POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Simulating Fluid Dynamics Effects on Small Bodies using N-body codes

Tesi di Laurea Magistrale in
Space Engineering - Ingegneria Spaziale

Author: **Andrea D'Uva**

Student ID: 942644
Advisor: Prof. Fabio Ferrari
Co-advisors: Dr. Eloy Peña Asensio
Academic Year: 2024-25

Contents

List of Figures

List of Tables

Acronyms

ADKE Adaptive Density Kernel Estimation.

API Application Programming Interface.

BCE Boundary Condition Enforcement.

BE Boltzmann Equations.

C::C Chrono::Core.

C::E Chrono::Engine.

C::FSI Chrono::FSI.

CFD Computational Fluid Dynamics.

CFL Courant-Friedrichs-Lewy.

CLL cell-linked list.

CPU Central Processing Unit.

CSPH Compressible SPH.

DSMC Direct Monte-Carlo Simulation.

EE Euler Equations.

GPU Graphics Processing Unit.

IISPH Implicit Incompressible SPH.

MBD Multybody Dynamics.

MBS Multybody System.

NSE Navier-Stokes Equations.

RK2 Runge-Kutta 2.

SPH Smoothed Particle Hydrodynamics.

VSG VulkanSceneGraph.

WCSPH Weakly Compressible SPH.

List of Symbols

Variable	Description
a	orbital semi-major axis
i	orbital inclination
T	Tisserand parameter
Kn	Knudsen number
C_D	drag coefficient
$W(\mathbf{x}-\mathbf{x}', h)$	smoothing kernel function
$\delta(\mathbf{x}-\mathbf{x})$	Dirac delta
m	particle mass
ρ	density
P	pressure
c	speed of sound
u	specific energy
$\hat{e}$	thermokinetic energy
s	specific entropy
γ	ratio of heat capacities
$\mathbf{x}$	particle position
$\mathbf{v}$	particle velocity
$\mathbf{a}$	particle acceleration
$\mathbf{g}$	volumetric force
h	smoothing length
κ	smoothing length multiplier
Π	artificial viscosity
α, β	artificial viscosity coefficients
ε	minimum inter-particle distance
H	artificial heat conduction
g_1, g_2	artificial heat conduction parameters
κ, ϵ	ADKE coefficients

Abstract

The availability of numerical solvers dedicated to the simulation of multibody systems subject to N-body gravitational interactions has significantly enriched the possibilities for studying complex astrophysical problems. Within this context, the present work aims at extending such capabilities to systems where fluid dynamics effects play a significant role, with particular focus on cometary atmospheres. Simulating these environments is a challenging task as it is not evident which model is appropriate to describe such a peculiar environment. In order to assess the feasibility of the original idea, a review of the main literature on cometary atmospheres is conducted. It is found that a fluid approach is often considered appropriate to describe the inner coma of highly active comets. At the same time, in recent years, Smoothed Particle Hydrodynamics has gained popularity in the context of computational fluid dynamics, and is now widely employed in several engineering applications. Originally designed for astrophysical problems, SPH is well suited for the study of shock interaction problems and supersonic compressible flows, which characterize the gas expansion in the first layers of cometary atmospheres. A summary of the main open-source SPH software packages is presented, with particular attention to codes that can deal with inviscid compressible fluids, and their coupling with multibody dynamics solvers is discussed. Since none of the available tools fully satisfies the initial objectives, a new CUDA-based SPH solver is proposed. Developed as an additional module for the open-source multiphysics library Project Chrono, the presented code aims at consistently solving the Euler Equations for compressible flows. Its architecture is explained in detail, and a set of test cases is analyzed in order to ensure it is capable of retaining the physical characteristics of the systems under consideration.

Keywords: Multibody dynamics, cometary atmosphere, computational fluid dynamics, Project Chrono, Smoothed Particle Hydrodynamics, inviscid compressible fluids

Sommario

La disponibilità di codici numerici per la simulazione di sistemi multibody soggetti a interazioni gravitazionali a N-corpi ha ampliato la possibilità di studiare sistemi astrofisici complessi. In questo contesto, l'idea alla base del presente lavoro consiste nell'espandere le capacità di tali strumenti alla simulazione di corpi celesti in cui i processi fluidodinamici rivestono un ruolo significativo, con particolare attenzione alle atmosfere cometarie. La simulazione di questi processi rappresenta una sfida non banale, perchè non è immediatamente evidente quale modello fisico sia più appropriato per descrivere ambienti così peculiari, in cui coesistono diversi regimi dinamici. Al fine di accertare la fattibilità dell'idea originale, in questa tesi viene presentata un'analisi della principale letteratura scientifica dedicata allo studio delle comete. Si conclude che, nei casi di elevata attività cometaria, un approccio basato sul modello di fluido continuo può essere ritenuto fisicamente adeguato per lo studio dei flussi gassosi nelle regioni più interne. Nel contesto della fluidodinamica computazionale, il metodo Smoothed Particle Hydrodynamics ha recentemente acquisito popolarità, e oggi è utilizzato in diverse applicazioni ingegneristiche. Sviluppato in origine per risolvere problemi astrofisici, il metodo SPH è particolarmente adatto nello studio dei flussi supersonici e dell'interazione tra onde d'urto, fenomeni che caratterizzano gli strati più interni delle atmosfere cometarie. Viene quindi presentata una rassegna di alcuni dei principali software open-source basati sul metodo SPH per lo studio dei fluidi comprimibili inviscidi, e viene discussa la loro compatibilità con simulatori di sistemi multibody. Constata la mancanza di strumenti adeguati a perseguire l'idea originale, viene presentato un nuovo codice SPH implementato tramite CUDA. Sviluppato come modulo aggiuntivo della libreria multifisica open-source Project Chrono, il codice è finalizzato alla risoluzione delle equazioni di Eulero per fluidi comprimibili non viscosi. L'architettura del software è discussa in modo dettagliato, e sono presentati alcuni test per verificare le capacità del programma di simulare in modo fisicamente coerente le caratteristiche dei sistemi considerati.

Parole chiave: sistemi multibody, atmosfere cometarie, fluidodinamica computazionale, Project Chrono, Smoothed Particle Hydrodynamics, fluidi comprimibili non viscosi

Introduction

The simulation of small Solar System bodies is a complex task as it involves several different processes, characterized by different physical natures. For instance, the formation phase of some classes of asteroids can be studied by considering the aggregation of a large number of smaller bodies that are influenced only by mutual gravitational forces. On the other side, phenomena such as the sublimation of active cometary nuclei involve understanding the mechanism ruling the expansion of a gas in an extremely rarefied environment. With the development of tools that can accurately simulate the evolution of multibody systems subjected to mutual collisions and N-body gravitational interaction, it comes natural to consider extending the capabilities of such solvers to include additional physical processes. For instance, the possibility of coupling fluid dynamics solvers and specialized multibody simulators would significantly enrich the classes of astrophysical bodies that could be studied numerically, providing useful insights on their physical activity and a better understanding of their nature. Therefore, the present thesis can be intended as an attempt to improve a previous work, that consisted in the development of an N-body capable code, based on the open-source multiphysics library Project Chrono, and include in it the capability of simulating fluid dynamics effects as well. However, different sources of complexity had to be addressed. First, it is not immediately evident whether a cometary environment can be modeled as an ordinary terrestrial fluid, which under most conditions can be represented as a continuum rather than as a set of discrete particles. Therefore, a review on the modeling techniques used in literature to study the cometary atmospheres needs to be carried out. Project Chrono was initially aimed only at the simulation of multibody dynamic systems, it has seen its capabilities extended, and can consider several different physical domains. Among others, an additional official module for the simulation of coupled fluid-solid systems has been developed. Such module is based on a Lagrangian method for the discretization and solution of the fluid equation, called Smoothed Particle Hydrodynamics. However, despite appearing as a well-suited tool to pursuit the original goal, this fluid-solid module has been designed for incompressible flows only. Therefore, the second source of complexity arises from the need to identify a CFD solver that could be seamlessly integrated with both Project Chrono and

the mentioned N-body code. Being Chrono inherently prone to be coupled with fluid solvers based on the SPH method, the relevant open-source SPH codes for compressible fluid dynamics have been analyzed. As it turned out, these solvers could not be easily integrated with Chrono, and it has been chosen to attempt to develop a new one. The success of this task would then enable to pursuit the original goal of coupling multibody dynamics, N-body gravitational interaction, and fluid dynamics effects, all in a single computational framework. The presented work aims at providing a new base tool which could complete the set of instruments required to perform such a complex simulation. In order to explain the different phases involved in this work, this thesis has been structured as follows:

- Chapter 1 is dedicated to comets. Their main characteristics are introduced, and the physics of their atmosphere is discussed, introducing the different models used in the literature to study such environments. The goal of this review is to establish whether a fluid approach to the simulation of cometary comae would have a proper physical meaning. In addition, a possible simple simulation of such environment is defined.
- Chapter 2 introduces the SPH method, describes the idea behind its development, and outlines its advantages and disadvantages when compared to the traditional Eulerian CFD methods.
- Chapter 3 summarizes the features of Project Chrono, introduces the N-body code, and presents some relevant open-source SPH solvers.
- Chapter 4 describes a proposed new module[1] for Chrono that aims to solve the Euler Equations for compressible fluids, from the point of view of both the software architecture and the numerical formulation at its base. It also explains how CUDA has been used to parallelize computations and to obtain an increase in performance, if compared to traditional solvers.
- Chapter 5 examines the results of test cases used to analyze the ability of the code to retain the physical characteristics of the simulated systems.
- Chapter 6 draws the conclusions of this work, and provides recommendations for future improvements and enhancements.

[1] available online on a GitHub repository

1 | Comets and models of cometary atmosphere

Comets and other small solar system bodies are present in great numbers in the Solar System, and their study provides a unique understanding of the primordial conditions, present at the times of its formation. They orbit inactive for most of their lifetimes, but when they come close enough to the Sun, they start experiencing high levels of activity, producing very complex environments. This chapter will address their main properties and classification, their origin, and it will provide an overview of how the cometary atmosphere can be modeled and studied numerically. Different physical descriptions have been used and compared in literature, from full collisionless kinetic theory to well known hydrodynamic approaches. The last section then introduces how the dynamics of gas and dust can be described, and which assumptions are usually made on the dominant phenomena.

1.1. Comet Taxonomy

Comets are Solar System bodies that can extend from hundreds of meters to tens of kilometers in size. They are mainly composed from volatile ices, dusty components, and rocky materials. Their orbits are highly elliptical and they often show a significant amount of inclination. Astronomers have classified comets based on their orbits for centuries. First, comets were divided into two classes:

- Long-period comets, with period greater than 200 years.
- Short-period comets, with period lower than 200 years.

Long-period comets can be further divided into two sets: dynamically "new" comets, and "returning" ones. This classification is based on the calculated semi-major axis, a. Estimating that a comet receives a gravitational kick of approximately $\pm 0.0005\,\mathrm{AU}^{-1}$, it was concluded, in the fifties of the last century, that comets with $a \gtrsim 10\,000$ AU were experiencing their first passage through the Solar System, while comets with $a \ll 20\,000$ AU

had already passed our system before.

In a similar way, Short-period comets had been classically divided into two groups, the first of which typically has a semi-major axis a in the order of 3 to 4 AU. Their orbits have a relatively low average inclination i of 10°. These comets have an aphelion that is close to the orbit of Jupiter, and are considered to be dominated by that planet. They are referred to as "Jupiter-family" comets. Short-period comets that are not members of the first group show significantly higher inclination, averaging at 41°, and can be placed on retrograde orbits. The boundary between the Jupiter-family and the Halley-type comets is typically set in terms of the orbital period value: comets with periods P $<$ 20 years belong to the first group, the others fall in the latter type.

The Oort Cloud and the Kuiper Belt

The main sources of Solar System comets are the Oort Cloud and the Kuiper Belt, which are two very distinct regions of space, containing a huge number of small solid bodies. In 1950, astronomer Jan Oort first proposed Long-Period comets to be originating from a vast and distant nearly spherical region, where countless icy bodies are stored in pristine conditions, from times coinciding with the formation of our system. The Oort cloud extends between 5000 and 100 000 AU, where the gravitational influence of the Sun is no longer dominant, and forces due to other stars are comparable. Gravitational interactions with the solar system bodies, with other stars, or with giant molecular clouds, are thought to scatter the comets orbiting inside this reservoir into the inner solar system. Short-Period comets were initially believed to be originating from the Oort Cloud as well, and that the gravitational interaction with other planets modified their orbits into smaller ones. However, it is known today that they originate from the Kuiper Belt, which is a region of complex disk-like shape that extends from $a \approx 35$AU to $a \approx 55$AU. Objects in the Kuiper belt have mostly elliptical orbits, and the interaction with giant planets, like Neptune, can inject them into orbiting towards the inner Solar System. While no object residing into the Oort Cloud has ever been observed, today many Kuiper Belt Objects have been directly identified and catalogued.

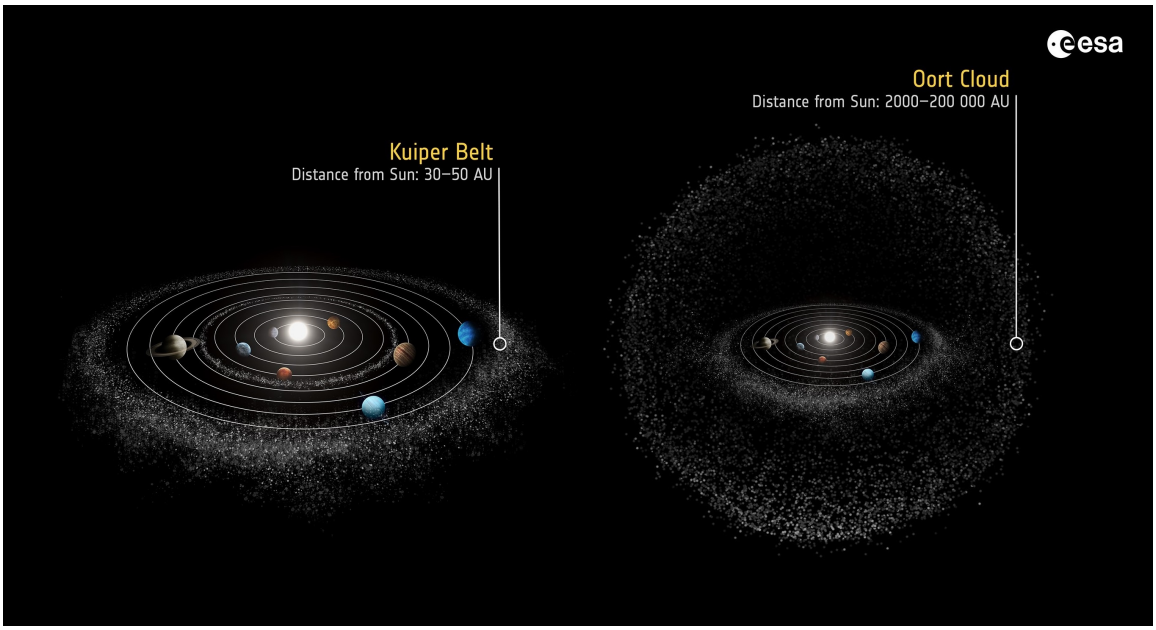


Figure 1.1: Descriptive image of Oort Cloud and Kuiper Belt. From [13].

Modern taxonomy

In modern astronomy, a new way of classifying comets in terms of their orbital motion has been introduced. This is because the classical taxonomy, based on the values of orbital period and semi-major axis, doesn't have a true intrinsic physical meaning. As a matter of fact, during their lifetime, most comets can move between Jupiter-family and Halley-type multiple times. It is more consistent to consider also the Tisserand parameter, defined as:

$$T_J = \frac{a_J}{a} + 2\cos i \sqrt{\frac{a}{a_J}\left(1 - e^2\right)} \tag{1.1}$$

where a_J is Jupiter's semi-major axis. The Tisserand parameter is an approximation of the Jacobi constant, a constant quantity in the circular restricted three-body problem. In this dynamic model, objects with $T_J > 3$ cannot cross Jupiter's orbit, and their orbit is either always exterior or always interior to Jupiter. While the classification on the orbital period allows comets to switch classes, this is not the case if the Tisserand parameter is selected as the driving value. Even if not exactly conserved in real life orbital motion, most comets are about the same value today as they first come into the inner Solar System.

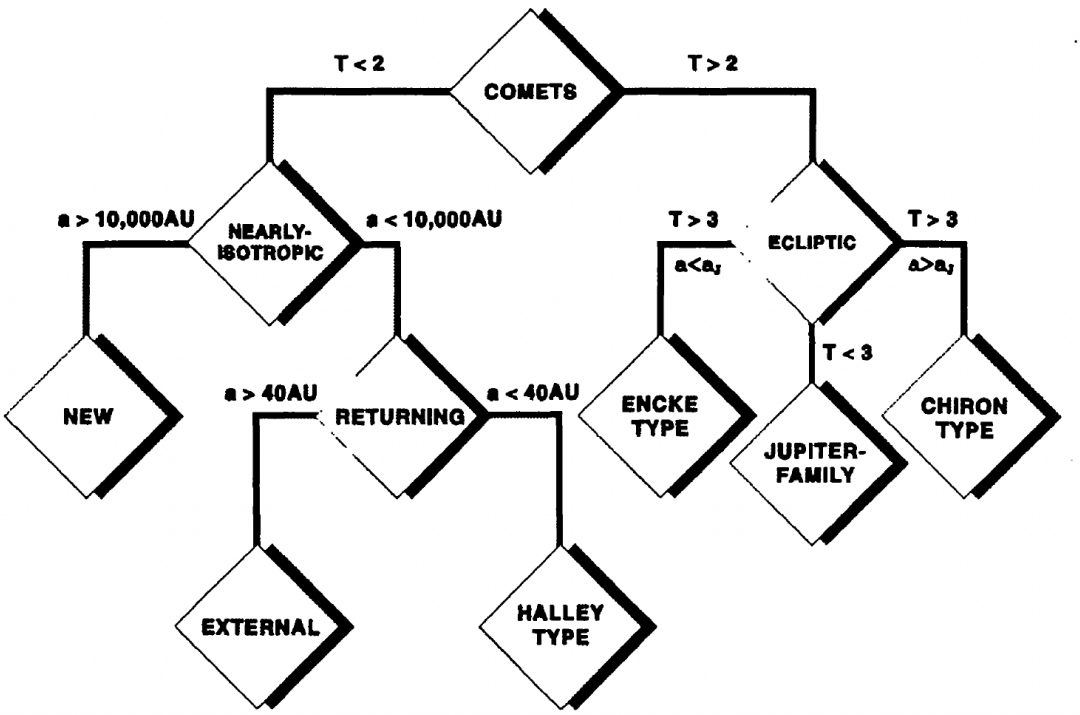


Figure 1.2: Classification of comets depending on the Tisserand parameter.

Figure 1.2 shows a possible way of defining classes depending on their Tisserand parameter value, as proposed by Levison[23]. Comets with $T > 2$ are designated as ecliptic, because most of them have very small inclinations, while the ones with $T < 2$, which mainly come from the Oort Cloud, have a wide inclination distribution; hence the label "nearly isotropic". The ecliptic comets are further divided depending on the combination of T and a. Comets with $2 < T < 3$ cross Jupiter's orbit and are dominated by that planet. Comets with $T > 3$ do not cross Jupiter's orbit and, if they have $a < a_J$, they are interior to it. These are designated as "Encke-type". A comet that has $T > 3$ and $a > a_J$ is exterior to Jupiter and is designated as "Chiron-type". Similarly to the classical taxonomy scheme, the nearly-isotropic comets can be divided depending on their orbital size into two groups: "new" and "returning". The former shares the same meaning as in the original scheme, while the latter covers a greater range of cases. One way of further specialization is to distinguish between orbits with small semi-major axis, which can thus be trapped in mean motion resonances with giant planets of the System and are labeled as "Halley-type", and comets that have larger orbits, named "external". This proposed classification of comets is not yet a definitive one, and still does not have clear boundaries between different types; still, it shows a fraction of the great complexity and variety that characterize the cometary motion.

1.2. Characteristic cometary activity

The composition of comets varies greatly from case to case, as it depends on their original states in the Oort Cloud or Kuiper Belt and from the history of their activity: comets that have passed multiple times near the Sun may have already depleted some of the chemical species inside their nuclei. Still, a somehow common pattern can be described in terms of physical processes occurring at various heliocentric distances. When orbiting at distances greater than approximately 6 AU, comets are inactive bodies made of volatile ices, rocky materials, and trapped dusty components. The heat flux coming from the Sun and from other sources is not sufficient to start any activity, and they cannot be directly observed. When the heliocentric distance decreases, the incoming thermal radiation progressively heats the surface layers of the nucleus, leading to the sublimation of ices, and the cometary activity becomes observable and measurable. At distances $\gtrsim 3$ AU subsurface sublimation of CO or CO_2 begins to be noticeable, and a faint coma develops. When the distance becomes lower than 3 AU, solar driven H_2O sublimation becomes dominant, and comets with high enough gas production rates exhibit the typical features:

- a bright solid nucleus, with very complex shapes, from which gases and dust grains are emitted;
- a dusty expanding atmosphere, called coma, where gas and dust interact with each other;
- an outer dust coma where the gas is extremely rarefied and the dynamics is uncoupled;
- the characteristic dust and ion tails, that interact with the solar wind and gravity.

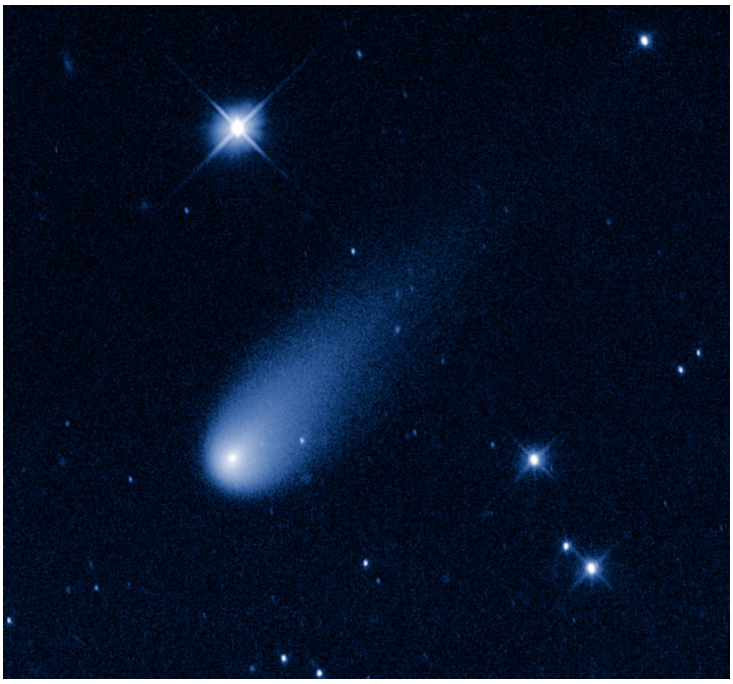


Figure 1.3: Image of comet ISON exhibiting the typical cometary activity, from [34].

1.3. The dusty gas coma

The first basic physical concepts concerning the nucleus activity and the dusty coma were formulated by Whipple in 1951[56], where he introduced his famous "dirty snowball" model. He assumed cometary nuclei to be:

- solid objects, not sand banks;
- made of a mixture of ices and dusty grains;
- submitted to sublimation at low heliocentric distances, due to solar illumination;
- ejecting dust grains, due to the drag of gas.

This model became famous as one of the building blocks of cometary activity understanding, before the spacecraft era began. Later, in 1968, Probstein [41] improved this work and set the foundations for the modern modeling of cometary comae. He first solved the Euler equations for a mixture of gas and dust, assuming:

- dust-gas coupling with free-molecular momentum and heat exchange;
- an isothermal and spherical nucleus;
- spherical and uniform-sized dust grains.

He first concluded that the cometary coma is formed by a supersonically expanding flow of gas, which issues at sound velocity from the surface if free from dust, but, when the surface dust-to-gas mass loss ratio is significant, the flow starts subsonically and undergoes a sonic transition near the surface. In the later decades, Computational Fluid Dynamics (CFD) codes were developed, and spacecraft technology saw a massive leap forward. On March 14, 1986, Giotto spacecraft passed 596 km from the nucleus of comet 1P/Halley and returned high-resolution images of the coma, as reported in Figure 1.4.

Figure 1.4: Image of comet Halley seen from ESA's Giotto spacecraft [12].

The studies and simulations then focused on one hand in determining which physical model for the coma gasdynamics were the most appropriate, and on the other hand started assessing the influence of how the complex geometry of the nucleus and the localization of emitting jets could influence the resulting coma. Papers [9, 20, 22] provide examples of the complex shock structures that can appear in cometary comae, with only slight deviations from assumptions of spherical nucleus and uniform, homogeneous gas emissions.

1.3.1. Models for the cometary atmosphere

As already mentioned, cometary atmospheres are formed by gases sublimating from the nucleus surface, and due to the very low gravity of the nucleus, they expand into vacuum up to scales several order of magnitudes larger than the nucleus charateristic dimensions. For a typical active comet in range of a few tenths to $\simeq$ 2AU from the Sun, the gas

flow undergoes strong temporal and spatial variations, in which many different regimes can coexist. At the very surface of the nucleus, where the volatile molecules undergo sublimation, there is a boundary layer, known in literature as "Knudsen layer", in which the gas is not in thermal equilibrium and its distribution is not close to a Maxwellian one. Inside this layer, that can extend from a few meters up to few hundreds of meters, depending on the heliocentric distance and on the strength of the sublimation, the gas molecules undergo thermalization by collision. In Paper [53], for comet 67P/Churyumov-Gerasimenko, the authors estimated the Knudsen layer thickness to vary from $\simeq 5\,\mathrm{m}$ at 1.29 AU up to 4 km at 3.25 AU. This thermal layer then develops rapidly into a transonic dusty-gas flow, with typical length scales of hundreds of meters. Then, up to scales of $\simeq$ 100km expansion and adiabatic cooling dominates the evolution of the flow, which is now highly supersonic. At these heights the gas can reach velocities of $\simeq 700\,\mathrm{m\,s^{-1}}$ and temperatures lower than 30 K. During this outflow, water dominates up to 90 % of the volatile species, but due to photodissociation, and to a much smaller extent through photoionization and dissociation, the parent H_2O molecule is destroyed, leading to the creation of daughter molecular products, OH and H_2, atomic products, O and H, and ion species. Inside the mentioned layers, these reactions do occur but at a slower rate, so they are often neglected in numerical simulations. Further outside, energetic products of solar UV dissociation begin to dominate the flow, and to heat it faster than it can cool, leading to an increase in both temperature and speed. Photoionization and charge impact ionization form a cometary ionosphere extending up to $10^3 - 10^4$km, which, interacting with the solar wind, forms the typical ion tail at distances of $10^5 - 10^7$km from the nucleus. Lastly, at heights of $10^6 - 10^7$km from the surface, the neutral coma is completely broken down into its atomic constituents O, H, C, N.

Such a complex environment means that, in order to fully model the coma, one has to cover a wide range of regimes: from a collision-dominated gas in the innermost regions to free-molecular and non-equilibrium conditions. In this scenario, the Boltzmann Equations (BE) constitute the model that can act as the common substrate and can describe all the cometary coma regions.

$$\frac{\partial f_i}{\partial t} + \mathbf{v}_i \cdot \nabla f_i + \frac{\mathbf{F}_i}{m_i} \cdot \nabla_{\mathbf{v}} f_i = \sum_j I(f_i, f_j) \tag{1.2}$$

Equation 1.2 reports the BE to be solved for each of the distribution functions $f_i = f_i(\mathbf{r}, \mathbf{v}_i, t)$ of particles of the kind i. In case of a monoatomic gas undergoing only elastic collisions with spherically symmetric interaction potential, the collision operator takes the form:

$$I(f_i, f_j) = \iiint \left(f_i' f_j' - f_i f_j\right) |\mathbf{v}_i - \mathbf{v}_j| \, \eta_c \, d\eta_c \, d\psi \, d\mathbf{v}_j \tag{1.3}$$

Equations 1.2 and 1.3 constitute, even in case of a single distribution function, a set of extremely complex integro-differential equations. As Papers [7, 47] show and discuss, under assumptions on the level of rarefaction of the medium, it is possible to derive different sets of fluid equations describing a continuum. Assuming a strict Maxwellian distribution function for the gas, pressure becomes a scalar quantity, and the first order approximation of the BE gives the Euler Equations (EE) for an inviscid compressible fluid. When the gas distribution function deviates from being Maxwellian, the stress tensor is not diagonal anymore (equivalent to the isotropic pressure), and effects such as viscosity and heat conductivity appear. Thus the Euler Equations become invalid and it is needed to consider the Navier-Stokes Equations (NSE), which corresponds to a first-order distortion in the distribution function. In order to solve the Boltzmann Equations, the Direct Monte-Carlo Simulation (DSMC) method was developed. It allows to numerically simulate non-continuum gas flows. However, being this method still computationally heavy, many authors had studied under which conditions the cometary coma could be well described by a continuum hydrodynamic approach. Works [6, 8, 9, 27] discuss the limits of validity of the fluid approximation inside the coma, with varying initial conditions and levels of cometary activity. An important parameter in these studies is the Knudsen number:

$$Kn = \frac{\lambda}{L} \tag{1.4}$$

where λ represents the mean free path of the molecules and L is a characteristic dimension of the flow, traditionally the equivalent radius of the nucleus. To characterize local effects, in Equation 1.4 the scale length of macroscopic gradients can also be used. Depending on the local Kn, three flow regimes can be identified:

1. continuum fluid regime: $Kn < 0,01$;
2. transitional regime: $0.01 < Kn < 100$;
3. free molecular regime: $Kn > 100$.

Figure 1.5 provides a visual representation of how flows should be modeled, depending on the level of rarefaction.

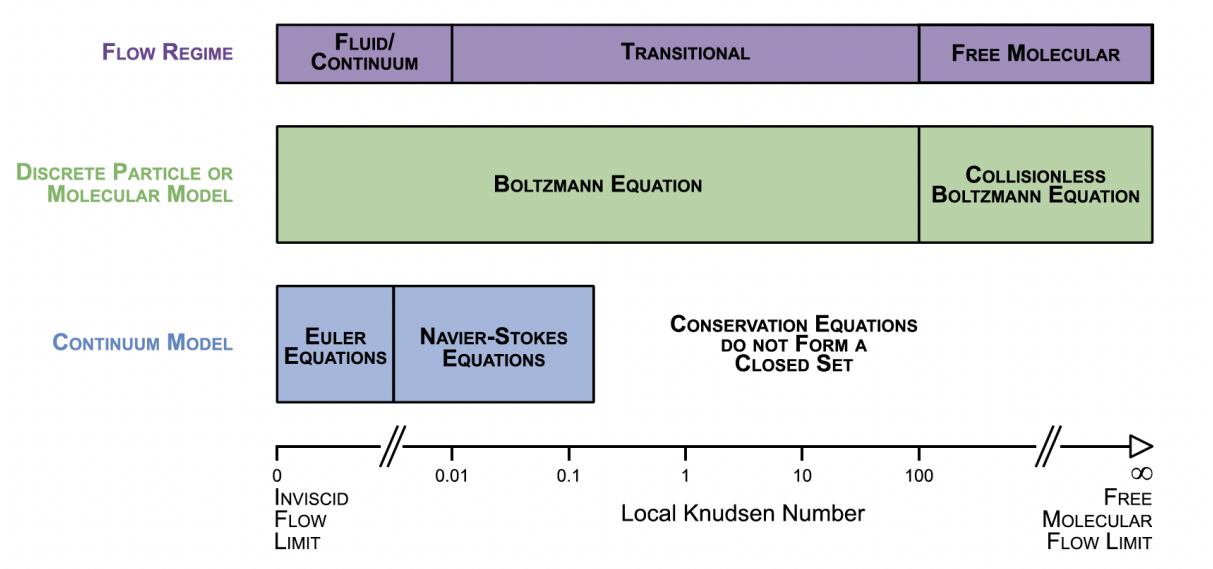


Figure 1.5: Flow regimes and valid models as functions of the Knudsen number, from [27].

Excluding the surface Knudsen layer, which is always to be treated with a full kinetic theory, the literature agrees that a fluid representation can provide a physically adequate description of the inner gas coma, provided that consistent boundary conditions are formulated to link the parameters on the surface to the top of the initial boundary layer. Therefore, inside the collision dominated region the continuum fluid approach can be used successfully. When present, such a region can extend from tens of kilometers up to $\simeq 10^3$km, depending on the specific comet. Both Euler Equations and the Navier-Stokes Equations are thus able to give a proper description of the circumnuclear coma environment, where the gas starts subsonic and is accelerated to supersonic conditions, although the accuracy level of the DSMC method is usually superior.

Figures 1.6 and 1.7, reported from [8], show the comparison between the DSMC and NSE solutions for a Hale-Bopp class spherical homogeneous nucleus at heliocentric distance of 1 AU. The Sun is placed horizontally on the far right of the nucleus. The two solutions show excellent agreement in this condition, both on the dayside and on the nightside.

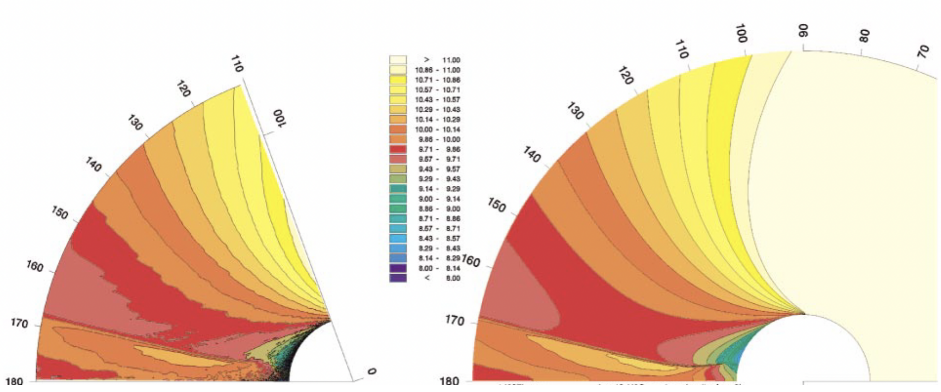


Figure 1.6: Comparison between DSMC and NSE solutions for gas number density.

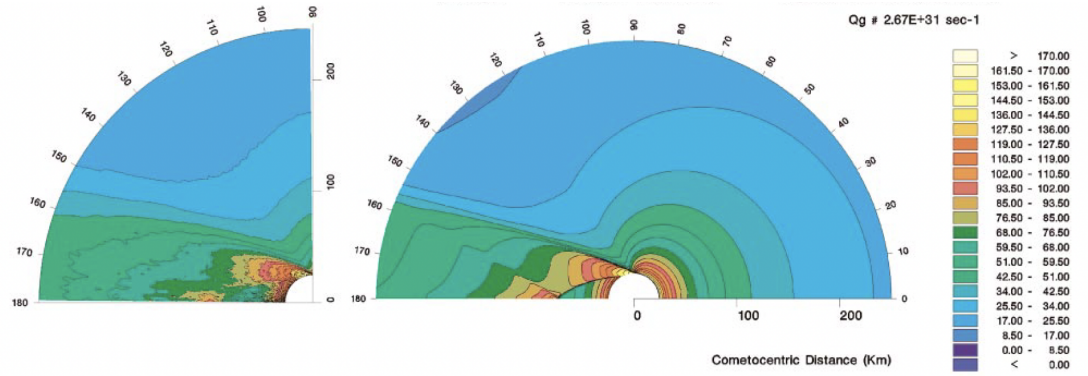


Figure 1.7: Comparison between DSMC and NSE solutions for gas temperature.

Additionally, when modeling the inner regions of the atmosphere, the importance of other physical phenomena is relatively low: for instance gasdynamics simulations of the near nucleus region can avoid dealing with fictitious forces arising from non-inertial reference frames, and can neglect photodissociation and other chemical reactions. Only when dealing with regions extending more than 10^4km it becomes necessary to include these processes, as their timescales, which are on the order of $10^5 - 10^7$s, become comparable to the timescales of the gas flow.

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = f_\rho, \\ \dfrac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) + \nabla p - \nabla \cdot \bar{\boldsymbol{\tau}} - \mathbf{F} = \mathbf{f}_v, \\ \dfrac{\partial (\rho h_0 - p)}{\partial t} - \nabla \cdot \mathbf{q} + \nabla \cdot (\bar{\boldsymbol{\tau}} \mathbf{v}) - \mathbf{F} \cdot \mathbf{v} = f_h \end{cases} \tag{1.5}$$

Equations 1.5 represent the form of NSE that is used the most in single fluid coma studies.

ρ is the gas density, h_0 is the specific enthalpy, p is the pressure, $\mathbf{q}$ is the heat conduction vector, $\bar{\boldsymbol{\tau}}$ represents the stress tensor and $\mathbf{F}$ the total macroscopic force. The term $\mathbf{F}$ is where the radiation pressure and gravitational effects can be introduced in the equations, but, as mentioned above, they are usually neglected even if one could include them to study with more detail their relative importance. Inertia forces do not appear in the equations if an inertial frame of reference is used. However, if using a non inertial frame, for instance one attached to the nucleus, these forces may become more and more relevant the more extended is the simulation domain considered. For instance, in [10] it is shown that, for very large comets with fluid regions reaching heights in the order of 10^4km, like Hale-Bopp, at such distance the ratio of pressure forces to the Coriolis force, assuming a nucleus angular velocity $\Omega = 10^{-5}\mathrm{rad\,s^{-1}}$, would be in the order of $1/10$: the latter would be a dominant factor. The "source-sinks" terms $f_p, \mathbf{f}_v, f_h$ at the right hand-side of Equation 1.5 allow to include effects coming from interactions of the fluid with different particles, for example dust, or photons, and to express possible inelastic processes internal to the gas that may influence its momentum and energy evolution. For instance, these term can represent the photodissociation of H_{20} or cooling through IR emission. The Euler Equations, which will be the focus of this work, can be obtained by setting $\mathbf{q} = \mathbf{0}$ and $\bar{\boldsymbol{\tau}} = \bar{\mathbf{0}}$. Despite being simpler than the full NSE, they are still used today in numerical simulations on inner dusty comae (see [28]).

1.3.2. Dust-gas interaction

The cometary atmosphere in not composed by gas only, but, along with sublimating molecules, a wide range of solid dust grains is ejected as well. For instance, when considering simulations on the dust populations of comet 1P/Halley, it has been estimated and measured that they can have sizes ranging from the order of 1 cm to 10^{-6}cm. Once lifted from the surface, several forces influence the motion of dust particles, and in general three dynamical regions can be identified:

1. a coupled coma region: here the dust dynamics is ruled by forces related to the nucleus, such as its gravitational attraction and the gas drag force. Because the gas flow can be strongly non-radial, even considering a spherical nucleus, the dust motion is also generally non-radial, converging to radiality as the distance from the surface increases. The size of this region can extend to approximately 10 radii;

2. a transitional coma region: here the nucleus related forces start losing relative importance, and solar forces like solar gravity and solar radiation pressure gradually become dominant. Gas and dust reach a complete uncoupling. The largest particles

may still be dominated by the nucleus gravity, and a separation in the dust coma can occur, with lighter grains escaping the field of the comet, while the largest ones can remain bound to it into local ballistic orbits, until eventually decaying back to the surface. This region extends in height approximately to the comet's Hill sphere;

3. the dust tail and trail: here dust particles do not feel anymore the comet influence, and their dynamics is governed purely by solar gravity and radiation pressure.

In principle, the dust presence can affect the gas flow in at least three different ways. First, it could be heated up to temperatures higher than the gas temperature, leading to local heating and potentially to alterations of the molecular distribution function. Secondly, if the dust-to-gas ratio reaches sufficient levels, the kinetic energy needed to accelerate the solid grains can become not negligible anymore, and can lead to a slowdown of the flow. Lastly, dusty grains may not be ejected from the surface completely dry and their icy content might undergo sublimation, producing an extended (or distributed) gas source, thus altering both the gas flow and the dust size distribution. Figure 1.8 provides a visual summary of the processes influencing the dust dynamics.

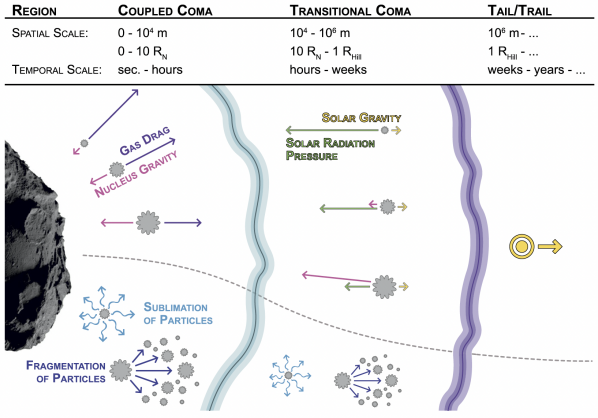


Figure 1.8: Schematic representation of different dynamical regions for dust, from [27].

In the literature two main approaches are used to model the dust flow and its interaction with the gas:

- the multifluid approximation: each size population of dust particles is represented by an individual set of Euler Equations complemented by source terms, that include the gas drag effect and the thermal heating of the solid grains. This approach leads to the need to solving $N_f + 1$ sets of EE: one for the gas itself and N_f for the dust fluids. While it may have computational advantages, this approach has the drawback that different dust flows cannot interact and cross each other. Therefore,

it may be needed to define multiple dust fluids even for the same size population, if identical grains are emitted from different surface locations. Rodionov and Crifo, in [47], provide a detailed description of this approximation.

- the test particle approach: the equations of motion for solid dust grains are written, and a set of test particles is generated and evolved through the gas flow.

The latter approach has been the most used in literature. It is usually complemented by the assumption of dust not affecting the gas, which is true under the following two conditions:

1. The cometary coma is gas dominated: true for very active comets such as 1P/Halley at small heliocentric distances, where the energy stored in the dust coma is a few percent of the energy within the gas coma.
2. No relevant distributed sources: dust grains are ejected almost completely dry, and they do not provide additional gas molecules due to sublimation when away from the nucleus surface. For comet 67P/CG there is no evidence of this phenomenon occurring (see [17]).

1.3.3. A simple model of the inner cometary coma

As a consequence of the previous discussion, the following simplifying, but still physically adequate, assumptions can be made (discussed in the cited literature and also in [26]):

- the cometary atmosphere is dominated by a single volatile species, H_2O, and secondary gaseous components can be neglected;
- the gas flow is in local thermal equilibrium: this allows to use the fluid representation;
- the gravitational field of the comet affects only the dust grains, being too weak to influence the gas flow;
- for sufficiently small simulation scales, the dust and gas velocities are high enough that the nucleus rotation can be neglected;
- dust-dust interactions can be neglected during average cometary activity;
- the dust is accelerated by the gas force, but the back reaction is negligible: the gas flow can be modeled without taking into account the presence of dust.

A simple cometary coma model can then consider the nucleus to be spherical, and to define a circular active region from which the ice sublimates and dust grains are also

ejected. First, the Euler Equations for the gas flow are solved, and then the dynamics of dust grains can be evaluated. Considering spherical dusty elements, the drag exerted by the gas can be expressed using free-molecular aerodynamics, while aerodynamic lift and torque would not need to be considered. The equation of motion for a single particle, in the innermost coma, where solar radiation pressure can not be taken into account, would then be:

$$m_d \frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2} = \mathbf{f}_G + \mathbf{f}_D = \mathbf{f}_G + \frac{3}{8}\frac{C_D \rho_g}{a \rho_d} |\Delta\vec{v}| \, \Delta\vec{v} \tag{1.6}$$

Equation 1.6 combines the gravitational attraction of the comet with the aerodynamic drag (see [39]). The latter force depends on the ratio ρ_g/ρ_d between the local gas density and the bulk dust density, on the grain size a, and on the relative velocity between the two components. C_D represents the drag coefficient, and under free-molecular aerodynamics its exact expression reads (see [4]):

$$C_D = \frac{2\zeta^2 + 1}{\sqrt{\pi}\,\zeta^3} e^{-\zeta^2} + \frac{4\zeta^4 + 4\zeta^2 - 1}{2\zeta^4} \operatorname{erf}(\zeta) + \frac{2(1-\varepsilon)\sqrt{\pi}}{3\zeta}\sqrt{\frac{T_d}{T_g}} \tag{1.7}$$

where ζ is defined as:

$$\zeta = \frac{|\mathbf{v}_g - \mathbf{v}_d|}{\sqrt{\dfrac{2k_b T_g}{m_g}}} \tag{1.8}$$

Equations 1.7 and 1.8 are complex functions of the gas temperature T_g, the dust temperature T_d and the fraction of specular reflection ε. However, it can be shown that C_D converges to 2 as $\zeta \to \infty$. To keep the model simple, in a first approximation, a constant or linearized version of C_D can be used. The motion of a dust particle could be obtained by integrating equations 1.6, after the surrounding gas field has been computed. Overall, this would be a simple simulation but would still retain physical meaning, and could serve as the foundation upon which more complex effects are progressively included at later stages.

2 | Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) is a Lagrangian mesh-free numerical method for the approximation and solution of partial differential equations. It was originally developed for astrophysical applications by Gingold and Monaghan (1977, [18]) and independently by Lucy (1977 [25]). The basic idea consists of approximating the fluid as a discrete set of material particles which transport mass, momentum and energy, whose motion is governed by ordinary differential equations (ODE), derived from the continuum partial differential equations (PDE). For this reason, the method is referred to as Lagrangian: the fluid field is computed by following the motion of each individual particle. Unlike traditional Eulerian methods, SPH does not use a grid to discretize the spatial domain. Discretization is obtained directly at particle level by means of a local process of interpolation. Although designed with astrophysical problems in mind, SPH now finds application in a wide range of problems, ranging from hydrodynamics to fracture and elasticity of solid bodies. This chapter introduces the main ideas of the method, the most relevant equations, and discusses its advantages and disadvantages, with a specific focus on the application to the Euler Equations for inviscid compressible fluids. Additional information and a deeper discussion can be found in reviews [31, 32, 50] and in books [24, 55].

2.1. SPH interpolation

The concept of integral interpolation in SPH is based on the following identity:

$$f(\mathbf{x}) = \int_{\Omega} f(\mathbf{x}')\delta(\mathbf{x} - \mathbf{x}')\, d\mathbf{x}' \tag{2.1}$$

where f is a generic function of the three-dimensional vector $\mathbf{x}$, and Ω is the volume of the space considered. The integral representation in equation 2.1 is exact, provided that $f(\mathbf{x}$ is regular enough in Ω. $\delta(\mathbf{x} - \mathbf{x}')$ is the Dirac delta distribution, which can be intuitively

thought as:

$$\delta(\mathbf{x}-\mathbf{x}') = \begin{cases} 1 & \text{if } \mathbf{x} = \mathbf{x}' \\ 0 & \text{if } \mathbf{x} \neq \mathbf{x}' \end{cases}$$

The first basic idea in SPH consists in replacing the Dirac delta with a smoother function, in order to get an approximated integral representation:

$$f_I(\mathbf{x}) = \int_\Omega f(\mathbf{x}')W(\mathbf{x}-\mathbf{x}', h)\, d\mathbf{x}' \tag{2.2}$$

In the SPH literature $W(\mathbf{x}-\mathbf{x}', h)$ is referred to as *smoothing kernel function*, *smoothing function*, *kernel function*, or simply *kernel*. In the kernel function, the parameter h, labelled as *smoothing length*, defines the characteristic width of W. Notice that $f_I(\mathbf{x})$ in equation 2.2, reproduces $f(\mathbf{x})$ exactly only if W is the Dirac delta function, otherwise f_I is only an approximation. This is the origin of the term *kernel approximation*, often found in SPH related works. To apply this integral interpolation to a fluid, let's suppose to divide it into a set of small mass elements, each having a mass m_i, density ρ_i and position $\mathbf{x}_i$. The interpolation integral of the function $f(\mathbf{x})$ can be rewritten as:

$$f_I(\mathbf{x}) = \int_\Omega \frac{f(\mathbf{x}')}{\rho(\mathbf{x}')} W(\mathbf{x}-\mathbf{x}', h)\, \rho(\mathbf{x}') d\mathbf{x}' \tag{2.3}$$

where $\rho d\mathbf{x}'$ represents a mass.

The second fundamental step consists in approximating the integral in equation 2.3 by a summation over the mass elements. This gives the *summation interpolant*:

$$f_s(\mathbf{x}) = \sum_j m_j \frac{f_j}{\rho_j} W(\mathbf{x}-\mathbf{x}_j, h) \tag{2.4}$$

With $f_j = f(\mathbf{x}_j)$. Theoretically, the summation in equation 2.4 should include all particles, but in practice, as will be discussed later, it considers only the closest neighbors, as W drops rapidly with distance. So, the principal idea of SPH consists in considering this two-step approximation process, in order to represent the value of the function f at particle i as:

$$f_i \simeq \sum_j m_j \frac{f_j}{\rho_j} W_{ij} \tag{2.5}$$

where

$$W_{ij} = W(\mathbf{x}_i - \mathbf{x}_j, h)$$

Equation 2.5 states that the value of a function at particle i is approximated using the average value of that function at all the neighboring particles, weighted by the smoothing function. From now on, it will be assumed that all equations related to this *particle approximation* process bring some kind of approximation, and the symbol $\simeq$ will be replaced by the ordinary $=$. As a final note to this introduction, it should be mentioned that if one assumes f to be the density ρ, the interpolation procedure gives the following estimate for density at point $\mathbf{x}$:

$$\rho(\mathbf{x}) = \sum_j m_j W_{ij} \tag{2.6}$$

Equation 2.6 shows how the mass of a set of particles, which is a constant property, is smoothed to produce the density field. This equation is sometimes used in SPH schemes to update the density value of the particles, without using the differential mass conservation equation. It should be noted that the mass of the system is always conserved, independently of the specific SPH scheme, because it is a constant property tied to the particles.

2.2. Smoothing kernel functions

In SPH, the choice of a proper kernel function is of utmost importance, because it influences the pattern for the function approximation, it defines the maximum distance within which particles do influence each other, and it determines the consistency of both the integral and the particle approximation. In order to be a suitable candidate, a smoothing function should satisfy the following requirements:

1. *Unity*: the kernel function should be normalized:

$$\int_{-\infty}^{+\infty} W(\mathbf{x} - \mathbf{x}', h) d\mathbf{x}' = 1$$

   This property allows constants to be interpolated exactly.

2. *Compact support*: the kernel function should have a finite compact support:

$$W(\mathbf{x} - \mathbf{x}', h) = 0 \quad \text{for} \quad |\mathbf{x} - \mathbf{x}'| > \kappa h \tag{2.7}$$

the dimension of the compact support is thus determined by the value of the smoothing length h and by a scaling factor κ. Having a compact support means that the summations in SPH for a generic particle i, such as in equation 2.6, despite being formally extended over all particles, are actually limited to a finite set of neighbors for which $|\mathbf{x}_i - \mathbf{x}_j| \leq \kappa h$.

3. *Positivity*: for any point $\mathbf{x}'$ inside the compact support of the particle at point $\mathbf{x}$ it should be:

$$W(\mathbf{x} - \mathbf{x}', h) \geq 0$$

this condition is not mathematically necessary for convergence requirements, but ensures a physically meaningful representation of some physical phenomena. For instance, if this was not satisfied, the interpolated density could potentially result in a negative value.

4. *Decay*: the smoothing function value for a particle at $\mathbf{x}$ should decrease monotonically as the distance from that particle increases. This ensures that closer particles have more influence on particle i than ones that are farther away.

5. *Delta function*: the kernel function should approach the Dirac delta function as the smoothing length h decreases:

$$\lim_{h \to 0} W(\mathbf{x} - \mathbf{x}', h) = \delta(\mathbf{x} - \mathbf{x}')$$

6. *Symmetric property*: the kernel function should be an even function. In this way particles at the same distance from the target particle i, but at opposite orientations, will produce the same effect.

7. *Smoothness*: the smoothing function should be sufficiently smooth for the problem under consideration.

Any function satisfying the properties can be chosen as kernel, and many different kinds of smoothing functions have been tested. Some of the frequently used ones, that are implemented in the available open-source SPH solvers, revied in the next chapter 3.4, are now reported. It should also be noted that, whatever the specific expression of $W(\mathbf{x}-\mathbf{x}', h)$ is, the dimension of a kernel function is always the inverse of a volume, $[\frac{1}{V}]$

2.2.1. Examples of kernel functions

The most commonly used kernels are originally based on Schoenberg M_n splines, which are piece-wise continuous functions with compact support, having continuous derivatives

up to order $(n-2)$. Among them, the *cubic spline* function, introduced by Monaghan and Lattanzio [33], is probably the most commonly used kernel in SPH computations. Defining $q = \frac{|\mathbf{x}|}{h}$, this function is built as a combination of cubic polynomials:

$$W(q,h) = \alpha_d \begin{cases} \frac{2}{3} - q^2 - \frac{1}{2}q^3 & 0 \leq q < 1 \\ \frac{1}{6}(2-q)^3 & 1 \leq q < 2 \\ 0 & q \geq 2 \end{cases} \tag{2.8}$$

where α_d is a normalization coefficient that depends on the dimensionality of the problem:

$$\alpha_d = \begin{cases} \frac{1}{h} & d = 1 \\ \frac{15}{7\pi h^2} & d = 2 \\ \frac{3}{2\pi h^3} & d = 3 \end{cases}$$

As equation 2.8 shows, $\kappa = 2$ for the cubic spline function, thus neighboring contributions are relevant only up to a distance of $2h$. This kernel is one of the most popular, due to its similarity to a Gaussian function, but with a compact support. Its second derivative is a piecewise linear function, so, in some applications, it can have stability properties that are inferior to higher order kernels. Despite that, it is a suitable choice when solving the Euler Equations numerically.

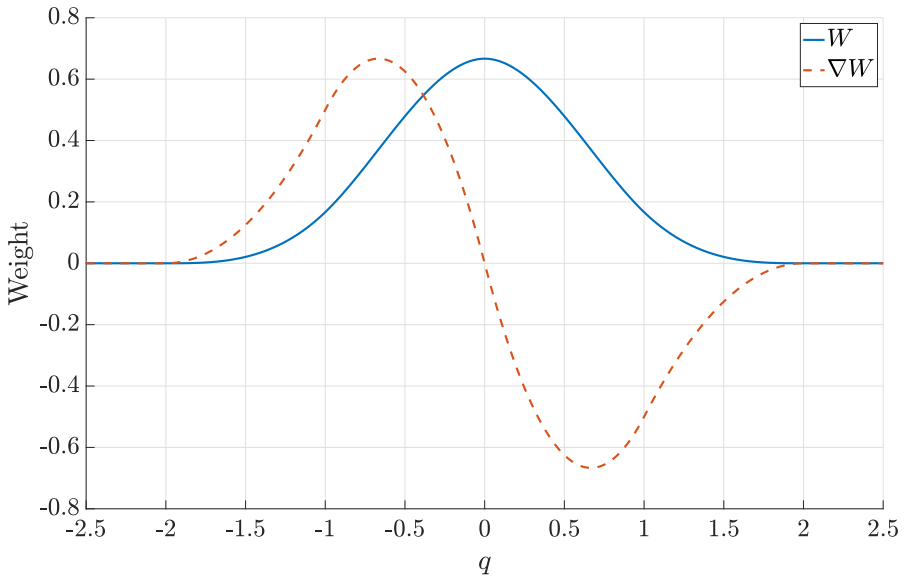


Figure 2.1: The cubic spline kernel and its gradient, with $\alpha_d = 1, h = 1$.

Higher order splines can be built as well. They more closely resemble the shape of a Gaussian function. The *quintic spline* is:

$$W(q) = \alpha_d \begin{cases} (3-q)^5 - 6(2-q)^5 + 15(1-q)^5, & 0 \leq q < 1, \\ (3-q)^5 - 6(2-q)^5, & 1 \leq q < 2, \\ (3-q)^5, & 2 \leq q < 3, \\ 0, & q \geq 3. \end{cases} \tag{2.9}$$

with α_d:

$$\alpha_d = \begin{cases} \frac{1}{120h} & d = 1 \\ \frac{7}{478\pi h^2} & d = 2 \\ \frac{3}{359\pi h^3} & d = 3 \end{cases}$$

Even though this function has a larger support, with $\kappa = 3$, its derivatives are smoother than the cubic B-spline ones, so it may be a relevant alternative to it.

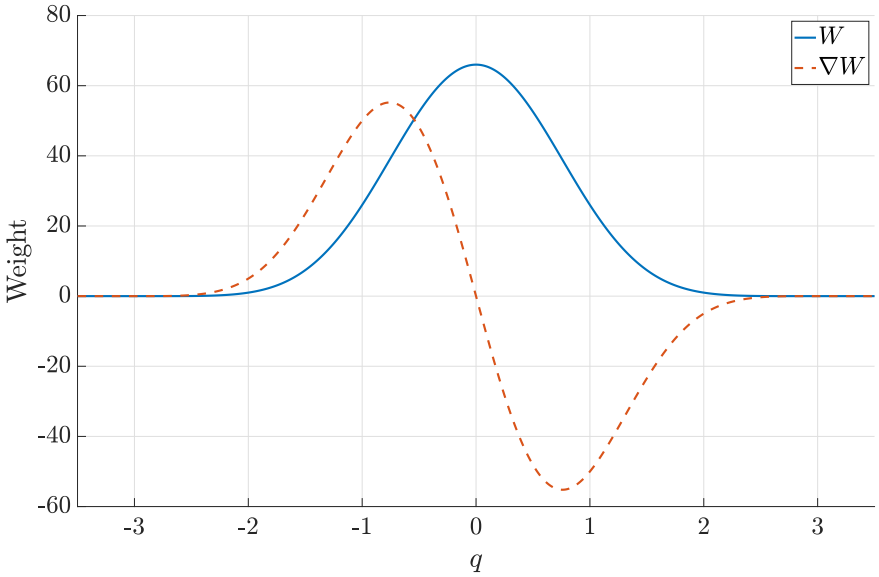


Figure 2.2: The quintic spline kernel and its gradient, with $\alpha_d = 1, h = 1$.

Another example of a common high order smoothing function is the *Wendland* C^2 kernel.

Its expression reads:

$$W(q) = \alpha_d \begin{cases} \left(1 - \frac{q}{2}\right)^4 (2q+1), & 0 \leq q < 2, \\ 0, & q \geq 2. \end{cases} \tag{2.10}$$

with the normalization coefficient α_d depending again on the dimensionality:

$$\alpha_d = \begin{cases} \frac{5}{8h} & d = 1 \\ \frac{7}{4\pi h^2} & d = 2 \\ \frac{21}{16\pi h^3} & d = 3 \end{cases}$$

This kernel has order five exactly as the quintic spline kernel of equation 2.9, but has a narrower spatial extension being $\kappa = 2$, and it is a valid alternative for application in compressible flows. For all the listed kernels, their spatial gradient, or the derivative in 1D, can be computed by making use of the chain rule:

$$\nabla_x W = \frac{\partial W}{\partial \mathbf{x}} = \frac{\partial W}{\partial q}\frac{\partial q}{\partial \mathbf{x}} = \frac{\partial W(q)}{\partial q}\frac{1}{h}\frac{\mathbf{x}}{|\mathbf{x}|} \tag{2.11}$$

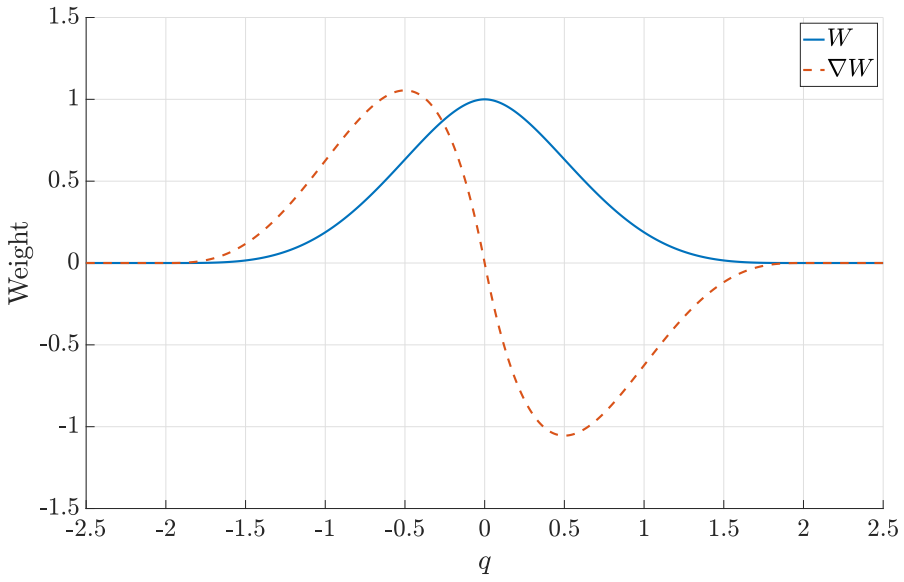


Figure 2.3: The Wendland C^2 kernel and its gradient, with $\alpha_d = 1, h = 1$.

2.3. SPH approximation of derivatives

In SPH, the derivatives of interpolated quantities can be estimated easily. If the kernel function W is differentiable, then one can apply the derivative operator directly onto equation 2.4:

$$\frac{\partial f_s}{\partial x} = \sum_j m_j \frac{f_j}{\rho_j} \frac{\partial W}{\partial x} \tag{2.12}$$

So, in SPH, the derivative is found by applying an exact differentiation to an approximated function. The issue with this formulation is that $\frac{\partial f_s}{\partial x}$ does not vanish if f is a constant function. One option to make sure this happens consists in introducing another arbitrary differentiable function Φ, and write the following identity:

$$\frac{\partial f}{\partial x} = \frac{1}{\Phi}\left(\frac{\partial (\Phi f)}{\partial x} - f\frac{\partial \Phi}{\partial x}\right) \tag{2.13}$$

Using equation 2.12 to express the derivatives in the right hand side of equation 2.13, one obtains its corresponding SPH formulation, where the subscript s has been dropped:

$$\left(\frac{\partial f}{\partial x}\right)_i = \frac{1}{\Phi_i} \sum_j m_j \frac{\Phi_j}{\rho_j} (f_j - f_i) \frac{\partial W_{ij}}{\partial x_i} \tag{2.14}$$

This form vanishes if f is constant. The notation $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$, $W_{ij} = W(\mathbf{x}_{ij}, h)$, $\nabla W_{ij} = \nabla W(\mathbf{x_{ij}})$ will be used from now on. $\left(\frac{\partial}{\partial x}\right)_i$ is to be intended as the derivative with respect to the coordinates centered at particle i. Different choices of Φ give all the various expressions of derivatives available in literature. The most commonly used is the form with $\Phi = \rho$:

$$\left(\frac{\partial f}{\partial x}\right)_i = \frac{1}{\rho_i} \sum_j m_j (f_j - f_i) \frac{\partial W_{ij}}{\partial x_i} \tag{2.15}$$

Equation 2.15 can then be extended to the gradient operator:

$$\nabla f_i = \frac{1}{\rho_i} \sum_j m_j (f_j - f_i) \nabla W_{ij} \tag{2.16}$$

The usual SPH expression for the divergence operator, applied to the velocity field $\mathbf{v}(\mathbf{x})$, is obtained by considering the differential identity $\rho \nabla \cdot \mathbf{v} = \nabla (\rho \mathbf{v}) - \mathbf{v} \cdot \nabla \rho$. If the right

hand side terms are expressed with SPH gradients, their difference leads to:

$$(\nabla \cdot \mathbf{v})_i = \frac{1}{\rho_i} \sum_j m_j \left(\mathbf{v}_j - \mathbf{v}_i\right) \cdot \nabla_j W_{ij} \tag{2.17}$$

This formulation, among all ones that can be obtained, has the advantage of actually vanishing if all particle velocities are equal, and turns out to be more accurate than the others, when tested in practice.

2.4. SPH discretization of the Euler Equations

The Euler Equations for an inviscid, compressible fluid in a Lagrangian frame, so moving with the fluid, are given by:

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} + \rho \nabla \cdot \mathbf{v} = 0 \tag{2.18}$$

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} + \frac{\nabla P}{\rho} = \mathbf{g} \tag{2.19}$$

$$\frac{\mathrm{d}u}{\mathrm{d}t} + \frac{P}{\rho} \nabla \cdot \mathbf{v} = 0 \tag{2.20}$$

Equations 2.18, 2.19, 2.20 express the conservation of mass, momentum, and energy, being ρ the density, $\mathbf{v}$ the velocity, P the pressure, and $\mathbf{g}$ the body force per unit mass. The time derivative is the derivative following the motion, also known as the material derivative in Eulerian formulations, defined as $\frac{\mathrm{d}}{\mathrm{d}t} = \frac{\partial}{\partial t} + \mathbf{v} \cdot \nabla$.

2.4.1. The SPH mass conservation equation

The SPH approximation of equation 2.18 for particle i is obtained directly by using the expression for the divergence reported by 2.17:

$$\frac{\mathrm{d}\rho_i}{\mathrm{d}t} = \sum_j m_j \left(\mathbf{v}_i - \mathbf{v_j}\right) \cdot \nabla_i W_{ij} \tag{2.21}$$

In some SPH schemes, the density is not actually evolved according to this differential equation, but it is updated at each time step, using the summation equation for density, Eq. 2.6. Either choice has its own advantages: the differential equation provides a simpler initialization phase, and is more capable of withstanding possible deficiencies in neighbors, when a particle finds itself close to an interface. However, depending on the problem, it can be beneficial to compute the density by interpolating the mass distribution, to obtain

a better accuracy. The specific treatment of the fluid density is a choice to be made when designing a specific SPH scheme.

2.4.2. The SPH momentum equation

In the original work, published in 1977, the momentum equation 2.19 was discretized by considering the SPH version of the pressure gradient, that leads to:

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = -\frac{1}{\rho_i}\sum_j m_j \frac{P_j}{\rho_j}\nabla_i W_{ij}$$

However, it turns out that this expression does not conserve linear nor angular momentum exactly: the force acting on particle i due to particle j is not equal and opposite to the force acting on j because of i. This is a violation of Newton's third law. A suitable form is obtained by considering that:

$$\frac{\nabla P}{\rho} = \nabla\left(\frac{P}{\rho}\right) + \frac{P^2}{\rho}\nabla\rho$$

Using again the SPH rules for the differential operators the momentum equation for particle i is:

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = -\sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right)\nabla_i W_{ij} \tag{2.22}$$

It can be shown that equation 2.22 does conserve exactly linear and angular momentum, and the forces acting on particles i and j are indeed equal and opposite.

2.4.3. The SPH energy equation

Using the SPH expression for $\nabla\cdot\mathbf{v}$ given by 2.17, equation 2.20 becomes:

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = \frac{P_i}{\rho_i^2}\sum_j m_j\left(\mathbf{v}_i - \mathbf{v}_j\right)\cdot\nabla_i W_{ij} \tag{2.23}$$

If one considers a different SPH expression for the velocity divergence:

$$\left(\nabla\cdot\mathbf{v}\right)_i = \sum_j \frac{m_j}{\rho_j}\mathbf{v}_j\cdot\nabla_i W_{ij}$$

the conservation of energy becomes:

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = \frac{P_i}{\rho_i} \sum_j \frac{m_j}{\rho_j} (\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla_i W_{ij} \tag{2.24}$$

To resolve the ambiguity, it is usually taken the expression resulting by considering the average between equation 2.23 and 2.24:

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = \frac{1}{2} \sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} \right) (\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla_i W_{ij} \tag{2.25}$$

Equation 2.25 is a widely used SPH expression for the conservation of energy when dealing with inviscid compressible flows. Sometimes however, depending on the problem, authors prefer to evolve the thermokinetic energy instead of the thermal one. Being the specific thermokinetic energy defined as:

$$\hat{e} = \frac{1}{2} v^2 + u$$

and its time derivative:

$$\frac{\mathrm{d}\hat{e}}{\mathrm{d}t} = -\frac{1}{\rho} \nabla \cdot (Pv)$$

its SPH expression is:

$$\frac{\mathrm{d}\hat{e}_i}{\mathrm{d}t} = -\sum_j m_j \left(\frac{P_i \mathbf{v_j}}{\rho_i^2} + \frac{P_j \mathbf{v_i}}{\rho_j^2} \right) \cdot \nabla_i W_{ij} \tag{2.26}$$

For some problems, involving shocks or relativistic effects, using 2.26 instead of 2.25 can lead to more accurate results. An additional option would be to instead evolve the SPH equation for the specific entropy s, not reported here. However, it should be noted that, when considering u or $\hat{e}$, the entropy is not exactly conserved. Conversely, if one chooses to evolve the SPH equation for entropy, the thermal energy will not be conserved. Unfortunately, both quantities cannot be conserved at the same time, and there is no better formulation.

2.4.4. Variational derivation

The derivation of SPH equation for the EE presented above suffers from the variety of options available to approximate the differential operators. As a consequence, the discretization process may seem built *ad hoc*. Additionally, the smoothing length h has been implicitly considered a constant parameter, not allowed to vary in space or in time.

A variational approach has been developed based on the definition of the Lagrangian for an inviscid ideal gas:

$$L = \int \rho \left(\frac{v^2}{2} - u \right) dV \tag{2.27}$$

By discretizing 2.27 in SPH particle terms, the equations of motion can be derived in a more consistent way, and it allows to include coherently the ∇h terms coming from a variable smoothing length. The precise demonstration can be found in [32] and in [50], and will not be reported here. This is because, in case of constant h, results coincide with the equations presented above. Moreover, even when considering a variable smoothing length, the additional ∇h terms appearing in the momentum equation are often neglected in actual SPH implementations. Authors argue that including them would indeed increase the accuracy, but not to a decisive extent, and the numerical solution would still correctly capture the relevant physical features.

2.5. Additional artificial terms to SPH equations

Even with smooth initial conditions, the gas dynamics described by the Euler Equations can produce discontinuities, such as shock waves and contact discontinuities. There, the differential form of the EE cannot be used anymore, and the integral form that leads to the Rankine-Hugoniot equations should be used. Furthermore, in these scenarios, entropy is not conserved, but it always increases across shocks. Some form of dissipation must be introduced when dealing with these phenomena, and in SPH it is done by introducing artificial terms in the momentum and energy equations.

2.5.1. Artificial viscosity

The artificial viscosity term has no real connection to real viscosity, but it is a term added to the SPH Euler Equations to allow the simulation of shock waves, or to provide a term that stabilizes the numerical scheme from the inevitable truncation and integration errors. Through artificial viscosity terms, shock fronts can be widened from true discontinuities into resolvable layers of finite width. The most common approach consists in introducing an artificial viscosity term Π_{ij} directly into the momentum equation:

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = -\sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} + \Pi_{ij} \right) \nabla_i W_{ij} \tag{2.28}$$

The "standard" formulation for Π_{ij} is:

$$\Pi_{ij} = \begin{cases} \dfrac{-\alpha c_{ij}\mu_{ij} + \beta\mu_{ij}^2}{\rho_{ij}}, & \text{if } \mathbf{v}_{ij}\cdot\mathbf{x}_{ij} < 0, \\ 0, & \text{otherwise,} \end{cases} \tag{2.29}$$

with:

$$\mu_{ij} = \frac{h_{ij}\,\mathbf{v}_{ij}\cdot\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|^2 + \varepsilon h_{ij}^2}. \tag{2.30}$$

Here $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$, $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$, h_{ij} and ρ_{ij} represent the average value between the respective quantities at particle i and j, c_{ij} is the average value of the speed of sound. The strength of this artificial viscosity is regulated by the two free parameters α and β, with typical values being: $\alpha \simeq 1$ and $\beta \simeq 2$. The parameter in equation 2.30 $\varepsilon \simeq 0.01$ is introduced to protect μ_{ij} from singularities when two particles get too close. Real viscosity dissipates the kinetic energy into thermal energy, and its contribution to u and s is always positive. To reproduce this behaviour, the term Π_{ij} is also added in the energy equation:

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = \left.\frac{\mathrm{d}u_i}{\mathrm{d}t}\right|_{pres} + \left.\frac{\mathrm{d}u_i}{\mathrm{d}t}\right|_{visc} = \left.\frac{\mathrm{d}u_i}{\mathrm{d}t}\right|_{pres} + \frac{1}{2}\sum_j m_j \Pi_{ij}\mathbf{v}_{ij}\cdot\nabla_i W_{ij} \tag{2.31}$$

The pressure term in equation 2.31 is exactly the one in 2.25. The switch in equation 2.29 guarantees that the viscosity is activated only for particles that are approaching each other, and thus the entropy and energy production is always positive. This artificial viscosity also vanishes for solid-body rotations and conserves both linear and angular momentum, but has the issue of being present in pure shear flows: if in an inviscid fluid two layers of particles move in the same direction, but with different magnitude of velocity, an undesired viscous effect would be present. To reduce this phenomenon, alternative formulations have been proposed. For instance, Balsara [3] used a switch based on local values of $\nabla\cdot\mathbf{v}$ and $\nabla\times\mathbf{v}$. Other authors introduced a variation in the parameter α, made specific to each particle and not constant anymore, so that an additional differential equation $\frac{\mathrm{d}\alpha}{\mathrm{d}t}$ should be solved. For the purpose of this work, but also in general SPH applications, the form 2.29 is more than accurate enough. For a detailed review on the different artificial viscosities used in SPH, the reader can refer to chapter 2.7 in [52]

2.5.2. Artificial heat conduction

Sometimes, artificial viscosity may produce excessive errors in the form of a too pronounced heating of the fluid. These may become higher in applications to very strong

shocks or in the case of a shock interacting with a fixed wall. An example of artificial heating term, to be added to the thermal energy equation to reduce the viscosity-induced errors, is:

$$H_{ij} = \frac{(\mathcal{H}_i + \mathcal{H}_j)(u_i - u_j)}{\tilde{\rho}_{ij}\left(\left|\mathbf{x}_{ij}^2\right| + \varepsilon^2\right)} \tag{2.32}$$

with the coefficient of thermal conduction for particle i:

$$\mathcal{H}_i = g_1 h_i c_i + g_2 h_i^2 \left(|\nabla \cdot \mathbf{v}_i| - |\nabla \cdot \mathbf{v}_i|\right) \tag{2.33}$$

The term ε in equation 2.32 is the same as found in equation 2.30. g_1 and g_2 are the two free parameters that control the amount of artificial heating introduced. Typically $g_1 = g_2 \simeq 0.5$, but their exact values are specific to the problem under investigation. In addition, the second term in the definition of $\mathcal{H}_i$ vanishes if $\nabla \cdot \mathbf{v}_i \geq 0$. Lastly, the artificial heating term is activated only for approaching particle pairs, just as the artificial viscosity. A more extensive dissertation on artificial heating can be found in [36]

2.6. Regularization techniques in SPH

The SPH numerical formulation can be affected by several well-known issues. Among them, particle disorder or slight inaccuracies in the evaluation of the density and velocity fields can deteriorate the solution, especially in compressible flows or in long-term simulations. Some corrective techniques have been introduced in the literature over the years, with the aim of obtaining more regular solutions. In the following sections, the density re-initialization technique and the XSPH method will be introduced, and their mathematical formulation is outlined.

2.6.1. Density re-initialization

When close to a free-surface region, where there is no fluid and so there are no SPH particles, the interpolation technique may suffer from spurious effects. These are due to the lack of particles in the support domain of the kernel function W centered on a boundary particle. Specifically, the support domain of W is not the complete one according to theory, but it is only a portion.

As a consequence, in these regions, the computed density and pressure fields may be inaccurate, exhibiting oscillations during the simulation phase. Among the alternatives, the density re-initialization technique was developed to overcome this issue, and constitutes a simple and computationally efficient procedure. The main idea consists of applying a

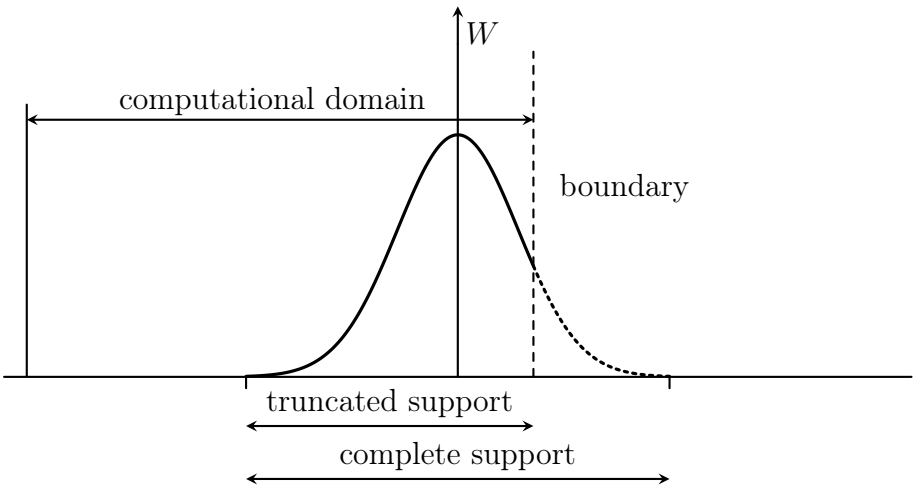


Figure 2.4: 1D representation of a kernel function with truncated support.

filter on the computed density of the particles, and assigning a new value of ρ to each particle. Different orders of correction exist, depending on the desired level of accuracy. The *zeroth order*, or *Shepard filter*, is the simplest and most popular approach. Every n time steps, with n usually in the range from 10 to 30, the corrected $\tilde{\rho}$ for each particle i is calculated as follows:

$$\tilde{\rho}_i = \frac{\sum_j m_j W_{ij}}{\sum_j W_{ij} \frac{m_j}{\rho_j}} \tag{2.34}$$

Despite being used mainly for incompressible fluid problems, the zeroth order density re-initialization technique can be a suitable correction to be applied also in the SPH formulations for inviscid compressible flows.

2.6.2. XSPH

In the "standard" SPH method, particle positions are updated according to the following equation:

$$\frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} = \mathbf{v}_i \tag{2.35}$$

In the so-called XSPH variant (see [30]), a corrective term is added to equation 2.35:

$$\frac{\mathrm{d}\mathbf{x}_i}{\mathrm{d}t} = \hat{\mathbf{v}}_i = \mathbf{v}_i + \varepsilon \sum_j \frac{2\, m_j}{\rho_i + \rho_j} \left(\mathbf{v}_j - \mathbf{v}_i\right) W_{ij} \tag{2.36}$$

being ϵ a constant parameter, $0 \leq \varepsilon \leq 1$. This XSPH variant moves the particle with a

velocity that is also accounting for the average velocity of the neighboring particles. This allows the numerical solution to have smoother distributions of particle velocities and has proven useful in the simulation of nearly incompressible flows. The literature studying the effects of the XSPH correction on inviscid compressible flows is limited; however, it can still be a useful resource.

2.7. Variable smoothing length

Until now, the smoothing length parameter has been implicitly assumed constant in time and uniform in space. However, in certain problems, there may be regions in the computational domain where density skyrockets, and other regions where density gets to levels that are much lower than the initial value. In these cases, it may be preferable to allow h to vary in space and time. Gingold and Monaghan, the inventors of SPH, suggested to relate h to the local value of density, for instance as:

$$h_i = \sigma \left(\frac{m_j}{\rho_j} \right)^{\frac{1}{d}} \tag{2.37}$$

where d is the number of dimensions and σ is a constant $\simeq 1.3$. If the density is computed by the summation equation 2.6, in case of a variable h:

$$\rho_i = \sum_j m_j W_{ji}(h_i) \tag{2.38}$$

Then, h_i can be updated using the resulting value of ρ_i or, in a more consistent way, one can consider equations 2.37 and 2.38 together as a single non-linear equation for the variable ρ_i, and solve it numerically at each time step.

Another approach would be to link h and ρ through a differential equation, an example of which is found in Monaghan, [30]:

$$\frac{\mathrm{d}h_i}{\mathrm{d}t} = -\left(\frac{h_i}{\rho_i \, d} \right) \frac{\mathrm{d}\rho_i}{\mathrm{d}t} \tag{2.39}$$

where again d is the number of dimensions. Here, the temporal variation of h is directly related to the mass conservation equation, and authors simply use the current values of ρ_i, $\frac{\mathrm{d}\rho_i}{\mathrm{d}t}$ to compute the derivative of h_i.

2.7.1. Adaptive Density Kernel Estimation technique

The Adaptive Density Kernel Estimation (ADKE), as discussed in [21, 48], is a method introduced to solve the sharp discontinuities that can arise in compressible flows. It consists of creating an estimate of the density profile to decide whether a particle belongs to a region of high density or to a low-density space. Then, based on this estimate, the bandwidth or smoothing length h of the kernel for each particle i is modified to allow for a point to point variation. The Adaptive Density Kernel Estimation (ADKE) technique consists in following a series of steps, at each time step $t = t_0 + n\Delta t$ of the simulation:

1. First, a pilot density $\tilde{\rho}_i$ is estimated for each particle using the density summation equation 2.6, but using an initial value $h = h_0 = D\Delta x_0$ of the smoothing length. The value of h_0 is thus linked to the initial inter-particle separation Δx_0 by a constant parameter D.

2. In the second step, the "local bandwidth" factors are computed according to a geometric average of the pilot densities:

$$\log(g) = \frac{1}{N}\sum_{j}^{N}\log(\tilde{\rho}_j) \tag{2.40}$$

$$\lambda_i = \kappa\left(\frac{g}{\tilde{\rho}_i}\right)^{\epsilon} \tag{2.41}$$

3. The new smoothing length is set for each particle as

$$h_i = \lambda_i h_0$$

4. The density is then computed again using the summation equation with the updated values of h_i.

In equation 2.40, N denotes the complete set of fluid particles in the system. In equation 2.41 κ is a constant scaling factor, and ϵ is the so-called sensitivity parameter. For compressible fluid problems, they usually assume the values $\kappa \simeq 1$, $\epsilon \simeq 0.5$. For the specific choice of $\kappa = 1, \epsilon = 0$, the Adaptive Density Kernel Estimation (ADKE) scheme is reduced to an SPH scheme with constant smoothing lengths.

In any case, whenever h is allowed to vary, the symmetry of the SPH equations of motion is formally lost: if two particles i and j have different h_i, h_j, then their interactions will be different, and the conservation properties of SPH are no longer valid. One way to restore

these properties consists in using a symmetric version of the kernel interaction, defined as either $\tilde{W}_{ij} = W_{ij}(\frac{h_i+h_j}{2})$ or $\tilde{W}_{ij} = \frac{1}{2}[W_{ij}(h_i) + W_{ij}(h_j)]$, in all relevant equations.

Finally, strict mathematical consistency would require including the corrective ∇h terms, inside the equations of motion. However, in general, the authors argue that these corrective terms would not change the accuracy by more than $\simeq 10\,\%$, so they are often neglected anyway.

2.8. Time integration

SPH reduces the continuum partial differential equations into a set of ordinary differential equations for $\mathbf{x}, \mathbf{v}, \rho, u$, which must be solved numerically for each particle in the system. As a consequence, any time integrator algorithm can be used, implicit or explicit. When dissipation is absent, it may be convenient to choose symplectic integrators, which are designed to conserve energy exactly, because they would preserve the conservation properties of the SPH schemes. However, often explicit low-order methods are employed. For instance, even the basic explicit Euler integrator is deemed suitable. Other popular choices include two-step explicit predictor-corrector schemes, such as the Runge-Kutta 2 (RK2) scheme. In the current literature about the application of SPH to compressible flows, implicit integrators are rarely mentioned. In any case, the time step Δt should be carefully selected, to ensure that the resulting numerical algorithm is stable. Fluid problems are intrinsically stiff, and for explicit integrator a proper check on the time step Δt needs to be made. In the Finite Differences method, the Courant-Friedrichs-Lewy (CFL) condition states that the Δt not be greater than the time it takes for a traveling wave to reach the neighboring grid points:

$$C = \frac{u\Delta t}{\Delta x} \leq C_{max} \tag{2.42}$$

From equation 2.42 one obtains a relation on the maximum Δt recommended for a grid of given spacing Δx. A similar condition can be formulated for a SPH problem, yielding a maximum allowed time step of:

$$\Delta t_{max} = C \min_i \left(\frac{h}{c_i} \right) \tag{2.43}$$

where C is a safety constant lower than 1. Equation 2.43 is the most basic condition for a problem without dissipation and with constant h. A more complete expression will be given in subsection 4.3.8.

2.9. Final considerations on SPH

Generally speaking, the SPH method has a number of different characteristics, when compared to traditional Eulerian methods such as the Finite Difference method, the Finite Volume method or the Finite Element method. They can be broadly summarized as follows:

- pure advection of properties tied to the particles is treated exactly;
- problems involving interfaces between different constituents are easier when formulated with SPH;
- the problem resolution can be made local and depending on time, making it attractive for problems involving very different length scales;
- when dealing with large void regions, SPH allows computations to be made only where matter is actually present;
- external forces can be easily applied in the equations of motion for particles;
- SPH equations naturally conserve linear and angular momentum, an attractive feature for problems where the dissipation is absent;
- when applied to shock problems, including a simple artificial viscosity term, SPH gives correct solutions for pre- and post-shock conditions;
- simulations often give results that are more accurate than *a priori* estimates would suggest.

However, SPH also has some disadvantages, with respect to grid-based methods:

- close to void boundaries, the interpolation of properties may suffer from not enough particles being close together;
- the derivation of the equations of motion, even when using a variational approach, is not obvious, and sometimes they may appear to be built *ad hoc* for the specific problem;
- there is a lot of freedom when building an actual SPH scheme. For example, density can be updated in several different ways, and there is no actual recommended option;
- the method is very sensitive to the initial arrangement of particles and to the selected kernel, for which, again, the choice among all existing alternatives, is not straightforward. The "best" choice of kernel is still an open question.

- when discretizing a problem with a large number of particles, the computational cost can become noticeably higher than that of the grid-based methods. This comes from the need of considering each neighboring contribution for each particle in the system at each time step;
- for shock problems, results in 2D and 3D can be noisier than ones obtained with other methods, such as Riemann solvers;
- in general, the predicted SPH accuracy is lower, so, in problems that involve small perturbations, it may not be the best choice;
- although errors in the integral interpolation procedure can be shown to be of the order of $O(h^2)$, literature studies on the accuracy, stability, and convergence of SPH are very limited, especially for problems in 2 and 3 dimensions. Therefore, such analysis should be conducted on the specific problems at hand.

Referring to the application described in subsection 1.3.3, modeling a coma as a fluid, either with Navier-Stokes Equations or with the Euler Equation, is already just an approximation, because a theoretically correct approach would involve the numerical solution of the Boltzmann Equations. Therefore, considering the advantages and disadvantages listed above, SPH can still be a valid choice to represent a simplified fluid model of a cometary coma.

3 | Project Chrono, GRAINS, and available SPH solvers

This chapter provides a brief introduction to the open-source software Project Chrono, and describes its relevant features and modules. Chrono provides a built-in SPH solver that will be analyzed. Then, an overview of the available SPH solvers is provided, where their specific characteristics are listed. Finally, there will be a discussion about the alternative paths that could be taken in order to couple Project Chrono with an SPH software capable of dealing with inviscid compressible flows.

3.1. Project Chrono

Project Chrono, [42, 43], is an open-source library, developed for multi-physics simulations. Its primary focus is the numerical solution of complex Multybody Dynamics (MBD), but over the years it has evolved, and now includes several different modules, or units, that expand the class of problems that can be simulated. Project Chrono is platform-independent, since its build system is based on CMake; it is written mainly in C++ but also has an extension called PyChrono, which is the Python API to use Chrono libraries. The standard notation used when referring to Chrono's submodules consists in pre-pending the name of the module with "Chrono::", for instance Chrono::GPU, or Chrono::Vehicle. Chrono::Core (C::C), or equivalently Chrono::Engine (C::E), is the very core of the whole library: it constitutes the engine responsible for assembling and solving dynamic systems. It allows the user to deal with:

- rigid bodies;
- deformable or flexible bodies;
- kinematic links;
- external forces;
- contact and collisions of bodies;

- several models for friction;
- constraints to enforce a prescribed motion;
- linear motors, actuators, pneumatic cylinders;
- linear and nonlinear solvers;
- explicit and implicit time integrators.

Project Chrono has a long history of validation studies, and it is a powerful and well-established option for numerical simulations of complex systems.

3.2. SPH in Chrono

The optional module Chrono::FSI (C::FSI) allows to consider fluid-solid interaction by modeling fluids using SPH. This unit can also be used as a standalone CFD solver or to simulate the dynamics of granular materials. Chrono::FSI is written in C++, but uses CUDA (Computing Unified Device Architecture) to parallelize computations, and provide a considerable saving in simulation times. As a consequence, NVDIA hardware is strictly required if one wishes to use this extension. The Fluid-Solid Interaction unit implements different SPH schemes:

- explicit Weakly Compressible SPH (WCSPH);
- Implicit Incompressible SPH (IISPH).

The WCSPH scheme considers the fluid continuity and momentum equations:

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} = -\rho\nabla\cdot\mathbf{v} \tag{3.1}$$

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2\mathbf{v} + \mathbf{f} \tag{3.2}$$

that are discretized as :

$$\frac{\mathrm{d}\rho_i}{\mathrm{d}t} = \rho_i \sum_j \frac{m_j}{\rho_j}(\mathbf{v}_i - \mathbf{v}_j)\cdot\nabla_i W_{ij} \tag{3.3}$$

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = -\sum_j m_j \left(\left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right)\nabla_i W_{ij} + \Pi_{ij}\right) + \mathbf{f}_i \tag{3.4}$$

In the equation 3.4, Π_{ij} is a representation of the physical viscosity of the fluid: it is not to be intended as the artificial term in 2.28. A possible definition can be derived by

directly discretizing the ∇^2 operator in 3.2:

$$\Pi_{ij} = -\frac{(\mu_i + \mu_j)\mathbf{x}_{ij} \cdot \nabla_i W_{ij}}{\bar{\rho}_{ij}^2 (x_{ij}^2 + \varepsilon \bar{h}_{ij}^2)}$$

where the parameters μ_I, μ_j are the actual viscosities of the particles i and j. The pressure p is evaluated using an equation of state, tailored to enhance incompressibility, such as the Tait's equation:

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

where ρ_0 is the reference density of the fluid, γ tunes the stiffness of the relation, and c_s represents the speed of sound, which in WCSPH is usually taken as 10 times the maximum fluid velocity. This model clearly does not consider a truly compressible fluid, moreover, it does not even take the energy as a state variable. IISPH, as the name suggests, is a variant of SPH that uses an implicit formulation to strictly enforce the incompressibility of the fluid of interest. Its implicit formulation allows the use of larger time steps compared to traditional SPH, at the expense of a noticeable increase in the computation burden, requested at each time iteration. A detailed discussion of this scheme can be found in [45].

3.2.1. Fluid-solid interaction

In Chrono::FSI, the interaction between solid bodies and fluid particles is obtained by introducing a different kind of SPH markers, called Boundary Condition Enforcement (BCE) markers. These are particles that are specifically designed to make it possible for the fluid to feel the presence of a solid body, either a moving one or a fixed wall, by generating contributions that enter the density and momentum equations. These BCE markers are arranged along the interface in several layers, usually 3, and are attached to a specific solid body. They do have the same set of physical properties as the fluid particles, but these are managed differently:

- Properties of BCE markers are not assigned by default and then evolved: they are computed at each time step by interpolating the contributions of the neighboring fluid.
- When computing the derivatives of density and momentum for a fluid particle, the summation shall also consider the contributions of BCE markers.
- $\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t}$ is assembled for all BCEs. This derivative is not integrated in time, but used to represent the effect of the fluid on the solid body.

- BCE markers actual velocities come from the motion of the solid bodies they are attached to.

Figure 3.1, from [19], shows in red a fluid particle, and in blue the set of BCE attached to the solid body to allow interaction of the fluid and solid domains. A more detailed discussion of BCE markers and the fluid-solid interaction algorithm will be held in section 4.3.5.

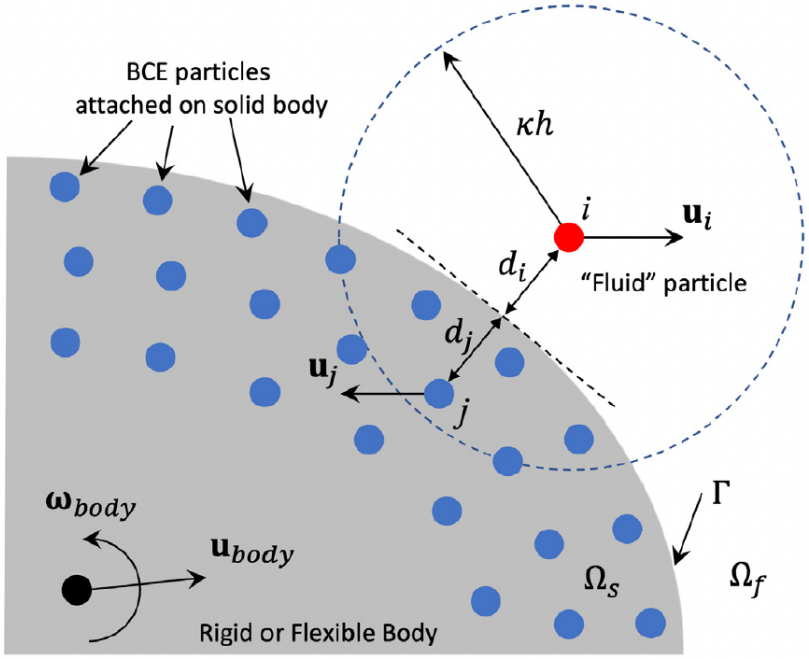


Figure 3.1: Fluid and BCE particles interaction near a fluid-solid interface.

3.3. GRAINS

Thanks to its modular architecture and support for user-developed extensions, Chrono offers the possibility to be easily expanded and customized. GRAINS, developed at Politecnico di Milano, as seen in [5, 15, 16], is a CUDA-based code that simulates the gravitational N-body problem, also considering contact interactions between the system bodies. It is built on Chrono::Core and can be thought of as a custom extension of Chrono. The present work focuses on the numerical solutions of the Euler Equations, thus GRAINS has not been used in this study. However, in view of possible future integrations, it has been deemed fundamental that an SPH scheme for inviscid compressible flows should be easily coupled with GRAINS. Such a coupling would enable the simulation of complex scenarios in which both fluid dynamics effects and N-body gravitational interactions are present simultaneously.

3.4. SPH solvers

Despite being developed in the 1970s, SPH became popular for engineering applications only in the early years of the current millennium. In 2005, an international organization, SPHERIC, acronym for SPH rEsearch and engineeRing International Community, was founded to promote theoretical research and to encourage the adoption of SPH in both research and industrial environments. Therefore, represents a reliable source of information on existing SPH solvers. In the following, some of the most relevant ones are briefly reviewed.

3.4.1. DualSPHysics

DualSPHysics is an open-source parallel code, based on the SPH method, that has been designed to deal with complex real-world engineering problems. It is written in a combination of C++ and CUDA, thereby enabling the simulation of CFD problems on highly-parallel architectures. The strength of this code lies in the extensive validation campaigns that have been conducted, which make it a reliable choice. In addition, DualSPHysics can be easily coupled with external MBD solvers, and developers have already provided a way for it to interact with Project Chrono [29]. Figure 3.2, taken from the official documentation of the SPH code, illustrates how the two software frameworks interact with each other: fluid and solid domains are solved and updated in separate environments, but at each time step information between them is exchanged, so that each domain accounts for the influence of the other. Compared to the Chrono::FSI module, DualSPHysics is a more mature and feature-complete SPH solver. However, similarly to Chrono, its main focus is on viscous incompressible flows, and compressibility is not really considered.

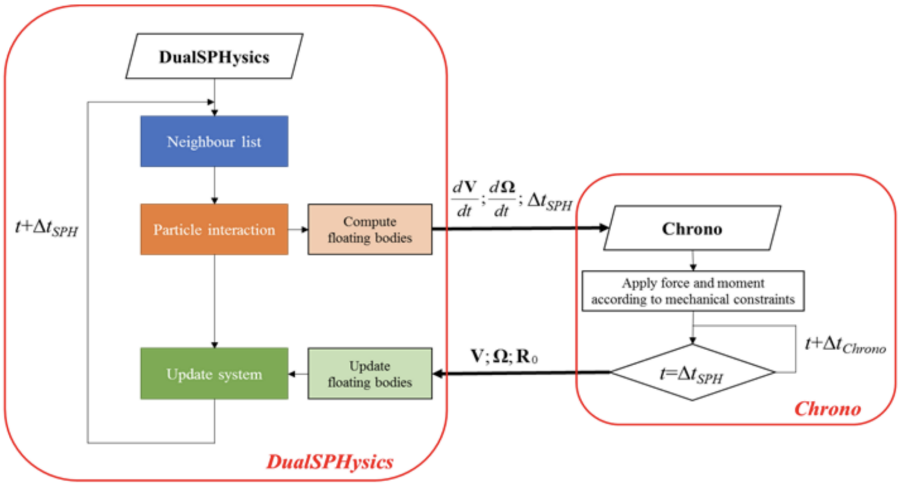


Figure 3.2: Interaction between DualSPHysics and Chrono.

3.4.2. GADGET and PHANTOM

Gadget [51] and Phantom [40] are two SPH-based open-source codes for astrophysical simulations. The first provides the tools required for cosmological simulations, that involve hydrodynamics and N-body interactions. Gadget is written in C++, and it is intended to run mainly on massively parallel computers by means of the message passing interface (MPI). In addition, Phantom provides a way to simulate problems that involve magneto-hydrodynamics effects. Both codes use SPH to model an inviscid compressible flow; this feature makes them attractive if referring to the simulation of a cometary coma proposed in section 1.3.3. However, despite also providing a way to include the N-body gravitational interaction, they do not include a framework to accurately simulate MBD systems: their representation of solid rigid bodies is very simplified, and no complex shapes can be used.

3.4.3. PySPH

PySPH [46] is an open-source framework for the SPH method. It provides a way to define a complete simulation setup using pure Python, and then automatically produces high-performance code. PySPH supports a wide range of SPH schemes, and deals with both compressible and incompressible fluid flows. The set of boundary conditions that can be applied includes: periodic, solid wall, and inlet/outlet conditions. This last boundary condition is rarely seen implemented in SPH codes, and is still under active research; therefore, it is an attractive feature of PySPH. It can deal with rigid solid bodies, but the limitation of this code is that it cannot deal with complex MBD problems.

3.5. Available options

The aim of this work consists of finding a way to enable the inclusion of fluid dynamics effects in MBD problems, possibly considering also N-body interaction, such that systems like cometary atmospheres could be simulated by progressively increasing the complexity. However, for several reasons, none of the codes listed above is fully suited for this task:

- Chrono::FSI and DualSPHysics offer an accurate and wide framework to simulate fluid-solid interaction and MBD domains. However, they are designed for viscous incompressible flows; when asked, the developers stated that the extension to compressible fluid is not currently planned.
- Gadget, Phantom and PySPH offer SPH models of the Euler Equations. Neverthless, they cannot deal with complex MBD systems.

Therefore, to proceed with the work and attempt to fulfill the initial objectives, two different paths emerged:

1. Select one SPH solver already tested and available, and develop a software infrastructure that allows it to communicate with Chrono. This choice would imply building a conversion tool between the different data types used by the software, that could make the two communicate at runtime.
2. Develop a custom external module, based on Chrono::FSI, that implements an SPH scheme built for the numerical solution of the fluid Euler Equations.

The first option would imply the development of a TCP/IP socket based communication protocol, and there is no real literature on how this could be achieved for Chrono. Such an approach would also introduce complexity, related to synchronization and data serialization between two independent frameworks. Moreover, such a choice would be less robust, making the software architecture more fragile and sensitive to future updates in one of the codes. Therefore, the choice was made to select Chrono and its FSI module as the reference framework, and to develop an extension to them that could deal with compressible flows. The resulting code, being internal to Chrono, could be better optimized and modified at a lower level, ensuring easier maintainability and the possibility of future upgrades or integrations.

4 | Implementation

This chapter presents the implementation of the custom external module, built upon the Chrono framework, that solves the inviscid compressible Euler Equations within the SPH formulation.

Similarly to Chrono::FSI, this module, named CSPH (Compressible SPH), is written combining C++ for the class management and the global organization of the different algorithms called at each simulation time step, and CUDA, to achieve a computational acceleration in those computations that can be parallelized.

Despite being developed to simulate only incompressible flows, Chrono::FSI provides some internal components that are general enough to be easily reused. Therefore, instead of designing a completely new and independent solver, CSPH has been designed to remain consistent with the general architecture of the C::FSI module. In the following sections, a top to bottom approach will discuss with increasing detail the components of the module. Section 4.1 provides a global overview of the simulator, explaining which input can be provided, how to evolve the system dynamics, and which output can be obtained. Section 4.3 describes in detail the different SPH equations that are implemented and how they can be combined together. Lastly, section 4.4 explains how the software in structured, which are the main classes involved, and how the SPH equations are implemented in CUDA.

4.1. Simulation workflow

From a global point of view, the proposed SPH solver is structured similarly to the Chrono::FSI module. A generic simulation can be divided into three main parts:

- input and system initialization;
- main simulation loop with, optionally,visualization of the current state;
- output and data export for subsequent post-processing.

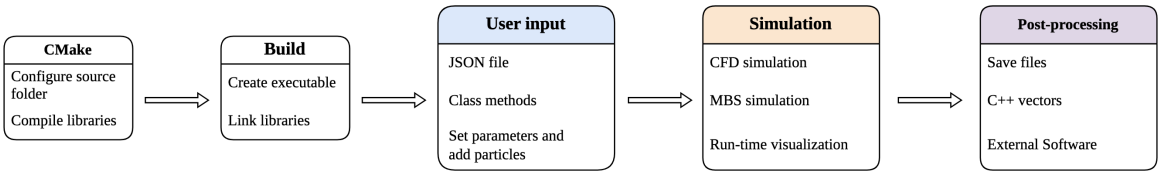


Figure 4.1: High-level view of the code pipeline.

4.1.1. Input and initialization

In any case, the user must always create at least two objects of the classes `ChFluidSystem_csph` and `ChFsiSystem_csph`. These will provide all the high-level APIs (Application Programming Interface) needed to create a complete simulation scenario. The input accepted by the system can be split into two families: the numerical solution parameters and the SPH particles that represent the simulated fluid. The parameters can be given as input in two different ways: either through a specified JSON file, or by calling the appropriate *SetParameter* public methods of the `ChFluidSystem_csph` class. Table 4.1, in section 4.4.1, will list some of the most relevant simulation constants; all of them can be modified during the initialization phase.

All simulation parameters already have a default value; therefore, only those of interest need to be properly modified. However, SPH fluid particles, rigid bodies, and BCE markers always need to be explicitly added to the system inside the main executable file. This can be done by using the `ChFluidSystem_csph.AddSPHParticle` and `AddBCEMarker` methods. The code implements several different overloads for them, so the user can add the markers by specifying either:

- the complete initial state: $\mathbf{x}_i, \mathbf{v}_i, \rho_i, P_i, u_i, h_i, m_i$, where m_i is the mass associated with the particle i;
- only a subset of values. Properties not explicitly specified are set to the corresponding default value, stored in the simulation parameters.

These two methods allow adding particles into the system one by one and thus defining arbitrarily complex initial configurations. Other methods, such as `AddBoxSPH` or `CreatePointsSphere`, enable an express creation of particles according to common simple geometries. Note that a Chrono Multybody System (MBS) must always be created and associated with at least one solid body. However, if the problem to be simulated does not involve an actual fluid-solid interaction, a dummy fixed body with no associated solid par-

ticles is enough for the simulation to start. A call to `ChFsiSystem_csph.Initialize()` marks the completion of the problem setup.

4.1.2. Simulation loop and visualization

Once the fluid and MBD systems have been correctly initialized, the main simulation loop can start. In the most simple formulation, the user only needs to define the initial time t_0, the final time t_{end} and a step size Δt. Then, the system dynamics is evolved in time through the `ChFsiSystem_csph.DoStepDynamics`(Δt) method. Usually, this call is placed inside a *while* loop that executes until the current simulation time reaches the specified t_{end}. The mentioned function internally hides all the complexity: is responsible for the independent time evolution of the fluid system and the Chrono MBS, and manages the possible interactions between the two. Note that the specified Δt should be intended more as a synchronization time step. Internally, the two systems can be set up to have specific time steps, namely t_{CFD} and t_{MBS}. As will be explained in section 4.3.8, t_{CFD} can even vary to keep the integration stable. If one of the two time steps is less than the input Δt, the corresponding system will be integrated multiple times, until $t + \Delta t$ is reached. If the opposite condition is met, the system is evolved for the time step Δt.

An optional visualization tool has also been implemented. It acts as a plugin for the Chrono::VSG module, a VulkanSceneGraph (VSG) -based run-time engine, and allows on-screen rendering of the SPH system at a specified frequency. The code offers the option to select the SPH coloring on either the velocity, density, pressure, or energy.

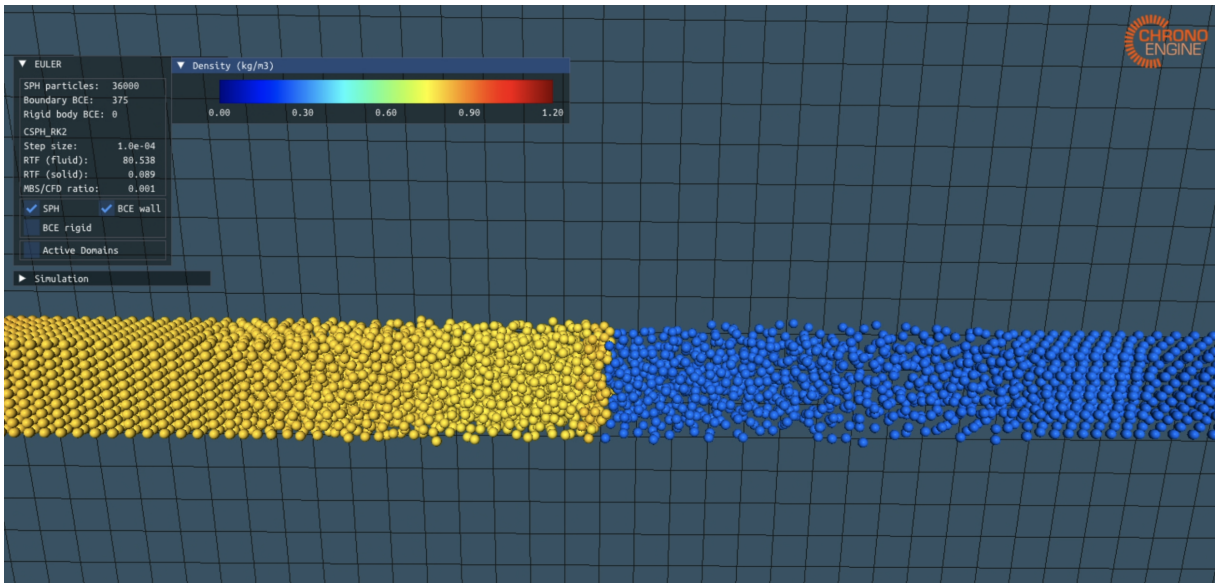


Figure 4.2: Run-time visualization of a 3D shock front.

4.2. Output and data export

The code can provide output in different ways, depending on the preferred post-processing tool. For example, data can be saved in files and then exported to be analyzed by external software such as MATLAB or ParaView. An alternative approach can rely on post-processing the results directly inside the main C++ executable, using either proper external libraries or additional Chrono modules, for instance Chrono::MATLAB or Chrono::Postprocess. Therefore, in general, the user is given freedom and control on how to manipulate the resulting simulation data.

The output can be retrieved either at specific time steps during the simulation or after it has ended. To obtain data at specified time steps, the user needs to call the fluid or solid system methods inside the main `while` loop. Otherwise, if they are invoked after the last integration step, only the quantities at t_{end} are retrieved. The code provides two families of APIs for data output:

1. the `GetParticleProperty` methods give back C++ vectors containing the values of the specified fluid property for all the SPH particles in the system, or for a subset of them;

2. the `SaveParticleData` method, requiring an existing folder as input, saves in a .csv file a set of properties for all SPH particles. This set of saved quantities depend on an internal variable of the `ChFluidSystem_csph` class, which can be modified during the initialization phase. In case the method is called multiple times, the output files are automatically labeled in a progressive way, so that no ambiguity can occur.

Algorithm 4.1 illustrates all the logic steps required to build and run a numerical simulation using Chrono and the proposed CSPH external module.

Algorithm 4.1 Complete simulation program

```
Link Chrono::Engine and custom module libraries
Inside the main function:
Create a Chrono MBS
Create the compressible fluid and the FSI systems
Set simulation parameters
if Domain has simple geometry then
    Create set of SPH particles with fluid system methods
else
    Create individual fluid SPH particles and add them to the fluid system
end if
Add a solid body to the Chrono MBS
Create associated solid BCE markers and add them to the fluid system
Initialize the FSI system
if Chrono::VSG available then
    Create VSG visualization system
    Add custom plugin for compressible SPH
end if
Define final time, time step, output frequency
while time < final time do
    Call method DoStepDynamics
    if Visualization is on then
        Render SPH particles
    end if
    Update current time
end while
if Post-process in same C++ executable then
    Retrieve fluid properties at final time
end if
```

4.3. Numerical formulation

As discussed in chapter 2, the SPH framework does not provide a unique formulation, nor there exists a "best" SPH scheme: each author derives his own one. This variety of approaches led to the need to design the CSPH module in such a way that a certain degree of customization was possible. The current implementation aims at solving the Euler Equations, considering (ρ, P, u) as the set of thermodynamic variables, which lead

to the classical differential formulation, already reported in section 2.4:

$$\begin{cases} \dfrac{\mathrm{d}\rho}{\mathrm{d}t} = -\rho \nabla \cdot \mathbf{v} \\ \dfrac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\dfrac{\nabla P}{\rho} + \mathbf{g} \\ \dfrac{\mathrm{d}u}{\mathrm{d}t} = -\dfrac{P}{\rho} \nabla \cdot \mathbf{v} \end{cases} \tag{4.1}$$

The system of equations 4.1 must be closed with an equation of state. In CSPH, the equation of state for calorically perfect ideal gasses has been implemented so far: $P = (\gamma - 1)\rho u$. However, a logical switch has been designed in order to allow for an easy integration of more complex formulations. In the current state, only the specific thermal energy u is evolved; thermokinetic energy $\hat{e}$ and entropy s are not explicitly defined.

4.3.1. Kernel functions and space dimensionality

The three kernel functions mentioned in chapter 2.2.1 have been implemented and can be chosen at the start of the simulation. Namely, they are:

1. the cubic spline function, set as the default choice;
2. the quintic spline function;
3. the Wendland C^2 kernel.

For each of them, the code automatically sets the proper value of the normalization constant α_d and of the parameter κ, that defines the extension of the support domain, so that for $q > q_{max} = \kappa h$, $W(q, h) = 0$.

Chrono::Engine and Chrono::FSI by construction deal only with 3D systems, so all vector variables, like position and velocity, have 3 spatial components. In Chrono::FSI this is true also for the kernel functions and their gradients, which are always considered with respect to a 3D scenario. The developed code, in contrast, allows to define fluid problems in either 1D, 2D or 3D, but always retaining full compatibility with Chrono. The dimensionality of the problem can be changed by the user, before starting the simulation. If a 1D or 2D problem is selected, then the normalization constant α_d for the kernel functions is changed accordingly, and the computation of the kernel gradient is modified as well. However, the gradient ∇W_{ij}, as well as the position $\mathbf{x}_i$ and the velocity $\mathbf{v}_i$, is still represented as a 3D vector. For instance, in the case of a 1D simulation, it will be computed as:

$$\nabla W_{ij,1D} = \begin{bmatrix} \frac{\mathrm{d}W_{ij,1D}}{\mathrm{d}x} & 0 & 0 \end{bmatrix}$$

and for 2D scenarios:

$$\nabla W_{ij,2D} = \begin{bmatrix} \frac{\partial W_{ij,2D}}{\partial x} & \frac{\partial W_{ij,2D}}{\partial y} & 0 \end{bmatrix}$$

This allows keeping the fluid domain fully compatible with the solid domain that is managed by C::E; at the same time, the flow can be restricted to evolve only along the x direction or along x and y for 2D simulations.

4.3.2. SPH density update

By design, two options are provided for the time evolution of the density. The first option uses the SPH summation equation:

$$\rho_i = \sum_j m_j W_{ij}$$

where in this case, the summation also includes the particle i itself and is performed right at the beginning of each time step. Otherwise, the user can choose to compute the density of each particle by evolving the mass conservation equation in the system 4.1. Its SPH version implemented in the code is:

$$\frac{\mathrm{d}\rho_i}{\mathrm{d}t} = \sum_j m_j (\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla_i W_{ij} \tag{4.2}$$

4.3.3. SPH momentum and energy computation

Currently, the code uses the following formulations for the SPH momentum and energy equations:

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = -\sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j} + \Pi_{ij} \right) \nabla_i W_{ij} \tag{4.3}$$

and:

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = \frac{1}{2} \sum_j \left[\left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} + \Pi_{ij} \right) (\mathbf{v}_i - \mathbf{v}_j) + 2H_{ij}(\mathbf{x}_i - \mathbf{x}_j) \right] \cdot \nabla_i W_{ij} \tag{4.4}$$

In the current state, the artificial viscosity and artificial heat conduction used have the

expressions of equations 2.29 and 2.32, here reported for clarity:

$$\Pi_{ij} = \begin{cases} \dfrac{-\alpha c_{ij}\mu_{ij} + \beta\mu_{ij}^2}{\rho_{ij}}, & \text{if } \mathbf{v}_{ij}\cdot\mathbf{x}_{ij} < 0, \\ 0, & \text{otherwise,} \end{cases}$$

being

$$\mu_{ij} = \frac{h_{ij}\,\mathbf{v}_{ij}\cdot\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|^2 + \varepsilon h_{ij}^2}.$$

and:

$$H_{ij} = \frac{(\mathcal{H}_i + \mathcal{H}_j)\,(u_i - u_j)}{\tilde{\rho}_{ij}\left(\left|\mathbf{x}_{ij}^2\right| + \varepsilon^2\right)}$$

with:

$$\mathcal{H}_i = g_1 h_i c_i + g_2 h_i^2\,(|\nabla\cdot\mathbf{v}_i| - |\nabla\cdot\mathbf{v}_i|)$$

$$(\nabla\cdot\mathbf{v})_i = \frac{1}{\rho_i}\sum_j m_j\,(\mathbf{v}_j - \mathbf{v}_i)\cdot\nabla_j W_{ij}$$

The parameters involved in the equations are to be given as input by the user, but they are given the default values of $\varepsilon = 0.01$, $\alpha = 1$, $\beta = 2$, $g_1 = g_2 = 0$.

4.3.4. Boundary conditions

Currently, CSPH offers the same set of boundary conditions implemented in Chrono::FSI, namely:

- *none*: no boundary condition is enforced, useful for closed system;
- *periodic*: they represent an open boundary, in which the fluid exiting the computational domain immediately re-enters on the opposite side. They allow to simulate flows that exhibit a behavior that periodically repeats in space.

These two types of boundary conditions can be freely combined along the x, y, and z directions of the computational domain. *Inflow/outflow* conditions, in which the values of either velocity, mass flow, or pressure are specified for the whole simulation, have not yet been implemented. This is because in an SPH system, one cannot simply add and

remove particles from the domain: if extra care is not taken, this action can generate spurious waves that propagate in all directions, possibly leading to dominant unphysical behaviors. These open boundary conditions are still not commonly implemented in SPH codes. One possible way of formulation would be to follow the ideas formulated in [14], and then studied deeper in [35], where some buffer regions in the domain are defined, and there SPH particles are generated or removed. Still, when inside either the inflow or the outflow buffer, the particles should be evolved differently from the actual fluid markers that represent the physical flow.

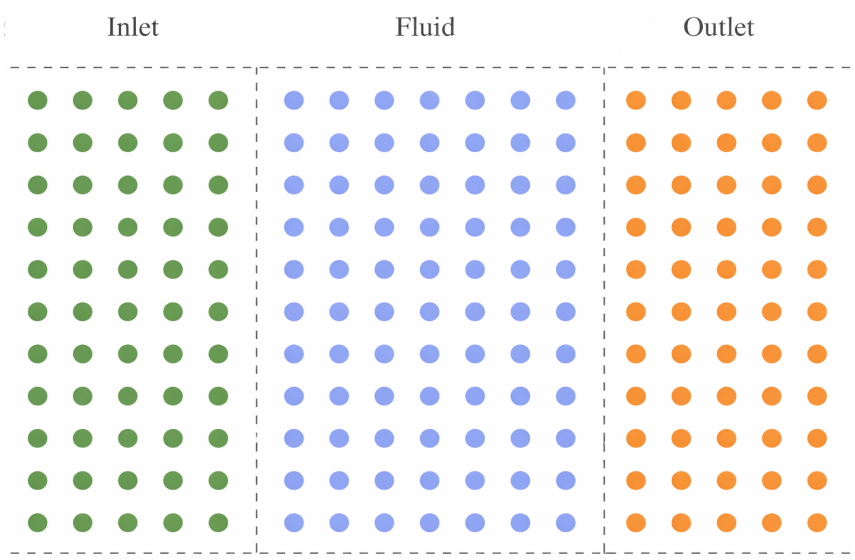


Figure 4.3: 2D representation of inflow and outflow buffer regions.

4.3.5. Fluid-solid interaction

To represent the interaction between fluid SPH particles and solid bodies, Chrono::FSI uses the so-called Boundary Condition Enforcement markers, described in section 3.2.1, and first introduced by Adami in [1]. These BCE markers are attached to solid bodies and allow the enforcement of the no-slip condition at the solid interface. Their position $\mathbf{x}$, velocity $\mathbf{v}$, and acceleration $\mathbf{a}$ are computed following the equations for rigid body kinematics:

$$\mathbf{x}_{BCE} = \mathbf{x}_B + \mathbf{R}\mathbf{x}_0 \tag{4.5}$$

$$\mathbf{v}_{BCE} = \mathbf{v}_B + \omega \times \mathbf{x} \tag{4.6}$$

$$\mathbf{a}_{BCE} = \mathbf{a}_B + \alpha \times \mathbf{x} + \omega \times (\omega \times \mathbf{x}) \tag{4.7}$$

where $\mathbf{x}_0$ is the position of the BCE marker in its body frame, $\mathbf{R}$ is the rotation matrix from the body frame to the inertial frame, $\mathbf{x} = \mathbf{R}\mathbf{x}_0$, ω and α here are the rigid body angular velocity and its derivative and $\mathbf{x}_B, \mathbf{v}_B, \mathbf{a}_B$ are the position, velocity and acceleration of the origin of the body frame.

The work by Adami was mainly devoted to flows where viscosity has a noticeable impact, and gave only a few suggestions on how to treat frictionless problems. However, since CSPH focuses on inviscid flows, only the no-penetration condition has to be prescribed, and so, close to the solid boundary, the fluid particles should have a velocity tangential to that of the interface, and there also the pressure Neumann condition should be satisfied. In CSPH, these are achieved in two steps:

1. *extrapolation*;
2. *post-extrapolation*.

First, BCE markers see their u and P quantities extrapolated from the surrounding fluid particles. If h is variable, it is extrapolated as well. The extrapolation procedure is based on the method of Adami. For each BCE marker, the property f is extrapolated as:

$$f_i = \frac{\sum_j f_j W_{ij}}{\sum_j W_{ij}} \tag{4.8}$$

In equation 4.8, the index i represents a boundary marker, and j denotes the fluid particles in its neighborhood. Therefore, the summation does not include solid-solid contributions. The extrapolated density ρ_i is then obtained from the equation of state, as suggested by Adami. The *post-extrapolation* phase consists in adjusting the pressure of the BCE markers, that represent accelerating interfaces, to mimic the pressure gradient in the fluid phase near the boundary. Adami showed that, starting from a force balance on a fluid-BCE pair, the resulting pressure P_{BCE} to be enforced is:

$$P_{BCE} = \frac{P_{extrapolated} + (\mathbf{g} - \mathbf{a}_{BCE}) \cdot \sum_j \rho_j (\mathbf{x}_{BCE} - \mathbf{x}_j) W_{BCE-j}}{\sum_j W_{BCE-j}} \tag{4.9}$$

where the index j covers only the neighboring fluid particles, and the contributions of other BCE markers are not considered. Adami also stated that the impermeability condition is then implicitly enforced by equations 4.2 and 4.3. In addition, when computing the momentum equation for a fluid particle i, the artificial viscosity interaction with a neighbor BCE should be omitted. Other authors, as seen in [54], suggested an alternative post-extrapolation method. This approach, despite being slightly more accurate, requires explicit information about the geometry of the interface, and has not been implemented in

CSPH. Concerning the density, momentum, and energy SPH equations, when computing them for a fluid element, there is no distinction between neighbor types: contributions from BCE markers that are inside the support domain should always be included. However, things are different for BCE particles. Since the density is extrapolated from the fluid neighbors, neither the summation equation 2.6 for ρ nor the mass conservation equation 4.2 are applied. The same holds for the energy equation 4.4. However, the momentum equation 4.3 is computed at each time step, without considering solid-solid contributions, to obtain the force that the fluid domain exerts on the solid bodies. Each BCE particle i computes the force that acts on itself as $m_i \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t}$. Then, each solid body in the system accumulates these individual forces to compute the force and torque resulting from the fluid domain, and uses them as external loads in the computation of solid dynamics. The described procedure allows both the fluid and solid systems to feel the presence of each other and to account for the interaction effects. Algorithm 4.2 shows how the CSPH module deals with the fluid-solid interaction; the order of the instructions is the same as Chrono::FSI, but executed in the framework of inviscid compressible flows.

Algorithm 4.2 Fluid-solid interaction

```
Advance MBS and fluid systems independently
Inside the fluid system:
for each solid body do
    for each BCE marker do
        Compute prescribed x_BCE, v_BCE, a_BCE
        Extrapolate P, ρ, u
    end for
end for
for each SPH particle in the system do
    Compute derivatives
end for
for each solid body do
    for each BCE marker do
        accumulate forces and torques
    end for
end for
Send forces and torques from fluid to MBD system
Send updated solid phase states from MBS to fluid system
```

The outlined coupling strategy has an explicit character, which means that an explicit

information exchange must occur at every time step. This can yield slight numerical artifacts, when compared to a global monolithic solver, but such an approach would increase the computational burden and would not allow the modularity that Chrono and this proposed CSPH module aim at offering. A more detailed analysis of the precision of this fluid-solid interaction strategy is found in chapter 4 of [45].

4.3.6. Pressure and speed of sound

Pressure P, and similarly the speed of sound c_s, are two fundamental quantities that drive the dynamics of SPH particles. However, they do not possess a differential equation for their time evolution, nor are they computed by means of an SPH interpolation. Instead, their values are obtained from an equation of state. The code currently considers only calorically perfect ideal gasses, so the pressure of particle i is computed as:

$$P_i = (\gamma - 1)\rho_i u_i$$

with γ being the specific heats ratio. The expression of the speed of sound is as follows:

$$c_{s,i} = \sqrt{\gamma \frac{P_i}{\rho_i}}$$

Both quantities must be updated every time a change in density or energy occurs. Therefore, the code automatically re-computes them for each particle after the computation of density through the summation equation, if the SPH scheme uses it, and immediately after the state of the system is propagated in time.

4.3.7. Variable smoothing length

When considering highly compressible flows, it is important to have a resolution that can vary in space and in time, to better represent regions of high and low density. In the SPH framework, this resolution depends on the value of the smoothing length parameter h. Chrono::FSI has been developed considering only a global value of h, which is then equal for all particles in the system and is kept constant in time. The proposed code, in contrast, offers different choices with respect to the value of h. First, a single value for the entire system can be specified, or each specific particle can be assigned its own value. Regarding the evolution in time, h can be:

- kept *constant*: at each time step, the value of h is equal to that at time $t = 0$, whether unique for the system, h_0, or individual for each particle, h_i;

- evolved *differentially*: the value of the smoothing length for particle i is related to the local value of density by means of the differential equation 2.39:

$$\frac{\mathrm{d}h_i}{\mathrm{d}t} = -\left(\frac{h_i}{\rho_i\, d}\frac{\mathrm{d}\rho_i}{\mathrm{d}t}\right)$$

  where d represents the dimensionality of the problem. This form of evolution in time is allowed only if the density is computed by solving the mass conservation equation 4.2.

- evolved through the *ADKE* technique, previously introduced in 2.7.1. At each time step, a pilot density estimate is computed for all fluid particles in the system. Then, based on this resulting value, all smoothing lengths h_i are updated to a new value. This update procedure can be coupled with both the computations of density: through summation or by integration of the mass conservation equation.

In all cases, when using a variable smoothing length parameter, the code automatically proceeds to symmetrize the density, momentum, and energy equations, by using a modified kernel function $\hat{W}_{ij}$, defined as:

$$\hat{W}_{ij} = W_{ij}\left(\frac{h_i + h_j}{2}\right)$$

In this first iteration of the code, even if h is set as a variable, the momentum equation still conserves the formulation described by equation 4.3. Similarly to what is stated in [48], no terms related to ∇h are included in the equations of motion. This can result in slight inaccuracies in the numerical solution, but allows to keep the computational weight relatively limited. In any case, the ability of an SPH scheme to capture the correct physics of the simulated phenomena is expected to be preserved.

Algorithm 4.3 lists the logic path followed by the code when computing the quantities involved in an update of the SPH state.

Algorithm 4.3 Logical flow of SPH state update

```
if ADKE scheme used then
    Compute pilot density estimate and update h_i
end if
if Density through summation equation then
    Compute ρ_i from current mass distribution
end if
Perform BCE markers extrapolation procedure
Compute SPH derivatives of momentum and energy. Derivative of density depends on the SPH scheme.
if h_i evolved with differential equation then
    Compute its derivative
end if
```

4.3.8. Time integrators

For the time integration of variables $\mathbf{x}, \mathbf{v}, e$ and optionally also ρ and h, the code provides two different explicit time stepping schemes. In the following equations, the expressions for the evolution of the generic variable y, with $\frac{\mathrm{d}y}{\mathrm{d}t} = f(y,t)$, are explicitly reported. y can be intended as any of the variables $\mathbf{x}, \mathbf{v}, \rho, e, h$. The subscripts n and $n+1$ are used to refer to the advancement from the generic time t_n to $t_n + \Delta t$. The equations must be solved for each SPH fluid particle in the system at each time step.

Explicit Euler The first option is the first-order explicit Euler method, which takes the following form:

$$y_{n+1} = y_n + f(y_n, t_n)\Delta t$$

This method is straightforward to implement and is the least computationally expensive. However, it has only first-order accuracy and is subjected to strict stability constraints on the time step Δt.

RK2 A second-order accurate alternative is the explicit two-step Runge-Kutta scheme, or RK2:

$$k_1 = y_n + \frac{\Delta t}{2} f(y_n, t_n)$$

$$y_{n+1} = y_n + \Delta t \; f(k_1, t_n + \frac{\Delta t}{2})$$

Compared to the explicit Euler scheme, the RK2 method is more accurate while still preserving the explicit character. This comes at a computational expense, because an additional evaluation of the derivatives must be performed at each time step.

CFL condition For any choice of the time integrator, the time step must still be chosen carefully. The proposed code offers two choices: the Δt can be set as constant and as a user input, or it can be computed automatically, according to the following expression of the CFL condition:

$$\Delta t_{CFL} = C_{CFL} \min(\Delta t_{vel}, \Delta t_{force}) \tag{4.10}$$

with:

$$\Delta t_{vel} = \frac{h_{min}}{\max(c_s)} \tag{4.11}$$

and:

$$\Delta t_{force} = C_{force} \sqrt{\frac{h_{min}}{\max \left| \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} \right|}} \tag{4.12}$$

where the constants C_{CFL}, C_{force} in equations 4.10 and 4.12 both take the default value of 0.5, but can be changed by the user at the beginning of the simulation. It should be noted that the value of Δt_{CFL} depends on the current properties of the fluid particles in the system, so it has to be explicitly computed at each step of the time integration. Therefore, this procedure enforces strict control on the stability of the numerical scheme, but this comes at the expense of an increased computational weight.

If this option is selected, a global value for the time step must still be provided, and, in this case, the system will advance from time t_n to time $t_n + \Delta t$, with:

$$\Delta t = \min(\Delta t_{input}, \Delta t_{CFL})$$

If the condition $\Delta t_{CFL} < \Delta t_{input}$, the fluid system is evolved multiple times, until the

synchronization point $t_n + \Delta t_{input}$ is reached.

4.3.9. Neighbor search

The SPH scheme involves, when computing a generic property f for the particle i, calculating all the effects on f_i due to fluid particles j. If the kernel functions W do not have compact support, this would imply computing, for each i, all terms resulting from its interaction with each particle j in the system. In the common case of smoothing functions that have a compact support, the summation operation required for the interpolation of f_i would be extended only to those fluid particles j that fall inside the kernel domain. That is, all particles j for which $|\mathbf{x}_i - \mathbf{x}_j| \leq \kappa h$. However, even in such scenario, the determination of which particles are actually within the interaction range of i would require computing all pair-wise distances $|\mathbf{x}_i - \mathbf{x}_j|$. If this intuitive strategy was followed, the computational complexity of the algorithm would be of the order of $O(N^2)$, being N the number of fluid markers in the system. Clearly, such a brute force approach would enforce strict limits on the global number of particles that could be defined to simulate a fluid problem. Moreover, the SPH scheme would become computationally highly inefficient, because the great majority of the simulation time would be devoted to the determination of particle neighbors instead of calculating the flow properties.

Over the years, several approaches have been developed to improve the determination of neighbors in SPH. The proposed code, similarly to Chrono::FSI, although with some modifications, uses the so called cell-linked list (CLL) method, as seen, for instance, in [11]. The idea under this approach is to discretize the computational domain into a set of cells, cubic in three-dimensions, each having a side length equal to the support domain of the chosen kernel function, thus equal to κh. Each cell is assigned a position in the domain expressed with integer coordinates. Then, a *hash function* is provided to give these cells a unique *hash* value. A common example of hash function for a cell i, used in the cell-linked approach, uses the spatial coordinates as:

$$\text{hash}_i = posCell_x + posCell_y \times numCells_x + posCell_z \times numCells_x \times numCells_z$$

So, the hash value depends on the position of the cell i inside the cell grid, and the number of cells along the x and y directions. Then, each particle is assigned the hash of the cell it belongs to, and all of them are sorted from the *original order* into *hash order*.

Due to the spatial extension being equal to the kernel support by construction, to deter-

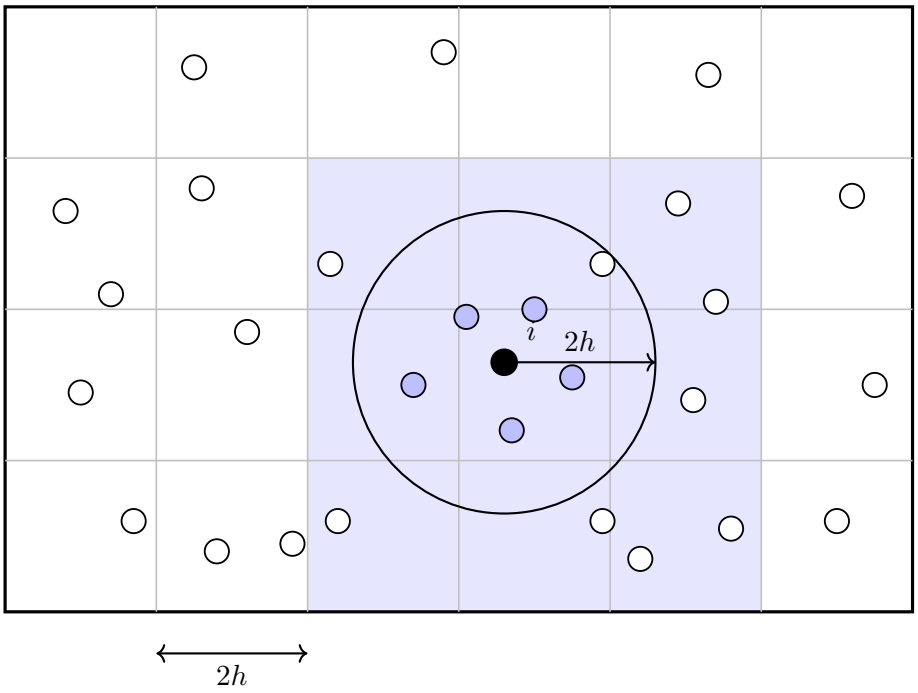


Figure 4.4: Uniform grid used in the neighbor search through the cell-linked list approach.

mine their neighbors, the SPH particles now only need to check the other markers inside their own cell and within adjacent ones. This means that a particle only needs to inspect 3 cells in 1D, 9 in 2D, and 27 in 3D. During this inspection, a total of 5 arrays are created in addition to the original vector in which the particles were stored:

- the *sorted particles* array stores the SPH markers in the domain, ordered by hash value;
- the *cellStart* array stores, for each cell, the index of sorted particles array, corresponding to the first particle within that specific cell;
- the *cellEnd* array is built similarly to the *cellStart* array but stores the index of the last particle belonging to a cell;
- the *numNeighPerPart* array stores, for each particle in hash order, the number of neighbors in adjacent cells that actually satisfy the condition $d \leq \kappa h$;
- the *NeighborList* array, of size equal to the total number of neighbors in the entire SPH system, stores, for each particle, the index in hash order of its neighbors.

Once this process has been completed, the computation of the summations involved in the update of the fluid properties considers only the indexes stored inside the *NeighborList* array, and this is valid for all calculations occurring during the same time step. This

list can be computed at every time step, or at every n steps, depending on the velocities involved in the specific problem. If the velocities are too high, and the list of neighbors is not updated frequently enough, there may be inaccuracies, and some neighbors may be missed. On the other hand, if the particles do not drastically change positions at each time step, a less frequent list update results in a lower computational effort.

Overall, this method allows to drastically reduce the computational effort required, at the expense of higher memory requirements. However, in general, memory constraints are less strict, and the procedure results advantageous.

When the smoothing length is constant and uniform, the grid extension can be computed only once, before the simulation of the dynamics starts. In case of SPH schemes where the smoothing length h is variable, the proposed code at each time step updates the dimensions of the cells, based on the maximum value h_{max} within the system. This ensures that no neighbors are missed, but it comes at the cost of potentially creating inefficiencies, especially in regions with low values of the smoothing length.

4.4. Software architecture

The code presented is written mainly in the C++ programming language. This choice was motivated by two main reasons. First, it ensures full compatibility with Chrono::Engine and Chrono::FSI, which are written in C++. Second, the language provides an efficient and complete memory management ecosystem. The following subsection describes how the code is structured in an object-oriented framework. Since the SPH formulation is naturally well-suited to parallelization, CUDA was employed to exploit the massively parallel computational capability that a modern Graphics Processing Unit (GPU) offers, in order to achieve a significant speed up of the calculations. From a structural perspective, this module for the simulation of inviscid compressible flows is easily integrated into the Chrono ecosystem, thanks to CMake. Integration requires copying the source files folder into the "src" folder of the Chrono source tree, and the addition of a few lines in the corresponding `CMakeLists.txt` file. This module is then compiled together with all the Chrono libraries. Furthermore, similarly to the Chrono::FSI module, the source code has been designed to be independent of the underlying floating-point precision. Therefore, a simulation can be executed using either `float` or `double` data types.

4.4.1. Hierarchy of classes

Chrono::FSI defines three pure virtual base classes, independent of the actual numerical scheme used, that an external CFD module must define:

- `ChFsiFluidSystem`: it is the high-level class, devoted to the management of the fluid system as a whole. It should provide the relevant APIs to add SPH particles to the simulation and to advance the system dynamics in time;
- `ChFsiInterface`: it represents the class that manages the fluid-solid interaction strategy, connecting the CFD solution with the Chrono::Engine MBD system evolution;
- `ChFsiSystem`: it is the highest-level class involved in a coupled fluid-solid simulation. It manages the fluid system, the solid system, and the interface system.

To ensure consistent integration with Chrono and to promote data encapsulation and abstraction, the following set of classes has been implemented:

- the *parameters* class: instantiated only once, after the initialization process, it store and delivers to other classes the requested simulation parameters and constants. Table 4.1 lists some of them.
- the *data manager* class: it manages and stores the arrays of all the quantities connected to the SPH fluid particles and BCE markers in the CFD system, including masses, positions, velocities, densities, and many others.
- the *BCE manager* class: it is responsible for computing the actual quantities involved in the fluid-solid interaction, such as the calculation of the resulting force and torque on a solid body, or the update of the state for each BCE marker;
- the *collision system*: for each particle, it computes the set of actual neighbors, according to the algorithm described in section 4.4.4;
- the *base force* class: pure virtual class that serves as defining the signature of derived methods that compute all the flow-related quantities in an SPH based scheme;
- the *derived force* class: it implements the methods to compute the SPH particles density, momentum, and energy derivatives;
- the *fluid dynamics* class: it provides an API to integrate the system in time, hiding how the above classes are combined when performing such computation.

These classes are the backbone of the presented SPH solver, and are the ones that actually

make all calculations. According to the prescription of Chrono::FSI, the following high level classes are then implemented:

- `ChFsiFluidSystemCSPH`: derived from `ChFsiFluidSystem`;
- `ChInterfaceCSPH`: derived from `ChInterface`;
- `ChFsiSystemCSPH`: derived from `ChFsiSystem`.

The first and last classes from the above list provide all the tools to build the simulation scenario, such as methods to add SPH fluid particles, add BCE markers, add solid bodies, and advance the simulation in time. They also offer functions to print and save results, and they represent the only two classes that the user will have to deal with.

Parameter	Description
RhoEvolution	method to compute density
Hevolution	method to update h
IntegrationScheme	time integration scheme
KernelType	kernel function W
ShiftingMethod	If XSPH correction is to be applied
Δx_0	initial inter-particle separation
h_0	initial kernel length, if uniform
ε	minimum inter-particle distance
$\mathbf{b}$	uniform body force on fluid particles
ρ_0	initial particles density, if uniform
P_0	initial particles pressure, if uniform
u_0	initial particles energy, if uniform
γ	ratio of heat capacities
Δt	reference time step for fluid system
C_{CFL}, C_{force}	constants for CFL condition
DensityReinitSteps	number of steps between density re-initializations
α, β	artificial viscosity coefficients
g_1, g_2	artificial heat conduction coefficients
$\kappa_{ADKE}, \varepsilon_{ADKE}, D_{ADKE}$	coefficients for ADKE method
BoundaryType	boundary conditions types along x, y, z
$\mathbf{c}_{MIN}$	lowest point of computational domain $(x_{MIN}, y_{MIN}, z_{MIN})$
$\mathbf{c}_{MAX}$	highest point of computational domain $(x_{MAX}, y_{MAX}, z_{MAX})$

Table 4.1: Examples of simulation constants stored in the *params* class

4.4.2. CUDA Parallelization

CUDA, or Compute Unified Device Architecture, is a parallel computing platform and programming model, developed by NVIDIA, that enables general purpose computing on GPUs; for detailed information about the CUDA framework the reader can refer to [2, 38]. In addition to the CUDA APIs, a consistent use of the Thrust library [37], included in the toolkit, has been made. Thrust provides a high level C++ template interface, for dynamic data structures and parallel algorithms. It implements STL-like containers to be used on

both the Host, or CPU side, and on the Device, or GPU side. It also enables seamless memory transfers and fast reduction, sort, scan, and transform operations. A function that checks the actual number of cores of the GPU is present, so that the number of threads and blocks launched for each CUDA kernel is independent of the actual hardware used.

4.4.3. Particle data structures and memory management

The proposed code performs the relevant calculations on the GPU, using either a series of lightweight CUDA kernels, which are functions executed in parallel by a large number of cores or *threads*, or the Thrust transform and reduce operations. Therefore, the data structures associated with SPH fluid and solid markers are mostly allocated on the Device side of the computing system. However, the creation of particles and their final post-processing occur on the GPU side. Particles are first inserted into the fluid-solid system, by calling appropriate the methods of the highest-level classes. The user can specify their initial mass, position, velocity, energy, density, pressure, and smoothing length. After the system is initialized, and before the dynamics start to be simulated, the *data manger* class transfers the data to the GPU memory. The data manager class itself is then responsible for their storage and manipulation. In order to favor *memory coalescence*, all properties are stored in different dynamic device vectors, but are grouped by means of Thrust::zip iterators, so that they occupy contiguous memory addresses. During the execution of the algorithm, other auxiliary containers are created on the global memory of the GPU side, a few of them are:

- the arrays for the particles in hash order;
- the array for the neighbor list;
- the array for the derivatives of momentum, energy and, if needed, also density and smoothing length.

In GPU computations, memory bandwidth utilization is often the primary limiting factor for programs performance. To optimize the code, the simulation parameters that remain constant in time and are common to all particles are allocated on the *constant* Device memory before the first time iteration begins. Therefore, when a kernel is launched and requires access to one of these values, it is dispatched at the same time for all *threads*, and the memory access latency has no impact. At the end of each time integration, the updated properties of the particles are then sorted back in the original order, and transferred back to the Host side, so that the user may store or analyze them. In this way, there is one memory transfer at the beginning of the simulation and only one at each

time step.

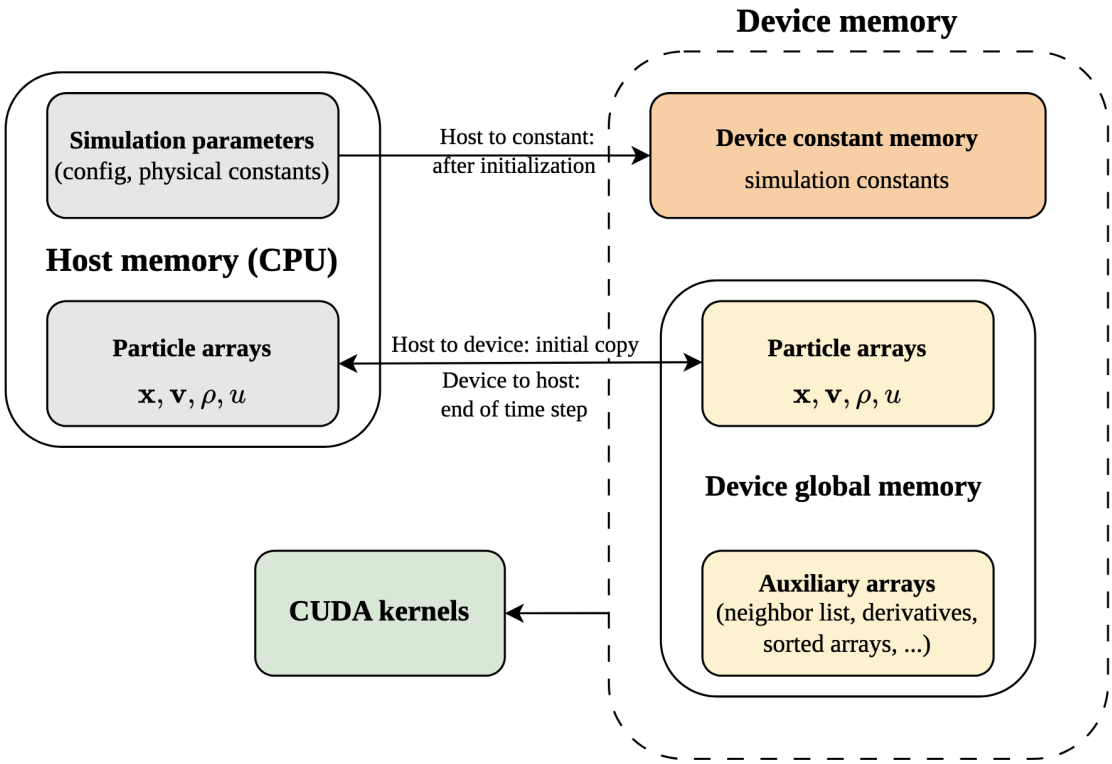


Figure 4.5: Host and device memory flow.

4.4.4. Parallel neighbor search

The neighbor search algorithm, for which its frequency of occurrence is an input parameter, is divided into a set of parallel kernels and sequential functions. With respect to a generic time step t_n, the first operation consists of checking if the smoothing length h has been set as variable in time. If this condition is verified, a new grid domain must be defined. The maximum value h_{max} is retrieved from the system using a Thrust method, and cubic cells of side length κh_{max} are created, filling the whole simulation domain. Successively, a series of specific parallel functions are called to:

1. compute the hash value associated with each particle;
2. compute the *cellStart* and *cellEnd* vectors;
3. store an auxiliary array that maps the hash order back into the original index order;
4. sort all the vectors containing the fluid properties in hash order;
5. compute the *numNeighborsPerPart* array entries.

At this point, a Thrust exclusive scan is carried over the array containing the number of neighbors for each particle. In this way, each entry of the container also represents how many neighbors in total the procedure has identified so far. The last step consists of creating the neighbor list vector, with size equal to the total number of neighboring particles in the system, and launching a kernel function that, for each SPH marker, populates its entries with the correct indices, referring to the hash order particle array.

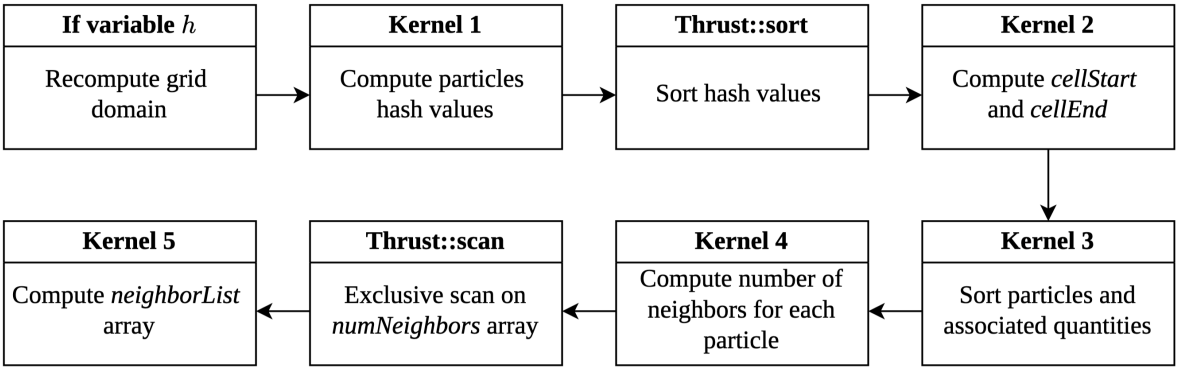


Figure 4.6: Series of operations involved in the parallel neighbor search.

4.4.5. Parallel force evaluation

After the neighbor list has been computed, the code proceeds to compute the SPH quantities and the relevant derivatives needed to integrate the system in time. This force evaluation part, as well as the neighbor search procedure, is divided into a set of CUDA kernels, to be launched sequentially. These parallel functions all work with the arrays sorted by hash, so that data for close particles are stored in contiguous memory addresses, therefore improving the efficiency of device memory access. Depending on the logical switches that are set automatically from the simulation setup, the sequence of operations is described as follows:

1. compute the acceleration of BCE markers from rigid body kinematics, using the current state of solid bodies;
2. depending on the problem parameters, optionally perform the density re-initialization procedure;
3. if evolving h through the ADKE method, update the smoothing lengths before proceeding;
4. if density is computed using the summation equation, call the corresponding CUDA

kernel;

5. process the BCE markers;
6. compute the right hand side of the differential equations. This operation is performed in a single kernel;
7. if the kernel radius h is evolved differentially, compute its derivative.

Algorithm 4.4 reproduces the CUDA kernel that computes the density ρ_i of a particle using the summation equation.

Algorithm 4.4 Pseudo code for the density summation kernel

```
Compute i = threadIdx + blockIdx × blockDim
Retrieve position x_i and h_i from sorted arrays
Retrieve initial and final neighbors indexes as:
First = numNeighPerPart(i)
Last = numNeighPerPart(i + 1)
for j in neigborList(First, Last) do
    Compute relative distance d_ij
    if h is variable then
        Compute average h_ij to symmetrize kernel interaction
    end if
    Compute W_ij
    if particles have same mass then
        access mass from constant memory
    else
        access mass from global memory
    end if
    Accumulate contribution to ρ_i due to particle j
end for
Update ρ_i, pressure, and speed of sound
```

Note that the first `if` condition checks a boolean value stored in global memory, so it is equal for all threads. The second `if` condition is instead templated. This makes the compiler evaluate it at compile time. Therefore, only the correct branch is compiled, and at run-time only the correct statement is executed. All CUDA kernels devoted to the computation of physical quantities are implemented with a similar logic.

After all kernels for the force evaluation process have executed, the system can be inte-

grated in time.

The available time integrators are based on different combinations of a simple Euler step, for which a parallel GPU function has also been implemented.

Depending on the choice of the integrator, different combinations of force evaluations and Euler steps are executed. If the problem velocities are estimated to be extremely high, the user can set the option to make the code update the grid and recompute the neighbor list, also after the mid-step integration in the Runge-Kutta2 scheme.

5 | Simulations and preliminary validation

It is essential for any numerical solver to be validated against some known solutions or by comparison with reliable results. This chapter describes some simulations that have been taken as a reference to study the performance of the proposed CSPH module. The code has been compiled using Microsoft Visual Studio Enterprise 2022, version 17.6.21, with NVIDIA CUDA Toolkit version 12.6. All simulations have been performed on a system equipped with an AMD Ryzen 7 5700x CPU and an NVIDIA GTX 970 GPU. Although the CSPH module has been developed to support both single and double precision arithmetic, this graphics card offers very limited double precision performance. More importantly, it lacks hardware support to perform certain CUDA operations, required in specific Thrust algorithms, when using the `double` data type. Fluid dynamics systems involving highly compressible flows are inherently stiff: perturbations in density or pressure propagate rapidly throughout the domain. Numerical errors can manifest themselves as artificial perturbations, and therefore insufficient computational precision can even have severe effects on the stability of the solution. As a consequence, in the following discussion, the primary goal is not to achieve high-order accuracy. Instead, the objective is to verify whether the code can retain the physical characteristics of the simulated system. As a final remark, the proposed code does not depend on the specific system of units. As long as the equation of state is satisfied and the physical quantities are expressed consistently, any set can be provided. However, as it is common practice in CFD simulations, the following cases are defined in terms of non-dimensional quantities, which should therefore be interpreted as relative to appropriate reference values.

5.1. 3D uniform flow

The first test case discussed is a simple 3D uniform flow, constrained in a solid channel. Although it does not have a true physical meaning or interesting applications, it can provide information about the stability of the SPH scheme with respect to numerical

errors. The fluid is characterized by homogeneous initial quantities: $\rho_0 = 1, P_0 = 1, u = 2.5$, with $\gamma = 1.4$ and a non-zero velocity only along x: $\mathbf{v}_0 = (0.2, 0, 0)$. The SPH particles are arranged in a uniform lattice with initial inter-particle separation $\Delta x = 0.01$. The computational domain has extension $(0.2, 0.1, 0.2)$, it contains 4389 fluid particles and 1386 BCE markers. The boundary conditions are periodic in x and y. The flow is constrained by two sets of BCE particles, placed in three layers in both the positive and negative z directions. The cubic spline kernel has been used, with an initial value of the smoothing length $h_0 = 1.2\Delta x_0$. Table 5.1 shows the two different SPH schemes tested in this scenario.

	ρ **evolution**	h **evolution**
Scheme1:	differential	constant
Scheme2:	differential	differential

Table 5.1: SPH schemes used for the uniform flow simulation.

5.1.1. Uniform with no artificial viscosity

The first test has consisted of simulating the uniform flow with both SPH schemes, without considering any artificial term. Using the RK2 integrator and a time step $\Delta t = 10^{-3}$sec, fully respecting the CFL condition, the simulation rapidly become unstable, leading to a collapse of the system after $t = 0.9$ sec. Figure 5.1 shows the SPH system before the crash. BCE markers are reported in green and have a smaller size with respect to the fluid particles. Note how some fluid has been able to exit the channel.

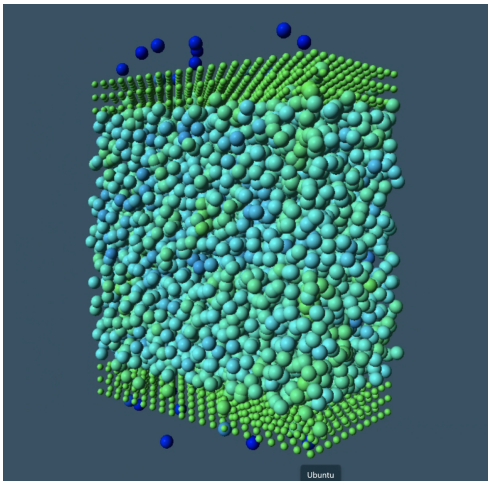
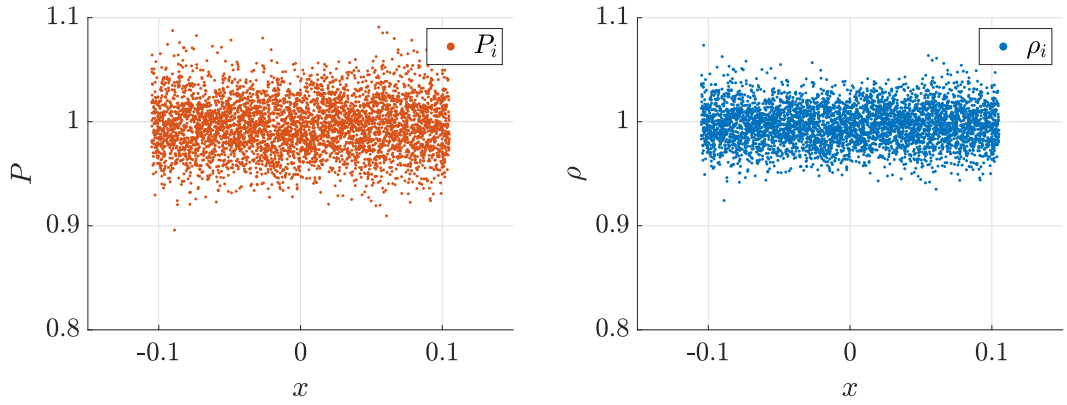


Figure 5.1: Uniform flow with no artificial viscosity at $t = 0.9$ sec.

The use of a variable time step that always respects the CFL condition has not led to any improvements at all. However, lowering it to much below the theoretical value, so using $\Delta t < 10^{-4}$, allowed the system to be simulated, reducing the spurious pressure and density oscillations induced by numerical truncation errors. Figure 5.2 reports the pressure and density distribution of the fluid particles at time $t = 2.5\,\text{sec}$. Even in such very simple scenario, the noise in the solution is relevant.



(a) Pressure distribution.

(b) Density distribution.

Figure 5.2: Pressure and density distribution over the x coordinate. No artificial viscosity and heating. Solution computed at time $t = 2.5\,\text{sec}$.

5.1.2. The effect of artificial viscosity

Including the artificial viscosity term, using the default parameters $\alpha = 1, \beta = 2$, led to an increase in the system stability by contributing to the dissipation of numerical errors. Figure 5.3 shows the state of the uniform flow at time $t = 5\,\text{sec}$. The particle positions show signs of an increasing disorder, but overall the homogeneity of the solution has been retained.

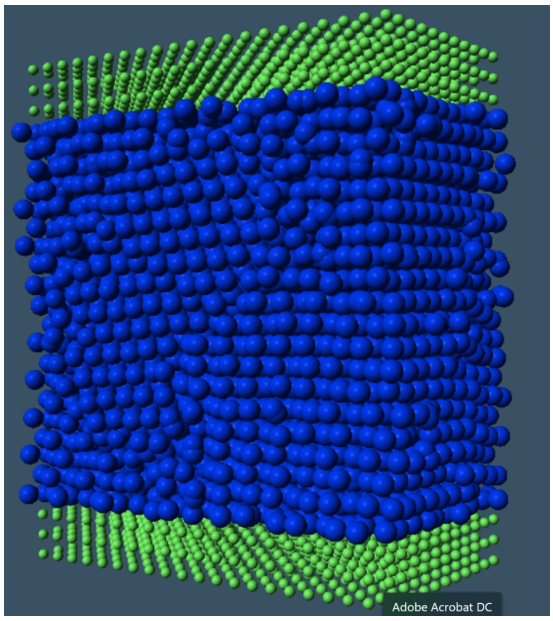
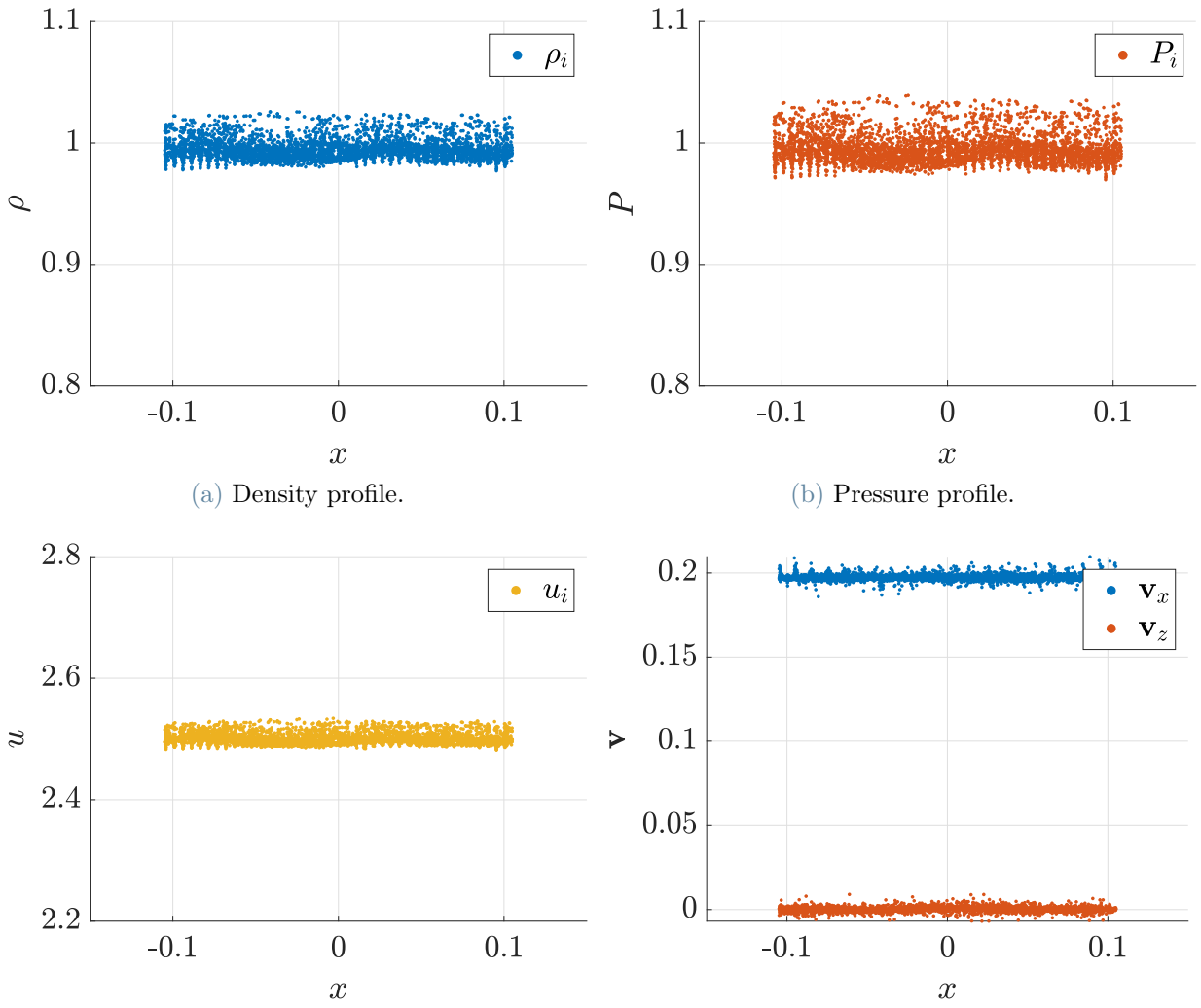


Figure 5.3: Uniform flow with artificial viscosity at $t = 5\,\text{sec}$.

With the additional artificial stabilization, the time step has been set to $\Delta t = 10^{-3}$sec, which is again below the value suggested by the CFL condition, but now sufficiently large to allow a long-time simulation without relevant instability. Figures 5.4 and 5.5 shows the distributions of the fluid properties along the x coordinates, obtained with Scheme 1 and Scheme 2, respectively. Both schemes used the density re-initialization filter and applied it after 1000 time steps. Both solutions show a noticeable noisy behavior, which is an inherent characteristic of 3D SPH simulations. Despite that, deviations from the average values are smaller if compared to the scheme without artificial viscosity. Moreover, the mean properties are only slightly lower than the initial conditions, therefore the artificial term this not produce relevant dissipation in the system as a whole.



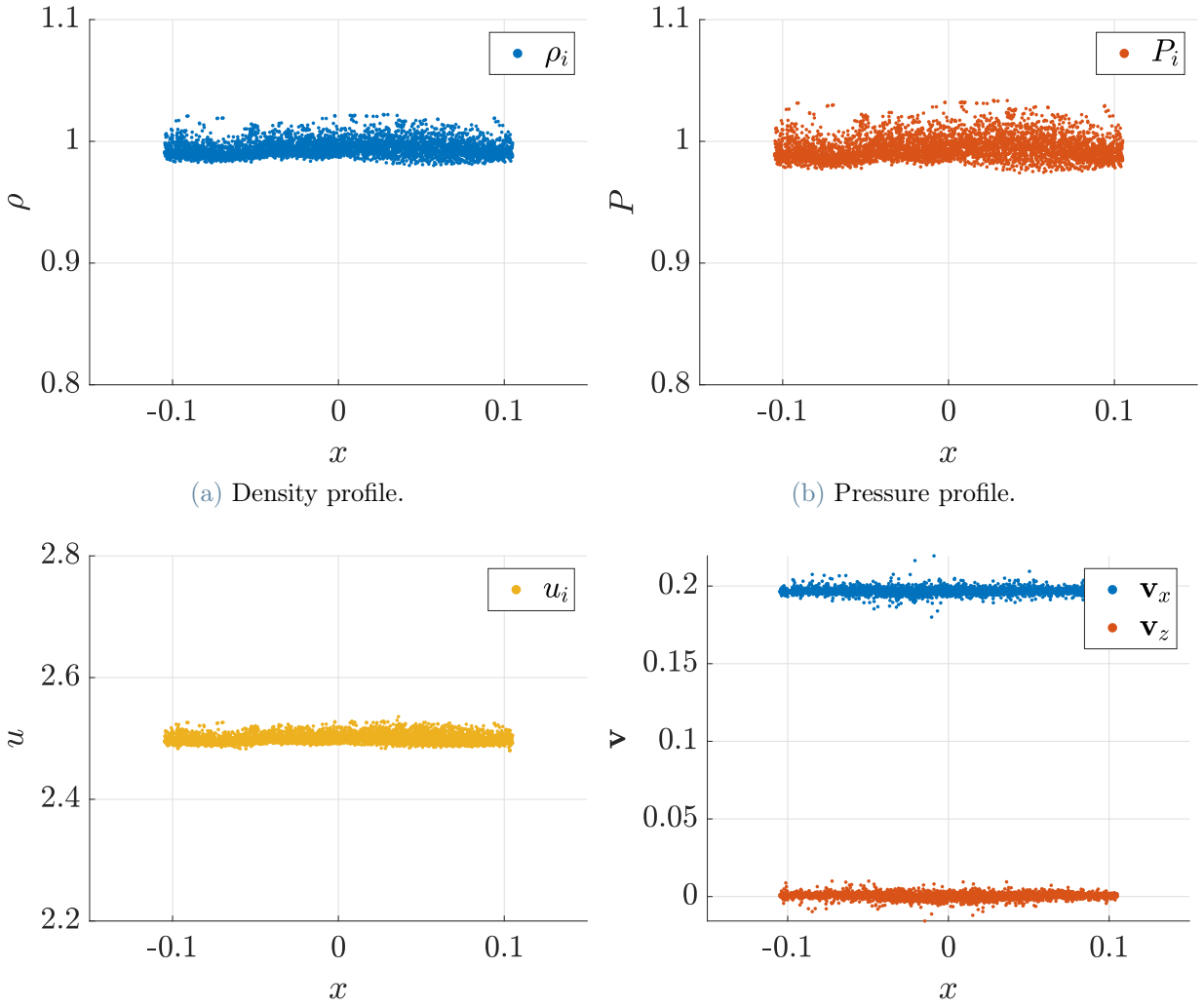
(a) Density profile.

(b) Pressure profile.

(c) Energy profile.

(d) Velocity profile.

Figure 5.4: Results of the uniform flow test case for Scheme 1, using artificial viscosity. Properties computed at $t = 5\,\text{sec}$.



(a) Density profile.

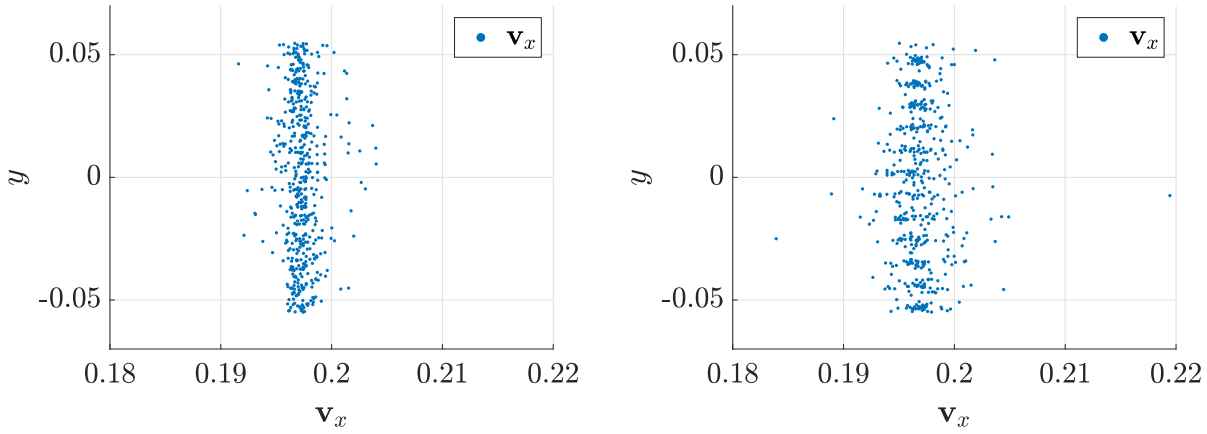
(b) Pressure profile.

(c) Energy profile.

(d) Velocity profile.

Figure 5.5: Results of the uniform flow test case for Scheme 2, using artificial viscosity. Properties computed at $t = 5\,\text{sec}$.

As a last note, figures 5.6a and 5.6b show the velocity profile along the coordinate y for particles having x coordinates in the interval $x = 0 \pm \frac{\Delta x_0}{2}$. Scheme 1 has proven to be more effective, with less deviations from the nominal value. However it is relevant that in both cases the treatment of the BCE markers has been proven consistent with the theory: only the no-penetration condition has been enforced, and no sign of viscosity has been detected in the fluid-solid interaction.



(a) Velocity profile for Scheme 1.

(b) Velocity profile for Scheme 2.

Figure 5.6: Velocity profile along the y coordinate centered around $x = 0$. Both SPH schemes are reported, simulation time $t = 5\,\text{sec}$.

5.2. Sod's shock tube test

A common test used in the literature to validate numerical solvers of inviscid compressible fluids is the 1D Sod's shock tube problem, [49]. The initial conditions describe two states, left and right, of a quiescent gas, with a diaphragm initially placed to separate them. This problem is of particular interest because, once the initial separation has been removed, all the relevant physical phenomena characterizing inviscid fluids appear: a shock wave is developed and propagates rightwards, an expansion fan extends leftwards and a traveling contact discontinuity is generated.

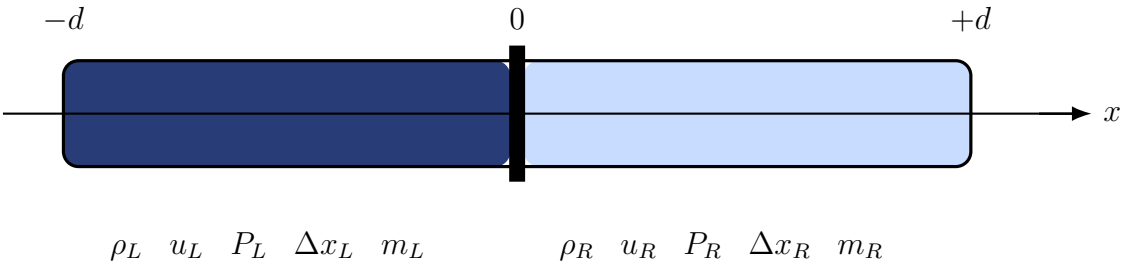
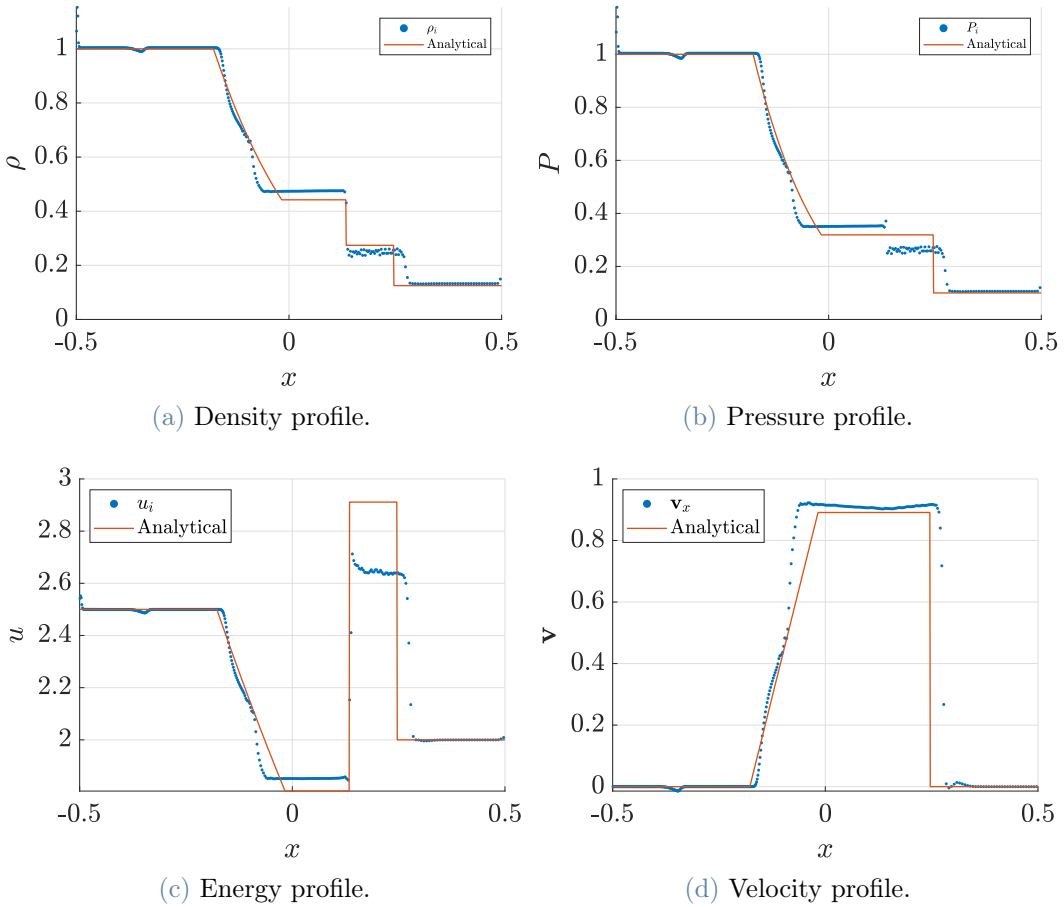


Figure 5.7: Initial states for the Sod's shock-tube problem.

5.2.1. 1D simulation with initial discontinuous conditions

The first test case for the solution of the shock-tube was inspired by the work in [44]. The left state is set as $\rho_L = 1, P_L = 1, \mathbf{v}_L = 0$, with the right state being $\rho_R = 0.125, P_R = 0.1, \mathbf{v}_R = 0$. For both states $\gamma = 1.4$. This simulation uses 400 total SPH fluid particles,

along the x direction, with 320 particles on the left and the remaining 80 ones on the right. This lead to the initial separations being $\Delta x_L = 0.0015625, \Delta x_R = 0.00625$. The left and right states are represented by two different values for the particle masses, set by considering the relation $m = \rho \Delta x$. The simulation have been performed using the cubic-spline kernel functions, and both artificial viscosity and heating have been switched on, with $\alpha = \beta = 1, g_1 = 0.2, g_2 = 0.4$. The specificSPH scheme has been configured to compute the density through the summation equation and the evolution of the kernel radius by means of the ADKE method, with $\kappa = 0.3, \epsilon = 0.5, h_0 = 1.2\Delta x_R$. The solution shown in figure 5.8 has been computed at $t = 1.5\,\text{sec}$, with a time step $\Delta t = 3 \times 10^{-4}\text{sec}$, using the Runge-Kutta 2 integrator.



(a) Density profile.

(b) Pressure profile.

(c) Energy profile.

(d) Velocity profile.

Figure 5.8: Results of the 1D shock-tube test, computed at $t = 1.5\,\text{sec}$.

The accuracy of the results is not high through all properties, in particular pressure experiences a slight jump close to the contact discontinuity, while the internal energy between the shock wave and such discontinuity is underestimated. However, overall the physical characteristics of the exact solution have been retained: the spatial lengths of the

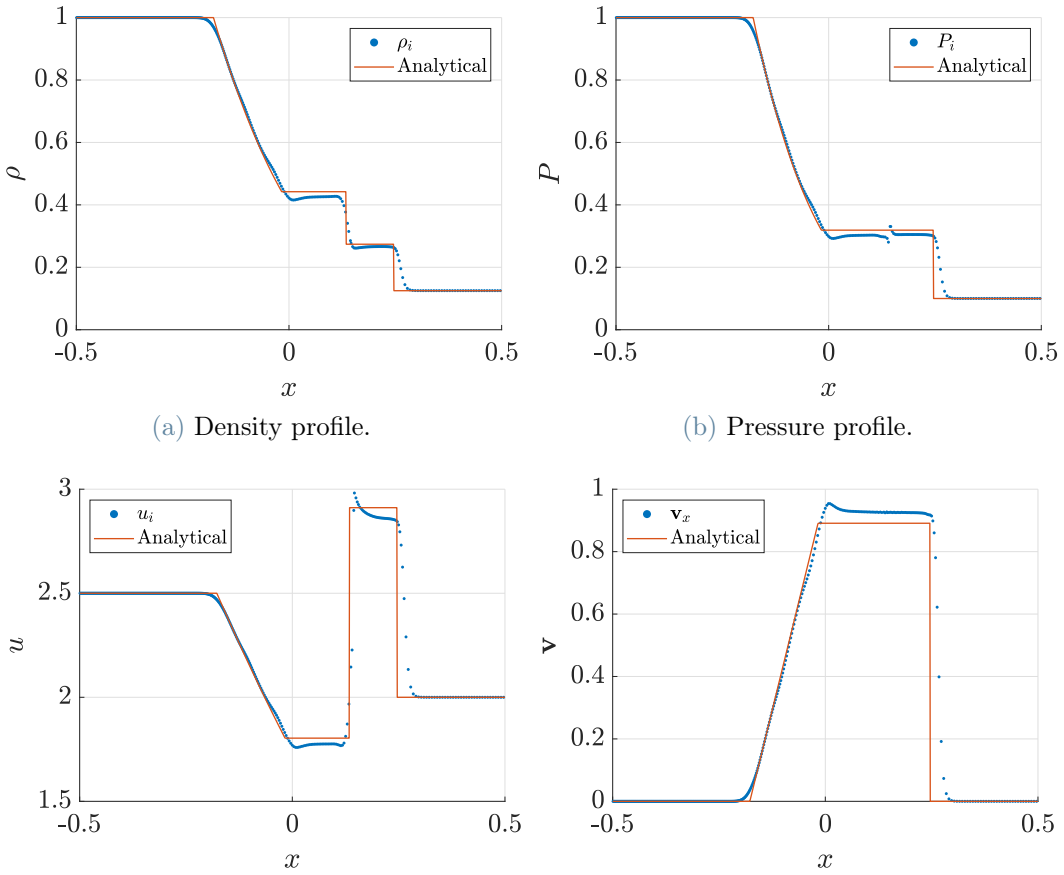
different phenomena are represented correctly, and the approximation of the rarefaction region is satisfying.

5.2.2. 1D simulation with smoothed initial discontinuity

Dealing with initial discontinuous solutions is a complex task even for SPH solvers designed specifically for the study of shock problems. A technique used in the SPH literature to soften these difficulties consists in the application of a filtering function across the discontinuity. Different families of these functions have been proposed over the years. The present work considered an initial smoothing based on a Fermi function. A generic quantity A is modified according to the relation:

$$A(x) = \frac{(A_L - A_R)}{1 + e^{\frac{x}{d}}} + A_R \tag{5.1}$$

where A_L, A_R are the values on the far left and far right of the domain, and the quantity d is usually selected as half the maximum initial inter-particle separation. The smoothing function 5.1 has been applied on the initial conditions of the shock tube discussed previously, and it has allowed the simulation to be performed even by the simplest SPH scheme that could be defined in the CSPH module. Figure 5.9 shows the numerical solution again at $t = 1.5\,\text{sec}$, with density evolved through the summation equation, and the initial smoothing length kept constant at $h_0 = 1.2\Delta x_R$. With this technique, the results are now accurate and precise, only a slight overshoot occurs in the approximation of the region between the shock wave and the contact discontinuity. A part from this, the computed solution provides a correct representation of the physics of the problem.



(a) Density profile.

(b) Pressure profile.

(c) Energy profile.

(d) Velocity profile.

Figure 5.9: Results of the 1D shock-tube test, computed at $t = 1.5\,\text{sec}$, with smoothed initial states, and density computed using the summation equation.

6 | Conclusions and future developments

This thesis has investigated the possibilities of including fluid dynamics effects on small Solar System bodies within a multibody N-body capable numerical solver. Achieving this goal would expand the set of tools available for the simulation of complex astrophysical systems. However, celestial bodies, such as comets, are characterized by a complex interaction of different physical processes, and it is not intuitive which physical model would be the most-suited to provide a proper description of them. Referring to cometary atmospheres, these are formed by gaseous molecules that, sublimating from the nucleus under sufficient Solar illumination, expand in an extremely rarefied environment. Therefore, the most relevant scientific articles devoted to the simulation of cometary comae have been consulted, and it has been found that, under specific circumstances, modeling the expanding gas as a continuum fluid would be an appropriate approximation, especially in the inner atmospheric regions. Since the considered N-body capable code provides an additional optional module for the simulation of fluid-solid systems using the SPH numerical method, the Smoothed Particle Hydrodynamics technique has been reviewed. Such a numerical approach to the solution of fluid equations has been originally developed for the simulation of astrophysical problems, but nowadays it finds application in a number of other physical fields, ranging from incompressible fluids to brittle solids. Due to its Lagrangian nature, SPH is naturally suited to describe the phenomena that occur inside a cometary atmosphere. For instance, it has advantages over traditional Eulerian solvers when treating flows involving shock interactions, void interfaces, or problems with a large variation in the length scales considered. The focus has then been shifted towards the fluid-solid interaction module available for the multibody dynamics solver, that was taken as the initial reference. Furthermore, some of the most relevant open-source SPH solver have been reviewed. Unfortunately, none of these codes has been proven suitable for achieving the original goal of coupling fluid simulators and MBD solvers. Therefore, the work has been shifted towards the development of an SPH code devoted to the simulation of inviscid compressible fluids. The result of this development phase has been discussed

and a preliminary validation phase has been performed. However, in view of future work, some considerations can be made. First, provided the capable hardware is available, a more expensive validation campaign should be performed, with the aim of determining its accuracy capabilities with greater precision. Second, its core formulation should be expanded to include SPH schemes of increasing complexity. In addition, the inclusion of SPH-based open boundary conditions should be regarded as a priority, to provide more flexibility with respect to the type of fluid systems that can be simulated. If these suggestions are implemented, it should then be possible to reach an effective coupling between compressible fluid dynamics and multibody dynamics with N-body gravitational interactions. At this point, a simplified but physically consistent simulation of cometary atmosphere could be actively performed.

Bibliography

[1] S. Adami, X. Hu, and N. Adams. A generalized wall boundary condition for smoothed particle hydrodynamics. *Journal of Computational Physics*, 231(21):7057–7075, 2012. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2012.05.005.

[2] R. Ansorge. *Programming in Parallel with CUDA: A Practical Guide*. Cambridge University Press, Cambridge, 2022. ISBN 9781108827064.

[3] D. S. Balsara. von neumann stability analysis of smooth particle hydrodynamics—suggestions for optimal algorithms. *Journal of Computational Physics*, 121(2):357–372, 1995.

[4] G. A. Bird. *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Oxford University Press, Oxford, 1994.

[5] E. Blazquez. Numerical simulation of n-body asteroid aggregation dynamics. Master of science thesis, Politecnico di Milano, Milan, Italy, 2017.

[6] M. Combi, W. Harris, and W. Smyth. Gas dynamics and kinetics in the cometary coma: Theory and observations. *Comets II*, 01 2004.

[7] J. Crifo and A. Rodionov. Modelling the circumnuclear coma of comets: objectives, methods and recent results. *Planetary and Space Science*, 47:797–826, 1999.

[8] J. Crifo, G. Lukianov, A. Rodionov, G. Khanlarov, and V. Zakharov. Comparison between Navier–Stokes and Direct Monte–Carlo Simulations of the Circumnuclear Coma: I. homogeneous, spherical source. *Icarus*, 156(1):249–268, 2002. ISSN 0019-1035. doi: https://doi.org/10.1006/icar.2001.6769. URL `https://www.sciencedirect.com/science/article/pii/S0019103501967697`.

[9] J. Crifo, G. Loukianov, and Zakharov. Navier–Stokes and Direct Monte Carlo Simulations of the circumnuclear coma II. homogeneous, aspherical sources. *Icarus*, 163: 479–503, 2003.

[10] J. Crifo, M. Fulle, N. Kömle, and K. Szegö. Nucleus–coma structural relationships:

Lessons from physical models. In M. C. Festou, H. U. Keller, and H. A. Weaver, editors, *Comets II*, pages 471–503. University of Arizona Press, Tucson, AZ, 2004.

[11] J. M. Domínguez, A. J. C. Crespo, M. Gómez-Gesteira, and J. C. Marongiu. Neighbour lists in smoothed particle hydrodynamics. *International Journal for Numerical Methods in Fluids*, 2010. doi: 10.1002/fld.2481.

[12] ESA. Giotto halley, 1986. URL `https://www.esa.int/ESA_Multimedia/Images/2016/03/Giotto_approaching_Comet_Halley`.

[13] ESA. Oort cloud and kuiper belt, 2014. URL `https://www.esa.int/ESA_Multimedia/Images/2014/12/Kuiper_Belt_and_Oort_Cloud_in_context`.

[14] I. Federico, S. Marrone, A. Colagrossi, F. Aristodemo, and M. Antuono. Simulating 2d open-channel flows through an sph model. *European Journal of Mechanics - B/Fluids*, 34:35–46, 2012. ISSN 0997-7546. doi: https://doi.org/10.1016/j.euromechflu.2012.02.002.

[15] F. Ferrari, A. Tasora, P. Masarati, et al. N-body gravitational and contact dynamics for asteroid aggregation. *Multibody System Dynamics*, 39(1):3–20, 2017. doi: 10.1007/s11044-016-9547-2.

[16] F. Ferrari, M. Lavagna, and E. Blazquez. A parallel-gpu code for asteroid aggregation problems with angular particles. *Monthly Notices of the Royal Astronomical Society*, 492(1):749–761, 12 2019. ISSN 0035-8711. doi: 10.1093/mnras/stz3458. URL `https://doi.org/10.1093/mnras/stz3458`.

[17] M. Fulle and et al. Evolution of the dust size distribution of comet 67p/Churyumov–Gerasimenko from 2.2 au to perihelion. *The Astrophysical Journal*, 821(1):19, 2016a. doi: 10.3847/0004-637X/821/1/19.

[18] R. A. Gingold and J. J. Monaghan. Smoothed Particle Hydrodynamics: Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, 1977.

[19] W. Hu, M. Rakhsha, L. Yang, K. Kamrin, and D. Negrut. Modeling granular material dynamics and its two-way coupling with moving solid bodies using a continuum representation and the sph method. *Computer Methods in Applied Mechanics and Engineering*, 385:114022, 2021. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.114022. URL `https://www.sciencedirect.com/science/article/pii/S0045782521003534`.

[20] Kitamura and Yoshimi. A numerical study of the interaction between two cometary

jets: A possibility of shock formation in cometary atmospheres. *Icarus*, 86:455–475, 1990.

[21] J. Klapp, L. Sigalotti, F. Peña-Polo, and L. Trujillo. *Strong Shocks with Smoothed Particle Hydrodynamics*, pages 69–79. 09 2012. ISBN 978-3-642-17957-0. doi: 10.1007/978-3-642-17958-7_6.

[22] A. Körösmezey and T. I. Gombosi. A time-dependent dusty gas dynamic model of axisymmetric cometary jets. *Icarus*, 84:118–153, 1990.

[23] H. F. Levison. Comet Taxonomy. In T. W. Rettig and J. M. Hahn, editors, *Completing the Inventory of the Solar System*, volume 107 of *Astronomical Society of the Pacific Conference Series*, pages 173–191. Astronomical Society of the Pacific, 1996.

[24] G. R. Liu and M. B. Liu. *Smoothed Particle Hydrodynamics*. WORLD SCIENTIFIC, 2003. doi: 10.1142/5340.

[25] L. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82:1013–1024, 1977.

[26] R. Marschall. *Inner gas and dust comae of comets*. PhD thesis, University of Bern, 2017.

[27] R. Marschall and et al. Cometary comae–surface links: The physics of gas and dust from the surface to a spacecraft. *Space Science Reviews*, 216:130, 2020. doi: 10.1007/s11214-020-00744-0.

[28] R. Marschall, Zakharov, and et al. Determining the dust environment of an unknown comet for a spacecraft flyby: The case of ESA's Comet Interceptor mission. *Astronomy & Astrophysics*, 666:A151, 2022. doi: 10.1051/0004-6361/202243648.

[29] I. Martínez-Estévez, J. Domínguez, B. Tagliafierro, R. Canelas, O. García-Feal, A. Crespo, and M. Gómez-Gesteira. Coupling of an sph-based solver with a multiphysics library. *Computer Physics Communications*, 283:108581, 2023. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2022.108581. URL `https://www.sciencedirect.com/science/article/pii/S0010465522003009`.

[30] J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30:543–574, 1992.

[31] J. Monaghan. Smoothed Particle Hydrodynamics and its diverse applications. *Annual Review of Fluid Mechanics*, 44(Volume 44, 2012):323–346, 2012. ISSN 1545-4479. doi: https://doi.org/10.1146/annurev-fluid-120710-101220.

[32] J. J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703, jul 2005. doi: 10.1088/0034-4885/68/8/R01.

[33] J. J. Monaghan and J. C. Lattanzio. A refined particle method for astrophysical problems. *Astronomy and Astrophysics*, 149:135–143, 1985.

[34] NASA. Comet ISON, 2013. URL `https://science.nasa.gov/asset/hubble/comet-ison/`.

[35] P. Negi and P. Ramachandran. How to train your solver: Verification of boundary conditions for smoothed particle hydrodynamics. *Physics of Fluids*, 34(11):117125, 11 2022. ISSN 1070-6631. doi: 10.1063/5.0126234.

[36] W. Noh. Errors for calculations of strong shocks using an artificial viscosity and an artificial heat flux. *Journal of Computational Physics*, 72:78–120, 1978.

[37] NVIDIA Corporation. *Thrust: Quick Start Guide*. NVIDIA Corporation, 2023. URL `https://developer.nvidia.com/thrust`. Part of the CUDA Toolkit documentation.

[38] NVIDIA Corporation. *CUDA C++ Programming Guide*. NVIDIA Corporation, 2024. URL `https://docs.nvidia.com/cuda/archive/12.6.0/cuda-c-programming-guide`. Version 12.6.

[39] L. Pinto. The Lagrangian SPH approach applied to the cometary gas-dust emission. In *European Planetary Science Congress 2018 (EPSC 2018)*, volume 12 of *EPSC Abstracts*, pages EPSC2018–409. Copernicus GmbH, 2018.

[40] D. J. Price, J. Wurster, T. S. Tricco, C. Nixon, S. Toupin, A. Pettitt, C. Chan, D. Mentiplay, G. Laibe, S. Glover, C. Dobbs, R. Nealon, D. Liptai, H. Worpel, C. Bonnerot, G. Dipierro, G. Ballabio, E. Ragusa, C. Federrath, R. Iaconi, T. Reichardt, D. Forgan, M. Hutchison, T. Constantino, B. Ayliffe, K. Hirsh, and G. Lodato. Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics. *Publications of the Astronomical Society of Australia*, 35:e031, Sept. 2018. doi: 10.1017/pasa.2018.25.

[41] R. F. Probstein. The dusty gasdynamics of comet heads. In M. A. Lavrentiev, editor, *Problems of Hydromechanics and Continuum Mechanics*, pages 568–583. SIAM, Philadelphia, 1968.

[42] Project Chrono. Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems. `http://projectchrono.org`.

[43] Project Chrono Development Team. Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems. `https://github.com/projectchrono/chrono`.

[44] K. Puri and P. Ramachandran. A comparison of sph schemes for the compressible euler equations. *Journal of Computational Physics*, 256:308–333, 2014. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2013.08.060.

[45] M. Rakhsha, A. Pazouki, R. Serban, and D. Negrut. Using a half-implicit integration scheme for the sph-based solution of fluid–solid interaction problems. *Computer Methods in Applied Mechanics and Engineering*, 345:100–122, 2019. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2018.09.027. URL `https://www.sciencedirect.com/science/article/pii/S0045782518304791`.

[46] P. Ramachandran, A. Bhosale, K. Puri, P. Negi, A. Muta, A. Dinesh, D. Menon, R. Govind, S. Sanka, A. S. Sebastian, A. Sen, R. Kaushik, A. Kumar, V. Kurapati, M. Patil, D. Tavker, P. Pandey, C. Kaushik, A. Dutt, and A. Agarwal. Pysph: A python-based framework for smoothed particle hydrodynamics. *ACM Transactions on Mathematical Software*, 47(4):1–38, Sept. 2021. ISSN 1557-7295. doi: 10.1145/3460773. URL `http://dx.doi.org/10.1145/3460773`.

[47] A. Rodionov, J. Crifo, K. Szegö, J. Lagerros, and M. Fulle. An advanced physical model of cometary activity. *Planetary and Space Science*, 50:983–1024, 2002.

[48] L. D. G. Sigalotti, H. López, A. Donoso, E. Sira, and J. Klapp. A shock-capturing sph scheme based on adaptive kernel estimation. *Journal of Computational Physics*, 212(1):124–149, 2006. doi: 10.1016/j.jcp.2005.07.020.

[49] G. A. Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1–31, 1978. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(78)90023-2. URL `https://www.sciencedirect.com/science/article/pii/0021999178900232`.

[50] V. Springel. Smoothed Particle Hydrodynamics in astrophysics. *Annual Review of Astronomy and Astrophysics*, 48(Volume 48, 2010):391–430, 2010. ISSN 1545-4282. doi: https://doi.org/10.1146/annurev-astro-081309-130914.

[51] V. Springel, R. Pakmor, O. Zier, and M. Reinecke. Simulating cosmic structure formation with the gadget-4 code. *Monthly Notices of the Royal Astronomical Society*, 506(2):2871–2949, July 2021. ISSN 1365-2966. doi: 10.1093/mnras/stab1855. URL `http://dx.doi.org/10.1093/mnras/stab1855`.

[52] R. Stephan. Astrophysical smooth particle hydrodynamics. *New Astronomy Reviews*, 53(4):78–104, 2009. ISSN 1387-6473. doi: https://doi.org/10.1016/j.newar.2009.08.007.

[53] V. Tenishev, M. Combi, and B. Davidsson. A global kinetic model for cometary comae: The evolution of the coma of the rosetta target comet churyumov–gerasimenko throughout the mission. *The Astrophysical Journal*, 685:659–677, 2008.

[54] N. Villodi and P. Ramachandran. Robust solid boundary treatment for compressible smoothed particle hydrodynamics. *Physics of Fluids*, 36(8):086130, 08 2024. ISSN 1070-6631. doi: 10.1063/5.0220606.

[55] D. Violeau. *Fluid Mechanics and the SPH Method: Theory and Applications*. Oxford University Press, 05 2012. doi: 10.1093/acprof:oso/9780199655526.001.0001.

[56] F. S. Whipple. A comet model II. Physical relations for comets and meteors. *The Astrophysical Journal*, 118:464–474, 1951.