UNIVERSITY OF BIRMINGHAM

Note: Please sign this form and present it to the Bindery with your thesis, so that it can be bound at the front of the Library deposit copy

RESEARCH THESIS

Author's Declaration

Full name (block capitals, surname first): ALBANO ANDREA

Full title of thesis/dissertation (block capitals): DISCRETE MULTI-PHYSICS MODEL FOR THE RAYLEIGH COLLAPSE OF A SINGLE CAVITY

College/School/Department (block capitals): SCHOOL OF CHEMICAL ENGINEERING

Date of award of degree (leave blank):

1. I understand that one printed and one electronic copy of my thesis/dissertation (the Work) will be deposited in the University Library (the Library) and in a suitable electronic repository, for permanent retention.
2. Without changing the content, the Library or any third party with whom it has an agreement to do so, may convert either copy into a format or medium for the purpose of long-term retention, accessibility and preservation.
3. The Library will publish, and/or arrange with appropriate third parties for the non-exclusive publication of, a bibliographic description of the thesis/dissertation, and the author's abstract.
4. Unless arrangements are made to the contrary, (see paragraph 6. below), the Library is authorised to make the Work available for consultation in the Library, and via a recognised inter library loans system. The Library is authorised to make the electronic copy of the Work freely accessible to individuals and institutions - including automated agents - via the Internet.
5. Rights granted to the University of Birmingham through this agreement are entirely non-exclusive. I retain all my rights in the Work in its present version or future derivative works.
6. I understand that I may apply to the University to retain the right to withhold access to the content of my thesis/dissertation. Access to the paper version may be withheld for a period which shall not normally exceed four calendar years from the congregation at which the degree is conferred. The actual length of the period will be specified in the application, together with the precise reasons for making that application. The electronic copy may be withheld from dissemination via the web or other networks for any period.
7. I have obtained permission for any use made of substantial amounts of published or unpublished copyright material (text, illustrations, etc) where the rights are owned by a third party, other than as permitted under either The Copyright Designs and Patents Act 1988 (as modified by any related or successor legislation) or the Terms and Conditions of any Licence governing its use.
8. The content of the copies I shall deposit with the Library will be the final version of my thesis, as approved by the Examiners.
9. I understand that the Library and administrators of any electronic theses repository do not hold any obligation to take legal action on behalf of myself, or other rights holders, in the event of a breach of intellectual property rights, or any other right, in the material deposited.
10. I understand that, in the event of my thesis/dissertation being not approved by the Examiners, this declaration will become null and void.

Signature:  **Date:** 15/10/2021

For Library use (please leave blank):

Classmark: **Accession number:** **Control number:**

eTheses Repository url:

DISCRETE MULTI PHYSIC MODEL FOR THE RAYLEIGH COLLAPSE OF A SINGLE CAVITY

by

ANDREA ALBANO

A thesis submitted to the University of Birmingham for the degree of
DOCTOR OF PHILOSOPHY

School of Chemical Engineering
College of Engineering and Physical Sciences
University of Birmingham
April 2021

Abstract

In this thesis, a Discrete Multi-Physics model based on Smoothed Particle Hydrodynamics is developed to simulate a Rayleigh collapse of a single bubble. All the simulations were run on a modified version of the open source software LAMMPS and visualised on OVITO. Initially a 2D model is validated by simulating a phenomenon that shares many similarities with a collapse mechanism, the interaction of a shock wave with a discrete gas inhomogeneity, showing similar performance to classic mesh based CFD. The model is then used to simulate a 2D Rayleigh collapse and validated against the 2D Rayleigh-Plesset equation for both empty and gas filled cavity. The validated model is used to investigate the role of heat diffusion at the gas-liquid interface of the cavity, and to study non-symmetrical collapse induced by the presence of a nearby surface. Enabling heat diffusion at the gas-liquid interface allowed to identify five different possible behaviours that range from isothermal to adiabatic, while the results of non symmetric collapse show that the surface is hit by a stronger shock when distance between the centre of the cavity and the surface is zero while showing more complex double peaks behaviour for other distances. In the final chapter a 3D model is used to model an attached non-symmetrical collapse and its hydrodynamic is compared with the equivalent 2D case.

Acknowledgments

First of all I need to express my gratitude to my supervisor, Alessio Alexiadis, who not only gave me the opportunity to start this journey but he was always available for a meeting, a call or a chat helping me to growth as a researcher.

Likely, this was not a solitary adventure. I feel like I need to thanks all the co-protagonist of the story starting form all the flatmates I had to the people from Chemical engineer passing by the friends of the library and my crew at the Depot climbing gym. I am really grateful to all of you. Because in the end, it was not just about science but more about personal development.

At last, I would like to thanks my family and my friend in Pisa for the understand and the support the gave me through the years. The final and biggest thanks must go to Laura. Her love and support was fundamental for me during the harsh times of 2020.

Contents

List of Figures

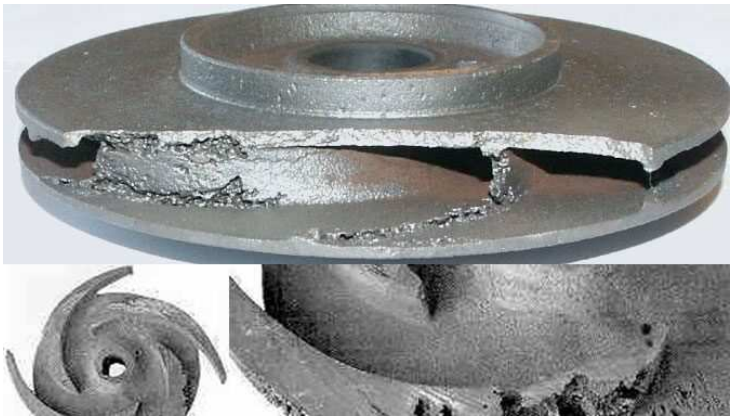
Chapter 1

Introduction

1.1 Background

The term "cavitation" is used to describe a phenomenon composed by two distinct phases: firstly, a vapour cavity, also called vapour bubble or void, nucleates and rapidly growth over a nucleation site in a liquid phase; subsequently, the vapour cavity rapidly collapses generating strong shock waves [Brennen, 2014]. Local temperatures can reach 40,000K [Flannigan and Suslick, 2010] and local pressure peaks of 12GPa [Supponen et al., 2017] causing erosion in many hydraulic and naval applications (see Figure 1.1).

In more than a hundred years, cavitation has been investigated theoretically [Besant, 1859; Rayleigh, 1917; Plesset, 1949; Tomita and Shima, 1986; Kudryashov and Sinelshchikov, 2014], experimentally [Naude and Ellis, 1961; Lauterborn and Bolle, 1975; Philipp and Lauterborn, 1998; Flannigan and Suslick, 2010] and numerically [Plesset and Chapman, 1971; Blake et al., 1986; Kim et al., 2014]. In all cases, assumptions and approximations are necessary to study such a complex phenomenon. Certain features like the role of temperature on erosion are still not very clear and we still lack a model that accounts, at the same time, for the hydrodynamic of the collapse, the compressibility of the gas cavities, the heat transfer at the interfaces (gas-liquid, gas-solid, liquid-solid), the deformation of the solid, and the erosion.
The aim of this thesis is a computer model that accounts for several of these features (i.e. hydrodynamic of the collapse, the compressibility of the gas cavities and the heat transfer at the gas-liquid interface) at the same time. This will be achieved with a Discrete Multi Physic approach [Alexiadis, 2014], where different meshfree particle-based methods are linked together. In many applications, in fact, particle-based methods can be used to model both liquids and



(a)

(b)

(c)

Figure 1.1: *Cavitation erosion pattern in different applications: centrifugal pump (a) impeller blade (b) mechanical valve (c)*

solids [Swegle and Attaway, 1995; Liu et al., 2002; Ariane et al., 2018a; Rahmat et al., 2019b; Mohammed et al., 2020] with specific advantages over traditional, mesh-based, methods especially in the case of large deformations and break up of solids structures [Liu and Liu, 2003]. For this reason, recently researchers had begun to investigate the cavity collapse with particle-methods [Nair and Tomar, 2019; Pineda et al., 2019; Joshi et al., 2019; Albano and Alexiadis, 2020].

1.1.1 Objective of the thesis

The aims of the thesis is to develop a Discrete Multi Physic model that accounts for:

- The hydrodynamic of symmetrical and non-symmetrical collapse.
- Heat generation during the collapse and heat transfer between the gas phase in the cavity and the surrounding liquid.
- The fluid-structure interaction of the collapsing cavity with a nearby solid surface.

1.1.2 Thesis layout

Chapter 2 introduces the concepts of cavitation and bubble dynamics, which are discussed in relation with past theoretical, experimental and computational studies.

Chapter 3 discusses the methodology used in this work. The chapter focuses on mesh free method and especially on Smoothed Particle Hydrodynamics (SPH), which is the method used to simulate the hydrodynamics of the cavity collapse.

Chapter 4 introduces LAMMPS, the software used for the simulations. The chapter shows its structure, how different type of particle interactions can be implemented, and how its source code can be modified.[1]

Chapter 5 develops the SPH model for simulating the interaction between a cylindrical gas inhomogeneities and a travelling shock wave inside a shock tube. The phenomenon shares many similarities with the cavitation collapse and it is used to test and validate the approach used in the following chapters.[2]

Chapter 6 adapts the previous model to simulate a gas-filled cylindrical Rayleigh collapse. It also investigates the role of heat generation and transfer between the gas and the liquid phase during the collapse.[3]

Chapter 7 extends the model to non-symmetrical collapse that occurs when the cavity collapses near a solid surfaces.[4]

Finally, Chapter 8 further extends the model of Chapter 7 to three-dimensional simulations.

[1] This chapter has been published in ChemEngineering as Albano A, le Guillou E, Danz A, Moulitsas I, Sahputra IH, Rahmat A, Duque-Daza CA, Shang X, Ching Ng K, Ariane M, Alexiadis A. How to Modify LAMMPS: From the Prospective of a Particle Method Researcher. ChemEngineering. 2021; 5(2):30

[2] This chapter has been published in Applied Sciences as Albano A, Alexiadis A. Interaction of Shock Waves with Discrete Gas Inhomogeneities: A Smoothed Particle Hydrodynamics Approach. Applied Sciences. 2019; 9(24):5435

[3] This chapter has been published in PLOS ONE as Albano A, Alexiadis A (2020) A smoothed particle hydrodynamics study of the collapse for a cylindrical cavity. PLoS ONE 15(9): e0239830

[4] This chapter has been published in Applied Sciences applied science as Albano A, Alexiadis A. Non-Symmetrical Collapse of an Empty Cylindrical Cavity Studied with Smoothed Particle Hydrodynamics. Applied Sciences. 2021; 11(8):3500

Publication arising from this thesis

The following articles have been published as part of this research

Journal paper

Albano A, Alexiadis A. Interaction of Shock Waves with Discrete Gas Inhomogeneities: A Smoothed Particle Hydrodynamics Approach. Applied Sciences. 2019; 9(24):5435

Albano A, Alexiadis A (2020) A smoothed particle hydrodynamics study of the collapse for a cylindrical cavity. PLoS ONE 15(9): e0239830

Albano A, Alexiadis A. Non-Symmetrical Collapse of an Empty Cylindrical Cavity Studied with Smoothed Particle Hydrodynamics. Applied Sciences. 2021; 11(8):3500

Albano A, le Guillou E, Danz A, Moulitsas I, Sahputra IH, Rahmat A, Duque-Daza CA, Shang X, Ching Ng K, Ariane M, Alexiadis A. How to Modify LAMMPS: From the Prospective of a Particle Method Researcher. ChemEngineering. 2021; 5(2):30

COVID-19 and Discrete Epidemiology

At the start of the COVID-19 outbreak, during the writing up of the thesis, I also collaborate to develop a discrete epidemiology model to predict the spread of infectious diseases within real cities. The work has been published as

Alexiadis, A., Albano, A., Rahmat, A., Yildiz, M., Kefal, A., Ozbulut, M., Bakirci, N., Garzoń-Alvarado, D., Duque-Daza, C., and Eslava-Schmalbach, J. (2021). Simulation of pandemics in real cities: enhanced and accurate digital laboratories. Proceedings of the Royal Society A, 477(2245):20200653.

Chapter 2

Cavitation

2.1 Cavitation inception and cavitation nuclei

An incompressible fluid, under the assumption of steady state inviscid flow, satisfies the Bernoulli equation

$$\frac{v^2}{2} + \frac{P}{\rho} + gz = constant \tag{2.1}$$

where v is the speed of the flow, P the pressure, ρ the fluid density, g gravity acceleration and z elevation. If the flow occurs in a pipe with variable cross section, when the fluid goes from point A to point B its speed increases due to the decrease in cross section (Figure 2.1).

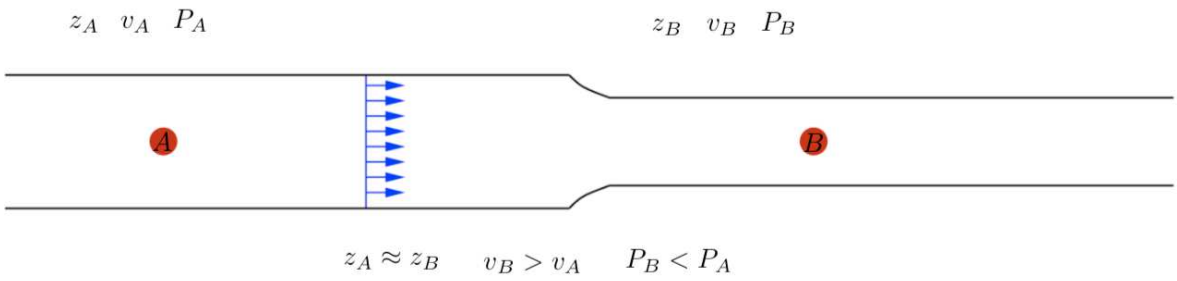


Figure 2.1: *Schematic representation of a fluid flows in a pipe with cross section change.*

According to Equation 2.1, this leads to a decrement in pressure. If P_B goes under the liquid vapour pressure, the fluid would locally evaporate generating a vapour cavity. This process is generally known as *cavitation inception* [Knapp et al., 1970; Rood, 1991; Franc and Michel, 2006; Brennen, 2014; Kim et al., 2014]. If the cavity moves into a high-pressure region, it will collapse. The term *cavitation* is used to describe the phenomenon of growth and sudden col-

lapse of the cavity under the effect of pressure.

This type of cavitation is often referred in the literature as a *travelling bubble cavitation* [De Chizelle et al., 1995; Li and Ceccio, 1996], which occurs in hydraulic machines and pipe flows [Knapp et al., 1970; Luo et al., 2016] and it is the subject of this thesis. However, there are other types of cavitation pattern. *Cavitation cloud* occurs near the leading edge of rudders [Paik et al., 2008; Weber et al., 2010] due to flow separation. *Vortex cavitation* is caused by liquid rotation in a propeller [Arndt et al., 1991; Huang et al., 2014]. Also shear flow [O'hern, 1990; Yu et al., 1995] and other conditions [Young, 1999; Franc and Michel, 2006; Brennen, 2014; Kim et al., 2014] can produce cavitation.

A condition for travelling bubble cavitation is the presence of nuclei in the liquid [Mørch, 2009, 2015]. Those nuclei [Kim et al., 2014] can be micro bubble, particles, or discontinuities at solid surfaces [Holl, 1970]. Nuclei acts as a nucleation site that dynamically reacts to the pressure variation of the system by oscillating and eventually cavitating (growing and rapidly collapsing).

During this oscillation it is common [Brennen, 2014; Kim et al., 2014] to assume that the content of the nucleus behaves as a polytropic gas, this implies that

$$p_B = p_{g,0}\left(\frac{R_0}{R}\right)^{3k} \tag{2.2}$$

where p_B is the pressure in the nucleus, $p_{g,0}$ is the initial pressure, R the radius, R_0 the initial radius and k the so-called polytrophic index. When $k = 1$ the compression or expansion is isothermal, when $k = c_p/c_v = \gamma$ is adiabatic.

2.2 Bubble dynamic

To understand the physics of cavitation we need to understand the dynamic of a single micro bubble that acts as a cavitation nucleus. Let us consider a spherical bubble with a radius $R(t)$, see Figure 2.2, immersed in an infinite liquid domain. Far from the bubble, the temperature is constant (T_∞); while pressure, which is the variable that controls the bubble dynamic, is time dependant $p_\infty(t)$. The liquid density, ρ_L, and dynamic viscosity, μ_L are also constant. The bubble content is homogeneous and uniform so that temperature, $T_B(t)$, and pressure, $p_B(t)$,

are also uniform and homogeneous within the bubble.

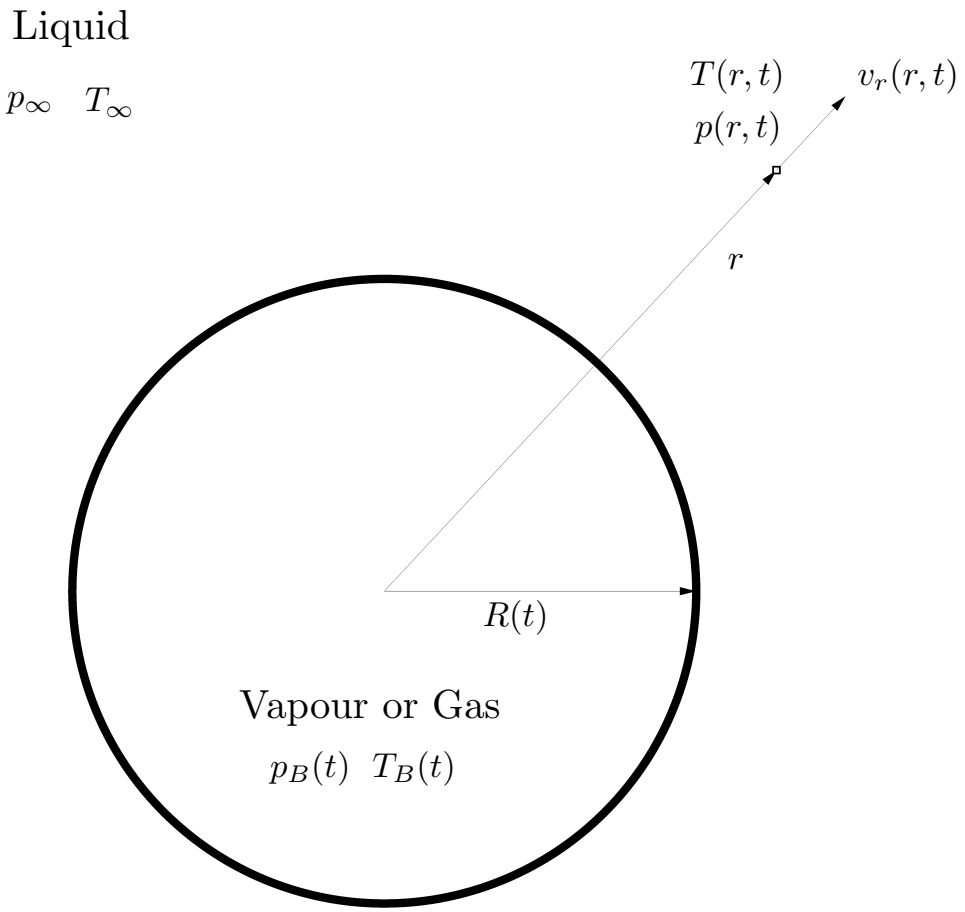


Figure 2.2: *Schematic representation bubble in an infinite liquid domain*

The independent variables are r (distance from the centre of the bubble) and t (time). The dependent variable $v_r(r,t)$ is the radial outward velocity, $T(r,t)$ the temperature and $p(r,t)$ the pressure. If we assume that the bubble only expands or contracts, between R and r, it is possible to write the transport equation as

$$\rho_L v_r(r,t) 4\pi r^2 = \rho_L v_r(R,t) 4\pi R^2(t) \rightarrow v_r(r,t) = \frac{R^2(t)}{r^2} v_r(R,t), \tag{2.3}$$

with no transport of mass through the interface we have $v_r(R,t) = \frac{dR}{dt} = \dot{R}(t)$ and hence

$$v_r(r,t) = \frac{R^2(t)}{r^2}\dot{R}(t). \tag{2.4}$$

Now we write the Navier-Stokes for a Newtonian fluid in the r direction

$$-\frac{1}{\rho_L}\frac{\partial p}{\partial r} = \frac{\partial v_r}{\partial t} + v_r\frac{\partial v_r}{\partial r} - \nu_L\left[\frac{1}{r^2}\frac{\partial^2(r^2 v_r)}{\partial r^2}\right], \tag{2.5}$$

with $v_r = \frac{R^2}{r^2}\dot{R}$ the viscous term goes to zero and we obtain

$$-\frac{1}{\rho_L}\frac{\partial p}{\partial r} = \frac{\partial}{\partial t}\left(\frac{1}{r^2}R^2\dot{R}\right) - \frac{1}{r^2}R^2\dot{R}\frac{\partial}{\partial r}\left(\frac{1}{r^2}R^2\dot{R}\right) - \nu_L\left[\frac{1}{r^2}\frac{\partial^2(r^2\frac{R^2}{r^2}\dot{R})}{\partial r^2}\right] \rightarrow$$

$$-\frac{1}{\rho_L}\frac{\partial p}{\partial r} = \frac{1}{r^2}\frac{\partial}{\partial t}(R^2\dot{R}) - \frac{2(R^2\dot{R})^2}{r^5} - \nu_L\left[\frac{1}{r^2}\frac{\partial^2(R^2\dot{R})}{\partial r^2}\right] \rightarrow$$

$$-\frac{1}{\rho_L}\frac{\partial p}{\partial r} = \frac{1}{r^2}\frac{\partial}{\partial t}(R^2\dot{R}) - \frac{2(R^2\dot{R})^2}{r^5}. \tag{2.6}$$

Equation 2.6 can be integrated between $p \rightarrow p_\infty$ and $r \rightarrow \infty$ giving

$$\begin{aligned} -\frac{1}{\rho_L}\int_p^{p_\infty} dp &= \int_r^\infty \left(\frac{1}{r^2}\frac{\partial}{\partial t}(R^2\dot{R}) - \frac{2(R^2\dot{R})^2}{r^5}\right)dr \rightarrow \\ \frac{p - p_\infty(t)}{\rho_L} &= \frac{1}{r}\frac{\partial(R^2\dot{R})}{\partial t} - \frac{1}{2}\frac{(R^2\dot{R})^2}{r^4}. \end{aligned} \tag{2.7}$$

The viscous term originally present in the Naiver-Stokes have vanished after substituting Equation 2.3 into Equation 2.4. In fact, the contribution of viscosity in the bubble dynamic comes from the boundary conditions at the gas-liquid interface. To determine the boundary conditions we need to balance the forces over a small control volume (Figure 2.3) of infinitesimal thickness, which contains a small portion of interface.

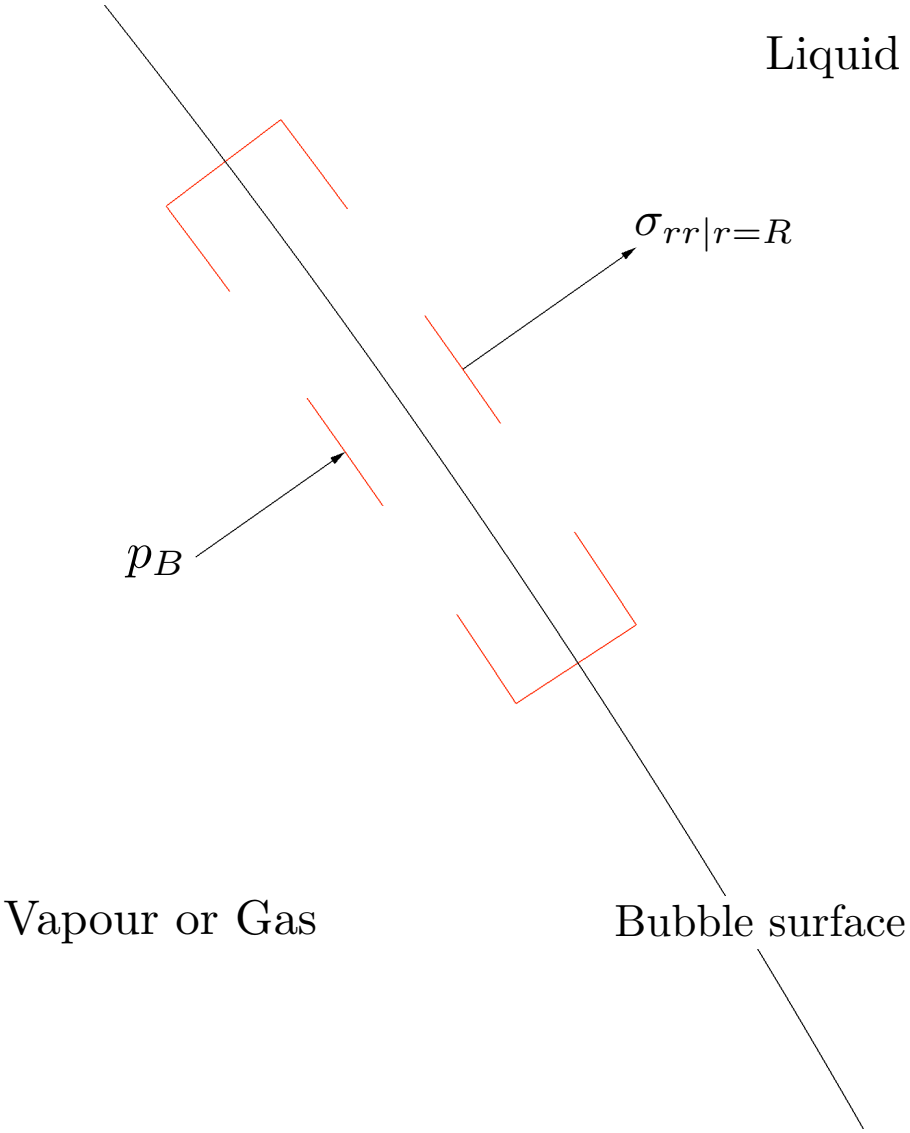


Figure 2.3: *Portion of the vapour/liquid interface*

The net force, per unit area, on this volume in the positive r direction is

$$\sigma_{rr}|_{r=R} + p_B - \frac{2S}{R} = 0, \tag{2.8}$$

with S surface tension of the cavity. The normal stress can be written as $\sigma_{rr} = -p + 2\mu_L \frac{dv_r}{dr} = -p + 2\mu_L(-\frac{2}{r^3}R^2\dot{R})$ and Equation 2.8 becomes

$$p_B - p|_{r=R} - 4\mu_L \frac{\dot{R}}{R} - \frac{2S}{R} = 0. \tag{2.9}$$

The zero in the right side of Equation 2.9 describe the absence of mass transfer across the

boundary. Substituting Equation 2.9 in Equation 2.7 for $r = R$ we obtain the Rayleigh-Plesset equation:

$$\frac{p_B - P_\infty(t)}{\rho_L} = R\frac{\partial^2 R}{\partial t^2} + \frac{3}{2}\left(\frac{\partial R}{\partial t}\right)^2 + \frac{4\nu_L}{R}\frac{dR}{dt} + \frac{2S}{\rho_L R} \tag{2.10}$$

If $p_\infty(t)$ and $p_B(t)$ are known the equation can be solved for $R(t)$. Equation 2.10 has been derived by Plesset [Plesset, 1949] as an extension of the study done by Rayleigh [Rayleigh, 1917] on the collapse of an empty cavity (with $p_B = 0$, $S = 0$) under the action of a constant pressure p_∞.

When $p_\infty < p_B$ the cavity grows up to a maximal radius (*cavity growth*). On the contrary, when $p_\infty > p_B$ the cavity shrinks until a minimal radius is reached (*cavity collapse*). The collapse phase is generally more studied because it produces strong shock waves [Vogel et al., 1996] that impact on nearby solid surfaces causing damage and material loss in a process known as *cavitation erosion* [Karimi and Martin, 1986; Philipp and Lauterborn, 1998; Kim et al., 2014]. When the driven force of the collapse is the pressure difference between the surrounding liquid and the cavity, as described by Equation 2.10, the collapse is referred as *Rayleigh collapse* to differentiate it from another collapse mechanism called *shock-induced collapse*, see Section 2.2.2.

In Equation 2.10, the viscosity and the surface tension terms are often neglected since their order of magnitude is smaller than that of the inertial term [Brennen, 2014]. Figure 2.4 shows the dimensionless solution of Equation 2.10 combined with of Equation 2.2 for cavity growth (a) and cavity collapse (b) under the action of a constant pressure p_∞ with $k = 1$ and neglecting viscosity and the surface tension.
The dimensionless radius is defined as $R(t)/R_0$, where R_0 is the initial cavity radius, while the dimensionless time as t/t_{TC} with t_{TC} the Rayleigh collapse time [Rayleigh, 1917], defined as

$$t_{TC} = 0.915 R_0 \left(\frac{\rho_L}{p_\infty - p_B}\right)^{\frac{1}{2}}. \tag{2.11}$$

Figure 2.4 shows periodic contractions and expansions of the cavity; When $p_\infty < p_B$, the bubble starts to growth and its pressure decreases as the volume increases up to a maximal value, R_{max}. After reaching R_{max} the bubble "bounces" back to R_0 because of the higher liquid pressure. When $p_\infty > p_B$ the bubble collapses and raises its pressure until a minimal volume of value R_{min}. Also in this case, because of the increment of p_B, the cavity "bounces" back to R_0. In this

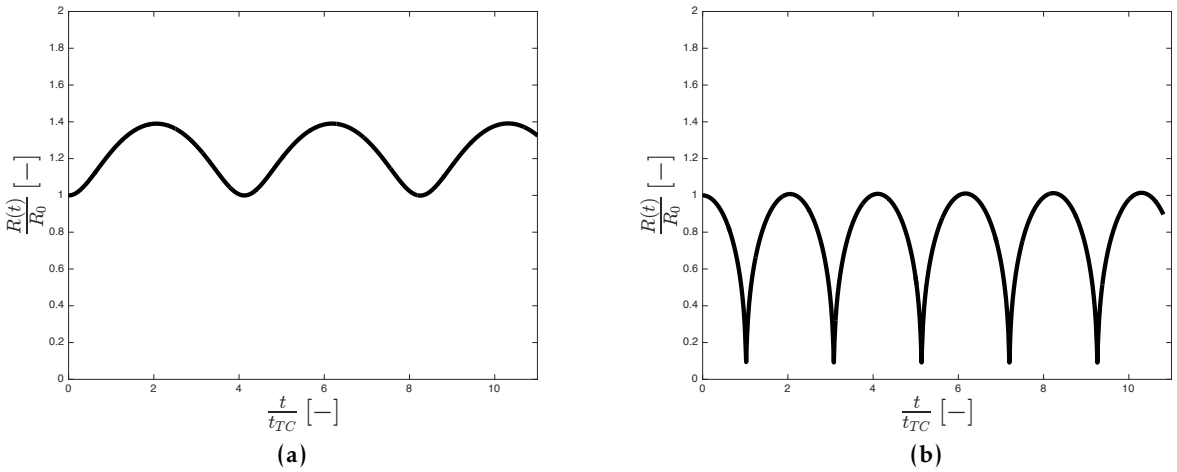


Figure 2.4: *Dimensionless solution of the Rayleigh-Plesset with* $p_\infty = 101$ *[KPa],* $p_{g,0} = 202$ *[KPa],* $\rho_L = 1000$ *[Kg* m^{-3}*],* $R_0 = 100\,[\mu m]$*,* $t_{TC} = 100$ *(a) and with* $p_\infty = 101$ *[KPa],* $p_{g,0} = 3.55$ *[KPa],* $\rho_L = 1000$ *[Kg* m^{-3}*],* $R_0 = 100\,[\mu m]$ *(b)*

periodic behaviour t_{TC} is the half period and indicates the time required for a single collapse.

2.2.1 2D bubble dynamic

The Rayleigh-Plesset equation is often used to validate the hydrodynamic of computational models [Brennen, 2014; Beig et al., 2018; Joshi et al., 2019]. Due to the computational cost of 3D models, cavitation is also simulated in 2D [Chen, 2010; Pineda et al., 2019; Nair and Tomar, 2019; Albano and Alexiadis, 2020]. In this case, Equation 2.10 is not longer valid and an alternative Rayleigh-Plesset for 2D is required for validation purposes. The 2D Rayleigh-Plesset equation can be derived by studying the same system described in Section 2.2 with a plane circular bubble instead of spherical. In this case the transport equation can be written as

$$\rho_L v(r,t)\pi r = \rho_L v(R,t)\pi R(t) \rightarrow v(r,t) = \frac{R(t)}{r} v(R,t). \tag{2.12}$$

As before, we assume that the bubble can only expand or contract (no rotation), and we derive the Navier-Stokes as

$$-\frac{1}{\rho_L}\frac{\partial p}{\partial r} = \frac{\partial v_r}{\partial t} + v_r\frac{\partial v_r}{\partial r} - \nu_L\left[\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial (r v_r)}{\partial r}\right)\right]. \tag{2.13}$$

Integrating between $p \rightarrow p_\infty$ and $r \rightarrow \infty$, neglecting surface tension and viscous effect, we obtain

$$\frac{P_\infty - p_B}{\rho_L} = \left(\left(\frac{\partial R}{\partial t}\right)^2 + R\frac{\partial^2 R}{\partial t^2}\right) ln\left(\frac{R}{r_\infty}\right) + \frac{1}{2}\left(R\frac{\partial R}{\partial t}\right)^2 \left(\frac{1}{R^2} - \frac{1}{r_\infty^2}\right) \tag{2.14}$$

2.2.2 Non symmetrical collapse

Equation 2.10 and 2.14 describe the dynamic of, respectively, a spherical and a cylindrical bubble under the action of an isotropic pressure field. In this case, cavities preserve their symmetry during collapse or growth. However, in many situations the pressure field cannot be considered isotropic because of the presence of *anisotropic drivers* such as gravitational field, nearby rigid or free surfaces, stationary potential flow, liquid interfaces, or inertial boundaries [Supponen et al., 2016]. When the driving pressure of a Rayleigh collapse is anisotropic, the symmetry of the collapse is broken generating a high-speed re-entrant jet (Figure 2.5).

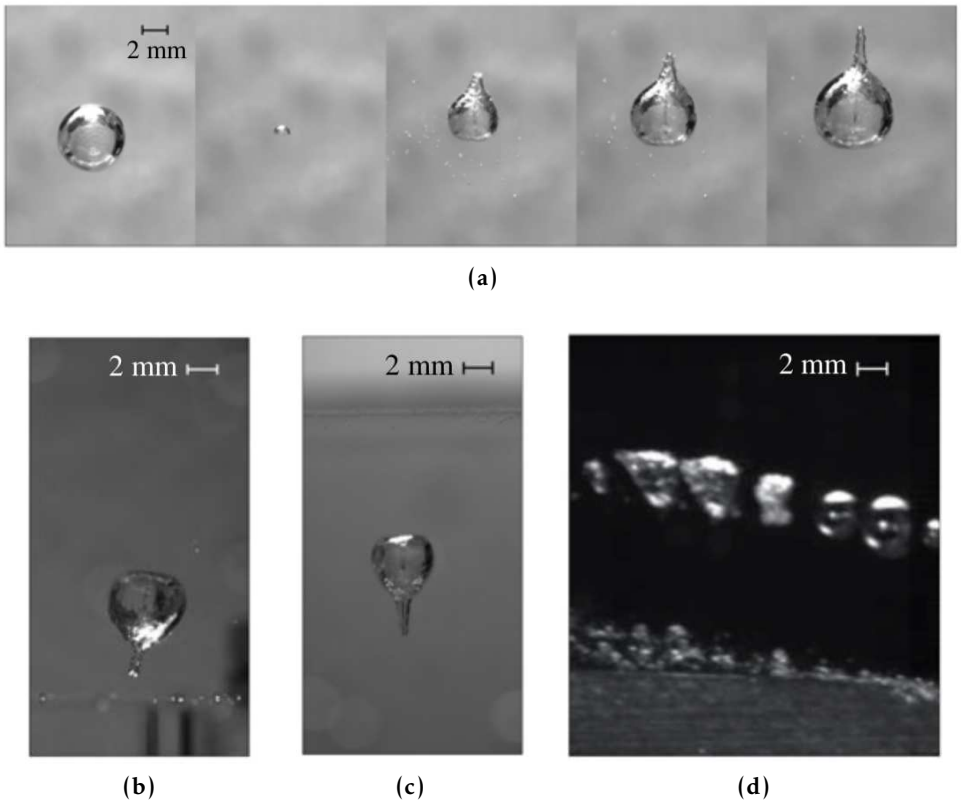


Figure 2.5: *Jet formation for different anisotropic drivers: gravitational field (a), nearby rigid surface (b) nearby free surface (c), stationary potential flow (d) [Tinguely, 2013]*

Alternatively, the symmetry of the collapse can also be broken when a shock wave passes through the cavity (shock-induced collapse) [Bourne and Field, 1992; Johnsen and Colonius, 2008; Turangan et al., 2017; Joshi et al., 2019]. This also results in a re-entrant jet, but shock induces collapse and Rayleigh collapse are considered different phenomena since the mechanic

of the jet formation is different [Johnsen and Colonius, 2009].

Through the years, researchers have investigated the anisotropic pressure field generated by a nearby rigid surface [Plesset and Chapman, 1971; Lauterborn and Bolle, 1975; Li and Ceccio, 1996; Beig et al., 2018; Pineda et al., 2019]. In fact, in this case the cavity folds in the direction of the surface [Supponen et al., 2016]. Eventually, if the cavity is initially attached to the surface, the jet will hit the surface generating a strong water hammer impact, which produces cavitation erosion [Philipp and Lauterborn, 1998; Joshi et al., 2019].

2.3 Theoretical, Experimental and Numerical studies

Since the first theoretical study of cavitation by Besant [Besant, 1859], countless researchers have investigated the phenomenon theoretically, experimentally and computationally. The theoretical work of Besant has been improved by Rayleigh [Rayleigh, 1917], Plesset [Plesset, 1949] (see Equation 2.10), and many other researchers that developed more complex model of cavitation taking into account liquid compressibility, heat transfer and non symmetrical collapse [Plesset and Prosperetti, 1977; Epstein and Keller, 1972; Keller and Miksis, 1980; Brennen, 2014]. Recently, Kudryashov and Sinelshchikov were able to obtain the first analytical solution of the Rayleigh equation for both empty and gas-filled cavities [Kudryashov and Sinelshchikov, 2014, 2015].

Reproducing and studying the collapse of a single bubble with an experimental set up has been a challenge because of the combination of short timescale of the collapse, small bubble dimension and the difficulty to generate a single spherical bubble in a controlled environment [Kim et al., 2014]. One of the first experimental approaches for studying the collapse of a single bubble was based on the so-called *spark discharge method* [Naude and Ellis, 1961; Benjamin and Ellis, 1966; Kling and Hammitt, 1970]: A single bubble is produced using a electric discharge between two electrodes that turns a small portion of water into plasma that eventually generates a vapour cavity. The combination of the spark discharge method with high-speed photography allowed researchers to confirm findings from theoretical studies. For instance, Benjamin and Ellis [Benjamin and Ellis, 1966] confirmed that a collapsing bubble migrates toward a solid boundary and generates a high speed flow (jet) when the bubble approaches the minimal volume as predicted by Kornfeld and Suvorov [Kornfeld and Suvorov, 1944].

One of the limitations of the spark discharge method is that the electrodes affects the pressure field near the cavity and this may lead to the loss of sphericity. *Laser Induced Cavitation* (LIC) has been developed to overcome this limitation [Lauterborn and Bolle, 1975; Vogel et al., 1989; Philipp and Lauterborn, 1998; Flannigan and Suslick, 2005, 2010]. A laser beam is directed on a single location generating a hot plasma spot that will eventually evolves into a vapour cavity. The pressure field is not affected by the presence of the electrodes preserving the sphericity of the bubble. By means of LIC, Philipp and Lauterborn [Philipp and Lauterborn, 1998] studied the collapse of a single bubble at different distances from an aluminium surface. They found that when the cavity is attached to the surface, the re-entrant jet plays a main role in the erosion process. Supponen et al. [Supponen et al., 2017] studied the broad luminescence spectrum obtained from individual collapses of laser-induced bubbles in water recording plasma temperature in the range of 7000K to 11500K.

As mentioned in Section 2.2.2, shock-induced collapse is an alternative mechanism of collapse occurring when the cavity interacts with a travelling shock wave. This collapse mechanism share many similarities with the shock wave interaction with a discrete light gas inhomogeneity [Kim et al., 2014]. The shock interaction with gas inhomogeneities has been studied for decades in the framework of shock propagation in non uniform media [Haas and Sturtevant, 1987; Layes et al., 2005; Layes and Le Métayer, 2007; Wang et al., 2015; Albano and Alexiadis, 2019]. In this case, the cavity and the environment are filled with two different types of gas. And, therefore, instead of a cavity we have an inhomogeneity. It is relatively easy to control the shape of the inhomogeneity by inflating gas into supports of different shapes inside a shock tube.

Because of practical difficulties in the experiments, researches started to study the cavitation process with numerical experiments. There are countless studies in the field based on traditional grid based methods. Among them, Plesset and Champan used the *particle-in-cell method* to study the effect of a nearby surface on cavity collapse finding, for attached cavity collapse, impact stress of order of 2000 atm [Plesset and Chapman, 1971], Blake et al. [Blake et al., 1986] used the *boundary integral method* for simulating the growth and collapse of transient vapour cavities near a rigid boundary in the presence of buoyancy forces and an incident stagnation-point flow. Johnsen and Colonius [Johnsen and Colonius, 2009] used a *high-order accurate shock-*

and interface-capturing scheme to study non spherical collapses for both the Rayleigh and the shock induced collapse and the interaction of the developed shock waves with a nearby surface. Beig et al. [Beig et al., 2018] combined the *interface-capturing scheme* to with a *semi-analytical heat transfer model* to study the temperature developed in a non spherical Rayleigh collapse and to predict the temperature of a nearby solid.

Only in the recent years researches began to use mesh-free particle methods, that show advantages over mesh-based method in the case of large liquid deformations and break up of solid structures, to study cavitation erosion. Joshi at al. [Joshi et al., 2019] developed a *Smoothed Particle Hydrodynamics* axisymmetric solver to simulate a shock-induced collapse of an empty cavity and the erosion process of a nearby elastic-plastic material. Pineda et al. [Pineda et al., 2019] used a *Smoothed Particle Hydrodynamics-Arbitrary Lagrangian Eulerian* model to simulate a gas filled cylindrical cavity collapse far and near a surface studying the emission of shock waves and the jet formation. Nair and Tomar [Nair and Tomar, 2019] used a compressible-incompressible *Smoothed Particle Hydrodynamics* model to simulate different bubble flow problems. Albano and Alexiadis [Albano and Alexiadis, 2020] developed a *Smoothed Particle Hydrodynamics* model to study the role of the heat diffusion at the gas-liquid interface in pressure and temperature peaks inside a cylindrical gas filled cavity.

The next Chapter introduces the methodology used in this thesis. In particular it focuses on the mesh-free particle method *Smoothed Particle Hydrodynamics* (SPH) used to model the dynamic of bubble collapse.

Chapter 3

Methodology

3.1 Numerical simulation

The aim of a numerical simulation is to reproduce a specific physical phenomenon by solving a set of governing equation [Bratley et al., 2011]. In fluid dynamics, those equation are derived by conservation laws of specific field variables such as mass, momentum or energy [Bird, 2002; Mauri, 2015]. In many cases, the governing equation express the evolution of these variables as sets of partial differential equations (PDE), boundary conditions (BC) and initial conditions (IC) [Liu and Liu, 2003].

These equations must be discretised in a set of algebraic equations or ordinary differential equations, which can be solved using the existing numerical routines. Finally, it must be verified that the results reproduce experimental data, theoretical solutions, or results from other established methods [Bratley et al., 2011; Liu and Liu, 2003].

Another important concept in numerical simulation is the domain discretisation. Domain discretisation is essentially a sectioning process where the continuum is divided in a grid composed by a finite number of smaller cells connected to form a mesh (see Figure 3.1), which acts as a computational frame for the numerical solution [Bratley et al., 2011].

3.1.1 Eulerian vs Lagrangian approach

There are two main mathematical approaches to describe the governing equations: *Eulerian* and *Lagrangian*. The Eulerian approach is a spacial description while the Lagrangian approach is a material description [Batchelor and Batchelor, 2000]. The difference can be expressed with

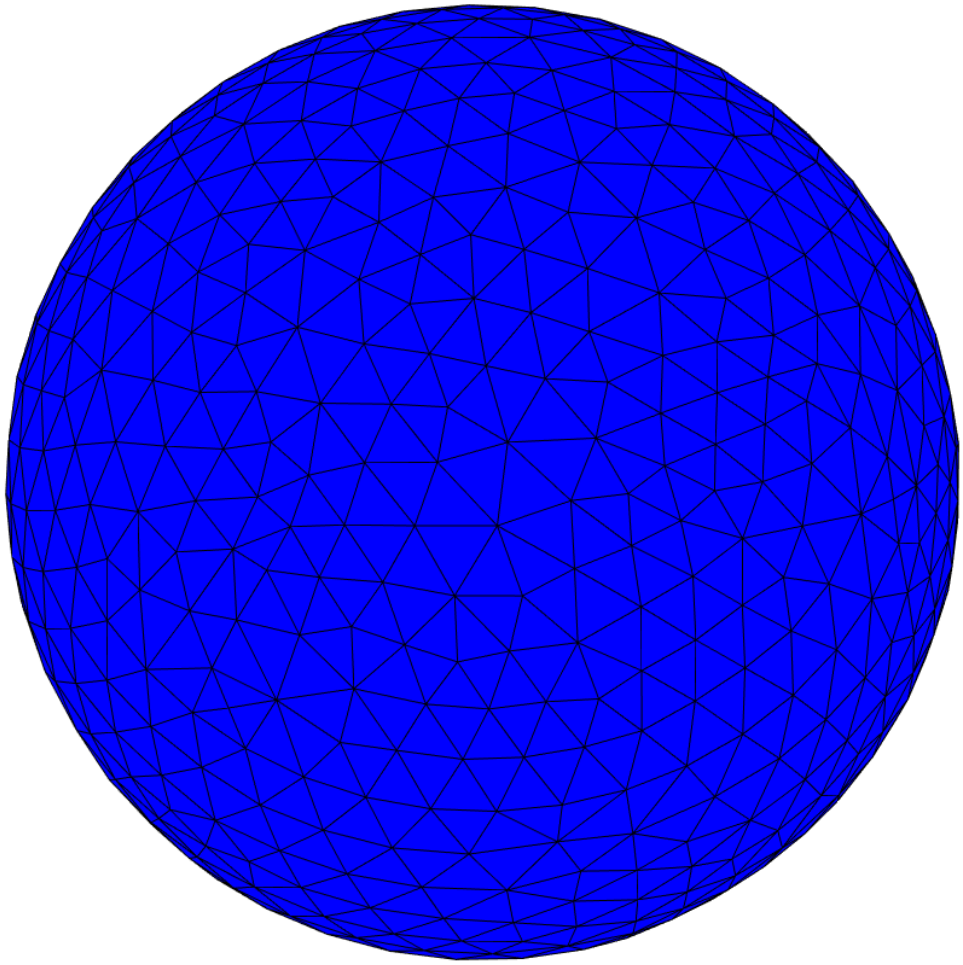


Figure 3.1: *A 3D sphere domain mesh discretisation.*

the definition of the so-called *Lagrangian derivate* [Mauri, 2015]: Given a tensor, scalar or vector $f(x,t)$, whose value only depends on position and time, its Lagrangian derivate is defined as

$$\frac{Df}{Dt} = \frac{\partial f}{\partial t} + v \cdot \nabla f, \tag{3.1}$$

where df/dt is the eulerian derivate which expresses the change of f in respect of time while $v \cdot \nabla f$ expresses the variation of f between two points. When we use the Lagrangian approach, we study the motion assuming a moving frame of reference that follows an individual fluid parcel moving in both time and space (see Figure 3.2a). On the other hand, with the Eulerian approach, we study the motion within a space-fixed frame of reference (see Figure 3.2b).
This naturally leads to different type of grids: the *Lagrangian grid* [Zienkiewicz et al., 2000] is fixed to the material during the computational process while the *Eulerian grid* [Benson, 1992] is fixed in space.

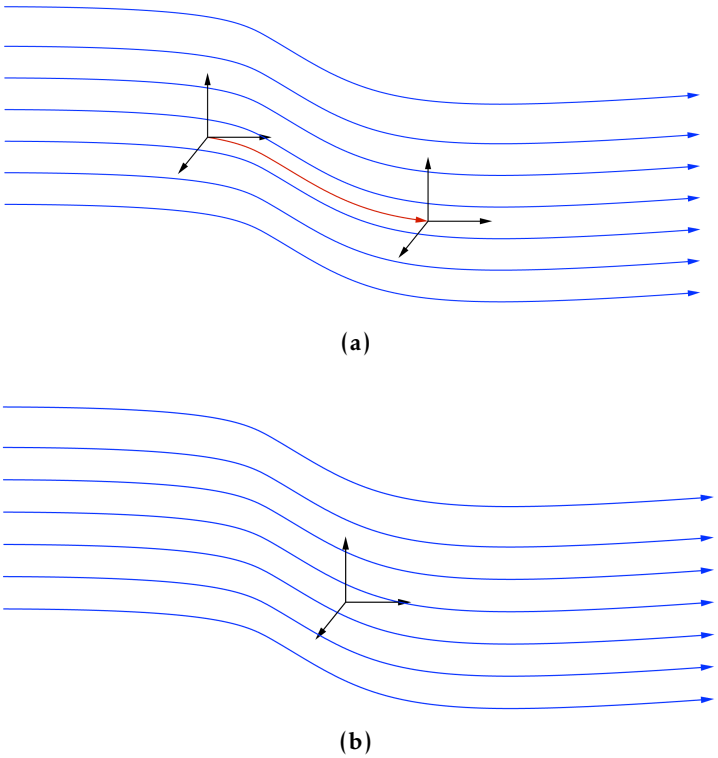


Figure 3.2: *Lagrangian frame of reference (a) Eulerian frame of reference (b)*

3.2 Grid based model limitation

Grid-based methods such as Finite Discrete Method and Finite Element Method have been widely used for Computational Fluid Dynamic and Computational Solid Mechanics and they are still the predominant methods in many simulations. However, despite their popularity, they have several limitations [Li and Liu, 2002; Liu and Liu, 2003].

Eulerian approaches have complex computer codes since the convective term, $v \cdot \nabla f$, is taken in account. Because the grid is fixed in space is difficult to: 1) track mass, energy and momentum flux for moving boundaries, free surfaces and material interfaces 2) generate mesh for irregular domain 3) obtain the time history of fields variables at a fixed point of the material since is only possible to access those variables at the fixed nodes of the grid. Despite this limitation Eulerian grids are often used in presence of large deformations, as in fluid dynamics, because the grid cannot be distorted.

On the other hand, Lagrangian grids are attached on the material. This simplifies: 1) computer codes 2) mesh generation for irregular domain 3) obtaining time history of fields variables for

material points. Moreover, by placing nodes on boundaries and material interfaces, Lagrangian methods automatically track moving boundaries, free surfaces and material interfaces. The main limitation of Lagrangian methods is when it deals with large deformations because it will generate large mesh distortion that affects the accuracy of the solution. This can be solved with mesh rezoning methods, but this dramatically increases the computational cost and may cause loss of material history during the process [Liu and Liu, 2003].

With the aim of overcoming these limitations, researchers have developed several meshfree methods [Oñate et al., 1996; Nayroles et al., 1992; Rapaport, 2004; Duarte and Oden, 1996]. Meshfree methods are more versatile and robust than mesh-based methods thanks to the absence of connectivities between nodes (mesh) [Liu and Liu, 2003]. Most of the meshfree methods use a Lagrangian frame to describe the motion of the nodes. However, unlike Lagrangian grid-based methods, they don't suffer mesh distortion. For this reason they are appealing for simulate fluid flows or solid mechanics in presence of large deformation.
Meshfree methods can be divided in three main groups:

1. Strong form formulation methods

   They are easy to implement, computationally efficient with no integration to evaluate the discrete expression of equations.

2. Weak form formulation methods

   They have excellent stability and accuracy. However, they are not considered "truly" meshfree, as they need a local or global background mesh for integration.

3. Particle based methods

   They employ a finite set of discrete particles to discretise the domain.

This works adopts particle-based methods based, in particular, on Smoothed Particle Hydrodynamics, as discussed in the next Sections. Strong and weak formulation methods are outside the interest of this work; the reader interested in these methods can refer to dedicated reviews [Li and Liu, 2002; Nguyen et al., 2008].

3.3 Meshfree particle model and discrete multi physics

The particle based methods, or simply particle methods, are meshfree methods employing a finite number of discrete particles to discretise a continuum domain (Figure 3.3). The particles

carry computational information depending on the application. For instance, in fluid-dynamic the particles stores a set of field variables such as mass, momentum, energy, position, and other variables (e.g., charge, vorticity) related to the specific problem [Liu and Liu, 2003].

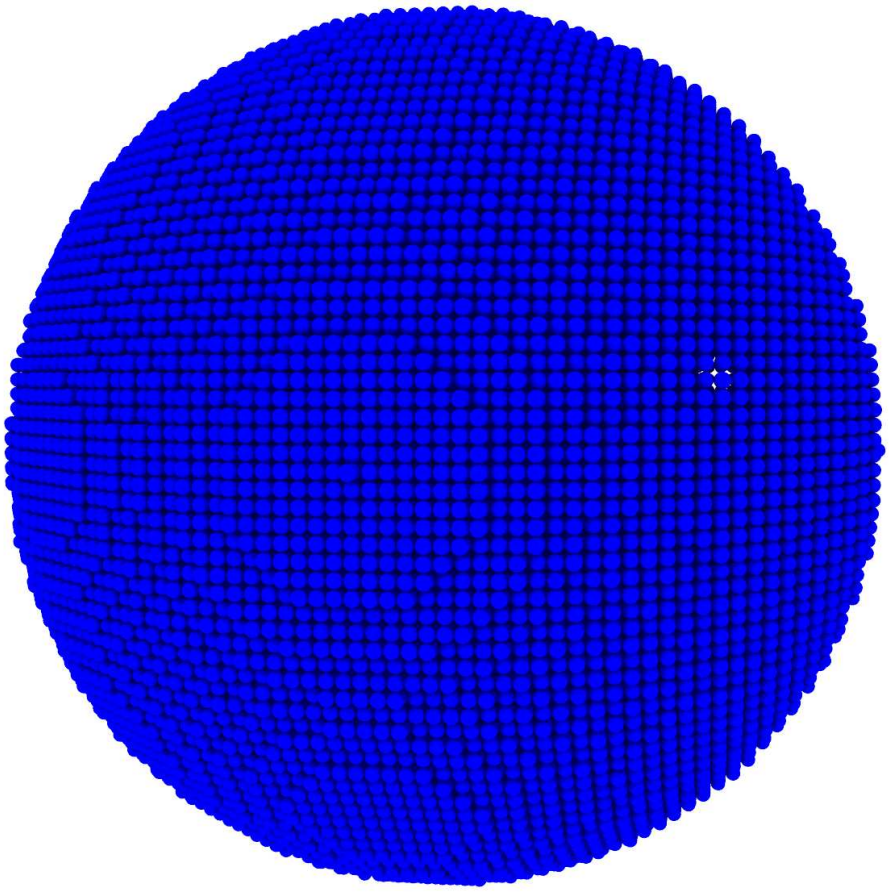


Figure 3.3: *A 3D sphere domain particle discretization.*

Initially meshfree methods such as Smoothed Particle Hydrodynamics, Molecular Dynamics or Discrete Element Method, were used for simulating inherently discrete systems [Gingold and Monaghan, 1977; Rapaport, 2004; Tavarez and Plesha, 2007]. Later they were extended for modelling continuum media [Morris, 1996]. Meshfree methods show advantages over traditional grid-based methods in some specific task such as:

- Simulating phenomena with large deformation;
- Discretise complex geometry;
- Identifying free surfaces, moving interfaces and deformable boundaries because the particles motion is always tracked.

3.3.1 Discrete Multi-Physic

Each particles methods is extremely good in simulating specific phenomena: Smoothed Particle Hydrodynamic (SPH) for hydrodynamic, Discrete Element Method (DEM) for inter-particle contact forces, Peridynamics for solid mechanics and crack propagation, Dissipative Particle Dynamics (DPD) for mesoscale modelling. However, when the system involves several phenomena, one method alone is not always adequate. Therefore, Discrete Multiphysics (DMP) has been developed as a common platform where particles methods can be linked together to address complex phenomena [Alexiadis, 2014, 2015a; Ariane et al., 2018a; Rahmat et al., 2019a; Albano and Alexiadis, 2020; Schütt et al., 2020; Alexiadis, 2019a; Alexiadis et al., 2021; Mohammed et al., 2020].

The methods used in a DMP model share the same particle representation of the computational domain and the same computational paradigm (see Figure 3.4). For each method used in DMP the forces exert by the particles are calculated using their own type of interaction. Then, by solving the Newton's equation of motion, the particles position is updated.

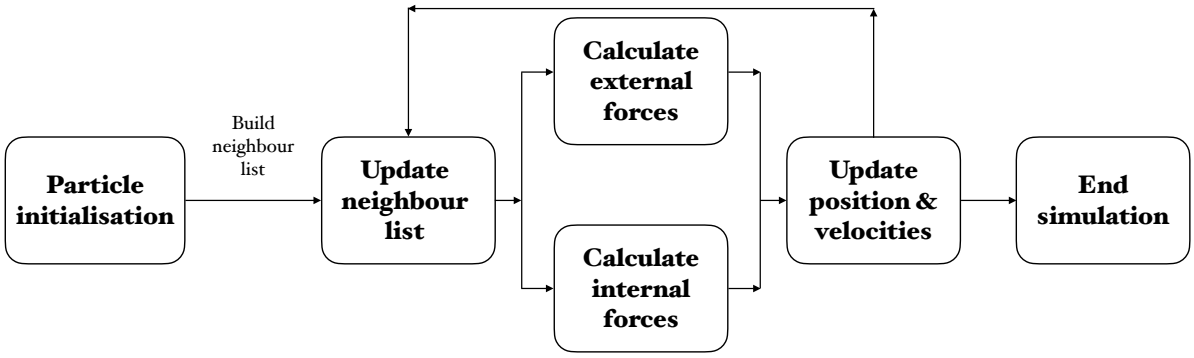


Figure 3.4: *Typical particle method routine for computer coding implementation*

When a continuum, or discrete, domain is represented in a DMP model it is important to define different particle types to correctly assign the right type of interaction. The example in Figure 3.5 shows a DMP simulation of non-spherical particles in a Poiseuille flow. In the domain two types of particle, white to model the liquid phase and black for the solid phase, and four types of interactions are accounted for: Interaction type 1 describes a SPH liquid-liquid interaction, type 2 follows a Coarse-Grained Molecular Dynamics (CGMD) solid-solid interaction within the cubes, type 3 uses a solid-solid interaction between different cubes from DEM and type 4 uses a repulsive Lennard-Jones potential to model the interaction between

white and black particles

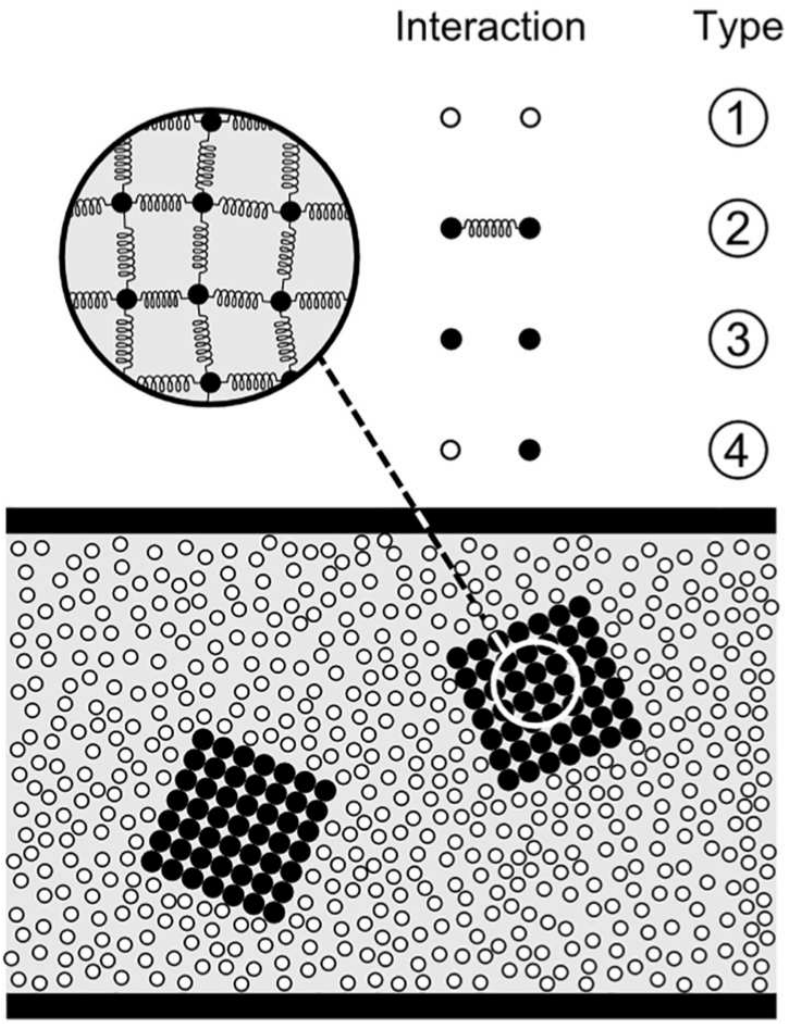


Figure 3.5: *Discrete Multi-Physic model for non-spherical particles in Poiseuille flow*

In the next section we focus on a specific particle method that is going to play a major role in this thesis, Smoothed Particle Hydrodynamics. SPH has been a natural choice for simulating the bubble collapse because of its ability to simulate both highly deformable interfaces and shock waves.

3.4 Smoothed Particle Hydrodynamics

SPH is a Lagrangian meshfree particle method developed by Lucy [Lucy, 1977] and Gingold & Monaghan [Gingold and Monaghan, 1977] for solving astrophysics problems and later extended to address a wide range of applications such as explosion [Liu et al., 2003b], high velocity impact phenomena [Swegle and Attaway, 1995], Riemann problem [Monaghan, 1997], multiphase flow [Shadloo and Yildiz, 2011; Shadloo et al., 2013, 2016; Rahmat and Yildiz, 2018], thermo-fluid application [Ng et al., 2020], non newtonian fluid flows [Shao and Lo, 2003; Hosseini et al., 2007], shock waves [Monaghan and Gingold, 1983; Morris and Monaghan, 1997; Liu et al., 2002; Albano and Alexiadis, 2019], nano-fluid flows [Nasiri et al., 2019], thermo-capillary flows [Hopp-Hirschler et al., 2018].

The SPH method has the following characteristic features [Liu and Liu, 2003]:

1. Meshfree

   The domain is discretised with a set of arbitrarily distributed particles with no connectivity required (see Section 3.3).

2. Integral function representation

   The field functions are approximated with an integral representation (see Section 3.4.1).

3. Compact support

   Thanks to the particle approximation (see Section 3.4.1) the integral representation is replaced by a summation over all the corresponding values of a set of particles in subdomain called support domain.

4. Adaptive

   For each timestep the particle approximation is updated by taking in consideration the current distribution of the particles, updated with a neighbour list (see Section 3.4.1).

5. Lagrangian

   All the PDE's related term are discretised in a set of ODEs only in respect of time (see Section 3.1.1).

6. Dynamic

For all the particles, it is possible to obtain the field variables history by using some explicit integration algorithm to solve the discretised ODEs (see Figure 3.4 and Chapter 4).

3.4.1 Kernel representation

The first step for deriving the SPH method is the integral representation of a function [Liu and Liu, 2003]: Given a continuum function $f(\mathbf{r})$, defined in a volume V, function of the position $\mathbf{r}$, we use the identity

$$f(\mathbf{r}) = \iiint f(\mathbf{r}')\delta(\mathbf{r}-\mathbf{r}')d\mathbf{r}', \tag{3.2}$$

where $\delta(\mathbf{r}-\mathbf{r}')$ is the Dirac delta function defined as

$$\delta(\mathbf{r}-\mathbf{r}') = \begin{cases} +\infty & \mathbf{r} = \mathbf{r}' \\ 0 & \mathbf{r} \neq \mathbf{r}'. \end{cases} \tag{3.3}$$

In SPH, the Dirac delta function is replaced with a bell-shaped function called smoothing function or Kernel, W. As explained in Section 3.4.2, W depends on the position r and on the smoothing length, h. By replacing δ with W is possible to approximate the integral representation, Equation 3.2, into the Kernel approximation:

$$f(\mathbf{r}) \approx \iiint f(\mathbf{r}')W(\mathbf{r}-\mathbf{r}',h)d\mathbf{r}'. \tag{3.4}$$

3.4.2 Smoothing function and smoothing length

The choice of the Kernel is important because it determines the interaction in the integral Kernel approximation (see Equation 3.4), the extension of the support domain but it also ensures consistency and accuracy of the SPH method [Liu and Liu, 2003]. Any Kernel has the following properties

- Normalisation, or Unity, condition

$$\iiint W(\mathbf{r}-\mathbf{r}',h)dr' = 1, \tag{3.5}$$

used to ensure a zero-th order consistency of the integral representation

- Delta function condition

$$\lim_{h\to 0} W(\mathbf{r}-\mathbf{r}',h) = \delta(\mathbf{r}-\mathbf{r}'). \tag{3.6}$$

Delta function and Unity conditions are necessary to obtain the Kernel approximation, see Equation 3.4.

- Compact support condition

$$W(\mathbf{r}-\mathbf{r}',h) = 0 \text{ if } \quad |\mathbf{r}-\mathbf{r}'| > \kappa h. \tag{3.7}$$

The compact support condition introduces the concept of smoothing length, h and the scaling factor, κ. The compact support defines the extension of the support domain of the particle at the position $\mathbf{r}$. This means that the value of W for the particle in position $\mathbf{r}$ depends on the particles of position $\mathbf{r}'$ within the distance κh. As explained later, this condition affects the computational cost and the smoothing effect of W on the information stored in the particles.

- Positivity condition

$$W(\mathbf{r}-\mathbf{r}',h) \geq 0 \text{ if } \quad |\mathbf{r}-\mathbf{r}'| < kh. \tag{3.8}$$

In many applications (i.e. hydrodynamics) this condition avoids unphysical representation of some physical parameter such as negative value of density and energy.

An example of Kernel function is the Lucy Kernel [Lucy, 1977], plotted in Figure 3.6, defined as

$$W(S,h) = \begin{cases} \chi(1+3S)(1-S)^3 & S \leq 1 \\ 0 & S > 1, \end{cases} \tag{3.9}$$

where $S = |\mathbf{r} - \mathbf{r}'|/h$ and χ is the parameter used to satisfy the unity condition. χ is, for one, two and three dimensions, equal to $5/4h$, $5/\pi h^2$ and $105/16\pi h^3$.

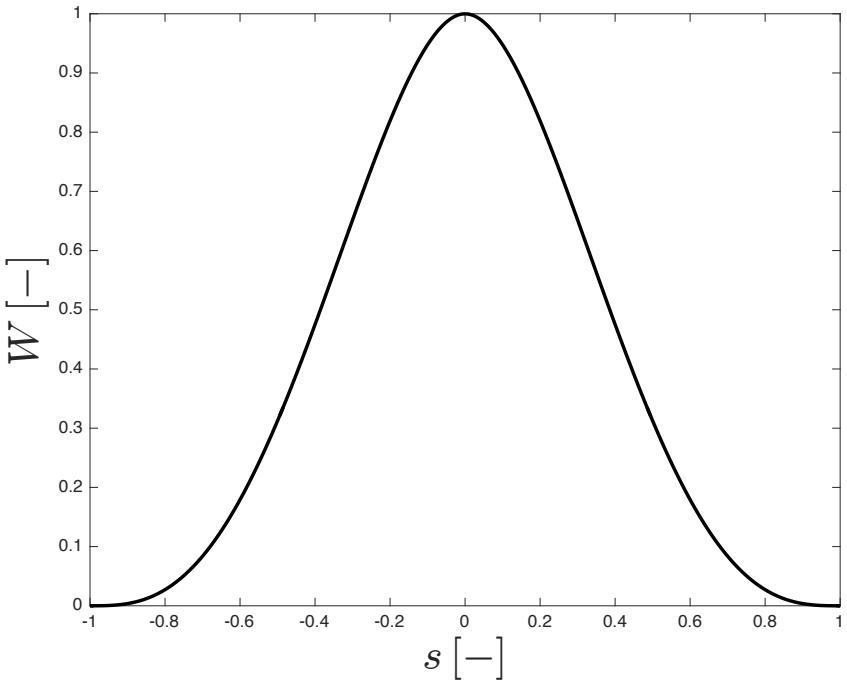


Figure 3.6: *Lucy Kernel function.*

In the Lucy Kernel the scaling factor is $\kappa = 1$ and thus the support domain has the range equal to h.

Smoothing length

As expressed by Equation 3.7 the smoothing length defines the extension of the support domain of a given particle. For this reason, the smoothing length has direct influence on the efficiency and accuracy of the method. When h is too small the support domain may not have enough particles to exert forces on the particle in position $\mathbf{r}$. When h is too large, the computational information stored in the particle may be smoothed out [Liu and Liu, 2003].

Also the computational cost of the simulation depends on h. In fact, functions, integrals and

derivates are approximated using the information stored in the particles within the support domain called neighbouring particles [Rapaport, 2004]. When the method is implemented in a computer code, see Chapter 4, the identities of the neighbouring particles of a given particle are stored in the so-called *neighbour list* whose size depends on the smoothing length. Usually, h is chosen to be multiple of the initial spacing between particles [Chaussonnet et al., 2015].

3.4.3 Particle representation of a function

Equation 3.4 approximates any function using a continuum representation. However, in SPH the domain is represented by a finite number of particles (see Figure 3.3) with their own volume and mass carrying computational information. For this reason, we need to discretise Equation 3.4 with a particle approximation: we consider an infinitesimal volume $d\mathbf{r}'^3$ portion of the domain V composed by computational particles occupying a finite volume $d\mathbf{r}^3$ with their own mass $m = \rho d\mathbf{r}^3$. With this assumption we can write the discretised form of the Kernel approximation

$$f(\mathbf{r_i}) \approx \sum \frac{m_j}{\rho_j} f(\mathbf{r}_j) W(|\mathbf{r}_i - \mathbf{r}_j|, h) = \sum \frac{m_j}{\rho_j} f_j W_{ij}, \tag{3.10}$$

where $\mathbf{r}_i$ is the position of the $i-th$ particle and m_j , ρ_j and $\mathbf{r}_j$ are mass, density and position of the j^{th} particle. Only particles for which $|\mathbf{r_i} - \mathbf{r_j}| < \kappa h$ are taken in account in the summation. To discretise a PDE or EDE we need to use Equation 3.10 and a particle expression for the gradient operator: since f and m are particle properties the gradient operator is only going to operate on W, thus is easy to obtain the particle expression of the gradient of a function

$$\nabla f(\mathbf{r_i}) \approx \nabla \sum \frac{m_j}{\rho_j} f(\mathbf{r}_j) W(|\mathbf{r}_i - \mathbf{r}_j|, h) = \sum \frac{m_j}{\rho_j} f_j \nabla W_{ij}. \tag{3.11}$$

3.4.4 Implementing a SPH discrete equation of motion in a DMP simulator

As said in Section 3.3.1 SPH can be coupled with other particle methods to build a DMP model where the forces exert by the particles are calculate using different type of interaction. In SPH this is done by, for example, discretising in particle form the Lagrangian equation of motion [Bird, 2002]:

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla P = -\frac{P}{\rho^2}\nabla\rho - \nabla\frac{P}{\rho}, \tag{3.12}$$

applying Equation 3.10 and 3.11 and we obtain

$$\frac{d\mathbf{v}}{dt} \approx \frac{d\mathbf{v}_i}{dt} = -\underbrace{\frac{P}{\rho^2}\nabla\rho}_{\frac{P_i}{\rho_i^2}\sum m_j\nabla_j W_{ij}} - \underbrace{\nabla\frac{P}{\rho}}_{\sum m_j\frac{P_j}{\rho_j^2}\nabla_j W_{ij}} \approx -\sum m_j\left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right)\nabla_j W_{ij}. \tag{3.13}$$

Equation 3.13 can be written as a force rather than acceleration by multiplying both side by the mass of the i-th particle

$$\mathbf{f}_i = m_i\frac{d\mathbf{v}_i}{dt} = -\sum m_i m_j\left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right)\nabla_j W_{ij}. \tag{3.14}$$

Time integration scheme

The time integration scheme used in the simulator used in this thesis, LAMMPS, is a modified version of the Velocity-Verlet scheme [Ganzenmüller et al., 2011]. In SPH mass and energy explicitly depend on the velocity, and using the standard Velocity-Verlet scheme [Verlet, 1967]:

1. $\mathbf{v}_i(t+\frac{1}{2}\delta t) = \mathbf{v}_i(t) + \frac{\delta t}{2m_i}\mathbf{f}_i(t)$
    a. $\mathbf{r}_i(t+\delta t) = \mathbf{r}_i(t) + \delta t\mathbf{v}_i(t+\frac{1}{2}\delta t)$
2. computing $\mathbf{f}_i(t+\delta t)$
3. $\mathbf{v}_i(t+\frac{1}{2}\delta t) = \mathbf{v}_i(t+\frac{1}{2}\delta t) + \frac{\delta t}{2m_i}\mathbf{f}_i(t+\delta t)$

introduces a velocities lag behind the positions by $1/2\delta t$ when the forces are computed [Ganzenmüller et al., 2011] and this translates into a poor mass and energy conservation. The lag can be reduced by computing an *extrapolated velocity* before computing forces:

1. $\mathbf{v}_i(t+\frac{1}{2}\delta t) = \mathbf{v}_i(t) + \frac{\delta t}{2m_i}\mathbf{f}_i(t)$

a. $\tilde{\mathbf{v}}_i(t+\delta t) = \mathbf{v}_i(t) + \frac{\delta t}{m_i}\mathbf{f}_i(t)$ **extrapolated velocity**

b. $\rho_i(t+\frac{1}{2}\delta t) = \rho_i(t) + \frac{\delta t}{2}\dot{\rho}_i(t)$

c. $E_i(t+\frac{1}{2}\delta t) = E_i(t) + \frac{\delta t}{2}\dot{E}_i(t)$

d. $\mathbf{r}_i(t+\delta t) = \mathbf{r}_i(t) + \delta t\mathbf{v}_i(t+\frac{1}{2}\delta t)$

2. computing $\mathbf{f}_i(t+\delta t)$, $\dot{\rho}_i(t+\delta t)$, $\dot{E}_i(t+\delta t)$

3. $\rho_i(t+\delta t) = \rho_i(t+\frac{1}{2}\delta t) + \frac{\delta t}{2}\dot{\rho}_i(t+\delta t)$

   a. $E_i(t+\delta t) = E_i(t+\frac{1}{2}\delta t) + \frac{\delta t}{2}\dot{E}_i(t+\delta t)$

   b. $\mathbf{v}_i(t+\frac{1}{2}\delta t) = \mathbf{v}_i(t) + \frac{\delta t}{2m_i}\mathbf{f}_i(t+\delta t)$

The next Chapter shows how to implement governing equations form different particle methods in the simulator used in this thesis, LAMMPS acronym for Large-scale Atomic/Molecular Massively Parallel Simulator.

Chapter 4

How to modify LAMMPS: From the prospective of a Particle method researcher

This Chapter introduces LAMMPS, the software used for the simulations and the results shown in Chapter 5, 6, 7, and 8. It shows LAMMPS structure, how different type of particle interactions can be implemented, and how its source code can be modified.

This Chapter has been published in *ChemEngineering* as:

Albano A, le Guillou E, Danzé A, Moulitsas I, Sahputra IH, Rahmat A, Duque-Daza CA, Shang X, Ching Ng K, Ariane M, Alexiadis A. How to Modify LAMMPS: From the Prospective of a Particle Method Researcher. ChemEngineering. 2021; 5(2):30

My contributions in this work were: Conceptualisation, Methodology, Validation, Writing the original draft, Reviewing and Editing.

I would like to thank all the authors who have contributed to this work.

chemengineering

MDPI

Article

How to Modify LAMMPS: From the Prospective of a Particle Method Researcher

Andrea Albano [1,*], Eve le Guillou [2], Antoine Danzé [2], Irene Moulitsas [2], Iwan H. Sahputra [1,3], Amin Rahmat [1], Carlos Alberto Duque-Daza [1,4], Xiaocheng Shang [5], Khai Ching Ng [6], Mostapha Ariane [7] and Alessio Alexiadis [1,*]

1 School of Chemical Engineering, University of Birmingham, Birmingham B15 2TT, UK; halimits@yahoo.com (I.H.S.); A.Rahmat@bham.ac.uk (A.R.); C.A.Duque-Daza@bham.ac.uk (C.A.D.-D.)
2 Centre for Computational Engineering Sciences, Cranfield University, Bedford MK43 0AL, UK; Eve.M.Le-Guillou@cranfield.ac.uk (E.l.G.); A.Danze@cranfield.ac.uk (A.D.); i.moulitsas@cranfield.ac.uk (I.M.)
3 Industrial Engineering Department, Petra Christian University, Surabaya 60236, Indonesia
4 Department of Mechanical and Mechatronic Engineering, Universidad Nacional de Colombia, Bogotá 111321, Colombia
5 School of Mathematics, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK; X.Shang.1@bham.ac.uk
6 Department of Mechanical, Materials and Manufacturing Engineering, University of Nottingham Malaysia, Jalan Broga, Semenyih 43500, Malaysia; KhaiChing.Ng@nottingham.edu.my
7 Department of Materials and Engineering, Sayens-University of Burgundy, 21000 Dijon, France; Mostapha.Ariane@u-bourgogne.fr
* Correspondence: axa1220@student.bham.ac.uk or aaalbano@gmail.com (A.A.); a.alexiadis@bham.ac.uk (A.A.)

Citation: Albano, A.; le Guillou, E.; Danzé, A.; Moulitsas, I.; Sahputra, I.H.; Rahmat, A.; Duque-Daza, C.A.; Shang, X.; Ching Ng, K.; Ariane, M.; et al. How to Modify LAMMPS: From the Prospective of a Particle Method Researcher. *ChemEngineering* **2021**, *5*, 30. https://doi.org/10.3390/chemengineering5020030

Academic Editors: Mark P. Heitz and Andrew S. Paluch

Received: 11 January 2021
Accepted: 26 May 2021
Published: 13 June 2021

Abstract: LAMMPS is a powerful simulator originally developed for molecular dynamics that, today, also accounts for other particle-based algorithms such as DEM, SPH, or Peridynamics. The versatility of this software is further enhanced by the fact that it is open-source and modifiable by users. This property suits particularly well Discrete Multiphysics and hybrid models that combine multiple particle methods in the same simulation. Modifying LAMMPS can be challenging for researchers with little coding experience. The available material explaining how to modify LAMMPS is either too basic or too advanced for the average researcher. In this work, we provide several examples, with increasing level of complexity, suitable for researchers and practitioners in physics and engineering, who are familiar with coding without been experts. For each feature, step by step instructions for implementing them in LAMMPS are shown to allow researchers to easily follow the procedure and compile a new version of the code. The aim is to fill a gap in the literature with particular reference to the scientific community that uses particle methods for (discrete) multiphysics.

Keywords: LAMMPS; particle method; discrete multiphysics

1. Introduction

LAMMPS, acronym for Large-scale Atomic/Molecular Massively Parallel Simulator, was originally written in F77 by Steve Plimpton [1] in 1993 with the goal of having a large-scale parallel classical Molecular Dynamic (MD) code. The project was a Cooperative Research and Development Agreement (CRADA) between two DOE labs (Sandia and LLNL) and three companies (Cray, Bristol Myers Squibb, and Dupont). Since the initial release LAMMPS has been improved and expanded by many researchers who implemented many mesh-free computational methods such as Perydynamics, Smoothed particle hydrodynamics (SPH), Discrete Elemet Method (DEM) and many more [2–11].

Such a large number of computational methods within the same simulator allows researchers to easily combine them for the simulation of complex phenomena. In particular, our research group has used during the years LAMMPS in a variety of settings that go from

classic Molecular Dynamics [12–16], to Discrete Multiphysics simulations of cardiovascular flows [17–20], Modelling drug adsorption in human organs [21–24], Cavitation [25–27], multiphase flow containing cells or capsules [28–31], solidification/dissolution [32–34], material properties [35,36] and even epidemiology [37] and coupling particles methods with Artificial Intelligence [38–40]. An example of a Discrete Multiphysics simulation run with the basic LAMMPS's code is shown in Appendix A.

Thanks to its modular design open source nature and its large community, LAMMPS has been conceived to be modified and expanded by adding new features. In fact, about 95% of its source code is add-on file [41]. However, this can be a tough challenge for researcher with no to little knowledge of coding. The LAMMPS user manual [41] describes the internal structure and algorithms of the code with the intent of helping researcher to expand LAMMPS. However, due to the lack of examples of implementation and validation, the document can be hard to read for user who are not programmers. In fact, the available material is either very basic [41] or requires advanced programming skills [42,43].

The aim of this work is to provide several step-by-step examples with increasing level of complexity that can fill the gap in the middle to help and encourage researchers to use LAMMPS for discrete multiphysics and expand it with new adds on to the code that could fit their needs. In fact, most of the available material focuses on Molecular Dynamics (MD) and implicitly assumes that the reader's background is in MD rather than other particle methods such as SPH or DEM. On the contrary, this paper is dedicated to the particle community and highlights how LAMMPS can be used and modified for methods other than MD. This goal fits particularly well with the scope of this Special Issue on "Discrete Multiphysics: Modelling Complex Systems with Particle Methods" In particular, it relates to some of the topics of the Special Issue such by exploring the potential of LAMMPS for coupling particle methods, and by sharing some "tricks of the trade" on how to modify its code that cannot be found anywhere else in the literature.

In Section 2 LAMMPS structure and hierarchy are explained introducing the concept of style. Following the LAMMPS authors advice, to avoid writing a new style from scratch, Sections 3–6 new styles are developed using existing style are as reference. Finally, in Section 7, all the steps to write a class from scratch are shown.

2. LAMMPS Structure

After initial releases in F77 and F90, LAMMPS is now written in C++, an object oriented language that allows any programmer to exploit the class programming paradigm. The declaration of a class, including the signature of the instance variables and functions (or methods), which can be accessed and used by creating an instance of that class. The data and functions within a class are called members of the class. The definition (or implementation) of a member function can be given inside or outside the class definition.

A class has private, public, and protected sections which contain the corresponding class members.

- The private members, defined before the keyword public, cannot be accessed from outside the class. They can only be accessed by class or "friend" functions, which are declared as having access to class members, without themselves being members. All the class members are private by default.
- The public members can be accessed from outside the class anywhere within the scope of the class object.
- The protected members are similar to private members but they can be accessed by derived classes or child classes while private members cannot.

2.1. Inheritance

An important concepts in object-oriented programming is that of inheritance. Inheritance allows to define a class in terms of another class and the new class inherits the members of the existing class. This existing class is called the base (or parent) class, and the new class is referred to as a subclass, or child class, or derived class.

The idea of inheritance implements the "is a" relationship. For example, Mammal IS-A Animal, Dog IS-A Mammal hence Dog IS-A Animal as well.

The inheritance relationship between the parent and the derived classes is declared in the derived class with the following syntax:

Listing 1: C++ syntax for classes inheritance

```
class name_child_class: access_specifier name_parent_class
{ /*...*/ };
```

The type of inheritance is specified by the access-specifier, one of public, protected, or private. If the access-specifier is not used, then it is private by default, but public inheritance is commonly used: public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.

2.2. Virtual Function

The signature of a function *f* must be declared with a virtual keyword in a base class C to allow its definition (implementation), or redefinition, in a derived class *D*. Then, when a derived class *D* object is used as an element of the base class *C*, and *f* is called, the derived class's implementation of the function is executed.

There is nothing wrong with putting the virtual in front of functions inside of the derived classes, but it is not required, unless it is known for sure that the class will not have any children who would need to override the functions of the base class. A class that declares or inherits a virtual function is called a polymorphic class.

2.3. LAMMPS Inheritance and Class Syntax

A schematic representation of the LAMMPS inheritance tree is shown in Figure 1: LAMMPS is the top-level class for the entire code, then all the core classes, highlighted in blue, inherit all the constructors, destructors, assignment operator members, friends and private members declared and defined in LAMMPS. The core classes perform LAMMPS fundamental actions. For instance, the Atom class collects and stores all the per-atom, or per-particle, data while Neighbor class builds the neighbor lists [41].

The style classes, highlighted in reds, inherit all the constructors, destructors, assignment operator members, friends and private members declared and defined in LAMMPS and in the corresponding core class. The style classes are also virtual parents class of many child classes that implement the interface defined by the parent class. For example, the fix style has around 100 child classes.

Each style is composed of a pair of files:

- namestyle.h
  The header of the style, where the class style is defined and all the objects, methods and constructors are declared.
- namestyle.cpp
  Where all the objects, methods and constructors declared in the class of style are defined.

When a new style is written both `namestyle.h` and `namestyle.cpp` files need to be created.

Each "family" style has its own set of methods, declared in the header and defined in the cpp file, in order to define the scope of the style. For example, the pair style are classes that set the formula(s) LAMMPS uses to compute pairwise interactions while bond style set the formula(s) to compute bond interactions between pairs of atoms [41].

Each pair style has some recurrent functions such as `compute`, `allocate` and `coeff`. Although the final scope of those functions can differ for different styles, they all share a similar role within the classes.

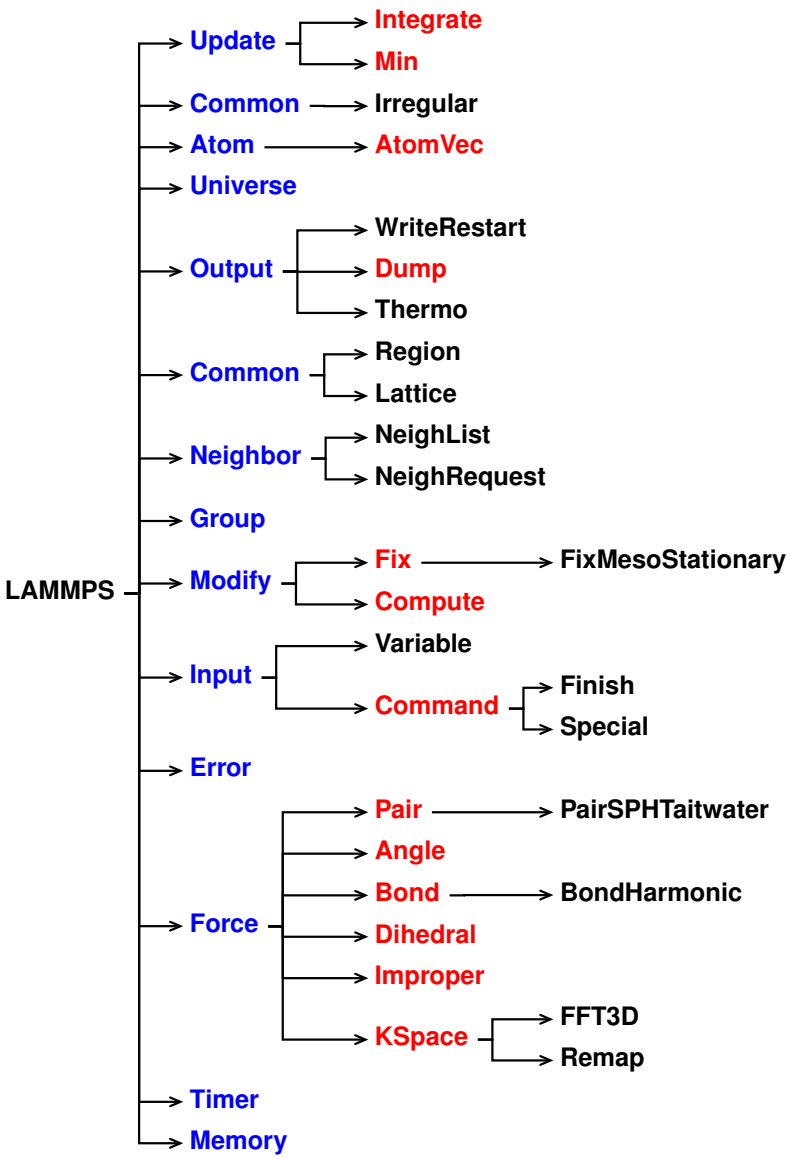


Figure 1. Class hierarchy within LAMMPS source code.

An example of a pair style, sph/taitwater, header in LAMMPS is shown in Listing 2.

Listing 2: Header file of sph/taitwater pair style (pair_sph_taitwater.h)

```
class PairSPHTaitwater: public Pair{// class definition, accessibility and Inheritance
 public:    // access specifier: public
  // public methods
  PairSPHTaitwater(class LAMMPS *); // Constructors
  virtual ~PairSPHTaitwater(); // Destructors
  virtual void compute(int, int);
  void settings(int, char **);
  void coeff(int, char **);
  virtual double init_one(int, int);
  virtual double single(int, int, int, int, double, double, double, double &);

 protected:  // access specifier: protected
  double *rho0, *soundspeed, *B;
  double **cut,**viscosity;
  int first;
// protected methods
  void allocate();
};
```

All the class members are defined in the cpp file. Taking sph/taitwater pair style as reference, each method declared in Listing 2 will be defined and commented in the next sections. Although this can be style-specific, the aim is to give an overview of how the methods are defined in the cpp in LAMMPS. Albeit different style has different methods, the understanding gained can be transferred into others style, as shown in Sections 3 and 6.

2.3.1. Constructor

Any class usually include a member function called constructors. The constructor is mechanically invoked when an object of the class is created. This allows the class to initialise members or allocate storage. Unlike the other member of the class, the constructor name must match the name of the class and it does not have a return type.

Listing 3: Constructor definition in sph/taitwater pair style (pair_sph_taitwater.cpp)

```
PairSPHTaitwater::PairSPHTaitwater(LAMMPS *lmp) : Pair(lmp)
{
  restartinfo = 0;
  first = 1;
}
```

2.3.2. Destructor

The role of destructors is to de-allocate the allocated dynamic memory, see Section 2.3.8, being mechanically invoked just before the end of the class lifetime. Similarly to constructors, destructors does not have a return type and have the same name as the class name with a tilde (∼) prefix.

Listing 4: Destructors definition in sph/taitwater pair style (pair_sph_taitwater.cpp)

```
PairSPHTaitwater::~PairSPHTaitwater() {
  if (allocated) { /// check if the pair style uses allocate, see Section 2.8
  ///  cleanup the memory used by allocate, see Section 2.8
    memory->destroy(setflag);
    memory->destroy(cutsq);
    memory->destroy(cut);
    memory->destroy(rho0);
    memory->destroy(soundspeed);
    memory->destroy(B);
    memory->destroy(viscosity);
  }
}
```

2.3.3. `compute`

`compute` is virtual member of the pair style and is one of the most relevant functions in a number of classes in LAMMPS. For instance, in pair style classes is used to compute pairwise interaction of the specific pair style. This can seen in the commented Listing 5, where the force applied on a pair of neighboring particles is derived using the Tait equation, lines 131–151. In `compute` all the local parameters needed to compute the pairwise interaction are declared and defined within the method.

Listing 5: `compute` definition in sph/taitwater pair style (pair_sph_taitwater.cpp)

```
void PairSPHTaitwater::compute(int eflag, int vflag) {

/// start variables and pointer declaration
  int i, j, ii, jj, inum, jnum, itype, jtype;
  double xtmp, ytmp, ztmp, delx, dely, delz, fpair;

  int *ilist, *jlist, *numneigh, **firstneigh;
  double vxtmp, vytmp, vztmp, imass, jmass, fi, fj, fvisc, h, ih, ihsq;
  double rsq, tmp, wfd, delVdotDelR, mu, deltaE;
// end

  if (eflag || vflag)
```

```
    ev_setup(eflag, vflag);
  else
    evflag = vflag_fdotr = 0;

/// others variables and pointers declaration and initialisation
  double **v = atom->vest; // pass the value of the pointer that points to a pointers
  // pointing to the first element of velocity vector of the particles
  double **x = atom->x; // pass the value of the pointer that points to a pointers
  // pointing to the first element of position vector of the particles
  double **f = atom->f; // pass the value of the pointer that points to a pointers
  // pointing to the first element of force vector of the particles
  double *rho = atom->rho; // pass the value of the pointer that points
  // to the density vector of the particles
  double *mass = atom->mass; // pass the value of the pointer that points
   // to the mass vector of the particles
  double *de = atom->de; // pass the value of the pointer that points
  // to the change of internal energy of the particles
  double *drho = atom->drho; // pass the value of the pointer that points
  // to the change of density of the particles
  int *type = atom->type; // pass the value of the pointer that points to the type of the
     particles
  int nlocal = atom->nlocal; // pass the value of the numbers of owned and ghost atoms on
     this proc
  int newton_pair = force->newton_pair;  // pass the value of the Newton's 3rd law
     settings
/// end


  // check consistency of pair coefficients

  if (first) {
    for (i = 1; i <= atom->ntypes; i++) {
      for (j = 1; i <= atom->ntypes; j++) {
        if (cutsq[i][j] > 1.e-32) {
          if (!setflag[i][i] || !setflag[j][j]) {
            if (comm->me == 0) {
              printf(
                  "SPH particle types %d and %d interact with cutoff=%g,
                  but not all of their single particle properties are set.\n",
                  i, j, sqrt(cutsq[i][j]));
            } } } } }
    first = 0;
  }


  inum = list->inum; // pass the value of number of I atoms neighbors are stored for
  ilist = list->ilist; // pass the value of the pointer pointing to the local indices of I
     atoms
  numneigh = list->numneigh; // pass the address of a pointer pointing to the number of J
     neighbors
  //  for each I atom
  firstneigh = list->firstneigh; // pass the value of a pointer that points to pointer
  // pointing to 1st J int value of each I atom




  for (ii = 0; ii < inum; ii++) { // loop for each i particles stored in inum
    i = ilist[ii]; // pass the index of the i particle
    xtmp = x[i][0]; // pass the x position of the i particle
    ytmp = x[i][1]; // pass the y position of the i particle
    ztmp = x[i][2]; // pass the z position of the i particle
    vxtmp = v[i][0]; // pass the x velocity of the i particle
    vytmp = v[i][1]; // pass the y velocity of the i particle
    vztmp = v[i][2]; // pass the z velocity of the i particle
    itype = type[i]; // pass the type of the i particle
    jlist = firstneigh[i]; // pass the 1st J int value of each I atom
    jnum = numneigh[i]; //pass number of J neighbors for each I atom

    imass = mass[itype]; // pass the mass of the i particle

```

```
    // compute force of atom i with Tait EOS
    tmp = rho[i] / rho0[itype];
    fi = tmp * tmp * tmp;
    fi = B[itype] * (fi * fi * tmp - 1.0) / (rho[i] * rho[i]);
    // end

    for (jj = 0; jj < jnum; jj++) {   // loop over neighbours list of particle i
      j = jlist[jj]; // pass the index of the j particle
      j &= NEIGHMASK;

      delx = xtmp - x[j][0]; // x distance between particles i and j
      dely = ytmp - x[j][1]; // y distance between particles i and j
      delz = ztmp - x[j][2]; // z distance between particles i and j
      rsq = delx * delx + dely * dely + delz * delz; // squared distance between particles i and j
      jtype = type[j]; // pass the type of the i particle
      jmass = mass[jtype]; // pass the mass of the j particle

      if (rsq < cutsq[itype][jtype]) { // check if i and j are neighbor

        h = cut[itype][jtype]; // pass the smoothing length
        ih = 1.0 / h; // calculate the inverse, divisions are computationally expensive
        ihsq = ih * ih; // squared inverse

        wfd = h - sqrt(rsq);

        if (domain->dimension == 3) {
          // Lucy Kernel, 3d
          wfd = -25.066903536973515383e0 * wfd * wfd * ihsq * ihsq * ihsq * ih;
        } else {
          // Lucy Kernel, 2d
          wfd = -19.098593171027440292e0 * wfd * wfd * ihsq * ihsq * ihsq;
        }

        // compute force of atom j with Tait EOS
        tmp = rho[j] / rho0[jtype];
        fj = tmp * tmp * tmp;
        fj = B[jtype] * (fj * fj * tmp - 1.0) / (rho[j] * rho[j]);
        // end

        // dot product of velocity delta and distance vector
        delVdotDelR = delx * (vxtmp - v[j][0]) + dely * (vytmp - v[j][1])
            + delz * (vztmp - v[j][2]);

        // artificial viscosity (Monaghan 1992)
        if (delVdotDelR < 0.) {
          mu = h * delVdotDelR / (rsq + 0.01 * h * h);
          fvisc = -viscosity[itype][jtype] * (soundspeed[itype]
              + soundspeed[jtype]) * mu / (rho[i] + rho[j]);
        } else {
          fvisc = 0.;
        }

        fpair = -imass * jmass * (fi + fj + fvisc) * wfd; // total pair force
        deltaE = -0.5 * fpair * delVdotDelR; // internal energy increment

    // change in force in each direction for particle i
        f[i][0] += delx * fpair;
        f[i][1] += dely * fpair;
        f[i][2] += delz * fpair;

        //change in density for particle i
        drho[i] += jmass * delVdotDelR * wfd;

        // change in internal energy for particle i
        de[i] += deltaE;

        if (newton_pair || j < nlocal) {
        // change in force in each direction for particle j
```

```
147           f[j][0] -= delx * fpair;
148           f[j][1] -= dely * fpair;
149           f[j][2] -= delz * fpair;
150
151           de[j] += deltaE; // change in internal energy for particle j
152
153           drho[j] += imass * delVdotDelR * wfd; // change in density for particle j
154         }
155
156         if (evflag)
157           ev_tally(i, j, nlocal, newton_pair, 0.0, 0.0, fpair, delx, dely, delz);
158       }
159     }
160   }
161
162   if (vflag_fdotr) virial_fdotr_compute();
163 }
```

2.3.4. `settings`

`settings` is a public void function that reads the input script checking that all the arguments of the pair style are declared. If arguments are present, `settings` stores them so they can be used by `compute`. Examples for no arguments pair style and arguments pair style input script with the corresponding `settings` are listed below:

- No arguments pair style: sph/taitwater
  As described in the SPH for LAMMPS manual [6], the command line to invoke the sph/taitwater pair style is shown in Listing 6.

  Listing 6: Command line to invoke sph/taitwater pair style

  ```
  1 pair_style sph/taitwater
  ```

  In this pair style there is just a string defining the pair style, `sph/taitwater`, with no arguments. For this reason in `settings`, Listing 7, when the `if` statement is true (number of arguments other than zero) an error is produced.

  Listing 7: `setting` definition in sph/taitwater pair style (pair_sph_taitwater.cpp)

  ```
  1 void PairSPHTaitwater::settings(int narg, char **arg) {
  2   if (narg != 0)  ///  check the number of arguments
  3     error->all(FLERR,"Illegal number of setting arguments for pair_style sph/
        taitwater");
  4 }
  ```

- Arguments pair syle: sph/rhosum
  As described in the SPH for LAMMPS manual [6], the command line to invoke the sph/rhosum pair style is shown in Listing 8.

  Listing 8: Command lines to invoke sph/rhosum pair style

  ```
  1 pair_style sph/rhosum Nstep
  ```

  In this pair style there is a string defining the pair style, `sph/rhosum`, plus one argument, `Nstep`. For this reason in `settings`, Listing 9, when the `if` statement is true (number of arguments other than one) an error is produced. When the `if` statement is false `settings` assigns the value of `Nstep` in the variable `nstep`, line 5, by using the `inumeric` function defined in the force class.

  Listing 9: `setting` definition in sph/rhosum pair style (pair_sph_rhosum.cpp)

  ```
  1 void PairSPHRhoSum::settings(int narg, char **arg) {
  2   if (narg != 1) ///  check the number of arguments
  3     error->all(FLERR,
  4         "Illegal number of setting arguments for pair_style sph/rhosum");
  5   nstep = force->inumeric(FLERR,arg[0]); // store the variable in the position 0 (Nstep)
        into nstep
  6 }
  ```

2.3.5. `coeff`

Similar to `setting`, `coeff` is a public void function that reads and set the coefficients used in by `compute` of the pair style. For each *i j* pair is possible to set different coefficients. The coefficients are passed in the input file with the command line pair coeff, see Listing 10. As before, examples for different pair coeff input script and the corresponding `coeff` are listed below:

- **sph/taitwater**
  As described in the SPH for LAMMPS manual [6], the command line to invoke sph/taitwater pair coeff is shown in Listing 10.

  Listing 10: Command line to invoke sph/taitwater pair coeff

```
pair_coeff I J rho_0 c_0 alpha h
```

  In total there are six arguments. Thus, in `coeff`, Listing 11, when `if` statement is true (number of arguments other than six) an error is produced. When the `if` statement is false `coeff` assigns the type of particles I and J plus the value of `rho_0`, `c_0`, `alpha` and `h` in from the string to the variables by using the `numeric` function defined in force class. At last, within the double `for` loop from line 19 to 32, the variables are assigned for each particles.

  Listing 11: `coeff` definition in sph/taitwater pair style (pair_sph_taitwater.cpp)

```
void PairSPHTaitwater::coeff(int narg, char **arg) {
  if (narg != 6) ///  check the number of arguments
    error->all(FLERR,
        "Incorrect args for pair_style sph/taitwater coefficients");
  if (!allocated) ///  check if allocate has been called
    allocate(); ///  call allocate, see section 2.8

  int ilo, ihi, jlo, jhi;
  force->bounds(arg[0], atom->ntypes, ilo, ihi);
  force->bounds(arg[1], atom->ntypes, jlo, jhi);

 ///  store the variables in the position 2--5
  double rho0_one = force->numeric(FLERR,arg[2]);
  double soundspeed_one = force->numeric(FLERR,arg[3]);
  double viscosity_one = force->numeric(FLERR,arg[4]);
  double cut_one = force->numeric(FLERR,arg[5]);
 /// B_one is a constant used in tait EOS inside compute, see section 2.3
  double B_one = soundspeed_one * soundspeed_one * rho0_one / 7.0;

///  assign the coefficient to the corresponding particle (i)
///  and to the pair of particles (i,j)
  int count = 0;
  for (int i = ilo; i <= ihi; i++) {
    rho0[i] = rho0_one;
    soundspeed[i] = soundspeed_one;
    B[i] = B_one;
    for (int j = MAX(jlo,i); j <= jhi; j++) {
      viscosity[i][j] = viscosity_one;
      cut[i][j] = cut_one;

      setflag[i][j] = 1;

      count++;
    }
  }
  if (count == 0) ///  check if the arguments have been assigned
    error->all(FLERR,"Incorrect args for pair coefficients");
}
```

- **sph/rhosum**
  As described in the SPH for LAMMPS manual [6], the syntax to invoke the command is shown in Listing 12.

Listing 12: Command lines to invoke sph/rhosum pair style

```
pair_coeff I J h
```

In this case there are three arguments. Thus, in the `coeff`, Listing 13, when the `if` statement is true (number of arguments other than six) an error is produced. When the error is not produced function assigns the type of particles I and J plus the value of *h* in the string to the variable `cut_one`, line 11, by using `bounds` and `numeric` function defined in force class. At last, within the double `for` loop from line 14 to 20, the variables are assigned for each particles.

Listing 13: `coeff` definition in sph/rhosum pair style (pair_sph_rhosum.cpp)

```
1  void PairSPHRhoSum::coeff(int narg, char **arg) {
2    if (narg != 3) ///  check the number of arguments
3      error->all(FLERR,"Incorrect number of args for sph/rhosum coefficients");
4    if (!allocated) /// check if allocate has been called
5      allocate();  /// call allocate, see section 2.8
6
7    int ilo, ihi, jlo, jhi;
8    force->bounds(arg[0], atom->ntypes, ilo, ihi);
9    force->bounds(arg[1], atom->ntypes, jlo, jhi);
10
11   double cut_one = force->numeric(FLERR,arg[2]);
12
13 ///  assign the coefficient to the pair of particles (i,j)
14   int count = 0;
15   for (int i = ilo; i <= ihi; i++) {
16     for (int j = MAX(jlo,i); j <= jhi; j++) {
17       cut[i][j] = cut_one;
18       setflag[i][j] = 1;
19       count++;
20     }
21   }
22
23   if (count == 0) ///  check if the arguments have been assigned
24     error->all(FLERR,"Incorrect args for pair coefficients");
25 }
```

2.3.6. `init_one`

`init_one` check if all the pair coefficients for a given *i j* pair have been assigned. If they were assigned the methods ensure the symmetry of the matrix.

Listing 14: `init_one` definition in sph/taitwater pair style (pair_sph_taitwater.cpp)

```
double PairSPHTaitwater::init_one(int i, int j) {
/// check if the coefficient of the pair of particles (i,j) were assigned
  if (setflag[i][j] == 0) {
    error->all(FLERR,"Not all pair sph/taitwater coeffs are set");
  }
/// ensure the matrix symmetry
  cut[j][i] = cut[i][j];
  viscosity[j][i] = viscosity[i][j];

  return cut[i][j];
}
```

2.3.7. `single`

In `single` the force and energy of a single pairwise interaction, or single bond or angle (in case of bond or angle style), between two atoms is evalutated. The method is specifically invoked by the command line compute pair/local (or compute bond/local) to calculate properties of individual pair, or bond, interactions [41].

Listing 15: `single` definition in sph/taitwater pair style (pair_sph_taitwater.cpp)

```
double PairSPHTaitwater::single(int i, int j, int itype, int jtype,
    double rsq, double factor_coul, double factor_lj, double &fforce) {
  fforce = 0.0;

  return 0.0;
}
```

2.3.8. `allocate`

`allocate` is a protected void function that allocates dynamic memory. The dynamic memory allocation is used when the amount of memory needed depends on user input. As explained before, at the end of the lifetime of the class, the destructors will de-allocate the memory the memory used by `allocate`.

Listing 16: `allocate` definition in sph/taitwater pair style (pair_sph_taitwater.cpp)

```
void PairSPHTaitwater::allocate() {
  allocated = 1;  ///  confirm that allocated has been called
  int n = atom->ntypes;  /// assigm the value of the number of types

  memory->create(setflag, n + 1, n + 1, "pair:setflag");
  for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++)
      setflag[i][j] = 0;

/// allocate the memory for the arguments of the pair style
  memory->create(cutsq, n + 1, n + 1, "pair:cutsq");
  memory->create(rho0, n + 1, "pair:rho0");
  memory->create(soundspeed, n + 1, "pair:soundspeed");
  memory->create(B, n + 1, "pair:B");
  memory->create(cut, n + 1, n + 1, "pair:cut");
  memory->create(viscosity, n + 1, n + 1, "pair:viscosity");
}
```

3. Kelvin–Voigt Bond Style

We can use what we learned in the previous section to generate a new dissipative bond potential that can be used to model viscoelastic materials. The Kelvin–Voigt model [44] is used to model viscoelastic material as a purely viscous damper and purely elastic spring connected in parallel as shown in Figure 2.

Since the two components of the model are arranged in parallel, the strain in each component is identical:

$$\varepsilon_{tot} = \varepsilon_{spring} = \varepsilon_{damper}. \tag{1}$$

On the other hand, the total stress σ_{tot} will be split into σ_{spring} and σ_{damper} to have $\varepsilon_{spring} = \varepsilon_{damper}$. Thus we have

$$\sigma_{tot} = \sigma_{spring} + \sigma_{damper}. \tag{2}$$

Combining Equations (1) and (2) with the constitutive relation for both the spring and the dumper, $\sigma_{spring} = k\varepsilon$ and $\sigma_{damper} = b\dot{\varepsilon}$, is possible to write that

$$\sigma = k\varepsilon(t) + b\frac{d\varepsilon(t)}{dt} = k\varepsilon(t) + b\dot{\varepsilon}, \tag{3}$$

where k is the elastic modulus and b is the coefficient of viscosity. Equation (3) relates stress to strain and strain rate for a Kelvin–Voigt material [44].

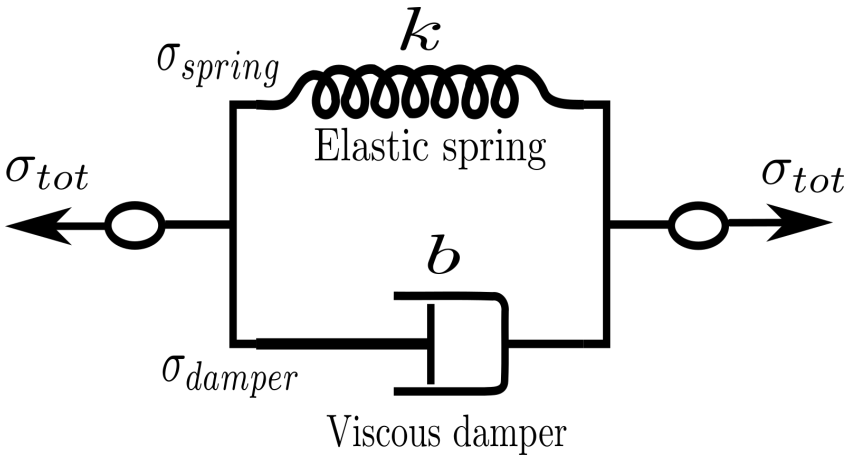


Figure 2. Schematic representation of Kelvin–Voigt model [44].

Similarly to bond test to write a new pair style called bond kv we take the bond harmonic pair style as reference. The new pair style is declared and initialised in bond_kv.h and bond_kv.cpp saved in the /src/MOLECULE directory and its hierarchy is shown in Figure 3.

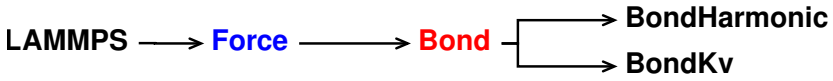


Figure 3. Class hierarchy of the new bond style.

3.1. Validation

The bond kv pair style has been validated by Sahputra et al. [45] in their Discrete Multiphysics model for encapsulate particles with a soft outer shell.

3.2. bond_kv.cpp

All the functions will be the same as in the reference bond harmonic. However, in our new bond kv, we need to substitute the "BondHarmonic" text by a new "BondKv" text, as can be seen in Listings 17 and 18. Form now on, when we show a side-by-side comparison between the reference and the modified file, we highlight in yellow the modified lines and in red the deleted lines.

Listing 17: Original script (bond_harmonic.cpp)

```
#include "math.h"
#include "stdlib.h"
#include "bond_harmonic.h"
#include "atom.h"
#include "neighbor.h"
#include "domain.h"
#include "comm.h"
#include "force.h"
#include "memory.h"
#include "error.h"

using namespace LAMMPS_NS;

BondHarmonic::BondHarmonic(LAMMPS *lmp) : Bond(lmp)
 {}
BondHarmonic::~BondHarmonic()
{ ... }
void BondHarmonic::compute(int eflag, int vflag)
{ ... }
void BondHarmonic::allocate()
{ ... }
void BondHarmonic::coeff(int narg, char **arg)
```

```
{ ... }
double BondHarmonic::equilibrium_distance(int i)
{ ... }
void BondHarmonic::write_restart(FILE *fp)
{ ... }
void BondHarmonic::read_restart(FILE *fp)
{ ... }
void BondHarmonic::write_data(FILE *fp)
{ ... }
double BondHarmonic::single(int type, double rsq,
 int i, int j, double &fforce)
{ ... }
```

Listing 18: Modified script (bond_kv.cpp)

```
#include "math.h"
#include "stdlib.h"
#include "bond_kv.h"
#include "atom.h"
#include "neighbor.h"
#include "domain.h"
#include "comm.h"
#include "force.h"
#include "memory.h"
#include "error.h"

using namespace LAMMPS_NS;

BondKv::BondKv(LAMMPS *lmp) : Bond(lmp)
 {}
BondKv::~BondKv()
{ ... }
void BondKv::compute(int eflag, int vflag)
{ ... }
void BondKv::allocate()
{ ... }
void BondKv::coeff(int narg, char **arg)
{ ... }
double BondKv::equilibrium_distance(int i)
{ ... }
void BondKv::write_restart(FILE *fp)
{ ... }
void BondKv::read_restart(FILE *fp)
{ ... }
void BondKv::write_data(FILE *fp)
{ ... }
double BondKv::single(int type, double rsq,
 int i, int j, double &fforce)
{ ... }
```

Compared to the bond harmonic we are introducing a new parameter, b, from the input file. For this reason we need to modify destructor, `compute`, `allocate`, `coeff`, `write_restart` and `read_restart`. Following the order of function initialisation, see Listing 18, the destructor is modified as shown in Listing 20.

Listing 19: Original destructor (bond_harmonic.cpp)

```
BondHarmonic::~BondHarmonic()
{
  if (allocated) {
    memory->destroy(setflag);
    memory->destroy(k);
    memory->destroy(r0);
  }
}
```

Listing 20: Modified destructor (bond_kv.cpp)

```
BondKv::~ BondKv()
{
  if (allocated) {
    memory->destroy(setflag);
    memory->destroy(k);
    memory->destroy(r0);
    memory->destroy(b);  /* dashpot/damper costant */
  }
}
```

The next function to modify is `compute`. The strain rate, $\dot{\epsilon}$, can also be seen as the speed of deformation. To use it within the new pair style we need to declared and initialised the velocities of each particles, see Listing 22.

Listing 21: Original `compute` (bond_harmonic.cpp)

```
void BondTest::compute(int eflag, int vflag)
{
  int i1,i2,n,type;
  double delx,dely,delz,ebond,fbond;
  double rsq,r,dr,rk;

  ebond = 0.0;
  if (eflag || vflag) ev_setup(eflag,vflag);
  else evflag = 0;

  double **x = atom->x;
  double **f = atom->f;
  }
}
```

Listing 22: Modified `compute` (bond_kv.cpp)

```
void BondTest::compute(int eflag, int vflag)
{
    int i1,i2,n,type;
  double delx,dely,delz,ebond,fbond;
  double rsq,r,dr,rk

/* declaration of new variables */
  double delv_x, delv_y, delv_z;
  double dir_vx1, dir_vy1, dir_vz1, dir_vx2, dir_vy2,
   dir_vz2, dir_vx1, dir_vx1;
  double rsq_x, rsq_y, rsq_z;
/* end declaration of new variables */



  ebond = 0.0;
  if (eflag || vflag) ev_setup(eflag,vflag);
  else evflag = 0;

  double **x = atom->x;
  double **f = atom->f;
  double **v = atom->v; /* delcaration and inizalitaizon
   of a new pointer*/
  }
}
```

Moreover, inside the loop `for (n = 0; n < nbondlist; n++)` of the original `compute`, we need to add a new set of lines between the lines to calculate the spring force and the lines to calculate force and energy increment. Those lines calculate velocities and directions to compute the dashpot forces, see Listing 23.

Now is possible to write the new expression of the force applied to pair of atoms.

Listing 23: Modified `compute` (bond_kv.cpp)

```
/* dashpot velocities and directions */
dev_x = v[ i1 ][ 0 ] - v[ i2 ][ 0 ];
dev_y = v[ i1 ][ 1 ] - v[ i2 ][ 1 ];
dev_z = v[ i1 ][ 2 ] - v[ i2 ][ 2 ];
rsq_vx = dev_x * dev_x;
rsq_vy = dev_y * dev_y;
rsq_vz = dev_z * dev_z;
velx = sqrt( rsq_vx );
vely = sqrt( rsq_vy );
velz = sqrt( rsq_vz );

if ( v[ i1 ][ 0 ] >= 0.0 ) dir_vx1 = 1;
else dir_vx1 = -1;

if ( v[ i1 ][ 1 ] >= 0.0 ) dir_vy1 = 1;
else dir_vy1 = -1;

if ( v[ i1 ][ 2 ] >= 0.0 ) dir_vz1 = 1;
else dir_vz1 = -1;

if ( v[ i2 ][ 0 ] >= 0.0 ) dir_vx2 = 1;
else dir_vx2 = -1;

if ( v[ i2 ][ 1 ] >= 0.0 ) dir_vy2 = 1;
else dir_vy2 = -1;

if ( v[ i2 ][ 2 ] >= 0.0 ) dir_vz2 = 1;
else dir_vz2 = -1;
```

Listing 24: Original `compute` (bond_harmonic.cpp)

```
    if (newton_bond || i1 < nlocal) {
      f[i1][0] += delx*fbond;
      f[i1][1] += dely*fbond;
      f[i1][2] += delz*fbond;
    }

    if (newton_bond || i2 < nlocal) {
      f[i2][0] -= delx*fbond;
      f[i2][1] -= dely*fbond;
      f[i2][2] -= delz*fbond;
    }

    if (evflag) ev_tally(i1,i2,nlocal,
    newton_bond,ebond,fbond,delx,dely,delz);
  }
}
```

Listing 25: Modified `compute` (bond_kv.cpp)

```
/// eq 3 implementation for each force component
    if (newton_bond || i1 < nlocal) {
      f[i1][0] += (delx*fbond) - (dir_vx1*b[type]*velx);
      f[i1][1] += (dely*fbond) - (dir_vy1*b[type]*vely);
      f[i1][2] += (delz*fbond) - (dir_vy1*b[type]*velz);
    }

    if (newton_bond || i2 < nlocal) {
      f[i2][0] -= (delx*fbond) - (dir_vx2*b[type]*velx);
      f[i2][1] -= (dely*fbond) - (dir_vy2*b[type]*vely);
      f[i2][2] -= (delz*fbond) - (dir_vz2*b[type]*velz);
    }

    if (evflag) ev_tally(i1,i2,nlocal,
    newton_bond,ebond,fbond,delx,dely,delz);
  }
}
```

With the introduction of a new parameter in the pair style we need to make a new dynamic memory allocation by modifying `allocate`.

Listing 26: Original `allocate` (bond_harmonic.cpp)

```
 void BondHarmonic::allocate()
{
  allocated = 1;
  int n = atom->nbondtypes;

  memory->create(k,n+1,"bond:k");
  memory->create(r0,n+1,"bond:r0");

  memory->create(setflag,n+1,"bond:setflag");
  for (int i = 1; i <= n; i++) setflag[i] = 0;
}
```

Listing 27: Modified `allocate` (bond_kv.cpp)

```
  void BondKv::allocate()
{
  allocated = 1;
  int n = atom->nbondtypes;

  memory->create(k,n+1,"bond:k");
  memory->create(r0,n+1,"bond:r0");
  memory->create(b,n+1,"bond:b"); // new line to
 // dynamically allocate b

  memory->create(setflag,n+1,"bond:setflag");
  for (int i = 1; i <= n; i++) setflag[i] = 0;
}
```

The viscosity of the damper, b, is given by the user in the input file. For this reason, we also need to modify `coeff`.

Listing 28: Original `coeff` (bond_harmonic.cpp)

```
void BondHarmonic::coeff(int narg, char **arg)
{
  if (narg != 3) error->all(FLERR,"Incorrect args for
  bond coefficients");
  if (!allocated) allocate();

  int ilo,ihi;
  force->bounds(arg[0],atom->nbondtypes,ilo,ihi);

  double k_one = force->numeric(FLERR,arg[1]);
  double r0_one = force->numeric(FLERR,arg[2]);

  int count = 0;
  for (int i = ilo; i <= ihi; i++) {
    k[i] = k_one;
    r0[i] = r0_one;
    setflag[i] = 1;
    count++;
  }

  if (count == 0) error->all(FLERR,"Incorrect args for
   bond coefficients");
}
```

Listing 29: Modified `coeff` (bond_kv.cpp)

```
 void BondKv::coeff(int narg, char **arg)
{
  if (narg != 4) error->all(FLERR,"Incorrect args for
  bond coefficients");
  if (!allocated) allocate();

  int ilo,ihi;
  force->bounds(arg[0],atom->nbondtypes,ilo,ihi);

  double k_one = force->numeric(FLERR,arg[1]);
  double r0_one = force->numeric(FLERR,arg[2]);
  double b_one = force->numeric(FLERR,arg[3]);
// to allocate in b_one the 3rd argument of bond_coeff
  int count = 0;
  for (int i = ilo; i <= ihi; i++) {
    k[i] = k_one;
    r0[i] = r0_one;
    b[i] = b_one;
// to allocate the value stored in b_one used in compute
    setflag[i] = 1;
    count++;
  }

  if (count == 0) error->all(FLERR,"Incorrect args for
  bond coefficients");
}
```

This pair style also has the `write_restart` and `read_restart` functions that have to be modified. They basically, write and read geometry file that can be used as a support file in the input file.

Listing 30: Original `write_restart` and `read_restart` (bond_harmonic.cpp)

```
void BondHarmonic::write_restart(FILE *fp)
{
  fwrite(&k[1],sizeof(double),atom->nbondtypes,fp);
  fwrite(&r0[1],sizeof(double),atom->nbondtypes,fp);
}
/*------------*/
void BondHarmonic::read_restart(FILE *fp)
{
  allocate();

  if (comm->me == 0) {
    fread(&k[1],sizeof(double),atom->nbondtypes,fp);
    fread(&r0[1],sizeof(double),atom->nbondtypes,fp);
  }
  MPI_Bcast(&k[1],atom->nbondtypes,MPI_DOUBLE,0,world);
  MPI_Bcast(&r0[1],atom->nbondtypes,MPI_DOUBLE,0,world);

  for (int i = 1; i <= atom->nbondtypes; i++)
  setflag[i] = 1;
}
```

Listing 31: Modified `write_restart` and `read_restart` (bond_kv.cpp)

```
void BondKv::write_restart(FILE *fp)
{
  fwrite(&k[1],sizeof(double),atom->nbondtypes,fp);
  fwrite(&r0[1],sizeof(double),atom->nbondtypes,fp);
  fwrite(&b[1],sizeof(double),atom->nbondtypes,fp);
}
/*------------*/
void BondKv::read_restart(FILE *fp)
{
  allocate();

```

```
  if (comm->me == 0) {
    fread(&k[1],sizeof(double),atom->nbondtypes,fp);
    fread(&r0[1],sizeof(double),atom->nbondtypes,fp);
    fread(&b[1],sizeof(double),atom->nbondtypes,fp);
  }
  MPI_Bcast(&k[1],atom->nbondtypes,MPI_DOUBLE,0,world);
  MPI_Bcast(&r0[1],atom->nbondtypes,MPI_DOUBLE,0,world);
  MPI_Bcast(&n[1],atom->nbondtypes,MPI_DOUBLE,0,world);


  for (int i = 1; i <= atom->nbondtypes; i++)
  setflag[i] = 1;
}
```

3.3. *bond_kv.h*

In the header of the new pair style we need to substitute the "BondHarmonic" text by a new "BondKv" text as well as declare a new protected member in the class, the pointer to *b*.

Listing 32: Original header (bond_harmonic.h)

```
#ifdef BOND_CLASS

BondStyle(harmonic,BondHarmonic)

#else

#ifndef LMP_BOND_HARMONIC_H
#define LMP_BOND_HARMONIC_H

#include "stdio.h"
#include "bond.h"

namespace LAMMPS_NS {

class BondHarmonic : public Bond {
 public:
  BondHarmonic(class LAMMPS *);
  virtual ~BondHarmonic();
  virtual void compute(int, int);
  void coeff(int, char **);
  double equilibrium_distance(int);
  void write_restart(FILE *);
  void read_restart(FILE *);
  void write_data(FILE *);
  double single(int, double, int, int, double &);

 protected:
  double *k,*r0;

  void allocate();
};
}
#endif
#endif
```

Listing 33: Modified header (bond_kv.h)

```
#ifdef BOND_CLASS

BondStyle(kv,BondKv)

#else

#ifndef LMP_BOND_KV_H
#define LMP_BOND_KV_H

#include "stdio.h"
```

```
#include "bond.h"

namespace LAMMPS_NS {

class BondKv : public Bond {
 public:
  BondKv(class LAMMPS *);
  virtual ~BondKv();
  virtual void compute(int, int);
  void coeff(int, char **);
  double equilibrium_distance(int);
  void write_restart(FILE *);
  void read_restart(FILE *);
  void write_data(FILE *);
  double single(int, double, int, int, double &);

 protected:
 double *k,*r0, *b; // new pointer

  void allocate();
};
}
#endif
#endif
```

3.4. Invoking kv Pair Style

Now the new pair style is completed. To run LAMMPS with the new style we need to compile it and then invoke it by writing the command lines in shown in Listing 34 in the input file.

Listing 34: Command lines to invoke the kv pair style

```
bond_style kv
bond_coeff K r0 b
```

4. Noble–Abel Stiffened-Gas Pair Style

In the SPH framework is possible to determine all the particles properties by solving the particle form of the continuity equation [6,26]

$$\frac{d\rho_i}{dt} = \sum_j m_j \mathbf{v}_{ij} \nabla_j W_{ij}; \tag{4}$$

the momentum equation [6,26]

$$m_i \frac{d\mathbf{v}_i}{dt} = \sum_j m_i m_j \left(\frac{P_i}{\rho_i} + \frac{P_i}{\rho_i} + \Pi_{ij} \right) \nabla_j W_{ij}; \tag{5}$$

and the energy conservation equation [6,26]

$$m_i \frac{de_i}{dt} = \frac{1}{2} \sum_j m_i m_j \left(\frac{P_i}{\rho_i} + \frac{P_i}{\rho_i} + \Pi_{ij} \right) : \mathbf{v}_{ij} \nabla_j W_{ij} - \sum_j \frac{m_i m_j}{\rho_i \rho_j} \frac{(\kappa_i + \kappa_j)(T_i - T_j)}{r_{ij}^2} \mathbf{r_{ij}} \cdot \nabla_j W_{ij}. \tag{6}$$

However, to be able to solve this set of equations an Equation of State (EOS) linking the pressure P and the density ρ is needed [46]. In the user-SPH package of LAMMPS one EOS is used for the liquid (Tait's EOS) and one for gas phase (ideal gas EOS). In this section we will implement a new EOS for the liquid phase. Note that with similar steps is also possible to implement a new gas EOS.

Le Métayer and Saurel [47] combined the "Noble–Abel" and the "Stiffened-Gas" EOS proposing a new EOS called Noble–Abel Stiffened-Gas (NASG), suitable for multiphase flow. The expression of the EOS does not change with the phase considered. For each phases, the pressure and temperature are calculated as function of density and specific internal energy, e.g.,

$$P(\rho, e) = (\gamma - 1)\frac{(e - q)}{\left(\frac{1}{\rho} - b\right)} - \gamma P_{\infty}, \tag{7}$$

and temperature-wise

$$T(\rho, e) = \frac{e - q}{C_v} - \left(\frac{1}{\rho} - b\right)\frac{P_{\infty}}{C_v}, \tag{8}$$

where P, ρ, e, and q are, respectively, the pressure, the density, the specific internal energy, and the heat bond of the corresponding phase. γ, P_{∞}, q, and b are constant coefficients that defines the thermodynamic properties of the fluid.

For this new pair style, called sph/nasgliquid, we take as a reference the sph/taitwater pair style declared and initialised in pair_sph_taitwater.h and pair_sph_taitwater.cpp files in the directory /src/USER-SPH. All the files regarding sph/nasgliquid must be saved in the /src/USER-SPH directory and its hierarchy is shown in Figure 4..

Figure 4. Class hierarchy of the new bond style.

4.1. Validation

The sph/nasgliquid pair style has validated by Albano and Alexiadis [26] to study the Rayleigh collapse of an empty cavity.

4.2. pair_sph_nasgliquid.cpp

All the functions will be the same as in the reference sph/taitwater. However, in our new sph/nasgliquid, we need to substitute the "PairSPHTaitwater" text in "PairSPHNasgliquid", as can be seen in Listings 35 and 36.

Listing 35: Original script (pair_sph_taitwater.cpp)

```
#include <cmath>
#include <cstdlib>
#include "pair_sph_taitwater.h"
#include "atom.h"
#include "force.h"
#include "comm.h"
#include "neigh_list.h"
#include "memory.h"
#include "error.h"
#include "domain.h"

using namespace LAMMPS_NS;

PairSPHTaitwater::PairSPHTaitwater(LAMMPS *lmp) :
Pair(lmp)
{...}
PairSPHTaitwater::~PairSPHTaitwater()
{...}
void PairSPHTaitwater::compute(int eflag, int vflag)
{...}
void PairSPHTaitwater::allocate()
```

```
{...}
void PairSPHTaitwater::settings(int narg, char **/*arg*/)
{...}
void PairSPHTaitwater::coeff(int narg, char **arg)
{...}
double PairSPHTaitwater::init_one(int i, int j)
{...}
```

Listing 36: Modified script (pair_sph_nasgliquid.cpp)

```
#include <cmath>
#include <cstdlib>
#include "pair_sph_nasgliquid.h"
#include "atom.h"
#include "force.h"
#include "comm.h"
#include "neigh_list.h"
#include "memory.h"
#include "error.h"
#include "domain.h"

using namespace LAMMPS_NS;

PairSPHNasgliquid:: PairSPHNasgliquid(LAMMPS *lmp) :
Pair(lmp)
{...}
PairSPHNasgliquid::~ PairSPHNasgliquid()
{...}
void PairSPHNasgliquid::compute(int eflag, int vflag)
{...}
void PairSPHNasgliquid::allocate()
{...}
void PairSPHNasgliquid::settings(int narg, char **/*arg*/)
{...}
void PairSPHNasgliquid::coeff(int narg, char **arg)
{...}
double PairSPHNasgliquid::init_one(int i, int j)
{...}
```

For the sph/nasgliquid we need to pass a total of 12 arguments from the input file, while they were only six for sph/taitwater. For this reason we need to modify destructor, `compute`, `allocate`, `settings` and `coeff`. Following the order of function initialisation, see Listing 36, the destructor is modified as shown in Listing 38.

Listing 37: Original destructor (pair_sph_taitwater.cpp)

```
PairSPHTaitwater::~PairSPHTaitwater() {
  if (allocated) {
    memory->destroy(setflag);
    memory->destroy(cutsq);
    memory->destroy(cut);
    memory->destroy(rho0);
    memory->destroy(soundspeed);
    memory->destroy(B);
    memory->destroy(viscosity);
  }
}
```

Listing 38: Modified destructor (pair_sph_nasgliquid.cpp)

```
PairSPHNasgliquid::~PairSPHNasgliquid() {
  if (allocated) {
    memory->destroy(setflag);
    memory->destroy(cutsq);
    memory->destroy(cut);
    memory->destroy(soundspeed);
    memory->destroy(B);
    memory->destroy(CP);
```

```
    memory->destroy(CV);
    memory->destroy(gamma);
    memory->destroy(P00);
    memory->destroy(b);
    memory->destroy(q);
    memory->destroy(q1);
    memory->destroy(viscosity);
  }
}
```

In the NASG EOS the pressure is function of both density, ρ, and internal energy, e. For this reason, we need to declare more pointers and variables in `compute` compared to the reference pair style, see line 6 and 20 in Listing 40.

Listing 39: Original `compute` (pair_sph_taitwater.cpp)

```
void PairSPHTaitwater::compute(int eflag, int vflag) {
  int i, j, ii, jj, inum, jnum, itype, jtype;
  double xtmp, ytmp, ztmp, delx, dely, delz, fpair;

  int *ilist, *jlist, *numneigh, **firstneigh;
  double vxtmp, vytmp, vztmp, imass, jmass,
  fi, fj, fvisc, h, ih, ihsq;
  double rsq, tmp, wfd, delVdotDelR, mu, deltaE;

  if (eflag || vflag)
    ev_setup(eflag, vflag);
  else
    evflag = vflag_fdotr = 0;

  double **v = atom->vest;
  double **x = atom->x;
  double **f = atom->f;
  double *rho = atom->rho;
  double *mass = atom->mass;
  double *de = atom->de;
  double *drho = atom->drho;
  int *type = atom->type;
  int nlocal = atom->nlocal;
  int newton_pair = force->newton_pair;
```

Listing 40: Modified `compute` (pair_sph_nasgliquid.cpp)

```
void PairSPHNasgliquid::compute(int eflag, int vflag) {
  int i, j, ii, jj, inum, jnum, itype, jtype;
  double xtmp, ytmp, ztmp, delx, dely, delz, fpair;

  int *ilist, *jlist, *numneigh, **firstneigh;
  double vxtmp, vytmp, vztmp, imass, jmass,
  fi, fj, fvisc, h, ih, ihsq, iirho, ijrho;
  double rsq, tmp, wfd, delVdotDelR, mu, deltaE;

  if (eflag || vflag)
    ev_setup(eflag, vflag);
  else
    evflag = vflag_fdotr = 0;

  double **v = atom->vest;
  double **x = atom->x;
  double **f = atom->f;
  double *rho = atom->rho;
  double *mass = atom->mass;
  double *de = atom->de;
  double *e = atom->e;
  double *drho = atom->drho;
  int *type = atom->type;
  int nlocal = atom->nlocal;
  int newton_pair = force->newton_pair;
```

Another modification for `compute` regards the expression of the force applied to the *i*-th, see Listing 42, and *j*-th, see Listing 44, particle.

Listing 41: Original `compute` (pair_sph_taitwater.cpp)

```
// compute pressure of atom i with Tait EOS
tmp = rho[i]/rho0[itype];
fi = tmp * tmp * tmp;
fi = B[itype] * (fi * fi * tmp - 1.0)/ (rho[i] * rho[i]);
```

Listing 42: Modified `compute` (pair_sph_nasgliquid.cpp)

```
// compute pressure of atom i with NASG EOS
tmp = e[i] / imass;
iirho= 1.0/rho[i];
iirho= iirho - b[itype];
fi = (( tmp - q[itype]) * B[itype] / iirho);
fi = fi - gamma[itype] * P00[itype];
fi = fi / (rho[i] * rho[i]);
```

Listing 43: Original `compute` (pair_sph_taitwater.cpp)

```
// compute pressure  of atom j with Tait EOS
tmp = rho[j] / rho0[jtype];
fj = tmp * tmp * tmp;
fj = B[jtype] * (fj * fj * tmp - 1.0) / (rho[j] * rho[j]);
```

Listing 44: Modified `compute` (pair_sph_nasgliquid.cpp)

```
// compute pressure  of atom j with NASG EOS
tmp = e[j] / jmass;
ijrho= 1/rho[j];
ijrho= ijrho - b[jtype];
fj = ((tmp - q[jtype])* B[jtype]/ijrho);
fj = fj - gamma[jtype]*P00[jtype];
fj = fj / (rho[j] * rho[j]);
```

With the introduction of a new parameter in the pair style we need to make a new dynamic memory allocation by modifying `allocate`.

Listing 45: Original allocate (pair_sph_taitwater.pp)

```
void PairSPHTaitwater::allocate() {
  allocated = 1;
  int n = atom->ntypes;

  memory->create(setflag, n + 1, n + 1, "pair:setflag");
  for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++)
      setflag[i][j] = 0;

  memory->create(cutsq, n + 1, n + 1, "pair:cutsq");
  memory->create(rho0, n + 1, "pair:rho0");
  memory->create(soundspeed, n + 1, "pair:soundspeed");
  memory->create(B, n + 1, "pair:B");
  memory->create(cut, n + 1, n + 1, "pair:cut");
  memory->create(viscosity,n + 1,n + 1,"pair:viscosity");
}
```

Listing 46: Modified allocate (pair_sph_nasgliquid.cpp)

```
void PairSPHNasgliquid::allocate() {
  allocated = 1;
  int n = atom->ntypes;

  memory->create(setflag, n + 1, n + 1, "pair:setflag");
  for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++)
      setflag[i][j] = 0;
```

```
9
10  memory->create(cutsq, n + 1, n + 1, "pair:cutsq");
11  memory->create(soundspeed, n + 1, "pair:soundspeed");
12  memory->create(B, n + 1, "pair:B");
13  memory->create(CP, n + 1, "pair:CP");
14  memory->create(CV, n + 1, "pair:CV");
15  memory->create(gamma, n + 1, "pair:gamma");
16  memory->create(P00, n + 1, "pair:P00");
17  memory->create(b, n + 1, "pair:b");
18  memory->create(q, n + 1, "pair:q");
19  memory->create(q1, n + 1, "pair:q1");
20  memory->create(cut, n + 1, n + 1, "pair:cut");
21  memory->create(viscosity,n + 1,n + 1,"pair:viscosity");
22 }
```

The 12 arguments used in the pair style are passed by the used in the input file. For this reason, we also have to modify `coeff`.

Listing 47: Original `coeff` (pair_sph_taitwater.cpp)

```
1  void PairSPHTaitwater::coeff(int narg, char **arg) {
2    if (narg != 6)
3      error->all(FLERR,
4          "Incorrect args for pair_style sph/taitwater
5          coefficients");
6    if (!allocated)
7      allocate();
8    int ilo, ihi, jlo, jhi;
9    force->bounds(FLERR,arg[0], atom->ntypes, ilo, ihi);
10   force->bounds(FLERR,arg[1], atom->ntypes, jlo, jhi);
11   double rho0_one = force->numeric(FLERR,arg[2]);
12   double soundspeed_one = force->numeric(FLERR,arg[3]);
13   double viscosity_one = force->numeric(FLERR,arg[4]);
14   double cut_one = force->numeric(FLERR,arg[5]);
15   double B_one=soundspeed_one*soundspeed_one*rho0_one/7.0;
16   int count = 0;
17   for (int i = ilo; i <= ihi; i++) {
18     rho0[i] = rho0_one;
19     soundspeed[i] = soundspeed_one;
20     B[i] = B_one;
21     for (int j = MAX(jlo,i); j <= jhi; j++) {
22       viscosity[i][j] = viscosity_one;
23       cut[i][j] = cut_one;
24       setflag[i][j] = 1;
25       count++; } }
```

Listing 48: Modified `coeff` (pair_sph_nasgliquid.cpp)

```
1  void PairSPHNasgliquid::coeff(int narg, char **arg) {
2    if (narg != 12)
3      error->all(FLERR,
4          "Incorrect args for pair_style sph/nasgliquid
5           coefficients");
6    if (!allocated)
7      allocate();
8    int ilo, ihi, jlo, jhi;
9    force->bounds(FLERR,arg[0], atom->ntypes, ilo, ihi);
10   force->bounds(FLERR,arg[1], atom->ntypes, jlo, jhi);
11   double soundspeed_one = force->numeric(FLERR,arg[2]);
12   double viscosity_one = force->numeric(FLERR,arg[3]);
13   double cut_one = force->numeric(FLERR,arg[4]);
14   double CP_one = force->numeric(FLERR,arg[5]);
15   double CV_one = force->numeric(FLERR,arg[6]);
16   double gamma_one = force->numeric(FLERR,arg[7]);
17   double P00_one = force->numeric(FLERR,arg[8]);
18   double b_one = force->numeric(FLERR,arg[9]);
19   double q_one = force->numeric(FLERR,arg[10]);
20   double q1_one = force->numeric(FLERR,arg[11]);
21   double B_one = (gamma_one - 1);
```

```
  int count = 0;
  for (int i = ilo; i <= ihi; i++) {
    soundspeed[i] = soundspeed_one;
    B[i] = B_one;
    CP[i] = CP_one;
    CV[i] = CV_one;
    gamma[i] = gamma_one;
    P00[i] = P00_one;
    b[i] = b_one;
    q[i] = q_one;
    q1[i] = q1_one;
    for (int j = MAX(jlo,i); j <= jhi; j++) {
      viscosity[i][j] = viscosity_one;
      cut[i][j] = cut_one;
      setflag[i][j] = 1;
      count++;  }  }
```

4.3. *pair_sph_nasgliquid.h*

In the header of the new pair style we need to substitute the "PairSPHTaitwater" text in "PairSPHNasgliquid" as well as declare new protected members in the class, the pointers to the new arguments.

Listing 49: Original header (pair_sph_taitwater.h)

```
#ifdef PAIR_CLASS

PairStyle(sph/taitwater,PairSPHTaitwater)

#else

#ifndef LMP_PAIR_TAITWATER_H
#define LMP_PAIR_TAITWATER_H

#include "pair.h"

namespace LAMMPS_NS {

class PairSPHTaitwater : public Pair {
 public:
  PairSPHTaitwater(class LAMMPS *);
  virtual ~PairSPHTaitwater();
  virtual void compute(int, int);
  void settings(int, char **);
  void coeff(int, char **);
  virtual double init_one(int, int);

 protected:
  double *rho0, *soundspeed, *B;
  double **cut,**viscosity;
  int first;
  void allocate();
};
}
#endif
#endif
```

Listing 50: Modified header (pair_sph_nasgliquid.h)

```
#ifdef PAIR_CLASS

PairStyle(sph/nasgliquid,PairSPHNasgliquid)

#else

#ifndef LMP_PAIR_NASGLIQUID_H
#define LMP_PAIR_NASGLIQUID_H

```

```
#include "pair.h"

namespace LAMMPS_NS {

class PairSPHNasgliquid : public Pair {
 public:
  PairSPHNasgliquid(class LAMMPS *);
  virtual ~PairSPHNasgliquid();
  virtual void compute(int, int);
  void settings(int, char **);
  void coeff(int, char **);
  virtual double init_one(int, int);

 protected:
  double *soundspeed, *B, *CP, *CV, *gamma, *P00,
  *b, *q, *q1;
  double **cut,**viscosity;
  int first;
  void allocate();
};
}
#endif
#endif
```

4.4. Invoking Sph/Nasgliquid Pair Style

Now the new pair style is completed. To run LAMMPS with the new style we need to compile it and then invoke it by writing the command lines shown in Listing 51 in the input file.

Listing 51: Command lines to invoke the NASG pair style for liquid

```
pair_style sph/nasgliquid
pair_coeff  I J c_0 alpha h Cv Cp gamma P00 b q q'
```

5. Multiphase (Liquid–Gas) Heat Exchange Pair Style

In LAMMPS thermal conductivity between SPH particles is enabled using the sph/heat-conduction pair style inside the user-SPH package. However, the pair style is designed only for mono phase fluid where the thermal conductivities is constant ($\kappa_i = \kappa$). When more than one phase is present, the heat conduction at the interface can be implemented by using [6,26]

$$m_i \frac{de_i}{dt} = \sum_j \frac{m_i m_j}{\rho_i \rho_j} \frac{(\kappa_i + \kappa_j)(T_i - T_j)}{r_{ij}^2} \mathbf{r_{ij}} \cdot \nabla_j W_{ij}. \tag{9}$$

In the new pair style, called sph/heatgasliquid, one phase is assumed to be liquid with an initial temperature of $T_{l,0}$ and the other is assumed to be and ideal gas. Each time-step the temperature of the fluid is updated as [26].

$$T_l = T_{l,0} + \frac{E_l - E_{l,0}}{C_{p,l}}, \tag{10}$$

where $T_{l,0}$ is the reference temperature, E_0 the internal energy in [J], E_l internal energy [J] at the current time step and $C_{p,l}$ is heat capacity of the fluid in [J K^{-1}]. The temperature of the gas is updated following the ideal EOS [26].

$$T_g = MM \frac{(\gamma - 1)e_g}{R}, \tag{11}$$

where MM is the molar mass [kg kmol^{-1}], e_g is the specific internal energy in [J kg^{-1}], γ is the heat capacity ratio and R is the ideal gas constant in [J K^{-1} kmol^{-1}]. Generally the choice of the reference states $E_{l,0}$ is arbitrary, but if the Equation of State (EOS) used for the phase is function of both density and internal energy of the reference state will be determined by the EOS.

In the sph/heatgasliquid pair style is important to check if the i-th and j-th particles are liquid or gas phase to apply either Equation (10) or Equation (11). This "phase check" is explained in Section 5.2 `compute` function is modified.

For the energy balance the new pair style needs $T_{l,0}$, $E_{l,0}$, $C_{p,l}$ and κ_l for the liquid phase and κ_g for the gas phase. Moreover, for the phase check, the particle types of each phases must be specified. All this informations is passed by the user in the in the input file.

The reference pair style is sph/heatconduction. It is declared and initialised in the pair_sph_heatconduction.cpp pair_sph_heatconduction.cpp files in the directory /src/USER-SPH. All the files regarding sph/heatgasliquid must be saved in the /src/USER-SPH directory and its hierarchy is shown in Figure 5.

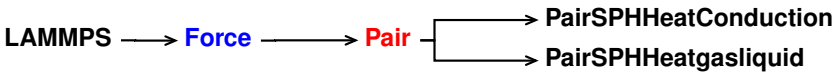


Figure 5. Class hierarchy of the new pair style.

5.1. Validation

The sph/heatgasliquid pair style has validated by Albano and Alexiadis [26] to study the role of the heat diffusion in for a gas filled Rayleigh collapse in water.

5.2. pair_sph_heatgasliquid.cpp

All the functions will be the same as in the reference sph/heatconduction. However, in our new sph/heatgasliquid, we need to substitute the "PairSPHHeatConduction" text in "PairSPHHeatgasliquid", as can be seen in Listings 52 and 53.

Listing 52: Original script (pair_sph_heatconduction.cpp)

```
#include "math.h"
#include "stdlib.h"
#include "pair_sph_heatconduction.h"
#include "atom.h"
#include "force.h"
#include "comm.h"
#include "memory.h"
#include "error.h"
#include "neigh_list.h"
#include "domain.h"

using namespace LAMMPS_NS;

PairSPHHeatConduction::PairSPHHeatConduction(LAMMPS *lmp)
: Pair(lmp)
{ ... }
PairSPHHeatConduction::~PairSPHHeatConduction()
{ ... }
void PairSPHHeatConduction::compute(int eflag, int vflag)
{ ... }
void PairSPHHeatConduction::allocate()
{ ... }
void PairSPHHeatConduction::settings(int narg, char **arg)
{ ... }
void PairSPHHeatConduction::coeff(int narg, char **arg)
{ ... }
double PairSPHHeatConduction::init_one(int i, int j)
{ ... }
double PairSPHHeatConduction::single(int i, int j,
 int itype, int jtype, double rsq, double factor_coul,
 double factor_lj, double &fforce)
{ ... }
```

Listing 53: Modified script (pair_sph_heatgasliquid.cpp)

```
#include <cmath>
#include <cstdlib>
#include "pair_sph_heatgasliquid.h"
#include "atom.h"
#include "force.h"
#include "comm.h"
#include "memory.h"
#include "error.h"
#include "neigh_list.h"
#include "domain.h"

using namespace LAMMPS_NS;

PairSPHHeatgasliquid::PairSPHHeatgasliquid(LAMMPS *lmp)
: Pair(lmp)
{ ... }
PairSPHHeatgasliquid::~PairSPHHeatgasliquid()
{ ... }
void PairSPHHeatgasliquid::compute(int eflag, int vflag)
{ ... }
void PairSPHHeatgasliquid::allocate()
{ ... }
void PairSPHHeatgasliquid::settings(int narg, char **arg)
{ ... }
void PairSPHHeatgasliquid::coeff(int narg, char **arg)
{ ...
double PairSPHHeatgasliquid::init_one(int i, int j)
{ ... }
double PairSPHHeatgasliquid::single(int i, int j,
int itype, int jtype, double rsq, double factor_coul,
double factor_lj, double &fforce)
{ ... }
```

For the sph/heatgasliquid we need to pass a total of nine arguments from the input file, while they were only seven for sph/heatconduction. For this reason we need to modify destructor, `compute`, `allocate`, `settings` and `coeff`. Following the order of function initialisation, see Listing 53, the destructor is modified by removing the heat diffusion coefficient, line 6 in Listing 54.

Listing 54: Original destructor (pair_sph_heatconduction.cpp)

```
PairSPHHeatConduction::~PairSPHHeatConduction() {
  if (allocated) {
    memory->destroy(setflag);
    memory->destroy(cutsq);
    memory->destroy(cut);
    memory->destroy(alpha);
  }
}
```

Listing 55: Modified destructor (pair_sph_heatgasliquid.cpp)

```
PairSPHHeatgasliquid::~PairSPHHeatgasliquid() {
  if (allocated) {
    memory->destroy(setflag);
    memory->destroy(cutsq);
    memory->destroy(cut);
  }
}
```

To compute Equation (6) we need to declare more variables in `compute` compared to the reference pair style, see line 4 in Listing 57.

Listing 56: Original `compute` (pair_sph_heatconduction.cpp)

```
void PairSPHHeatConduction::compute(int eflag, int vflag){
  int i, j, ii, jj, inum, jnum, itype, jtype;
  double xtmp, ytmp, ztmp, delx, dely, delz;
```

Listing 57: Modified `compute` (pair_sph_heatgasliquid.cpp)

```
void PairSPHHeatgasliquid::compute(int eflag, int vflag){
  int i, j, ii, jj, inum, jnum, itype, jtype;
  double xtmp, ytmp, ztmp, delx, dely, delz;
  double Ti, Tj, ki, kj;  /// new parameters
```

Another important modification is to add the phase check inside `compute`. The phase check has to be implemented for both the *i*-th particle and the *j*-th particle inside the loop over neighbours, `for (ii = 0; ii < inum; ii++)` in the reference pair style. The phase check for the *i*-th particle starts after the assignment of `imass`, line 3 of Listing 58.

Listing 58: Modified `compute` (pair_sph_heatgasliquid.cpp)

```
imass = mass[itype];

if (itype == liquidtype)
     {
        Ti= e[i] - el0;
        Ti= Ti/CPl;
        Ti= T0l + Ti;
        ki=kl;
      }
else {
        Ti=0.40*e[i]*18;
        Ti= Ti/imass;
        Ti= Ti/8314.33;
        ki=kg;
      }
```

Similarly, for the *j*-th the phase check start at line 3 of Listing 59.

Listing 59: Modified `compute` (pair_sph_heatgasliquid.cpp)

```
    jmass = mass[jtype];

  if (jtype == liquidtype)
    {
        Tj= e[j] - el0;
        Tj= Tj/CPl;
        Tj= T0l + Tj;
        kj=kl;
  }
  else {
        Tj=0.40*e[j]*18;
        Tj= Tj/jmass;
        Tj= Tj/8314.33;
        kj=kg;
  }

```

The last change in `compute` is to implement the change in internal energy as shown in Equation (9).

Listing 60: Original `compute` (pair_sph_heatconduction.cpp)

```
       D = alpha[itype][jtype]; // diffusion coefficient

        deltaE = 2.0 * imass * jmass / (imass+jmass);
        deltaE *= (rho[i] + rho[j]) / (rho[i] * rho[j]);
        deltaE *= D * (e[i] - e[j]) * wfd;

        de[i] += deltaE;
        if (newton_pair || j < nlocal) {
          de[j] -= deltaE;
        }
```

Listing 61: Modified `compute` (pair_sph_heatgasliquid.cpp)

```
        deltaE = imass * jmass / (rho[i] * rho[j]);  ///
        deltaE *= (ki + kj) * (Ti - Tj) * wfd;   ///
        ///  implementation of eq 3.4
        de[i] += deltaE;
        if (newton_pair || j < nlocal) {
          de[j] -= deltaE;
        }
```

With the introduction of new arguments in the pair style we need to make a new dynamic memory allocation by modifying `allocate`.

Listing 62: Original allocate (pair_sph_heatconduction.cpp)

```
void PairSPHHeatConduction::allocate() {
  allocated = 1;
  int n = atom->ntypes;

  memory->create(setflag, n + 1, n + 1, "pair:setflag");
  for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++)
      setflag[i][j] = 0;

  memory->create(cutsq, n + 1, n + 1, "pair:cutsq");
  memory->create(cut, n + 1, n + 1, "pair:cut");
  memory->create(alpha, n + 1, n + 1, "pair:alpha");
}
```

Listing 63: Modified allocate (pair_sph_heatgasliquid.cpp)

```
void PairSPHHeatgasliquid::allocate() {
  allocated = 1;
  int n = atom->ntypes;

  memory->create(setflag, n + 1, n + 1, "pair:setflag");
  for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++)
      setflag[i][j] = 0;

  memory->create(cutsq, n + 1, n + 1, "pair:cutsq");
  memory->create(cut, n + 1, n + 1, "pair:cut");
}
```

The nine arguments used in the pair style are passed by the user in the input file. For this reason, we also have to modify `coeff`.

Listing 64: Original `coeff` (pair_sph_heatconduction.cpp)

```
void PairSPHHeatConduction::coeff(int narg, char **arg) {
  if (narg != 4)
    error->all(FLERR,"Incorrect number of args for
    pair_style sph/heatconduction coefficients");
  if (!allocated)
    allocate();

```

```
  int ilo, ihi, jlo, jhi;
  force->bounds(arg[0], atom->ntypes, ilo, ihi);
  force->bounds(arg[1], atom->ntypes, jlo, jhi);

  double alpha_one = force->numeric(FLERR,arg[2]);
  double cut_one   = force->numeric(FLERR,arg[3]);

  int count = 0;
  for (int i = ilo; i <= ihi; i++) {
    for (int j = MAX(jlo,i); j <= jhi; j++) {
//printf("setting cut[%d][%d] = %f\n", i, j, cut_one);
      cut[i][j] = cut_one;
      alpha[i][j] = alpha_one;
      setflag[i][j] = 1;
      count++;
    }
  }

  if (count == 0)
    error->all(FLERR,"Incorrect args for pair
    coefficients");
}
```

Listing 65: Modified `coeff` (pair_sph_heatgasliquid.cpp)

```
void PairSPHHeatgasliquid::coeff(int narg, char **arg) {
  if (narg != 9)
    error->all(FLERR,"Incorrect number of args for
     pair_style sph/heatgasliquid coefficients");
  if (!allocated)
    allocate();

  int ilo, ihi, jlo, jhi;
  force->bounds(FLERR,arg[0], atom->ntypes, ilo, ihi);
  force->bounds(FLERR,arg[1], atom->ntypes, jlo, jhi);

  el0 = force->numeric(FLERR,arg[2]);
  kl = force->numeric(FLERR,arg[3]);
  kg = force->numeric(FLERR,arg[4]);
  T0l = force->numeric(FLERR,arg[5]);
  double cut_one = force->numeric(FLERR,arg[6]);
  CPl = force->numeric(FLERR,arg[7]);
  liquidtype = force->numeric(FLERR,arg[8]);

  int count = 0;
  for (int i = ilo; i <= ihi; i++) {
    for (int j = MAX(jlo,i); j <= jhi; j++) {
//printf("setting cut[%d][%d] = %f\n", i, j, cut_one);
      cut[i][j] = cut_one;
      setflag[i][j] = 1;
      count++;
    }
  }
  if (count == 0)
    error->all(FLERR,"Incorrect args for pair
     coefficients");
}
```

5.3. *pair_sph_heatgasliquid.h*

In the header of the new pair style we need to substitute the "PairSPHHeatConduction" text in "PairSPHHeatgasliquid" and declare new protected members in the class.

Listing 66: Original header (pair_sph_heatconduction.h)

```
#ifdef PAIR_CLASS

PairStyle(sph/heatconduction,PairSPHHeatConduction)

#else

#ifndef LMP_PAIR_SPH_HEATCONDUCTION_H
#define LMP_PAIR_SPH_HEATCONDUCTION_H

#include "pair.h"

namespace LAMMPS_NS {

class PairSPHHeatConduction : public Pair {
 public:
  PairSPHHeatConduction(class LAMMPS *);
  virtual ~PairSPHHeatConduction();
  virtual void compute(int, int);
  void settings(int, char **);
  void coeff(int, char **);
  virtual double init_one(int, int);
  virtual double single(int, int, int, int, double,
  double, double, double &);

 protected:
  double **cut, **alpha;
  void allocate();
};
}
#endif
#endif
```

Listing 67: Modified header (pair_sph_heatgasliquid.h)

```
#ifdef PAIR_CLASS

PairStyle(sph/heatgasliquid,PairSPHHeatgasliquid)

#else

#ifndef LMP_PAIR_SPH_HEATGASLIQUID_H
#define LMP_PAIR_SPH_HEATGASLIQUID_H

#include "pair.h"

namespace LAMMPS_NS {

class PairSPHHeatgasliquid : public Pair {
 public:
  PairSPHHeatgasliquid(class LAMMPS *);
  virtual ~PairSPHHeatgasliquid();
  virtual void compute(int, int);
  void settings(int, char **);
  void coeff(int, char **);
  virtual double init_one(int, int);
  virtual double single(int, int, int, int, double,
  double, double, double &);

 protected:
  int liquidtype;
  double el0, kg, kl, T0l, CPl;
  double **cut;
  void allocate();
};
}
#endif
#endif
```

5.4. Invoking Sph/Heatgasliquid Pair Style

Now the new pair style is completed. To run LAMMPS with the new style we need to compile it and then invoke it by writing the command lines shown in Listing 68 in the input file.

Listing 68: Command lines to invoke the sph/heatgasliquid pair style

```
pair_style    sph/heatgasliquid
pair_coeff     i j el0 kl kg Tl0 h Cpl liquidtype
```

6. Full Stationary Fix Style

In LAMMPS a `fix` style is any operation that is applied to the system, usually to a group of particles, during time stepping or minimisation used to alter some property of the sytem [41]. There are hundreds of fixes defined in LAMMPS and new ones can be added. Usually fixes are used for time integration, force constraints, boundary conditions and diagnostics.

In the user-sph package in LAMMPS there is the so called meso/stationary fix used to set boundary condition. With meso/stationary is possible to fix position and velocity for a group of particles, walls as example, but internal energy and density will be updated. In some cases, it is useful to have a fully stationary conditions that maintains constant also the energy and the density. For this new fix, called meso/fullstationary, we take as a reference the meso/stationary fix declared and initialised in fix_meso_stationary.h and fix_meso_stationary.cpp files in the directory /src/USER-SPH. All the files regarding meso/fullstationary must be saved in the /src/USER-SPH directory and its hierarchy is shown in Figure 6.

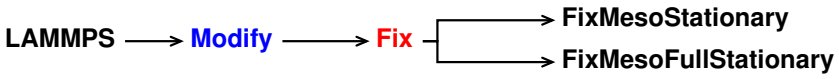


Figure 6. Class hierarchy of the new fix style.

6.1. Validation

The meso/fullstationary has been used in the validation of the new viscosity class to set the boundary condition of a constant asymmetric heated walls, see Section 7.2.

6.2. fix_meso_fullstationary.cpp

All the functions will be the same as in the reference meso/stationary. However, in our new fullstationary, we need to substitute the "FixMesoStationary" text in "FixMeso-FullStationary", as can be seen in Listings 69 and 70.

Listing 69: Original script (fix_meso_stationary.cpp)

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <cstdlib>
#include "fix_meso_stationary.h"
#include "atom.h"
#include "comm.h"
#include "force.h"
#include "neighbor.h"
#include "neigh_list.h"
#include "neigh_request.h"
#include "update.h"
#include "integrate.h"
#include "respa.h"
#include "memory.h"
#include "error.h"
#include "pair.h"

```

```
using namespace LAMMPS_NS;
using namespace FixConst;

FixMesoStationary:: FixMesoStationary(LAMMPS *lmp,
int narg, char **arg) : Fix(lmp, narg, arg)
{...}
int FixMesoStationary::setmask()
{...}
void FixMesoStationary::init()
{...}
void FixMesoStationary::initial_integrate(int /*vflag*/)
{...}
void FixMesoStationary::final_integrate()
{...}
void FixMesoStationary::reset_dt()
{...}
```

Listing 70: Modified script (fix_meso_fullstationary.cpp)

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <cstdlib>
#include "fix_meso_fullstationary.h"
#include "atom.h"
#include "comm.h"
#include "force.h"
#include "neighbor.h"
#include "neigh_list.h"
#include "neigh_request.h"
#include "update.h"
#include "integrate.h"
#include "respa.h"
#include "memory.h"
#include "error.h"
#include "pair.h"

using namespace LAMMPS_NS;
using namespace FixConst;

FixMesoFullStationary::FixMesoFullStationary(LAMMPS *lmp,
 int narg, char **arg) : Fix(lmp, narg, arg)
{...}
int FixMesoFullStationary::setmask()
{...}
void FixMesoFullStationary::init()
{...}
void FixMesoFullStationary::initial_integrate
(int /*vflag*/)
{...}
void FixMesoFullStationary::final_integrate()
{...}
void FixMesoFullStationary::reset_dt()
{...}
```

For the meso/fullstationary we need to modify two function: `initial_integrate`, see Listing 72 line 16 and 17, and `final_integrate`, see Listing 74 line 14 and 15.

Listing 71: Original initial_integrate (fix_meso_stationary.cpp)

```
void FixMesoStationary::initial_integrate(int vflag) {

  double *rho = atom->rho;
  double *drho = atom->drho;
  double *e = atom->e;
  double *de = atom->de;
  int *type = atom->type;
  int *mask = atom->mask;
  int nlocal = atom->nlocal;
  int i;

  if (igroup == atom->firstgroup)
    nlocal = atom->nfirst;

  for (i = 0; i < nlocal; i++) {
    if (mask[i] & groupbit) {
      e[i] += dtf * de[i];
// with this line is possible to update internal energy
      rho[i] += dtf * drho[i];
// ... and density every half-step
    }}}
```

Listing 72: Modified initial_integrate (fix_meso_fullstationary.cpp)

```
void FixMesoFullStationary::initial_integrate(int vflag) {

  double *rho = atom->rho;
  double *drho = atom->drho;
  double *e = atom->e;
  double *de = atom->de;
  int *mask = atom->mask;
  int nlocal = atom->nlocal;
  int i;

  if (igroup == atom->firstgroup)
    nlocal = atom->nfirst;

  for (i = 0; i < nlocal; i++) {
    if (mask[i] & groupbit) {
      e[i] +=0; // with this line internal energy
      rho[i] += 0; // ... and density are constant
    }}}
```

Listing 73: Original final_integrate (fix_meso_stationary.cpp)

```
void FixMesoStationary::final_integrate() {

  double *e = atom->e;
  double *de = atom->de;
  double *rho = atom->rho;
  double *drho = atom->drho;
  int *type = atom->type;
  int *mask = atom->mask;
  double *mass = atom->mass;
  int nlocal = atom->nlocal;
  if (igroup == atom->firstgroup)
    nlocal = atom->nfirst;

  for (int i = 0; i < nlocal; i++) {
    if (mask[i] & groupbit) {
      e[i] += dtf * de[i];
      rho[i] += dtf * drho[i];
    }}}
```

Listing 74: Modified final_integrate (fix_meso_fullstationary.cpp)

```
vvoid FixMesoFullStationary::final_integrate() {

  double *e = atom->e;
  double *de = atom->de;
  double *rho = atom->rho;
  double *drho = atom->drho;
  int *mask = atom->mask;
  int nlocal = atom->nlocal;
  if (igroup == atom->firstgroup)
    nlocal = atom->nfirst;

  for (int i = 0; i < nlocal; i++) {
    if (mask[i] & groupbit) {
      e[i] += 0; //  with this line internal energy
      rho[i] += 0; //... and density are constant
    }}}
```

6.3. *fix_mes_fullstationary.h*

In the header of the new fix we need to substitute the "FixMesoStationary" text in "FixMesoFullStationary".

Listing 75: Original header (pair_sph_heatconduction.h)

```
#ifdef FIX_CLASS

FixStyle(meso/stationary,FixMesoStationary)

#else

#ifndef LMP_FIX_MESO_STATIONARY_H
#define LMP_FIX_MESO_STATIONARY_H

#include "fix.h"

namespace LAMMPS_NS {

class FixMesoStationary : public Fix {
 public:
  FixMesoStationary(class LAMMPS *, int, char **);
  int setmask();
  virtual void init();
  virtual void initial_integrate(int);
  virtual void final_integrate();
  void reset_dt();

 private:
  class NeighList *list;
 protected:
  double dtv,dtf;
  double *step_respa;
  int mass_require;

  class Pair *pair;
};
}
#endif
#endif
```

Listing 76: Modified header (pair_sph_heatgasliquid.h)

```
#ifdef FIX_CLASS

FixStyle(meso/fullstationary,FixMesoFullStationary)

#else

```

```
#ifndef LMP_FIX_MESO_FULLSTATIONARY_H
#define LMP_FIX_MESO_FULLSTATIONARY_H

#include "fix.h"

namespace LAMMPS_NS {

class FixMesoFullStationary: public Fix {
 public:
  FixMesoFullStationary(class LAMMPS *, int, char **);
  int setmask();
  virtual void init();
  virtual void initial_integrate(int);
  virtual void final_integrate();
  void reset_dt();

 private:
  class NeighList *list;
 protected:
  double dtv,dtf;
  double *step_respa;
  int mass_require;

  class Pair *pair;
};
}
#endif
#endif
```

6.4. Invoking Meso/Fullstationary Fix

Now the new fix is completed. To run LAMMPS with the new style we need to compile it and then invoke it by writing the command lines shown in Listing 77 in the input file.

Listing 77: Command lines to invoke the new pair style

```
fix ID group-ID meso/fullstationary
```

7. Viscosity Class

Viscosity in the SPH method has been addressed with different solutions [46]. Shock waves, for example, have been a challenge to model due to the arise of numerical oscillations around the shocked region. Monaghan solved this problem with the introduction of the so-called Monaghan artificial viscosity [48]. Artificial viscosity is still used nowadays for energy dissipation and to prevent unphysical penetration for particles approaching each other [25,49]. The SPH package of LAMMPS uses the following artificial viscosity expression [6], within the sph/idealgas and sph/taitwater pair style.

$$\Pi_{ij} = -\alpha h \frac{c_i + c_j}{\rho_i + \rho_j} \frac{\mathbf{v}_{ij} \cdot \mathbf{r}_{ij}}{r_{ij}^2 + \epsilon h^2}, \tag{12}$$

where α is the dimensionless dissipation factor, c_i and c_j are the speed of sound of particle i and j. The dissipation factor, α, can be linked with the real viscosity in term of [6]

$$\alpha = 8 \frac{\mu}{ch\rho}, \tag{13}$$

where c is the speed of sound, ρ the density, μ the dynamic viscosity and h the smoothing length.

The artificial viscosity approach performs well at a high Reynolds number but better solutions are available for laminar flow: Morris et al. [50] approximated and implemented the viscosity momentum term for SPH. The same solution can be found in the sph/taitwater/morris pair style with the expression [6].

$$\sum_j \frac{m_i m_j(\mu_i + \mu_j)\mathbf{v}_{ij}}{\rho_i \rho_j}\left(\frac{1}{r_{ij}}\frac{\partial W_{ij}}{\partial r_i}\right), \tag{14}$$

where μ is the real dynamic viscosity.

In LAMMPS both the dissipation factor and the dynamic viscosity are treated as a constant between a pair of particles when they interact within the smoothing length. In this section we want to make the viscosity a per atom property instead of a pair property only existing within a pair style. Moreover, five temperature dependent viscosity models are added. For this example, no reference file is used; a new class, Viscosity, is implemented in LAMMPS from scratch and its hierarchy is shown in Figure 7.

Figure 7. Class hierarchy of the new class.

7.1. Temperature Dependant Viscosity

In literature multiples empirical models that correlate viscosity with temperature are available [51–53]. In the new viscosity class five different viscosity models have been implemented:

1. Andrade's equation [54]

$$\mu = Aexp\left(\frac{B}{T} + CT + DT^2\right), \tag{15}$$

where μ is the viscosity in [Kg m^{-1} s^{-1}], T is the static temperature in Kelvin, A, B, C and D are fluid-dependent dimensional coefficients available in literature.

2. Arrhenius viscosity by Raman [55,56]

$$\mu = C_1 exp(C_2/T), \tag{16}$$

where μ is the dynamic viscosity in [Kg m^{-1} s^{-1}], T is the temperature in Kelvin, C_1 and C_2 are fluid-dependent dimensional coefficients available in literature.

3. Sutherland's viscosity [57,58] for gas phase
Sutherland's law can be expressed as:

$$\mu = \frac{C_1 T^{3/2}}{T + C_2}, \tag{17}$$

where μ is the viscosity in [Kg m^{-1} s^{-1}], T is the static temperature in Kelvin, C_1 and C_2 are dimensional coefficients.

4. Power-Law viscosity law [57] for gas phase
A power-law viscosity law with two coefficients has the form :

$$\mu = BT^n, \tag{18}$$

where μ is the viscosity in [Kg m^{-1} s^{-1}], T is the static temperature in Kelvin, and B is a dimensional coefficient.

5. Constant viscosity

With constant viscosity both dissipation factor and dynamic viscosity will be constant during the simulation.

When the artificial viscosity is used the dissipation factor of Equation (12) is defined as the arithmetic mean of the dissipation factors of i-th particle and j-th particle.

$$\alpha_{ij} = -\frac{4}{h}\left(\frac{\mu_i}{c_i\rho_i} + \frac{\mu_j}{c_j\rho_j}\right), \tag{19}$$

where α_{ij} is the dissipation factor of the particles pair i and j.

7.2. Validation

In order to validate the new Viscosity class, we will study the effect of asymmetrically heating walls in a channel flow, and more specifically the effect on the velocity field of the fluid. The data obtained with our model will be compared with the analytical solution obtained by Sameen and Govindarajan [59].

The water flows between two walls in the x-direction with periodic conditions. The walls are set at different temperatures T_{cold} and T_{hot} , see Figure 8. Both water and walls are modelled as fluid following the tait EOS. The physical properties of the walls are set constant throughout the simulation using the full stationary conditions described in Section 6.

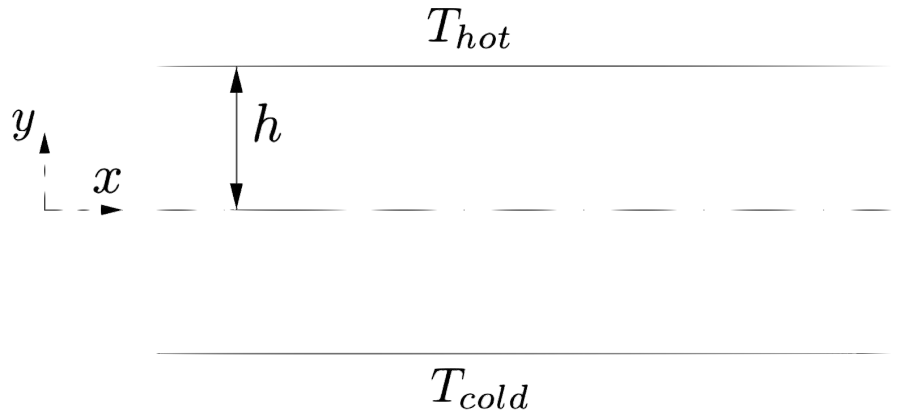


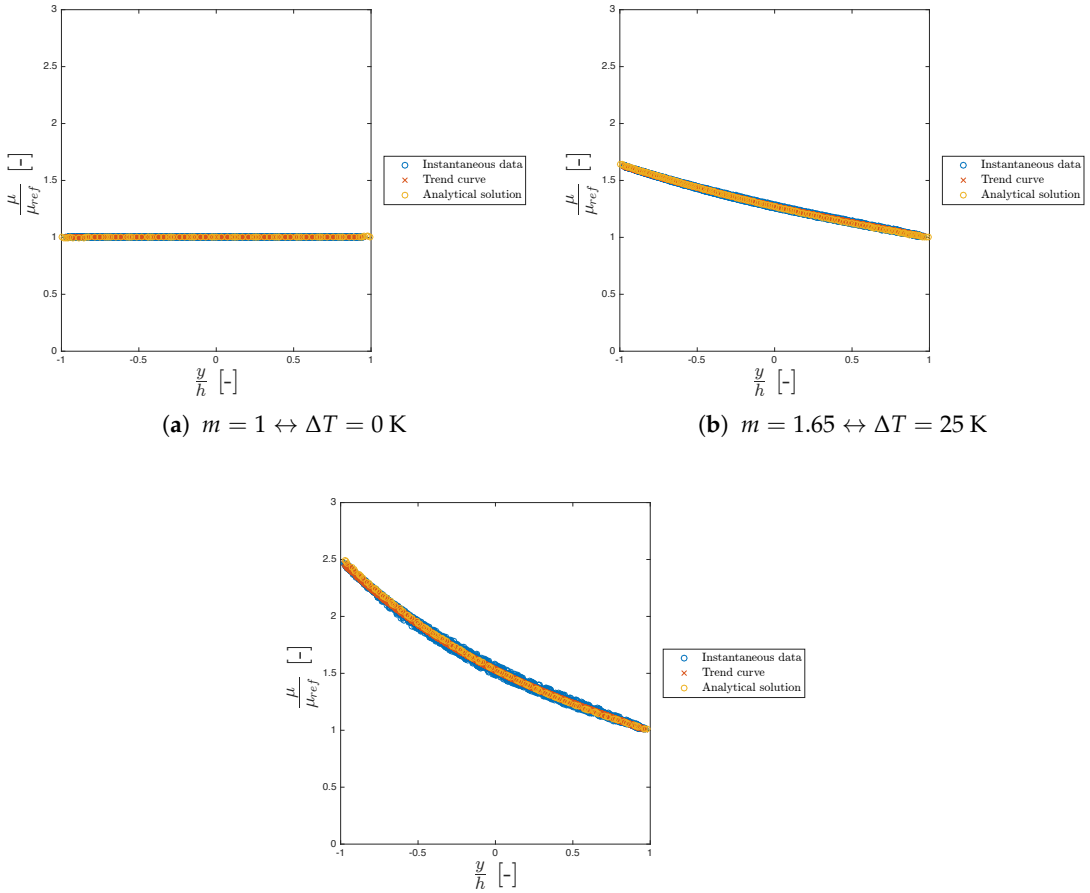
Figure 8. Geometry of the simulation.

To match the condition used by Sameen and Govindarajan we set the cold wall temperature to $T_{cold} = 295$ K and the temperature dependence of the dynamic viscosity described by the Arrhenius model, Equation (16), with $C_1 = 0.000183$ [Ns m^{-2}] and $C_2 = 1879.9$ K [59]. To describe the asymmetric heating Sameen and Govindarajan introduced the parameter m, defined as:

$$m = \frac{\mu_{cold}}{\mu_{ref}} \tag{20}$$

where $\mu_{ref} = \mu_{hot}$ is the viscosity at the hot wall in the case of asymmetric heating and μ_{cold} is the viscosity at the cold wall. By combining (16) and (20), with the given T_{cold}, is possible to express the temperature difference of the walls ΔT as function of m.

Figure 9 shows the viscosity trend for different values of m and the corresponding ΔT. Sometimes, in particle methods, instantaneous data can be noisy (scattered) as can be seen from the blue circles of both Figures 9 and 10.



(**a**) $m = 1 \leftrightarrow \Delta T = 0$ K (**b**) $m = 1.65 \leftrightarrow \Delta T = 25$ K

(**c**) $m = 2.5 \leftrightarrow \Delta T = 50$ K

Figure 9. Dimensionless viscosity profile for different $m = \mu_{cold}/\mu_{ref}$. Blue circles are the instantaneous data in the x direction, the orange curve is the trend curve extrapolated from the instantaneous data, yellow circles are obtained form the analytical solution from Sameen and Govindarajan [59].

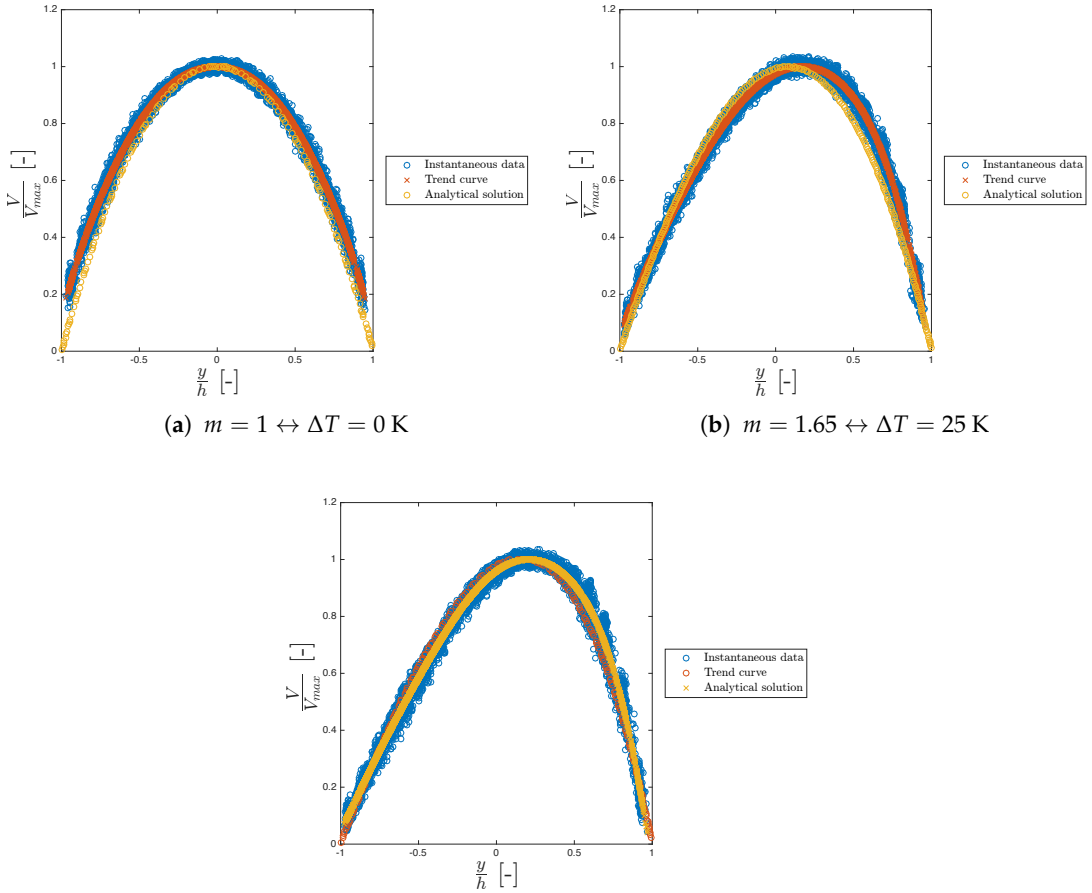
In all the cases considered, the model is in good agreement with the work of Sameen and Govindarajan.

Figure 10 shows the dimensionless velocity trend for different values of m.

Again, the model is in good agreement with the analytical solution of Sameen and Govindarajan always laying within the velocity scattered points. In both our model and in the analytical solution the maximum of the velocity shifts to the right as m increases. We can conclude that our model is in good agreement with the literature, showing the typical viscosity and velocity profiles for asymmetric heating confirming the correct functionality of the new viscosity class.

7.3. New Abstract Class: Viscosity

To implement the new viscosity model a new abstract class has been created, called Viscosity. The class has no attribute, and one virtual method: `compute_visc`, that is used to compute the viscosity using one of the Equations (15)–(18). As usual, the Viscosity class is divided in two files, see Listings 78 and 79. As it is an abstract class, it cannot be instantiated. It is used as a base, a mold, to implement the viscosity models. All implemented viscosity classes, such as the ones implementing the Arrhenius viscosity or the Sutherland viscosity, will inherit from this class.



(**a**) $m = 1 \leftrightarrow \Delta T = 0$ K (**b**) $m = 1.65 \leftrightarrow \Delta T = 25$ K

(**c**) $m = 2.5 \leftrightarrow \Delta T = 50$ K

Figure 10. Dimensionless velocity profile for different $m = \mu_{cold}/\mu_{ref}$. Blue circles are the instantaneous data in the x direction, the orange curve is the trend curve extrapolated from the instantaneous data, yellow circles are obtained form the analytical solution from Sameen and Govindarajan [59].

Listing 78: viscosity.cpp

```
#include "viscosity.h"

using namespace LAMMPS_NS;

Viscosity::Viscosity() {};
```

Listing 79: viscosity.h

```
#ifndef LAMMPS_VISCOSITY_H
#define LAMMPS_VISCOSITY_H

namespace LAMMPS_NS {
    class Viscosity {
        /**
         * Abstract base class for the viscosity attribute.
         * All viscosity types should inherit from this class.
         */
    public:
        Viscosity();
        /**
         * Virtual function.
         * Returns the viscosity, given the temperature.
         */
        virtual double compute_visc(double temperature) = 0;
```

```
    };
}
#endif //LAMMPS_VISCOSITY_H
```

This type of base class is called an interface, though as the code is written in C++, there is no actual difference in the implementation. The difference is only in concepts.

This structure allows for a very simple procedure to add a new viscosity type to LAMMPS, as one doesn't have to go through all of the code everytime a new viscosity type is implemented. All that is required is to implement a new viscosity class inheriting from the Viscosity abstract class and modify the `add_viscosity` function. The details of the changes required for those two actions are detailed later in this section.

Another structure one might think of to implement the viscosity abstract base class would be a template. Indeed, templates are more efficient than inherited classes as inherited classes create additional virtual calls when calling the class's methods. However, the choice of which viscosity should be called is made at runtime, and not at compile time, which means the abstract base class would be a better fit. When runtime polymorphism is needed, the structure preferred is an abstract base class.

The abstract class is not the most efficient implementation, but it allows for simplicity of use, which is important considering most of LAMMPS users are not programmers. In this work, we have chosen to sacrifice a bit of efficiency to gain ease of use.

7.4. Implementing a New Viscosity Class

In this section the steps to implement one of Equations (15)–(18) are shown, using the four parameter exponential viscosity as an example.

A new class is created that inherits from the Viscosity abstract class. The new class have as much attributes as the viscosity type has parameters. In this example that means four, as shown in the header in Listing 80.

Listing 80: viscosity_four_parameter_exp.h

```
#ifndef LAMMPS_VISCOSITY_FOURPARAMETEREXP_H
#define LAMMPS_VISCOSITY_FOURPARAMETEREXP_H

#include "math.h"
#include "viscosity.h"
namespace LAMMPS_NS{

class ViscosityFourParameterExp  : public Viscosity{
    /**
     * Implementation of the four parameter exponential viscosity.
     * This viscosity has four attributes.
     */
private:
    double A;
    double B;
    double C;
    double D;
public:
    ViscosityFourParameterExp(double A, double B, double C, double D);

    double compute_visc(double temperature) override final;

};
};
#endif //LAMMPS_VISCOSITY_FOURPARAMETEREXP_H
```

The constructor therefore should take as arguments the four parameters of the Andrade's equation and initialise the class's attributes with those values. The last step is to implement the `compute_visc` method so it returns the value of the viscosity at the given temperature. The implementation of both those functions is shown in Listing 81.

Listing 81: viscosity_four_parameter_exp.cpp

```
#include "viscosity_four_parameter_exp.h"

using namespace LAMMPS_NS;

ViscosityFourParameterExp::ViscosityFourParameterExp(double A, double B, double C,
     double D) {
    this->A = A;
    this->B = B;
    this->C = C;
    this->D = D;
}

double ViscosityFourParameterExp::compute_visc(double temperature) {
    return  A*exp(B/temperature +C*temperature + D *temperature*temperature);
}
```

Similar steps have to be taken to implement the classes corresponding to the other viscosity models, see the Supplementary material.

7.5. Processing the Viscosity in the Atom Class

In the header of the Atom class we need to include the new viscosity class and declare a new set of public members.

Listing 82: Original header (atom.h)

```
#include "pointers.h"
#include <map>
#include <string>
```

Listing 83: Modified header (atom.h)

```
#include "pointers.h"
#include "viscosity.h"
#include <map>
#include <string>
```

We add two new attributes in the USER-SPH section of the Atom attribute lists: `viscosity`, a pointer to a Viscosity object and `viscosities`, a pointer to an array containing the values of dynamic viscosities for all atoms at the current time step.

Listing 84: Original header (atom.h)

```
 // USER-SPH package
  double *rho,*drho,*e,*de,*cv;
  double **vest;
```

Listing 85: Modified header (atom.h)

```
 // USER-SPH package
  double *rho,*drho,*e,*de,*cv;
  double **vest;
  Viscosity *viscosity;
  double *viscosities;

```

We want to be able to choose which type of viscosity is being used in the simulation from the input file, using a new command called `viscosity`. Let's discuss the implementation of this feature. First we need to define the `viscosity` command. This is done by modifying the `execute_command` method of the Input class. We then define a new function called `add_viscosity`, whose declaration is shown in Listing 86 and definition in Listing 87. This function will have to be modified each time one wants to create a new viscosity class. In `add_viscosity`, the element arg[0] is the string representing the type of viscosity. For each viscosity class, the method performs the following procedure:

- It checks which type of viscosity is asked to be created using the function strcmp on arg[0] (for Andrade's viscosity it corresponds to line 3 of Listing 87)
- It checks if the number of arguments is coherent with the number of parameter of the viscosity type (line 4–5)
- It scans the coefficients of that viscosity type (line 6–10)
- It creates the appropriate viscosity and initializes the Viscosity attribute (line 11).

This process should be followed for any new implementation.

Listing 86: Modified header (atom.h)

```
1  void add_viscosity(int narg, char **arg);
```

Listing 87: New add_viscosity function (atom.cpp)

```
1  void Atom::add_viscosity(int narg, char **arg) {
2      if (narg < 1) error->all(FLERR, "Too few arguments for creation of viscosity");
3      if (!strcmp(arg[0], "FourParameterExp")) {
4          if (narg != 5)
5            error->all(FLERR, "Wrong number of arguments for creation of four
6            parameter exponential viscosity");
7          double A, B, C, D;
8          sscanf(arg[1], "%lg", &A);
9          sscanf(arg[2], "%lg", &B);
10         sscanf(arg[3], "%lg", &C);
11         sscanf(arg[4], "%lg", &D);
12         this->viscosity = new ViscosityFourParameterExp(A, B, C, D);
13         std::cout <<"Viscosity created" <<std::endl;
14     } else {
15         if (!strcmp(arg[0], "SutherlandViscosityLaw")) {
16             if (narg != 3)
17                 error->all(FLERR, "Wrong number of arguments for creation of Sutherland
       viscosity");
18             double A, B;
19             sscanf(arg[1], "%lg", &A);
20             sscanf(arg[2], "%lg", &B);
21             this->viscosity = new SutherlandViscosityLaw(A, B);
22             std::cout <<"Viscosity created" <<std::endl;
23         } else {
24             if (!strcmp(arg[0], "PowerLawGas")) {
25                 if (narg != 2)
26                   error->all(FLERR, "Wrong number of arguments for creation of power law
       gas viscosity");
27                 double B;
28                 sscanf(arg[1], "%lg", &B);
29                 this->viscosity = new PowerLawGas(B);
30                 std::cout <<"Viscosity created" <<std::endl;
31             } else {
32                 if (!strcmp(arg[0], "Arrhenius")) {
33                     if (narg != 3)
34                       error->all(FLERR, "Wrong number of arguments for creation of
       Arrhenius viscosity");
35                     double A;
36                     double B;
37                     sscanf(arg[1], "%lg", &A);
38                     sscanf(arg[2], "%lg", &B);
39                     this->viscosity = new ViscosityArrhenius(A, B);
40                     std::cout <<"Viscosity created" <<std::endl;
41                 } else {
42                     if (!strcmp(arg[0], "Constant")) {
43                         if (narg != 2)
44                             error->all(FLERR, "Wrong number of arguments for creation
45                             of Constant viscosity");
46                         double A;
47                         sscanf(arg[1], "%lg", &A);
48                         this->viscosity = new ViscosityConstant(A);
49                         std::cout <<"Viscosity created" <<std::endl;
50                     } else {
51                         std::cout <<"Nothing implemented for " << arg[0]<< std::endl;
```

```
                    }
                }
            }
        }
    }
}
```

All headers of the new viscosity types implemented in the `add_viscosity` function need to be included in the Atom class, see Listing 88.

Listing 88: New include (atom.cpp)

```
#include <string.h>
#include <iostream>
#include "viscosity_four_parameter_exp.h"
#include "viscosity_sutherland_law.h"
#include "viscosity_power_law_gas.h"
#include "viscosity_arrhenius.h"
#include "viscosity_constant.h"
```

The viscosity attribute is initialised to NULL in the constructor, see Listing 89.

Listing 89: Inside Atom::Atom(LAMMPS *lmp) : Pointers(lmp) (atom.cpp)

```
viscosity = NULL;
```

In the destructor of the Atom class, we add a line to delete the viscosity attribute, see Listing 90.

Listing 90: Inside Atom:: Atom() (atom.cpp)

```
  memory->destroy(viscosity);
```

The `extract` function is modified to process the viscosity attribute, see Listing 91.

Listing 91: Modified extract function (atom.cpp)

```
  if (strcmp(name, "viscosity") == 0) return (void *) viscosity;
```

7.6. Using compute_Visc in SPH Pair Styles: Tait Water Implementation

The dynamic viscosity is used to compute the artificial viscosity force, that is used in the compute function of the following SPH pair style: sph/idealgas, sph/lj, sph/taitwater and sph/taitwater/morris. In this section the steps to implement compute_visc in sph/taitwater are shown, the others required a similar procedure.

The first function to modify is the destructor, as we don't have to allocate the viscosity parameter anymore.

Listing 92: Original file (pair_sph_taitwater.cpp)

```
PairSPHTaitwater::~PairSPHTaitwater() {
  if (allocated) {
    memory->destroy(setflag);
    memory->destroy(cutsq);
    memory->destroy(cut);
    memory->destroy(rho0);
    memory->destroy(soundspeed);
    memory->destroy(B);
    memory->destroy(viscosity);
  }
}
```

Listing 93: Modified file (pair_sph_taitwater.cpp)

```
PairSPHTaitwater::~PairSPHTaitwater() {
  if (allocated) {
    memory->destroy(setflag);
    memory->destroy(cutsq);
```

```
    memory->destroy(cut);
    memory->destroy(rho0);
    memory->destroy(soundspeed);
    memory->destroy(B);
  }
}
```

For the same reason as the destructor we need to modify allocate.

Listing 94: Original file (pair_sph_taitwater.cpp)

```
void PairSPHTaitwater::allocate() {
  allocated = 1;
  int n = atom->ntypes;
  memory->create(setflag, n + 1, n + 1, "pair:setflag");
  for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++)
      setflag[i][j] = 0;
  memory->create(cutsq, n + 1, n + 1, "pair:cutsq");
  memory->create(rho0, n + 1, "pair:rho0");
  memory->create(soundspeed, n + 1, "pair:soundspeed");
  memory->create(B, n + 1, "pair:B");
  memory->create(cut, n + 1, n + 1, "pair:cut");
  memory->create(viscosity, n + 1,n + 1,"pair:viscosity");
}
```

Listing 95: Modified file (pair_sph_taitwater.cpp)

```
void PairSPHTaitwater::allocate() {
  allocated = 1;
  int n = atom->ntypes;
 memory->create(setflag, n + 1, n + 1, "pair:setflag");
  for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++)
      setflag[i][j] = 0;
  memory->create(cutsq, n + 1, n + 1, "pair:cutsq");
  memory->create(rho0, n + 1, "pair:rho0");
  memory->create(soundspeed, n + 1, "pair:soundspeed");
  memory->create(B, n + 1, "pair:B");
  memory->create(cut, n + 1, n + 1, "pair:cut");
}
```

Inside the compute function of the sph/taitwater pair style we need to declare a new set of variables. Where e is the energy and cv the heat capacity, now needed to calculate the temperature and thus the viscosity.

Listing 96: Original file (pair_sph_taitwater.cpp)

```
int *type = atom->type;
int nlocal = atom->nlocal;
int newton_pair = force->newton_pair;
```

```
        [linebackgroundcolor={\listyellow{4,5,6,7}},
        label=820, caption={\small Modified file (pair\textunderscore sph\textunderscore
    taitwater.cpp)}\label{32}]  % Start your code-block

int *type = atom->type;
int nlocal = atom->nlocal;
int newton_pair = force->newton_pair;
double *e = atom->e;
double *cv = atom->cv;
Viscosity* viscosity = atom->viscosity;
double* viscosities = atom->viscosities;
```

The next modification is inside the loop over the j-th atom when the force induced by the artificial viscosity is calculated inside the pair's `compute` function.

The dynamic viscosities μ_i and μ_j are calculated for each atoms, using the formula implemented in the `compute_visc` method. The temperature for the i-th atom is obtained

using $T_i = e_i/cv_i$. It is important to note that using such expression for the energy balance prevents the reference state of the internal energy to be set at 0.

The constant viscosity matrix element is replaced by the formula defined in Equation (19), see Listings 97 and 98.

Listing 97: Original file (pair_sph_taitwater.cpp)

```
// artificial viscosity (Monaghan 1992)
if (delVdotDelR < 0.) {
    mu = h * delVdotDelR / (rsq + 0.01 * h * h);
    fvisc = -viscosity[itype][jtype] * (soundspeed[itype]
     + soundspeed[jtype]) * mu / (rho[i] + rho[j]);
} else {
    fvisc = 0.;
}
```

Listing 98: Modified file (pair_sph_taitwater.cpp)

```
viscosities[i] = viscosity->compute_visc(e[i]/cv[i]);
viscosities[j] = viscosity->compute_visc(e[j]/cv[j]);
// artificial viscosity (Monaghan 1992)
if (delVdotDelR < 0.) {
    mu = h * delVdotDelR / (rsq + 0.01 * h * h);
    fvisc =-4/h*(viscosities[i]/(soundspeed[itype]*rho[i])
    +viscosities[j]/(soundspeed[jtype]*rho[j]))
    *(soundspeed[itype]+ soundspeed[jtype])
    * mu / (rho[i] + rho[j]);
} else {
    fvisc = 0.;
}
```

Viscosity is now a per atom property, this means that we don't have to pass its value then the pair style is invoked. For this reason we need to delete the viscosity related lines inside coeff.

Listing 99: Original coeff (pair_sph_taitwater.cpp)

```
void PairSPHTaitwater::coeff(int narg, char **arg) {
  if (narg != 6)
    error->all(FLERR,
        "Incorrect args for pair_style sph/taitwater
         coefficients");
  if (!allocated)
    allocate();

  int ilo, ihi, jlo, jhi;
  force->bounds(FLERR,arg[0], atom->ntypes, ilo, ihi);
  force->bounds(FLERR,arg[1], atom->ntypes, jlo, jhi);

  double rho0_one = force->numeric(FLERR,arg[2]);
  double soundspeed_one = force->numeric(FLERR,arg[3]);
  double viscosity_one = force->numeric(FLERR,arg[4]);
  double cut_one = force->numeric(FLERR,arg[5]);
  double B_one = soundspeed_one*soundspeed_one*rho0_one/7;

  int count = 0;
  for (int i = ilo; i <= ihi; i++) {
    rho0[i] = rho0_one;
    soundspeed[i] = soundspeed_one;
    B[i] = B_one;
    for (int j = MAX(jlo,i); j <= jhi; j++) {
      viscosity[i][j] = viscosity_one;
      cut[i][j] = cut_one;
      setflag[i][j] = 1;
      count++;
    }
  }
  if (count == 0)
    error->all(FLERR,"Incorrect args for pair
```

```
    coefficients");
}
```

Listing 100: Modified coeff (pair_sph_taitwater.cpp)

```
void PairSPHTaitwater::coeff(int narg, char **arg) {
  if (narg != 5)
    error->all(FLERR,
        "Incorrect args for pair_style sph/taitwater
         coefficients");
  if (!allocated)
    allocate();

  int ilo, ihi, jlo, jhi;
  force->bounds(FLERR,arg[0], atom->ntypes, ilo, ihi);
  force->bounds(FLERR,arg[1], atom->ntypes, jlo, jhi);

  double rho0_one = force->numeric(FLERR,arg[2]);
  double soundspeed_one = force->numeric(FLERR,arg[3]);
  double cut_one = force->numeric(FLERR,arg[4]);
  double B_one = soundspeed_one*soundspeed_one*rho0_one/7;

  int count = 0;
  for (int i = ilo; i <= ihi; i++) {
    rho0[i] = rho0_one;
    soundspeed[i] = soundspeed_one;
    B[i] = B_one;
    for (int j = MAX(jlo,i); j <= jhi; j++) {
      cut[i][j] = cut_one;
     setflag[i][j] = 1;
      count++;
    }
  }
  if (count == 0)
    error->all(FLERR,"Incorrect args for pair
     coefficients");
}
```

The last modification is in init_one. Again, we delete lines related to the former viscosity attribute.

Listing 101: Original file (pair_sph_taitwater.cpp)

```
double PairSPHTaitwater::init_one(int i, int j) {
  if (setflag[i][j] == 0) {
    error->all(FLERR,"All pair sph/taitwater coeffs
     are set");
  }
  cut[j][i] = cut[i][j];
  viscosity[j][i] = viscosity[i][j];
  return cut[i][j];
}
```

Listing 102: Modified file (pair_sph_taitwater.cpp)

```
double PairSPHTaitwater::init_one(int i, int j) {
  if (setflag[i][j] == 0) {
    error->all(FLERR,"All pair sph/taitwater coeffs
     are set");
  }
  cut[j][i] = cut[i][j];
  return cut[i][j];
}
```

7.7. Running the New Software with Mpirun

At this stage, the software is designed to only run in serial. Changes need to be made to make it run with Message Passing Interface (MPI). This will allow the software

to run in parallel: some computations being independent from each other, they can be performed at the same time. Instead of using one processor for a long time, we will use multiple processors for a shorter period. The simulation will therefore take more computing resources but will take a lot shorter to compute. The original SPH module can already be run with MPI however as we have modified the code that is no longer true. We need to make additional changes to the software. All those changes are located in the Atom Vec Meso class of the SPH module.

In LAMMPS, the different MPI processes have to communicate with each other as the computations they perform are not completely independent from each other. They need data from other processes in order to perform their own calculations. They communicate with each other using a buffer that will contain all the necessary data. The buffer is simply an array that we will fill with the data. The different methods for packing and unpacking this buffer are defined in the Atom Vec Meso class. We need to add a new data to transmit: the calculated viscosity.

The first thing to do is to increase the size of the buffers in their initialisation so they can accept the viscosity value, an example is shown in Listings 103 and 104.

Listing 103: Original constructor (atom_vec_meso.cpp)

```
AtomVecMeso::AtomVecMeso(LAMMPS *lmp) : AtomVec(lmp)
{
  molecular = 0;
  mass_type = 1;
  forceclearflag = 1;

// we communicate not only x forward but also vest ..
  comm_x_only = 0; .
// we also communicate de and drho in reverse direction
  comm_f_only = 0;
// 3 + rho + e + vest[3], that means we may
// only communicate 5 in hybrid
  size_forward = 8;
  size_reverse = 5; // 3 + drho + de
  size_border = 12; // 6 + rho + e + vest[3] + cv
  size_velocity = 3;
  size_data_atom = 8;
  size_data_vel = 4;
  xcol_data = 6;

  atom->e_flag = 1;
  atom->rho_flag = 1;
  atom->cv_flag = 1;
  atom->vest_flag = 1;
}
```

Listing 104: Modified constructor (atom_vec_meso.cpp)

```
AtomVecMeso::AtomVecMeso(LAMMPS *lmp) : AtomVec(lmp)
{
  molecular = 0;
  mass_type = 1;
  forceclearflag = 1;

// we communicate not only x forward but also vest ...
  comm_x_only = 0;
// we also communicate de and drho in reverse direction
  comm_f_only = 0;
// 3 + rho + e + vest[3] + viscosities, that means we may
// only communicate 6 in hybrid
  size_forward = 9;
  size_reverse = 5; // 3 + drho + de
// 6 + rho + e + vest[3] + cv + viscosities
  size_border = 13;
  size_velocity = 3;
  size_data_atom = 8;
  size_data_vel = 4;
```

```
  xcol_data = 6;

  atom->e_flag = 1;
  atom->rho_flag = 1;
  atom->cv_flag = 1;
  atom->vest_flag = 1;
}
```

Then, we added the relevant elements of the attribute `viscosities` to the buffer in all the methods handling buffers, an example is shown in Listings 105 and 106.

Listing 105: Original `pack_vec_hybrid` (atom_vec_meso.cpp)

```
int AtomVecMeso::pack_comm_hybrid(int n, int *list,
 double *buf) {
  //printf("in AtomVecMeso::pack_comm_hybrid\n");
  int i, j, m;

  m = 0;
  for (i = 0; i < n; i++) {
    j = list[i];
    buf[m++] = rho[j];
    buf[m++] = e[j];
    buf[m++] = vest[j][0];
    buf[m++] = vest[j][1];
    buf[m++] = vest[j][2];
  }
  return m;
}
```

Listing 106: Modified `pack_vec_hybrid` (atom_vec_meso.cpp)

```
int AtomVecMeso::pack_comm_hybrid(int n, int *list,
 double *buf) {
  //printf("in AtomVecMeso::pack_comm_hybrid\n");
  int i, j, m;

  m = 0;
  for (i = 0; i < n; i++) {
    j = list[i];
    buf[m++] = rho[j];
    buf[m++] = e[j];
    buf[m++] = vest[j][0];
    buf[m++] = vest[j][1];
    buf[m++] = vest[j][2];
    buf[m++] = viscosities[j];
  }
  return m;
}
```

After making those changes for all the methods in the class, the software can be run using mpirun.

7.8. Invoking, Selecting and Computing a Viscosity Object

To compute the new viscosity a new argument was added to the `compute` command: `viscosities`. This allows the user to use the `compute` command to output the dynamic viscosity to the dump file. This can be done by the following command:

```
compute           viscosities_peratom all meso/viscosities/atom
```

The implementation of this feature is simple, as it is very similar to other `compute` argument implementation. All that needs to be done is to modify another `compute`'s implementation, such as `compute_meso_rho_atom` so it processes the variable `viscosities` instead of `rho`.

The viscosity used in the simulation can be invoked in the input file, using the following command:

```
viscosity     [type of viscosity]    [parameters of the viscosity]
```

The type of viscosity can be chosen from the following list:

- FourParameterExp: the four parameter exponential viscosity law.
- SutherlandViscosityLaw: the Sutherland viscosity law.
- PowerLawGas: the power viscosity law for gases.
- Arrhenius: the Arrhenius viscosity law.
- Constant: a constant viscosity.

For example, to invoke the four parameter exponential viscosity, we can write in the input file:

```
viscosity      FourParameterExp C1 C2 C3 C4
```

As stated earlier, this list can easily be extended by the user by modifying the `add_viscosity` function defined earlier.

8. Conclusions

Particle methods are very versatile and can be applied in a variety of applications, ranging from modelling of molecules to the simulation of galaxies. Their power is even amplified when they are coupled together within a discrete multiphysics framework. This versatility matches well with LAMMPS, which is a particle simulator, whose open-source code can be extended with new functionalities. However, modifying LAMMPS can be challenging for researchers with little coding experience and the available support material on how to modify LAMMPS is either too basic or too advanced for the average researcher. Moreover, most of the available material focuses on MD; while the aim of this paper is to support researchers that use other particle methods such as SPH or DEM.

In this work, we present several examples, explained step-by-step and with increasing level of complexity. We begin with simple cases and concluding with more complex ones: Section 3 shows the implementation of the Kelvin–Voigt bond style used to model encapsulate particles with a soft outer shell and validated validated by simulating spherical homogeneous linear elastic and viscoelastic particles [45]; Section 7 show how to implement a new per-atom temperature dependant viscosity property and is validated finding the same viscosity and velocity trend shown by Sameen and Govindarajan [59] in their analytical solution for a channel flow in a asymmetrical heating walls.

The work perfectly fits in the "Discrete Multiphysics: Modelling Complex Systems with Particle Methods" special issue by sharing some in dept know-how and "trick and trades" developed by our group in years of use of LAMMPS. In fact, the aim is to support, in several ways, researchers that use computational particle methods. Often researchers tend to write their own code. The advantage of this approach is that the code is well understood by the researcher and, therefore, easily extendible. However, this sometimes implies reinventing the wheel and countless hours of debugging. Familiarity with a code like LAMMPS, which has an active community of practice and is periodically enriched with new features would be beneficial to this type of researchers allowing them to save considerable time. In the long term, there is another advantage. Modules written for in-house code are hardly sharable. At the moment, the largest portion of the LAMMPS community is dedicated to MD. While this article was under review, for instance, a new book dedicated to modifying LAMMPS came out [60]. However, it focuses only on MD and it does not mention other discrete methods like SPH or DEM. Instead, the aim of this paper is to make LAMMPS more accessible for the Discrete Multiphysics community facilitating sharing reusable code among practitioners in this field.

Supplementary Materials: The codes used in this work are freely available under the GNU General Public License v3 and can be downloaded from the University of Birmingham repository (http://edata.bham.ac.uk/560/).

Author Contributions: Conceptualization, A.A. (Andrea Albano) and A.A. (Alessio Alexiadis); methodology, A.A. (Andrea Albano); validation, A.A. (Andrea Albano), E.l.G., A.D., I.H.S. and A.R.; writing—original draft preparation, A.A. (Andrea Albano), E.l.G., A.D., I.H.S.; writing—review and editing, A.A. (Andrea Albano), A.A. (Alessio Alexiadis), C.A.D.-D., X.S., K.C.N. and M.A.; supervision, A.A. (Alessio Alexiadis), A.R. and I.M.; funding acquisition, A.A. (Alessio Alexiadis). All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the US office of Naval Research Global (ONRG) under NICOP Grant N62909-17-1-2051.

Acknowledgments: The authors would like to thank Prof Albano (University of Pisa) for his advice and comments. The computations described in this paper were performed using the University of Birmingham's BlueBEAR HPC service, which provides a High Performance Computing service to the University's research community. See http://www.birmingham.ac.uk/bear for more details.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MS	Molecular Dynamics
DMP	Discrete MultiPhysics
SPH	Smoothed Particle Hydrodynamics
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator
EOS	Equation Of State
LSM	Lattice Spring Model

Appendix A. An Example of Discrete Multiphysics Simulation in LAMMPS

In this section we present a simple case of DMP simulation with LAMMPS. It is an explanatory example deliberately simple for illustrative purposes. It involves only a small number of particles. Sensitivity analysis of the results with the model resolution or other numerical parameters are beyond the scope of this example and not carried out.

The geometry is a 2D tube with an elastic membrane at one end (Figure A1). The tube contains a liquid simulated with the SPH model, Tait EOS and Morris viscosity. The wall is simulated with stationary particles and the membrane with the LSM using Hookean springs. In Figure A1, the liquid particles are red, the wall particles blue and the membrane particles yellow. During the simulation, the fluid is subjected to a force in the x-direction that pushed the particles against the membrane. Because the membrane is elastic, it stretches inflating the right end of the tube like a balloon. The resolution of the membrane is ten times higher than the fluid. This ensures that, as the membrane stretches, fluid particles do not 'leak' in the gaps formed between two consecutive membrane particles. The Lennard Jones potential, truncated to consider only the repulsive part, is used to avoid compenetration between solid and liquid particles. A weaker Lennard Jones potential is used as 'artificial pressure' to avoid excessive compression of the fluid particles.

The initial data file (data.initial) for the geometry was create according to LAMMPS' rules for formatting the Data File [41] and is shared as additional material. In Data File, the fluid particles are called `type 1`, the wall particles `type 2` and the membrane particles `type 3`. Here we focus on the input file (membrane.lmp), which is also shared in its entirety as additional material. We do not discuss LAMMPS syntax (the reader can refer to LAMMPS User's Guide for this [41]), but only on specific parts of the input file that concern the DMP implementation.

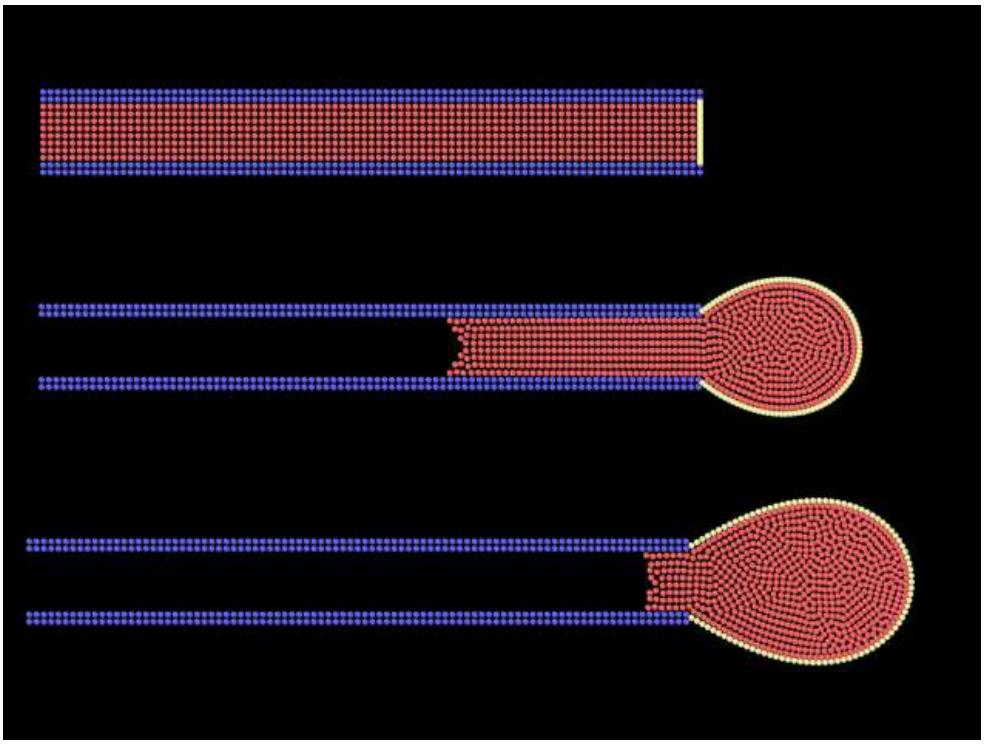


Figure A1. The inflating balloon simulation.

The first section of the input file determines the dimensionality of the problem (2D), the boundary conditions (periodic), the units used (SI), the type of potential used in the simulation (`atom_style`) and the input file that contains the initial position of all the particles

```
dimension         2
boundary       p p p
units             si
atom_style  hybrid meso bond angle
read_data      data.initial
```

The crucial line for DMP simulations is the `hybrid` keyword of the `atom_style`, which allows for combining different particle models. The keyword `meso` refers to the SPH model and `bond`, in the case under consideration, to the LSM. The `angle` keyword corresponds to angular springs, but, as it will be clear later, it is not used in this simulation.

The following section contains several variables that are going to be used later on. In particular, the initial particle distance is `dL` and their mass `m`. The resolution of the membrane is `Nt` times higher than the fluid. The initial distance between membrane particle is therefore `db=dL/Nt` and their mass `mM=m/Nt`.

```
variable    dL       equal  0.000111111
variable    m     equal  1.23457e-05
variable    Nt       equal 10
variable    dB       equal ${dL}/${Nt}
variable    mM       equal  ${m}/${Nt}
variable    h     equal 1.5*${dL}
variable    h2       equal ${dL}/${Nt}
variable    c     equal 0.1
variable    mu       equal 1.0e-3
variable    rho      equal 1000
variable    kA       equal 1.e-8
variable    kB       equal 100
variable    skin     equal 0.3*${h}
variable    epsL     equal 1.e-12
variable    epsS     equal 1.e-10
variable    sgmL  equal ${dL}
```

```
variable    sgmS  equal 0.5*${sgmL}/${Nt}
variable    fmax     equal 0.00005
variable    ft    equal ramp(0.,${fmax})
```

The section below identifies particles `type 1` as a group called `fluid`, particles `type 2` as a group called `wall` and particles `type 3` as a group called `membrane`. The mass of `type 3` particles is assigned (the mass of `type 1, 2` was assigned in the data.initial file). The density of all particle is also assigned based on the value `rho` defined previously.

```
group           fluid      type 1
group           wall    type 2
group           membrane   type 3
mass 3 ${mM}
set group all meso/rho ${rho}
```

The next section defines the pair potentials for non-bonded particles. In this simulation, we use different styles together (keyword `hybrid/overlay`). The `sph/taitwater/morris` pair style, which is used for all pair interactions except 2-2 (i.e., wall particles with themselves); and the Lennard Jones potential `lj/cut`, which, as explained above, is used both as 'artificial pressure' and to avoid compenetration of solid and fluid particles.

```
pair_style hybrid/overlay sph/taitwater/morris lj/cut ${sgmL}
pair_coeff 1 * sph/taitwater/morris ${rho} ${c} ${mu} ${h}
pair_coeff 2 3 sph/taitwater/morris ${rho} ${c} ${mu} ${h2}
pair_coeff 3 3 sph/taitwater/morris ${rho} ${c} ${mu} ${h2}

pair_coeff 1 * lj/cut ${epsL} ${sgmL}
pair_coeff 2 * lj/cut ${epsL} ${sgmL}
pair_coeff 1 3 lj/cut ${epsS} ${sgmL}
pair_coeff 3 3 lj/cut ${epsS} ${sgmS}
```

After the non-bonded potentials, the script assigns the `harmonic` potential, with Hook constant `kB` and equilibrium distance `dB`, to the bonded particles (i.e., the membrane). All pairs of bonded particles are assigned in the data.initial file.

```
bond_style  harmonic
bond_coeff  1 ${kB} ${dB}
angle_style none
```

The next section assigns several parameters that determine how the Newton equation of motion is solved numerically. The force `fmax` is added to all fluid particle in the x-direction, and an artificial viscosity is added for stability reasons.

```
fix 2 fluid addforce ${fmax} 0.0 0.0
fix 5 fluid meso
fix 6 membrane meso
fix 8 wall meso/stationary
fix 9 all viscous 0.01
```

The last commands determine the value and the number of timesteps used in the simulation plus a variety of computations for output and other purposes that are not discussed here (the reader can refer to the User's Guide).

```
compute rho_peratom all meso/rho/atom
compute rho_ave all reduce ave c_rho_peratom
compute vmax fluid reduce max vx
thermo             10000
thermo_style custom step c_rho_ave c_vmax
thermo_modify      norm no
neighbor       ${skin} bin
dump   dump_id all custom 10000 dump.lammpstrj id type x y z vx vy
timestep           1.e-6
run                2500000
```

Appendix B. How to Compile LAMMPS

LAMMPS is build as a library and executable [41] either by using GNU make [61] or a build environment with CMake [62]. In this appendix LAMMPS will be compiled only using make and it is compiled in BlueBEAR. For more details of the compiling process in LAMMPS refer to the user manual [41].

To compile LAMMPS in your own directory you can follow those steps

1. Download the file from here. Select the code you want, click the "Download Now" button, and your browser should download a gzipped tar file. Save the file in your directory on BlueBEAR
2. Unpack the file with the following command line command prompt:

Listing A1: Command to open the tar file on BlueBEAR

```
tar -xvf lammps-stable.tar.gz
```

3. Before compiling is important to set up the environment, with BlueBEAR

Listing A2: Commands to set the environment for compile LAMMPS on BlueBEAR

```
module purge

module load bluebear

module load Eigen/3.3.4-foss-2019a
```

4. Enter in the /src directory in your new LAMMPS directory. The src directory directory contains the C++ source and header files for LAMMPS. It also contains a top-level Makefile and a MAKE sub-directory with low-level Makefile.* files for many systems and machines.
5. Type the following command to compile a serial version of LAMMPS:

Listing A3: Command to compile LAMMPS on BlueBEAR

```
make serial
```

or a multi-threaded (parallel) version of LAMMPS:

Listing A4: Command to compile LAMMPS on BlueBEAR

```
make mpi
```

If you get no errors and an executable file lmp_mpi is produced.

6. Depending on the features you need, you will have to install same packages in your compiled LAMMPS. Is possible to check which packages is installed in your compiled LAMMPS by typing

Listing A5: Command to check the list of installed packages (you must be inside the /src directory)

```
make ps
```

It is possible to install the packages you need with the command line

Listing A6: Command to install a specific package

```
make yes-NAMEPACK
```

or un-install them with

Listing A7: Command to un-install a specific package

```
make no-NAMEPACK
```

More `make` commands are explained in LAMMPS user manual [41]. After the installation of the desired packages you need to compile it again (step 5).

References

1. Plimpton, S. *Fast Parallel Algorithms for Short-Range Molecular Dynamics*; Technical Report; Sandia National Labs.: Albuquerque, NM, USA, 1993.
2. Plimpton, S.; Pollock, R.; Stevens, M. Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. In Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN, USA, 14–17 March 1997.
3. Auhl, R.; Everaers, R.; Grest, G.S.; Kremer, K.; Plimpton, S.J. Equilibration of long chain polymer melts in computer simulations. *J. Chem. Phys.* **2003**, *119*, 12718–12728.
4. Parks, M.L.; Lehoucq, R.B.; Plimpton, S.J.; Silling, S.A. Implementing peridynamics within a molecular dynamics code. *Comput. Phys. Commun.* **2008**, *179*, 777–783.
5. Petersen, M.K.; Lechman, J.B.; Plimpton, S.J.; Grest, G.S.; Veld, P.J.; Schunk, P. Mesoscale hydrodynamics via stochastic rotation dynamics: Comparison with Lennard-Jones fluid. *J. Chem. Phys.* **2010**, *132*, 174106.
6. Ganzenmüller, G.C.; Steinhauser, M.O.; Van Liedekerke, P.; Leuven, K.U. The implementation of Smooth Particle Hydrodynamics in LAMMPS. *Paul Van Liedekerke Kathol. Univ. Leuven* **2011**, *1*, 1–26.
7. Jaramillo-Botero, A.; Su, J.; Qi, A.; Goddard III, W.A. Large-scale, long-term nonadiabatic electron molecular dynamics for describing material properties and phenomena in extreme environments. *J. Comput. Chem.* **2011**, *32*, 497–512.
8. Coleman, S.; Spearot, D.; Capolungo, L. Virtual diffraction analysis of Ni [0 1 0] symmetric tilt grain boundaries. *Model. Simul. Mater. Sci. Eng.* **2013**, *21*, 055020.
9. Singraber, A.; Behler, J.; Dellago, C. Library-based LAMMPS implementation of high-dimensional neural network potentials. *J. Chem. Theory Comput.* **2019**, *15*, 1827–1840.
10. Ng, K.; Alexiadis, A.; Chen, H.; Sheu, T. A coupled Smoothed Particle Hydrodynamics-Volume Compensated Particle Method (SPH-VCPM) for Fluid Structure Interaction (FSI) modelling. *Ocean Eng.* **2020**, *218*, 107923.
11. Daraio, D.; Villoria, J.; Ingram, A.; Alexiadis, A.; Stitt, E.H.; Munnoch, A.L.; Marigo, M. Using Discrete Element method (DEM) simulations to reveal the differences in the γ-Al2O3 to α-Al2O3 mechanically induced phase transformation between a planetary ball mill and an attritor mill. *Miner. Eng.* **2020**, *155*, 106374.
12. Qiao, G.; Lasfargues, M.; Alexiadis, A.; Ding, Y. Simulation and experimental study of the specific heat capacity of molten salt based nanofluids. *Appl. Therm. Eng.* **2017**, *111*, 1517–1522.
13. Qiao, G.; Alexiadis, A.; Ding, Y. Simulation study of anomalous thermal properties of molten nitrate salt. *Powder Technol.* **2017**, *314*, 660–664.
14. Anagnostopoulos, A.; Navarro, H.; Alexiadis, A.; Ding, Y. Wettability of $NaNO_3$ and KNO_3 on MgO and Carbon Surfaces—Understanding the Substrate and the Length Scale Effects. *J. Phys. Chem. C* **2020**, *124*, 8140–8152.
15. Sahputra, I.H.; Alexiadis, A.; Adams, M.J. Effects of Moisture on the Mechanical Properties of Microcrystalline Cellulose and the Mobility of the Water Molecules as Studied by the Hybrid Molecular Mechanics–Molecular Dynamics Simulation Method. *J. Polym. Sci. Part B Polym. Phys.* **2019**, *57*, 454–464.
16. Sahputra, I.H.; Alexiadis, A.; Adams, M.J. Temperature dependence of the Young's modulus of polymers calculated using a hybrid molecular mechanics–molecular dynamics method. *J. Phys. Condens. Matter* **2018**, *30*, 355901.
17. Mohammed, A.M.; Ariane, M.; Alexiadis, A. Using Discrete Multiphysics Modelling to Assess the Effect of Calcification on Hemodynamic and Mechanical Deformation of Aortic Valve. *ChemEngineering* **2020**, *4*, 48.
18. Ariane, M.; Vigolo, D.; Brill, A.; Nash, F.; Barigou, M.; Alexiadis, A. Using Discrete Multi-Physics for studying the dynamics of emboli in flexible venous valves. *Comput. Fluids* **2018**, *166*, 57–63.
19. Ariane, M.; Wen, W.; Vigolo, D.; Brill, A.; Nash, F.; Barigou, M.; Alexiadis, A. Modelling and simulation of flow and agglomeration in deep veins valves using discrete multi physics. *Comput. Biol. Med.* **2017**, *89*, 96–103.
20. Ariane, M.; Allouche, M.H.; Bussone, M.; Giacosa, F.; Bernard, F.; Barigou, M.; Alexiadis, A. Discrete multi-physics: A mesh-free model of blood flow in flexible biological valve including solid aggregate formation. *PLoS ONE* **2017**, *12*, e0174795.
21. Schütt, M.; Stamatopoulos, K.; Simmons, M.; Batchelor, H.; Alexiadis, A. Modelling and simulation of the hydrodynamics and mixing profiles in the human proximal colon using Discrete Multiphysics. *Comput. Biol. Med.* **2020**, *121*, 103819.
22. Alexiadis, A.; Stamatopoulos, K.; Wen, W.; Batchelor, H.; Bakalis, S.; Barigou, M.; Simmons, M. Using discrete multi-physics for detailed exploration of hydrodynamics in an in vitro colon system. *Comput. Biol. Med.* **2017**, *81*, 188–198.
23. Ariane, M.; Kassinos, S.; Velaga, S.; Alexiadis, A. Discrete multi-physics simulations of diffusive and convective mass transfer in boundary layers containing motile cilia in lungs. *Comput. Biol. Med.* **2018**, *95*, 34–42.
24. Ariane, M.; Sommerfeld, M.; Alexiadis, A. Wall collision and drug-carrier detachment in dry powder inhalers: Using DEM to devise a sub-scale model for CFD calculations. *Powder Technol.* **2018**, *334*, 65–75.
25. Albano, A.; Alexiadis, A. Interaction of Shock Waves with Discrete Gas Inhomogeneities: A Smoothed Particle Hydrodynamics Approach. *Appl. Sci.* **2019**, *9*, 5435.
26. Albano, A.; Alexiadis, A. A smoothed particle hydrodynamics study of the collapse for a cylindrical cavity. *PLoS ONE* **2020**, *15*, e0239830.
27. Albano, A.; Alexiadis, A. Non-Symmetrical Collapse of an Empty Cylindrical Cavity Studied with Smoothed Particle Hydrodynamics. *Appl. Sci.* **2021**, *11*, 3500.
28. Alexiadis, A. The discrete multi-hybrid system for the simulation of solid-liquid flows. *PLoS ONE* **2015**, *10*, e0124678.

29. Alexiadis, A. A new framework for modelling the dynamics and the breakage of capsules, vesicles and cells in fluid flow. *Procedia IUTAM* **2015**, *16*, 80–88.
30. Alexiadis, A. A smoothed particle hydrodynamics and coarse-grained molecular dynamics hybrid technique for modelling elastic particles and breakable capsules under various flow conditions. *Int. J. Numer. Methods Eng.* **2014**, *100*, 713–719.
31. Rahmat, A.; Barigou, M.; Alexiadis, A. Deformation and rupture of compound cells under shear: A discrete multiphysics study. *Phys. Fluids* **2019**, *31*, 051903.
32. Alexiadis, A.; Ghraybeh, S.; Qiao, G. Natural convection and solidification of phase-change materials in circular pipes: A SPH approach. *Comput. Mater. Sci.* **2018**, *150*, 475–483.
33. Rahmat, A.; Barigou, M.; Alexiadis, A. Numerical simulation of dissolution of solid particles in fluid flow using the SPH method. *Int. J. Numer. Methods Heat Fluid Flow* **2019**, *30*, 290–307.
34. Rahmat, A.; Meng, J.; Emerson, D.; Wu, C.Y.; Barigou, M.; Alexiadis, A. A practical approach for extracting mechanical properties of microcapsules using a hybrid numerical model. *Microfluid. Nanofluidics* **2021**, *25*, 1–17.
35. Ruiz-Riancho, I.N.; Alexiadis, A.; Zhang, Z.; Hernandez, A.G. A Discrete Multi-Physics Model to Simulate Fluid Structure Interaction and Breakage of Capsules Filled with Liquid under Coaxial Load. *Processes* **2021**, *9*, 354.
36. Sanfilippo, D.; Ghiassi, B.; Alexiadis, A.; Hernandez, A.G. Combined Peridynamics and Discrete Multiphysics to Study the Effects of Air Voids and Freeze-Thaw on the Mechanical Properties of Asphalt. *Materials* **2021**, *14*, 1579.
37. Alexiadis, A.; Albano, A.; Rahmat, A.; Yildiz, M.; Kefal, A.; Ozbulut, M.; Bakirci, N.; Garzón-Alvarado, D.; Duque-Daza, C.; Eslava-Schmalbach, J. Simulation of pandemics in real cities: Enhanced and accurate digital laboratories. *Proc. R. Soc. A* **2021**, *477*, 20200653.
38. Alexiadis, A. Deep Multiphysics and Particle–Neuron Duality: A Computational Framework Coupling (Discrete) Multiphysics and Deep Learning. *Appl. Sci.* **2019**, *9*, 5369.
39. Alexiadis, A. Deep multiphysics: Coupling discrete multiphysics with machine learning to attain self-learning in-silico models replicating human physiology. *Artif. Intell. Med.* **2019**, *98*, 27–34.
40. Alexiadis, A.; Simmons, M.; Stamatopoulos, K.; Batchelor, H.; Moulitsas, I. The duality between particle methods and artificial neural networks. *Sci. Rep.* **2020**, *10*, 1–7.
41. Sandia Corporation LAMMPS Users Manual. 2003. Available online: https://lammps.sandia.gov/doc/Developer.pdf (accessed on 11 Jauaury 2021).
42. Plimpton. LAMMPS Developer Guide. Available online: https://lammps.sandia.gov/doc/Developer.pdf (accessed on 11 Jauaury 2021).
43. Plimpton, S.J. *Modifying & Extending LAMMPS*; Technical Report; Sandia National Lab. (SNL-NM): Albuquerque, NM, USA, 2014.
44. Flügge, W. *Viscoelasticity*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2013.
45. Sahputra, I.H.; Alexiadis, A.; Adams, M.J. A Coarse Grained Model for Viscoelastic Solids in Discrete Multiphysics Simulations. *ChemEngineering* **2020**, *4*, 30.
46. Liu, M.; Liu, G. Smoothed particle hydrodynamics (SPH): An overview and recent developments. *Arch. Comput. Methods Eng.* **2010**, *17*, 25–76.
47. Le Métayer, O.; Saurel, R. The Noble-Abel stiffened-gas equation of state. *Phys. Fluids* **2016**, *28*, 046102.
48. Monaghan, J.J.; Gingold, R.A. Shock simulation by the particle method SPH. *J. Comput. Phys.* **1983**, *52*, 374–389.
49. Lattanzio, J.; Monaghan, J.; Pongracic, H.; Schwarz, M. Controlling penetration. *SIAM J. Sci. Stat. Comput.* **1986**, *7*, 591–598.
50. Morris, J.P.; Fox, P.J.; Zhu, Y. Modeling low Reynolds number incompressible flows using SPH. *J. Comput. Phys.* **1997**, *136*, 214–226.
51. Cornelissen, J.; Waterman, H. The viscosity temperature relationship of liquids. *Chem. Eng. Sci.* **1955**, *4*, 238–246.
52. Seeton, C.J. Viscosity–temperature correlation for liquids. *Tribol. Lett.* **2006**, *22*, 67–78.
53. Stanciu, I. A new viscosity-temperature relationship for vegetable oil. *J. Pet. Technol. Altern. Fuels* **2012**, *3*, 19–23.
54. Gutmann, F.; Simmons, L. The temperature dependence of the viscosity of liquids. *J. Appl. Phys.* **1952**, *23*, 977–978.
55. De Guzman, J. Relation between fluidity and heat of fusion. *Anales Soc. Espan. Fis. Quim* **1913**, *11*, 353–362.
56. Raman, C. A theory of the viscosity of liquids. *Nature* **1923**, *111*, 532–533.
57. Chapman, S.; Cowling, T.G.; Burnett, D. *The Mathematical Theory of Non-Uniform Gases: An Account of the Kinetic Theory of Viscosity, Thermal Conduction and Diffusion in Gases*; Cambridge University Press: Cambridge, UK, 1990.
58. Rathakrishnan, E. *Theoretical Aerodynamics*; John Wiley & Sons: Hoboken, NJ, USA, 2013.
59. Sameen, A.; Govindarajan, R. The effect of wall heating on instability of channel flow. *J. Fluid Mech.* **2007**, *577*, 417–442.
60. Mubin, S.; Li, J. *Extending and Modifying LAMMPS*; Packt Publishing Ltd.: Birmingham, UK, 2021.
61. Project, G. Make-GNU Project-Free Software Fondation. Available online: https://www.gnu.org/software/make/ (accessed on 14 October 2020).
62. Martin, K.; Hoffman, B. *Mastering CMake: A Cross-Platform Build System*; Kitware: Clifton Park, NY, USA, 2010.

Chapter 5

Interaction of shock waves with discrete gas inhomogeneities: an Smoothed Particle Hydrodynamics approach

In this Chapter a Smoothed Particle Hydrodynamics model is developed for simulating the interaction between a cylindrical gas inhomogeneities and a travelling shock wave inside a shock tube. As explained in Chapter 2, the physics of this phenomenon shares many similarities with the shock-induced collapse mechanism. For this reason, it is used to test and validate the approach used in Chapter 6, 7, and 8.

This Chapter has been published in *Applied Sciences* as:

Albano A, Alexiadis A. Interaction of Shock Waves with Discrete Gas Inhomogeneities: A Smoothed Particle Hydrodynamics Approach. Applied Sciences. 2019; 9(24):5435

My contributions in this work were: Designed the work and performed the simulations, Methodology, Validation, Writing the original draft, Reviewing and Editing.

I would like to thank all the authors who have contributed to this work.

applied sciences

MDPI

Article

Interaction of Shock Waves with Discrete Gas Inhomogeneities: A Smoothed Particle Hydrodynamics Approach

Andrea Albano *,† **and Alessio Alexiadis** †

School of Chemical Engineering, University of Birmingham, Birmingham B15 2TT, UK; Alexiadis@bham.ac.uk

* Correspondence: AXA1220@student.bham.ac.uk

† These authors contributed equally to this work.

Received: 18 November 2019; Accepted: 5 December 2019; Published: 11 December 2019

Abstract: In this study, we propose a smoothed particle hydrodynamics model for simulating a shock wave interacting with cylindrical gas inhomogeneities inside a shock tube. When the gas inhomogeneity interacts with the shock wave, it assumes different shapes depending on the difference in densities between the gas inhomogeneity and the external gas. The model uses a piecewise smoothing length approach and is validated by comparing the results obtained with experimental and CFD data available in the literature. In all the cases considered, the evolution of the inhomogeneity is similar to the experimental shadowgraphs and is at least as accurate as the CFD results in terms of timescale and shape of the gas inhomogeneity.

Keywords: particle method; smoothed particle hydrodynamics; modelling; simulations; shock wave

1. Introduction

In the last 30 years, the study of a planar shock wave interacting with an isolated, gas inhomogeneity has been investigated both experimentally (e.g., [1–5]) and numerically (e.g., [4,6–8]). Nowadays, this system has acquired importance for computational models up to the point of becoming a benchmark for validating shock-induced flows [9].

A gas inhomogeneity is created in a tube (known as shock tube) filled with gas by slowly introducing a different type of gas. The shock tube is generally a tube with either a rectangular or circular cross section; the shock wave can be generated either from an explosion (blast-driven) or due to high-pressure differences between two gasses separated by a diaphragm (compressed gas-driven). When the diaphragm breaks out, a shock wave is generated and propagates through the gas at lower pressure. In Figure 1, the lower pressure gas is called driven gas and the high-pressure gas that generates the shock wave driver gas. Analogously, the section of the tube where the driver gas is confined is called driver section while the section of the driven gas is called driven section. As a result of the shock wave, a net flow, in the direction of the shock wave but with lower speed, is generated. In rectangular shock tubes, mixing between the two gasses is initially avoided by injecting the inhomogeneity in a nitrocellulose membrane (cylindrical inhomogeneity). In this way the inhomogeneity remains "cylindrical" during its evolution (Figure 2). When the shock wave reaches the inhomogeneity, the inhomogeneity deforms in a way that depends on the density difference between the inhomogeneity and the driven gas. The different shapes that the inhomogeneity assumes during the passage of the shock wave are typically used for validation of numerical codes (e.g., [6,10]).

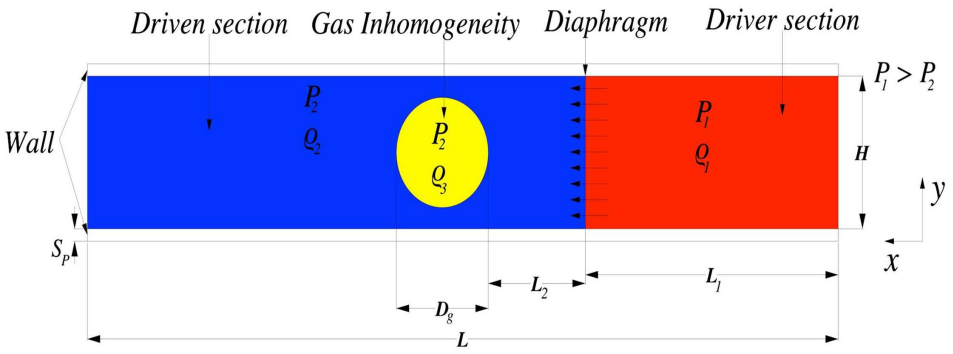


Figure 1. Geometry of the simulation box.

In this work, we use smoothed particle hydrodynamics (SPH) to simulate the shock wave and the inhomogeneity interaction. The different shapes of the inhomogeneity calculated during the simulation are compared with experimental data available in the literature for assessing the precision of the model.

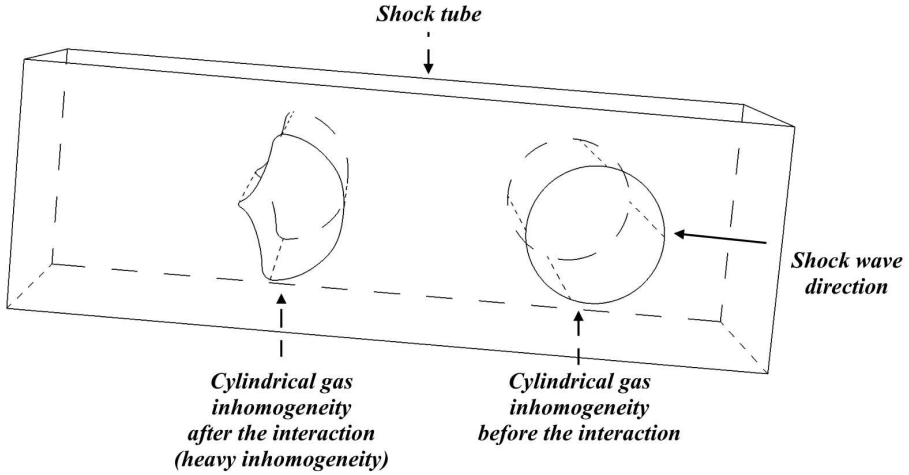


Figure 2. Geometry of the shock tube with the cylindrical inhomogeneity.

2. Smoothed Particle Hydrodynamics

Smoothed particle hydrodynamics is a meshfree computational method initially developed by Gingold and Monaghan [11] and Lucy [12] for solving astrophysical problems. Later it was used to solve fluidynamics problems to overcome some of the limitations of the grid-based method in the case, for instance, of explosions and high velocity impact phenomena ([13,14]). The method was also widely validated against shock waves in particular for the well known Riemann problem ([15–17]). The SPH approximation is based on the so-called integral representation of a function. Given the function $f(\mathbf{r})$, defined in a volume V, function of the three-dimensional position $\mathbf{r}$, is defined by the identity

$$f(\mathbf{r}) = \iiint f(\mathbf{r}')\delta(\mathbf{r} - \mathbf{r}')d\mathbf{r}', \tag{1}$$

where $\delta(\mathbf{r} - \mathbf{r}')$ is the Dirac delta function defined as

$$\delta(\mathbf{r}-\mathbf{r}') = \begin{cases} +\infty & \mathbf{r} = \mathbf{r}' \\ 0 & \mathbf{r} \neq \mathbf{r}'. \end{cases} \tag{2}$$

It is possible to approximate the integral representation by replacing the Dirac delta function with a bell-shaped function called smoothing function or kernel, W, which depends on the position $\mathbf{r}$ and on the so-called smoothing length, h. When h approaches zero the kernel function W has the property

$$\lim_{h\to 0} = W(\mathbf{r}-\mathbf{r}', h) = \delta(\mathbf{r}-\mathbf{r}'), \tag{3}$$

which approximates the integral representation, Equation (1), to the so-called kernel approximation:

$$f(\mathbf{r}) \approx \iiint f(\mathbf{r}')W(\mathbf{r}-\mathbf{r}', h)d\mathbf{r}'. \tag{4}$$

In this work, the kernel used is Lucy kernel function

$$W(R,h) = \begin{cases} \frac{1}{s}\left[1+3\frac{R}{h}\right]\left[1-\frac{R}{h}\right]^3 & R \leq 1 \\ 0 & R > 1, \end{cases} \tag{5}$$

where $R = |\mathbf{r}-\mathbf{r}'|$ and s is a parameter used to normalise the kernel function, which, for one, two and three dimensional space is, respectively, $\frac{4h}{5}$, $\frac{\pi h^2}{5}$ and $\frac{16\pi h^3}{105}$. The last step is to approximate the infinitesimal volume $d\mathbf{r}'$ to a finite volume dr composed by computational particles with their own mass $m = \rho dr$. With this approximation it is possible to discretise Equation (4)

$$f(\mathbf{r}) \approx \sum \frac{m_i}{\rho_i} f(\mathbf{r}_i)W(\mathbf{r}-\mathbf{r_i}, h), \tag{6}$$

where m_i, ρ_i and $\mathbf{r}_i$ are mass, density and position of the ith particle. Only particles for which $|\mathbf{r}-\mathbf{r_i}| < h$ are taken into account in the summation. With Equation (6) it is possible to discretise any set of equations such as the energy balance or the Navier–Stokes equation on an arbitrarily set of computational particles. Within the SPH framework, for instance, the momentum–conservation equation can be rewritten as

$$m_i\frac{d\mathbf{v}_i}{dt} = \sum_j m_i m_j \left(\frac{P_i}{\rho_i} + \frac{P_i}{\rho_i} + \Pi_{ij}\right)\nabla_j W_{ij}, \tag{7}$$

where $W_{ij} = W(r_j - r_i, h)$ and $\nabla_j W_{ij}$ is the kernel gradient in the r_j direction. P is the pressure while Π_{ij} is the so-called artificial viscosity introduced by Monaghan [18] for simulating shock waves

$$\Pi_{ij} = -\alpha h \frac{c_i + c_j}{\rho_i + \rho_j}\frac{\mathbf{v}_{ij}\cdot\mathbf{r}_{ij}}{r_{ij}^2 + \epsilon h^2}, \tag{8}$$

where c_i and c_j are speed of sound of particles i and j and α is dimensionless parameter that controls the strength of the viscous dissipation and $\epsilon \approx 0.01$ is used to avoid singularities when particles are close to each other. The parameter α can be linked to the kinematic viscosity by means of

$$\nu = \frac{\alpha h c}{8}. \tag{9}$$

During the simulation, Equation (7) updates, at every time step, the velocities of the Lagrangian particles; the density is updated by the continuity equation in discrete form

$$\frac{d\rho_i}{dt} = \sum_j m_j \mathbf{v}_{ij} \nabla_j W_{ij}, \tag{10}$$

where $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$. Equation (10) requires a closure term relating ρ and P. In this work, the ideal gas equation of state is used

$$P(\rho, e) = (\gamma - 1)\rho e, \tag{11}$$

where $\gamma = \frac{C_p}{C_v}$ is the capacity heat ratio and e is the specific internal energy.

3. Dimensional Groups

The interaction between the shock wave and the gas inhomogeneity depends on the physical properties of the driven and inhomogeneity gasses and on the shock wave speed, which can be represented as dimensionless groups. In this section, we define the dimensionless groups used in this study.

3.1. Atwood Number

The Atwood number is defined as:

$$\mathrm{A} = \frac{\rho_3 - \rho_2}{\rho_3 + \rho_2}, \tag{12}$$

where ρ_2 and ρ_3 are the density of the gas inhomogeneity and air respectively (Figure 1). The Atwood number expresses the interaction between the gas inhomogeneity and the planar incident shock wave. When $A < 0$, we have a "light inhomogeneity", where the inhomogeneity density is lower than that of the driven gas. When $A > 0$, on the contrary, we have a "heavy inhomogeneity", where the inhomogeneity density is higher than that of the driven gas.

3.2. Pressure Ratio

The pressure ratio is defined as

$$\mathrm{P_r} = \frac{P_1}{P_2} \tag{13}$$

and represents the pressure ratio between the driver gas, P_1, and the driven gas, P_2.

3.3. Mach Number

An important factor that influences the dynamics in the system is the speed with which the shock wave propagates in the driven gas. The shock wave propagates with a speed greater than the sound speed in the fluid expressed as dimensionless Mach number

$$\mathrm{Ma} = \frac{U_s}{c_2}, \tag{14}$$

where U_s is the speed of the shock wave and c_2 is the speed of sound of the driven gas.

3.4. Dimensionless Time

The last dimensionless number used in this work is the dimensionless time τ, defined as

$$\tau = \frac{t}{\tau_0}, \tag{15}$$

where t is the time of the simulation and τ_0 is the time required for the shock wave to pass through the gas inhomogeneity.

4. Shape Analysis

In this section, we look at how, according to the literature, the shape of the inhomogeneity changes with the Atwood number.

4.1. Standard Shapes for Light Inhomogeneity ($A < 0$)

Due to the higher sound speed in the gas inhomogeneity, the shock wave finds less resistance and thus moves faster than in the driven gas. At first, the gas inhomogeneity flattens in the direction of the shock wave (x-direction according to Figure 1) and expands in the y-direction gaining a Semi-prolate shape (Figure 3a). Later a re-entrant jet forms at the centre of the inhomogeneity (crescent moon shape, Figure 3b) and, as it grows, the inhomogeneity changes shapes first to semi claw shape (Figure 3c) and finally to claw shape (Figure 3d).

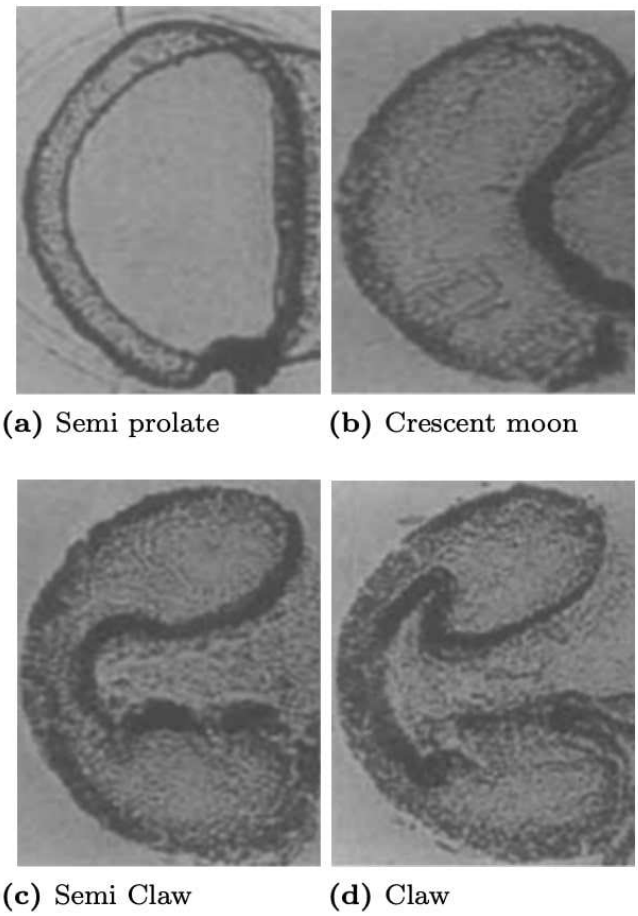


Figure 3. Standard gas inhomogeneity shapes definition using the experimental shadowgraphs [1] for Atwood number: $A = -0.79$. Times: (**a**) 102 µs (**b**) 245 µs (**c**) 427 µs (**d**) 674 µs.

4.2. Standard Shapes for Heavy Inhomogeneity ($A > 0$)

When $A > 0$, the speed of sound in the gas inhomogeneity is slower than in the driven gas leading to completely different shapes. Initially the compression effect is predominant and the gas inhomogeneity tends to flatten (flatfish shape, Figure 4a) followed by a crescent shape (jellyfish head shape, Figure 4b). Later, the passage of the shock wave front causes the formations of filaments at the top and bottom of the inhomogeneity (Jellyfish shape, Figure 4c).

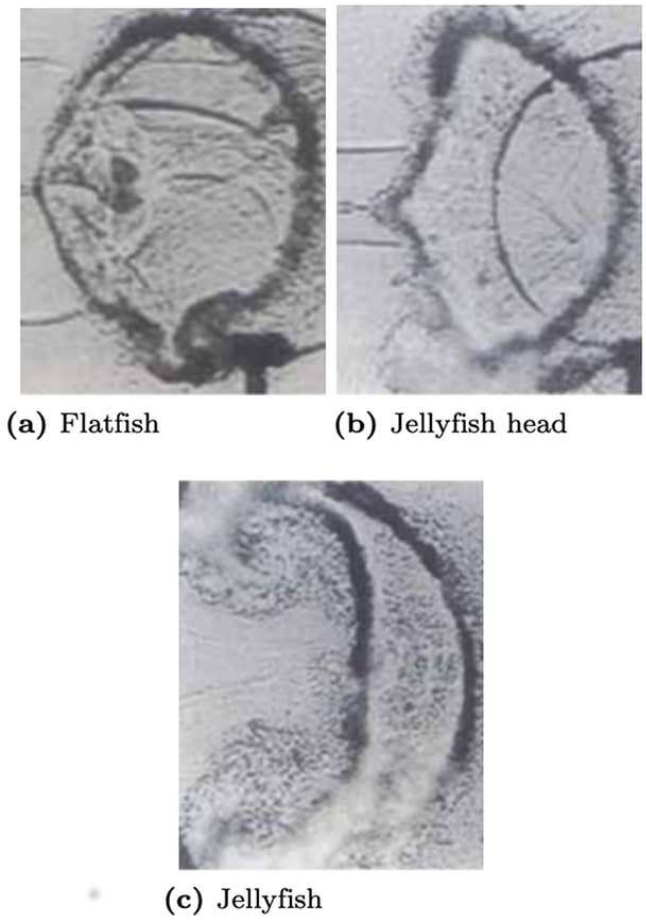


Figure 4. Standard gas inhomogeneity shapes definition using the experimental shadowgraphs [1] for Atwood number: $A = 0.51$. Times: (**a**) 247 μs (**b**) 417 μs (**c**) 1020 μs.

5. Model

5.1. Geometry

A simplified 2D replica of a shock tube was developed and is shown in Figure 1. The dimensions of the shock tube and of the gas inhomogeneity were chosen to match the experimental set-up by Haas & Sturtevant [1], who used a rectangular cross section shock tube and a cylindrical gas inhomogeneity, with (referring to Figure 1) $D_g = 0.50$ m, $H = 0.89$ m, $L_1 = 0.50$ m, $L_2 = 0.0225$ m, $L = 1.00$ m and $s_p = 0.00445$ m. In the simulation four types of SPH particles are accounted for: type 1 particles are in the driver gas group, type 2 in the driven gas group, type 3 in the gas inhomogeneity group and type 4 in the wall group. The wall interacts with the fluid with a repulsive force by using the Lennard-Jones potential to avoid compenetration between the fluid and the walls. Along the x-axis the boundaries are set as shrink-wrapping. Shrink-wrapping (SW) boundaries are non-periodic boundaries, where the edge of the simulation box moves with the expanding atoms to make sure that all the particles remain within the computational domain [17]. Because in our simulations we only model a section of the tube, SW is used to make sure the shock wave is not reflected when it reaches the boundary. Preliminary simulations were carried out with different resolutions (e.g., total number of particles N = 175,050, 280,450, 565,577, 750,100). The value of N = 375,050 was chosen as the best compromise between accuracy and computational speed.

5.2. Shock Wave Generation

In the experiments ([1,3,6,19]), the pressure ratio P_r used to generate the shock wave is not specified. For this reason, we initially run several simulations with the goal to determine which P_r brings to a shock wave with Ma = 1.22, as in Haas [1] and Quirk [6].

Firstly, we used a standard single smoothing length approach, where h is greater than the initial particles spacing dL (various solutions with h between 1.05 dL and 1.2 dL were investigated). In this way, however, the speed of the shock wave is always higher than the experimental one. This seems to be a recurring issue in SPH simulations and often the actual speed of the shock wave is not well addressed in the SPH literature. To address a similar issue, SPH simulations of explosions, where the Mach number can reach values of 8.5, often adopt a variable smoothing length changing with the density of the particles in the domain (e.g., [20,21]). In this study, however, considering that the Mach number is lower, a simpler approach based on a piecewise constant smoothing length is adopted here. To achieve correct Mach numbers in our simulations, the first smoothing length, $h_1 >$ dL, is used to simulate the interaction of the particles of the driver section and at the interface between particles of type 1 and particles of type 2. The second smoothing length, $h_2 <$ dL, is used for the particles in the driven section. From a physical point of view, $h_2 <$ dL reflects the fact that the speed of sound is the speed at which a perturbation can move in a fluid. Therefore, the computational particles in the driven section should not "feel" the presence of the shock wave before the passage of the front. From the theoretical point of view, the relation between h_2 and Ma deserves more investigation. The issue, however, is beyond the scope of the present study; here we simply identified the value of h_2 that brings to the correct Mach number; further analysis is left for future work. The correct speed of the shock wave (i.e., Ma = 1.22) was achieved with $P_r = 20$, $h_1 = 1.15$ dL and $h_2 = 0.5$ dL. The dissipation factor was chosen to be $\alpha = 0.1$ as in Morris & Monaghan [22].

6. Result and Discussion

In the literature, comparison between numerical and experimental data is usually done by comparing the shapes of the inohomogenity at different Atwood numbers. This study follows the same approach and the model is assessed by comparing our simulations with the experimental shadowgraphs reported by Haas [1]. Additionally, we also validate our results with Quirk's CFD simulations [6].

6.1. Standard Shapes Comparison

6.1.1. Light Inhomogeneity Cases ($A = -0.79$)

Simulation parameters used are shown in Table 1 for the pair Air/Helium ($A = -0.79$) as in [1,6]. For the time step we use the CFL criterion ([23,24]) and $Dt = 10^{-9}s$.

Table 1. Computational set-up for the Air-Helium system: $N_{p,1}$, $N_{p,2}$, $N_{p,3}$, $N_{p,4}$ are the number of particles of type 1, 2, 3 and 4. ρ_1, ρ_2 and ρ_3 are the density of particles of type 1, 2, and 3 expressed as Kg m^{-3}. P_r is the pressure ratio. h_1 is the smoothing length of particles type 1. h_2 is the smoothing length of particles type 2 and 3. τ_s is the dimensionless time step of the simulation. α is the dimensionless factor controlling the dissipation strength. Ma is the shock wave Mach number in the simulation.

$N_{p,1}$	$N_{p,2}$	$N_{p,3}$	$N_{p,4}$	ρ_1	ρ_2	ρ_3	P_r	h_1	h_2	τ_s	α	Ma
125,050	244,464	5536	42,121	5	1.16	0.18	20	1.15 dL	0.5 dL	$1.39 \cdot 10^{-11}$	0.1	1.22

Comparison between Figures 3 and 5 shows that SPH is in good agreement with the experimental shadowgraphs. With respect to the CFD simulations, the SPH results are slightly less reliable at initial

times (e.g., the first two shapes in Figure 5) but more reliable at later times. The relatively small differences in timescale are discussed in the next section.

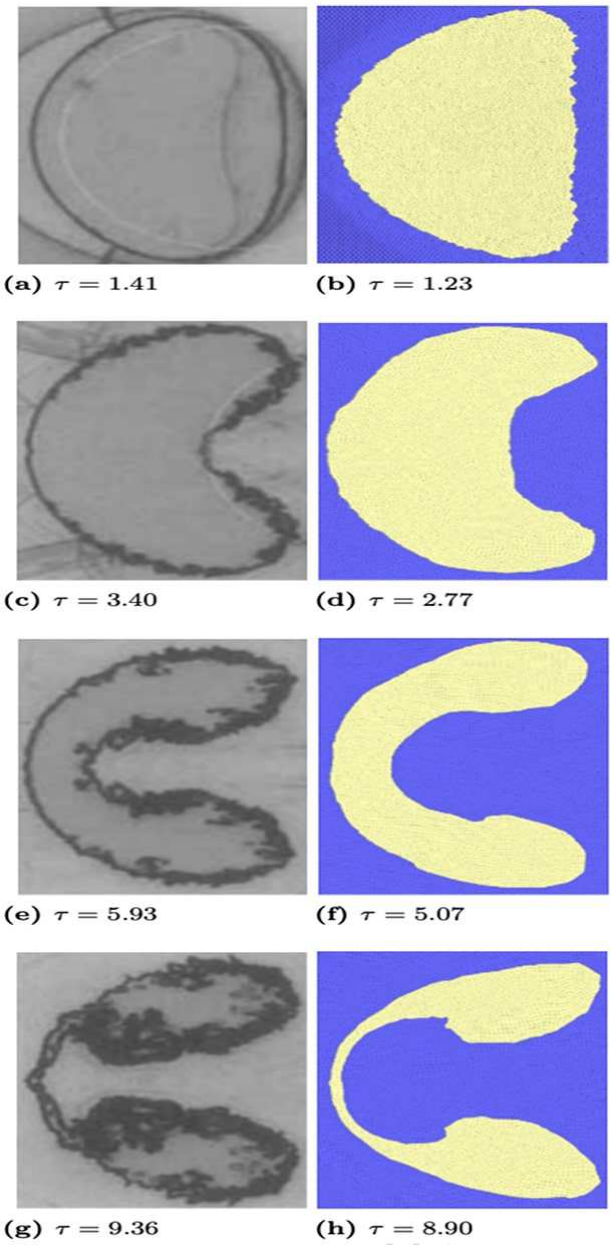


Figure 5. Shape comparison between results [6] (left column) and smoothed particle hydrodynamics (SPH) results (right column) for $A = -0.79$. See Figure 3 for the equivalent experimental data [1].

6.1.2. Heavy Inhomogeneity Cases ($A = 0.51$)

Simulations parameters are shown in Table 2 for the pair Air/Dichlorodifluoromethane (R12) ($A = 0.51$) as in [1,6]. The timestep is $Dt = 10^{-9}$ s as before.

Table 2. Computational set-up for the Air-R12 system: $N_{p,1}$, $N_{p,2}$, $N_{p,3}$, $N_{p,4}$ are the number of particles of type 1, 2, 3 and 4. ρ_1, ρ_2 and ρ_3 are the density of particles of type 1, 2, and 3 expressed as Kg m^{-3}. P_r is the pressure ratio. h_1 is the smoothing length of particles type 1. h_2 is the smoothing length of particles type 2 and 3. τ_s is the dimensionless time step of the simulation. α is the dimensionless factor controlling the dissipation strength. *Ma* is the shock wave Mach number in the simulation.

$N_{p,1}$	$N_{p,2}$	$N_{p,3}$	$N_{p,4}$	ρ_1	ρ_2	ρ_3	P_r	h_1	h_2	τ_s	α	Ma
125,050	244,464	5536	42,121	5	1.16	3.65	20	1.15 dL	0.5 dL	$1.39 \cdot 10^{-11}$	0.1	1.22

In addition, in this case, the SPH results (Figure 6) show good agreement with the available experimental data (Figure 4). With respect to the CFD simulations, the SPH results look more reliable especially at longer times (e.g., the jellyfish head shape in the last figure of Figure 6). Again, the small timescale differences are discussed in the next section.

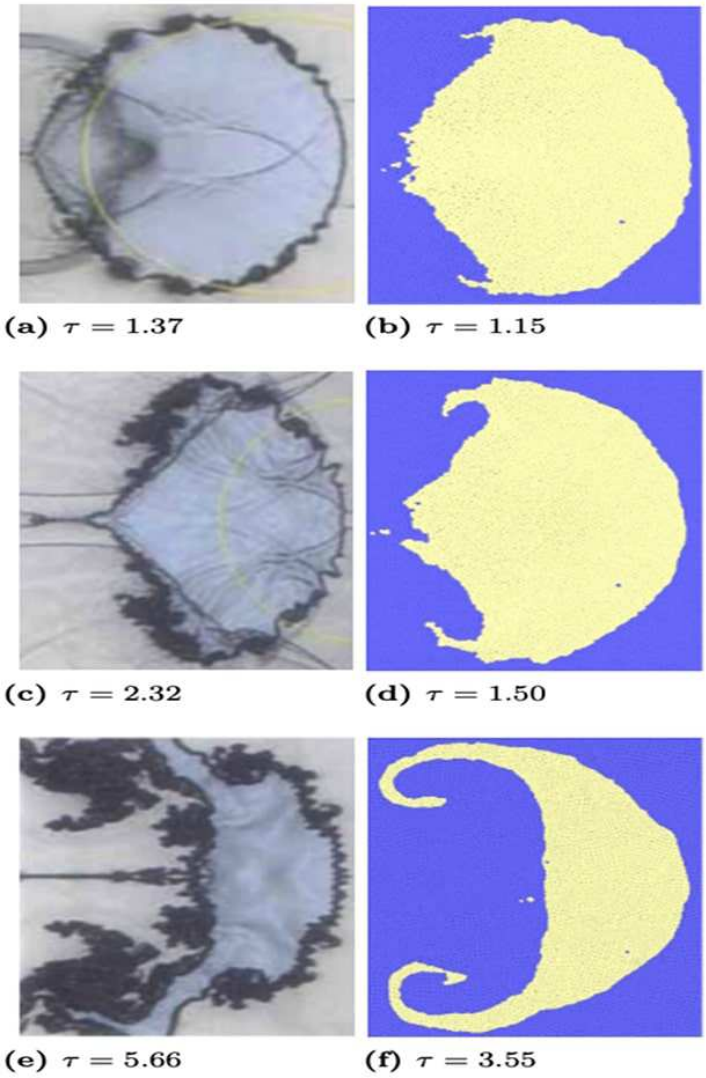


Figure 6. Shape comparison between CFD results [6] (**left** column) and SPH results (**right** column) for $A = 0.51$. See Figure 4 for the equivalent experimental data [1].

6.2. Timescale Comparison

Our results and those of Quirk [6] show approximately the same shapes at slightly different computational times. In this section we will compare the timescale from different experiments to verify the validity of our timescale. In fact, the identification of the different shapes is usually performed visually and, therefore, a certain inaccuracy is expected. The data presented in Figure 7 are from Haas & Sturtevant [1], Levy [19] and Layes [7,25] and refer to conditions analogous to our SPH model. Results are only presented for the air-helium system. Since the R12 is toxic, the air-R12 system is less investigated and there are not enough data in literature for an exhaustive comparison.

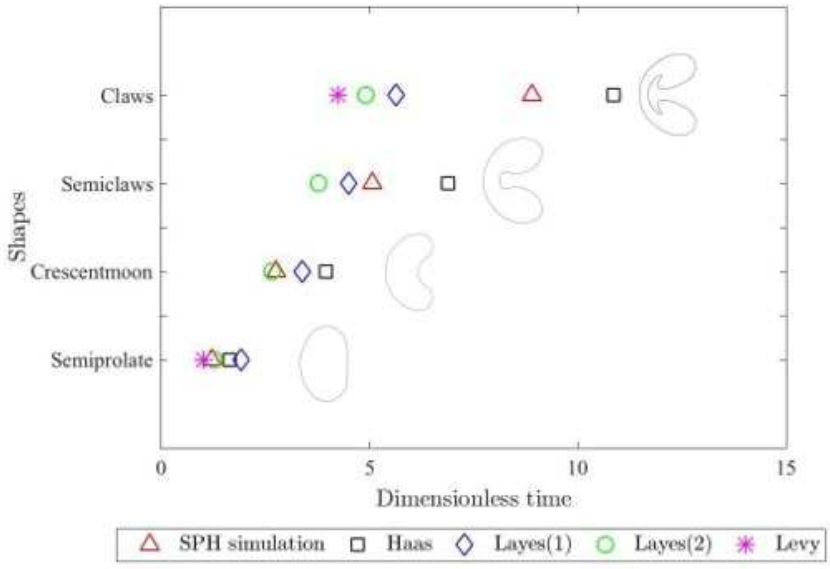


Figure 7. Standard shapes timescale comparison.

Figure 7 shows the deviation of experimental data and that our results lie within the experimental uncertainty.

6.3. Time Depending Artificial Viscosity

Normally, in SPH simulation, the dissipation constant α is maintained constant during the simulation. However, as a means of improving results in the case of shock waves, Morris & Monaghan [22] introduced a time-varying coefficient, $\alpha(t)$,

$$\alpha(t) = \alpha^* + \alpha_0 exp\left(-\frac{t}{\tau_e}\right), \tag{16}$$

where α^* is the minimum dissipation factor, α_0 is the initial dissipation factor and τ_e is the e-folding time. In this section, we test the variable dissipation of Equation (16) with $\alpha_0 = 0.1$, $\alpha^* = 10^{-6}$ and, following the procedure of Morris & Monaghan [22], $\tau_e = 1.05 \cdot 10^{-5}$, to assess if it can further improve the results. Figures 8 and 9 show the comparison between the results calculated with $\alpha = 0.1$ and $\alpha(t)$. The results at lower times are almost identical to those in Figures 5 and 6 and are not reported. At higher computational times there is a small improvement, especially for the claw shape. However, in general the results with constant α seem reasonably accurate.

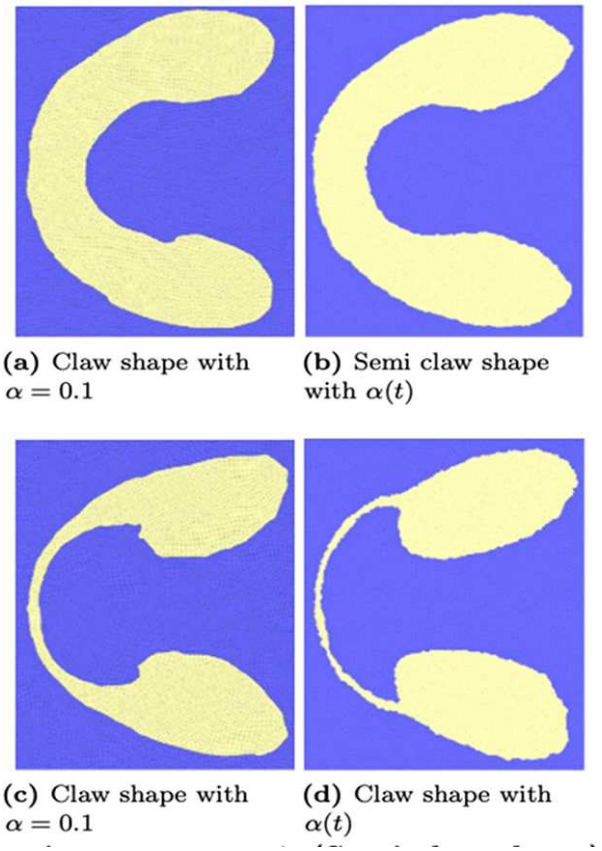


Figure 8. Shapes comparison at $\tau = 5.07$ (Semi claw shape) and $\tau = 8.90$ (Claw shape) between constant viscosity (**left**) and time-varying viscosity (**right**).

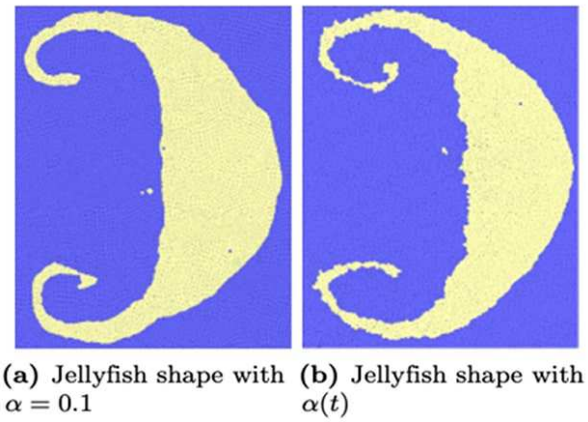


Figure 9. Shape comparison at $\tau = 3.55$ (Jellyfish shape) between constant viscosity (**left**) and time-varying viscosity (**right**).

6.4. Pressure Field

This section reports the pressure field calculated with the method proposed in this study for both the light and the heavy inhomogeneity. Analysis of the pressure field, in fact, allows to understand and explain why the inhomogeneity assumes specific shapes during its evolution. In the case of light inhomogeneity

(Figure 10), initially the incident shock wave impacts and reflects on the inhomogeneity (Figure 10a) generating a lower pressure reflected-wave behind the inhomogeneity. At a later stage, a high-pressure region behind the inhomogeneity (Figure 10b) is observed and, consequently, the lighter gas moves away from the high-pressure area giving a crescent moon shape to the inhomogeneity. In the case of heavy inhomogeneity (Figure 11), a reflected-wave (Figure 11a) and a high pressure region (Figure 11b) also forms, but, this time, the reflected wave has a higher pressure than the surroundings and the high-pressure area forms in front of the inhomogeneity. Moreover, the high pressure area is partly outside and partly inside the gas inhomogeneity. As a result of this, the tip of the inhomogeneity stretches, forming the central wedge typical of the jellyfish head shape.

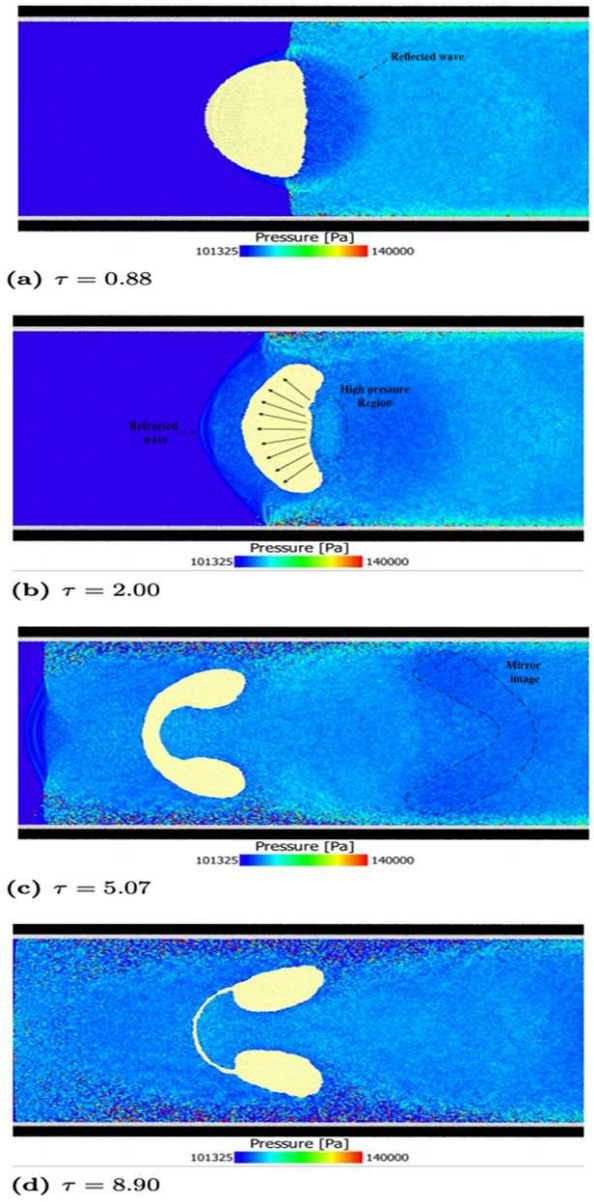


Figure 10. Pressure field in the driven gas for the light inhomogeneity ($A = -0.79$) case at different dimensionless times.

The included video files Video1.avi and Video2.avi, see Supplementary Materials, show the evolution of the pressure field at the same conditions, respectively, of Figures 10 and 11.

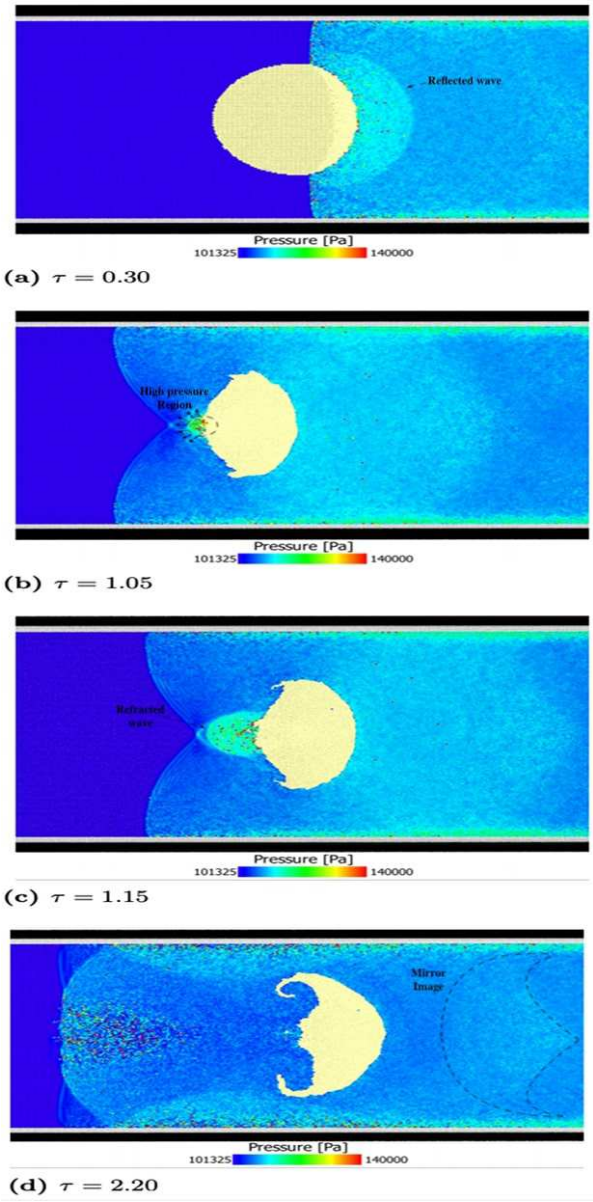
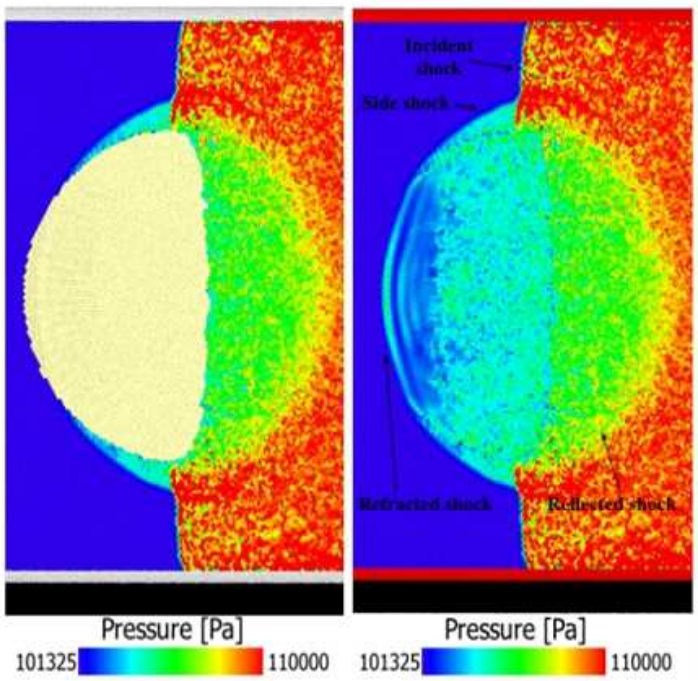


Figure 11. Pressure field in the driven gas for the heavy inhomogeneity ($A = 0.51$) case at different dimensionless times.

For clarity, Figures 10 and 11 show the pressure field only in the driven gas, while Figure 12 shows the pressure field both outside and inside the inhomogeneity. Our calculations also capture the twin regular reflection-refraction (TRR) configuration (Figure 12), firstly observed by Henderson [26]. The TRR is a four-shock configuration where the refracted shock moves faster than the incident shock, with the reflected shock moving in the opposite direction. The fourth shock is the side shock, which connects the refracted shock with the incident shock (Figure 12b). The pressure field in some of the figures is slightly "noisy" at certain locations. This is probably the result of the piecewise constant smoothing length used in this study. However, this occurs far from the gas inhomogeneity and does not affect the inhomogeneity shape evolution analysis carried out in this study.



(a) Pressure field in the gas driver.

(b) Pressure field in both driven gas and gas inhomogeneity.

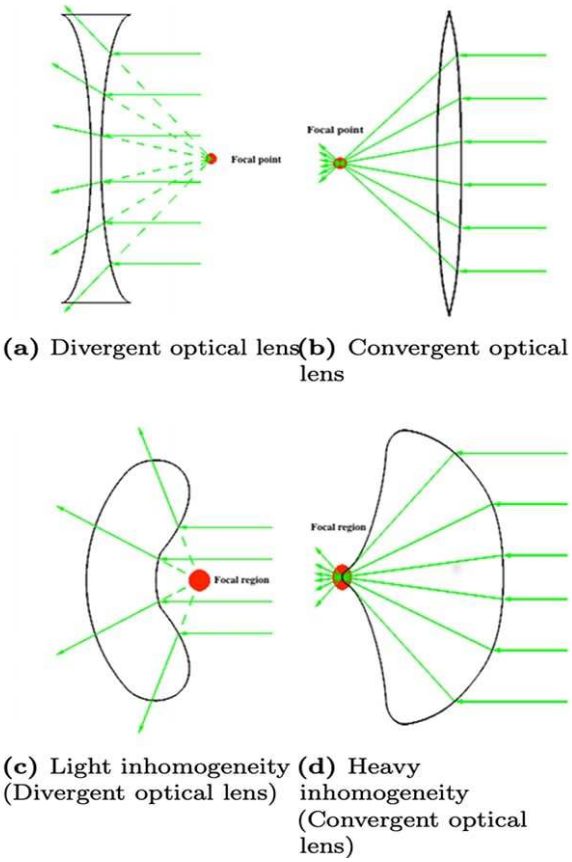
Figure 12. Twin regular reflection-refraction (TRR) in the light inhomogeneity case ($A = -0.79$) at $\tau = 0.88$.

Refracted and Reflected Waves (Acoustic Lens and Acoustic Mirror)

When the shock passes through the light inhomogeneity, the speed of the wave increases, while its pressure decreases. As a result of this, the direction of the refracted wave diverges following a direction given by the high-pressure region behind the light inhomogeneity (Figure 10b). Conversely, when the pressure wave passes through the heavy inhomogeneity, the speed decreases and the pressure increases; in this case the refracted wave converges to the high-pressure region in front of the heavy inhomogeneity (Figure 11b). This behaviour suggests that a gas inhomogeneity could behave, to a certain degree, like an acoustic lens.

When a collimate beam of light passes through an optical lens, the direction of the beam changes according to the position of the focal point of the lens. In divergent lenses (Figure 13a), the focal point is behind the lens; in convergent lenses (Figure 13b), it is in front of the lens, similar to a collimate beam of

light diverged or converged. With respect to the focal point, the shock wave is either diverged or converged with respect to the high pressure region area discussed in the previous section (compare Figures 10b and 11b with Figure 13c,d). Considering, therefore, that the focal point and the high pressure region play a similar role, it is possible to identify the high pressure region as a focal region of the system.



(a) Divergent optical lens **(b)** Convergent optical lens

(c) Light inhomogeneity (Divergent optical lens) **(d)** Heavy inhomogeneity (Convergent optical lens)

Figure 13. Comparison between optical lenses and gas inhomogeneities behaving as acoustic lenses.

Another interesting observation concerns the shape of the reflected wave (Figures 10 and 11). Observing the pressure field, it is possible to see how the evolution of the reflected wave mirrors reflects, to a certain degree, that of the inhomogeneity. In the light inhomogeneity, the mirror image (Figure 10c) has a shape similar to the semi-claw standard shape, it is just more elongated in the y-direction. In the heavy inhomogeneity, the mirror image (Figure 11d) has a shape similar to the jellyfish head standard shape.

7. Conclusions

Typically, in gas dynamics with SPH, the approach is to employ a smoothing length varying with density. In this paper we show that a simpler piecewise constant smoothing length is sufficient to model the dynamics of inhomogeneities in shock tubes. With this device, which fits the underlying physics, we obtain (i) the correct shapes, (ii) the correct timescale and (iii) the correct refraction/reflection of the wave (something CFD simulations sometimes fail to achieve). This can be useful in at least two directions. Firstly, in combination with the energy conservation equation, the proposed SPH approach could be adapted for simulating a variety of phenomena related with the so-called Richtmyer–Meshkov instability [9] such as supersonic mixing and gas combustion in Scramjet. Secondly, it can be integrated with

Discrete Multiphysics (DMP) for the simulation of cavitation erosion. Discrete Multi-Physics (e.g., [27,28]) is a multiphysics technique that, contrary to traditional multiphysics, is based on computational particles rather than computational meshes. It combines different particle-based modelling techniques such as smooth particle hydrodynamics, discrete element method and the lattice spring model, and it has been effectively used for fluid-structure interaction problems (e.g., [29–31]). DMP, in particular, is superior to traditional multiphysics in the case of phase-transition [32], agglomeration [33] and break-up of solid structures [34]. Specifically, the SPH model presented in this study, in particular, could be coupled with the break-up module in DMP to model cavitation generated shock waves and their effects, including erosion, on nearby solid surfaces.

Supplementary Materials: The following are available online at http://www.mdpi.com/2076-3417/9/24/5435/s1, Video1: Heavy inhomogeneity evolution with pressure field, Video2: Light inhomogeneity evolution with pressure field.

Author Contributions: A.A. (Andrea Albano) and A.A. (Alessio Alexiadis) conceptualise the work; A.A. (Andrea Albano) designed the work and performed the simulations; A.A. (Andrea Albano) and A.A. (Alessio Alexiadis) contributed in writing–review and editing the paper.

Funding: This work was supported by the US Office of Naval Research Global (ONRG) under 256 NICOP Grant N62909-17-1-2051.

Conflicts of Interest: The authors declare no conflict of interest

References

1. Haas, J.F.; Sturtevant, B. Interaction of weak shock waves with cylindrical and spherical gas inhomogeneities. *J. Fluid Mech.* **1987**, *181*, 41–76. [CrossRef]
2. Jacobs, J. The dynamics of shock accelerated light and heavy gas cylinders. *Phys. Fluids A Fluid Dyn.* **1993**, *5*, 2239–2247. [CrossRef]
3. Layes, G.; Jourdan, G.; Houas, L. Experimental investigation of the shock wave interaction with a spherical gas inhomogeneity. *Phys. Fluids* **2005**, *17*, 028103. [CrossRef]
4. Wang, X.; Yang, D.; Wu, J.; Luo, X. Interaction of a weak shock wave with a discontinuous heavy-gas cylinder. *Phys. Fluids* **2015**, *27*, 064104. [CrossRef]
5. Medvedev, S.; Khomik, S.; Cherepanova, T.; Agafonov, G.; Cherepanov, A.; Mikhalkin, V.; Kiverin, A.; Petukhov, V.; Yakovenko, I.; Betev, A. Interaction of blast waves with helium-filled rubber balloons. *J. Phys. Conf. Ser.* **2019**, *1147*, 012021. [CrossRef]
6. Quirk, J.J.; Karni, S. On the dynamics of a shock–bubble interaction. *J. Fluid Mech.* **1996**, *318*, 129–163. [CrossRef]
7. Layes, G.; Le Métayer, O. Quantitative numerical and experimental studies of the shock accelerated heterogeneous bubbles motion. *Phys. Fluids* **2007**, *19*, 042105. [CrossRef]
8. Georgievskiy, P.Y.; Levin, V.; Sutyrin, O. Interaction of a shock with elliptical gas bubbles. *Shock Waves* **2015**, *25*, 357–369. [CrossRef]
9. Giordano, J.; Burtschell, Y. Richtmyer-Meshkov instability induced by shock-bubble interaction: Numerical and analytical studies with experimental validation. *Phys. Fluids* **2006**, *18*, 036102. [CrossRef]
10. Fan, E.; Guan, B.; Wen, C.Y.; Shen, H. Numerical study on the jet formation of simple-geometry heavy gas inhomogeneities. *Phys. Fluids* **2019**, *31*, 026103. [CrossRef]
11. Gingold, R.A.; Monaghan, J.J. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Mon. Not. R. Astron. Soc.* **1977**, *181*, 375–389. [CrossRef]
12. Lucy, L.B. A numerical approach to the testing of the fission hypothesis. *Astron. J.* **1977**, *82*, 1013–1024. [CrossRef]
13. Swegle, J.; Attaway, S. On the feasibility of using smoothed particle hydrodynamics for underwater explosion calculations. *Comput. Mech.* **1995**, *17*, 151–168. [CrossRef]
14. Liu, M.; Liu, G.; Lam, K.; Zong, Z. Smoothed particle hydrodynamics for numerical simulation of underwater explosion. *Comput. Mech.* **2003**, *30*, 106–118. [CrossRef]

15. Monaghan, J. SPH and Riemann solvers. *J. Comput. Phys.* **1997**, *136*, 298–307. [CrossRef]
16. Molteni, D.; Bilello, C. Riemann solver in SPH. *Mem. Della Soc. Astron. Ital. Suppl.* **2003**, *1*, 36.
17. Ganzenmüller, G.C.; Steinhauser, M.O.; Van Liedekerke, P.; Leuven, K.U. The Implementation of Smooth Particle Hydrodynamics in LAMMPS. *Paul Van Liedekerke Katholieke Universiteit Leuven* **2011**, *1*, 1–26.
18. Monaghan, J.; Gingold, R.A. Shock simulation by the particle method SPH. *J. Comput. Phys.* **1983**, *52*, 374–389. [CrossRef]
19. Levy, K.; Sadot, O.; Rikanati, A.; Kartoon, D.; Srebro, Y.; Yosef-Hai, A.; Ben-Dor, G.; Shvarts, D. Scaling in the shock–bubble interaction. *Laser Part. Beams* **2003**, *21*, 335–339. [CrossRef]
20. Benz, W. Smooth particle hydrodynamics: A review. In *The Numerical Modelling of Nonlinear Stellar Pulsations*; Springer: Berlin, Germany, 1990; pp. 269–288.
21. Liu, M.; Liu, G.; Lam, K. Investigations into water mitigation using a meshless particle method. *Shock Waves* **2002**, *12*, 181–195. [CrossRef]
22. Morris, J.; Monaghan, J. A switch to reduce SPH viscosity. *J. Comput. Phys.* **1997**, *136*, 41–50. [CrossRef]
23. Morris, J.P.; Fox, P.J.; Zhu, Y. Modeling low Reynolds number incompressible flows using SPH. *J. Comput. Phys.* **1997**, *136*, 214–226. [CrossRef]
24. Courant, R.; Friedrichs, K.; Lewy, H. Über die partiellen Differenzengleichungen der mathematischen Physik. *Math. Ann.* **1928**, *100*, 32–74. [CrossRef]
25. Layes, G.; Jourdan, G.; Houas, L. Experimental study on a plane shock wave accelerating a gas bubble. *Phys. Fluids* **2009**, *21*, 074102. [CrossRef]
26. Henderson, L.F.; Colella, P.; Puckett, E.G. On the refraction of shock waves at a slow–fast gas interface. *J. Fluid Mech.* **1991**, *224*, 1–27. [CrossRef]
27. Alexiadis, A. The discrete multi-hybrid system for the simulation of solid-liquid flows. *PLoS ONE* **2015**, *10*, e0124678. [CrossRef]
28. Alexiadis, A. A smoothed particle hydrodynamics and coarse-grained molecular dynamics hybrid technique for modelling elastic particles and breakable capsules under various flow conditions. *Int. J. Numer. Methods Eng.* **2014**, *100*, 713–719. [CrossRef]
29. Ariane, M.; Kassinos, S.; Velaga, S.; Alexiadis, A. Discrete multi-physics simulations of diffusive and convective mass transfer in boundary layers containing motile cilia in lungs. *Comput. Biol. Med.* **2018**, *95*, 34–42. [CrossRef]
30. Ariane, M.; Allouche, M.H.; Bussone, M.; Giacosa, F.; Bernard, F.; Barigou, M.; Alexiadis, A. Discrete multi-physics: A mesh-free model of blood flow in flexible biological valve including solid aggregate formation. *PLoS ONE* **2017**, *12*, e0174795. [CrossRef]
31. Alexiadis, A.; Stamatopoulos, K.; Wen, W.; Batchelor, H.; Bakalis, S.; Barigou, M.; Simmons, M. Using discrete multi-physics for detailed exploration of hydrodynamics in an in vitro colon system. *Comput. Biol. Med.* **2017**, *81*, 188–198. [CrossRef]
32. Alexiadis, A.; Ghraybeh, S.; Qiao, G. Natural convection and solidification of phase-change materials in circular pipes: A SPH approach. *Comput. Mater. Sci.* **2018**, *150*, 475–483. [CrossRef]
33. Ariane, M.; Wen, W.; Vigolo, D.; Brill, A.; Nash, F.; Barigou, M.; Alexiadis, A. Modelling and simulation of flow and agglomeration in deep veins valves using discrete multi physics. *Comput. Biol. Med.* **2017**, *89*, 96–103. [CrossRef] [PubMed]
34. Alexiadis, A. A new framework for modelling the dynamics and the breakage of capsules, vesicles and cells in fluid flow. *Procedia IUTAM* **2015**, *16*, 80–88. [CrossRef]

Chapter 6

A Smoothed Particle Hydrodynamics Study of the collapse for a cylindrical cavity

In this Chapter the model developed in Chapter 5 is adapted to simulate a gas-filled cylindrical Rayleigh collapse. The Chapter also investigates the role of heat generation and transfer between the gas and the liquid phase during the collapse.

This chapter has been published in *PLOS ONE* as:

Albano A, Alexiadis A (2020) A smoothed particle hydrodynamics study of the collapse for a cylindrical cavity. PLoS ONE 15(9): e0239830

My contributions in this work were: Designed the work and performed the simulations, Methodology, Validation, Writing the original draft, Reviewing and Editing.

I would like to thank all the authors who have contributed to this work.

PLOS ONE

RESEARCH ARTICLE

A smoothed particle hydrodynamics study of the collapse for a cylindrical cavity

Andrea Albano[☯]*, **Alessio Alexiadis**[☯]*

School of Chemical Engineering, University of Birmingham, Birmingham, United Kingdom

☯ These authors contributed equally to this work.
* AXA1220@student.bham.ac.uk (AA); Alexiadis@bham.ac.uk (AA)

Citation: Albano A, Alexiadis A (2020) A smoothed particle hydrodynamics study of the collapse for a cylindrical cavity. PLoS ONE 15(9): e0239830. https://doi.org/10.1371/journal.pone.0239830

Editor: Michael H Peters, Virginia Commonwealth University, UNITED STATES

Received: March 12, 2020

Accepted: September 14, 2020

Published: September 29, 2020

Peer Review History: PLOS recognizes the benefits of transparency in the peer review process; therefore, we enable the publication of all of the content of peer review and author responses alongside final, published articles. The editorial history of this article is available here: https://doi.org/10.1371/journal.pone.0239830

Data Availability Statement: All relevant data are within the manuscript and its Supporting Information files.

Funding: This work was supported by the US office of Naval Research Global (ONRG) under NICOP Grant N62909-17-1-2051.

Abstract

In this study, we propose a mesh-free (particle-based) Smoothed Particle Hydrodynamics model for simulating a Rayleigh collapse. Both empty and gas cavities are investigates and the role of heat diffusion is also accounted for. The system behaves very differently according to the ratio between the characteristic time of collapse and the characteristic time of thermal diffusion. This study identifies five different possible behaviours that range from isothermal to adiabatic.

Introduction

The term "cavitation" describes a phenomenon composed by two distinct phases: first, a vapour cavity, also called vapour bubble or void, develops and rapidly grows in a liquid phase; subsequently, the vapour cavity rapidly collapses generating strong shock waves.

Cavitation causes erosion and it is mostly undesirable in engineering applications such as turbo-machines, propellers, and fuel injectors [1–3]. However, other applications such as ultrasonic cleaning or cataract surgery [4–6] are specifically designed to take advantage of the erosion power of the collapsing bubble.

According to the circumstances, the bubble collapse can follow two distinct, but similar, mechanisms [7] called, respectively, Rayleigh collapse and shock-induced collapse. During the Rayleigh collapse, the collapse is driven by the pressure difference between the surrounding liquid and the cavity. In this case, if the pressure field is perfectly isotropic, the bubble maintains a spherical shape during the whole duration of the collapse. Shock-induced collapse is caused by the passages of a shock-wave through the bubble. In this case, the spherical shape is not preserved and the bubble folds in the shock direction.

Our current understanding of cavitation is based on three different approaches: (i) theoretical investigations, (ii) experiments and (iii) computer simulations.

The first analytical study of an empty cavity surrounded by an incompressible fluid at given pressure was carried out by W. H. Besant (1859) [8], who obtained an integral expression for determining the time required for the cavity to collapse due to the effect of a constant external pressure. Sixty years later, Lord Rayleigh [9] was able to integrate this equation determining that, during the collapse, the pressure of the liquid near the boundary exceeds the pressure of surrounding liquid. Plesset [10] introduced the effect of surface tension and viscosity obtaining

Competing interests: The authors have declared that no competing interests exist.

the well-known Rayleigh-Plesset equation that describes the dynamics of a spherical bubble in an infinite body of incompressible fluid. Later, other studies included thermal effect and liquid compressibility [11–14]. Theoretical investigation of the isotropic collapse has continued up to the present day and, recently, Kudryashov & Sinelshchikov [15] found a closed form general solution of the Rayleigh equation for both empty and gas-filled spherical bubbles. The same authors also found an analytical solution of the Rayleigh equation where the surface tension is account for [16].

Experimentally, the study of a collapsing bubble has been a challenge due to difficulty of generating a perfectly spherical bubble, and practical difficulties of measuring relevant data during the short duration of the collapse. The first issue was solved with laser produced cavitation bubbles (eg. [17–19]). This technique, coupled with High-speed photography, increased in particular our understanding of the dynamics of a collapsing bubble in non-isotropic conditions (e.g. near a solid surface) highlighting the role of the so-called jet formation in cavitation erosion. However, the second issue remains an open challenge. In fact, theoretical studies (eg. [20, 21]) calculated temperatures inside the collapsing bubbles to be between 6700 K and 8800 K and pressures up to 848 bar. These peak values, however, occur only for very small intervals of time ($\approx 2\mu$s) and, up to now, the short timescale has prevented accurate experimental analysis of the phenomenon.

The use of computer simulations for investigate cavitation is more recent. Computer simulation can perform "numerical experiments" that, contrary to actual experiments, are not limited by short time-scales and small bubble sizes. During the years, a variety of simulations methods have been used for simulating the collapse of a bubble both near and away from a solid surface: Plesset-Champan used the particle-in-cell method [22], Blake used the boundary integral method [23], Klaseboer [24] used the boundary element method, while Johnsen [7] a high-order accurate shock- and interface-capturing scheme.

All these studies are based on mesh-based computational methods. Meshfree methods are generally considered easier to implement for highly deformable interfaces [25] but, surprisingly, only few articles have simulated cavitation with meshfree methods. One of the few exceptions is Joshi et al. [26, 27] that took advantage of the meshfree nature of Smoothed Particle hydrodynamics (SPH) to develop an axisymmetric model simulating not only the collapse of the cavity, but also the effect of the shock waves on a nearby solid surface (e.g. deformation, erosion). However, the empty cavity used in their model prevents thermal analysis. Albano & Alexiadis [28] developed a SPH model for shock wave interacting with a discrete gas inhomogeneity, this phenomenon share similar physics to the shock induced collapse [29].

This study proposes the first SPH model simulating a Rayleigh collapse of a cavity filled with non-condensable gas induced by abruptly change in pressure. Moreover, by implementing the diffusive heat transfer mechanism, both adiabatic and heat diffusive collapse are simulated. The aim is to investigate the role of heat diffusion in the pressure and temperature development.

The role of heat transfer in reducing the peak temperature of the collapse is known since the '80s [21]. Nevertheless, the diffusion mechanism is often neglected in modelling work and the collapse is assumed adiabatic without justification [30, 31].

Smoothed particle hydrodynamics

Originally, Gingold and Monaghan [32] and Lucy [33] developed Smoothed-Particle Hydrodynamics (SPH) as a mesh-free particle method for solving astrophysical problems. However, earliest applications also focused on solving fluid dynamics problems [34–36]. In fact, SPH has major advantages in simulating free surface flows and large deformations due to its Lagrangian

nature [37, 38]. The method has been validated for wide range of applications such as explosion [39], underwater explosion [40], shock waves [28, 41, 42], high (or hyper) velocity impact [43], water/soil-suspension flows [44], free surface flows [45, 46], nano-fluid flows [47], thermo-fluid application [48]. Moreover, SPH is also a component of the Discrete multi-physics simulations [49–53]

SPH bases its discrete approximation of a continuum medium on the expression

$$f(\mathbf{r}) \approx \int\int\int f(\mathbf{r}')W(\mathbf{r}-\mathbf{r}',h)d\mathbf{r}', \tag{1}$$

where $f(\mathbf{r})$ is any continuum function depending of the three-dimensional position vector $\mathbf{r}$, while W is the smoothing function or kernel. The kernel function W defines the extension of the support domain, the consistency, and accuracy of the particle approximation [25]. When the computational domain is divided in computational particles with their own mass, $m = \rho dr$, it is possible to rewrite Eq 1 in particle form

$$f(\mathbf{r}) \approx \sum \frac{m_i}{\rho_i} f(\mathbf{r}_i)W(\mathbf{r}-\mathbf{r}_i,h), \tag{2}$$

where m_i, ρ_i and $\mathbf{r}_i$ are mass, density and position of the i^{th} particle. Within the SPH framework, it is possible to use Eq 2 to discretise a set of equations such as the continuity equation

$$\frac{d\rho}{dt} = -\rho\nabla\cdot\mathbf{v}, \tag{3}$$

which in particle form becomes

$$\frac{d\rho_i}{dt} = \sum_j m_j\mathbf{v}_{ij}\nabla_j W_{ij}; \tag{4}$$

the momentum equation

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla\cdot P, \tag{5}$$

which in particle form becomes

$$m_i\frac{d\mathbf{v}_i}{dt} = \sum_j m_i m_j\left(\frac{P_i}{\rho_i}+\frac{P_i}{\rho_i}+\Pi_{ij}\right)\nabla_j W_{ij}; \tag{6}$$

where Π_{ij} is called artificial viscosity and was introduced by Monaghan [41] for simulating shock waves, having the expression

$$\Pi_{ij} = -\beta h\frac{c_i+c_j}{\rho_i+\rho_j}\frac{\mathbf{v}_{ij}\cdot\mathbf{r}_{ij}}{r_{ij}^2+\epsilon h^2}, \tag{7}$$

where β is the dimensionless dissipation factor, c_i and c_j are the speed of sound of particle i and j.

The energy conservation equation

$$\frac{de}{dt} = -\frac{1}{\rho}(P(\nabla\cdot\mathbf{v})+\tau:\nabla\mathbf{v}) - \frac{1}{\rho}\nabla\cdot(-k\nabla T), \tag{8}$$

which in particle form becomes

$$m_i \frac{de_i}{dt} = \frac{1}{2}\sum_j m_i m_j \left(\frac{P_i}{\rho_i} + \frac{P_i}{\rho_i} + \Pi_{ij}\right) : \mathbf{v}_{ij}\nabla_j W_{ij} - \sum_j \frac{m_i m_j}{\rho_i \rho_j}\frac{(\kappa_i + \kappa_j)(T_i - T_j)}{r_{ij}^2}\mathbf{r}_{\mathbf{ij}} \cdot \nabla_j W_{ij}, \quad (9)$$

where κ is the thermal conductivity. The first term of the right side of Eq 9 is the particle form of the sum of the reversible rate of the internal energy increase by compression and the irreversible rate of internal energy increase by viscous dissipation; the second term is the particle form of the rate of internal energy increment by heat conduction following Fourier's Law. The thermal conductivity is related to the thermal diffusivity, α, by the relationship

$$\alpha = \frac{\kappa}{\rho c_p}, \quad (10)$$

where c_p is the specific heat capacity.

Kernels function

In this work, two kernels (Lucy Kernel function and quintic spline) are used and their effect on the accuracy of the results compared. The Lucy kernel function [33]

$$W(q,h) = \begin{cases} \frac{1}{s}[1+3q][1-q]^3, & q \leq 1 \\ 0, & q > 1, \end{cases} \quad (11)$$

is one of the simplest kernels used in literature. Where $q = |\mathbf{r} - \mathbf{r}'|/h$, s is a parameter used to normalise the kernel function, which, for one, two and three dimensional space is, respectively, $\frac{4h}{5}$, $\frac{\pi h^2}{5}$ and $\frac{16\pi h^3}{105}$. The quintic spline is a piecewise kernel function [54]

$$W(Q,h) = s\begin{cases} (3-q)^5 - 6(1-q)^5 + 15(1-q)^5, & 0 < q < 1 \\ (3-q)^5 - 6(1-q)^5, & 1 < q < 2 \\ (3-q)^5, & 2 < q < 3 \\ 0, & q > 3 \end{cases} \quad (12)$$

where $q = |\mathbf{r} - \mathbf{r}'|/h$ and s for one, two and three dimensional space is, respectively $\frac{1}{120h}$, $\frac{7}{478\pi h^2}$ and $\frac{3}{359\pi h^3}$. In the quintic kernel is normally more accurate, but at the expenses of higher computational costs because it requires a neighbor list three times larger than the Lucy kernel [25].

Model

Problem description. In the Rayleigh collapse, the driver force is the pressure difference between the pressure in the liquid, $p_\infty = P_L$, and the pressure in the cavity, p_b.

Two different scenarios are analysed: empty cavity collapse, where the cavity is void, with $p_b = 0$, surrounded by a liquid phase, and vapour cavity collapse, where the cavity is filled of a non condensable gas with an initial pressure equal to the vapour pressure of water at the temperature T_0, $p_b = p_{sat}(T_0)$ and density equal to the density of an ideal gas at that pressure and temperature, $\rho_b(p_b, T_0)$.

The liquid phase is water with $\rho_L = 1000$ kg/m^3, $P_L = 5$ MPa and $T_0 = 300$ K. The liquid pressure of 5 MPa has been chosen as it is commonly reached in various hydraulic applications [55]. Given the liquid temperature, the pressure in the bubble is $p_b = 3.55$ kPa with a density of

$\rho_b(p_b, T_0) = 2.7 \cdot 10^{-3}$ kg/m^3. At the short timescale considered, water is compressible. The equation of state for compressible water is discussed later on.

In rapid collapse, the water vapour is considered trapped within the liquid, assuming zero mass transport across the interface. This simplification is justified because mass transport mechanisms across the interface and non-equilibrium condensation require higher timescales to play an effective role in the collapse phase [56]. The short timescale also justifies neglecting the surface tension in modelling the collapse.

The short timescale of the phenomenon may suggest an adiabatic collapse [20, 57]. However, especially in the last stage of the collapse, the high temperature gradient between gas and liquid phases could introduce a non-negligible heat transfer between the two phases [21, 58]. In this study, both scenarios (e.g. adiabatic and non-adiabatic collapse) are investigated.

SPH model. The axisymmetric water domain is shown in Fig 1. The domain is dived in three concentric regions, delimited by three different radiuses, where different types of computational particles are used.

Cavity ($r < R_0$): inside this region particles are removed (in the case of empty cavity) or modelled as non-condensable gas following a gas phase equation of state (EOS) in the case of vapour collapse. In the rest of the paper, particles inside the cavity (when present) will be referred as particle Type 1.

Liquid ($R_0 < r < R_S$): inside this region particles are modelled as compressible fluid following a liquid EOS. Particles inside the liquid will be referred as particle Type 2.

Shell ($r > R_S$): inside this region particles are modelled as fluid with a fixed position and density to represent the boundary conditions of the system and maintain a fixed pressure at the boundaries. Particles inside the shell will be referred as particle Type 3. We also run several simulations with cubic control volumes and periodic conditions that account only for Type 1 and Type 2 particles. The results do not change and, therefore, we prefer the system in Fig 1 that overall requires less computational particles.

To avoid compenetration between gas and liquid particles during the gas cavity collapse, we employed a penalty force, similar to the one used by Liu et al. [40] between these types of particles.

$$f_p = \begin{cases} -C\dfrac{\gamma}{r_{ij}}\left(\dfrac{\sigma}{r_{ij}}\right)^{\gamma}, & r_{ij} \leq \sigma \\ 0, & r_{ij} > \sigma, \end{cases} \tag{13}$$

with $C = 10^{-4}$, $\gamma = 9$ and σ equal to the initial particle spacing.

The initial radius of the cavity is $R_0 = 100\mu$m (typical radius of a collapsing cavity [26, 59]). The ratio $R_C/R_0 = 30$ is used as a compromise between computational cost and accuracy. Different R_S has been tested, as explained in the Hydrodynamic section.

Different resolutions (i.e. total number of computational particles) have been tested ($5.79 \cdot 10^5$, $1.30 \cdot 10^6$ and $2.66 \cdot 10^6$); $N = 1.06 \cdot 10^6$ was chosen as best compromise between accuracy and computational speed (more details in the Hydrodynamic section.

Equation of state. To solve the set of Eqs 3–8, an EOS that links pressure P and density ρ is required. Each phase requires a different EOS: in this work multiple EOS are used and compared.

Liquid EOS. For liquids, we used and compared two EOS: the Tait and the Mie-Gruneisen EOS.

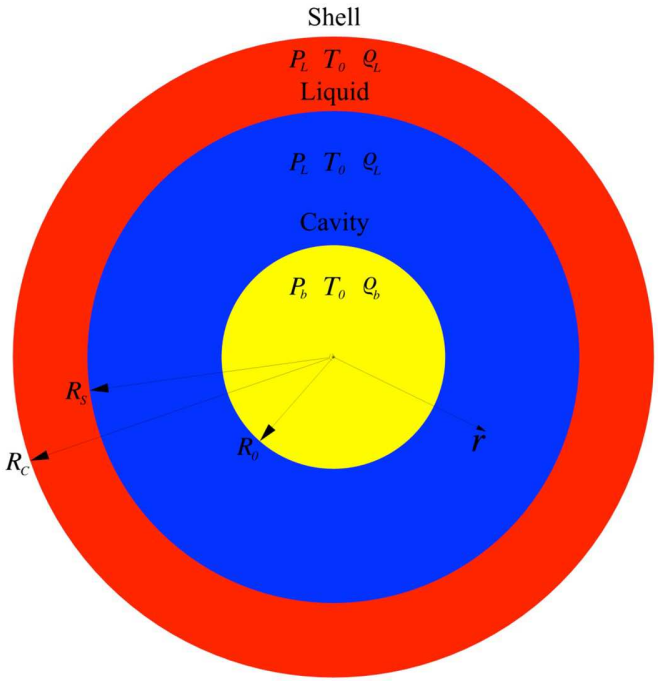


Fig 1. Geometry of the simulation box.

https://doi.org/10.1371/journal.pone.0239830.g001

The Tait equation is probably the most used EOS in SPH to model water

$$P(\rho) = \frac{c_0^2 \rho_0}{7}\left(\left(\frac{\rho}{\rho_0}\right)^7 - 1\right), \tag{14}$$

where c_0 is speed of sound of the liquid and ρ_0 is the reference density. The Tait EOS takes in account the compressibility of the liquid. Is possible to regulate the compressibility by selecting the appropriate sound of speed [60] in Eq 14.

For simulating underwater explosion Liu et all [40] used the Mie-Gruneisen EOS [61] to model the water as a compressible fluid having different expressions for compression and expansion state. Shin et al. [62] derived a polynomial expression for both compression and expansion states: for compression state,

$$P(\rho, e) = a_1\mu + a_2\mu^2 + a + 3\mu^3 + (b_0 + b_1\mu + b_1\mu^2)\rho_0 e, \tag{15}$$

while for expansion state,

$$P(\rho, e) = a_1\mu + (b_0 + b_1\mu)\rho_0 e, \tag{16}$$

Where $\mu = \rho/\rho_0 - 1$ and e is the specific internal energy. The coefficients are $a_1 = 2.19 \cdot 10^9$ N/m^2, $a_2 = 9.224 \cdot 10^9$ N/m^2,$a_3 = 8.767 \cdot 10^9$ N/m^2, $b_0 = 0.4934$ and $b_1 = 1.3937$ evaluated for water with $\rho_0 = 1000$ kg/m^3 and $c_0 = 1480$ m/s.

Vapour EOS. The vapour phase in the cavity is modelled as a non-condensable gas. In our simulations, we used and compared two EOS: the ideal gas EOS and the NASG EOS.

The ideal gas EOS, for the pressure, is given by

$$P(\rho, e) = (\gamma - 1)\rho e, \tag{17}$$

temperature

$$T(e) = M_m \frac{(\gamma - 1)e}{R}, \tag{18}$$

where $\gamma = c_p/c_v$ is the capacity heat ratio, M_m the molar mass of the gas and R is the ideal gas constant. The NASG EOS, which is a multiphase EOS is discussed in the next section.

Multiphase EOS. Le Métayer & Saurel [63] combined the "Noble-Abel" and the "Stiffened-Gas" EOS proposing a EOS called Noble-Abel Stiffened-Gas (NASG), suitable for mulfiphase flow. The expression of the EOS does not change with the phase considered, and, for each phases, is possible to determine both the pressure and temperature as function of density and specific internal energy. Pressure-wise the expression of NASG is

$$P(\rho, e) = (\gamma - 1)\frac{(e - q)}{\left(\frac{1}{\rho} - b\right)} - \gamma P_\infty, \tag{19}$$

and temperature wise

$$T(\rho, e) = \frac{e - q}{C_v} - \left(\frac{1}{\rho} - b\right)\frac{P_\infty}{C_v}, \tag{20}$$

where P, ρ, e, and q are, respectively, the pressure, the density, the specific internal energy, and the heat bond of the corresponding phase. γ, P_∞, q, and b are constant coefficients that defines the thermodynamic properties of the fluid. The coefficients for liquid water and steam used in our simulations are given in Table 1.

Software for simulation, visualisation and post-process. All the simulation were run with the open source code simulator LAMMPS [64, 65]. Visualisation and data post-processing were generated with the Open Source code OVITO [66].

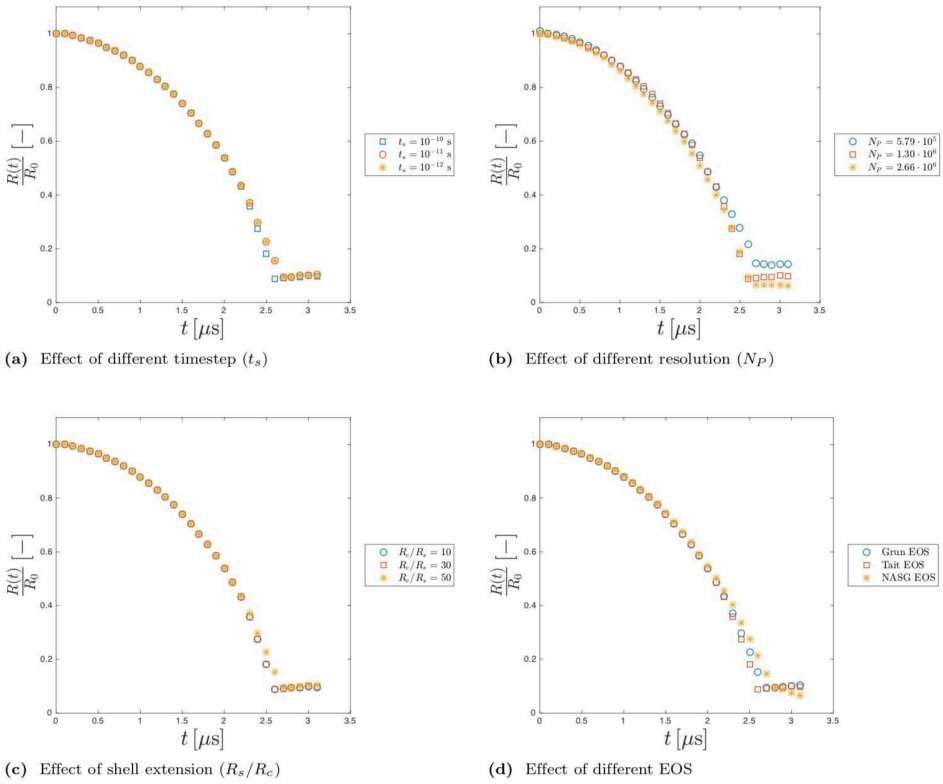
Hydrodynamic

Empty cavity. Different simulations have been run to assess the quality of results with respect of numerical parameters such as number of computational particles, kernel function and time step. Preliminary simulations have been run using both Lucy (Eq 11) and quintic spline (Eq 12) kernel functions obtaining similar results. Therefore, we chose the Lucy kernel over the over the quintic because it requires less computational cost because accounts for a smaller neighbour list. In all cases smoothing length and the dissipation factor are $h = 1.3 \cdot dL$, where dL is the initial particle spacing, and $\beta = 1$, coherent with literature in shock-wave problems [25, 42].

Table 1. NASG coefficients for liquid water and steam.

Coefficient	Liquid phase	Vapor phase
C_p [J kg^{-1} K^{-1}]	4285	1401
C_v [J kg^{-1} K^{-1}]	3610	955
γ [-]	1.19	1.47
P_∞ [Pa]	$7028 \cdot 10^5$	0
b [m^{3}kg^{-1}]	$6.61 \cdot 10^{-4}$	0
q [J kg^{-1}]	-1177788	2077616
q' [J kg^{-1} K^{-1}]	0	14317

https://doi.org/10.1371/journal.pone.0239830.t001



(a) Effect of different timestep (t_s)

(b) Effect of different resolution (N_P)

(c) Effect of shell extension (R_s/R_c)

(d) Effect of different EOS

Fig 2. Effect of different simulation parameters on the Hydrodynamic the SPH model. A: Effect of different timestep (t_s). B: Effect of different resolution (N_P). C: Effect of shell extension. R_s/R_c D:Effect of different EOS.

https://doi.org/10.1371/journal.pone.0239830.g002

Fig 2 shows the evolution of the dimensionless radius $R(t)/R_0$ of a collapsing cavity. The collapsing time obtatined with the SPH model is around 2.76μs, which is very close to $t_c = 2.70\mu$s the collapsing time obtained by solving the axisymmetric Rayleigh-Plesset (ARP) equation [67].

Fig 2 summarise the effect of different parameters on the simulation: Fig 2(a) shows the effect of different timestep, the higher timestep value, $t_s = 10^{-10}$ s, was chosen according to the CFL criterion. Fig 2(b) shows the effect of different resolution (number or particles). Fig 2(c) shows the effect of the extension of the shell region, expressed with the ratio R_C/R_S. Fig 2(d) shows the profile of the collapse obtained with different liquid EOS.

Note that our dimensionless radius does not goes to zero, but it rebounds, like the axisymmetric Raylerigh-Plesset equation, this is explainable with the particle nature of the SPH method: with particle methods, in fact, there is always a small spacing between particles.

Based on the analysis of this section we decided, for an empty cavity collapse, to use the parameters shown in Table 2.

All the parameter listed are chosen as the best compromise between speed an accuracy.

Table 2. Parameters used for simulating the empty cavity Rayleigh collapse.

Kernel	Water EOS	h	β	t_s [s]	N_p	R_s/R_c	c_0 [m s^{-1}]
Lucy Kernel	Tait	$1.3 \cdot dL$	1	10^{-10}	$1.30 \cdot 10^6$	0.3	1484

https://doi.org/10.1371/journal.pone.0239830.t002

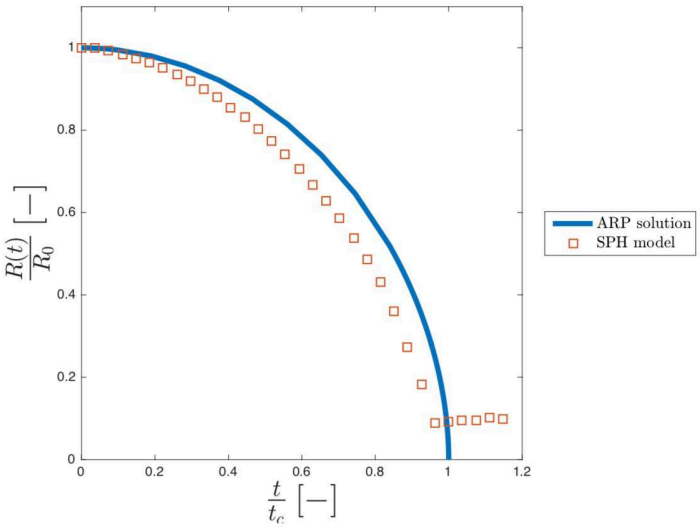


Fig 3. Effect of different timestep (t_s) on the energy trend.

https://doi.org/10.1371/journal.pone.0239830.g003

Vapour cavity. Fig 3 shows the trend of the dimensionless internal energy, e/e_0, versus dimensionless time for different timestep.

Where e_0 is the initial internal energy of particles. The timestep $t_s = 10^{-12}$ s was chosen since it is the largest timestep where the internal energy decreases after the collapse as required by the physics of the problem. The parameters used are summarised in Table 3.

Comparison with the axisymmetric Rayleigh-Plesset equation. The Hydrodynamic of the model is compared with the solution of the axisymmetric Rayleigh-Plesset (ARP) Eq [7] for both the empty and vapour cavity. Fig 4 shows the dimensionless radius, $R(t)/R_0$, plotted against dimensionless time, t/t_c, of our model against the solution of equation ARP for the empty collapse. R_0 is the initial radius of the cavity, $t_c = 2.76\mu$s is the collapsing time obtained with the ARP.

In our model, the cavity collapse slightly faster than the theoretical, leading to $t/t_c \approx 0.98$ instead of 1. This difference is explainable with the compressibility of the liquid [13]. The theoretical model assumes that water is perfectly incompressible, while the Tait EOS in the SPH model accounts for the compressibility of water. At these timescales, the compressibility cannot be neglected, and, from this point of view, our compressible SPH model should be more accurate than the theoretical, fully incompressible model. It is also important to highlight that, in our simulation, the parameter c_0 (sound speed in the medium) in the Tait EOS, we use is 1484 m/s, which is the actual speed of sound in water at 25˚C.

According to our calculations, the effect of the compressibility affects the rate of collapse. At the beginning, the compressibility produces a higher collapsing rate because a compressible fluid fills the void in the cavity faster than an incompressible fluid. As the cavity shrinks, however, the curvature of the cavity acts as an arch and the speed of the collapse slows down more than in the more rigid (incompressible) case. Overall, these two effects cancels each other out

Table 3. Parameters used for simulating the vapour cavity Rayleigh collapse.

Kernel	Water EOS	Gas EOS	h	β	$t_s[s]$	N_p	R_s/R_c	c_0 [m s^{-1}]
Lucy Kernel	Tait	Ideal gas	$1.3 \cdot dL$	1	10^{-10}	$1.30 \cdot 10^6$	0.3	1484

https://doi.org/10.1371/journal.pone.0239830.t003

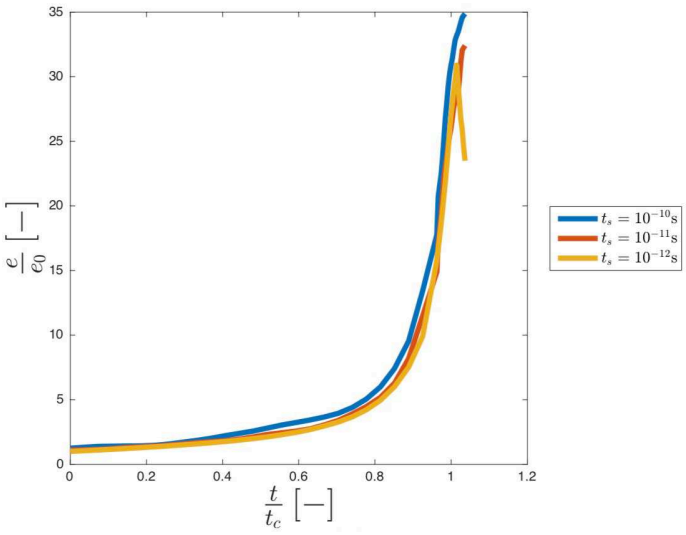


Fig 4. Dimensionless ratio (R/R_0) against dimensionless time (t/t_c) for both SPH (square dot) and ARP (continuum blue curve) for the empty cavity collapse.

https://doi.org/10.1371/journal.pone.0239830.g004

and the final collapsing time, is almost identical in the case of the theoretical Rayleigh-Plesset (incompressible) case and the SPH model based on the (compressible) Tait EOS.

Fig 5 shows the dimensionless radius, $R(t)/R_0$, plotted against dimensionless time, t/t_c, of our model against the solution of equation ARP for the vapour collapse.

The considerations done for the empty collapse are still valid for the vapour cavity case. However, additional discussion is required for the final phase of the collapse and the rebound phases: unlike the empty cavity (see Fig 6a) when the vapour cavity approaches the final phase of the collapse, the cavity loses the cylindrical symmetry (Fig 6b), and differs from the ARP.

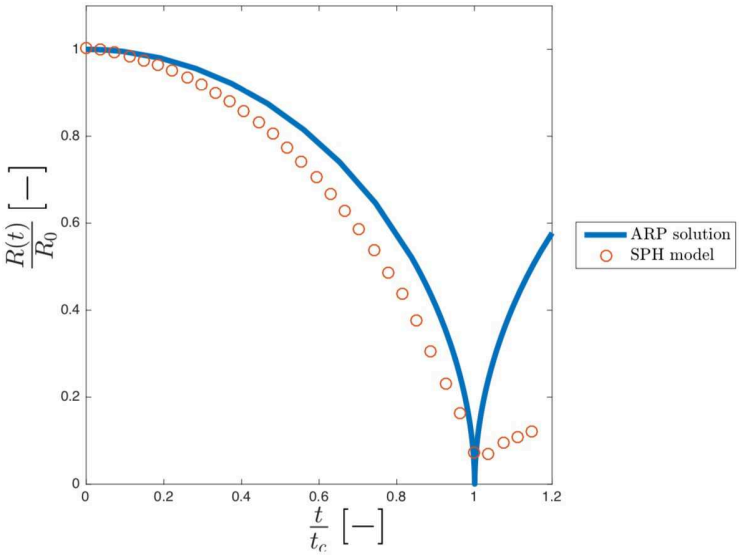
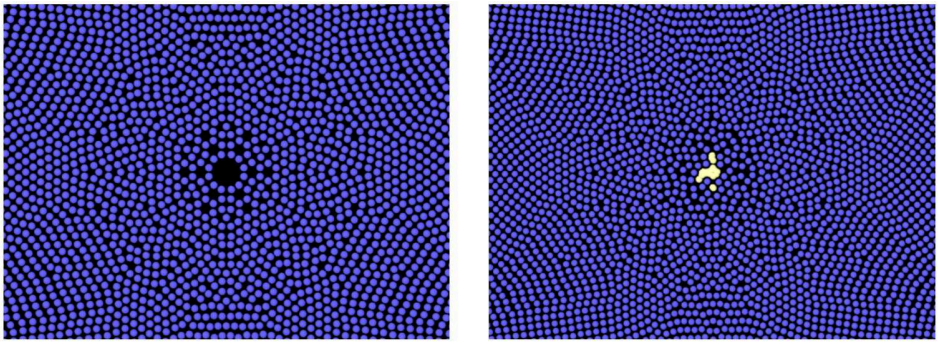


Fig 5. Dimensionless ratio (R/R_0) against dimensionless time (t/t_c) for both SPH (square dot) and ARP (continuum blue curve) for the vapour cavity collapse.

https://doi.org/10.1371/journal.pone.0239830.g005



(a) (b)

Fig 6. Final stage of the collapse for the empty cavity (a) and the vapour cavity (b).

https://doi.org/10.1371/journal.pone.0239830.g006

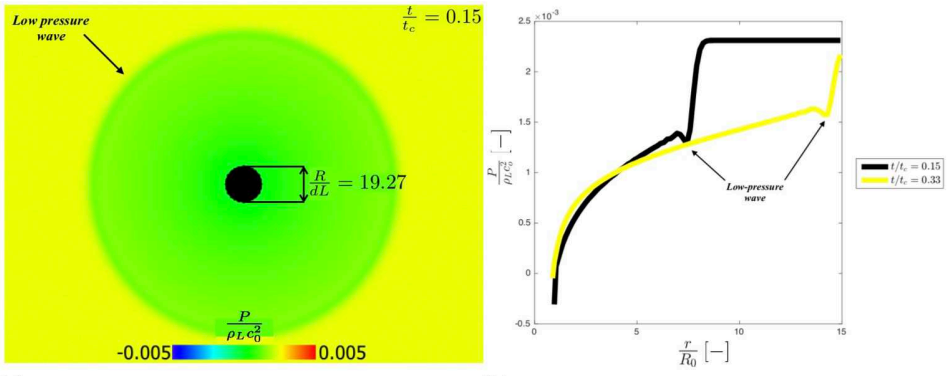
This "artificial" asymmetry in the rebound phase is attributable to low resolution occurring when, during the last stage of the collapse, the size of the caivty is comparable to the size of the smoothing length. However, this work focuses only on the bubble collapse (as usual in computer simulations of cavitation e.g. [7, 26, 59]) and the rebound phase is not considered.

Results and discussion

Pressure field

Pressure field in the liquid (empty cavity). Initially, at $t = 0$, the pressure is uniform along the domain. As the cavity shrinks, due to the pressure difference between the liquid and the cavity, the liquid starts to fill the cavity. This causes a decrement in pressure in the liquid generating a low-pressure wave that moves through the liquid phase (see Fig 7).

As the collapse proceed, a high-pressure area arises near the cavity border (see Fig 8a and 8b) that abruptly increases reaching the max at the collapse (see Fig 8c and 8d). Locally, the max pressure calculated is around 120 MPa. This value is one order of magnitude lower than the theoretical value calculated by Hickling and Plesset [68] for the 3D collapse, but this difference is consistent with the fact that our model refers to a 2D collapse [67, 69]. This is also reflected in the difference in the collapse time between 2D and 3D [9, 70]



(a) (b)

Fig 7. Dimensionless pressure field in the liquid phase fort/t_c = 0.15 and R/dL = 19.27 (a); Dimensionless pressure spatial trend for t/t_c = 0.15 and t/t_c = 0.33 (b).

https://doi.org/10.1371/journal.pone.0239830.g007

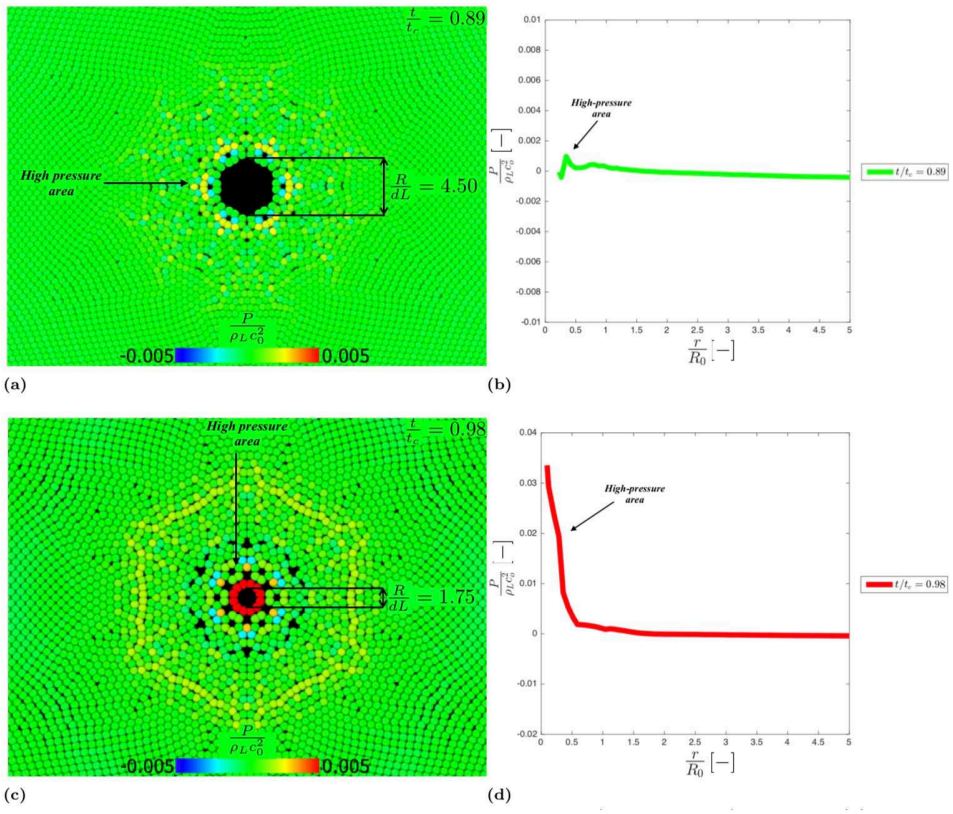


Fig 8. Dimensionless pressure field in the liquid phase fort/t_c = 0.89, R/dL = 4.50 (a) andt/t_c = 0.98, R/dL = 1.75 (d); Dimensionless pressure spatial trend for t/t_c = 0.89 (b) and t/t_c = 0.98 (d).

https://doi.org/10.1371/journal.pone.0239830.g008

The collapse generates a high-pressure wave (Fig 9 and 9b) that moves away form the cavity (Fig 9c and 9d). There is a theoretical reason for the hexagonal patterns in Figs 8 and 9, which is discussed in the next section.

With the absence of heat diffusivity the presence of the vapour in the cavity does not affect significantly the pressure in the liquid and, therefore, pressure fields for the vapour cavity collapse are not shown here.

Acoustic diffraction. As mentioned in the previous section, when the empty cavity reaches the minimum radius, a high-pressure shock wave is generated and propagates in the liquid phase. After the wave bounces back, it loses its spherical symmetry and assumes an unphysical hexagonal symmetry.

This is a numerical artefact and depends on the fact that, below a certain raito R/dL, the initial particle resolution is not adequate to correctly describe the cavity shape (the reader can compare Fig 7, where R/dL = 19.27, with Fig 8, where R/dL = 4.50-1.75). The cavity assumes a hexagonal shape (see Fig 8a), which is related with initial hexagonal particle distribution of the model. When the high-pressure shock wave bounces back, therefore, it propagates from a hexagonal cavity rather than a circular one.

This behaviour closely resembles light diffraction from a hexagonal aperture, Fig 10. Light diffraction, in fact, follows specific patterns [71] defined by the shape of the aperture.

Fig 10 shows the similarity between the light intensity pattern generated by diffraction trough a hexagonal opening and the pressure intensity pattern of the wave generated at the

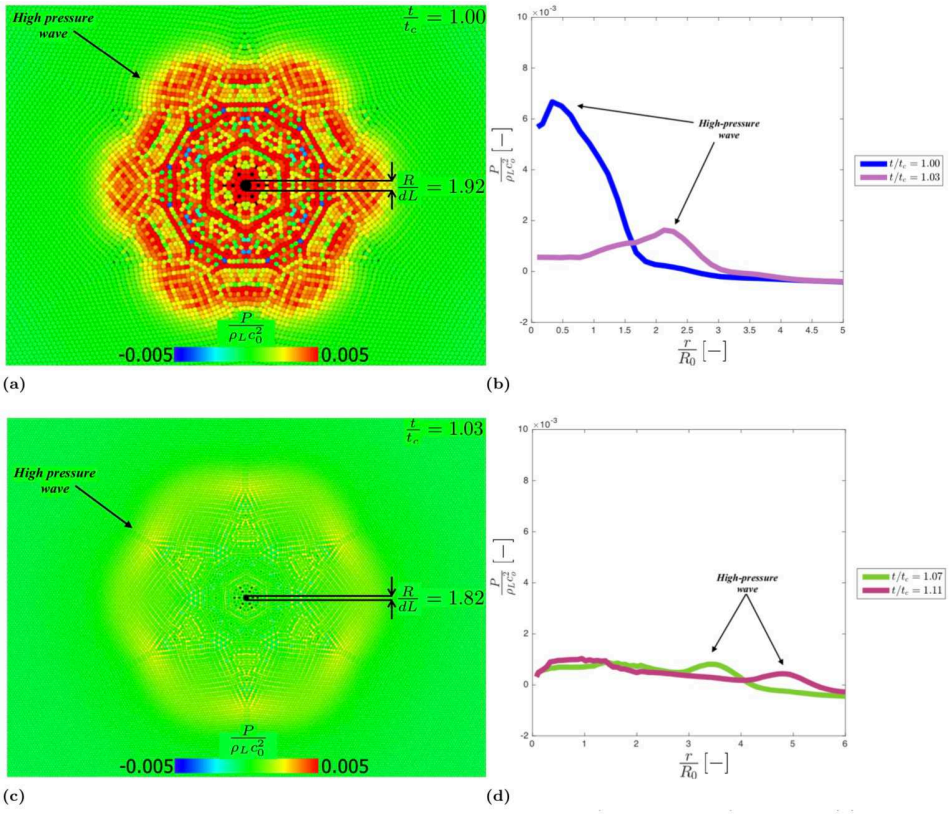


Fig 9. Dimensionless pressure field in the liquid phase fort/t_c = 1.00, R/dL = 1.92(a) andt/t_c = 1.03, R/dL = 1.82 (d); Dimensionless pressure spatial trend for t/t_c = 1.00, t/t_c = 1.03(b) and t/t_c = 1.07, t/t_c = 1.11 (d).

https://doi.org/10.1371/journal.pone.0239830.g009

collapse of the cavity. The pressure peaks and valleys in Fig 10b (and Fig 9a), therefore, are the results of Fresnel like positive and negative interference of the interfering diffracted waves rather than the result of effect of numerical instability.

This issue, however, only occurs at the end of the collapse, when the size of the cavity is comparable to the smoothing length and does not affect the overall collapsing time.

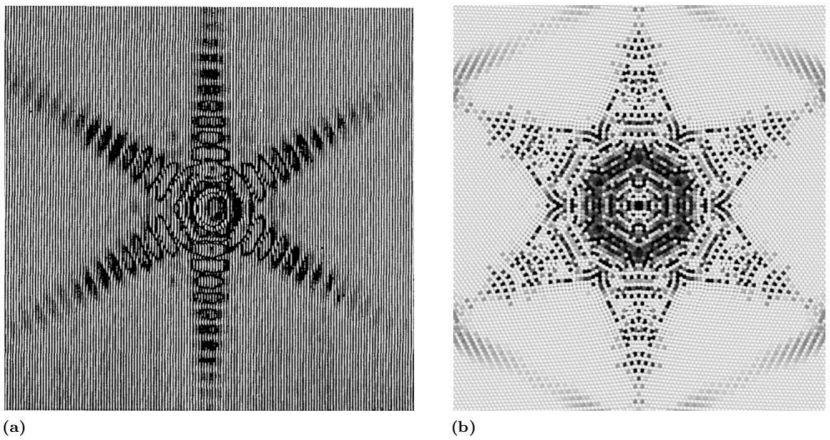


Fig 10. Comparison between light intensity for a hexagonal aperture [71] (a) and pressure intensity calculated with our model (b).

https://doi.org/10.1371/journal.pone.0239830.g010

Thermal dffects

During the collapse, the compression of gas in the bubble generates heat. This heat, in turn, can affect the dynamic of the collapsing bubble [14]. When thermal effects are absent, or negligible, the collapse is "inertially controlled" as in the previous Section. When the thermal effects are not negligible, the collapse can be "thermally controlled". In a thermally controlled collapse, the bubble dynamic differs from the inertially controlled because the thermal terms in the ARP equation are not negligible.

Two scenarios are analysed. In the adiabatic collapse, only the first term of Eq 9 is accounted for. In the heat diffusive collapse, both term are enabled to model heat transfer between gas and liquid.

Finally, the temperature peak in the gas cavity is investigated in relation to the ratio between the characteristic time of collapse and the characteristic time of heat transfer.

Adiabatic collapse. The average pressure and the temperature in the vapour cavity increase during the collapse (see Fig 11), locally reaching a max $P \approx 40$ MPa and $T \approx 10000$ K. Those values, despite some difference in the operating conditions, are comparable to those measured by Obreschkow et al. [72].

The pressure and temperature field distribute differently in the cavity:

1. In the first stage of the collapse, the interaction between gas and fluid results in a rapid increment of pressure at the cavity interface (Fig 12a) generating a shock wave inside the cavity. Later, because of the combined effect of cavity compression and shock wave propagation, the pressure increases in the centre of bubble (Fig 12b) becoming almost uniform at the collapse.
2. Similarly to the pressure, the temperature increases at the cavity interface during the first stages of the collapse (see Fig 12c). The absence of heat diffusion mostly affects the final stage of the collapse: the internal energy does not diffuse and heat is confined and accumulated at the cavity interface (Fig 12d).

(Heat) Diffusive collapse. Our results show that time of the collapse is not significantly affected by the presence of heat transfer in the model. However, the pressure and temperature peak inside the bubble decreases with respect to the adiabatic case, see Fig 13.

Also the pressure and temperature fields inside the cavity change:

1. In the first phase of the collapse, the pressure field in the cavity remains uniform (Fig 14a). Approaching the final stage of the collapse, the pressure is slightly lower at the cavity interface, Fig 14b, because of the presence of the diffusive heat transfer mechanism. In fact, the pressure of an ideal gas is function of both the density and the internal energy, as shown by Eq 17. The presence of a high temperature gradient at the cavity interface greatly reduces the internal energy of the particles in that area.
2. Similarly to the adiabatic case, in the first stage the interaction between gas and the fluid tend to increment the temperature near the cavity interface, see Fig 14c. However, in this case the heat generated at the interface diffuses both towards the centre of the cavity and into the liquid. This leads to higher temperatures at the centre than at the interface (see Fig 14d).

By comparing Fig 14 with Fig 12, it is clear that, despite the total collapsing time is almost the same, the heat exchange mechanism has important consequences on both temperature

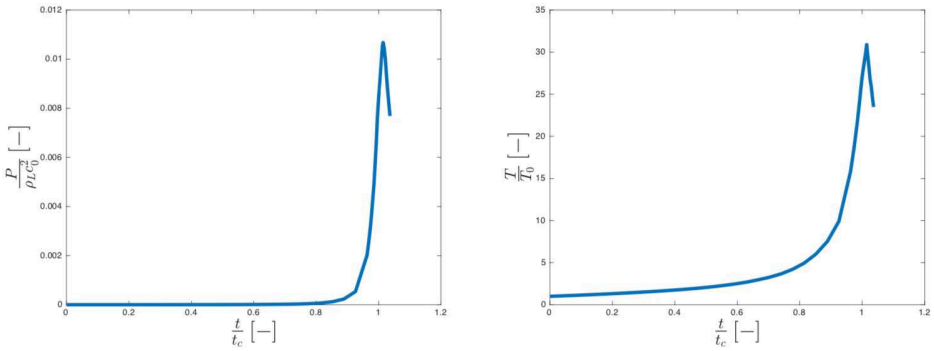


Fig 11. Dimensionless pressure trend (a) and dimensionless temperature (b) in the gas phase for the adiabatic collapse.

https://doi.org/10.1371/journal.pone.0239830.g011

and pressure in the cavity. Our results, therefore, show that adiabatic conditions, despite being often used in the literature [30, 31, 73–75], are not always realistic. The collapse generates great amount of heat and high temperatures are reached in the cavity. Despite the small timescale of the process, temperature in the cavity rapidly grows and the heat transfer from the cavity interface to the surroundings cannot be neglected.

Effect of thermal diffusivity on the temperature peak. In the previous section, we calculated a specific case, where the liquid is water, the gas is water vapour, $\Delta P = P_L - p_G = 5MPa$ and $R_0 = 100\mu$m. In this section, we study how different parameters and initial conditions would affect the temperature rise T/T_0 in the cavity. In order to simplify the study, we perform

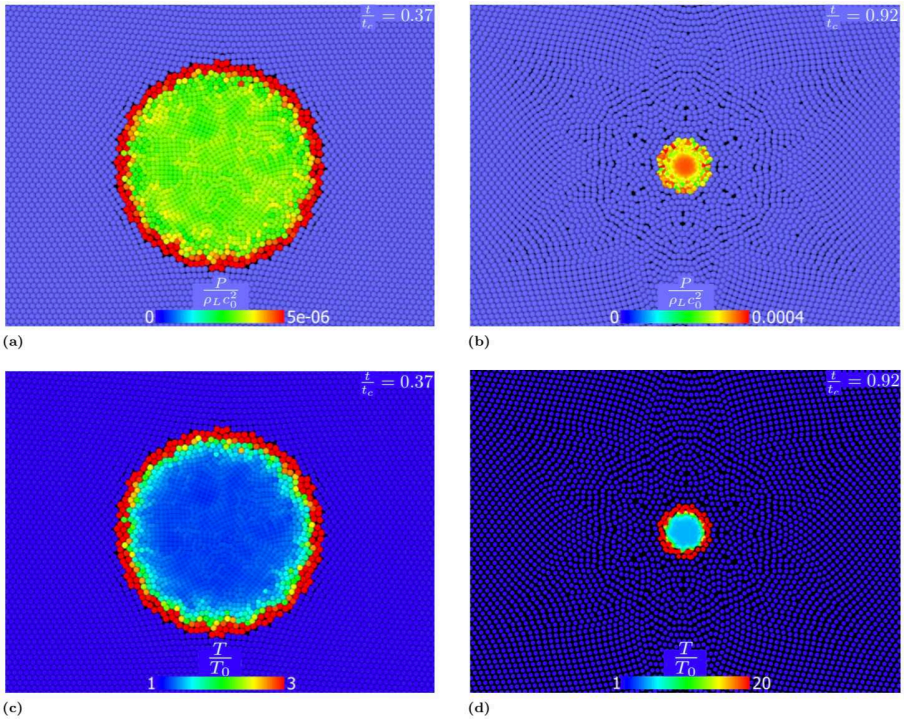


Fig 12. Dimensionless pressure field in the cavity fort/t_c = 0.37 (a) and t/t_c = 0.92 (b) and dimensionless temperature field for t/t_c = 0.37 (c) and t/t_c = 0.92 (d) for adiabatic collapse.

https://doi.org/10.1371/journal.pone.0239830.g012

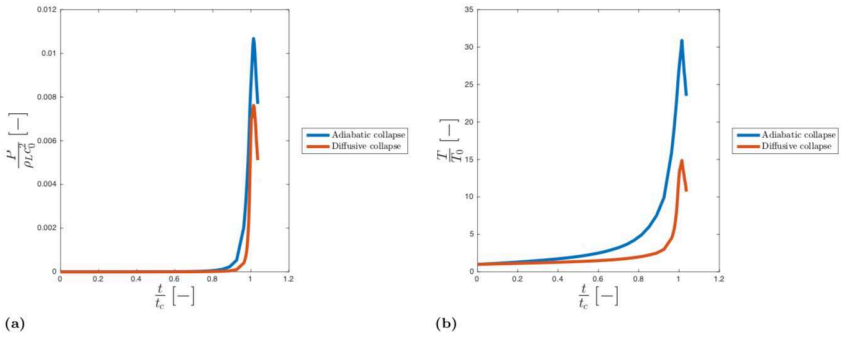


Fig 13. Comparison between: Dimensionless pressure trend (a) and dimensionless temperature (b) of the gas phase for adiabatic (blue) and diffusive (red) collapse.

https://doi.org/10.1371/journal.pone.0239830.g013

a dimensional analysis of our system to reduce the number of significant parameters. Assuming that the collapse depends on ΔP, ρ_L, $\alpha_{L,G} = (\alpha_L + \alpha_G)/2$ and R_0, and using the Buckingham π theorem, it is possible to determine that the system depends on two fundamental dimensionless groups

$$\Pi_1 = \frac{R}{\alpha_{L,G}}\sqrt{\frac{\Delta P}{\rho_l}}, \tag{21}$$

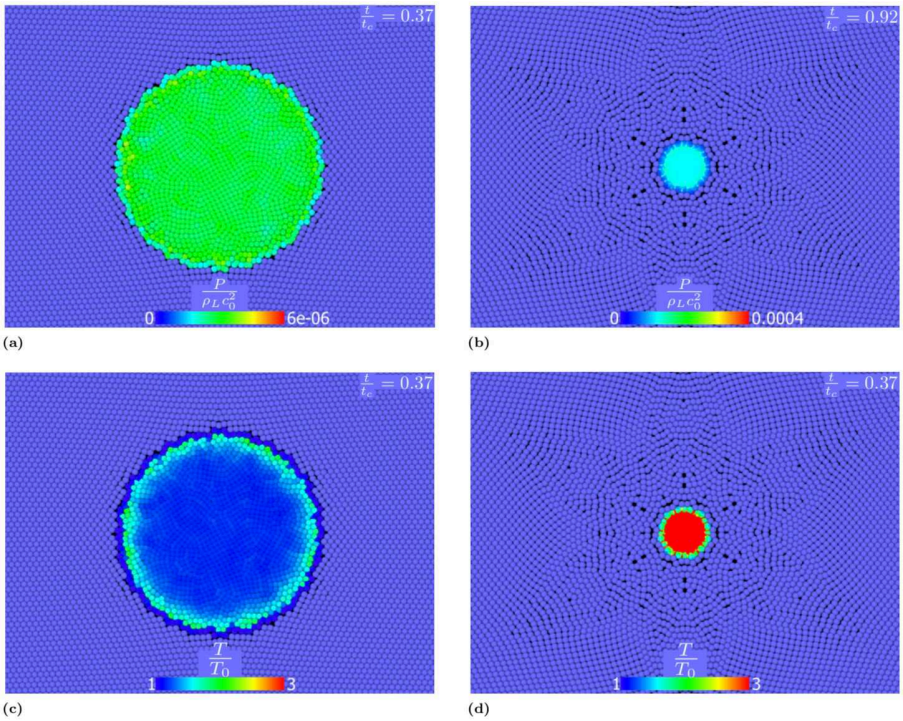


Fig 14. Dimensionless pressure field in the cavity fort/t_c = 0.37 (a) and t/t_c = 0.92 (b) and dimensionless temperature field for t/t_c = 0.37 (c) and t/t_c = 0.92 (d) for diffusive collapse. The heat diffusivity in the liquid is $\alpha_L = 1.48 \cdot 10^{-7}$ m^2/s (liquid water) in the gas $\alpha_G = 4.09{\cdot}10^{-4}$ m^2/s (water vapour).

https://doi.org/10.1371/journal.pone.0239830.g014

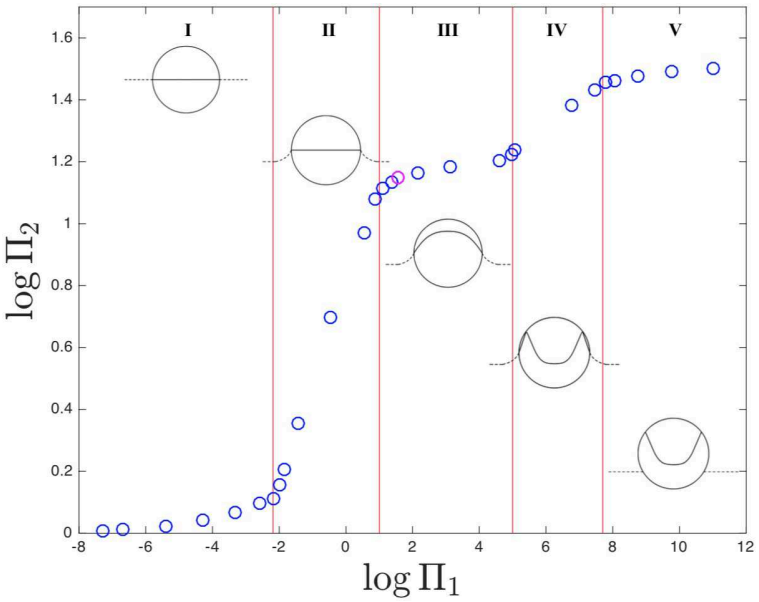


Fig 15. Temperature peak (T_{max}/T_0) as a function of Π_1.

https://doi.org/10.1371/journal.pone.0239830.g015

$$\Pi_2 = \frac{T}{T_0}. \tag{22}$$

The dimensionless group Π_1 can also been seen as the ration between the characteristic heat diffusion time scale of the process, $R^2/\alpha_{L,G}$, and the Rayleigh collapsing time, $R\sqrt{\rho_L/\Delta P}$:

$$\Pi_1 = \frac{\tau_d}{t_c} = \frac{R^2}{\alpha_{L,G}} \cdot \frac{1}{R}\sqrt{\frac{\Delta P}{\rho_l}}, \tag{23}$$

When $t_c \ll \tau_d$, the collapse is faster than the characteristic time of heat transfer, the heat generated is trapped in the cavity, and the process can be considered adiabatic. When $t_c \gg \tau_d$, the characteristic time of heat transfer is smaller that the collapsing time and the process can be considered isotherm.

In Fig 15 T_{max}/T_0 is plotted against different values of Π_1.

The "magenta" point represent the collapse analysed in the previous section with Π_1 = 34.76.

Conclusion

This work proposes the first SPH model of a collapsing cavity filled with non condensable gas coupled with the heat transfer mechanism.

The aim of the work is to understand the role of diffusive heat transfer during the Rayleigh collapse. This was achieved by introducing the dimensionless group, Π_1. Π_1 defined as ratio between the characteristic time of collapse and the characteristic time of thermal diffusion.

In Fig 15 five regions are identified. For each of these regions the temperature field of the gas-liquid system distributes differently:

- *I* region ($0 < \Pi_1 < 6.5 \cdot 10^{-3}$): in this region, both the gas and liquid behave isothermally. As a result of this, the liquid drains all the energy developed by the gas.
- *II* region ($6.5 \cdot 10^{-3} < \Pi_1 < 10$): in this region, the gas behaves isothermally while the liquid shows a temperature profile. The energy rapidly diffuses inside the cavity, flattening the temperature profile in the cavity.
- *III* region ($10 < \Pi_1 < 1 \cdot 10^5$): in this region, neither the gas nor the cavity are adiabatic. This scenario is described in (heat) diffusive collapse section and by Fig 14.
- *IV* region ($1 \cdot 10^5 < \Pi_1 < 5 \cdot 10^7$): in this region, the gas in the cavity behaves adiabatically, but the liquid does not. Therefore, part of the energy generated at the interface is transferred to the liquid phase.
- *V* region ($\Pi_1 > 5 \cdot 10^7$): in this region both the gas in the cavity and the liquid behave adiabatically. All the energy generated by the collapse is trapped in the cavity and the temperature increment is concentrated in the cavity interface (see Fig 12).

In brief, this analysis shows that for $\Pi_1 > 5 \cdot 10^7$ the collapse can be considered adiabatic. At smaller Π_1, the heat generated at the cavity interface is taken away by the liquid phase, or diffuses in the cavity, homogenising the temperature field. When $\Pi_1 < 6.5 \cdot 10^{-3}$, all the heat generated in the collapse is drained by the liquid and the collapse can be considered isotherm.

This shows that, despite the short timescale, the presence of the heat transfers mechanism leads to a temperature peak drop of around 50% compared to the adiabatic case. This suggest that the adiabatic assumption for the Rayleigh collapse could leads to a not reliable pressure and temperature peak estimation and its use should be properly justified.

Supporting information

S1 File. Input file for LAMMPS. All the results presented in this work were obtained with this input file.
(LMP)

Author Contributions

Conceptualization: Andrea Albano, Alessio Alexiadis.

Data curation: Andrea Albano.

Funding acquisition: Alessio Alexiadis.

Supervision: Alessio Alexiadis.

Validation: Andrea Albano.

Writing – original draft: Andrea Albano.

Writing – review & editing: Alessio Alexiadis.

References

1. Arndt RE. Cavitation in fluid machinery and hydraulic structures. Annual Review of Fluid Mechanics. 1981; 13(1):273–326. https://doi.org/10.1146/annurev.fl.13.010181.001421
2. Knapp R, Daily J. and Hammitt, FG, Cavitation; 1970.
3. Giannadakis E, Gavaises M, Arcoumanis C. Modelling of cavitation in diesel injector nozzles. Journal of Fluid Mechanics. 2008; 616:153–193. https://doi.org/10.1017/S0022112008003777

4. Walmsley A, Laird W, Williams A. Dental plaque removal by cavitational activity during ultrasonic scaling. Journal of clinical periodontology. 1988; 15(9):539–543. https://doi.org/10.1111/j.1600-051X.1988.tb02126.x PMID: 2848873
5. Packer M, Fishkind WJ, Fine IH, Seibel BS, Hoffman RS. The physics of phaco: a review. Journal of Cataract & Refractive Surgery. 2005; 31(2):424–431. https://doi.org/10.1016/j.jcrs.2004.11.027 PMID: 15767168
6. Mourad PD, Roberts FA, McInnes C. Synergistic use of ultrasound and sonic motion for removal of dental plaque bacteria. Compendium of continuing education in dentistry (Jamesburg, NJ: 1995). 2007; 28 (7):354–358. PMID: 17687897
7. Johnsen E, Colonius T. Numerical simulations of non-spherical bubble collapse. Journal of fluid mechanics. 2009; 629:231–262. https://doi.org/10.1017/S0022112009006351 PMID: 19756233
8. Besant WH. A treatise on hydrostatics and hydrodynamics. Deighton, Bell; 1859.
9. Rayleigh L. VIII. On the pressure developed in a liquid during the collapse of a spherical cavity. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science. 1917; 34(200):94–98. https://doi.org/10.1080/14786440808635681
10. Plesset MS. The dynamics of cavitation bubbles. Journal of applied mechanics. 1949; 16:277–282.
11. Epstein D, Keller JB. Expansion and contraction of planar, cylindrical, and spherical underwater gas bubbles. The Journal of the Acoustical Society of America. 1972; 52(3B):975–980. https://doi.org/10.1121/1.1913203
12. Plesset MS, Prosperetti A. Bubble dynamics and cavitation. Annual review of fluid mechanics. 1977; 9 (1):145–185. https://doi.org/10.1146/annurev.fl.09.010177.001045
13. Keller JB, Miksis M. Bubble oscillations of large amplitude. The Journal of the Acoustical Society of America. 1980; 68(2):628–633. https://doi.org/10.1121/1.384720
14. Brennen CE. Cavitation and bubble dynamics. Cambridge University Press; 2014.
15. Kudryashov NA, Sinelshchikov DI. Analytical solutions of the Rayleigh equation for empty and gas-filled bubble. Journal of Physics A: Mathematical and Theoretical. 2014; 47(40):405202. https://doi.org/10.1088/1751-8113/47/40/405202
16. Kudryashov NA, Sinelshchikov DI. Analytical solutions for problems of bubble dynamics. Physics Letters A. 2015; 379(8):798–802. https://doi.org/10.1016/j.physleta.2014.12.049
17. Lauterborn W, Bolle H. Experimental investigations of cavitation-bubble collapse in the neighbourhood of a solid boundary. Journal of Fluid Mechanics. 1975; 72(2):391–399. https://doi.org/10.1017/S0022112075003448
18. Vogel A, Lauterborn W, Timm R. Optical and acoustic investigations of the dynamics of laser-produced cavitation bubbles near a solid boundary. Journal of Fluid Mechanics. 1989; 206:299–338. https://doi.org/10.1017/S0022112089002314
19. Philipp A, Lauterborn W. Cavitation erosion by single laser-produced bubbles. Journal of Fluid Mechanics. 1998; 361:75–116. https://doi.org/10.1017/S0022112098008738
20. TOMITA Y, SHIMA A. On the behavior of a spherical bubble and the impulse pressure in a viscous compressible liquid. Bulletin of JSME. 1977; 20(149):1453–1460. https://doi.org/10.1299/jsme1958.20.1453
21. Fujikawa S, Akamatsu T. Effects of the non-equilibrium condensation of vapour on the pressure wave produced by the collapse of a bubble in a liquid. Journal of Fluid Mechanics. 1980; 97(3):481–512. https://doi.org/10.1017/S0022112080002662
22. Plesset MS, Chapman RB. Collapse of an initially spherical vapour cavity in the neighbourhood of a solid boundary. Journal of Fluid Mechanics. 1971; 47(2):283–290. https://doi.org/10.1017/S0022112071001058
23. Blake J, Taib B, Doherty G. Transient cavities near boundaries. Part 1. Rigid boundary. Journal of Fluid Mechanics. 1986; 170:479–497. https://doi.org/10.1017/S0022112086000988
24. Klaseboer E, Turangan C, Fong SW, Liu TG, Hung KC, Khoo BC. Simulations of pressure pulse–bubble interaction using boundary element method. Computer methods in applied mechanics and engineering. 2006; 195(33-36):4287–4302. https://doi.org/10.1016/j.cma.2005.08.014
25. Liu GR, Liu MB. Smoothed particle hydrodynamics: a meshfree particle method. World scientific; 2003.
26. Joshi S, Franc JP, Ghigliotti G, Fivel M. SPH modelling of a cavitation bubble collapse near an elasto-visco-plastic material. Journal of the Mechanics and Physics of Solids. 2019; 125:420–439. https://doi.org/10.1016/j.jmps.2018.12.016
27. Joshi S, Franc JP, Ghigliotti G, Fivel M. An axisymmetric solid SPH solver with consistent treatment of particles close to the symmetry axis. Computational Particle Mechanics. 2020; p. 1–15.

28. Albano A, Alexiadis A. Interaction of Shock Waves with Discrete Gas Inhomogeneities: A Smoothed Particle Hydrodynamics Approach. Applied Sciences. 2019; 9(24):5435. https://doi.org/10.3390/app9245435

29. Kim KH, Chahine G, Franc JP, Karimi A. Advanced experimental and numerical techniques for cavitation erosion prediction. vol. 106. Springer; 2014.

30. Merouani S, Hamdaoui O, Rezgui Y, Guemini M. Theoretical estimation of the temperature and pressure within collapsing acoustical bubbles. Ultrasonics sonochemistry. 2014; 21(1):53–59. https://doi.org/10.1016/j.ultsonch.2013.05.008 PMID: 23769748

31. Fostiropoulos SR, Malgarinos I, Strotos G, Nikolopoulos N, Kakaras E, Koukouvinis P, et al. Role of heat transfer in bubble dynamics neglecting phase change. A numerical study. In: Ilass Europe. 28th european conference on Liquid Atomization and Spray Systems. Editorial Universitat Politècnica de València; 2017. p. 904–911.

32. Gingold RA, Monaghan JJ. Smoothed particle hydrodynamics: theory and application to non-spherical stars. Monthly notices of the royal astronomical society. 1977; 181(3):375–389. https://doi.org/10.1093/mnras/181.3.375

33. Lucy LB. A numerical approach to the testing of the fission hypothesis. The astronomical journal. 1977; 82:1013–1024. https://doi.org/10.1086/112164

34. Swegle J, et al. Report at Sandia National Laboratories; 1992.

35. Monaghan J, Kocharyan A. SPH simulation of multi-phase flow. Computer Physics Communications. 1995; 87(1-2):225–235. https://doi.org/10.1016/0010-4655(94)00174-Z

36. Zhu Y, Fox PJ, Morris JP. A pore-scale numerical model for flow through porous media. International journal for numerical and analytical methods in geomechanics. 1999; 23(9):881–904. https://doi.org/10.1002/(SICI)1096-9853(19990810)23:9%3C881::AID-NAG996%3E3.0.CO;2-K

37. Vignjevic R, Campbell J. Review of development of the smooth particle hydrodynamics (SPH) method. In: Predictive modeling of dynamic processes. Springer; 2009. p. 367–396.

38. Shadloo MS, Oger G, Le Touzé D. Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: Motivations, current state, and challenges. Computers & Fluids. 2016; 136:11–34. https://doi.org/10.1016/j.compfluid.2016.05.029

39. Liu M, Liu G, Zong Z, Lam K. Computer simulation of high explosive explosion using smoothed particle hydrodynamics methodology. Computers & Fluids. 2003; 32(3):305–322. https://doi.org/10.1016/S0045-7930(01)00105-0

40. Liu M, Liu G, Lam K, Zong Z. Smoothed particle hydrodynamics for numerical simulation of underwater explosion. Computational Mechanics. 2003; 30(2):106–118. https://doi.org/10.1007/s00466-002-0371-6

41. Monaghan J, Gingold RA. Shock simulation by the particle method SPH. Journal of computational physics. 1983; 52(2):374–389. https://doi.org/10.1016/0021-9991(83)90036-0

42. Morris J, Monaghan J. A switch to reduce SPH viscosity. Journal of Computational Physics. 1997; 136(1):41–50. https://doi.org/10.1006/jcph.1997.5690

43. Johnson GR, Stryk RA, Beissel SR. SPH for high velocity impact computations. Computer methods in applied mechanics and engineering. 1996; 139(1-4):347–373. https://doi.org/10.1016/S0045-7825(96)01089-4

44. Ulrich C, Rung T. Sph modelling of water/soil-suspension flows. In: 5th International SPHERIC Workshop. BD Rogers. Manchester, UK, 5th Inernational SPHERC Workshop Local Organising Committee; 2010. p. 61–68.

45. Violeau D, Rogers BD. Smoothed particle hydrodynamics (SPH) for free-surface flows: past, present and future. Journal of Hydraulic Research. 2016; 54(1):1–26. https://doi.org/10.1080/00221686.2015.1119209

46. Ehigiamusoe NN, Maxutov S, Lee YC. Modeling surface tension of a two-dimensional droplet using smoothed particle hydrodynamics. International Journal for Numerical Methods in Fluids. 2018; 88(7):334–346. https://doi.org/10.1002/fld.4663 PMID: 16196705

47. Nasiri H, Jamalabadi MYA, Sadeghi R, Safaei MR, Nguyen TK, Shadloo MS. A smoothed particle hydrodynamics approach for numerical simulation of nano-fluid flows. Journal of Thermal Analysis and Calorimetry. 2019; 135(3):1733–1741. https://doi.org/10.1007/s10973-018-7022-4

48. Ng K, Ng Y, Sheu T, Alexiadis A. Assessment of Smoothed Particle Hydrodynamics (SPH) models for predicting wall heat transfer rate at complex boundary. Engineering Analysis with Boundary Elements. 2020; 111:195–205. https://doi.org/10.1016/j.enganabound.2019.10.017

49. Alexiadis A. The discrete multi-hybrid system for the simulation of solid-liquid flows. PloS one. 2015; 10(5):e0124678. https://doi.org/10.1371/journal.pone.0124678 PMID: 25961561

50. Ariane M, Allouche MH, Bussone M, Giacosa F, Bernard F, Barigou M, et al. Discrete multi-physics: A mesh-free model of blood flow in flexible biological valve including solid aggregate formation. PloS one. 2017; 12(4):e0174795. https://doi.org/10.1371/journal.pone.0174795 PMID: 28384341

51. Ariane M, Kassinos S, Velaga S, Alexiadis A. Discrete multi-physics simulations of diffusive and convective mass transfer in boundary layers containing motile cilia in lungs. Computers in biology and medicine. 2018; 95:34–42. https://doi.org/10.1016/j.compbiomed.2018.01.010 PMID: 29438794

52. Rahmat A, Barigou M, Alexiadis A. Numerical simulation of dissolution of solid particles in fluid flow using the SPH method. International Journal of Numerical Methods for Heat & Fluid Flow. 2019. https://doi.org/10.1108/HFF-05-2019-0437

53. Alexiadis A. Deep multiphysics: Coupling discrete multiphysics with machine learning to attain self-learning in-silico models replicating human physiology. Artificial intelligence in medicine. 2019; 98:27–34. https://doi.org/10.1016/j.artmed.2019.06.005 PMID: 31521250

54. Morris JP. Analysis of smoothed particle hydrodynamics with applications. Monash University Australia; 1996.

55. Franc JP, Riondet M, Karimi A, Chahine GL. Impact load measurements in an erosive cavitating flow. Journal of Fluids Engineering. 2011; 133(12):121301. https://doi.org/10.1115/1.4005342

56. Storey BD, Szeri AJ. Water vapour, sonoluminescence and sonochemistry. Proceedings of the Royal Society of London Series A: Mathematical, Physical and Engineering Sciences. 2000; 456(1999):1685–1709. https://doi.org/10.1098/rspa.2000.0582

57. Benjamin TB. Pressure waves from collapsing cavities. In: 2nd Symposium on Naval Hydrodynamics; 1958. p. 207–229.

58. Hickling R. Effects of thermal conduction in sonoluminescence. The Journal of the Acoustical Society of America. 1963; 35(7):967–974. https://doi.org/10.1121/1.1918641

59. Beig S, Aboulhasanzadeh B, Johnsen E. Temperatures produced by inertially collapsing bubbles near rigid surfaces. Journal of Fluid Mechanics. 2018; 852:105–125. https://doi.org/10.1017/jfm.2018.525

60. Monaghan JJ. Simulating free surface flows with SPH. Journal of computational physics. 1994; 110 (2):399–406. https://doi.org/10.1006/jcph.1994.1034

61. Rice MH, Walsh JM. Equation of state of water to 250 kilobars. The Journal of Chemical Physics. 1957; 26(4):824–830. https://doi.org/10.1063/1.1743415

62. Shin Y, Lee M, Lam K, Yeo K. Modeling mitigation effects of watershield on shock waves. Shock and Vibration. 1998; 5(4):225–234. https://doi.org/10.1155/1998/782032

63. Le Métayer O, Saurel R. The noble-abel stiffened-gas equation of state. Physics of Fluids. 2016; 28 (4):046102. https://doi.org/10.1063/1.4945981

64. Plimpton S. Fast parallel algorithms for short-range molecular dynamics. Sandia National Labs., Albuquerque, NM (United States); 1993.

65. Ganzenmüller GC, Steinhauser MO, Van Liedekerke P, Leuven KU. The implementation of Smooth Particle Hydrodynamics in LAMMPS. Paul Van Liedekerke Katholieke Universiteit Leuven. 2011; 1:1–26.

66. Stukowski A. Visualization and analysis of atomistic simulation data with OVITO–the Open Visualization Tool. Modelling and Simulation in Materials Science and Engineering. 2009; 18(1):015012. https://doi.org/10.1088/0965-0393/18/1/015012 PMID: 27114926

67. Chen X. Simulation of 2D cavitation bubble growth under shear flow by lattice Boltzmann model. Communications in Computational Physics. 2010; 7(1):212. https://doi.org/10.4208/cicp.2009.09.015

68. Hickling R, Plesset MS. Collapse and rebound of a spherical bubble in water. The Physics of Fluids. 1964; 7(1):7–14. https://doi.org/10.1063/1.1711058

69. Nair P, Tomar G. Simulations of gas-liquid compressible-incompressible systems using SPH. Computers & Fluids. 2019; 179:301–308. https://doi.org/10.1016/j.compfluid.2018.11.015

70. Obreschkow D, Bruderer M, Farhat M. Analytical approximations for the collapse of an empty spherical bubble. Physical Review E. 2012; 85(6):066303. https://doi.org/10.1103/PhysRevE.85.066303

71. Smith RC, Marsh JS. Diffraction patterns of simple apertures. JOSA. 1974; 64(6):798–803. https://doi.org/10.1364/JOSA.64.000798

72. Obreschkow D, Tinguely M, Dorsaz N, Kobel P, De Bosset A, Farhat M. The quest for the most spherical bubble: experimental setup and data overview. Experiments in Fluids. 2013; 54(4):1503. https://doi.org/10.1007/s00348-013-1503-9

73. Moss WC, Levatin JL, Szeri AJ. A new damping mechanism in strongly collapsing bubbles. Proceedings of the Royal Society of London Series A: Mathematical, Physical and Engineering Sciences. 2000; 456(2004):2983–2994. https://doi.org/10.1098/rspa.2000.0649

74. Szeri AJ, Storey BD, Pearson A, Blake JR. Heat and mass transfer during the violent collapse of non-spherical bubbles. Physics of Fluids. 2003; 15(9):2576–2586. https://doi.org/10.1063/1.1595647

75. Niazi S, Hashemabadi SH, Razi MM. CFD simulation of acoustic cavitation in a crude oil upgrading sonoreactor and prediction of collapse temperature and pressure of a cavitation bubble. Chemical Engineering Research and Design. 2014; 92(1):166–173. https://doi.org/10.1016/j.cherd.2013.07.002

Chapter 7

Non-symmetrical collapse of an empty cylindrical cavity studied with Smoothed Particle Hydrodynamics

In this Chapter the model shown in Chapter 6 is extended to simulate a non-symmetrical collapse that occurs when the cavity collapses near a solid surfaces.

This Chapter has been published in *Applied Sciences* as:

Albano A, Alexiadis A. Non-Symmetrical Collapse of an Empty Cylindrical Cavity Studied with Smoothed Particle Hydrodynamics. Applied Sciences. 2021; 11(8):3500

My contributions in this work were: Designed the work and performed the simulations, Methodology, Validation, Writing the original draft, Reviewing and Editing.

I would like to thank all the authors who have contributed to this work.

applied sciences

MDPI

Article

Non-Symmetrical Collapse of an Empty Cylindrical Cavity Studied with Smoothed Particle Hydrodynamics

Andrea Albano *,† **and Alessio Alexiadis** *,†

School of Chemical Engineering, University of Birmingham, Birmingham B15 2TT, UK
* Correspondence: AXA1220@student.bham.ac.uk(A.A.); Alexiadis@bham.ac.uk (A.A.)
† These authors contributed equally to this work.

Abstract: The non-symmetrical collapse of an empty cylindrical cavity is modeled using Smoothed Particle Hydrodynamics. The presence of a nearby surface produces an anisotropic pressure field generating a high-velocity jet that hits the surface. The collapse follows a different dynamic based on the initial distance between the center of the cavity and the surface. When the distance is greater than the cavity radius (detached cavity) the surface is hit by traveling shock waves. When the distance is less than the cavity radius (attached cavity) the surface is directly hit by the jet and later by other shock waves generated in the last stages of the of the collapse. The results show that the surface is hit by a stronger shock when distance between the center of the cavity and the surface is zero while showing more complex double peaks behavior for other distances.

Keywords: particle method; smoothed particle hydrodynamics; modeling; simulations; shock wave

Citation: Albano, A.; Alexiadis, A. Non-Symmetrical Collapse of an Empty Cylindrical Cavity Studied with Smoothed Particle Hydrodynamics. *Appl. Sci.* **2021**, *11*, 3500. https://doi.org/10.3390/app11083500

Academic Editor: Leonid Burakovsky

Received: 11 January 2021
Accepted: 23 March 2021
Published: 14 April 2021

1. Introduction

Cavitation is a phenomenon occurring in a liquid that undergoes rapidly changes in pressure. At first a bubble, or cavity, nucleates and growth over a nucleation site. After growing up to maximum value, the cavity collapses generating shock waves [1].

When the pressure field in the liquid is isotropic, the cavity preserves its symmetry and the collapse has spherical symmetry [2–4]. However, when an anisotropic pressure field drives the collapse, the cavity does not preserve its symmetry and folds in a specific direction generating a high-velocity flow known as jet [1,5]. The anisotropic pressure field is generated by the so-called anisotropic driver, which also defines the jet folding direction [6]. The most common anisotropic drivers are rigid or free surface, gravity, presence of neighbor bubbles or a combination of the above. Another mechanism that makes a cavity folds during the collapse is the interaction with a shock waves that folds the cavity in the shock direction. This situation is known as shock-induced collapse, while the pressure driven collapse is called Rayleigh collapse [7,8].

From an engineering point of view, the high-speed jet generated in the Rayleigh collapse by the presence of a nearby solid surface is the scenario most addressed in the literature [8,9]. In fact, the high-speed flow can hit the surface causing erosion and eventually loss of material. This phenomenon, known as cavitation erosion, is undesirable in industrial, military and power station equipment such as pump impellers, high-speed propellers and turbine blades [10–12].

Thanks to its Lagrangian nature, Smoothed Particle Hydrodynamics (SPH) can handle problems with large deformations better than mesh-based technique [13] and, for this reason, it is particularly suited for studying cavitation. Joshi et al. [14] developed a Smoothed Particle Hydrodynamics axisymmetric solver to simulate a shock-induced collapse of an empty cavity and the erosion process of the nearby surface. Pineda et al. [15] used a SPH-ALE method to study a gas filled cylindrical cavity Rayleigh collapse far and near a surface. Nair and Tomar [16] used SPH to simulate an oscillating gas filled cylindrical cavity under a variable isotropic pressure field. Albano and Alexiadis [17] developed a

SPH model to study the Rayleigh collapse of a gas filled cylindrical cavity that takes in account the heat diffusion at the gas-liquid interface. Among these studies, only Joshi et al. and Pineda et al. take in account anisotropic collapses. Joshi et al. only for the shock-induced collapse case, while Pineda at al only for cavity detached from the surface. Despite its importance in practical applications, non-symmetrical Rayleigh collapse of surface attached cavities, to the best of our knowledge, has never been investigated with the SPH method. Moreover, cavities commonly nucleate on surfaces rather than away from it [18–20]. This work, therefore, develops a SPH model for a non-symmetrical Rayleigh collapse of a cylindrical cavity which is considered the greatest causes of erosion and material loss [8,9].

2. Model

2.1. Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics is a mesh-free particle method originally developed by Gingold Monaghan [21] and Lucy [22] and used in a wide range of applications: astrophysics [23], shock waves [24–27], explosion [28–30], thermo-fluid flows [31], thermo-capillary flows [32], multiphase flow [33,34], Biological flows [35], non-Newtonian fluid flows [36,37].

Thanks to its particle nature SPH is part of the Discrete Multi-Physics framework where, coupled with other particle methods [38–40], is used to address multi-complex physics phenomena [41–45] to overcome the single weakness of each method.

The idea behind the SPH formulation lies in the integral representation of any continuum function $f(\mathbf{r})$ depending on the three-dimensional position vector $\mathbf{r}$

$$f(\mathbf{r}) \approx \iiint f(\mathbf{r}')W(\mathbf{r}-\mathbf{r}',h)d\mathbf{r}', \tag{1}$$

where W is the smoothing function or kernel and h is the smoothing length. Dividing the domain in a finite number of computational particles with their own mass $m = \rho dr^3$, is possible approximate any continuum function $f(\mathbf{r})$ as

$$f(\mathbf{r}) \approx \sum \frac{m_i}{\rho_i} f(\mathbf{r}_i)W(\mathbf{r}-\mathbf{r_i},h), \tag{2}$$

where m_i, ρ_i and r_i are mass, density and position of the i-th particle. Equation (2) is known as particle approximation in SPH literature [13].

We used the SPH particle approximation to discretize continuity, momentum and energy conservation equation:

$$\begin{cases} \frac{d\rho}{dt} = -\rho\frac{\partial \mathbf{v}^\beta}{\partial \mathbf{x}^\beta}, \\ \frac{d\mathbf{v}^\alpha}{dt} = -\frac{1}{\rho}\frac{\partial P}{\partial \mathbf{x}^\alpha}, \\ \frac{de}{dt} = -\frac{P}{\rho}\frac{\partial \mathbf{v}^\beta}{\partial \mathbf{x}^\beta}, \end{cases} \xrightarrow{\text{Eq 1}} \begin{cases} \frac{d\rho_i}{dt} = \sum_j m_j \mathbf{v}_{ij}^\beta \frac{\partial W_{ij}}{\partial \mathbf{x}_i^\beta}, \\ m_i\frac{d\mathbf{v}_i^\alpha}{dt} = \sum_j m_i m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_i}{\rho_j^2} + \Pi_{ij}\right)\frac{\partial W_{ij}}{\partial \mathbf{x}_i^\alpha}, \\ m_i\frac{de_i}{dt} = \frac{1}{2}\sum_j m_i m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_i}{\rho_j^2} + \Pi_{ij}\right)\mathbf{v}_{ij}^\beta\frac{\partial W_{ij}}{\partial \mathbf{x}_i^\beta}, \end{cases} \tag{3}$$

where $\mathbf{v}$ is the velocity vector with $\mathbf{v}_{ij} = \mathbf{v_i} - \mathbf{v_j}$, e internal energy and Π_{ij} is the artificial viscosity introduced by Monaghan [24]. The Monaghan artificial viscosity depends on a constant parameter called dimensionless dissipation factor, α, and from the speed of sound of particle i and j, c_i and c_j, with the following relationship.

$$\Pi_{ij} = -\alpha h \frac{c_i + c_j}{\rho_i + \rho_j}\frac{\mathbf{v}_{ij}\cdot\mathbf{r}_{ij}}{r_{ij}^2 + \epsilon h^2}. \tag{4}$$

To solve the set of Equations shown before, an Equation Of State (EOS) that links pressure P and density ρ is required.

2.2. Computational Set Up

In this work, we focus on the non-symmetrical collapse induced by an anisotropic pressure field. The anisotropy is generated by non-symmetric water domain shown in Figure 1. The domain is divided in three concentric regions, delimited by three different radii. In each region, different types of computational particles are used:

- Cavity ($r < R_0$): inside the yellow region in Figure 1, particles are removed to generate an empty cavity with $P_b = 0$.
- Liquid ($R_0 < r < R_S$): inside the blue region in Figure 1, particles are modeled as water following a liquid EOS. The density is set as $\rho_L = 1000$ [kg m^{-3}] with initial pressure P_∞.
- Shell ($r > R_S$): inside the red region of Figure 1, are modeled as in the liquid region. Moreover, they have fixed position and density to keep constant pressure as boundary condition. The extension of shell region has been discussed in previous work [17]. The lower part of this region also acts as a wall inducing anisotropy in the pressure field during the collapse.

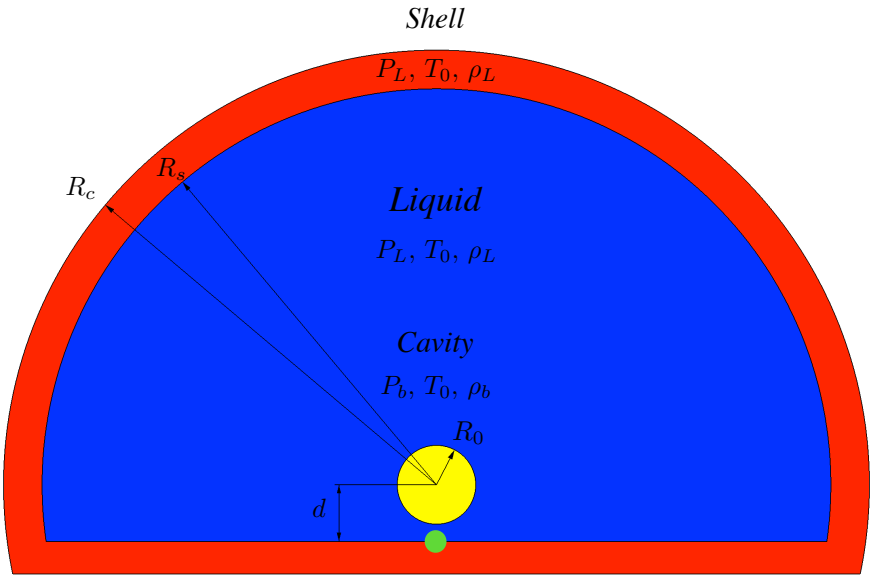


Figure 1. Geometry of the simulation box.

The green region in Figure 1 does not refer to a different type of particle. We highlight it because, later, the pressure generated during the collapse will be monitored in the green region. As explained in next section, when a non-symmetric collapse is studied, it is necessary to quantify the anisotropy of the collapse. When the anisotropy is induced by the presence of a nearby solid surface, is common to use the stand-off, γ [6] defined as

$$\gamma = \frac{d}{R_0}, \tag{5}$$

where d is the distance between the cavity center and the wall (see Figure 1) and R_0 its initial radius. The dynamic of a cylindrical cavity in a Rayleigh collapse, where the driving force is the pressure difference between the pressure in the liquid, $P_\infty = P_L$, and the pressure in the cavity, p_b, is described by a 2D Rayleigh-Plesset (2DRP) equation [46]:

$$\frac{P_\infty - P_b}{\rho} = \left(\left(\frac{dR}{dt}\right)^2 + R\frac{d^2R}{dt}\right) ln\left(\frac{R}{r_\infty}\right) + \frac{1}{2}\left(R\frac{dR}{dt}\right)^2\left(\frac{1}{R^2} - \frac{1}{r_\infty^2}\right). \tag{6}$$

However, the 2DRP equation only describe the dynamic of a symmetrical collapse. Form our knowledge an equation to validate the dynamic of a non-symmetrical collapse has still to

be developed. For this reason, as commonly done in the literature [15,17,46,47], the model presented as been validated for the case of symmetric collapse against Equation (6) [17].

In the simulations we use the Lucy Kernel [13] and the Tait EOS [13] with a smoothing length of $h = 1.3 \cdot dL$ where dL is the initial particle spacing. The dimensionless dissipation factor, time step and speed of sound were set as $\alpha = 1$, $t_s = 1e-10$ [s] and $c_0 = 1484$ [m s^{-1}]. The sensitivity of the results to parameters such as the kernel function, EOS or h, was investigated in a previous publication for the case of symmetric collapse [17]. In the next section, the particle resolution for the specific case of non-symmetric collapse is discussed.

2.3. Software for Simulation, Visualization and Post-Process

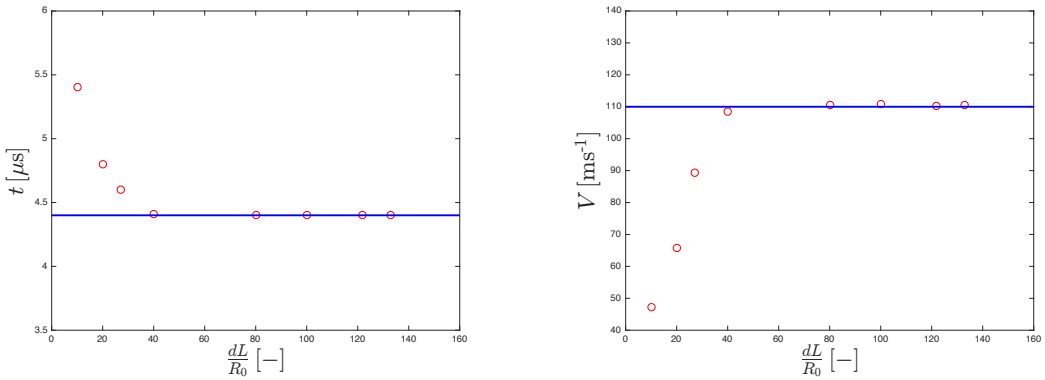
The simulations were run with the open-source code simulator LAMMPS [48,49]. The visualization and data post-processing were generated with the open-source code OVITO [50].

3. Results

As mentioned in the introduction, collapsing cavities formed during cavitation generate high-pressure shock wave [1]. When the collapse occurs near a solid surface, the symmetry of the cavity is not preserved and this produces a high-velocity flow known as re-entrant jet [5,7,8]. The characteristics of the jet depend on γ (see Equation (5)). When $\gamma > 1$, the jet hits the opposite side of the cavity generating a high-pressure shock wave that travels in the direction of the solid surface. When $\gamma \leq 1$, the jet hits directly the solid surface generating a water hammer pressure [51,52] that causes erosion.

3.1. Preliminary Results

Four preliminary simulations with different particle resolutions have been carried out with $R_0 = 100$ [µm], $P_\infty = 5$ MPa and $\gamma = 1.2$. These values are chosen to match those commonly used in computational studies [14,17,52]. Figure 2 shows the total collapse time and the max jet speed (defined as $V = |\mathbf{v}|$) for the different resolutions. Particle resolution is specified as the ratio between the initial particle spacing dL and the initial cavity radius R_0.



(a) **(b)**

Figure 2. Resolution convergences for collapse time (**a**) and max jet speed (**b**).

Both the collapsing time and max jet velocity converge for $dL/R_0 \geq 40$. This resolution value was also independently found by Joshi et al. and Pineda et al. At this resolution, the model correctly reproduces jet formation [5,7]. Figure 3 shows that as the collapse proceeds, wall proximity produces a pressure difference between the top and the bottom of the cavity. This pushes down the upper-side of the cavity inducing the formation of a high-velocity/low-pressure jet.

In Section 3.3 we show simulations at higher resolution. In fact, at higher resolutions, we will be able to uncover more details of the velocity pattern occurring during the collapse.

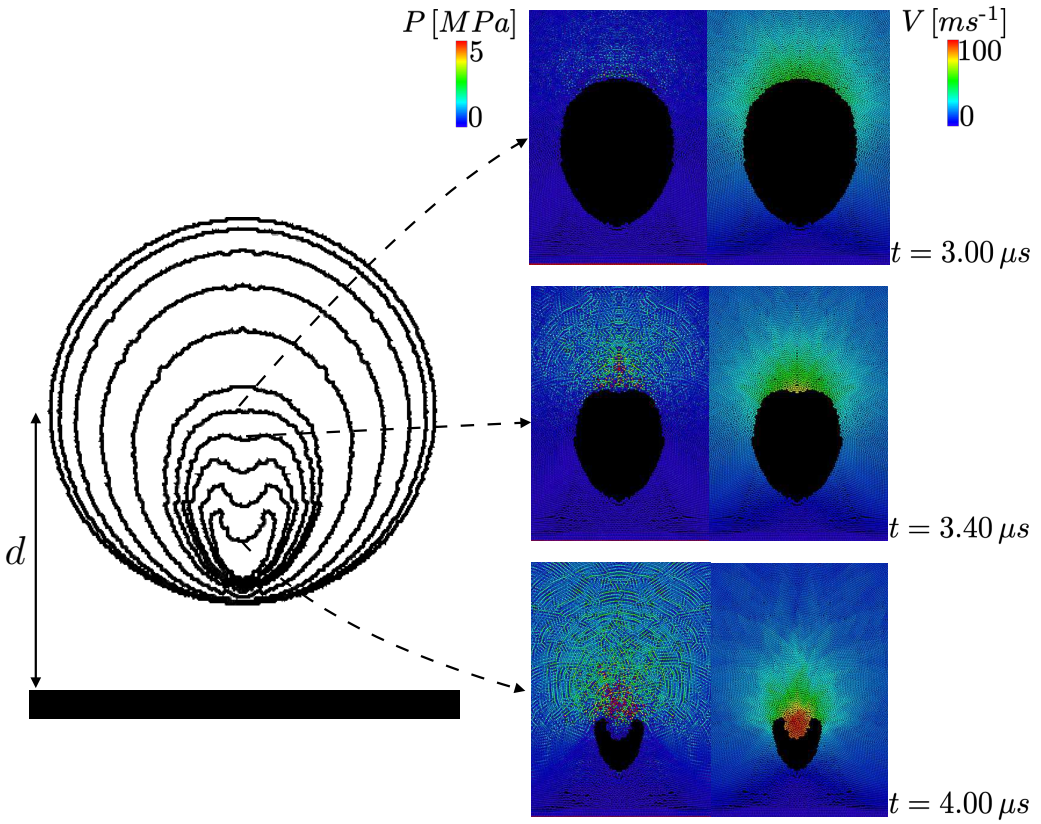
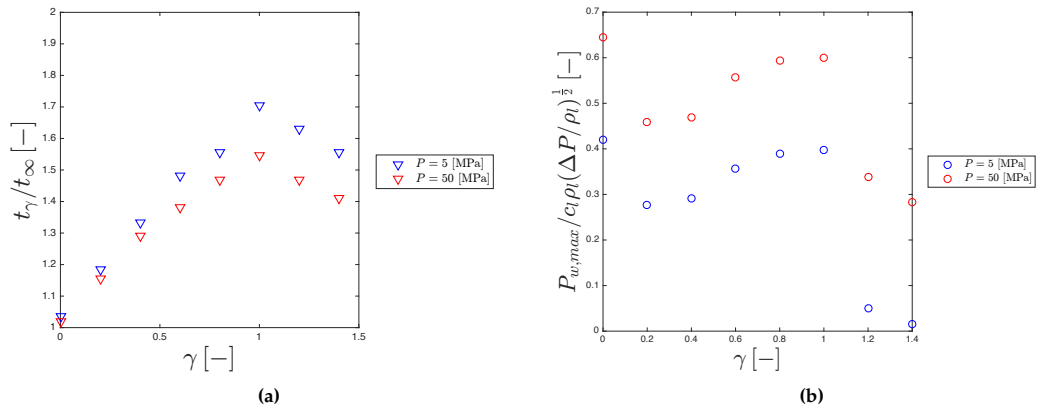


Figure 3. Shape evolution with pressure and velocity field ($dL/R_0 = 80$, $\gamma = 1.2$ and $P_\infty = 5$ MPa).

3.2. Stand-Off and Pressure Analysis

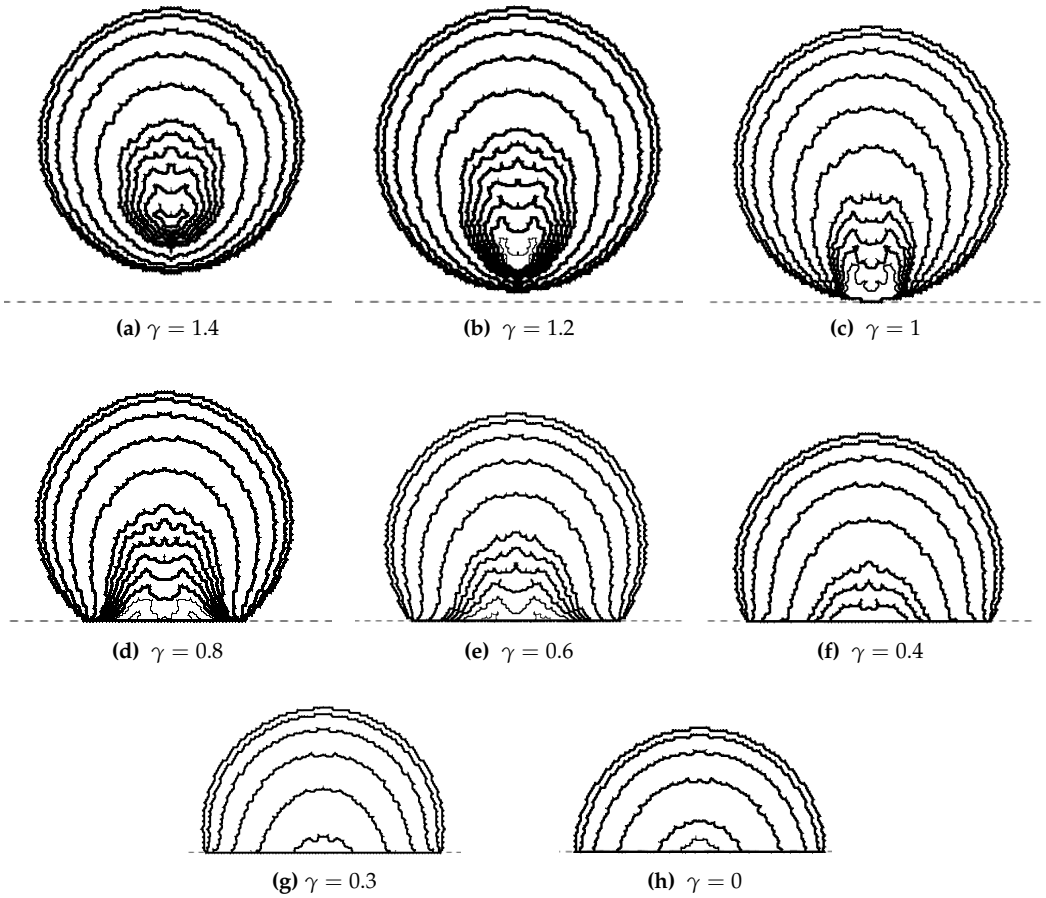
To study the effect of γ on the collapse, eight different stand-offs, in the range [0, 1.4], are simulated with $P_\infty = 5$ and 50 MPa. The upper limit $\gamma = 1.4$ was chosen since this study focuses on strongly deformed collapses, which are known to produce cavitation erosion [1,8]. In Figure 4, the dimensionless collapsing time and the maximal pressure at the wall are plotted for different γ. The dimensionless time is defined as the ratio between t_γ, the collapsing time obtained in the simulation for a given γ, and t_∞, the collapsing time for $\gamma \to \infty$ [17]. The dimensionless pressure is defined as the ratio between the maximal pressure at the wall and the characteristic pressure of a Rayleigh collapse [52]. The pressure at the wall is calculated at the green circle shown in Figure 1 positioned below the center of the cavity with a diameter of 0.01 mm.



(a) **(b)**

Figure 4. Dimensionless collapse time (**a**) and dimensionless pressure (**b**) for different stand-off and pressure ($dL/R_0 = 40$).

The collapsing time shows a maximum for $\gamma = 1$ and a minimum for $\gamma = 0$. When $\gamma = 1$, the lower part of the cavity touches the wall, while the upper part is free to move (Figure 5c). During the collapse, the cavity loses its symmetry generating a re-entrant jet, which travels for the whole diameter of the cavity before reaching the surface. When $\gamma = 0$ the cavity reduces to a semi sphere. Therefore, the collapse is symmetric again (Figure 5h) and, in fact, $t_\gamma / t_\infty \approx 1$ (Figure 4a) as for $\gamma \to \infty$ [17].



(a) $\gamma = 1.4$ **(b)** $\gamma = 1.2$ **(c)** $\gamma = 1$

(d) $\gamma = 0.8$ **(e)** $\gamma = 0.6$ **(f)** $\gamma = 0.4$

(g) $\gamma = 0.3$ **(h)** $\gamma = 0$

Figure 5. Cavity shapes for different stand-offs ($dL/R_0 = 40$ and $P_\infty = 5$ MPa). Each shape shows the cavity outline at a different time.

The pressure is shown in Figure 4b. The normalized maximum wall pressure is thought to scale with γ^{-1} [1,6]. However, this was reported for values of $\gamma > 1.4$, which are outside the range investigated in this study. Other studies show results similar to ours in the range of investigation [9,53].

3.3. Pressure and Speed Developed in High-Resolution Collapse

For 3 different stand-offs (0.0, 0.6 and 1.0), pressure histories at the center of the surface (i.e., green circle in Figure 1) are plotted in Figure 6.

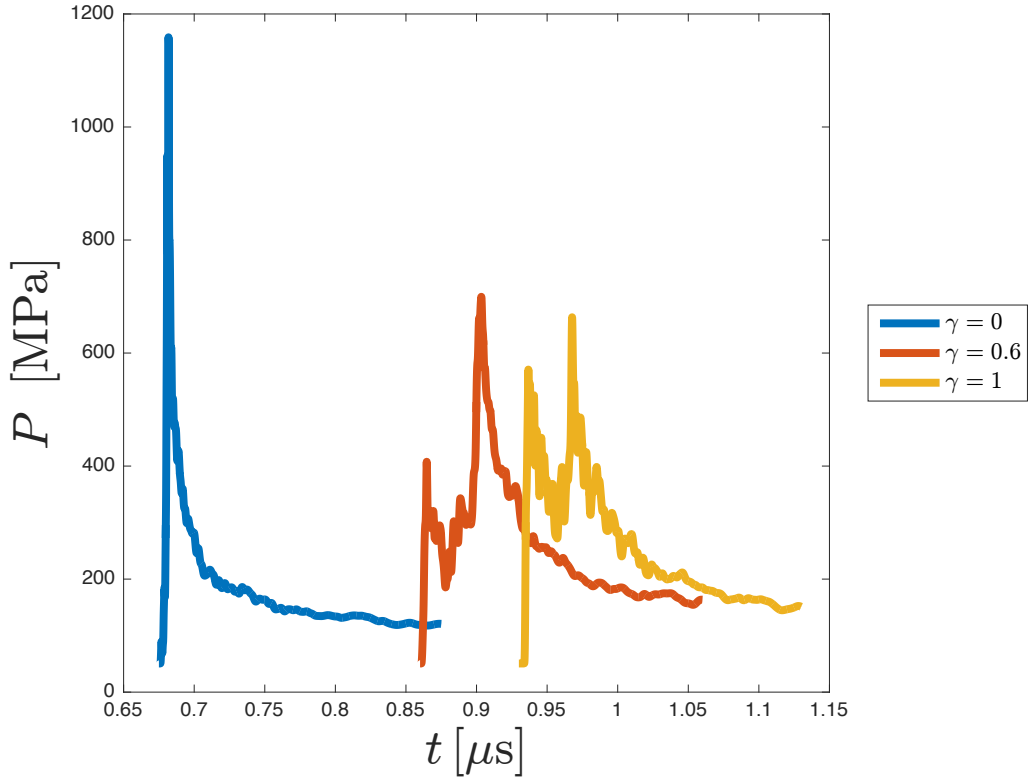


Figure 6. Pressure trend for different stand-offs (dL/R_0=133 and $P_\infty = 50$ MPa).

For $\gamma = 0$, the pressure shows a single maximum of roughly 1200 MPa. This pressure is generated by a water hammer impact of the collapsing cavity on the surface. Since the symmetry is preserved, see Section 3.2, the collapse ends when the cavity impacts on the surface generating a high-pressure shock wave (Figure 7a). After the impact, the pressure at wall decreases as the shock moves into the liquid (Figure 7b,c).

When γ = 0.6 and 1 the pressure shows a double peak. As with $\gamma = 0$, the first peak is generated by the water hammer impact of the cavity with the surface (Figures 8a and 9a). After the impact, the jet splits into two high-speed/low-pressure lateral jets (Figure 8b) that will later impact with the sides of the cavity generating collapsing circles and a pair of "side" pressure waves (Figures 8c and 9b).

The collapse ends when the fluid fills the circles. When this happens a third pair of pressure waves is generated (Figures 8d and 9c). The waves will later merge at the center resulting in a second pressure peak (Figures 8e and 9d). Figure 10 shows a schematic representation of a wall-attached cavity collapse with the three pairs shock formation for $\gamma = 0.6$. It can be difficult to clearly identify these hydrodynamics patterns if the simulation is run at lower resolution (e.g., Joshi et al. 2019).

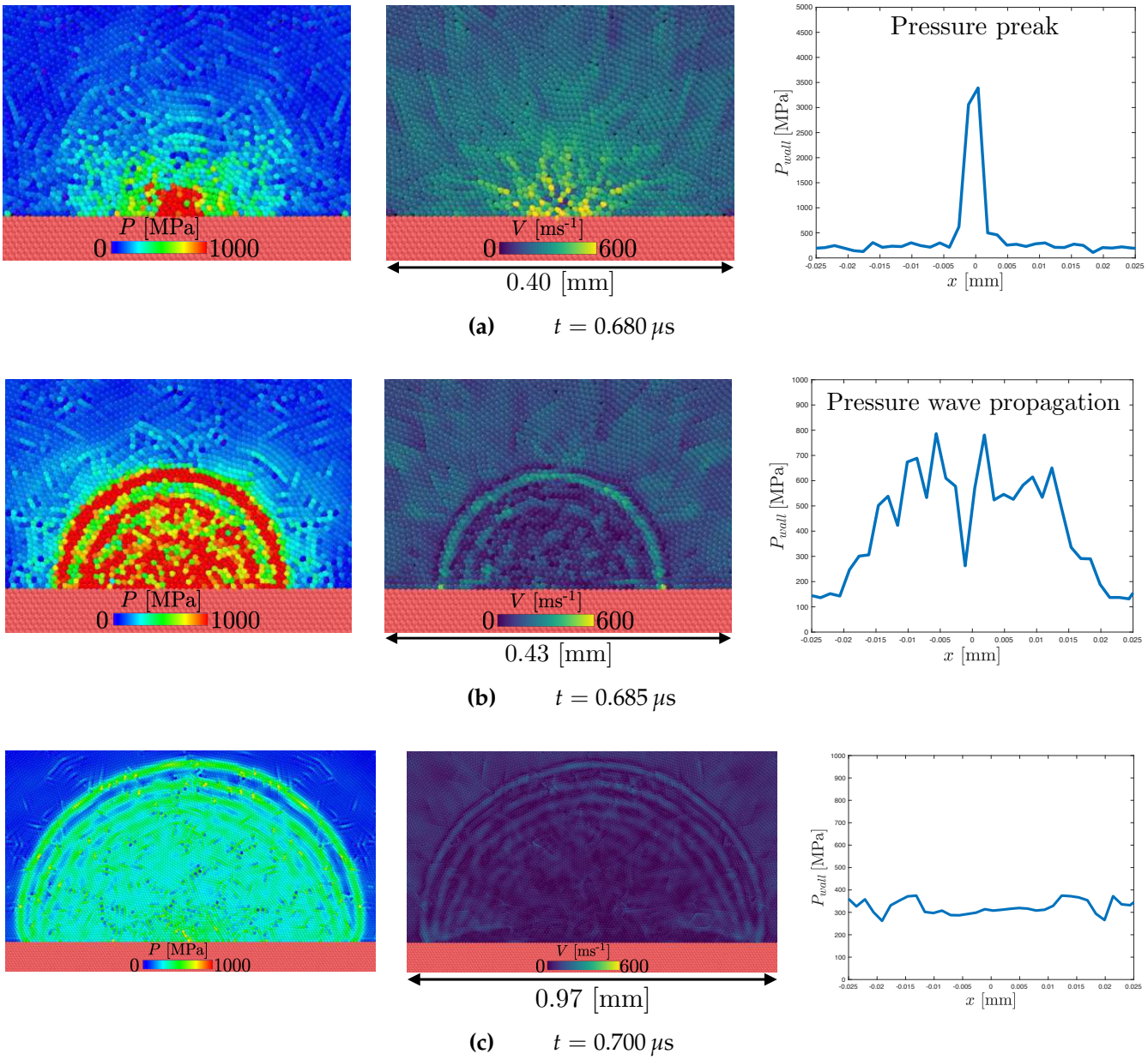
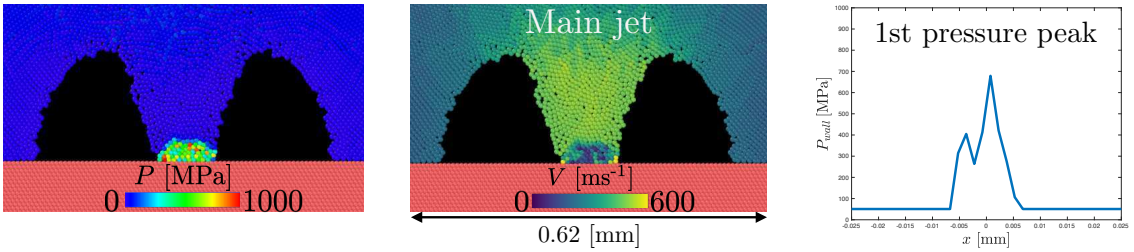
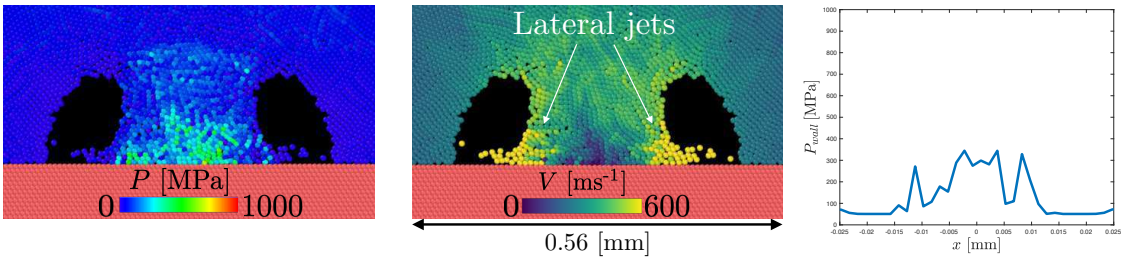


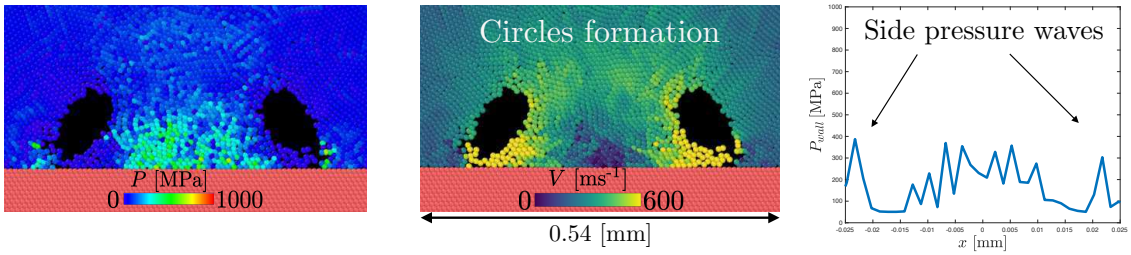
Figure 7. Pressure and velocity field in the liquid and the spatial pressure trend at the wall (dL/R_0 = 133, $\gamma = 0$ and $P_\infty = 50$ MPa).



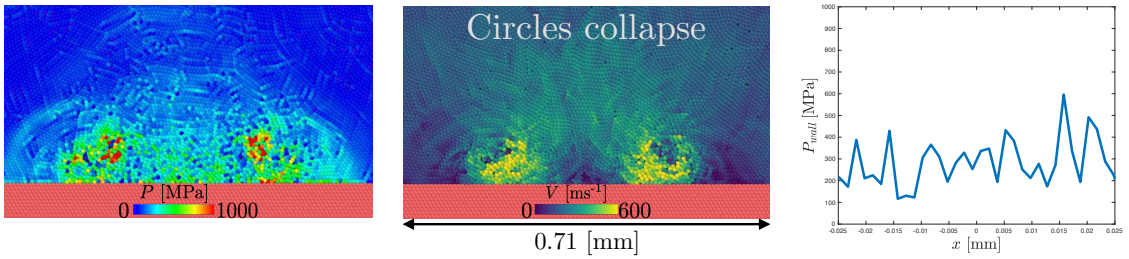
(a) $t = 0.865\,\mu$s



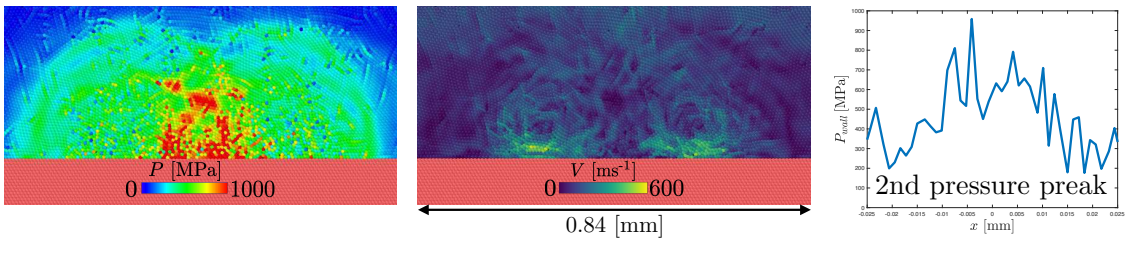
(b) $t = 0.880\,\mu$s



(c) $t = 0.885\,\mu$s

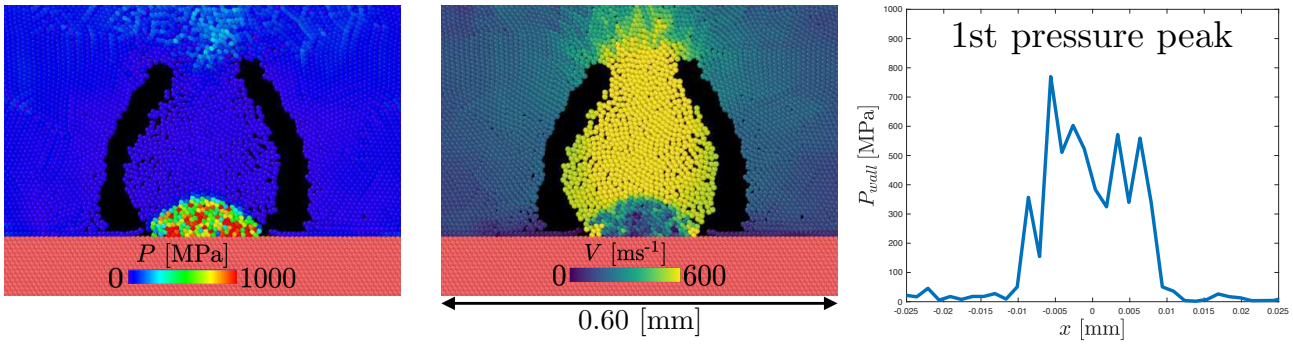


(d) $t = 0.895\,\mu$s

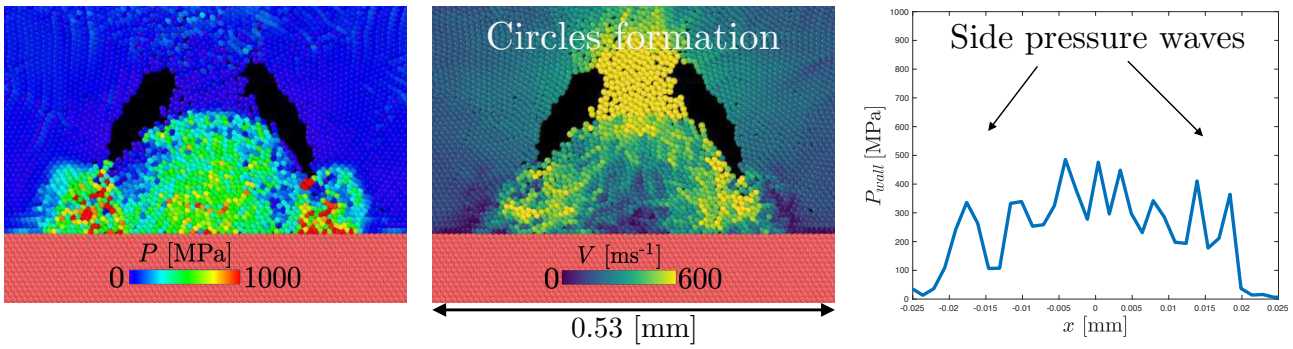


(e) $t = 0.910\,\mu$s

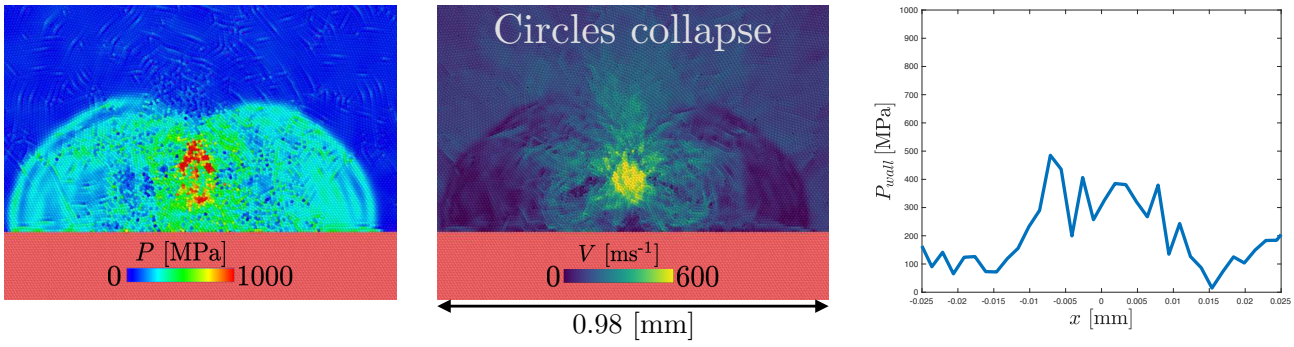
Figure 8. Pressure and velocity field in the liquid and the spatial pressure trend at the wall (dL/R_0 = 133, $\gamma = 0.6$ and $P_\infty = 50$ MPa).



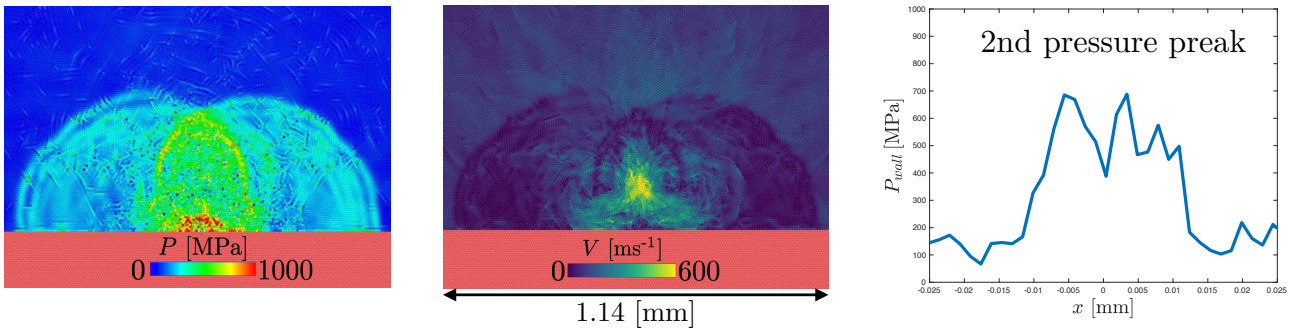
(a) $t = 0.940\,\mu$s



(b) $t = 0.950\,\mu$s



(c) $t = 0.965\,\mu$s



(d) $t = 0.970\,\mu$s

Figure 9. Pressure and velocity field in the liquid and the spatial pressure trend at the wall (dL/R_0 = 133, $\gamma = 1$ and $P_\infty = 50$ MPa).

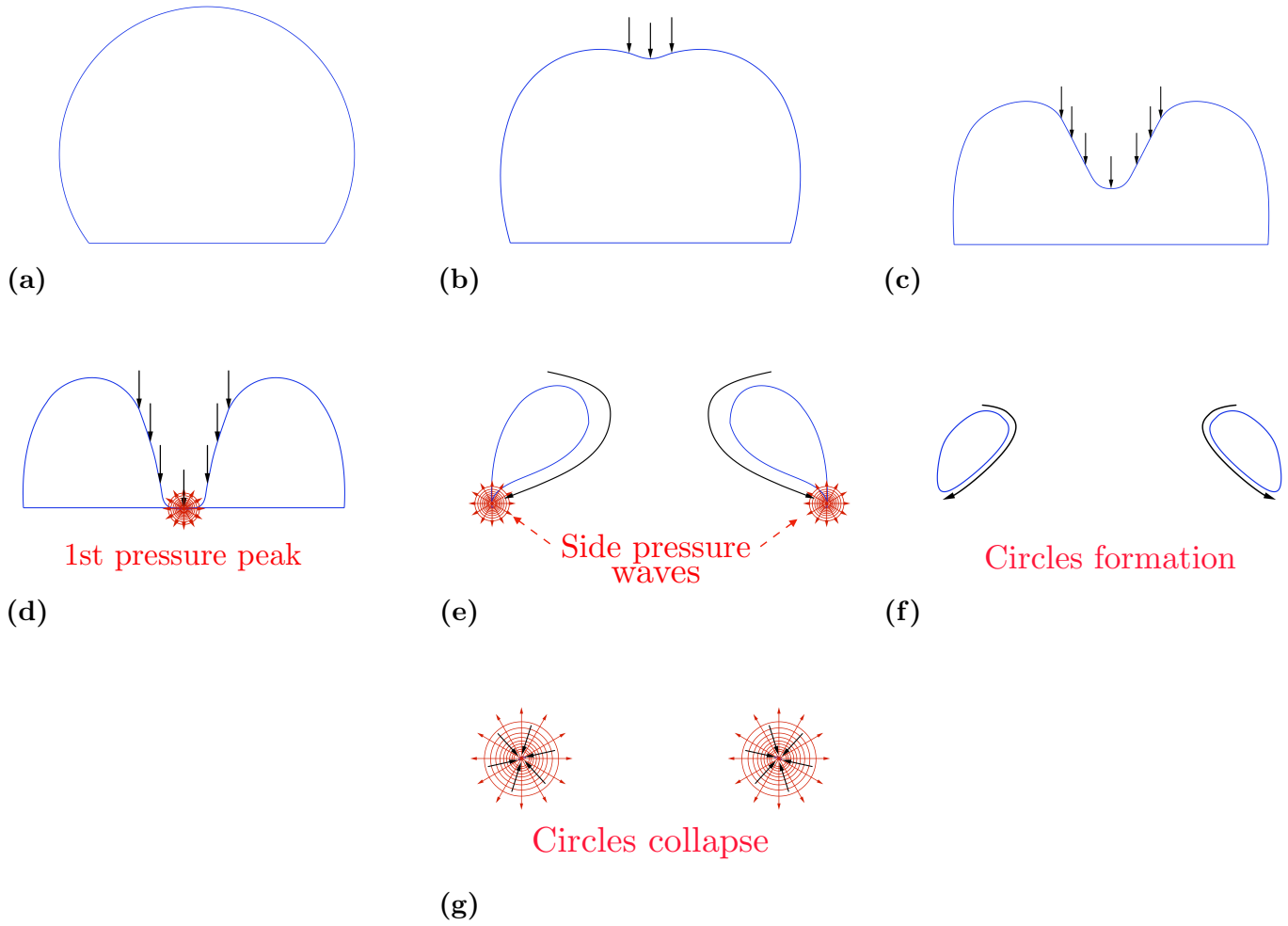


Figure 10. Schematic representation of a wall-attached cavity collapse with the three pairs shock formation for $\gamma = 0.6$. The blue lines represents the cavity, black arrows represent the liquid flow, red circles represent the shock waves propagation at different collapse stages.

As shown in Figure 6, when $\gamma = 1$ the first peak is higher compared to the first peak of $\gamma = 0.6$. Before hitting the surface, the jet travels a longer distance for $\gamma = 1$ rather than $\gamma = 0.6$. Therefore, it is accelerated by the pressure gradient for longer resulting in a higher water hammer pressure at the wall [8,51,54].

However, for the second peak, the behavior is reversed, and the peak is higher for $\gamma = 0.6$. The water between the collapsing rings and the wall acts as a shield mitigating the shock wave [26,55]. When $\gamma = 1$ the distance between the rings and the wall is higher than $\gamma = 0.6$ resulting in a greater mitigation effect and lower pressure peak.

4. Conclusions

A SPH model is developed to study non-symmetrical Rayleigh collapse of an empty cylindrical cavity. When the cavity collapse near a solid surface the anisotropy of the pressure field induce the formation of the re-entrant jet. This anisotropy is quantified with the stand-off, γ.

Different collapses in the range $0 \leq \gamma \leq 1.4$ are discussed showing that the model can correctly simulate the jet physics and the consequent pressure fields at the surface.

We study the collapse of attached cavities ($\gamma \leq 1$), where the liquid exerts the maximum pressure over the surface. When $\gamma = 0$ the collapse is symmetric again and pressure shows a single max, which corresponds to the end cavity collapse. When $0 < \gamma \leq 1$ the

pressure shows multiple peaks, which reflect a more complex behavior with the circles' formation and collapse.

This work shows the importance of studying the collapse of cavities attached to the surface. This case is less investigated than the detached collapse, but a better understanding of the final stages of the attached collapse is essential to improve predictions of cavitation erosion. For this reason, the simulations are run at higher resolution than previous studies. Only in this way, in fact, certain hydrodynamic features of the collapse can clearly emerge from the simulations.

Author Contributions: A.A. (Andrea Albano) and A.A. (Alessio Alexiadis) conceptualise the work; A.A. (Andrea Albano) designed the work and performed the simulations; A.A. (Andrea Albano) and A.A. (Alessio Alexiadis) contributed in writing–review and editing the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the US Office of Naval Research Global (ONRG) under 256 NICOP Grant N62909-17-1-2051.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Brennen, C.E. *Cavitation and Bubble Dynamics*; Cambridge University Press: Cambridge, UK, 2014.
2. Rayleigh, L. VIII. On the pressure developed in a liquid during the collapse of a spherical cavity. *Lond. Edinb. Dublin Philos. Mag. J. Sci.* **1917**, *34*, 94–98. [CrossRef]
3. Plesset, M.S. The dynamics of cavitation bubbles. *J. Appl. Mech.* **1949**, *16*, 277–282. [CrossRef]
4. Hickling, R.; Plesset, M.S. Collapse and rebound of a spherical bubble in water. *Phys. Fluids* **1964**, *7*, 7–14. [CrossRef]
5. Plesset, M.S.; Chapman, R.B. Collapse of an initially spherical vapour cavity in the neighbourhood of a solid boundary. *J. Fluid Mech.* **1971**, *47*, 283–290. [CrossRef]
6. Supponen, O.; Obreschkow, D.; Tinguely, M.; Kobel, P.; Dorsaz, N.; Farhat, M. Scaling laws for jets of single cavitation bubbles. *J. Fluid Mech.* **2016**, *802*, 263–293. [CrossRef]
7. Johnsen, E.; Colonius, T. Numerical simulations of non-spherical bubble collapse. *J. Fluid Mech.* **2009**, *629*, 231–262. [CrossRef]
8. Kim, K.H.; Chahine, G.; Franc, J.P.; Karimi, A. *Advanced Experimental and Numerical Techniques for Cavitation Erosion Prediction*; Springer: Berlin, Germany, 2014; Volume 106.
9. Philipp, A.; Lauterborn, W. Cavitation erosion by single laser-produced bubbles. *J. Fluid Mech.* **1998**, *361*, 75–116. [CrossRef]
10. Knapp, R.; Daily, J.; Hammitt, F. *Cavitation (Engineering Societies Monographs)*; McGraw-Hill: New York, NY, USA, 1970; Volume 39.
11. Arndt, R.E. Cavitation in fluid machinery and hydraulic structures. *Annu. Rev. Fluid Mech.* **1981**, *13*, 273–326. [CrossRef]
12. Giannadakis, E.; Gavaises, M.; Arcoumanis, C. Modelling of cavitation in diesel injector nozzles. *J. Fluid Mech.* **2008**, *616*, 153–193. [CrossRef]
13. Liu, G.R.; Liu, M.B. *Smoothed Particle Hydrodynamics: A Meshfree Particle Method*; World Scientific: Singapore, 2003.
14. Joshi, S.; Franc, J.P.; Ghigliotti, G.; Fivel, M. SPH modelling of a cavitation bubble collapse near an elasto-visco-plastic material. *J. Mech. Phys. Solids* **2019**, *125*, 420–439. [CrossRef]
15. Pineda, S.; Marongiu, J.C.; Aubert, S.; Lance, M. Simulation of a gas bubble compression in water near a wall using the SPH-ALE method. *Comput. Fluids* **2019**, *179*, 459–475. [CrossRef]
16. Nair, P.; Tomar, G. Simulations of gas-liquid compressible-incompressible systems using SPH. *Comput. Fluids* **2019**, *179*, 301–308. [CrossRef]
17. Albano, A.; Alexiadis, A. A smoothed particle hydrodynamics study of the collapse for a cylindrical cavity. *PLoS ONE* **2020**, *15*, e0239830. [CrossRef]
18. Pruppacher, H.R.; Klett, J.D. Microphysics of clouds and precipitation. *Nature* **1980**, *284*, 88. [CrossRef]
19. Sear, R.P. Quantitative studies of crystal nucleation at constant supersaturation: Experimental data and models. *CrystEngComm* **2014**, *16*, 6506–6522. [CrossRef]
20. Ferraro, G.; Jadhav, A.J.; Barigou, M. A Henry's law method for generating bulk nanobubbles. *Nanoscale* **2020**, *12*, 15869–15879. [CrossRef]
21. Gingold, R.A.; Monaghan, J.J. Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Mon. Not. R. Astron. Soc.* **1977**, *181*, 375–389. [CrossRef]
22. Lucy, L.B. A numerical approach to the testing of the fission hypothesis. *Astron. J.* **1977**, *82*, 1013–1024. [CrossRef]
23. Springel, V. Smoothed particle hydrodynamics in astrophysics. *Annu. Rev. Astron. Astrophys.* **2010**, *48*, 391–430. [CrossRef]
24. Monaghan, J.; Gingold, R.A. Shock simulation by the particle method SPH. *J. Comput. Phys.* **1983**, *52*, 374–389. [CrossRef]
25. Morris, J.; Monaghan, J. A switch to reduce SPH viscosity. *J. Comput. Phys.* **1997**, *136*, 41–50. [CrossRef]
26. Liu, M.; Liu, G.; Lam, K. Investigations into water mitigation using a meshless particle method. *Shock Waves* **2002**, *12*, 181–195. [CrossRef]

27. Albano, A.; Alexiadis, A. Interaction of Shock Waves with Discrete Gas Inhomogeneities: A Smoothed Particle Hydrodynamics Approach. *Appl. Sci.* **2019**, *9*, 5435. [CrossRef]
28. Liu, M.; Liu, G.; Lam, K.; Zong, Z. Smoothed particle hydrodynamics for numerical simulation of underwater explosion. *Comput. Mech.* **2003**, *30*, 106–118. [CrossRef]
29. Liu, M.; Liu, G.; Zong, Z.; Lam, K. Computer simulation of high explosive explosion using smoothed particle hydrodynamics methodology. *Comput. Fluids* **2003**, *32*, 305–322. [CrossRef]
30. Sirotkin, F.V.; Yoh, J.J. A Smoothed Particle Hydrodynamics method with approximate Riemann solvers for simulation of strong explosions. *Comput. Fluids* **2013**, *88*, 418–429. [CrossRef]
31. Ng, K.; Ng, Y.; Sheu, T.; Alexiadis, A. Assessment of Smoothed Particle Hydrodynamics (SPH) models for predicting wall heat transfer rate at complex boundary. *Eng. Anal. Bound. Elem.* **2020**, *111*, 195–205. [CrossRef]
32. Hopp-Hirschler, M.; Shadloo, M.S.; Nieken, U. A smoothed particle hydrodynamics approach for thermo-capillary flows. *Comput. Fluids* **2018**, *176*, 1–19. [CrossRef]
33. Shadloo, M.S.; Oger, G.; Le Touzé, D. Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: Motivations, current state, and challenges. *Comput. Fluids* **2016**, *136*, 11–34. [CrossRef]
34. Rahmat, A.; Yildiz, M. A multiphase ISPH method for simulation of droplet coalescence and electro-coalescence. *Int. J. Multiph. Flow* **2018**, *105*, 32–44. [CrossRef]
35. Ye, T.; Pan, D.; Huang, C.; Liu, M. Smoothed particle hydrodynamics (SPH) for complex fluid flows: Recent developments in methodology and applications. *Phys. Fluids* **2019**, *31*, 011301.
36. Shao, S.; Lo, E.Y. Incompressible SPH method for simulating Newtonian and non-Newtonian flows with a free surface. *Adv. Water Resour.* **2003**, *26*, 787–800. [CrossRef]
37. Hosseini, S.; Manzari, M.; Hannani, S. A fully explicit three-step SPH algorithm for simulation of non-Newtonian fluid flow. *Int. J. Numer. Methods Heat Fluid Flow* **2007**, *17*, 715–735. [CrossRef]
38. Pazdniakou, A.; Adler, P. Lattice spring models. *Transp. Porous Media* **2012**, *93*, 243–262. [CrossRef]
39. Silling, S.A.; Epton, M.; Weckner, O.; Xu, J.; Askari, E. Peridynamic states and constitutive modeling. *J. Elast.* **2007**, *88*, 151–184. [CrossRef]
40. Munjiza, A.A. *The Combined Finite-Discrete Element Method*; John Wiley & Sons: Hoboken, NJ, USA, 2004.
41. Alexiadis, A. A smoothed particle hydrodynamics and coarse-grained molecular dynamics hybrid technique for modelling elastic particles and breakable capsules under various flow conditions. *Int. J. Numer. Methods Eng.* **2014**, *100*, 713–719. [CrossRef]
42. Alexiadis, A. The discrete multi-hybrid system for the simulation of solid-liquid flows. *PLoS ONE* **2015**, *10*, e0124678. [CrossRef] [PubMed]
43. Ariane, M.; Kassinos, S.; Velaga, S.; Alexiadis, A. Discrete multi-physics simulations of diffusive and convective mass transfer in boundary layers containing motile cilia in lungs. *Comput. Biol. Med.* **2018**, *95*, 34–42. [CrossRef]
44. Rahmat, A.; Barigou, M.; Alexiadis, A. Numerical simulation of dissolution of solid particles in fluid flow using the SPH method. *Int. J. Numer. Methods Heat Fluid Flow* **2019**. [CrossRef]
45. Alexiadis, A. Deep multiphysics: Coupling discrete multiphysics with machine learning to attain self-learning in-silico models replicating human physiology. *Artif. Intell. Med.* **2019**, *98*, 27–34. [CrossRef]
46. Chen, X. Simulation of 2D cavitation bubble growth under shear flow by lattice Boltzmann model. *Commun. Comput. Phys.* **2010**, *7*, 212. [CrossRef]
47. Nasiri, H.; Jamalabadi, M.Y.A.; Sadeghi, R.; Safaei, M.R.; Nguyen, T.K.; Shadloo, M.S. A smoothed particle hydrodynamics approach for numerical simulation of nano-fluid flows. *J. Therm. Anal. Calorim.* **2019**, *135*, 1733–1741. [CrossRef]
48. Plimpton, S. *Fast Parallel Algorithms for Short-Range Molecular Dynamics*; Technical Report; Sandia National Labs.: Albuquerque, NM, USA, 1993.
49. Ganzenmüller, G.C.; Steinhauser, M.O.; Van Liedekerke, P.; Leuven, K.U. The Implementation of Smooth Particle Hydrodynamics in LAMMPS. 2011. Available online: https://www.gpusph.org/documentation/gpusph-setup.pdf (accessed on 13 April 2021).
50. Stukowski, A. Visualization and analysis of atomistic simulation data with OVITO–the Open Visualization Tool. *Model. Simul. Mater. Sci. Eng.* **2009**, *18*, 015012. [CrossRef]
51. Zhang, S.; Duncan, J.H.; Chahine, G.L. The final stage of the collapse of a cavitation bubble near a rigid wall. *J. Fluid Mech.* **1993**, *257*, 147–181. [CrossRef]
52. Beig, S.; Aboulhasanzadeh, B.; Johnsen, E. Temperatures produced by inertially collapsing bubbles near rigid surfaces. *J. Fluid Mech.* **2018**, *852*, 105–125. [CrossRef]
53. Tomita, Y.; Shima, A. Mechanisms of impulsive pressure generation and damage pit formation by bubble collapse. *J. Fluid Mech.* **1986**, *169*, 535–564. [CrossRef]
54. Ghidaoui, M.S.; Zhao, M.; McInnis, D.A.; Axworthy, D.H. A review of water hammer theory and practice. *Appl. Mech. Rev.* **2005**, *58*, 49–76. [CrossRef]
55. Malvar, L.J.; Tancreto, J.E. *Analytical and Test Results for Water Mitigation of Explosion Effects*; Technical Report; Naval Facilities Engineering Service Center Port: Hueneme, CA, USA, 1998.

Chapter 8

Conclusions and future applications

8.1 Introduction

In this chapter we extend the 2D model used in Chapter 7 to study a non-symmetrical Rayleigh collapse with a stand-off of $\gamma = 0.6$ using 3D SPH model. With the 2D model we were able to simulate many collapsing mechanisms such as symmetrical empty cavity Rayleigh collapse, symmetrical gas filled cavity Rayleigh collapse with heat exchange, and non-symmetrical empty cavity Rayleigh collapse. However, the phenomenon is inherently three-dimensional and therefore a 3D model is required for a future DMP model.

8.2 Model

The 3D model uses the same set of discretised equations used in Chapter 7 with the Tait equation as EOS. Also the computational domain shares many similarities with the one represented in Figure 3 of Chapter 7.
The domain shown in Figure 8.1 is divided in three concentric regions, delimited by three different radii. Two types of computational particles are used:

- Cavity ($r < R_0$): is an empty region representing the collapsing bubble with $P_b = 0$ and $\rho_b = 0$.
- Liquid ($R_0 < r < R_S$): the particles inside the blue region in Figure 8.1 are modelled as water following the Tait EOS. The density is set as $\rho_L = 1000$ [kg m^{-3}] with initial pressure $P_\infty = 50$ [MPa].

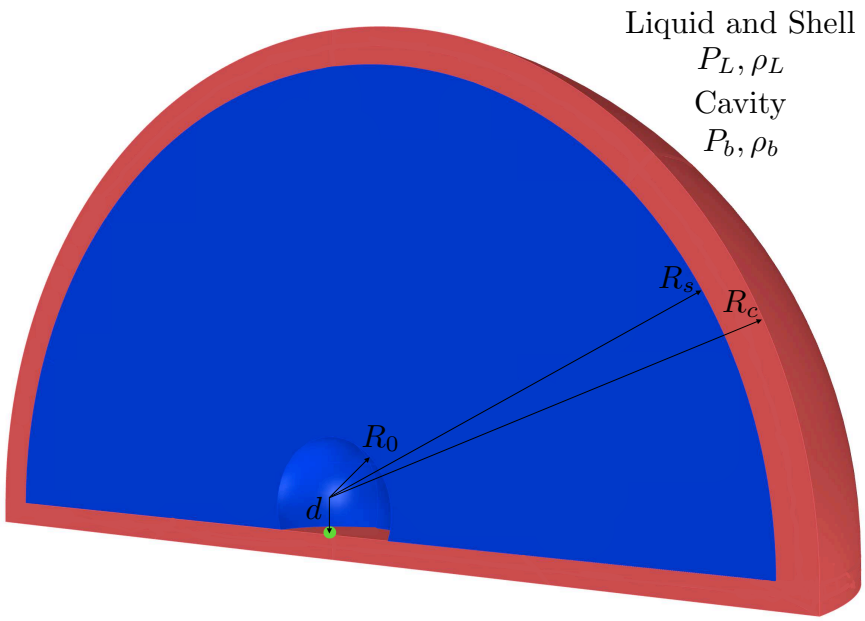


Figure 8.1: *Geometry of the simulation box*

- Shell ($r > R_S$): the particles inside the red region of Figure 8.1 are modelled as in the liquid region. However, they have fixed position and density to keep constant pressure as boundary condition. The lower part of this region also acts as an anisotropy driver wall inducing anisotropy in the pressure field during the collapse.

The green region, with a radius of 0.01 mm, is highlighted to show where the pressure generated during the collapse is monitored. In this Chapter we only focus a single case of non-symmetrical Rayleigh collapse with $\gamma = 0.6$. Unlike the previous models, the hydrodynamic of the collapse must be validated against the Rayleigh-Plesset equation of motion 2.10 for a symmetrical collapse.

Figure 8.2 shows the dimensionless radius evolution of a SPH empty spherical cavity with $P_\infty = 50$ [MPa] and $R_0 = 100\,[\mu\text{m}]$ against the numerical solution of Equation 2.10
The model is in good agreement with the theoretical solution. There is a certain difference in the final phase of the collapse where the particles resolution is not enough to preserve the spherical symmetry (this issue has been discussed in Chapter 6). The final non-dimensional collapse time, defined as ratio between the collapse time of the simulation and the Rayleigh collapse time, is $\tau = 1.08$.

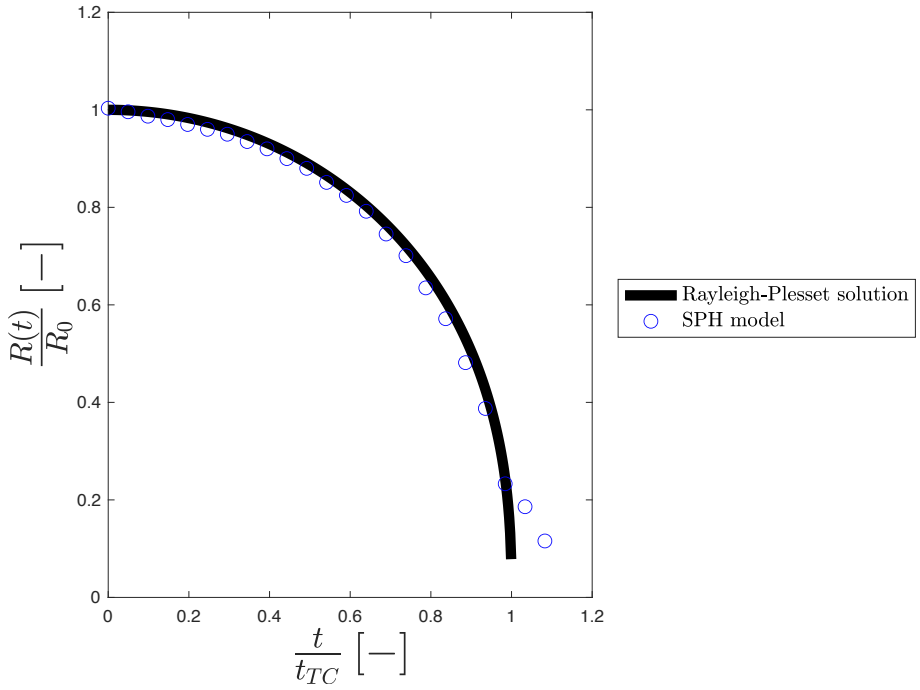


Figure 8.2: *Dimensionless ratio (R/R_0) against dimensionless time (t/t_c) for both SPH (blue circle dot) and the numerical solution of the Rayleigh-Plesset equation (continuum black curve) for the empty cavity collapse*

8.3 Results and discussions

The results shown in this section are obtained using the Lucy Kernel Liu and Liu [2003], and a smoothing length of $h = 1.3 \cdot dL$ where dL is the initial particle spacing. The dimensionless dissipation factor, time step, and speed of sound were set as $\alpha = 1$, $t_s = 1e{-}10$ [s] and $c_0 = 1484$ [m s^{-1}]. The resolution, the collapse driving force, and the magnitude of the anisotropic driver were chosen to be, respectively, dL/R_0=133, $P_\infty = 50$ [MPa] and $\gamma = 0.6$. Those values were chosen to match the ones used in Chapter 7.

8.3.1 Pressure trends

Figure 8.3 shows the pressure history over the green region highlighted in Figure 8.2.

The maximal pressure peak of the 3D model results to be approximately twice the maximal peak of the 2D model. The double peak behaviour is still present but the second peak results smaller than the first one, in contrast with the observed behaviour in the 2D model. In principle, differences between 2D and 3D models are expected since the collapse follows a different dynamic described, respectively, by Equation 2.14 and Equation 2.10.

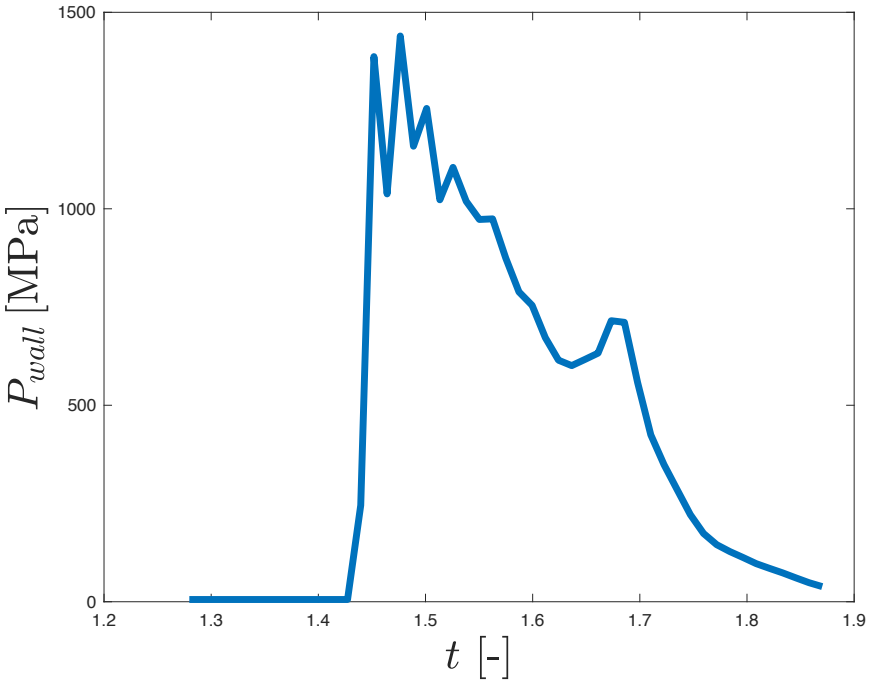


Figure 8.3: *Pressure trend over the green region of Figure 8.1 for a non-symmetrical wall attached collapse (dL/R_0=133, $\gamma = 0.6$ & $P_\infty = 50$ MPa).*

The difference in the maximal peak also reflect a different collapse time, $t_{3D} = 0.43\,[\mu s]$ for the 3D collapse and $t_{2D} = 0.86\,[\mu s]$ for the 2D. The shorter collapse time is directly correlated with the speed of the re-entrant jet. In fact, maximal jet speed at the impact is $V_{3D} \approx 1200$ [ms^{-1}] for the 3D model whereas for the 2D is $V_{2D} \approx 500$ [ms^{-1}]. A higher jet speed translates into a higher pressure peak when the jet impacts with the surface.

The difference in the double peak behaviour is investigated in the next section by comparing the Hydrodynamics in the last phase of the collapse.

8.3.2 Hydrodynamics comparison

Figure 8.4 shows a side-by-side comparison of the last phase for a high-resolution collapse between the 3D model and the 2D model

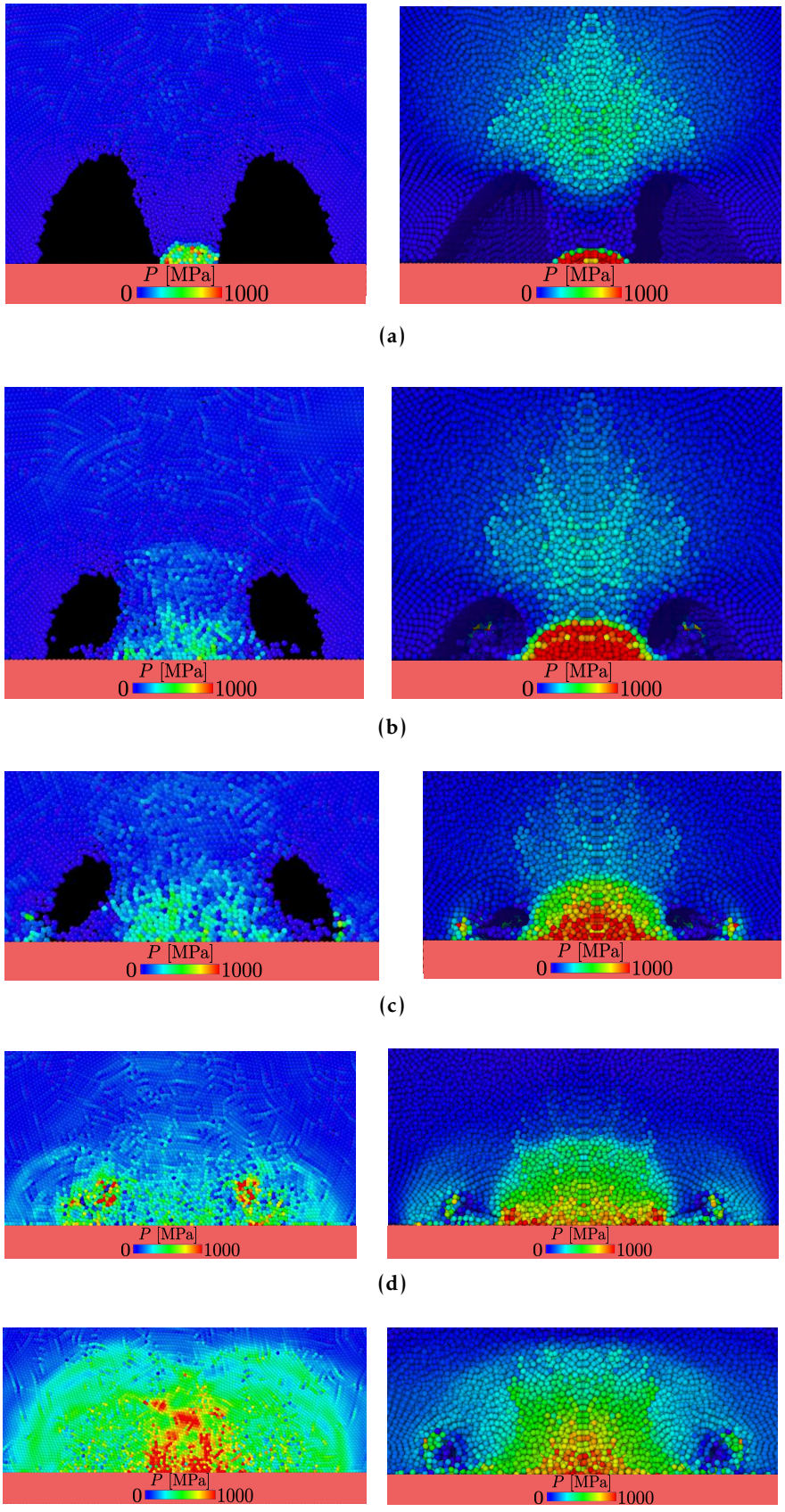


Figure 8.4: *Pressure field for 2D (left side) and 3D (right side) SPH model for a non symmetrical Rayleigh collapse (dL/R_0=133, γ = 0.6 & P_∞ = 50 MPa)*

In all the phases of the collapse, the 3D model shows a smoother pressure field. In the 2D model the particles are constrained in a plane and this may affect the smoothness of the pressure field but not affect other particles properties (see velocity field of Figure 7, 8, 9 of Chapter 7).

The 3D model shows many hydrodynamics features discussed in Chapter 7 such as: 1) formation of re-entrant jet (Figure 8.4a) 2) formation of lateral jets after the impact (Figure 8.4b) 3) ring formation (corrisponding to circle formation in 2D, see Figure 8.4c) 4) the generation of side pressure waves (Figure 8.4c). However, the third pair of pressure wave generated because of the ring collapse is not detected. The collapsing ring, unlike the collapsing circles, behaves as a vortex ring which dissipates the energy gained during the collapse by spinning and expanding around a central axis line, see Figure 8.5, as can be seen by the Video[1].

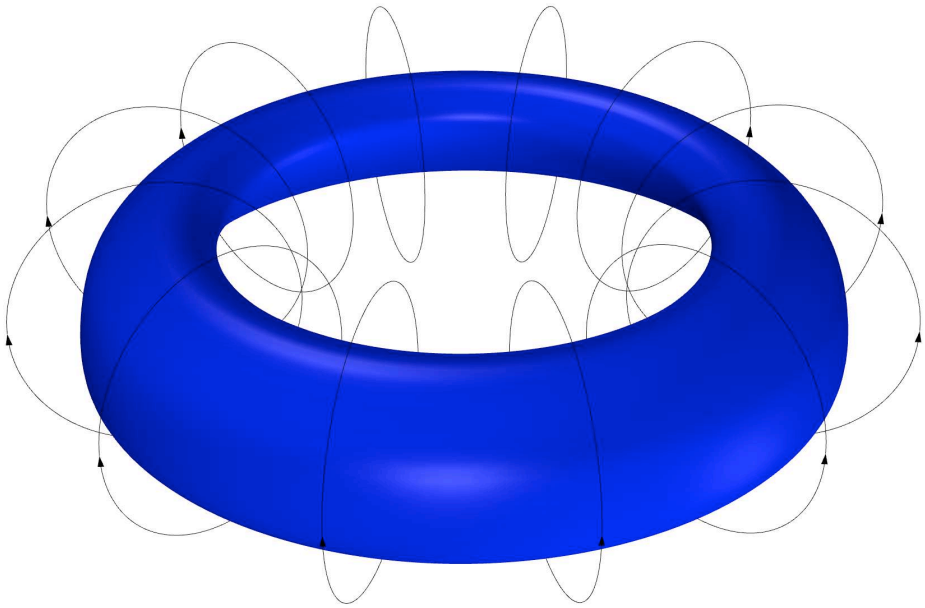


Figure 8.5: *Schematic representation collapsing vortex ring: blue surface represents the collapsing ring, black arrows represent the liquid flow.*

8.4 Conclusion and future applications

In this thesis we developed a SPH model with the aim of modelling the collapse phase of a single cavitating bubble. The model has been applied in different applications:

[1] The video of the last phase of the collapse is available at https://www.youtube.com/watch?v=9JCDHrv4UoU

- Chapter 5 we use the model to simulate a shock wave interacting with a discrete gas inhomogeneity. The model fits the underlying physics obtaining: (i) the correct shapes evolution, (ii) the correct timescale and (iii) the correct refraction/reflection of the wave for both light and heavy gas inhomogeneity.
- Chapter 6 the model was used to simulate a symmetrical collapsing cylindrical cavity filled with non-condensable gas coupled with the gas-liquid heat transfer mechanism. After introducing dimensionless group, Π_1, we were able to identify five regions where the temperature field of the gas-liquid system has different distribution.
- Chapter 7 the model was used to study the non-symmetrical Rayleigh collapse of an empty cylindrical cavity. With the model we were able to correctly simulate the jet physics and the consequent pressure fields at the surface.

Moreover, in this final chapter, we shown that the model can be extended in a 3D general improving the simulation output especially in the smoothness of the pressure field, compare Figure 8.6 with Figure 3 of Chapter 7, at cost of a dramatic increment of computational cost. The time required for run a single 3D simulation is 27hr instead of 4hr for the 2D model.

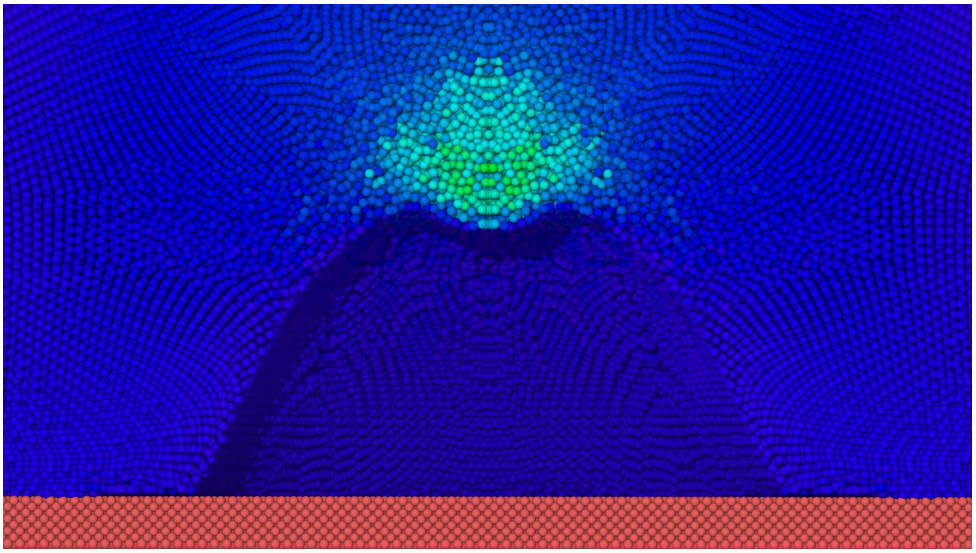


Figure 8.6: *Anisotropic pressure field in a 3D Rayleigh collapse of a wall attached cavity*

A natural development of this thesis is to couple SPH with Peridynamics to simulate a non symmetrical Rayleigh collapse of an attached cavity. The liquid would be simulated with the model presented in this thesis while the wall would be modelled with an on going Peridynamic model to simulate the erosion process induced by an empty cavity. The final step would be to

fill the cavity with a gas and, using the model discussed in Chapter 6, enabling a multi-hybrid gas-liquid/gas-solid/liquid-sold heat transfer for studying the effect of temperature on the erosion process.

Appendix

Appendix A

Chapter 5 LAMMPS input file

```
1
2  dimension          2
3  atom_style         hybrid meso atomic
4  boundary           s p p
5  units              si
6
7  ################### Gas domain ########################
8
9  #### x-direction
10 variable    xmin equal -0.5
11 variable    xmax equal  1
12
13 #### y-direction
14 variable    Gymin equal  0
15 variable    Gymax equal  0.089
16
17
18 ############### Wall domain  ##########################
19
20 variable    sp   equal  ${Gymax}*0.1-${Gymin}*0.1
21
22 ############### box domain  ##########################
23
24 variable    ymin equal  ${Gymin}-${sp}/2
25 variable    ymax equal  ${Gymax}+${sp}/2
26
27 variable    Lx    equal  ${xmax}-${xmin}
28 variable    Ly    equal  ${ymax}-${ymin}
```

```
29
30
31 ############### sphere position ##################
32
33 variable    xspe  equal  0.2725
34 variable    yspe  equal  0.0445
35 variable zspe  equal  0
36 variable    radi  equal  0.025
37
38 ############### Volume simulation box#########################
39
40 variable    Vtot  equal  ${Lx}*${Ly}
41
42 variable V1 equal    0.5*0.089
43 variable V3 equal 3.14*${radi}^2
44 variable V2 equal    ${xmax}*${Gymax}-${V3}
45
46
47 variable    dL   equal  0.00125
48 variable    dV   equal  ${dL}*${dL}*0.866
49
50 variable    Np1   equal 32841
51 variable    Np2   equal 64149
52 variable    Np3   equal 1451
53
54
55 ############### Ratio mass bubble and air ###########
56
57 variable r    equal  2.93
58
59 variable    rho1  equal  5
60 variable    rho2  equal  1
61 variable    rho3  equal  ${rho2}*${r}
62
63 variable    m1    equal  ${V1}*${rho1}/${Np1}
64 variable    m2    equal  ${V2}*${rho2}/${Np2}
65 variable    M    equal   ${V3}*${rho3}/${Np3}
66
67 variable prat  equal  20
68
69 variable    p2   equal  101325
```

```
variable    p3   equal  101325

variable    p1   equal  ${p2}*${prat}

variable    e1   equal  ${p1}*${m1}/0.4/${rho1}
variable    e2   equal  ${p2}*${m2}/0.4/${rho2}
variable    e3   equal  ${p3}*${M}/0.4/${rho3}

variable    h    equal  1.15*${dL}
variable    h1   equal  0.5*${dL}
variable    c    equal  400
variable    nu   equal  1.81e-5
variable    alpha equal 0.1

variable   skin   equal  0.3*${h}
variable   Nout   equal  1000
variable   dt  equal  1e-9

# create simulation box
region             box block ${xmin} ${xmax} ${ymin} ${ymax} -1.0e-4 1.0e-4 units box

create_box         4 box

lattice        hex ${dL}
create_atoms      1 box

region             left block EDGE 0.0 EDGE EDGE EDGE EDGE units box
region             right block 0.0000001 EDGE EDGE EDGE EDGE EDGE units box
region             hole sphere ${xspe} ${yspe} ${zspe} ${radi} side in units box
region             wallup block EDGE EDGE ${Gymax} EDGE EDGE EDGE units box
region             walldw block EDGE EDGE ${ymin} ${Gymin} EDGE EDGE units box


set            region right type 2
set            region hole type 3

delete_atoms      region wallup
delete_atoms      region walldw

create_atoms      4 region wallup
create_atoms      4 region walldw
```

```
111
112
113 mass                  1 ${m1}
114 mass                2 ${m2}
115 mass                  3 ${M}
116 mass                  4 ${M}
117
118
119
120 group    piston      type 1
121 group    gas              type 2
122 group    bubble      type 3
123 group    walls       type 4
124
125
126
127 set               type 1 meso/e ${e1}
128 set               type 2 meso/e ${e2}
129 set               type 3 meso/e ${e3}
130 set               type 4 meso/e ${e3}
131
132 set               type 1 meso/rho ${rho1}
133 set               type 2 meso/rho ${rho2}
134 set               type 3 meso/rho ${rho3}
135 set               type 4 meso/rho ${rho1}
136
137
138 pair_style         hybrid/overlay sph/rhosum 1 sph/idealgas lj/cut ${h}
139
140 pair_coeff         1 1 sph/rhosum ${h}
141 pair_coeff         1 2 sph/rhosum ${h}
142 pair_coeff         2 2 sph/rhosum ${h1}
143 pair_coeff         2 3 sph/rhosum ${h1}
144 pair_coeff         3 3 sph/rhosum ${h1}
145 pair_coeff         3 4 sph/rhosum ${h1}
146 pair_coeff         4 4 sph/rhosum ${h1}
147
148
149 pair_coeff         1 1 sph/idealgas ${alpha} ${h}
150 pair_coeff         1 2 sph/idealgas ${alpha} ${h}
151 pair_coeff         2 2 sph/idealgas ${alpha} ${h1}
```

```
pair_coeff         2 3 sph/idealgas ${alpha} ${h1}
pair_coeff         3 3 sph/idealgas ${alpha} ${h1}


pair_coeff     * 4 lj/cut 1.e-5 ${h}

compute            rhoatom all meso/rho/atom
compute            ieatom all meso/e/atom
compute            emeso all reduce sum c_ieatom
compute            ke all ke
variable            etot equal c_ke+c_emeso

dump               dump_id all custom ${Nout} "dump.lammpstrj" id type x y z vx vy c_rhoatom
    c_ieatom
dump_modify        dump_id first yes
neighbor       ${skin} bin


thermo             ${Nout}
thermo_style       custom step c_ke c_emeso v_etot
thermo_modify     norm no

fix 10 all meso
fix 20 all enforce2d
fix 30 walls meso/stationary
fix 40 walls setforce 0.0 0.0 0.0

timestep       ${dt}
run                 5000000
```

Listing A.1: *LAMMPS input file*

Appendix B

Chapter 6 LAMMPS input file

```
atom_style         hybrid meso atomic
dimension          2
units                   si
boundary           p p p


## Thermodynamic property of vapour

#### water

variable      C1 equal 73.649
variable      C2 equal -7258.2
variable      C3 equal -7.3037
variable      C4 equal 4.1653e-6
variable      C5 equal 2

variable      Tv equal 300

variable       Pv equal exp(${C1}+${C2}/${Tv}+${C3}*ln(${Tv})+${C4}*${Tv}^${C5})

variable       Mm equal 18

variable       R equal 8.314472*1000


###### parameters for NASG EOS

##### SPH/NASGLIQUID#######
```

```
variable       Cvl equal 3610
variable       Cpl equal 4285
variable       gammal equal 1.19
variable       P00 equal 7028e5
variable       b equal 6.61e-4
variable       ql equal -1177788
##### SPH/NASGLIQUID#######


##### SPH/NASGGAS
variable       Cvg equal 955
variable       Cpg equal 1401
variable       gammag equal 1.47
variable       qg equal 2077161
variable       q1g equal 14317
##### SPH/NASGGAS

variable       radius equal 1e-4
variable       radiusd equal  0.0015
variable       rshell equal 0.0030


###### Geometry and mass

variable               Xmin equal -0.0031
variable               Xmax equal  0.0031

variable               Ymin equal -0.0031
variable               Ymax equal  0.0031

variable               Lx equal ${Xmax}-${Xmin}
variable               Ly equal ${Ymax}-${Ymin}

variable       dL equal 0.000005

variable       Vtot equal 3.14*${rshell}^2
variable       V2 equal 3.14*${radius}^2
variable       V1 equal ${Vtot}-${V2}


variable       Np1 equal 1304330
```

```
70
71 variable               Np2 equal 1459
72
73
74 variable       rho0 equal 997.71
75 variable       rho1 equal 1000
76 variable       rho2 equal ${Pv}*${Mm}/(${R}*${Tv})
77
78 variable       cl equal 1484
79
80
81 variable       m1 equal ${V1}*${rho1}/${Np1}
82 variable             m2 equal ${V2}*${rho2}/${Np2}
83
84
85 ###### PRESSURE OF THE SYSTEM
86 variable       atm equal 101325
87
88 variable       P  equal  50*${atm}
89
90
91 variable       e1 equal ${P}*${m1}/0.4934/${rho1}
92 variable       e2 equal ${Pv}*${m2}/0.4/${rho2}
93
94
95 variable       h equal 1.30*${dL}
96 variable       hint equal 1.3*${dL}
97 variable       sigma equal 1*${dL}
98
99 variable          alpha equal 1
100
101 variable   skin   equal  0.3*${h}
102 variable   Nout   equal  1000
103 variable   dt  equal  1e-10
104
105
106 region             box block ${Xmin} ${Xmax} ${Ymin} ${Ymax} -1e-6 1e-6 units box
107 create_box     3 box
108 lattice             hex ${dL}
109
110 region             shellt sphere 0.0 0.0 0.0 ${rshell} side in units box
```

```
111
112 create_atoms       1 region shellt
113
114 region             cilindro sphere 0.0 0.0 0.0 ${radiusd} side in units box
115 set                  region cilindro type 2
116
117 region             hole sphere 0.0 0.0 0.0 ${radius} side in units box
118 set                  region hole type 3
119
120 group       shell type 1
121 group       liquid type 2
122 group       gas type 3
123 group       fluid type 2 3
124
125
126 mass               1 ${m1}
127 mass               2 ${m1}
128 mass               3 ${m2}
129
130 set                    type 1 meso/e ${e1}
131 set              type 2 meso/e ${e1}
132 set              type 3 meso/e ${e2} #con gaS
133
134
135 set                    type 1 meso/rho ${rho1}
136 set                    type 2 meso/rho ${rho1}
137 set                    type 3 meso/rho ${rho2} #con gas
138
139 pair_style         hybrid/overlay sph/rhosum 0 sph/taitwater sph/idealgas mie/cut ${sigma} sph/
        heatconduction sph/heatgasliquid
140
141 pair_coeff         * * sph/rhosum ${h}
142
143 pair_coeff        1 1 sph/taitwater ${rho0} ${cl} ${alpha} ${h}
144 pair_coeff        1 2 sph/taitwater ${rho0} ${cl} ${alpha} ${h}
145 pair_coeff        2 2 sph/taitwater ${rho0} ${cl} ${alpha} ${h}
146
147 pair_coeff        3 3 sph/idealgas ${alpha} ${h}
148
149 pair_coeff         2 3 mie/cut 1.e-6 ${sigma} 9 0
150
```

```
151
152 pair_coeff         1 1 sph/heatconduction 1.48e-7 ${h}
153 pair_coeff         1 2 sph/heatconduction 1.48e-7 ${h}
154 pair_coeff         2 2 sph/heatconduction 1.48e-7 ${h}
155 pair_coeff         2 3 sph/heatconduction 0 0
156 pair_coeff         3 3 sph/heatconduction 4.09e-4 ${h}
157
158
159 ######           liqtype || gatype || e0l || e0g || kg || kl || T0l || T0g || h || Cpl || Cpg
        || liqtype || gatype
160
161 pair_coeff         2 3 sph/heatgasliquid ${e1} ${e2} 0.02 0.61 300 300 ${hint} 4117.5 1914 2 3
162
163
164
165 compute            rhoatom all meso/rho/atom
166 compute           averagerho gas reduce ave c_rhoatom
167 compute            ieatom all meso/e/atom
168 compute           averageie gas reduce ave c_ieatom
169
170 variable             temperature    atom  (c_ieatom*0.40*18/(${m2}*8314.13))
171
172 compute           averagtemp gas reduce ave v_temperature
173 compute            emeso all reduce sum c_ieatom
174 compute            ke all ke
175 variable             etot equal c_ke+c_emeso
176
177
178 dump               dump_id all custom ${Nout} "dump.lammpstrj" id type x z y mass c_rhoatom
        c_ieatom v_temperature fx fy
179 dump_modify        dump_id first yes
180
181 neighbor           ${skin} bin
182 thermo             ${Nout}
183 thermo_style       custom step c_averagerho c_averageie c_averagtemp
184 thermo_modify      norm no
185
186 fix          integration_fix fluid  meso
187 fix         14 shell meso/stationary
188
189 timestep      ${dt}
```

```
190 run             30000
```

Listing B.1: *LAMMPS input file*

Appendix C

Chapter 7 LAMMPS input file

```
atom_style         hybrid meso atomic
dimension          2
units             si
boundary           p p p


###### Geometry and mass
variable       radius equal 1e-4
variable       radiusd equal  0.0015
variable       rshell equal 0.0016


variable               Xmin equal -0.0031
variable               Xmax equal  0.0031


variable               Ymin equal -0.0031
variable               Ymax equal  0.0031

variable               Lx equal ${Xmax}-${Xmin}
variable               Ly equal ${Ymax}-${Ymin}


variable       dL equal 0.00000075

variable       Vtot equal 3.14*${rshell}^2
variable       V2 equal 3.14*${radius}^2/2

```

```
variable       standoff equal 0
variable       d  equal -0.00021
variable       gamma equal -${standoff}*${radius}
variable       LC equal 2*sqrt(${rshell}^2-${d}^2)
variable       xlimit equal ${LC}/2
variable       theta equal 2*asin(${LC}/(2*${rshell}))
variable       VC equal ((${rshell}^2)/2)*(${theta}-sin(${theta}))

variable       V1 equal ${Vtot}-${V2}-${VC}

variable       Np1 equal 9599019
variable       rho0 equal 978.46
variable       rho1 equal 1000
variable       cl equal 1484
variable       m1 equal ${V1}*${rho1}/${Np1}


###### PRESSURE OF THE SYSTEM
variable       atm equal 101325
variable       P  equal   500*${atm}

variable       e1 equal   0

variable       h equal 1.30*${dL}
variable       sigma equal 1*${dL}
variable          alpha equal 1

variable   skin   equal  0.3*${h}
variable   Nout   equal  200
variable   dt  equal  1e-10


region             box block ${Xmin} ${Xmax} ${Ymin} ${Ymax} -1e-7 1e-7 units box
create_box         3 box
lattice            hex ${dL}

region             shellt sphere 0.0 0.0 0.0 ${rshell} side in units box

create_atoms       1 region shellt

region             cilindro sphere 0.0 0.0 0.0 ${radiusd} side in units box
```

```
set                region cilindro type 2

region             wall block -${xlimit} ${xlimit} ${d} ${gamma} -1e-7 1e-7 units box
set         region wall type 1

region             walld block ${Xmin} ${Xmax} ${Ymin} ${d} -1e-7 1e-7 units box
set                region walld type 3

region             hole sphere 0.0 0.0 0.0 ${radius} side in units box
set                region hole type 3

region             center block -5e-6 5e-6 -6.2e-7 6.2e-7 -1e-7 1e-7 units box


delete_atoms      region hole compress yes
delete_atoms      region walld compress yes
delete_atoms       region wall compress yes

create_atoms       1 region wall

group       shell type 1
group       liquid type 2


mass               1 ${m1}
mass               2 ${m1}

set                type 1 meso/e ${e1}
set         type 2 meso/e ${e1}

set                type 1 meso/rho ${rho1}
set                type 2 meso/rho ${rho1}

pair_style         hybrid/overlay sph/rhosum 0 sph/taitwater

pair_coeff         * * sph/rhosum ${ht}

pair_coeff         1 1 sph/taitwater ${rho0} ${cl} ${alpha} ${h}
pair_coeff         1 2 sph/taitwater ${rho0} ${cl} ${alpha} ${h}
pair_coeff         2 2 sph/taitwater ${rho0} ${cl} ${alpha} ${h}

```

```
111
112 compute             rhoatom all meso/rho/atom
113 compute     avPcenter shell reduce/region center ave c_rhoatom
114 compute             ieatom all meso/e/atom
115 compute     avEcenter shell reduce/region center ave c_ieatom
116 compute             emeso all reduce sum c_ieatom
117 compute             ke all ke
118 variable             etot equal c_ke+c_emeso
119
120
121 dump                dump_id all custom ${Nout} "dump.lammpstrj" id type x z y vx vy vz fx fy fz
        mass c_rhoatom c_ieatom
122 dump_modify         dump_id first yes
123
124 neighbor            ${skin} bin
125 thermo              ${Nout}
126 thermo_style        custom step c_ke c_emeso v_etot c_averagerho c_averageie c_averagtemp
        c_avPcenter c_avEcenter
127 thermo_modify       norm no
128
129 fix           integration_fix fluid  meso
130 fix          14 shell meso/stationary
131
132 timestep       ${dt}
133 run          10000
```

Listing C.1: *LAMMPS input file*

Appendix D

Chapter 8 LAMMPS input files

```

     atom_style         hybrid meso atomic
dimension          3
units            si
boundary           p p p


#
###### geometry
variable               Xmin equal -0.0012
variable               Xmax equal  0.0012

variable               Ymin equal -0.0012
variable               Ymax equal  0.0012

variable       radius equal 1e-4
variable       radiusd equal  0.0003
variable       rshell equal 0.00031


##### simulation parameter
variable       atm equal 101325

variable       P  equal  500*${atm}

variable       dL equal 0.0000025

variable       standoff equal 0.6
```

```
variable       gamma equal -${standoff}*${radius}
#variable         d  equal -0.00021  ###
variable       d  equal -1e-5+${gamma} ###
variable       xlimit equal sqrt(${rshell}^2-${gamma}^2)

variable       Vtot equal (4*3.14*${rshell}^3)/3
variable       V2 equal (4*3.14*${radius}^3)/3*0.6
variable       VC equal (3.14*(${rshell}-${gamma})^2)*(${rshell}-(${rshell}-${gamma})/3)
variable       V1 equal ${Vtot}+3.14e-5*${gamma}^2-${V2}-${VC}

variable       Np1 equal 20292752

##### sph variables
variable       rho0 equal 978.46
variable       rho1 equal 1000

variable       e1 equal ${P}*${m1}/0.4934/${rho1}
variable       m1 equal ${V1}*${rho1}/${Np1}

variable       h equal 1.30*${dL}
variable       cl equal 1484
variable          alpha equal 1

variable   skin   equal  0.3*${h}
variable   Nout   equal  200
variable   dt  equal  1e-10

#### simulation box
region             box block ${Xmin} ${Xmax} ${Ymin} ${Ymax} ${Ymin} ${Ymax} units box
create_box         3 box
lattice            fcc ${dL}

#### outer sphere
region             shellt sphere 0.0 0.0 0.0 ${rshell} side in units box
create_atoms       1 region shellt

##### fluid domain
region             cilindro sphere 0.0 0.0 0.0 ${radiusd} side in units box
set                region cilindro type 2

#### wall thickness
```

```
region          wall cylinder y 0.0 0.0 ${xlimit} ${d} ${gamma} units box
set         region wall type 1

#### Wall to be deleted
region            walld cylinder y 0.0 0.0 ${xlimit} ${Ymin} ${d} units box
set                region walld type 3

###### cavity region
region             hole sphere 0.0 0.0 0.0 ${radius} side in units box
set                region hole type 3

delete_atoms      region hole compress yes
delete_atoms      region walld compress yes
delete_atoms       region wall compress yes

create_atoms       1 region wall

group       shell type 1
group       liquid type 2
group       fluid type 2


mass               1 ${m1}
mass               2 ${m1}

set                type 1 meso/e ${e1}
set         type 2 meso/e ${e1}

set                type 1 meso/rho ${rho1}
set                type 2 meso/rho ${rho1}

pair_style         hybrid/overlay sph/rhosum 0 sph/taitwater

pair_coeff         * * sph/rhosum ${h}

pair_coeff         1 1 sph/taitwater ${rho0} ${cl} ${alpha} ${h}
pair_coeff         1 2 sph/taitwater ${rho0} ${cl} ${alpha} ${h}
pair_coeff         2 2 sph/taitwater ${rho0} ${cl} ${alpha} ${h}

compute            rhoatom all meso/rho/atom
compute            ieatom all meso/e/atom
```

```
compute             emeso all reduce sum c_ieatom
compute             ke all ke
variable            etot equal c_ke+c_emeso

dump                dump_id all custom ${Nout} "dumpfile.lammpstrj" id type x z y vx vy vz fx fy
     fz mass c_rhoatom c_ieatom
dump_modify         dump_id first yes

neighbor            ${skin} bin
thermo              ${Nout}
thermo_style        custom step
thermo_modify       norm no

fix          integration_fix fluid  meso

fix         14 shell meso/stationary
timestep            ${dt}

run         7000

##########
label    here

variable   Nout   equal 50

dump                dump_id all custom ${Nout} "final_dumpfile.lammpstrj"  id type x z y vx vy
    vz fx fy fz mass c_rhoatom c_ieatom
dump_modify         dump_id first yes


run  600

##########
```

Listing D.1: *LAMMPS input file*

Bibliography

Albano, A. and Alexiadis, A. (2019). Interaction of shock waves with discrete gas inhomogeneities: A smoothed particle hydrodynamics approach. *Applied Sciences*, 9(24):5435.

Albano, A. and Alexiadis, A. (2020). A smoothed particle hydrodynamics study of the collapse for a cylindrical cavity. *PLOS ONE*, 15(9):1–22.

Albano, A. and Alexiadis, A. (2021). Non-symmetrical collapse of an empty cylindrical cavity studied with smoothed particle hydrodynamics. *Applied Sciences*, 11(8):3500.

Alexiadis, A. (2014). A smoothed particle hydrodynamics and coarse-grained molecular dynamics hybrid technique for modelling elastic particles and breakable capsules under various flow conditions. *International Journal for Numerical Methods in Engineering*, 100(10):713–719.

Alexiadis, A. (2015a). The discrete multi-hybrid system for the simulation of solid-liquid flows. *PloS one*, 10(5):e0124678.

Alexiadis, A. (2015b). A new framework for modelling the dynamics and the breakage of capsules, vesicles and cells in fluid flow. *Procedia IUTAM*, 16:80–88.

Alexiadis, A. (2019a). Deep multiphysics and particle–neuron duality: A computational framework coupling (discrete) multiphysics and deep learning. *Applied Sciences*, 9(24):5369.

Alexiadis, A. (2019b). Deep multiphysics: Coupling discrete multiphysics with machine learning to attain self-learning in-silico models replicating human physiology. *Artificial intelligence in medicine*, 98:27–34.

Alexiadis, A., Albano, A., Rahmat, A., Yildiz, M., Kefal, A., Ozbulut, M., Bakirci, N., Garzón-Alvarado, D., Duque-Daza, C., and Eslava-Schmalbach, J. (2021). Simulation of pandemics in real cities: enhanced and accurate digital laboratories. *Proceedings of the Royal Society A*, 477(2245):20200653.

Alexiadis, A., Ghraybeh, S., and Qiao, G. (2018). Natural convection and solidification of phase-change materials in circular pipes: A sph approach. *Computational Materials Science*, 150:475–483.

Alexiadis, A., Simmons, M., Stamatopoulos, K., Batchelor, H., and Moulitsas, I. (2020). The duality between particle methods and artificial neural networks. *Scientific Reports*, 10(1):1–7.

Alexiadis, A., Stamatopoulos, K., Wen, W., Batchelor, H., Bakalis, S., Barigou, M., and Simmons, M. (2017). Using discrete multi-physics for detailed exploration of hydrodynamics in an in vitro colon system. *Computers in biology and medicine*, 81:188–198.

Anagnostopoulos, A., Navarro, H., Alexiadis, A., and Ding, Y. (2020). Wettability of nano3 and kno3 on mgo and carbon surfaces?understanding the substrate and the length scale effects. *The Journal of Physical Chemistry C*, 124(15):8140–8152.

Ariane, M., Allouche, M. H., Bussone, M., Giacosa, F., Bernard, F., Barigou, M., and Alexiadis, A. (2017a). Discrete multi-physics: A mesh-free model of blood flow in flexible biological valve including solid aggregate formation. *PloS one*, 12(4):e0174795.

Ariane, M., Kassinos, S., Velaga, S., and Alexiadis, A. (2018a). Discrete multi-physics simulations of diffusive and convective mass transfer in boundary layers containing motile cilia in lungs. *Computers in biology and medicine*, 95:34–42.

Ariane, M., Sommerfeld, M., and Alexiadis, A. (2018b). Wall collision and drug-carrier detachment in dry powder inhalers: using dem to devise a sub-scale model for cfd calculations. *Powder Technology*, 334:65–75.

Ariane, M., Vigolo, D., Brill, A., Nash, F., Barigou, M., and Alexiadis, A. (2018c). Using discrete multi-physics for studying the dynamics of emboli in flexible venous valves. *Computers & Fluids*, 166:57–63.

Ariane, M., Wen, W., Vigolo, D., Brill, A., Nash, F., Barigou, M., and Alexiadis, A. (2017b). Modelling and simulation of flow and agglomeration in deep veins valves using discrete multi physics. *Computers in biology and medicine*, 89:96–103.

Arndt, R., Arakeri, V., and Higuchi, H. (1991). Some observations of tip-vortex cavitation. *Journal of fluid mechanics*, 229:269–289.

Arndt, R. E. (1981). Cavitation in fluid machinery and hydraulic structures. *Annual Review of Fluid Mechanics*, 13(1):273–326.

Atomic, L.-s. and Simulator, M. M. P. (2003). Lammps users manual.

Auhl, R., Everaers, R., Grest, G. S., Kremer, K., and Plimpton, S. J. (2003). Equilibration of long chain polymer melts in computer simulations. *The Journal of chemical physics*, 119(24):12718–12728.

Batchelor, C. K. and Batchelor, G. (2000). *An introduction to fluid dynamics*. Cambridge university press.

Beig, S., Aboulhasanzadeh, B., and Johnsen, E. (2018). Temperatures produced by inertially collapsing bubbles near rigid surfaces. *Journal of Fluid Mechanics*, 852:105–125.

Benjamin, T. B. (1958). Pressure waves from collapsing cavities. In *2nd Symposium on Naval Hydrodynamics*, pages 207–229.

Benjamin, T. B. and Ellis, A. T. (1966). The collapse of cavitation bubbles and the pressures thereby produced against solid boundaries. *Philosophical Transactions for the Royal Society of London. Series A, Mathematical and Physical Sciences*, pages 221–240.

Benson, D. J. (1992). Computational methods in lagrangian and eulerian hydrocodes. *Computer methods in Applied mechanics and Engineering*, 99(2-3):235–394.

Benz, W. (1990). Smooth particle hydrodynamics: a review. In *The numerical modelling of nonlinear stellar pulsations*, pages 269–288. Springer.

Besant, W. H. (1859). *A treatise on hydrostatics and hydrodynamics*. Deighton, Bell.

Bird, R. B. (2002). Transport phenomena. *Appl. Mech. Rev.*, 55(1):R1–R4.

Blake, J., Taib, B., and Doherty, G. (1986). Transient cavities near boundaries. part 1. rigid boundary. *Journal of Fluid Mechanics*, 170:479–497.

Bourne, N. and Field, J. (1992). Shock-induced collapse of single cavities in liquids. *Journal of Fluid Mechanics*, 244:225–240.

Bratley, P., Fox, B. L., and Schrage, L. E. (2011). *A guide to simulation*. Springer Science & Business Media.

Brennen, C. E. (2014). *Cavitation and bubble dynamics*. Cambridge University Press.

Chapman, S., Cowling, T. G., and Burnett, D. (1990). *The mathematical theory of non-uniform gases: an account of the kinetic theory of viscosity, thermal conduction and diffusion in gases*. Cambridge university press.

Chaussonnet, G., Braun, S., Wieth, L., Koch, R., and Bauer, H.-J. (2015). Influence of particle disorder and smoothing length on sph operator accuracy. In *Tenth International SPHERIC Workshop, Parma, Italy, June*, pages 16–18.

Chen, X. (2010). Simulation of 2d cavitation bubble growth under shear flow by lattice boltzmann model. *Communications in Computational Physics*, 7(1):212.

Coleman, S., Spearot, D., and Capolungo, L. (2013). Virtual diffraction analysis of ni [0 1 0] symmetric tilt grain boundaries. *Modelling and Simulation in Materials Science and Engineering*, 21(5):055020.

Cornelissen, J. and Waterman, H. (1955). The viscosity temperature relationship of liquids. *Chemical Engineering Science*, 4(5):238–246.

Courant, R., Friedrichs, K., and Lewy, H. (1928). Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische annalen*, 100(1):32–74.

Daraio, D., Villoria, J., Ingram, A., Alexiadis, A., Stitt, E. H., Munnoch, A. L., and Marigo, M. (2020). Using discrete element method (dem) simulations to reveal the differences in the γ-al2o3 to α-al2o3 mechanically induced phase transformation between a planetary ball mill and an attritor mill. *Minerals Engineering*, 155:106374.

De Chizelle, Y. K., Ceccio, S., and Brennen, C. (1995). Observations and scaling of travelling bubble cavitation. *Journal of Fluid Mechanics*, 293:99–126.

De Guzman, J. (1913). Relation between fluidity and heat of fusion. *Anales Soc. Espan. Fis. Y. Quim*, 11:353–362.

Duarte, C. A. and Oden, J. T. (1996). An hp adaptive method using clouds. *Computer methods in applied mechanics and engineering*, 139(1-4):237–262.

Ehigiamusoe, N. N., Maxutov, S., and Lee, Y. C. (2018). Modeling surface tension of a two-dimensional droplet using smoothed particle hydrodynamics. *International Journal for Numerical Methods in Fluids*, 88(7):334–346.

Epstein, D. and Keller, J. B. (1972). Expansion and contraction of planar, cylindrical, and spherical underwater gas bubbles. *The Journal of the Acoustical Society of America*, 52(3B):975–980.

Fan, E., Guan, B., Wen, C.-Y., and Shen, H. (2019). Numerical study on the jet formation of simple-geometry heavy gas inhomogeneities. *Physics of Fluids*, 31(2):026103.

Ferraro, G., Jadhav, A. J., and Barigou, M. (2020). A henry's law method for generating bulk nanobubbles. *Nanoscale*, 12(29):15869–15879.

Flannigan, D. J. and Suslick, K. S. (2005). Plasma formation and temperature measurement during single-bubble cavitation. *Nature*, 434(7029):52–55.

Flannigan, D. J. and Suslick, K. S. (2010). Inertially confined plasma in an imploding bubble. *Nature Physics*, 6(8):598–601.

Flügge, W. (2013). *Viscoelasticity*. Springer Science & Business Media.

Fostiropoulos, S. R., Malgarinos, I., Strotos, G., Nikolopoulos, N., Kakaras, E., Koukouvinis, P., and Gavaises, M. (2017). Role of heat transfer in bubble dynamics neglecting phase change. a numerical study. In *Ilass Europe. 28th european conference on Liquid Atomization and Spray Systems*, pages 904–911. Editorial Universitat Politècnica de València.

Franc, J.-P. and Michel, J.-M. (2006). *Fundamentals of cavitation*, volume 76. Springer science & Business media.

Franc, J.-P., Riondet, M., Karimi, A., and Chahine, G. L. (2011). Impact load measurements in an erosive cavitating flow. *Journal of Fluids Engineering*, 133(12):121301.

Fujikawa, S. and Akamatsu, T. (1980). Effects of the non-equilibrium condensation of vapour on the pressure wave produced by the collapse of a bubble in a liquid. *Journal of Fluid Mechanics*, 97(3):481–512.

Ganzenmüller, G. C., Steinhauser, M. O., Van Liedekerke, P., and Leuven, K. U. (2011). The implementation of smooth particle hydrodynamics in lammps. *Paul Van Liedekerke Katholieke Universiteit Leuven*, 1:1–26.

Georgievskiy, P. Y., Levin, V., and Sutyrin, O. (2015). Interaction of a shock with elliptical gas bubbles. *Shock Waves*, 25(4):357–369.

Ghidaoui, M. S., Zhao, M., McInnis, D. A., and Axworthy, D. H. (2005). A review of water hammer theory and practice. *Appl. Mech. Rev.*, 58(1):49–76.

Giannadakis, E., Gavaises, M., and Arcoumanis, C. (2008). Modelling of cavitation in diesel injector nozzles. *Journal of Fluid Mechanics*, 616:153–193.

Gingold, R. A. and Monaghan, J. J. (1977). Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389.

Giordano, J. and Burtschell, Y. (2006). Richtmyer-meshkov instability induced by shock-bubble interaction: Numerical and analytical studies with experimental validation. *Physics of Fluids*, 18(3):036102.

Gutmann, F. and Simmons, L. (1952). The temperature dependence of the viscosity of liquids. *Journal of Applied Physics*, 23(9):977–978.

Haas, J.-F. and Sturtevant, B. (1987). Interaction of weak shock waves with cylindrical and spherical gas inhomogeneities. *Journal of Fluid Mechanics*, 181:41–76.

Henderson, L. F., Colella, P., and Puckett, E. G. (1991). On the refraction of shock waves at a slow–fast gas interface. *Journal of Fluid Mechanics*, 224:1–27.

Hickling, R. (1963). Effects of thermal conduction in sonoluminescence. *The Journal of the Acoustical Society of America*, 35(7):967–974.

Hickling, R. and Plesset, M. S. (1964). Collapse and rebound of a spherical bubble in water. *The Physics of Fluids*, 7(1):7–14.

Holl, J. W. (1970). Nuclei and cavitation.

Hopp-Hirschler, M., Shadloo, M. S., and Nieken, U. (2018). A smoothed particle hydrodynamics approach for thermo-capillary flows. *Computers & Fluids*, 176:1–19.

Hosseini, S., Manzari, M., and Hannani, S. (2007). A fully explicit three-step sph algorithm for simulation of non-newtonian fluid flow. *International Journal of Numerical Methods for Heat & Fluid Flow*.

Huang, B., Zhao, Y., and Wang, G. (2014). Large eddy simulation of turbulent vortex-cavitation interactions in transient sheet/cloud cavitating flows. *Computers & Fluids*, 92:113–124.

Jacobs, J. (1993). The dynamics of shock accelerated light and heavy gas cylinders. *Physics of Fluids A: Fluid Dynamics*, 5(9):2239–2247.

Jaramillo-Botero, A., Su, J., Qi, A., and Goddard III, W. A. (2011). Large-scale, long-term nonadiabatic electron molecular dynamics for describing material properties and phenomena in extreme environments. *Journal of computational chemistry*, 32(3):497–512.

Johnsen, E. and Colonius, T. (2008). Shock-induced collapse of a gas bubble in shockwave lithotripsy. *The Journal of the Acoustical Society of America*, 124(4):2011–2020.

Johnsen, E. and Colonius, T. (2009). Numerical simulations of non-spherical bubble collapse. *Journal of fluid mechanics*, 629:231–262.

Johnson, G. R., Stryk, R. A., and Beissel, S. R. (1996). Sph for high velocity impact computations. *Computer methods in applied mechanics and engineering*, 139(1-4):347–373.

Joshi, S., Franc, J. P., Ghigliotti, G., and Fivel, M. (2019). Sph modelling of a cavitation bubble collapse near an elasto-visco-plastic material. *Journal of the Mechanics and Physics of Solids*, 125:420–439.

Joshi, S., Franc, J. P., Ghigliotti, G., and Fivel, M. (2020). An axisymmetric solid sph solver with consistent treatment of particles close to the symmetry axis. *Computational Particle Mechanics*, pages 1–15.

Karimi, A. and Martin, J. (1986). Cavitation erosion of materials. *International Metals Reviews*, 31(1):1–26.

Keller, J. B. and Miksis, M. (1980). Bubble oscillations of large amplitude. *The Journal of the Acoustical Society of America*, 68(2):628–633.

Kim, K.-H., Chahine, G., Franc, J.-P., and Karimi, A. (2014). *Advanced experimental and numerical techniques for cavitation erosion prediction*, volume 106. Springer.

Klaseboer, E., Turangan, C., Fong, S. W., Liu, T. G., Hung, K. C., and Khoo, B. C. (2006). Simulations of pressure pulse–bubble interaction using boundary element method. *Computer methods in applied mechanics and engineering*, 195(33-36):4287–4302.

Kling, C. L. and Hammitt, F. G. (1970). A photographic study of spark-induced cavitation bubble collapse. Technical report.

Knapp, R., Daily, J., and Hammitt, F. (1970). Cavitation, eng. soc. *Monographs McGraw-Hill NY*, 39.

Kornfeld, M. and Suvorov, L. (1944). On the destructive action of cavitation. *Journal of Applied Physics*, 15(6):495–506.

Kudryashov, N. A. and Sinelshchikov, D. I. (2014). Analytical solutions of the rayleigh equation for empty and gas-filled bubble. *Journal of Physics A: Mathematical and Theoretical*, 47(40):405202.

Kudryashov, N. A. and Sinelshchikov, D. I. (2015). Analytical solutions for problems of bubble dynamics. *Physics Letters A*, 379(8):798–802.

Lattanzio, J., Monaghan, J., Pongracic, H., and Schwarz, M. (1986). Controlling penetration. *SIAM Journal on Scientific and Statistical Computing*, 7(2):591–598.

Lauterborn, W. and Bolle, H. (1975). Experimental investigations of cavitation-bubble collapse in the neighbourhood of a solid boundary. *Journal of Fluid Mechanics*, 72(2):391–399.

Layes, G., Jourdan, G., and Houas, L. (2005). Experimental investigation of the shock wave interaction with a spherical gas inhomogeneity. *Physics of fluids*, 17(2):028103.

Layes, G., Jourdan, G., and Houas, L. (2009). Experimental study on a plane shock wave accelerating a gas bubble. *Physics of Fluids*, 21(7):074102.

Layes, G. and Le Métayer, O. (2007). Quantitative numerical and experimental studies of the shock accelerated heterogeneous bubbles motion. *Physics of Fluids*, 19(4):042105.

Le Métayer, O. and Saurel, R. (2016). The noble-abel stiffened-gas equation of state. *Physics of Fluids*, 28(4):046102.

Levy, K., Sadot, O., Rikanati, A., Kartoon, D., Srebro, Y., Yosef-Hai, A., Ben-Dor, G., and Shvarts, D. (2003). Scaling in the shock–bubble interaction. *Laser and Particle Beams*, 21(3):335–339.

Li, C.-Y. and Ceccio, S. L. (1996). Interaction of single travelling bubbles with the boundary layer and attached cavitation. *Journal of Fluid Mechanics*, 322:329–353.

Li, S. and Liu, W. K. (2002). Meshfree and particle methods and their applications. *Appl. Mech. Rev.*, 55(1):1–34.

Liu, G.-R. and Liu, M. B. (2003). *Smoothed particle hydrodynamics: a meshfree particle method*. World scientific.

Liu, M. and Liu, G. (2010). Smoothed particle hydrodynamics (sph): an overview and recent developments. *Archives of computational methods in engineering*, 17(1):25–76.

Liu, M., Liu, G., and Lam, K. (2002). Investigations into water mitigation using a meshless particle method. *Shock waves*, 12(3):181–195.

Liu, M., Liu, G., Lam, K., and Zong, Z. (2003a). Smoothed particle hydrodynamics for numerical simulation of underwater explosion. *Computational Mechanics*, 30(2):106–118.

Liu, M., Liu, G., Zong, Z., and Lam, K. (2003b). Computer simulation of high explosive explosion using smoothed particle hydrodynamics methodology. *Computers & Fluids*, 32(3):305–322.

Lucy, L. B. (1977). A numerical approach to the testing of the fission hypothesis. *The astronomical journal*, 82:1013–1024.

Luo, X.-w., Ji, B., and Tsujimoto, Y. (2016). A review of cavitation in hydraulic machinery. *Journal of Hydrodynamics*, 28(3):335–358.

Malvar, L. J. and Tancreto, J. E. (1998). Analytical and test results for water mitigation of explosion effects. Technical report, NAVAL FACILITIES ENGINEERING SERVICE CENTER PORT HUENEME CA.

Martin, K. and Hoffman, B. (2010). *Mastering CMake: a cross-platform build system*. Kitware.

Mauri, R. (2015). *Transport phenomena in multiphase flows*, volume 112. Springer.

Medvedev, S., Khomik, S., Cherepanova, T., Agafonov, G., Cherepanov, A., Mikhalkin, V., Kiverin, A., Petukhov, V., Yakovenko, I., and Betev, A. (2019). Interaction of blast waves with helium-filled rubber balloons. In *Journal of Physics: Conference Series*, volume 1147, page 012021. IOP Publishing.

Merouani, S., Hamdaoui, O., Rezgui, Y., and Guemini, M. (2014). Theoretical estimation of the temperature and pressure within collapsing acoustical bubbles. *Ultrasonics sonochemistry*, 21(1):53–59.

Mohammed, A. M., Ariane, M., and Alexiadis, A. (2020). Using discrete multiphysics modelling to assess the effect of calcification on hemodynamic and mechanical deformation of aortic valve. *ChemEngineering*, 4(3):48.

Molteni, D. and Bilello, C. (2003). Riemann solver in sph. *Memorie della Societa Astronomica Italiana Supplementi*, 1:36.

Monaghan, J. (1997). Sph and riemann solvers. *Journal of Computational Physics*, 136(2):298–307.

Monaghan, J. and Gingold, R. A. (1983). Shock simulation by the particle method sph. *Journal of computational physics*, 52(2):374–389.

Monaghan, J. and Kocharyan, A. (1995). Sph simulation of multi-phase flow. *Computer Physics Communications*, 87(1-2):225–235.

Monaghan, J. J. (1994). Simulating free surface flows with sph. *Journal of computational physics*, 110(2):399–406.

Mørch, K. A. (2009). Cavitation nuclei: experiments and theory. *Journal of Hydrodynamics*, 21(2):176–189.

Mørch, K. A. (2015). Cavitation inception from bubble nuclei. *Interface Focus*, 5(5):20150006.

Morris, J. and Monaghan, J. (1997). A switch to reduce sph viscosity. *Journal of Computational Physics*, 136(1):41–50.

Morris, J. P. (1996). *Analysis of smoothed particle hydrodynamics with applications*. Monash University Australia.

Morris, J. P., Fox, P. J., and Zhu, Y. (1997). Modeling low reynolds number incompressible flows using sph. *Journal of computational physics*, 136(1):214–226.

Moss, W. C., Levatin, J. L., and Szeri, A. J. (2000). A new damping mechanism in strongly collapsing bubbles. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 456(2004):2983–2994.

Mourad, P. D., Roberts, F. A., and McInnes, C. (2007). Synergistic use of ultrasound and sonic motion for removal of dental plaque bacteria. *Compendium of continuing education in dentistry (Jamesburg, NJ: 1995)*, 28(7):354–358.

Mubin, S. and Li, J. (2021). *Extending and Modifying LAMMPS*. Packt Publishing Ltd.

Munjiza, A. A. (2004). *The combined finite-discrete element method*. John Wiley & Sons.

Nair, P. and Tomar, G. (2019). Simulations of gas-liquid compressible-incompressible systems using sph. *Computers & Fluids*, 179:301–308.

Nasiri, H., Jamalabadi, M. Y. A., Sadeghi, R., Safaei, M. R., Nguyen, T. K., and Shadloo, M. S. (2019). A smoothed particle hydrodynamics approach for numerical simulation of nano-fluid flows. *Journal of Thermal Analysis and Calorimetry*, 135(3):1733–1741.

Naude, C. F. and Ellis, A. T. (1961). On the mechanism of cavitation damage by nonhemispherical cavities collapsing in contact with a solid boundary.

Nayroles, B., Touzot, G., and Villon, P. (1992). Generalizing the finite element method: diffuse approximation and diffuse elements. *Computational mechanics*, 10(5):307–318.

Ng, K., Alexiadis, A., Chen, H., and Sheu, T. (2020). A coupled smoothed particle hydrodynamics-volume compensated particle method (sph-vcpm) for fluid structure interaction (fsi) modelling. *Ocean Engineering*, 218:107923.

Nguyen, V. P., Rabczuk, T., Bordas, S., and Duflot, M. (2008). Meshless methods: a review and computer implementation aspects. *Mathematics and computers in simulation*, 79(3):763–813.

Niazi, S., Hashemabadi, S. H., and Razi, M. M. (2014). Cfd simulation of acoustic cavitation in a crude oil upgrading sonoreactor and prediction of collapse temperature and pressure of a cavitation bubble. *Chemical Engineering Research and Design*, 92(1):166–173.

Obreschkow, D., Bruderer, M., and Farhat, M. (2012). Analytical approximations for the collapse of an empty spherical bubble. *Physical Review E*, 85(6):066303.

Obreschkow, D., Tinguely, M., Dorsaz, N., Kobel, P., De Bosset, A., and Farhat, M. (2013). The quest for the most spherical bubble: experimental setup and data overview. *Experiments in Fluids*, 54(4):1503.

O'hern, T. (1990). An experimental investigation of turbulent shear flow cavitation. *Journal of fluid mechanics*, 215:365–391.

Oñate, E., Idelsohn, S., Zienkiewicz, O., and Taylor, R. (1996). A finite point method in computational mechanics. applications to convective transport and fluid flow. *International journal for numerical methods in engineering*, 39(22):3839–3866.

Packer, M., Fishkind, W. J., Fine, I. H., Seibel, B. S., and Hoffman, R. S. (2005). The physics of phaco: a review. *Journal of Cataract & Refractive Surgery*, 31(2):424–431.

Paik, B.-G., Kim, K.-Y., Ahn, J.-W., Kim, Y.-S., Kim, S.-P., and Park, J.-J. (2008). Experimental study on the gap entrance profile affecting rudder gap cavitation. *Ocean Engineering*, 35(1):139–149.

Parks, M. L., Lehoucq, R. B., Plimpton, S. J., and Silling, S. A. (2008). Implementing peridynamics within a molecular dynamics code. *Computer Physics Communications*, 179(11):777–783.

Pazdniakou, A. and Adler, P. (2012). Lattice spring models. *Transport in porous media*, 93(2):243–262.

Petersen, M. K., Lechman, J. B., Plimpton, S. J., Grest, G. S., in?t Veld, P. J., and Schunk, P. (2010). Mesoscale hydrodynamics via stochastic rotation dynamics: Comparison with lennard-jones fluid. *The Journal of chemical physics*, 132(17):174106.

Philipp, A. and Lauterborn, W. (1998). Cavitation erosion by single laser-produced bubbles. *Journal of Fluid Mechanics*, 361:75–116.

Pineda, S., Marongiu, J.-C., Aubert, S., and Lance, M. (2019). Simulation of a gas bubble compression in water near a wall using the sph-ale method. *Computers & Fluids*, 179:459–475.

Plesset, M. S. (1949). The dynamics of cavitation bubbles. *Journal of applied mechanics*, 16:277–282.

Plesset, M. S. and Chapman, R. B. (1971). Collapse of an initially spherical vapour cavity in the neighbourhood of a solid boundary. *Journal of Fluid Mechanics*, 47(2):283–290.

Plesset, M. S. and Prosperetti, A. (1977). Bubble dynamics and cavitation. *Annual review of fluid mechanics*, 9(1):145–185.

Plimpton, S. (1993). Fast parallel algorithms for short-range molecular dynamics. Technical report, Sandia National Labs., Albuquerque, NM (United States).

Plimpton, S., Pollock, R., and Stevens, M. (1997). Particle-mesh ewald and rrespa for parallel molecular dynamics simulations. In *PPSC*. Citeseer.

Plimpton, S. J. (2014). Modifying & extending lammps. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Pruppacher, H. R. and Klett, J. D. (1980). Microphysics of clouds and precipitation. *Nature*, 284(5751):88–88.

Qiao, G., Alexiadis, A., and Ding, Y. (2017a). Simulation study of anomalous thermal properties of molten nitrate salt. *Powder technology*, 314:660–664.

Qiao, G., Lasfargues, M., Alexiadis, A., and Ding, Y. (2017b). Simulation and experimental study of the specific heat capacity of molten salt based nanofluids. *Applied Thermal Engineering*, 111:1517–1522.

Quirk, J. J. and Karni, S. (1996). On the dynamics of a shock–bubble interaction. *Journal of Fluid Mechanics*, 318:129–163.

Rahmat, A., Barigou, M., and Alexiadis, A. (2019a). Deformation and rupture of compound cells under shear: A discrete multiphysics study. *Physics of Fluids*, 31(5):051903.

Rahmat, A., Barigou, M., and Alexiadis, A. (2019b). Numerical simulation of dissolution of solid particles in fluid flow using the sph method. *International Journal of Numerical Methods for Heat & Fluid Flow*.

Rahmat, A., Meng, J., Emerson, D., Wu, C.-Y., Barigou, M., and Alexiadis, A. (2021). A practical approach for extracting mechanical properties of microcapsules using a hybrid numerical model. *Microfluidics and Nanofluidics*, 25(1):1–17.

Rahmat, A. and Yildiz, M. (2018). A multiphase isph method for simulation of droplet coalescence and electro-coalescence. *International Journal of Multiphase Flow*, 105:32–44.

Raman, C. (1923). A theory of the viscosity of liquids. *Nature*, 111(2790):532–533.

Rapaport, D. C. (2004). *The art of molecular dynamics simulation*. Cambridge university press.

Rathakrishnan, E. (2013). *Theoretical aerodynamics*. John Wiley & Sons.

Rayleigh, L. (1917). Viii. on the pressure developed in a liquid during the collapse of a spherical cavity. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 34(200):94–98.

Rice, M. H. and Walsh, J. M. (1957). Equation of state of water to 250 kilobars. *The Journal of Chemical Physics*, 26(4):824–830.

Rood, E. (1991). Mechanisms of cavitation inception.

Ruiz-Riancho, I. N., Alexiadis, A., Zhang, Z., and Hernandez, A. G. (2021). A discrete multiphysics model to simulate fluid structure interaction and breakage of capsules filled with liquid under coaxial load. *Processes*, 9(2):354.

Sahputra, I. H., Alexiadis, A., and Adams, M. J. (2018). Temperature dependence of the young?s modulus of polymers calculated using a hybrid molecular mechanics–molecular dynamics method. *Journal of Physics: Condensed Matter*, 30(35):355901.

Sahputra, I. H., Alexiadis, A., and Adams, M. J. (2019). Effects of moisture on the mechanical properties of microcrystalline cellulose and the mobility of the water molecules as studied by the hybrid molecular mechanics–molecular dynamics simulation method. *Journal of Polymer Science Part B: Polymer Physics*, 57(8):454–464.

Sahputra, I. H., Alexiadis, A., and Adams, M. J. (2020). A coarse grained model for viscoelastic solids in discrete multiphysics simulations. *ChemEngineering*, 4(2):30.

Sameen, A. and Govindarajan, R. (2007). The effect of wall heating on instability of channel flow. *Journal of Fluid Mechanics*, 577:417–442.

Schütt, M., Stamatopoulos, K., Simmons, M., Batchelor, H., and Alexiadis, A. (2020). Modelling and simulation of the hydrodynamics and mixing profiles in the human proximal colon using discrete multiphysics. *Computers in Biology and Medicine*, page 103819.

Sear, R. P. (2014). Quantitative studies of crystal nucleation at constant supersaturation: experimental data and models. *CrystEngComm*, 16(29):6506–6522.

Seeton, C. J. (2006). Viscosity–temperature correlation for liquids. *Tribology letters*, 22(1):67–78.

Shadloo, M., Zainali, A., and Yildiz, M. (2013). Simulation of single mode rayleigh–taylor instability by sph method. *Computational Mechanics*, 51(5):699–715.

Shadloo, M. S., Oger, G., and Le Touzé, D. (2016). Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: Motivations, current state, and challenges. *Computers & Fluids*, 136:11–34.

Shadloo, M. S. and Yildiz, M. (2011). Numerical modeling of kelvin–helmholtz instability using smoothed particle hydrodynamics. *International Journal for Numerical Methods in Engineering*, 87(10):988–1006.

Shao, S. and Lo, E. Y. (2003). Incompressible sph method for simulating newtonian and non-newtonian flows with a free surface. *Advances in water resources*, 26(7):787–800.

Shin, Y., Lee, M., Lam, K., and Yeo, K. (1998). Modeling mitigation effects of watershield on shock waves. *Shock and Vibration*, 5(4):225–234.

Silling, S. A., Epton, M., Weckner, O., Xu, J., and Askari, E. (2007). Peridynamic states and constitutive modeling. *Journal of Elasticity*, 88(2):151–184.

Singraber, A., Behler, J., and Dellago, C. (2019). Library-based lammps implementation of high-dimensional neural network potentials. *Journal of chemical theory and computation*, 15(3):1827–1840.

Sirotkin, F. V. and Yoh, J. J. (2013). A smoothed particle hydrodynamics method with approximate riemann solvers for simulation of strong explosions. *Computers & Fluids*, 88:418–429.

Smith, R. C. and Marsh, J. S. (1974). Diffraction patterns of simple apertures. *JOSA*, 64(6):798–803.

Springel, V. (2010). Smoothed particle hydrodynamics in astrophysics. *Annual Review of Astronomy and Astrophysics*, 48:391–430.

Stanciu, I. (2012). A new viscosity-temperature relationship for vegetable oil. *Journal of Petroleum Technology and Alternative Fuels*, 3(2):19–23.

Storey, B. D. and Szeri, A. J. (2000). Water vapour, sonoluminescence and sonochemistry. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 456(1999):1685–1709.

Stukowski, A. (2009). Visualization and analysis of atomistic simulation data with ovito–the open visualization tool. *Modelling and Simulation in Materials Science and Engineering*, 18(1):015012.

Supponen, O., Obreschkow, D., Kobel, P., and Farhat, M. (2017). Luminescence from cavitation bubbles deformed in uniform pressure gradients. *Physical Review E*, 96(3):033114.

Supponen, O., Obreschkow, D., Tinguely, M., Kobel, P., Dorsaz, N., and Farhat, M. (2016). Scaling laws for jets of single cavitation bubbles. *Journal of Fluid Mechanics*, 802:263–293.

Swegle, J. and Attaway, S. (1995). On the feasibility of using smoothed particle hydrodynamics for underwater explosion calculations. *Computational Mechanics*, 17(3):151–168.

Swegle, J. et al. (1992). Report at sandia national laboratories.

Szeri, A. J., Storey, B. D., Pearson, A., and Blake, J. R. (2003). Heat and mass transfer during the violent collapse of nonspherical bubbles. *Physics of Fluids*, 15(9):2576–2586.

Tavarez, F. A. and Plesha, M. E. (2007). Discrete element method for modelling solid and particulate materials. *International journal for numerical methods in engineering*, 70(4):379–404.

Tinguely, M. (2013). The effect of pressure gradient on the collapse of cavitation bubbles in normal and reduced gravity. Technical report, EPFL.

TOMITA, Y. and SHIMA, A. (1977). On the behavior of a spherical bubble and the impulse pressure in a viscous compressible liquid. *Bulletin of JSME*, 20(149):1453–1460.

Tomita, Y. and Shima, A. (1986). Mechanisms of impulsive pressure generation and damage pit formation by bubble collapse. *Journal of Fluid Mechanics*, 169:535–564.

Turangan, C., Ball, G., Jamaluddin, A., and Leighton, T. (2017). Numerical studies of cavitation erosion on an elastic–plastic material caused by shock-induced bubble collapse. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2205):20170315.

Ulrich, C. and Rung, T. (2010). Sph modelling of water/soil-suspension flows. In *5th International SPHERIC Workshop. BD Rogers. Manchester, UK, 5th Inernational SPHERC Workshop Local Organising Committee*, pages 61–68.

Unknown (2020). Lammps developer guide. https://lammps.sandia.gov/doc/Developer.pdf.

Verlet, L. (1967). Computer" experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical review*, 159(1):98.

Vignjevic, R. and Campbell, J. (2009). Review of development of the smooth particle hydrodynamics (sph) method. In *Predictive modeling of dynamic processes*, pages 367–396. Springer.

Violeau, D. and Rogers, B. D. (2016). Smoothed particle hydrodynamics (sph) for free-surface flows: past, present and future. *Journal of Hydraulic Research*, 54(1):1–26.

Vogel, A., Busch, S., and Parlitz, U. (1996). Shock wave emission and cavitation bubble generation by picosecond and nanosecond optical breakdown in water. *The Journal of the Acoustical Society of America*, 100(1):148–165.

Vogel, A., Lauterborn, W., and Timm, R. (1989). Optical and acoustic investigations of the dynamics of laser-produced cavitation bubbles near a solid boundary. *Journal of Fluid Mechanics*, 206:299–338.

Walmsley, A., Laird, W., and Williams, A. (1988). Dental plaque removal by cavitational activity during ultrasonic scaling. *Journal of clinical periodontology*, 15(9):539–543.

Wang, X., Yang, D., Wu, J., and Luo, X. (2015). Interaction of a weak shock wave with a discontinuous heavy-gas cylinder. *Physics of Fluids*, 27(6):064104.

Weber, P. W., Howle, L. E., and Murray, M. M. (2010). Lift, drag, and cavitation onset on rudders with leading-edge tubercles. *Marine technology*, 47(1):27–36.

Ye, T., Pan, D., Huang, C., and Liu, M. (2019). Smoothed particle hydrodynamics (sph) for complex fluid flows: Recent developments in methodology and applications. *Physics of Fluids*, 31(1):011301.

Young, F. R. (1999). *Cavitation*. World Scientific.

Yu, P.-W., Ceccio, S. L., and Tryggvason, G. (1995). The collapse of a cavitation bubble in shear flows?a numerical study. *Physics of Fluids*, 7(11):2608–2616.

Zhang, S., Duncan, J. H., and Chahine, G. L. (1993). The final stage of the collapse of a cavitation bubble near a rigid wall. *Journal of Fluid Mechanics*, 257:147–181.

Zhu, Y., Fox, P. J., and Morris, J. P. (1999). A pore-scale numerical model for flow through porous media. *International journal for numerical and analytical methods in geomechanics*, 23(9):881–904.

Zienkiewicz, O. C., Taylor, R. L., Taylor, R. L., and Taylor, R. L. (2000). *The finite element method: solid mechanics*, volume 2. Butterworth-heinemann.