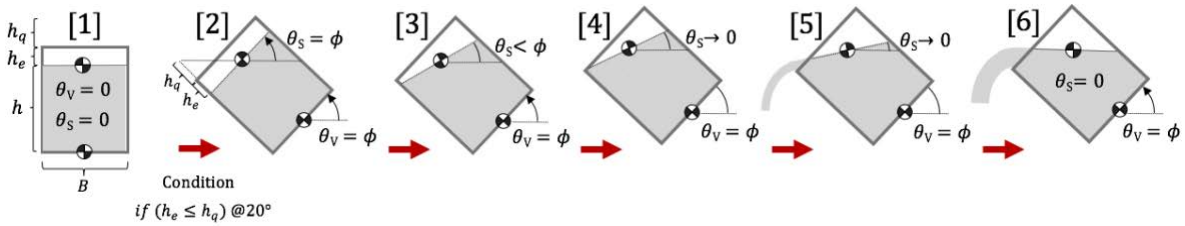


Figure 4.19. Transitions of AFS with SIM. Transition 1 defines initial conditions in hydrostatic equilibrium where the angle of the pouring vessel and the angle of the free surface are zero. Transitions 2 to 6 define the SIM steps at condition ($h_e \leq h_q$@ 20°).

From the moment of rotation of θ and till θ reaches a conditional value ϕ, SIM should not develop changes in the liquid volume shape. Leaving a free surface inclination of ϕ degrees with respect to the horizontal and retaining a linear model for the slosh. Transitions 1-2 in Figure 4.19 illustrate this motion. After θ_1 and θ become ϕ, free surface will freely return to the horizontal, see Figure 4.19 transitions 3-6, allowing the slosh to travel back to the opposite pouring vessel wall and run out of the vessel with a velocity closed to the fundamental velocity of the induced slosh. This is the methodology of SIM.

There is no need to process AFS control and CFS can be initiated when the liquid level condition is not met. In that case, the outflow will follow free fall motion starting from $0°$ and to the ending angle ($\theta > 90°$), the x-axis position of the fluid fall is predictable since the x-component of the outflow velocity (v_q) is known by the SPH processing. This is the advantage of having SPH method used as the fluid model. Testing walls has been placed in the simulator to measure flow velocities of particles at different locations where needed as well as two particle indicators of the free surface angle θ_1. Two fake particles colored in red are also added to measure the angle of the fluid free surface. Figure 4.20 shows these measurement implementations.

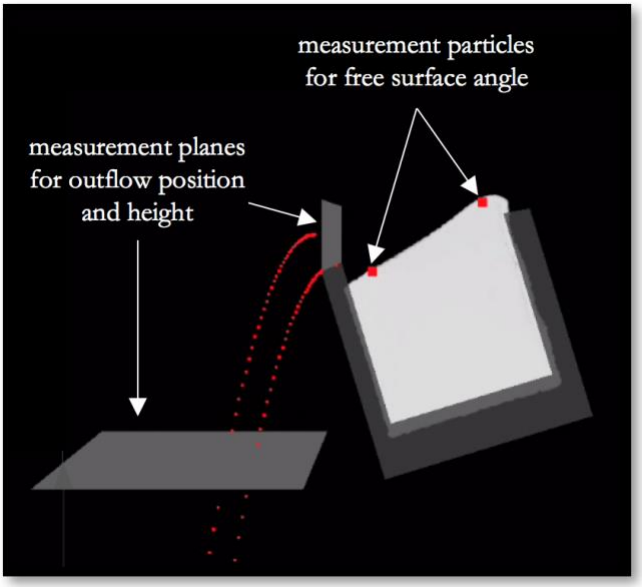


Figure 4.20. Virtual collision planes for flow measurement.

Lee, in [93] (p. 138), studied the moment of inertia of liquid in a vessel, and demonstrate that, having a symmetrical fluid area, the moment of inertia is small, and, in the same way, the portion of fluid that rotates together with the vessel becomes small too. So, as the moment of inertia dictates, with no SIM usage, the location of the center of rotation should be at the center of mass making the moment of inertia closed to zero and retaining the free surface orthogonal to the gravity vector. In contrast, AFS with SIM uses the z-axis as the axis of rotation located at the middle bottom of vessel 1. This translation of the center of rotation increases the moment of inertia to allow the free surface to elevate along the vessel bottom plane. See Figure 4.19, transition 2 for reference.

In this work AFS without SIM is not going to be studied, however, there would be two steps required to simulate such case: (1) find the center of mass of the fluid volume to locate the center of rotation, and (2), rotate the vessel over this center, and reach the proposed outflow height h_q. Recall that in such case, different scenarios could be found, like the scenario of having a low volume to pour with respect to the total vessel volume. Thus, CFS would not begin with the proposed initial condition of h_q. This is important to remark because the selection of h_q must be always as short as possible (equal or greater than the smoothing length h of SPH to allow simulation comparisons) specially at low pouring volumes. In Video [4.4] a running simulation performs AFS without SIM.

This reasoning of AFS needs to be declared in a mathematical approach such that a slosh control can be applied to a motion kinematics. One mathematical approach used is the solution of the boundary condition equations from the velocity potential of the flow [91]. This solution will lead to a nonlinear slosh formulation that will require high computational cost, even for robotic elements with a single DoF. Instead of this approach, a linear model is preferred and planned in different modelling schemes [94-96]. This linear model proves physical equivalence for small oscillation amplitudes of the slosh. The slosh in free surface is described as a finite number of oscillators using poorly damped second order pendulums that dissipate energy according to their order. Each pendulum represents each oscillation component of the surface slosh. In the pouring case, since the only variable required is the velocity of the fundamental component of the slosh during AFS, this linear model seem to be appropriate. In the graph of [93] (pp. 4-135), the fundamental component of the slosh contributes to more than the 80% of the total energy of the slosh, and only less than 20% to the rest components, so the processing of this 20% will not add a significant intensity to the slosh velocity, having only a one-pendulum task to process. Further details on this model can be found in [94].

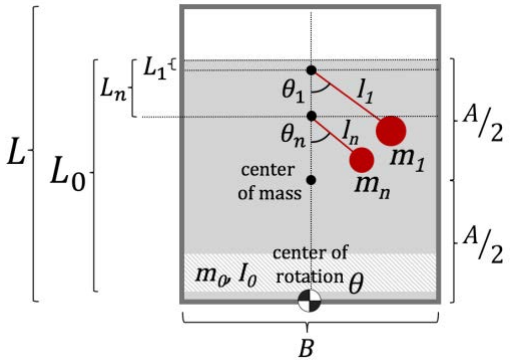


Figure 4.21. Pendulum model of the slosh including n^{th} oscillations of the free surface.

Figure 4.21 diagram describes the employed model with full pendulum components, where m_0, L_0, and I_0 describe the mass, the height and the moment of inertia of the solid mass; m_n, L_n, l_n and θ_n are the mass, height, distance and angle of the n^{th} pendulum component; being $n = 1$ the fundamental pendulum component index and $n > 1$ the rest pendulum component indexes. A/L and B are the height and width of vessel 1. g is the gravity constant. Notice that θ is the tilting angle of vessel 1 while θ_n represents the rotation angles of the slosh components represented by n pendulums. By solving Lagrange's equation for lateral and rotational excitations, Ibrahim, in [94], provides a slosh equation indicated in Equation (4.20). Notice in this equation

the independence of the moment of inertia, I_0, for rectangular and cylindrical vessels having only dependence of three state variables, x, θ and θ_n, everyone as function of time (t).

$$\sum_{n=1}^{\infty} m_n l_n \left[\left(\frac{A}{2} - L_n - l_n\right)\ddot{\theta} + l_n\ddot{\theta_n} + \ddot{x}\right] + \sum_{n=1}^{\infty} m_n g l_n(\theta + \theta_n) = 0 \qquad (4.20)$$

Notice in Equation (4.20) that θ and θ_n have a linear relation. This linearity is typically reached by approaching the sine function of the native pendulum equation to only the angle $(sin\ \theta \approx \theta)$ for small angular displacements. Also, notice that Equation (4.20) lacks angular velocity, $\dot{\theta}_n$. This corresponds to an undamped time response of the slosh. If our interest is only the fundamental oscillation of the slosh and no lateral excitation will be needed $(\ddot{x} = 0)$, then, making $n = 1$ and cancelling the lateral motion x, the slosh Equation (4.20) is simplified to,

$$-l_1\ddot{\theta}_1 - g\theta_1 = \left(\frac{A}{2} - L_1 - l_1\right)\ddot{\theta} + g\theta \qquad (4.21)$$

From Equation (4.21) and using [94], if the pendulum of interest is at index $n = 1$, the distances l_n and L_n are calculated with Equation (4.22) and (4.23).

$$l_1 = \frac{L}{\pi}\coth\left(\frac{\pi A}{L}\right) \qquad (4.22)$$

$$L_1 = -\left[l_1 + \frac{A}{2} - \left(\frac{2L}{\pi}\tanh\left(\frac{\pi A}{L}\right)\right)\right] \qquad (4.23)$$

The timeline designed in this work, where the angles θ and θ_1 relate each other in a AFS time response with SIM, is presented in Figure 4.22. t_0 is the period of time where a partial sinusoidal input signal of $\theta(t)$ is applied to vessel 1 allowing the free surface angle θ_1 to elevate at the same rotational velocity as θ and both reaching ϕ angle value. This period of time corresponds to t_0 in Figure 4.22. In the next period, t_1, θ_1 returns to zero by the effect of gravity while θ remains at that initial conditional angle ϕ. As soon as θ_1 yields to zero, t_2 starts the control action of the slosh. The goal of this control action is to turn the surface elevation to zero avoiding the pure undamped nature of the slosh. Recall that the pendulum model of the slosh and its oscillation response will take place if no control action is applied. The optimum control action would lead θ_1 to oscillate close to zero as fast as possible so the outflow velocity is set at a constant h_q that will migrate to CFS with a stable flow and constant pouring. At the beginning of

t_3, where the returning oscillation of the slosh devices free exit to flow out of vessel 1, AFS will end and CFS will begin.

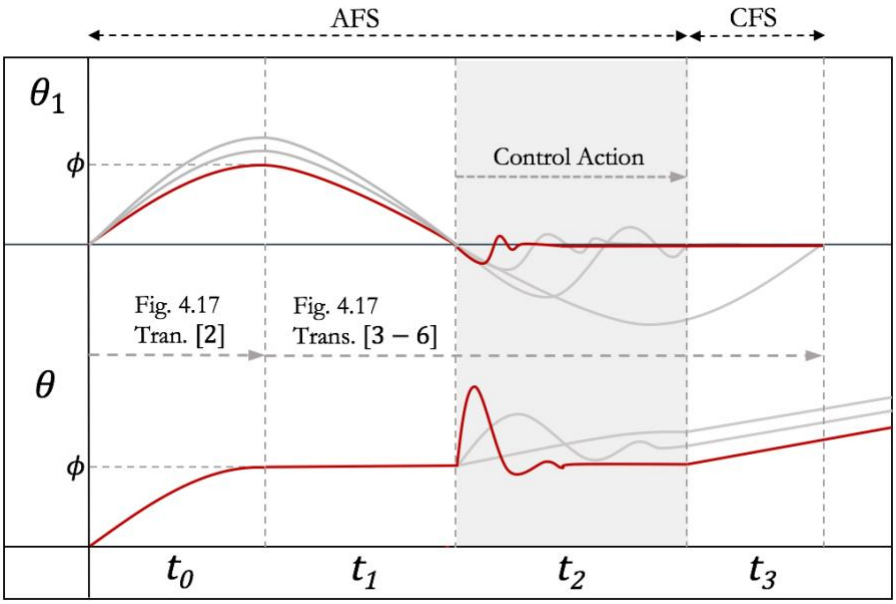


Figure 4.22. Proposed timeline of SIM pouring. AFS ranges from the initial tilting to the end of the control action. CFS initiates at the end of AFS and continues till the end of pouring.

A control action is proposed based on a proportional-integral-derivative (PID) controller. A system transfer function (STF) is given by Equation (4.24) as the plant gain (Gp). This Gp comes from Equation (4.21) where the input and output of STF are θ and θ_1 respectively. The PID control is capable to remove a steady state error from Gp which in Equation (4.24) is one half. From this equation, it is deducted that roots only exist in the imaginary plane having an undamped time response, so, a trained proportional-integral-derivative (PID) controller/compensator, in closed loop with Gp, can add the additional zeros to dissolve the pure undamped system and turn it into a stable damped one. Using the root locus tuning tool in Matlab®, the PID constants can be easily determined to reach a desired damped response. STF equation depends on the vessel dimensions and liquid height, follow Equations (4.22) and (4.23), so any further experimentation would have different PID constants at different vessel dimensions. The selected values of dimensions in this work, only provides a reference for simulation and model testing. Nonetheless, the system will have the same response at different dimensioning. For the sample simulation, Gp is populated with these values: $A = B = 0.159$, $L_1 = 0.0207$ and $l_1 = 0.0615$ meters.

$$STF(\mathrm{s}) = G_p = \frac{\theta_1(\mathrm{s})}{\theta(\mathrm{s})} = \frac{\frac{b_2}{l1}\mathrm{s}^2+\frac{g}{l1}}{\mathrm{s}^2+\frac{g}{l1}} = \frac{0.6324\mathrm{s}^2+159.51}{\mathrm{s}^2+159.51} \tag{4.24}$$

$$G_c = K_\mathrm{P} + \frac{K_\mathrm{I}}{\mathrm{s}} + \frac{K_\mathrm{D}N}{1+\frac{N}{\mathrm{s}}} \xrightarrow{N=10^7} \frac{K_\mathrm{D}\mathrm{s}^2+K_\mathrm{P}\mathrm{s}+K_\mathrm{I}}{\mathrm{s}} = \frac{0.22797\mathrm{s}^2+0.760269\mathrm{s}+20.96905}{\mathrm{s}} \tag{4.25}$$

The controller gain, Gc, in Equation (4.25), described by the PID control in AFS has the same block diagram as the one used in the profile rotation case (Figure 4.12) but cancelling the derivative filter by making $N = 10^7 \cong \infty$. This controller adds two zeros and one pole to the second order Gp. The idea behind this PID controller is to add a dominant pole to the closed loop system so this will become stable-damped and with minimum steady state error. The new added zeros will affect the amplitude gain of the system, but not the nature of the response though.

4.3.3. Pouring action

Because forces, velocities and positions of particles and solid bodies are about to be controlled, speed calibration of the system is an important issue worth testing to secure the real-time performance. The sync of the main simulation loop time with the real fluid speed is crucial. This time adjustment is conducted by setting a number of calls of the system loop that refresh fluid particles so that the simulation of the slosh frequency approximates to its theoretical slosh frequency in horizontal motion.

An analytical solution to the slosh frequency of the first oscillation component for squared vessel can be found in [95] (p. 59), and it is shown in Equation (4.26), where $Freq$ is the slosh frequency in hertz of a free surface elevation cycle during horizontal motion, the vessel width B and the height of fluid at rest A.

$$Freq = \frac{1}{2\pi}\sqrt{\frac{g\pi}{B}\tanh\left(\frac{A\pi}{B}\right)} \tag{4.26}$$

Figure 4.23 shows the values of $Freq$ for a complete slosh cycle at three values of A. Equation (4.26) has been used as reference to set the main loop system time and to find the equivalent time values for the experimental simulations.

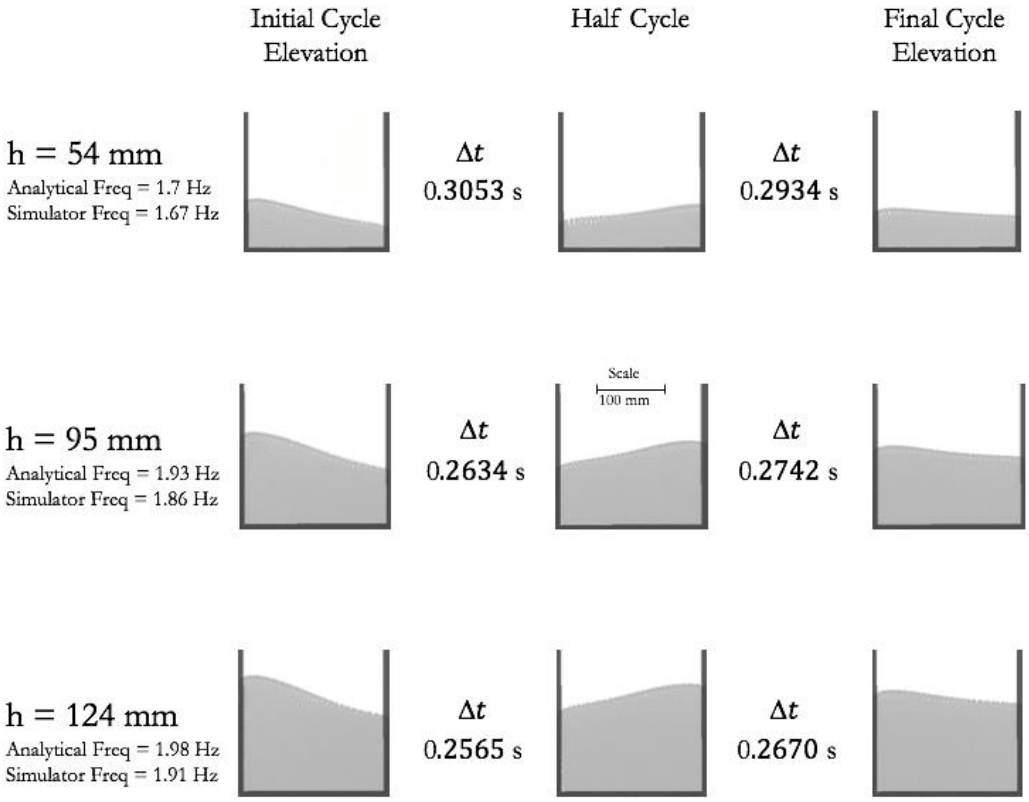


Figure 4.23. Free surface elevation for timing calibration.

Notice that the difference between analytical and simulated values of $Freq$ barely diverges when A increases. The results, illustrated in Figure 4.23, confirm a maximum value of Δt to run real-time simulations. However, it is clear that having a smaller Δt value, a more accurate result will be found. Video [4.5] illustrates lateral excitation of the slosh in a running simulation from where Figure 4.23 results are obtained.

Now that the fluid time respond of the SPH slosh is equivalent to the analytical formulation in Equation (4.26), it is time to apply the proposed PID controller to the CLF system. By simple inspection in the product of Equation (4.24) and (4.25), $GpGc$, zeros and poles are added to the closed loop system. As mentioned early, the location of the roots defines the stability and time response, so the controller Gc adds a dominant pole with 0.00 value in the real axis to minimize the undamped nature of the system. Two additional zeros are added by Gc so

the gain is modulated to reach certain threshold. In Figure 4.24, the roots of the plant STF and the CLF system are plotted in complex plane.

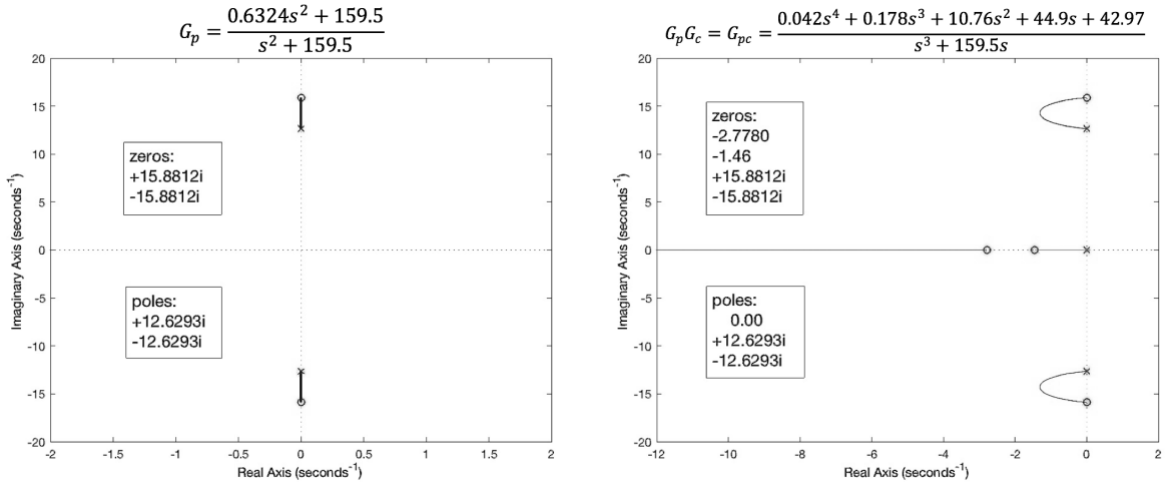


Figure 4.24. (Left) Root locus plots of the plant gain Gp. (Right) The plant-controller gain $GpGc$ (right).

In Figure 4.25, two step time-response graphs are shown; the one on top, shows the one from the Gp in closed loop and its steady state error; and the one at the bottom, the product of $GpGc$ in closed loop using the PID controller. Notice that the steady state error in Gp is significantly removed in $GpGc$. Also, the bias undamped nature of Gp is corrected by having a damped time respond that if, not ideal, it suppresses in a quarter of its magnitude.

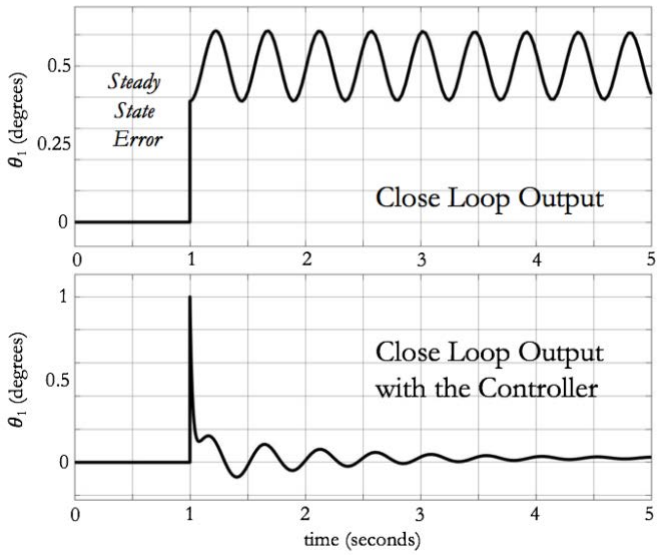


Figure 4.25. Step transient response of $GpGc$.

Figure 4.26 shows the rotational motion traces of θ and θ_1 captured by the running simulation. The full kinematics of angles is described including the two stages of motion (AFS with SIM and CFS) and the proposed four time lapses. As expected, θ_1 will oscillate momentarily, but it will settle down to a stable state as a consequence of the control action applied to θ. In Video [4.6], full kinematics of the pouring and catching motion is illustrated. Considering the outcome of these results, the proposed SIM-based pouring action is a useful model to implemented. Modelling and implemention has been developed and proven equivalent.

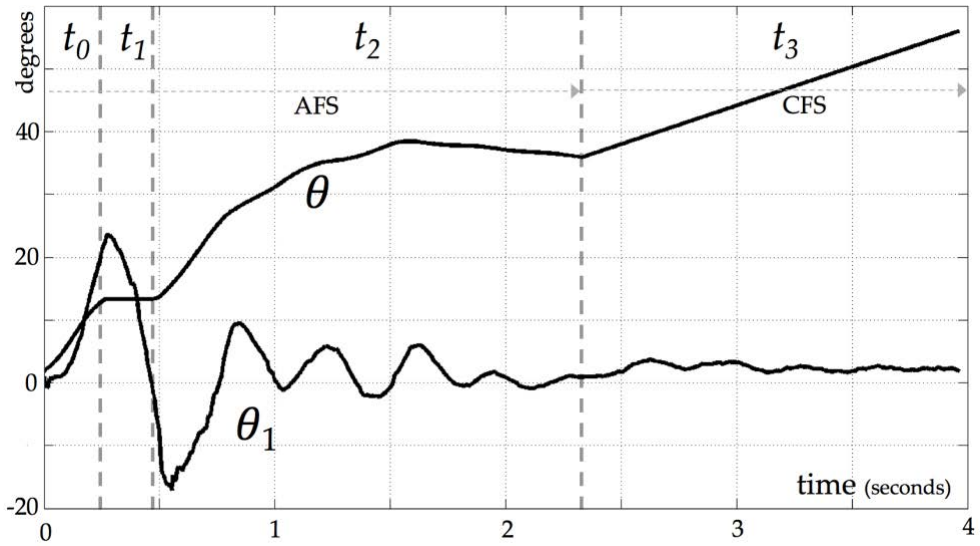


Figure 4.26. Timeline of Pouring with SIM Control. The trace of θ to be used by an actuator.

The final idea of a running simulation with a real-time interactive SPH flow has helped to redesign the model by visual feedback rather than in an off-line batched process.

4.3.4. Results and discussion

In the cases where SIM is not required, a tilting velocity needs to be constant and the center of rotation equaled to the center of mass so the fluid's moment of inertia can be cancelled to avoid sloshing. However, tilting velocities during implementation are not perfect due to particle distribution in space and slosh elevation would occur. For such reason, a test has been performed to find the valid range of tilting velocities in this simulator. Figure 4.27 shows the maximum finding of tilting velocities in AFS without SIM. For a better illustration of rotation on

the center of mass, Video [4.7] shows a running simulation where poor elevation occurs in this center of rotation.

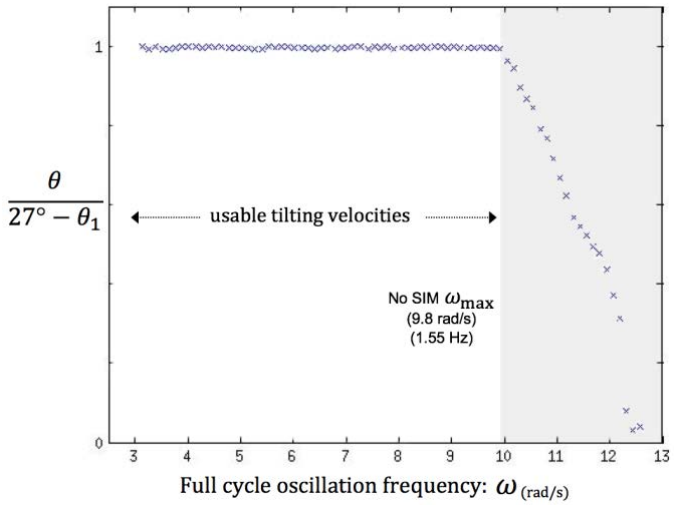


Figure 4.27. Usable full cycle tilting velocities ω in the case of AFS without SIM.

The equivalence of the pendulum model for the fluid free surface can be compared with the tilting angle symmetry in running simulations. For such comparison, Figure 4.28 illustrates the transient respond of angle θ during the four periods of time described in Figure 4.19.

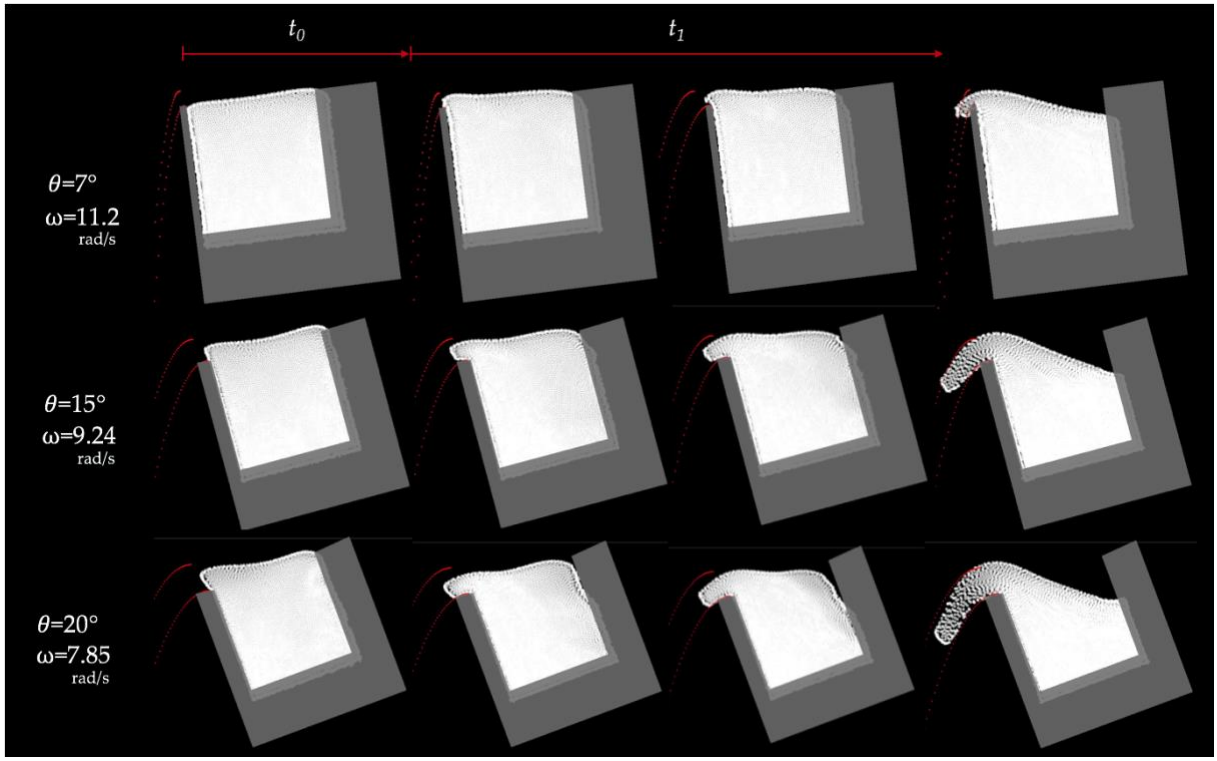


Figure 4.28. Transient time response of the slosh induction method using different vessel angles of rotation (7°, 15° and 20°) and at the end of t_0 and t_1.

It is remarkable to notice the elevation of the slosh as required by SIM in the first (top) figure, column t_0. A relaxation of the slosh (θ_1) occurs during the second, third and fourth figure columns where the back elevation diminishes to find the horizontal and induce the pouring. By increasing the angle θ a decrease in the angular velocity is required. This means that there is a valid range of θ for the transition of t_0 to t_1, but the optimal angle value ϕ, according to our approach, is the one obtained by Equation (4.19) using the smoothing length as h_q.

As described in Figure 4.29, simulations have demonstrated the feasibility of AFS with SIM. A running simulation can be found in Video [4.6]. In that video, an evident improvement of the free surface control can be appreciated compared with the CFS pouring in Video [4.7].

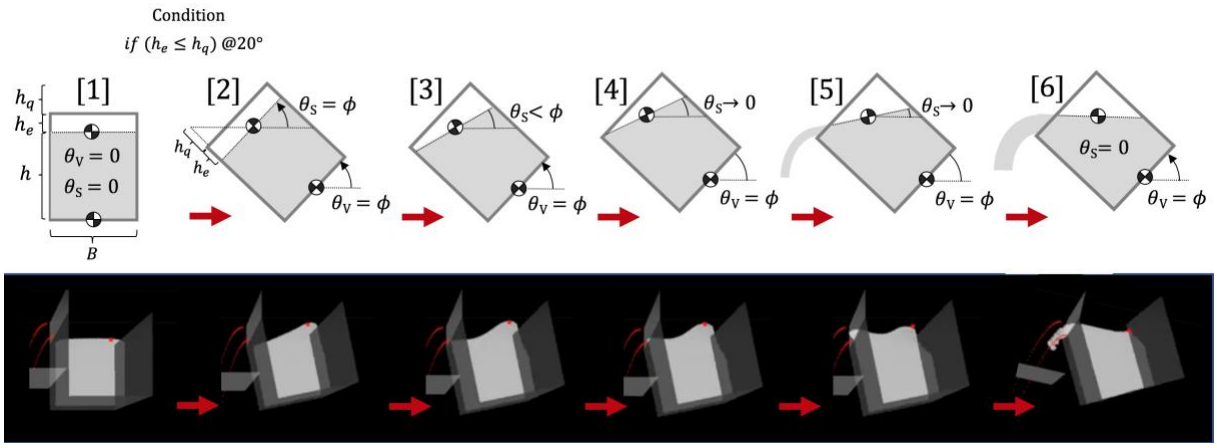


Figure 4.29. Comparison between proposed kinematics vs. simulation kinematics in AFS with SIM.

As mentioned earlier, the benchmark case and the profile case constitute the basis that tests the fluid properties, the collision scheme, the computational time performance and the coupling model for the final target case: the liquid pouring. SPH for free surface has been studied for many years, but applications are yet to come. In this work, a simplified fluid-robot model description has been studied and implemented using scaled dimensioning sample application of pouring liquids between vessels. Such model description can be useful to directly apply this kinematics to robot manipulators.

GPU computing has added computation value to the presented simulations in a way that the experiment redesign can be performed in real-time interaction to overcome the training in

offline batches. The presented model kinematics of pouring can only be designed in scale for different experiment dimensions. A 3D model of the fluid would be desirable but not necessary for the linear model of the slosh where flow paths run perpendicular to the z-axis and never diverge from the 2D motion.

A remarkable contribution in the pouring case is the real-time performance of the coupling model, where the AFS+CFS scheme including SIM and a single PID controller will provide full control of pouring when the fill heights are critic due to the adhesion forces which, although, those are not implemented to the fluid model, they are always present in physics. It is also important to emphasize the property of the model to conduct blind motion kinematics, where no complex sensing systems are required for feedback. This feature allows to be implemented in cross-platforms where the minimum requirement is the GPU capability. By adding filling height and vessel dimensions, one can find the time respond using the suggested model and furtherly feed a robot manipulator to perform a successful full non-spill pouring action.

Chapter 5

Conclusions and further work

The implementation of fluid simulations involves many considerations and constrains to the discrete model that may yield to considering twice its application to robotics. This can be evidenced by the limited number of research publications found in robotics journals. However, the application of robotics into the fluids are unquestionable. In the following Section 5.1, a broad conclusion to this work is provided, and in Section 5.2 the proposed further work is exposed.

5.1. Conclusions

In this work organization, the technical conclusions are offered on each section according to the achievements of these thesis objectives (Section 4.1.4). However, a general conclusion is provided in the following paragraphs to offer a recap of this thesis work.

The very well documented KVS effect behind a cylinder has been simulated in this work in 2D. The results found the correctness of the simulated model, not only in the qualitative aspect by showing the classical vortex street behind the cylinder, but also in the quantitative aspect by predicting the correct Strouhal number in simulations.

An angle reference PID control over a profile has been successfully implemented in this work as well. This model allowed to transit, from a fluid-solid simulation with static solids, to a fluid-solid simulation with solids in motion. Such change allows to confirm, not only the applied PID scheme, which is frequently used in the undamped response case to avoid induced and unwanted oscillations, but also to confirm the potential and limitations of the particle-solid collision scheme.

Uncoupling models of fluids and robotic elements could require vast number of simulations to find a desired response. But coupled models provide a simpler solution where simulation can focus on demonstrating the model. This is always desired, but not always available. Thankfully, for free surface flows the coupled model has been provided from Ibrahim [94] for lineal sloshing. This model provides a method to implement control based on free surface. With free surface control the application can include the slosh suppression as in many applications of fluid transportation. In this work, a full liquid pouring application between vessels has been successfully implemented. The findings during this simulation allowed to apply conventional control theory and find a straightforward time respond to the model (Figure 4.26). A robust full coupled-model simulation was applied, which allows to develop full liquid pouring in a blind process, that did not require special sensing elements nor knowledge-driven system to be conducted.

The lack of validated coupled fluid-solid models is one of the major issues in the field of robotics to develop more complex or with higher degree of freedom (DoF) robotic systems in the fluid domain. By offering a coupled fluid-robot model descriptors, like the pendulum model in Section 4.3, an important contribution to robotics in contact with fluids is proposed in this work. Such proposal yielded to develop, from scratch, a simulation framework that allows to add solids in motion and control their position and displacement.

GPU programming has facilitated the real-time implementation in this work. A real-time simulator has been implemented with a complex parallel architecture based on CPU/GPU programming that allows to test performance of response in time (Figure 5.4). The user interaction is another main advantage of using GPU real-time programming, that allows the user to perform interaction during time execution. Parallel programming is not an easy topic the study or implement, but certainly, the line to follow for the implementation of the SPH method. It is also worth mentioning that simulators, similar to this work's simulator, employ other supporting libraries like GLUT and OpenGL that requires special understanding and implementation to

build the simulator framework. For this reason, the implementation itself including GPU programming structures (Sections 3.5 and 3.6) is also an important asset in this research.

The two main challenges, described in Section 1.2, were successfully addressed in this work. The first one allowed to find a balance between the features of the SPH model and the complexity of the collision scheme. The boundary condition complexity requires to be as elemental as possible to avoid time lag that would ruin the real-time performance. It is clear, that elevating boundary condition complexity will allow to handle other features of fluid-solid interaction (*e.g.* added surface tension) and to approach the model better to real physics. So, the proposed simulations set the basic requirements to accomplish real-time performance in fluid-solid coupled implementation with a proper runtime simulation that match the proposed specifications. By focusing on the real-time performance, the second challenge of this work has been accomplished. No batch processing was needed to be processed, so further implementations of more complex fluid-solid interactions can be performed using this implementation model as foundation.

5.2. Further work

In the following Subsections, the three main fields that where addressed during this work are commented on how to proceed for further work.

5.2.1. Computational architecture

The compatibility of a given parallel computing system is sometime puzzling. Basically because of the heterogeneous nature of the processing device in GPU hardware these days. Parallel processing is a hardware-depending feature, where the adoption of parallel processing systems requires a rule of requirements of the processing demand. Some systems use the number of GPUs, others, the GPU memory, and others, the manufacturer compatibility with AMD® or NVIDIA® (the largest manufacturers), but, so far, there is no clear rules to measure cross-platform parallel capabilities in heterogeneous computing. OpenCL-based systems are, may be,

the most cross-platform parallel processing available for the GPU. This work relies in such library, but in the long term, the usability of OpenCL through manufacturers may be limited or disabled and systems will be enforced to migrate or expire. For such reason, the development of this work has been as conceptual as possible for a better understanding, rather than in a pseudocode or scripting writing fashion.

Many of them has been explained on GPU performance for simulation of classic physics problems. Nowadays, the limits of modern GPUs have been improved significantly for real-time interaction on 3D sceneries using the available libraries for the GPU. But these improvements are mostly based on graphics and visualization, and less in strengthen the quality in time and/or space of the discrete physical model to simulate. This work pursues the implementation of the physical model rather than pure visualization of such model, contributing to increase the available modelling that can be exploit with GPU potential. Further work will go in this direction to develop new physical models with GPU parallel processing, or to improve this model adding physical complexity or different pouring modalities (*e.g.* partial liquid transfer or pouring on accelerated motion).

The technologies used for the purposes of this work are correct in both hardware and software. However, all simulations were conducted in 2D in this work, and results are expressed in 2D as well. A 3D fluid may enrich these simulations, but that implementation will heavily increase the number of fluid particles to simulate. With the hardware technology employed in this work, a large 3D real-time simulation was impossible to develop for this reason. Then, further work will include the implementation of these simulations in 3D with a much modern GPU hardware with higher capabilities.

5.2.2. Fluid-solid motion model

There are vast applications for closed channel and for free surface flows, and most of them can be modelled with minimum assumptions so a replicable model can be used and adapted to robotic motion kinematics. The logic step in modelling is the experimentation of such model so the simulation parametrization can be tested on real robotics applications. In the case of the profile rotation, the experimentation seems not to be challenging from the setup, but it would if the rotation motor does not provide equivalent torque response. That is the potential of

simulation that allows, not only to define the response, but also to size the elements that will develop the final task for which those were simulated. In fact, a more specific application to torque control can be used in further robotics work, like the study of efficient fish propulsion that will derive from the profile rotation presented in this thesis.

In this work, the simulated solids had one DoF to simulate and control. One interesting future work, in the robotics field, can include more DoF to the solids to simulate, so a kinematic chain could allow more complex control techniques to implement.

3D modelling, in the cases presented, is expected to be redundant since the simulation are limited to the laminar or inviscid regime and then the flow lines are orthogonal to the x-plane. However, in the pouring case, in unconstrained scenarios, vessels are usually cylindrical rather than hexahedral. In such cylindrical vessel, the flow curvature can turn from turbulent to laminar (free fall) faster than in the hexahedral. This, due to the vessel edge. For such reason, it will be interesting the test and compare them between each other in further work.

5.2.3. Interactive model design

As mentioned earlier, the industry development of GPU processing is focus on visualization specially in interactive games and simulations. Then, further work could improve user interaction by adding realistic visualization. This, without compromising the importance of focusing on the physical model accuracy.

Another future improvement to the current simulator could include a graphical user interface which could offer to the user more practical interaction within the simulator. Events calls and simulation parameters could be stablished by the user in this interface so the redesign process (Figure 1.1) could be more robust and practical.

The user interaction, in this work, has helped to redesign the model parameters by simple inspection of the real-time kinematics. However, it would be interesting to use the user interaction capability to add different force obstacles and constrains, in such a way that the model can respond to its limits in time, frequency and fluid parametrization. That type of user

interaction can be powered with haptic forces, where force feedback can be returned to the user or another coupled interface.

Abbreviations

CFD	Computational Fluid Dynamics
GPU	Graphic Processing Unit
CPU	Central Processing Unit
RANS	Reynolds-average Navier-Stokes *equations*
LES	Large Eddy Simulation
SPH	Smoothed Particle Hydrodynamic
Re	Reynolds Number
FDM	Finite Difference Method
FEM	Finite Element Method
VoF	Volume of Fluid *method*
KVS	Karman Vortex Street
NSE	Navier-Stokes Equations
CFL	Courant-Friedrich-Lewy *condition*
KJT	Kutta-Joukowski Theorem
SOS	Second Order System
CLF	Closed Loop Feedback *system*
CFS	Constant Flow Stage
AFS	Approaching Flow Stage
SIM	Slosh Induction Method
PID	Proportional, Integral and Derivative *controller*
STF	System Transfer Function
Gp	Plant Gain
Gc	Controller Gain
DoF	Degree of Freedom

Notation

Matrix are in uppercase with regular letter format

Vectors are in **bold** (also indicated with diacritic mark ⇀)

Scalars are in *italic*, may go in lowercase or uppercase

In continuous time, variables noted in *italic* as a function of time t

Nomenclature

Variable/symbol	**Description**
v, v_0	fluid velocity, initial fluid velocity
x, y, z	distance in cartesian coordinates
p, p_0	pressure, reference pressure
ρ, ρ_0	density, rest density
τ	viscous stress
β and α superscripts	generalized space coordinates
M, m, t	total mass, mass, time
μ	dynamic viscosity
g, $\mathbf{g}$	gravity constant, gravity vector
k_p, c_s	pressure constant, speed of sound
U_0	maximum flow velocity
Ω	fluid volume
f	function of the integral approximation
δ, W	Dirac delta, smoothing function
$\mathbf{r}$	particle position vector
h	distance (also noted as length or height)
∇, ∇^2	gradient operator, laplacian operator
V_p, V_f	particle volume, domain volume
N_{max}	maximum number of particles
d_{rest}	rest distance between particles
δt	integration time step
a	particle acceleration force
λ	CFL index
$\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2}$	triangle vertices
$\mathbf{a}, \mathbf{b}, \mathbf{c}$	position vectors
$\mathbf{n}$	normal vector of triangle

F^x	force in x-direction
C_d, S_t	drag coefficient, Strouhal number
d	cylinder diameter
$freq$, $Freq$	vortex frequency, free-surface frequency
$\mathbf{p}(t)$	position vector as a function of time
$\mathrm{R}(t)$	rotation matrix
$\mathbf{x}$, $\mathbf{n}$	translation vector, normal vector
r	element of the rotation matrix
$\mathbf{v}$, $\boldsymbol{\omega}$	linear and angular velocity vectors
( ), [ ]	continuous and discrete function indicators
L_m, I	angular momentum, tensor of inertia
pr_w	profile width
θ, $\boldsymbol{\tau}$, $\boldsymbol{\alpha}$	angular position, torque and acceleration
Y	Set of state variables
Γ	flow circulator
l, L, A, B	dimensioning constants
K, N	Controller constant, derivative filter constant

References

1. M. Dashti, A. Fedorova, "Analyzing memory management methods on integrated CPU-GPU systems", *ACM SIGPLAN Notices*, vol. 52, no. 9, pp. 59–69, Jun. 2017. [CrossRef]
2. R. N. Jazar, *Theory of Applied Robotics; Kinematics, Dynamics, and Control*, 2nd ed., USA: Springer, 2010. [CrossRef]
3. SolidWorks, Dassault Systems, https://www.solidworks.com/ (accessed: October 21, 2020)
4. I. Millington, *Game Physics Engine Development*, USA: Elsevier, 2007. [CrossRef]
5. A. Andrade, Game engines: a survey, *EAI Endorsed Transactions on Serious Games*, vol. 2, no. 6, pp. 10 -11, 2015. [CrossRef]
6. Blender, Free open source 3D creation software, https://www.blender.org/ (accessed: October 21, 2020)
7. MatLab®, Mathworks, https://www.mathworks.com (accessed: October 21, 2020)
8. Z. Pan, Ch. Park, D. Manocha, "Robot motion planning for pouring liquids", In Proceedings of the 26th Int. Conf. on Automated Planning and Scheduling, UK., Jun, 2016. [CrossRef]
9. C. Schenk, D. Fox, "Visual close loop control for pouring liquids", In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Singapore, May-Jun. 2017. [CrossRef]
10. E. Davis, "Pouring liquids: A study in commonsense physical reasoning", *Artificial Intelligence*, vol. 172, no. 12-13, pp. 1540-1578, Aug., 2008. [CrossRef]
11. K. Yano, T. Yoshida, M. Hamagushi, K. Terashima, "Liquid container transfer considering the suppression of sloshing for the change of liquid level", IFAC 13th Triennial World Congress, USA, 1996. [CrossRef]
12. T. Lopez-Guevara, N. K. Taylor, M. U. Gutmann, S. Ramamoorthy, K. Subr, "Adaptable Pouring: Teaching robots not to spill using fast but approximate fluid simulation", In Proceedings of the 1st Conference on Robot Learning, USA, Nov. 2017. [CrossRef]
13. D. Baraff, "An introduction to physically based modeling: Rigid body simulation I - Unconstrained rigid body dynamics", Robotics Institute, Carnegie Mellon University, USA, Technical Report, 1997. [CrossRef]
14. O. E. Krog, "GPU-based Real-Time Snow Avalanche Simulations", M. Sc. Thesis, Norwegian University of Science and Technology, Norway, 2010. [CrossRef]
15. A. Boeing, T. Bräunl, "Evaluation of real-time physics simulation systems", In Proceedings of the 5th international conference on Computer graphics and interactive techniques, Australia, Dec. 2007. [CrossRef]
16. C. Ericson, Real–Time Collision Detection, 1st ed.; Sony Computer Entertainment America, Technical Report, San Francisco, CA, USA, 2004. [CrossRef]
17. Bullet Real-time physics simulator, https://pybullet.org/ (accessed: October 21, 2020)
18. ReactPhysics3D, Open-sourced C++ physics engine, https://www.reactphysics3d.com/ (accessed: October 21, 2020)
19. OpenGL, The Industry Standard for High Performance Graphics, https://www.opengl.org/ (accessed: October 21, 2020)
20. Vulkan, The Khronos Group, https://www.khronos.org/vulkan/ (accessed: October 21, 2020)
21. The Free OpenGL Utility Toolkit, http://freeglut.sourceforge.net/ (accessed: October 21, 2020)

22. SFML: Simple and Fast Multimedia Library, https://www.sfml-dev.org/ (accessed: October 21, 2020)
23. SDL: Simple Direct Media Layer, http://www.libsdl.org/ (accessed: October 21, 2020)
24. M. Müller, D. Charypar, M. Gross, "Particle-Based Fluid Simulation for Interactive Applications", Eurographics/SIGGRAPH Symposium on Computer Animation, San Diego, Ca., USA, July 2003. [CrossRef]
25. C. Moulinec, R. Issa, D. Latino, P. Vezolle, D. R. Emerson, X.-J. Gu, "High-Performance Computing and Smoothed Particle Hydrodynamics", Parallel Computational Fluid Dynamics, Springer-Verlag Berlin Pub. Lecture Notes in Computational Science and Engineering, Vol. 74, pp 265-272, 2010. [CrossRef]
26. F. M. White, "*Mecánica de fluidos*", 5th ed., Spain: McGrawHill, 2004. [CrossRef]
27. Liu, G.R.; Liu, M.B. Smoothed Particle Hydrodynamics, A meshfree particle method, 1st ed.; World Scientific Publishing Company: Toh Tuck, Singapore, 2003. [CrossRef]
28. J. Allard, H. Courtecuisse, F. Faure, "Implicit FEM and Fluid Coupling on GPU for Interactive Multiphysics Simulation", ACM SIGGRAPH, Canada, Aug. 2011, [CrossRef]
29. R. Reddy, R. Banerjee, "GPU accelerated VOF based multiphase flow solver and its application to sprays", *Computers and Fluids*, vol. 117, pp. 287–303, Aug. 2015. [CrossRef]
30. T. Kajishima, K. Taira, "*Computational Fluid Dynamics, Incompressible Turbulent flows*", Springer International Publishing, 2017. [CrossRef]
31. S. B. Pope, "*Turbulent flows*", 1st ed., UK: Cambridge University Press, 2000. [CrossRef]
32. N. D. Katopodes, "*Free-Surface Flow: Computational Methods*", 1st ed., UK: Elsevier Inc., 2018. [CrossRef]
33. A. A. Mohamad, *Lattice Boltzmann Method Fundamentals and Engineering Applications with Computer Code*, 2nd ed., London, UK: Springer-Verlag, 2011. [CrossRef]
34. S. Jain, N. Tripathi, P. J. Narayanan, "Interactive Simulation of Generalized Newtonian Fluids using GPUs", In Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing, India, Dec. 2014. [CrossRef]
35. R.A. Gingold; J.J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars", *Monthly Notices of the Royal Astronomical Society*, vol. 181, no. 3, pp. 375–389, Dec. 1977. [CrossRef]
36. L.B. Lucy, "A numerical approach to the testing of the fission hypothesis". *Astronomical Journal*, vol. 82, pp. 1013–1024, Dec. 1977. [CrossRef]
37. OpenMP, The API specification for parallel programming, https://www.openmp.org/ (accessed: October 21, 2020)
38. W. Li, Z. Fan, X. Wei, A. Kaufman, "*GPU Gems 2: Flow simulation with complex boundaries*", Chapter 47, USA: Addison-Wesley Professional, 2005. [CrossRef]
39. D. Marsh, "Molecular dynamics-Lattice Boltzmann hybrid method on graphics processors" M. Sc. Thesis, University of Illinois at Urbana-Champaign, Urbana, IL., USA, 2010. [CrossRef]
40. P. Kipfer, M. Segal, R. Westermann, "UberFlow: A GPU-Based Particle Engine", In Proceedings of the ACM SIGGRAPH 2004, USA, Aug. 2004. [CrossRef]
41. A. Kolb, N. Cuntz, "Dynamic Particle Coupling for GPU-based Fluid Simulation", In International Proceedings of the 18th Symposium on Simulation Technique, 2005. [CrossRef]
42. T. Harada, S.Koshizuka, Y. Kawaguchi, "Smoothed Particle Hydrodynamics on GPUs", In Proceedings of the 5th International Conference on Computer Graphics, Petropolis, Brazil, May-Jun. 2007. [CrossRef]
43. M. Becker, M. Teschner, "Weakly compressible SPH for free surface flows", In Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, USA, Aug. 2007. [CrossRef]

44. M. Kass, G. Miller, "Rapid, stable fluid dynamics for computer graphics", *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 4, pp. 49–57, Sept. 1990. [CrossRef]
45. R. C. Hoetzlein, T. Höllerer, "Analyzing Performance and Efficiency of Smoothed Particle Hydrodynamics", UC Santa Barbara, USA, Technical Article, 2008. [CrossRef]
46. W. Jian, D. Liang, S. Shao, R. Chen, K. Yang, "SPH Simulations of Dam-break Flows around Movable Structures", *International Journal of Offshore and Polar Engineering*, vol. 26, no. 1, Feb. 2016. [CrossRef]
47. S. Marrone, A. Colagrossi, M. Antuono, G. Colicchio, G. Graziani, "An accurate SPH modeling of viscous flows around bodies at low and moderate Reynolds numbers", *Journal of Computational Physics,* vol. 245, pp. 456–475, Jul. 2013. [CrossRef]
48. L. K. H. Yndestad, "Particle-based Powder-snow Avalanche Simulation Using GPU", M. Sc. Thesis, Norwegian University of Science and Technology, Norway, Jun. 2011. [CrossRef]
49. E. Didier, D. Neves, P. Teixeira, J. Dias, M. G. Neves, "Smoothed particle hydrodynamics numerical model for modeling an oscillating water chamber", *Ocean Engineering*, vol. 123, pp. 397–410, Sept. 2016. [CrossRef]
50. J. J. Monaghan, "Smoothed Particle Hydrodynamics", *Annu. Rev. Astron. Astrophys.*, vol. 30, pp. 543-574, Sept. 1992. [CrossRef]
51. M. Desbrun and M.-P. Cani, "Smoothed Particles: A new paradigm for animating highly deformable bodies", In Proceedings of the Eurographics workshop on Computer animation and simulation, Poitiers, France, Aug.-Sept. 1996. [CrossRef]
52. J. P. Morris, P. J. Fox, Y. Zhu, "Modeling Low Reynolds Number Incompressible Flows Using SPH", *Journal of Computational Physics*, vol. 136, no. 1, 214–226, Sept. 1997. [CrossRef]
53. J. J. Monaghan, Simulating Free Surface flows with SPH, *Journal of Computational Physics*, vol. 110, no. 2, pp. 399-406, Feb. 1994. [CrossRef]
54. J.P. Morris, "Analysis of smoothed particle hydrodynamics with applications", Ph.D. Thesis, Dept. Math., Monash University, Australia, 1996. [CrossRef]
55. J. J. Monaghan and J. C. Lattanzio. A refined particle method for astrophysical problems. *Astronomy and Astrophysics*, vol. 149, no. 1, pp. 135–143, August 1985. [CrossRef]
56. D. De Padova, R. A. Dalrymple, M. Mossa, A. F. Petrillo, "SPH Simulations of Regular and Irregular Waves and their Comparison with Experimental Data", *Cornell University Library*, USA, Nov. 2009. [CrossRef]
57. I. Johnson, "Real-Time Particle System in the Blender Game Engine", M. Sc. Thesis, The Florida State University, USA, 2011. [CrossRef]
58. J. K. Chen, J. E. Beraun, T. C. Carney, "A corrective smoothed particle method for boundary value problems in heat conduction", *International Journal for Numerical Methods in Engineering*, vol. 46, no. 2, pp. 231-252, Sept. 1999. [CrossRef]
59. R. P. Fedkiw, "Coupling an Eulerian fluid calculation to a Lagrangian solid calculation with the ghost fluid method", *Journal of Computational Physics*, vol. 175, no. 1., pp. 200-224, Jan. 2002. [CrossRef]
60. V. Jones, Q. Yang, L. McCue-Weil, "SPH Boundary Deficiency Correction for Improved Boundary Conditions at Deformable Surfaces", Ship Science and Technology, vol. 4, no. 7, pp. 21-30, 2010. [CrossRef]
61. H. Schechter, R. Bridson, "Ghost SPH for animating water", *ACM Transactions on Graphics*, vol. 31, no. 4, Jul. 2012. [CrossRef]
62. M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, M. Teschner, "SPH Fluids in Computer Graphics", Eurographics 2014. [CrossRef]
63. M. Ihmsen, N. Akinci, M. Gissler, M. Teschner, "Boundary Handling and Adaptive Time-stepping for PCISPH", In Proceedings of the Seventh Workshop on Virtual Reality

Interactions and Physical Simulations, Denmark, 2010. [CrossRef]
64. R. Courant, K. Friedrichs, H. Lewy, "Uber die partiellen differen zengleichungen der mathematik hysichysik", *Mathematische Annalen*, Vol. 100, pp. 32-74, 1928. [CrossRef]
65. Khronos Group website, https://www.khronos.org/about/ (accessed: October 21, 2020)
66. Online Math Tools, Z-order Curve Generator, https://onlinemathtools.com/generate-z-order-curve (accessed: October 21, 2020)
67. G. M. Morton, A computer oriented geodetic data base and a new technique in file sequencing, IBM, Ottawa, Canada, Technical Report, 1966. [CrossRef]
68. S. Green, "Particle Simulation using CUDA", v. 1.3, NVIDIA, USA, Technical Report, May, 2010. [CrossRef]
69. T. Harada, L. Howes, "Introduction to GPU Radix Sort", Advanced Micro Devices, Inc., USA, Technical Report, 2011. [CrossRef]
70. NVIDIA CUDA Sample Code, http://developer.download.nvidia.com/compute/DevZone/C/html_x64/samples.html (accessed: October 21, 2020)
71. OpenCL implementation of Radix Sort Algorithm, C++ class for sorting integer lists in OpenCL, https://github.com/avinashcpandey/ocl-radix-sort (accessed: October 21, 2020)
72. N. Satish, M. Harris, M. Garland, "Designing efficient sorting algorithms for many core GPUs", In Proceedings of the IEEE International Parallel and Distributed Processing Symposium, Italy, May 2009. [CrossRef]
73. R. C. Hoetzlein , "Fast fixed-radius nearest neighbors : Interactive million-particle fluids", Graphics Devtech, NVIDIA, USA, Technical Report, 2014. [CrossRef]
74. W. W. Hwu, *GPU Computing Gems, Jade Edition*, 1st ed., USA: ElSevier, 2012. [CrossRef]
75. OpenCL Sorting Networks, NVIDIA 2011, http://developer.download.nvidia.com/compute/cuda/3_0/sdk/website/OpenCL/website/samples.html#oclSortingNetworks (accessed: October 21, 2020)
76. R. Fernando, M. Pharr, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, USA: Addison-Wesley Professional, 2005. [CrossRef]
77. P. Young, "The leapfrog method and other "symplectic" algorithms for integrating Newton's laws of motion", University of California, Santa Cruz, Physics Course Notes, 2014. [CrossRef]
78. H. Takeda, S. M. Miyama, M. Sekiya, "Numerical Simulation of Viscous Flow by Smoothed Particle Hydrodynamics", *Progress of Theoretical Physics*, Vol. 92, No. 5, pp. 939-960, Nov. 1994. [CrossRef]
79. A. Bauer, V. Springel, "Subsonic turbulence in smoothed particle hydrodynamics and moving-mesh simulations", *Monthly Notices of the Royal Astronomical Society*, vol. 423, no. 3, pp. 2558-2578, Jun. 2012. [CrossRef]
80. S. Adami, X. Y. Hu, N. A. Adams, "Simulating three-dimensional turbulence with SPH", Center for Turbulence Research, In Proceedings of the Summer Program, USA, 2012. [CrossRef]
81. C. Tapia, R. Chellali, "Simple Karman Street model", Oceans 2010 IEEE, Sydney, Australia, May 2010. [CrossRef]
82. C. H. K. Williamson, Vortex dynamics in the cylinder wake, *Annu. Rev. Fluid Mech.*, vol. 28, pp 477-539, 1996. [CrossRef]
83. R. C. Z. Cohen, P.W. Cleary, "Computational Studies of the Locomotion of Dolphins and Sharks Using Smoothed Particle Hydrodynamics", In Proceedings of the 6th World Congress of Biomechanics, Singapore, 2010. [CrossRef]
84. I. Abbott, A. Doenhoff, L. Stivers, "Summary of Airfoil Data", National Advisory Committee for Aeronautics, USA, Technical Report, 1945. [CrossRef]
85. C. F. Heddleson, D. L. Brown, R. T. Cliffe, "Summary of drag coefficients of various

shaped cylinders", General Electric, USA, Technical Report, 1957. [CrossRef]
86. J. N. Woods, M. Breuer, G. De Nayer, "Experimental investigations on the dynamic behavior of a 2-DOF airfoil in the transitional Re number regime based on digital-image correlation measurements", *Journal of Fluids and Structures*, vol. 96, 2020. [CrossRef]
87. L. Goffin, S. Erpicum, B. Dewals, M. Pirotton, P. Archambeau, "Validation of a SPH Model for Free Surface Flows", SimHydro: Modelling of rapid transitory flows, France, Jun. 2014. [CrossRef]
88. J. Kajtar, J. J. Monaghan, "SPH Simulations of Swimming linked bodies", *J. Comput. Phys.*, vol. 227, no. 19, pp. 8568-8587, 2008. [CrossRef]
89. J. He, X. Chen, Z. Wang, C. Cao, H. Yan, Q. Peng, "Real-time adaptive fluid simulation with complex boundaries", *The Visual Computer*, vol. 26, pp. 243-252, 2010. [CrossRef]
90. A. Ito, Y. Noda, K. Terashima, "Outflow Liquid Falling Position Control by Considering Lower Ladle Position and Clash Avoidance with Mold", In Proceedings of the 3rd IFAC Workshop on Automation in the Mining, Mineral, and Metal Industries, Japan, Sept. 2012. [CrossRef]
91. K. Raju, G. L. Asawa, "Viscosity and Surface Tension effects on Weir Flow", *Journal of Hydraulics Division*, , vol. 103, no. 10, pp. 1227-1231, 1977. [CrossRef]
92. S. W. Bauer, "The Free Overfall as a Flow Measurement Device", M. Sc. Thesis, Lehigh University, Bethlehem, PA, USA, 1970. [CrossRef]
93. G. J. Lee, "Moment of inertia of liquid in a tank", *Int. J. Nav. Archit. Ocean Eng.*, vol. 6, no. 1, pp. 132-150, 2014. [CrossRef]
94. R. A. Ibrahim, "*Liquid Sloshing Dynamics Theory and Applications*", 1st ed., UK: Cambridge University Press, 2005. [CrossRef]
95. M. Grundelius, "Methods for Control of Liquid Slosh" Ph.D. Thesis. Lund University, Lund Sweden, October 2001. [CrossRef]
96. H. N. Abramson, "The Dynamic Behavior of Liquids in Moving Containers"; H.N. NASA SP-106, Technical Report, Washington, DC, USA, 1966. [CrossRef]
97. A. Munsh, B. Gaster, T. Mattson, J. Fung, D. Ginsburg, "*OpenCL Programming Guide*", 1st ed., Pearson Education, Inc., 2012. [CrossRef]

Appendix A

Mathematical framework of SPH

A.1. The gradient of pressure force ∇p

The gradient term of Equation (3.5) that turns into Equation (3.10) is developed by applying the symmetric gradient approximation formula in [96] which demonstrate symmetric forces to fulfill Newton's law.

Considering the vector calculus identity,

$$\nabla(A \cdot B) = A \cdot \nabla B + B \cdot \nabla \mathrm{A} \tag{A.1.1}$$

being $A \equiv f$ and $B \equiv \rho^{-1}$, (A.1.1) yields to,

$$\nabla\left(\frac{f}{\rho}\right) = \nabla(f\rho^{-1}) = f \cdot \nabla(\rho^{-1}) + \rho^{-1} \cdot \nabla(f) \tag{A.1.2}$$

$$= -f\rho^{-2} \cdot \nabla(\rho) + \rho^{-1} \cdot \nabla(f) \tag{A.1.3}$$

clearing $\frac{\nabla(f)}{\rho}$ from last equation the following expression is found,

$$\frac{\nabla(f)}{\rho} = \nabla\left(\frac{f}{\rho}\right) + \frac{f\nabla(\rho)}{\rho^2} \tag{A.1.4}$$

Now, SPH interpolation of Equation (3.9) can be applied to this last one equation, knowing that $f \equiv p$ and in SPH notation the following is found,

$$\frac{\nabla(p_j)}{\rho_j} = \sum_i \left[\frac{m_i}{\rho_i}\frac{p_i}{\rho_i}\nabla_i W(\mathrm{r}_j - \mathrm{r}_i, h)\right] + \frac{p_j}{{\rho_j}^2}\sum_i \left[\frac{m_i}{\rho_i}\rho_i \nabla_i W(\mathrm{r}_j - \mathrm{r}_i, h)\right] \tag{A.1.5}$$

$$= \sum_i \left[m_i \frac{p_i}{{\rho_i}^2}\nabla_i W(\mathrm{r}_j - \mathrm{r}_i, h) \; + \; m_i \frac{p_j}{{\rho_j}^2}\nabla_i W(\mathrm{r}_j - \mathrm{r}_i, h)\right] \tag{A.1.6}$$

$$\boxed{\frac{\nabla(p_j)}{\rho_j} = \sum_i m_i \left[\frac{p_i}{{\rho_i}^2} + \frac{p_j}{{\rho_j}^2}\right]\nabla_i W(\mathrm{r}_j - \mathrm{r}_i, h)} \tag{A.1.7}$$

A.2. The laplacian of viscosity force $\nabla^2 v$

By reference of [96], the Laplacian term of Equation (3.5) that turns into Equation (3.12) is developed.

Considering two times the gradient vector identity of the product and one time the divergence identity the second order derivative (laplacian) term can be formed as,

$$\nabla^2(\rho \cdot f) = \nabla\big(\nabla(\rho \cdot f)\big) = \nabla(\rho\nabla f + \; f\nabla\rho) = \nabla(\rho\nabla f) + \; \nabla(f\nabla\rho)$$

(A.2.1)

$$\nabla^2(\rho \cdot f) = \nabla\rho\nabla f + \; \rho\nabla^2 f + \nabla\rho\nabla f + f\nabla^2\rho \tag{A.2.2}$$

$$\nabla^2(\rho \cdot f) = \underset{*^1}{\rho\nabla^2 f} + \underset{*^2}{f\nabla^2\rho} + \underset{*^3}{2\nabla\rho\nabla f} \tag{A.2.3}$$

If $f \equiv v_i$, and clearing $*^1$, yields to,

$$\nabla^2(v) = \frac{\nabla^2(\rho v)}{\rho} - \frac{v\nabla^2(\rho)}{\rho} - \frac{2\nabla(v)\nabla(\rho)}{\rho} \tag{A.2.4}$$

The last right term of (A.2.4) can be reduced to 0 since ρ is constant and $\nabla(\rho) = 0$. So it ends being,

$$\nabla^2(v) = \frac{\nabla^2(\rho v)}{\rho} - \frac{v\nabla^2(\rho)}{\rho} \tag{A.2.5}$$

Now, SPH interpolation in Equation (3.11) can be applied to Equation (A.2.5), and, in SPH notation, ending as,

$$\rho_j\nabla^2(v_j) \approx \sum_i \left[\frac{m_i}{\rho_i}\rho_i v_i {\nabla_i}^2 W(\mathrm{r}_j - \mathrm{r}_i, h)\right] - v_j \sum_i \left[\frac{m_i}{\rho_i}\rho_i {\nabla_i}^2 W(\mathrm{r}_j - \mathrm{r}_i, h)\right] \tag{A.2.6}$$

$$\rho_j\nabla^2(v_j) \approx \sum_i [m_i v_i {\nabla_i}^2 W(\mathrm{r}_j - \mathrm{r}_i, h)] - \sum_i [m_i v_j {\nabla_i}^2 W(\mathrm{r}_j - \mathrm{r}_i, h)] \tag{A.2.7}$$

$$\boxed{\nabla^2(v_j) \approx \sum_i m_i \frac{(v_i - v_j)}{\rho_j} {\nabla_i}^2 W(\mathrm{r}_j - \mathrm{r}_i, h)} \tag{A.2.8}$$

Appendix B

C/C++ programming

B.1. Rendering

OpenGL uses its own custom C programming Shading Language (GLSL). This language is used to write scripts called "shaders" that run in the GPU. A render pipeline is used by the GPU to run shaders. The purpose of this pipeline is to render native 3D data into a 2D pixel screen. There are two modes to render in OpenGL. One, in the immediate mode that writes in GLSL the vertices of 3D geometries of the user. This mode make use of the commands glBegin and glEnd or glDrawArrays to write the vertex primitives of geometries. The disadvantage of this mode is that during the call of these commands the user data is transferred from CPU memory to GPU memory blocking the rendering pipeline and the user application until these calls end. This mode is only valid for static geometric content that is not called repeatedly, and not for dynamic content that needs to be refreshed with the pipeline framerate.

For such dynamic data, the best mode is the retained mode. Under this mode, user data is attached to OpenGL buffers and these buffers are directly handle by an OpenGL process. Then, the user application only needs to update its data and leaves OpenGL process to handle the pipeline update. Also, in this mode, the user can send OpenGL commands to draw different geometries without caring much about rendering timing because the OpenGL process will queue these commands and handle the rendering itself. This is especially interesting when dealing with a

real-time applications where heavy loads of data is intended to be processed seamlessly with the rendering. In this type of applications, it is desirable to leave the rendering task to the system and focus on data processing, and this is the case of this work simulator. It is an enormous enhancement for real-time simulations that reduces computational cost significantly.

In Section 3.6.1, the simulator structure is presented and illustrated in a diagram block. In Figure 3.5, a rendering block is presented at the end of the diagram. The rendering task is based on the retained mode. Under this mode, the OpenGL rendering process is handled by the OpenGL vertex buffer object (VBO). VBO is an OpenGL resource to transfer OpenGL vertex data (position and color) to the GPU memory. In the simulator of this work, these vertex data is defined by the position vertex of the fluid particles. The color vertex are also considered for the color of the fluid particles.

In the following table code a method procedure for the VBO is presented just as it has been programmed in the simulator of this work. OpenGL version is 4.5.

No.	Retained mode process	Sample code

1	Create a buffer object and return its identification integer number (**ids**) starting from 1. 0 is reserved. **n** is the number of vars to be uploaded, *e.g.* one for vertex data and two for color data.
	
	void glGenBuffers(GLsizei n, GLuint* ids)
		//Create a variable to hold the VBO identifier for vertices and colors
		GLuint triangleVBO[2];
		
		//Define the vertices of a triangle to be loaded to the GPU
		float verts[] = {1.0f, 0.2f, 0.05f, -0.98, -0.6, 1.0, 0.3, 0.2, 7.0};
		float colors[] = {0.8f, 0.8f, 0.8f};
		
		//Initialize VBO once at the start of program
		glGenBuffers(2, &triangleVBO);

2	Activate a previous buffer object. **target** is the OpenGL object type of the **buffer** the VBO identifier. **glBindBuffer(enum target, uint buffer)**	//Make the new buffer object active **glBindBuffer(GL_ARRAY_BUFFER, triangleVBO);**
3	Upload data to the active buffer object. Arguments are **data**, data **size**. **usage** argument is an memory optimization hint to OpenGL. **glBufferData(enum target, sizeiptrARB size, const void *data, enum usage)**	//Upload vertex data to the GPU device **glBufferData(GL_ARRAY_BUFFER, sizeof(verts), verts, GL_STATIC_DRAW);**
4	(Optional) Delete a number **n** of **buffer** objects from data or by VBO identifier. **glDeleteBuffers(sizei n, const uint *buffers)**	//Delete buffer 1 located for verts. **glDeleteBuffers(1, & triangleVBO);**

Table B.1. Retained mode process for OpenGL VBO.

B.2. OpenCL Scripting

OpenCL scripting is supported by a derivation of the C language, defined as ISO/IEC 9899:1999 C, and also known as C99 for short. Yet, there are C++ bindings and C++ Wrapper APIs that allow to code in C++ without C99 coding, it is a good practice to know native OpenCL code to explode maximum performance.

OpenCL's language has been selected for maximum portability, and its Kernels (OpenCL structures that run in the GPU) are compiled in a just-in-time mode. This means that the Kernel, typically written in a separated file with4 distinctive .cl extension, are compiled during the execution of the program, at run-time, instead of being previously compiled during the C++ host program. This is the intended portability feature of OpenCL that reduces programming overhead and allows to use Kernel files in multiple applications. This, by considering that parallel programming solves characteristic algorithms that happen to improve computational cost and that could be shared in cross-platforms.

One notable disadvantage of C99 OpenCL is code debugging. In fact, debugging is not feasible through common C++ debugging tools, only a console output can be implemented. This drawback is typically resolved by simple data comparison of CPU processed output vs GPU processed output.

Since GPU selection over CPU is based on the computational cost performance, then OpenCL profiling is one important test to tack in. But again, the profiling tools depend on the available tools developed by each GPU manufacturer. For instance, Intel has its VTune tool, NVIDIA its Visual Profiler and AMD its CodeXL. Apparently, this is a common issue in heterogeneous computing. Code needs to target specific hardware or provide a limited range of graphical boards with previous base software and driver requirements.

As mentioned earlier, Kernels are functional components or structures that run in the GPU. These are, in fact, the programming core of OpenCL. There are many other topics regarding OpenCL programming flow like context, queues, buffers, programs, groups-items and many more, but those topics can be followed according to procedures in literature [97] or by API references, so no further detail will be provided on this regard in this thesis book. To abstract all this OpenCL programming concepts, as mentioned in Section 3.5, this work simulator adopts the

OpenCL wrapper included in [57]. The focus of this appendix, is based on the Kernel coding. For such task, a comprehensive description of a Kernel used in this simulator is provided in the following Table B.2.

Line	Code	Comments

1	`__kernel void leapfrog_integration(`
	`  __global float3* pos,`
	`  __global float3* vel,`
	`  __global float3* force,`
	`  float dt`
	`)`
		`// Kernel declaration`
		`// Variables passed to GPU,`
		`// including memory qualifiers`
		`// (global, local, constant or`
		`// private) and data type (float,`
		`// int, etc.), dt is time step`

2	`{` `  int pk = get_global_id(0);`	`// Get global number of items where each item is defined by a particle kernel, so, pk is the current number of particles`
3	`  float3 p_tmp = pos[pk];`	`// Temporal particle position`
4	`  float3 v_tmp = vel[pk];`	`// Temporal particle velocity`
5	`  float3 f_tmp = force[pk];`	`// Temporal particle force`
6	`  float3 v_next = v + 0.5*dt*f_tmp;`	`// Discrete integration of force in // a temporal velocity v_next`
7	`  p_tmp += dt * v_next;`	`// Discrete integration of velocity // in a temporal position v_tmp`
8	`  vel[pk] = vnext;`	`// Update global vel variable`
9	`  pos[pk] = (float3)(p.xyz);` `}`	`// Update global pos variable`

Table B.2. OpenCL Kernel code of the leapfrog integration process.

Line 1 in Table B.2. defines the Kernel declaration. Each Kernel call computes each fluid particle data, in this case, new velocity and position out of the leapfrog integration method. Current particle requires its actual force, velocity and position to compute the next step same data. The function `get_global_id`, in line 2, provides the identifier that will index the current particle data. The work group model in this script supposes a lineal distribution of Kernels, so, no work item counting is required.