Université de Liège

MATH0471-A-A : Multiphysics integrated computational project

Smoothed-particle hydrodynamics: microfluidics applications

KOLLASCH Killian s202793
SANTORO Luca s201807

Professor: C. Geuzaine
Professor: R. BOMAN

Academic year : 2023-2024

Contents

List of Figures

List of Tables

I Introduction

Smoothed Particle Hydrodynamics (SPH) is a powerful computational method used to simulate fluid dynamics and other continuum mechanics problems. In SPH, the fluid domain is discretized into particles, each carrying properties such as mass, density, and velocity. These particles interact with each other through a smoothing kernel function, which computes weighted contributions from neighbouring particles.

It offers several advantages, including its meshfree nature, ability to handle large deformations, and suitability for complex geometries. It finds applications in various fields, from astrophysics and oceanography to engineering simulations like fluid flow and impact dynamics. Understanding the term "Smoothed Particle Hydrodynamics" entails dissecting its components:

- **Particle**: At the core of SPH lies the concept of particles, serving as the primary entities within the system. Interactions among these particles define the behaviour of the fluid or matter being simulated.
- **Smoothed**: This refers to the smoothing or weighting function employed to account for the influence of neighbouring particles on a given one, ensuring the continuity and coherence of the simulation.
- **Hydrodynamics**: SPH finds its most natural application in the realm of hydrodynamics, where its particle-based approach proves particularly effective in simulating fluid flow and related phenomena.

I.1 Motivation

In the frame of our Multiphysics integrated computational project, it has been proposed to improve the SPH method implemented in the thesis of Louis Goffin [4] by specialize it for microfluidic applications.

I.2 Basics of the method

The SPH method is said to be Lagrangian, meshfree and particle-based. All these three concepts are presented below:

Lagrangian Grid

A Lagrangian grid is a grid fixed on the material. This means that as the object under study deforms, the grid also deforms accordingly. Refer to Fig.1 for a visual representation of a Lagrangian grid. This technique finds extensive use in the finite element method. However, when the material experiences significant deformation, the grid itself becomes highly distorted. This distortion compromises the accuracy of calculations. To mitigate this, it becomes necessary to remesh when the grid distortion exceeds a certain threshold. However, this remeshing process increases the computational time required for the analysis.

Meshfree

SPH is considered mesh free because it does not require a mesh for field variable interpolation. However, SPH is not strictly mesh free; initialization typically involves arranging particles using a mesh, and mesh data may be required when searching for neighbouring particles.

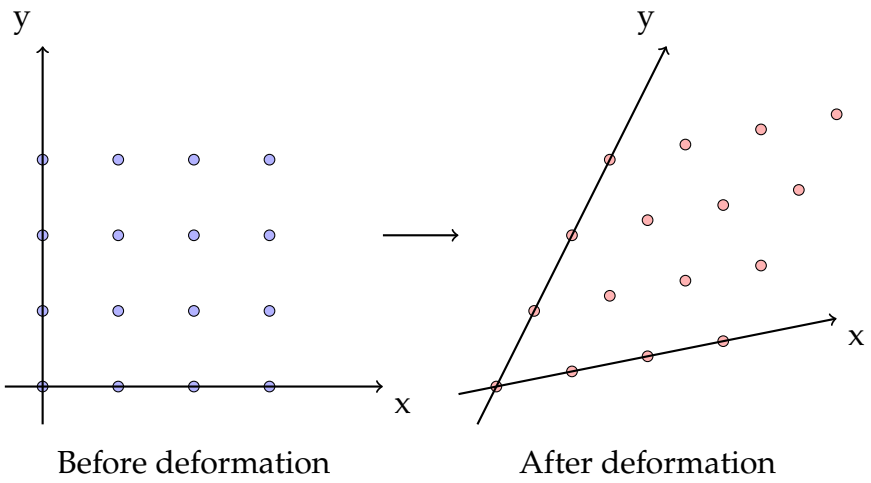


Figure 1: Lagrangian grid.

Particle-based

The matter or fluid is represented by a collection of particles. These particles interact with each other based on their distances, forming the foundation of the method's computational framework.

- **Particle**: At the core of SPH lies the concept of particles, serving as the primary entities within the system. Interactions among these particles define the behaviour of the fluid or matter being simulated.
- **Smoothed**: This refers to the smoothing or weighting function employed to account for the influence of neighbouring particles on a given one, ensuring the continuity and coherence of the simulation.
- **Hydrodynamics**: SPH finds its most natural application in the realm of hydrodynamics, where its particle-based approach proves particularly effective in simulating fluid flow and related phenomena.

Understanding these fundamental concepts lays a solid foundation for developing deeper understandings of SPH simulations and their applications in various fields of science and engineering.

II SPH algorithm

II.1 Integral representation of a function and its derivatives

The main idea behind SPH mesh-free method is to use the integral representation of a function as follows:

$$f(\boldsymbol{x}) = \int_{\Omega} f(\boldsymbol{x'})\delta(\boldsymbol{x} - \boldsymbol{x'})d\boldsymbol{x'}, \tag{II.1}$$

where Ω is the domain of integration and δ is the Dirac function defined as follows:

$$\delta(\boldsymbol{x} - \boldsymbol{x'}) = \begin{cases} \infty & \text{if } \boldsymbol{x} = \boldsymbol{x'}, \\ 0 & \text{if } \boldsymbol{x} \neq \boldsymbol{x'}, \end{cases} \tag{II.2}$$

This representation holds true for any values of $\boldsymbol{x'}$, yet it cannot be directly applied. The reason being, the entirety of the physical domain contributes to the value of any function at any point within the space. To address this limitation, the Dirac function is approximated by a normalized function, referred to as the kernel function $W(\boldsymbol{x} - \boldsymbol{x'}, h)$ (discussed in Sec.II.2). This function depends on the distance between the point where the value is to be computed and a smoothing length denoted as h.

The smoothing length signifies the range at which particles influence each other and remains constant throughout our Multiphysics integrated computational project.

Consequently, a function f and its derivative assessed at a specific point $\boldsymbol{x}$ are approximated using a kernel function in the following manner:

$$f(\boldsymbol{x}) \approx \int_{\Omega} f(\boldsymbol{x'})W(\boldsymbol{x} - \boldsymbol{x'}, h)d\boldsymbol{x'}, \tag{II.3}$$

$$\nabla.f(\boldsymbol{x}) \approx -\int_{\Omega} f(\boldsymbol{x'}).\nabla W(\boldsymbol{x} - \boldsymbol{x'}, h)d\boldsymbol{x'}. \tag{II.4}$$

II.2 Smoothing function

In order to respect the integral representation, the associated trial function (here the smoothing function) has to be defined on a compact support such that this function vanishes at the boundaries. The smoothing function is also called *kernel function*. Since the kernel function has to be defined on a compact support, it implies that the support domain $2\kappa h$ is contained in the problem domain, ensuring the validity of both Eq.(II.3) and Eq.(II.4). Different kernels may be used.

II.2.1 Cubic spline kernel

The kernel used in almost all our simulations is the *cubic spline* which is defined as follows:

$$W(r) = \begin{cases} \alpha\left(\frac{2}{3} - \left(\frac{r}{h}\right)^2 + 0.5\left(\frac{r}{h}\right)^3\right) & \text{if } 0 < \frac{r}{h} < 1, \\ \alpha\frac{1}{6}\left(2 - \frac{r}{h}\right)^3 & \text{if } 1 < \frac{r}{h} < 2, \\ 0 & \text{otherwise,} \end{cases} \tag{II.5}$$

where r is the distance between the particle i and its neighbour j and α is a constant of normalisation.

The scaling α value depends on both of the spatial dimension and the kernel type. For instance, using the cubic spline, the constant can take different values:

$$\alpha = \begin{cases} \frac{1}{h} & \text{(1D)} , \\ \frac{15}{7\pi h^2} & \text{(2D)} , \\ \frac{3}{2\pi h^3} & \text{(3D)} . \end{cases} \tag{II.6}$$

Indeed, this kernel was selected for its accuracy within the domain and its relatively minor error near the boundaries. The kernel and its derivative vanish at the upper boundary and also at $r = 0$. Both are illustrated in Fig.2.

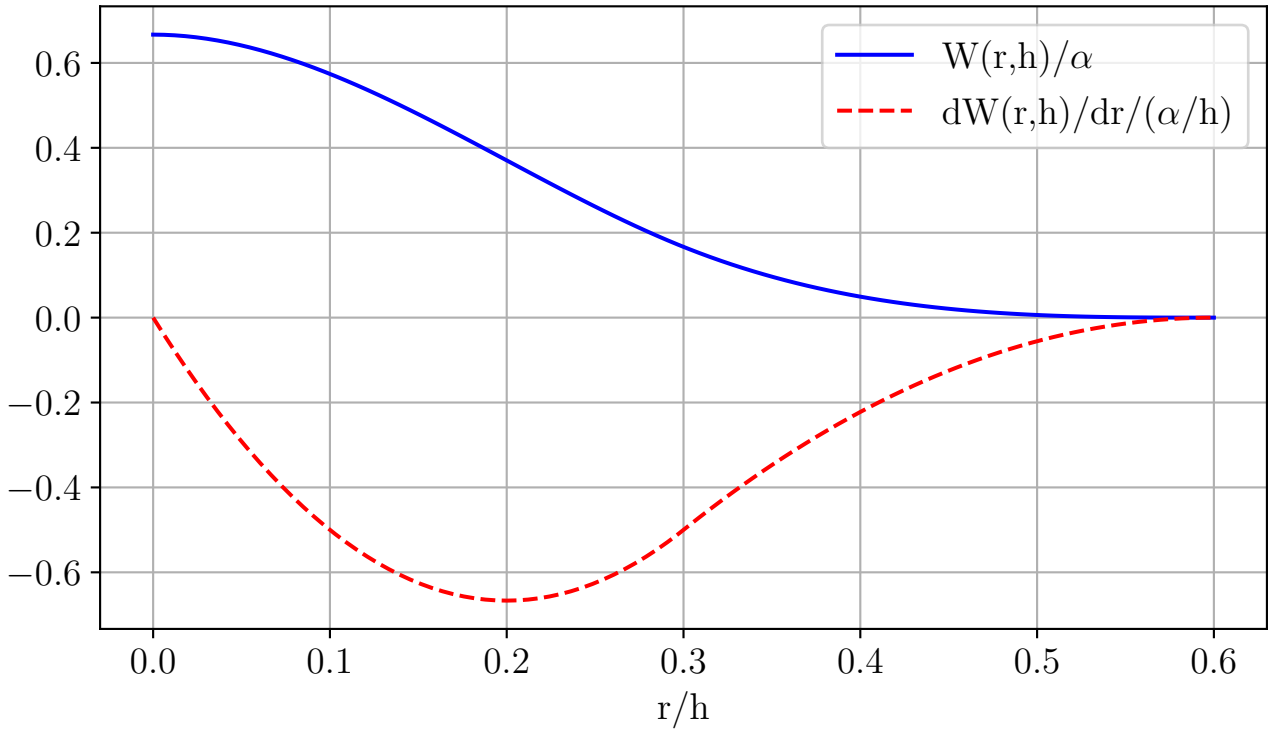


Figure 2: Cubic spline function and its derivative on the support domain $2\kappa h$ where (κ, h, s) = (2, 1.2s, 0.25).

It can be observed that the smoothing length h, within α, is raised to a power corresponding to the increasing dimensional domain. This adjustment is essential to maintain the independence of the kernel function from the selected smoothing length h (and thus spacing s). This observation applies to any kernel used in further sections (Sec.IV.1 and Sec.IV.2). To facilitate the use of the linked list algorithm for determining the neighbouring support of all particles (discussed in Sec.II.5), the smoothing length h must remain constant.

II.3 Spatial discretization

The physical domain undergoes discretization through the use of material points or particles. These particles possess constant mass and are propelled by the local flow velocity at each

time-step while carrying the fluid's physical properties. Consequently, Eq.II.3 and Eq.II.4 are discretized in the following manner:

$$f(\boldsymbol{x_i}) = \sum_{j=1}^{N} \frac{m_j}{\rho_j} f(\boldsymbol{x_j}) W_{ij}, \tag{II.7}$$

$$\nabla . f(\boldsymbol{x}) = -\sum_{j=1}^{N} \frac{m_j}{\rho_j} f(\boldsymbol{x_j}) . \nabla W_{ij}, \tag{II.8}$$

where i denotes the particle undergoing discretization of the function, j denotes all particles within the support domain of i (including particle i itself). m_j and ρ_j represent respectively the mass and density of the particle j, and $W_{ij} = W(\boldsymbol{x_i} - \boldsymbol{x_j}, h)$.

The impact of both mass and density of each particle is crucial within the SPH algorithm. Particularly, the mass of each particle is determined prior to simulation initiation via the total volume occupied by all particles. This necessitates the prior knowledge of this total volume, achieved by establishing a set of regularly spaced particles in a parallelepiped rectangle configuration, such that the total volume aligns with the dimensions of this rectangle as depicted in Fig.3.

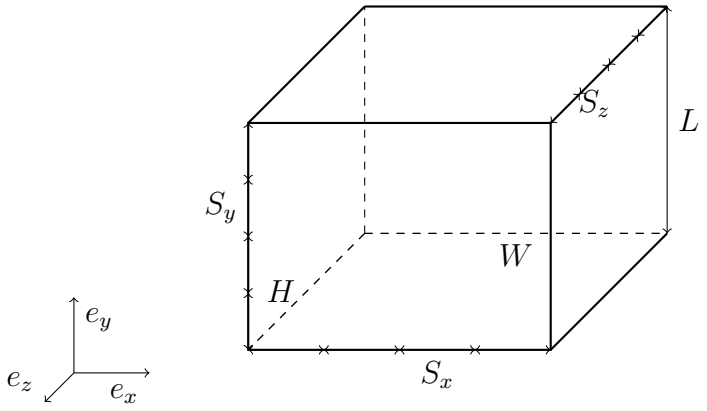


Figure 3: Mass particles initialisation.

In a fixed Cartesian system $\boldsymbol{e_x}, \boldsymbol{e_y}, \boldsymbol{e_z}$, where particles are evenly spaced, the following relationships hold:

$$V_{\text{tot}} = W \times L \times H = N_x S_x \times N_y S_y \times N_z S_z = \frac{M_{\text{tot}}}{\rho} = \frac{\sum_{i=0}^{N} m_i}{\rho}. \tag{II.9}$$

Here, W, L, and H represent the length in $\boldsymbol{e_x}$, $\boldsymbol{e_y}$ and $\boldsymbol{e_z}$ direction of the rectangle. $N_{x,y,z}$ and $S_{x,y,z}$ denote the number and spacing of particles in the specified direction.

Assuming uniformity among all particles within a single rectangle, the mass of an individual particle is determined by:

$$m_i = \rho_i \times S_x \, S_y \, S_z, \tag{II.10}$$

for 3D simulations. For 2D simulation, there is no y component and thus the mass m_i is given by

$$m_i = \rho_i \times S_x \, S_z. \tag{II.11}$$

Consequently, each particle's volume equals the total volume of the rectangle divided by the number of particles contained within it.

Furthermore, density is initially determined based on the selected state equation (as discussed in Sec.II.8).

II.4 Time discretization

Time discretization stands out as a crucial aspect of the SPH algorithm. While other elements of the method mainly impact the accuracy of particle variables, inaccuracies in time discretization could potentially disrupt the entire simulation. The thesis of Louis [4] employed two finite difference schemes : the Euler explicit and the second-order Runge-Kutta methods.

Euler explicit

The Euler explicit scheme emerges as an initial choice due to its simplicity and relatively low computational cost for function integration. It is defined as:

$$y_{i+1} = y_i + \Delta t \frac{dy}{dt}(t_i, y_i). \tag{II.12}$$

Runge-Kutta

The second-order Runge-Kutta method (RK22) necessitates two evaluations of the given function to obtain the desired value:

$$\begin{cases} y_{i+1} = y_i + \Delta t \left[(1-\theta)k_1 + \theta\, k_2\right], \\ k_1 = \dfrac{dy}{dt}(t_i, y_i), \\ k_2 = \dfrac{dy}{dt}(x_i + \dfrac{1}{2\theta}\Delta t, y_i + \dfrac{1}{2\theta}\Delta t\, k_1). \end{cases} \tag{II.13}$$

The RK22 algorithm essentially comprises a weighted sum of two Euler explicit schemes computed at two distinct locations. The parameter θ determines the weighting of $k_{1,2}$, representing the slope of the function at the beginning and end of the step. A value of $\theta = 1/2$ corresponds to the midpoint method.

II.5 Neighbours search

To assess the neighbours of each particle may be tedious while trying to get efficient CPU performances.

Naive algorithm

The first method, so-called *naive algorithm*, is the simplest one and relatively straightforward. Indeed, given a particle i, the algorithm consists in iterating over all other particles and keep only particles which are on the support domain (thus at a distance smaller than κh):

Obviously, this method is not optimal at all since:

$$CPU_{\text{naive algo}} \sim \mathcal{O}(n^2), \tag{II.14}$$

where n is a multiple computational operation. This algorithm will thus not be used later.

Algorithm 1 Naive algorithm

```
for i = 1, 2, ..., N do
    for j = 0, ..., N do
        Compute distance r_ij between particle i and j
        if r_ij^2 ≤ (κh)^2 then
            Add particle j to neighbours of i
        end if
    end for
end for
```

Linked-list algorithm

Another more efficient method is the so-called *linked-list algorithm*. The primary concept of this approach, as outlined in L. Goffin's thesis, involves enclosing all particles within cells (subdomains) and exclusively examining the neighbours of a given particle in adjacent cells. These cells have dimensions equal to or greater than the neighbouring radius κh. Currently, κ is set to 1 and h is set to 1.2s. The choice of κ will be either 1, 2, or 3 depending on the kernel selected for computation. Also, the *linked-list* algorithm requires (for efficient performances) that the smoothing length h is kept as constant between each time step.], its efficiency is estimated to:

$$CPU_{\text{linked list}} \sim \mathcal{O}(n), \tag{II.15}$$

if the number of particles per cell is low enough.

One first needs to determine the total number of cells in each direction x, y, z of the domain (assuming a 3D rectangular domain). Given $L[0], L[1], L[2]$ (denoted L_0, L_1, L_2 lately) being respectively the lengths of the domain in the x, y, z direction, the number of cells in a prescribed direction is given by:

$$N_{x,y,z} = \left\lfloor \frac{L_{0,1,2}}{\kappa h} \right\rfloor. \tag{II.16}$$

For instance, if L_1 = 1.0 and s = 0.25:

$$N_x = \left\lfloor \frac{1}{2 \times 1.2 \times 0.25} \right\rfloor = 6, \tag{II.17}$$

hence, each cell will be referenced by three indexes (i, j, k).

Once the total number of cells in each direction is set, one iterates over all particles in order to assign, for each particles, their associated cell (e.g. where they lye). The relationship between a position (x, y, z) of a particle and its associated cell is:

$$\text{index}_{i,j,k} = \frac{\text{position}_{x,y,z}}{L_{x,y,z}/N_{x,y,z}}, \tag{II.18}$$

where i, j, k are respectively associated to x, y, z.

But one needs to pay attention if an edging cell is encountered. This is done by the following coding lines:

It can be noticed that particles outside the domain are skipped such that they have no influence any more on the particle inside the domain.

Algorithm 2 Linked list (1/2) : cell sorting

for $n = 0, 1, \ldots, N$ **do**
  $i \leftarrow \frac{x}{Lx/Nx}$
  $j \leftarrow \frac{y}{Ly/Ny}$
  $k \leftarrow \frac{z}{Lz/Nz}$
  if $(i, j, k) < 0$ or $(i, j, k) > N_{x,y,z}$ **then**
    Skip particle n (outside the domain)
  end if
  if $(i, j, k) = N_{x,y,z}$ **then**
    reduce index (i,j or k) by 1 (particle at boundaries)
  end if
  Add particle n to corresponding cell
end for

Next, one can accurately iterate through neighbouring cells and then iterate through all their associated particles to identify the corresponding neighbours. Subsequently, when a particle is encountered in a neighbouring cell (or in the same cell as the particle i), their relative distance r_{ij} is calculated and compared to κh. If $r_{ij}^2 \leq (\kappa h)^2$, the particle j is added to the neighbour list of particle i:

Algorithm 3 Linked list (1/2) : find neighbours

for $k_{adjacent} = k - 1, k, k + 1$ **do**
  for $j_{adjacent} = j - 1, j, j + 1$ **do**
    for $i_{adjacent} = i - 1, i + 1$ **do**
      cell $\leftarrow$ cell_matrix$[i + j \cdot Nx + k \cdot Nx \cdot Ny]$
      for $idx = 0$ to $size\_cell - 1$ **do**
        $idx\_cell \leftarrow$ cell$[idx]$
        if $idx\_cell \neq n$ **then**
          Compute r_{ij}^2
          if $r_{ij}^2 \leq (\kappa h)^2$ **then**
            $neighbours[n] \leftarrow idx\_cell$
          end if
        end if
      end for
    end for
  end for
end for

Also, the maximum number of neighbours for each particle is set to 100 to avoid dynamic memory allocation. The *type* variable in the code simply indicates if a moving or fixed particle in encountered.

Comparison of the two methods

It is clear that the *linked-list* method is faster than the *naive* method. To determine the point at which the naive method becomes impractical, one can measure the average time taken by both algorithms to sort all the particles for a timestep for varying numbers of particles.

As shown in Fig. 4, the *linked-list* algorithm becomes significantly more advantageous when the number of particles exceeds several thousands. Also, the data relative to this figure have been obtained using the OpenMP version of the code. This comparison is made with the

linked_list_comparaison.json file, it creates a cube of one meter side and different spacing are tested. Table 1 groups every spacing s and their corresponding number of particles

s	0.05	0.025	0.02	0.01
Number of particle	9261	68 921	132 651	1 030 301

Table 1: Spacing and the corresponding number of particle used to compare the linked list algorithm and the naive algorithm

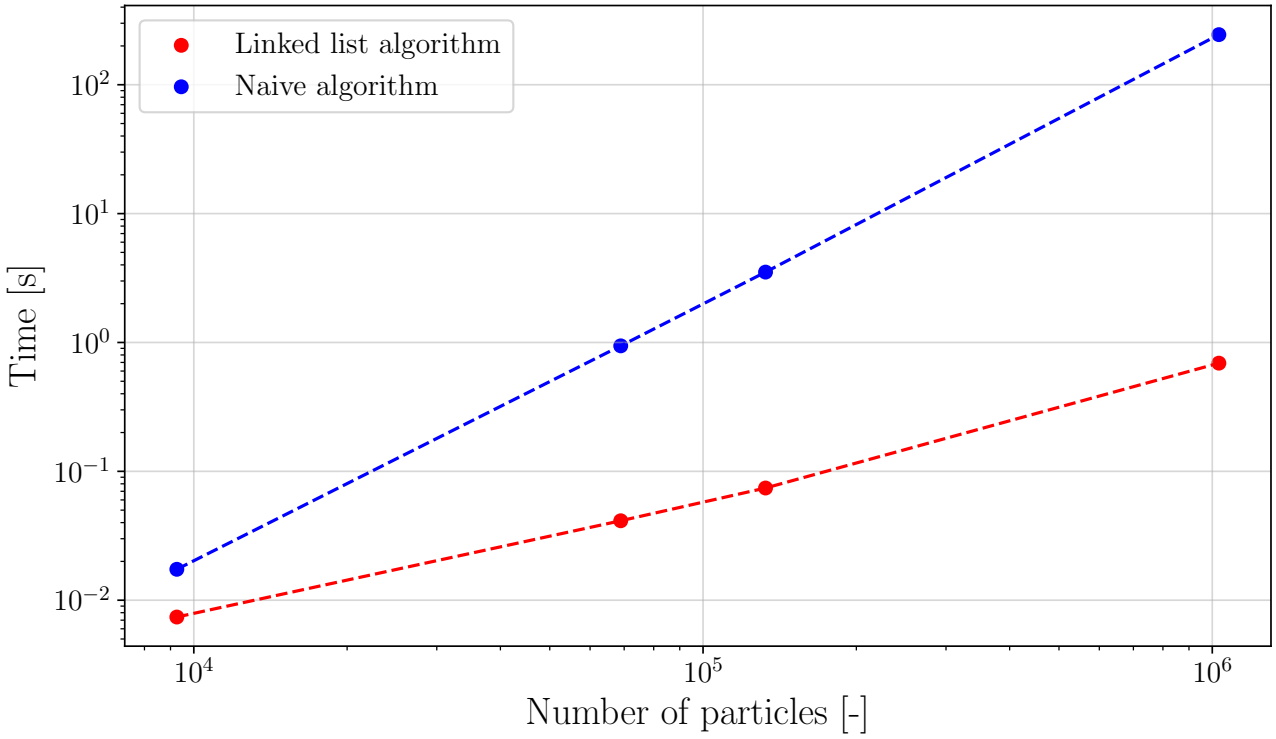


Figure 4: Comparison of the Linked-list and Naive sorting algorithm.

II.6 Navier-Stokes equations

The SPH method, being Lagrangian and mesh-free, uses governing equations in the Lagrangian frame. These governing equations are the conserving mass and momentum, expressed as:

$$\frac{D\rho}{Dt} = \boldsymbol{u}.\nabla\rho - \nabla.(\rho\boldsymbol{u}), \tag{II.19}$$

$$\frac{D\boldsymbol{u}}{Dt} = -\frac{\nabla p}{\rho} + \frac{\boldsymbol{F}}{\rho} = -\left(\nabla\left(\frac{p}{\rho}\right) + \frac{p}{\rho^2}\nabla\rho\right) + \frac{\boldsymbol{F}}{\rho}, \tag{II.20}$$

where ρ is the density, D/Dt the material derivative, $\boldsymbol{u}$ the velocity field, p the local static pressure and $\boldsymbol{F}$ the volume body forces. In this study, only body and friction forces are taken into account.

Using the spatial discretization introduced in Sec.II.3 allows us to rewrite these two equations in the following manner:

$$\frac{D\rho_i}{Dt} = \sum_{j=1}^{N} m_j \boldsymbol{u_{ij}} \cdot \boldsymbol{\nabla} W_{ij}, \tag{II.21}$$

$$\frac{D\boldsymbol{u_i}}{Dt} = -\sum_{j=1}^{N} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} + \Pi_{ij} \right) \boldsymbol{\nabla} W_{ij} + \boldsymbol{F}, \tag{II.22}$$

where $\boldsymbol{u}_{ij}$ is the relative velocity between particle i and j, Π_{ij} is the artificial viscosity introduced in the following.

Artificial viscosity

The friction forces will be considered in this work with an artificial viscosity. As explained in L.Goffin master's thesis.This term has two main purposes:

1. to create a viscosity term in order to reproduce the friction that occurs in a real fluid

2. to avoid numerical instabilities when the particles are moving away from each other. This artificial viscosity has been introduced by [Monaghan, 1992]. Its expression is:

$$\Pi_{ab} = \begin{cases} \dfrac{\alpha \bar{c_{ij}} \mu_{ij} + \beta\, \mu_{ij}^2}{\bar{\rho_{ij}}} & \text{for } \boldsymbol{u_{ij}} \cdot (\boldsymbol{x_i} - \boldsymbol{x_j}) < 0, \\ 0 & \text{for } \boldsymbol{u_{ij}} \cdot (\boldsymbol{x_i} - \boldsymbol{x_j}) \geq 0, \end{cases} \tag{II.23}$$

with $\bar{\rho_{ij}}$ and $\bar{c_{ij}}$ being the mean density and speed of sound between particle i and j. α and β are parameters given by the user , where β is usually taken as 0, and μ_{ij} has the following formula:

$$\mu_{ij} = \frac{h \boldsymbol{u_{ij}} \cdot (\boldsymbol{x_i} - \boldsymbol{x_j})}{(\boldsymbol{x_i} - \boldsymbol{x_j})^2 + \eta^2}, \tag{II.24}$$

with η^2 = 0.01 h^2. This viscosity term does not directly introduce the dynamic or kinematic viscosity. However, according to [3], it can be linked by the following relation:

$$\nu = \frac{c\, h\, \alpha}{8}. \tag{II.25}$$

This shows that the kinematic viscosity depends on h and thus on the spacing s. For a small spacing, and therefore a large number of particles, the viscosity term becomes less significant.

Volume body forces

The two body forces taken into account in this study are the gravity force and the surface tensions.

The gravity is straightforward to implement, such that:

$$\boldsymbol{F_{gravity}} = -g \boldsymbol{e_z}, \tag{II.26}$$

where g = 9.81 [m/s^2].

Nevertheless, the surface tensions are much harder to handle due to the fact that the outside boundaries of the fluid are no trivially known (discussed in Sec.IV).

II.7 Ghost particles

In order to extract physical variables from the simulations, *ghost particles* are introduced. These ghost particles have neighbours, but they are not detected by other fluid or fixed particles.

By doing so, one can apply onto them the *summation density* method which is nothing else than Eq.(II.7). This method ensures exact mass conservation [7] but offers less stability and accuracy if scenarios involve high-velocity free surface flows. Nevertheless, since *ghost particles* do not have any influence on the other particles (by definition), the *summation density* method can be applied to them without any problem.

II.8 State equation

The relation between the pressure is everything but trivial and has thus to be discussed. While lots of research have been performed to assess accurate relation between ρ and p, one has simply used two well known state equations, being the *ideal gas law* and *quasi incompressible fluid*. Also, as explained in Sec.II.3, the initialization of the density depends on these state equations.

Ideal gas law

The *ideal gas law* reads:

$$p = \frac{RT}{M}\left(\frac{\rho}{\rho_0} - 1\right), \tag{II.27}$$

where p [N/m^2] is the absolute pressure, ρ [kg/m^3] the density, R = 8.314 [J/(K mol)] the ideal gas constant, T [K] the absolute temperature (= 293.15 [K] and kept constant) and M [kg/mol] is the molar mass.

The key aspect of this law is that it implies a linear relationship between pressure and density, allowing density to increase significantly under high pressure conditions. This equation proves particularly valuable for examining highly compressible fluids like air. However, this research focuses on water, renowned for its quasi-incompressible nature. Therefore, alternative models need to be explored.

Another vital parameter in the SPH method is the speed of sound, denoted as c, for each particle. This value dictates the speed at which information propagates among particles, thus influencing the time integration process (as discussed in Sec.II.9). The speed of sound is determined as follows:

$$c_i = c_0, \tag{II.28}$$

where $i \in \{0, 1, ..., N\}$ is the particle index and c_0 a constant speed of sound.

Here is the evolution of the pressure and speed of sound:

Finally, the initialization of the density given a hydrostatic pressure for the *ideal gas law* reads:

$$\rho = \rho_0\left(1 + \frac{M}{RT}\rho_0 g h\right). \tag{II.29}$$

Quasi incompressible fluid

The *quasi compressible* equation of state used in this study is the one used by Monaghan [8] as follows:

$$p_i = B\left(\left(\frac{\rho_i}{\rho_0}\right)^{\gamma} - 1\right), \tag{II.30}$$

where B is a constant parameter which decides the maximum density fluctuation or compressibility, ρ_0 is the reference density, and γ is the heat capacity ration equal to 7 in this study.

B is obtained from:

$$B = \frac{c_0^2 \rho_0}{\gamma}, \tag{II.31}$$

in which c_0 is the speed of sound at reference density and is usually set to 10 times the maximum flow velocity in order to limit the compressibility of the flow.

$$c = c_0 \left(\frac{\rho}{\rho_0} \right)^{\gamma - 1}, \tag{II.32}$$

for a *quasi-incompressible* fluid.

Finally, the initialization of the density given a hydrostatic pressure reads:

$$\rho = \rho_0 \left(1 + \frac{1}{B} \rho_0 g h \right)^{1/\gamma}, \tag{II.33}$$

Here is the evolution of the pressure and speed of sound:

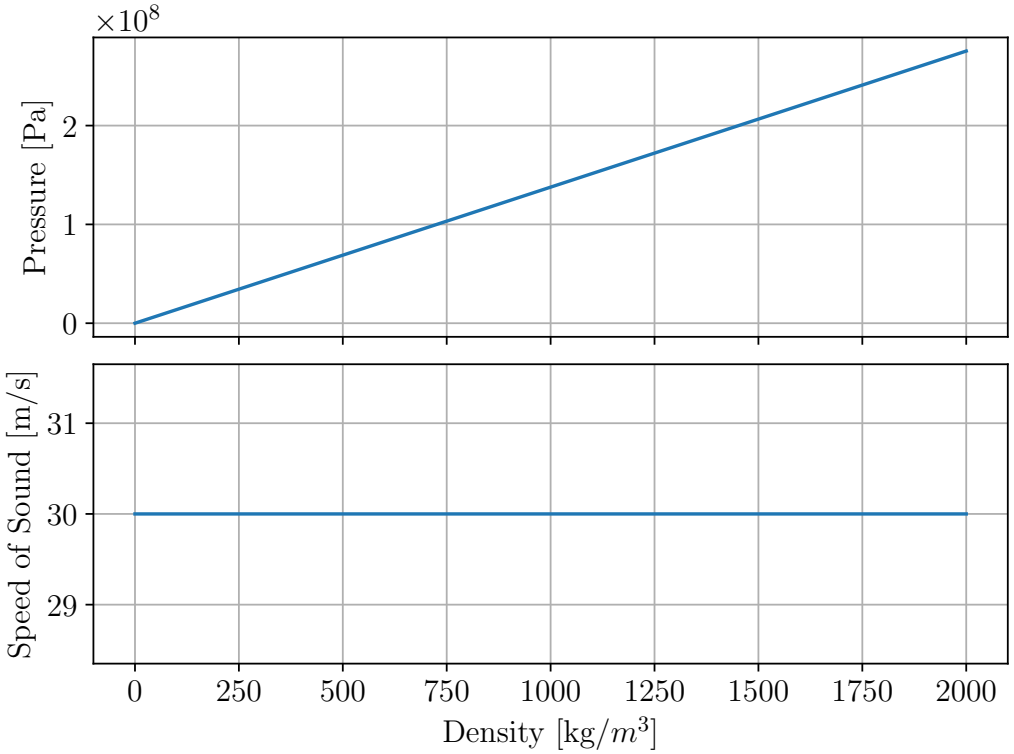


Figure 5: Ideal gas law with c_0 = 30 [m/s], ρ_0 = 1000 [kg/m^3], R = 8.314 [J/kg.mol], T = 298.15 [K], M = 18e-3 [kg/mol].

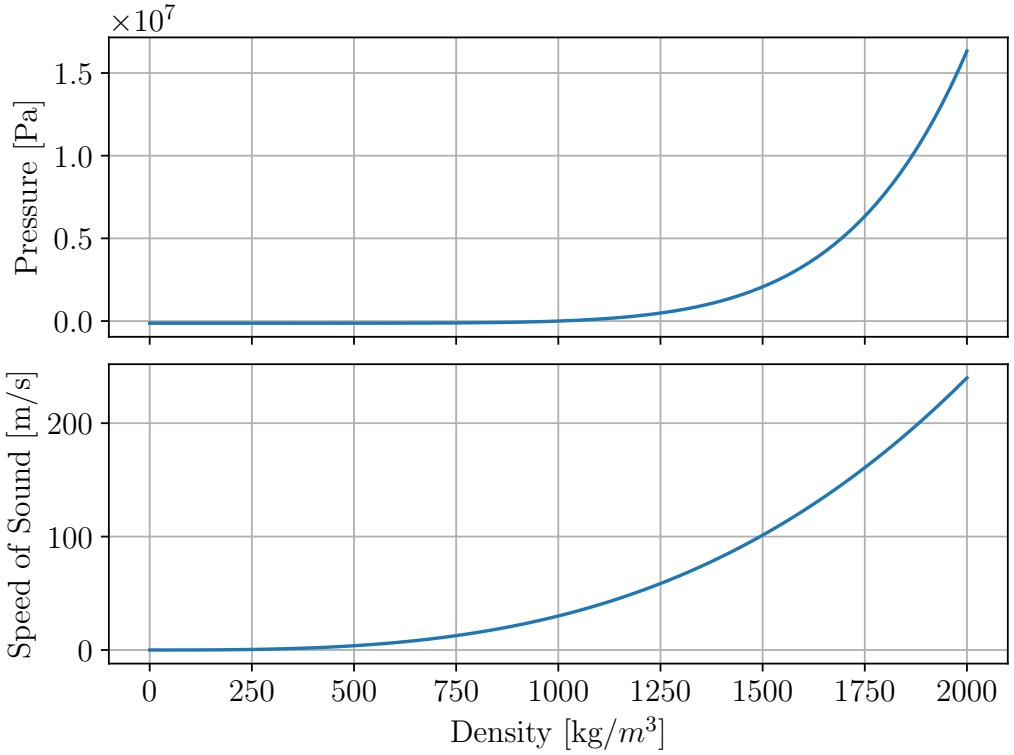


Figure 6: Quasi incompressible fluid with $c_0 = 30$, $\rho_0 = 1000$, $B = 128571.43$, $\gamma = 7$.

Determination of density

To derive the pressure field from the equation of state, it is necessary to initially compute the density of each particle. Density can be determined either through the discretized Eq.(II.19) or by utilizing the SPH representation of density from Eq.(II.7).

As discussed in Sec.II.7, the *summation density* method ensures exact mass conservation, whereas the *continuity density* method generally struggles to conserve mass [8]. Conversely, the *continuity density* method tends to offer greater stability and accuracy in scenarios involving high-velocity free surface flows and impacts, such as the Dam break problem. This advantage comes from the underestimation of the density on the free surfaces in the *summation density* method, coming from the truncated support domain of the kernel function.

In this study, the *continuity density* method is used to update the density and velocity of each particle.

II.9 Time integration

Given the discussions raised in Sec. II.3 and Sec.II.6, the time integration (using Euler explicit scheme for simplicity) is performed using the following formula:

$$\rho_{i+1} = \rho_i + \Delta t \frac{D\rho_i}{Dt}, \tag{II.34}$$

$$\boldsymbol{u}_{i+1} = \boldsymbol{u}_i + \Delta t \frac{D\boldsymbol{u}_i}{Dt}, \tag{II.35}$$

where Δt is the time increment between each step.

To ensure stability of the numerical scheme, the time step has to be carefully chosen smaller than a critical value. But one needs to determine this threshold value.

Body force condition

The first expression given is relative to the body forces:

$$\Delta t_f = \min_i \left\{ \sqrt{\frac{h_i}{|\boldsymbol{F_i}|}} \right\}, \tag{II.36}$$

with $i = 1, \ldots, N$, N being the total number of particles.

If other body forces will be added, $|\boldsymbol{F_i}|$ will be modified such that:

$$|\boldsymbol{F_i}| = \sqrt{\sum_a \left(F_i^{(a)}\right)^2}, \tag{II.37}$$

where the sum over a represents all the body forces applied to particle i.

Courant condition

The second expression is relative to the Courant condition as well as the viscous diffusion:

$$\Delta t_{cv} = \min_i \left\{ \frac{h}{c_i + 0.6(\alpha c_i + \beta \max_j \mu_{ij})} \right\}, \tag{II.38}$$

where the coefficients α and β are the same coefficients from Eq.(II.23) and c_i is the speed of sound at the particle i.

The final time step is obtained by:

$$\Delta t_{qi} = \min\left(0.4\Delta t_f,\, 0.25\Delta t_{cv}\right). \tag{II.39}$$

The coefficients 0.25 and 0.4 come from computer experiments [8].

II.10 Json file

For each simulation, a Json file containing all the simulation parameters is needed. These different parameters a classify under several groups. The Json file structure is explain here

- name_file: string used to name the output file
- omp:
  - chose_nb_of_threads: true or false saying if the user chose the number of thread used (if false, the maximum number available is used)
  - nb_of_threads: number of thread that the users wants to used
- simulation
  - theta: θ used for the RK22 scheme
  - s: spacing between particles
  - nstepT: number of step
  - nsave: number of iteration between two output
  - kappa: κ used for the Kernel used (always 2 here)
  - alpha: α used for artificial viscosity
  - beta: α used for artificial viscosity (always 0 here)

  - alpha_st: α_{st} used for the surface tension model
  - beta_adh: β_{adh} used for adhesion model
  - dimension: the dimension of the simulation (2 ou 3)
  - scheme_integration: Euler or RK22
  - comparison_algorithm: used to compare the linked list algorithm and the naive algorithm (true or false)
- following_part:
  - part: say if it follows properties of a particle (true or false)
  - min: say if it has to find the minimum value of the simulation (true or false)
  - max: say if it has to find the maximum value of the simulation (true or false)
  - particle: particle that it is followed
  - pressure: give the pressure (true or false)
  - rho: give the density (true or false)
  - position: vector that give the x, y or z position (true or false)
  - velocity: vector that give the x, y or z velocity (true or false)
- domain:
  - matrix_long: vector of vectors giving the length in the x, y, z direction for each rectangular parallelepiped generated
  - matrix_origin: vector of vectors giving the origin position for each rectangular parallelepiped generated
  - sphere
    - do: vector saying if a sphere is done for the corresponding element generated
    - radius: give the radius of the sphere wanted
  - vector_type: give the type the parallelepiped generated (0 fixed particles, 1 moving particles, 2 ghost particles)
  - L_d: give the x, y and z length of the domain
  - o_d: give the origin of the domain
- post_process:
  - do: create or not the ghost particle (true or false)
  - xyz_init: vector giving the initial position of the ghost particles
  - xyz_end: vector giving the final position of the ghost particles
- thermo:
  - rho_0: initial density
  - rho_moving: density of moving particle
  - rho_fixed: density of fixed particle
  - T: temperature

  - u_init: initial velocity of moving particles
  - c_0: initial sound speed
  - gamma: heat capacity ratio
  - M: molar mass
  - R: perfect gaz constant
  - sigma: surface tension
- forces
  - gravity: active or not gravity (true or false)
  - surface_tension_1: active or not the first method of surface tension (true or false)
  - surface_tension_2: active or not the second method of surface tension (true or false)
  - adhesion: active or not adhesion forces (true or false)
- condition
  - print_debug: active print in the code
  - state_equation: ideal gaz or quasi incompresible
  - initial_condition: hydrostatic or constant

II.11 Speed-up due to non OOP implementation

The primary drawback of Louis's SPH algorithm lies in the adoption of objects in his Fortran code, inevitably impacting the simulation's CPU time due to the frequent retrieval of information stored in these object variables, especially when employing OpenMP. Thus, the initial improvement involves developing a non-OOP implementation of the SPH algorithm.

The SPH method is well-suited for parallelisation because it requires multiple loops over particles. However, within a single loop, each particle operates independently of the others. As a result, it is possible to use the command before each loop over particles

```
1            #pragma omp parallel for.
```

For the comparison, the JSON file *3D_splash.json* has been used each time, but either the spacing s (and thus the number of particles) or the number of threads will vary to assess the speed-up induced by the parallelisation.

As can be seen on Fig.7, using 22 threads does not significantly increase the speed of the code for the number of particles used in this test. However, the difference may become more noticeable with a larger number of particles.

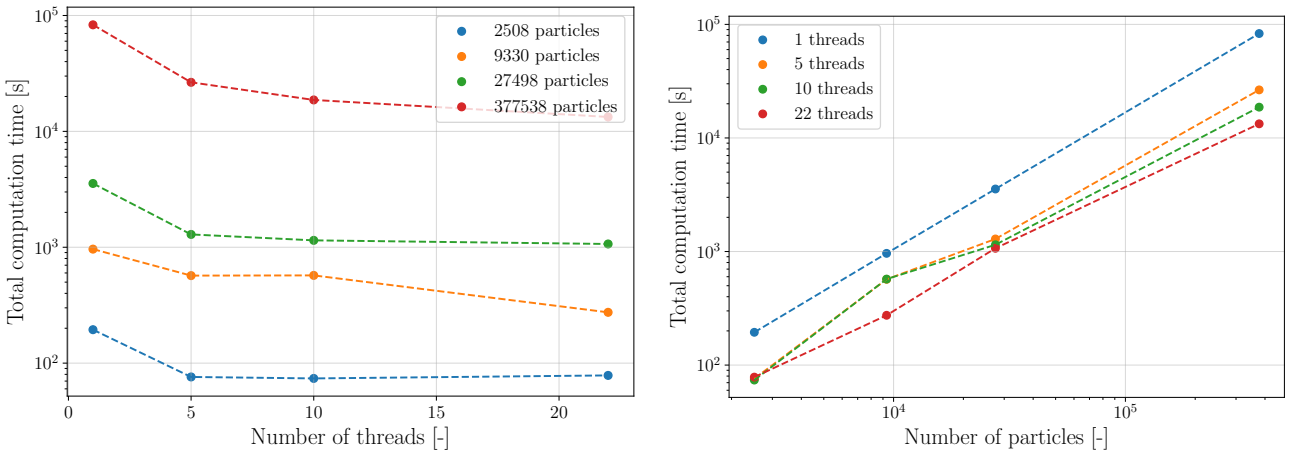


Figure 7: Time variation of the splash simulation depending on the number of threads and the number of particles.

III Classical applications

Before applying the code to microfluidic applications, it is essential to verify its accuracy using basic test cases. Three cases are studied: the first involves a falling cube where gravity causes the cube to fall with uniform acceleration; the second is a cube of water at rest in a tank, where the final result is expected to show a linear pressure gradient. The last test is a dam break, where the wavefront found numerically is compared to experimental results in a dimensionless form. The 2D implementation is also discussed, and a comparison is made between the Euler and RK22 integration schemes. For each simulation, the number of moving particles (Nb M.P.), the number of fluid particles, and the total number of particles (Nb Part) are also provided.

III.1 Falling cube

The first basic application is the falling cube. This test consists of a cube of fluid falling due to gravity and splashes in a box. The goal of this test is to verify if particles follow Newton second Law. From this equation, and by only considering gravity, one retrieves the parabolic motion of a particle falling and it is given by

$$z(t) = z_0 - \frac{1}{2} g\, t^2, \tag{III.1}$$

with z_0 the initial position and g the gravity 9.81 m/s^2. The initial geometry of this test is shown in Fig.8.

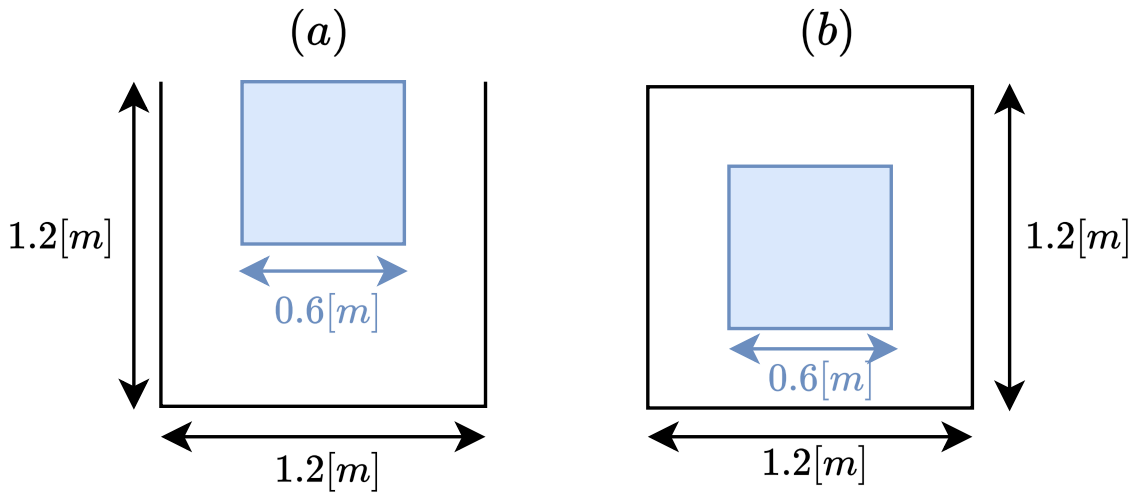


Figure 8: Initial geometry of the falling cube test. Representation of the front view (a) and top view (b).

All the characteristics of this problem are given in the json file *3D_splash.json*. In Table 2 bellow some important parameters are expressed:

Simulation time	Nb M.P.	Nb Part	s	α	c_0	Calculation time
0.99005	2197	9330	0.05	0.5	30	371

Table 2: Different parameters of the 3D cube falling.

Fig. 9 (left) shows the z position over time for a particle of the cube. During the first 0.3 seconds, the particle follows the theoretical curve. After this period, the particle reaches the

floor and ceases to move according the parabolic motion. In Figure 9 (right), the relative error remains small until 0.35 seconds, confirming that the particle is no longer undergoing uniformly accelerated motion as described by Eq. (**??**).(III.1).

On a general aspect of this test, when the cube reaches the bottom of the tank, particles start to be repelled and are expelled in different directions. Due to the symmetry of the problem, it is expected that the "splash" is symmetric. Fig. 10 illustrates different snapshots of the simulation over different time and shows the symmetry that was expected (f).

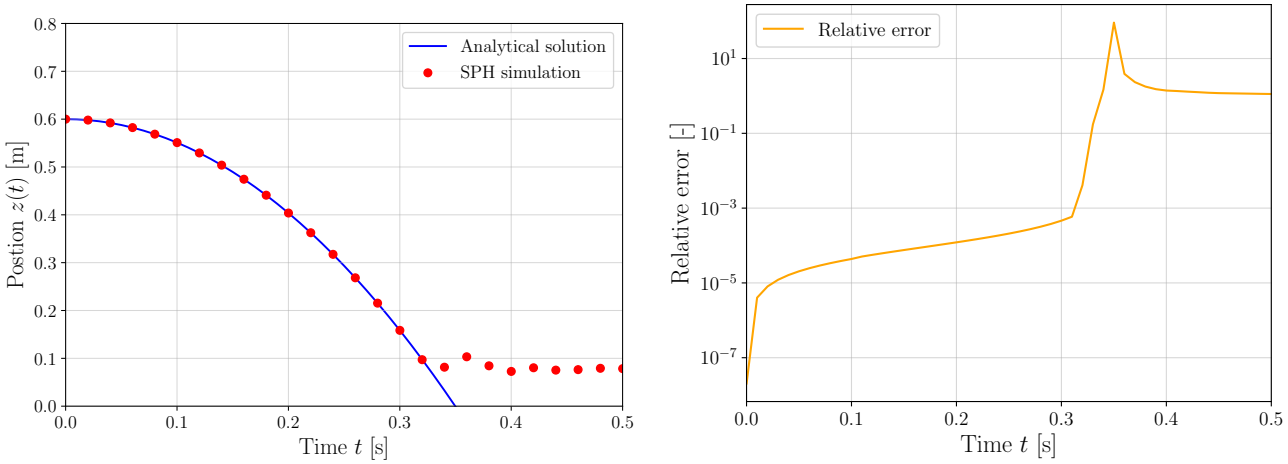


Figure 9: z position over time of a moving particle and comparison with the analytical solution (left). Relative error over time between the solution of the numerical simulation and the analytical solution (right) for the 3D simulation.

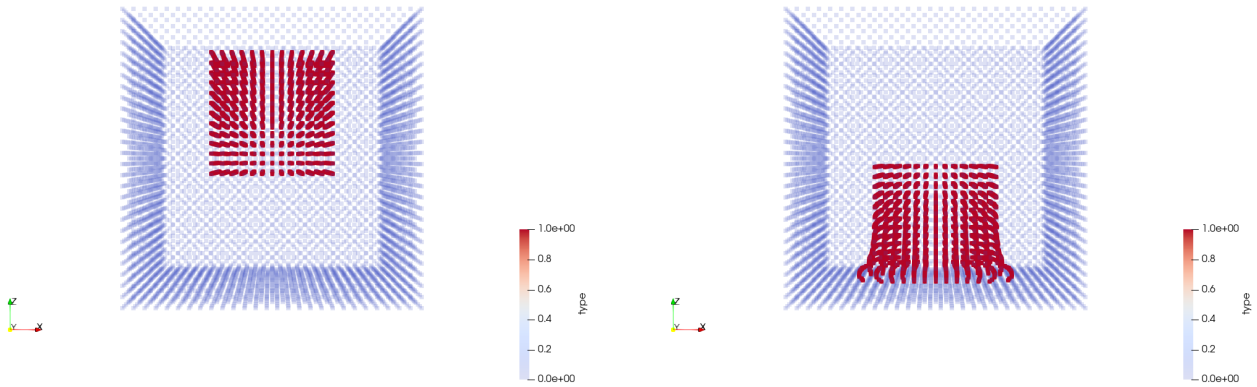
III.1.1 2D test

Thanks to the symmetry of the problem, a 2D simulation can be used to speed up the computation. By using the same geometry shown in Fig. 8, only the left half of the system is generated, resulting in fewer particles. Table 3 provides various parameters of the simulation, with the corresponding configuration file being *2D_splash.json*. Comparing the simulation times between the 2D and 3D cases reveals that the 2D simulation is approximately 14 times faster.

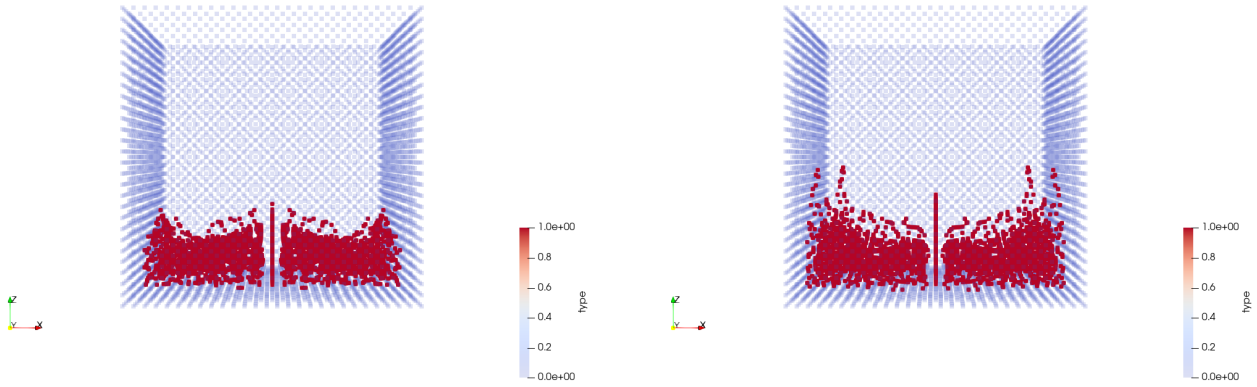
Simulation time [s]	Nb M.P.	Nb Part	s	α	c_O	Calculation time [s]
0.99005	169	473	0.05	0.5	30	26

Table 3: Different parameters of the 3D cube falling.

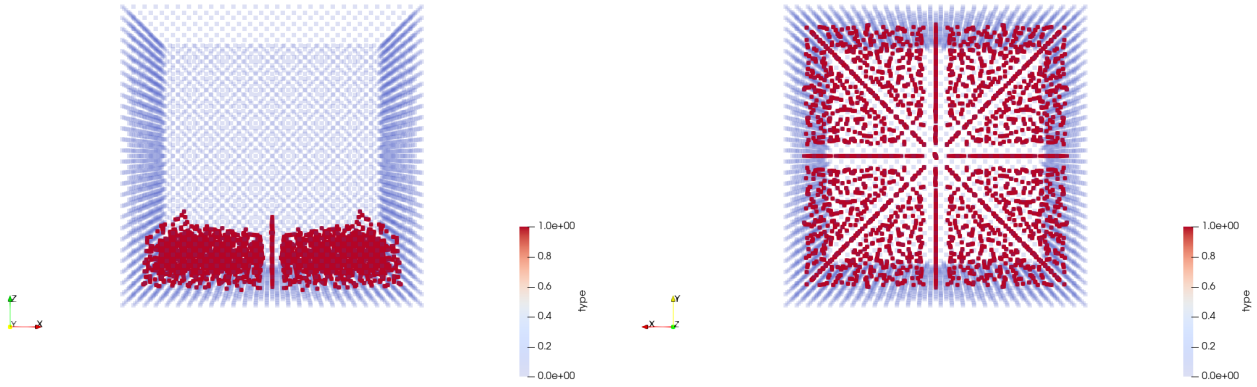
Figure 11 demonstrates that a particle in a 2D simulation follows the theoretical curve, validating the use of 2D simulations for modeling problems with certain symmetries. This approach enables faster simulations with the same particle spacing, or alternatively, the spacing can be reduced to generate more particles, leading to more accurate physical results.



(a) $t = 0.000050[s]$ (b) $t = 0.340050[s]$



(c) $t = 0.400050[s]$ (d) $t = 0.650050[s]$



(e) $t = 0.990050[s]$ (f) $t = 0.990050[s]$, upper view

Figure 10: Snapshots of the cube falling for different simulations time where red particles represent the fluid (moving particles) and the light blue particle are boundaries (fixed particles).

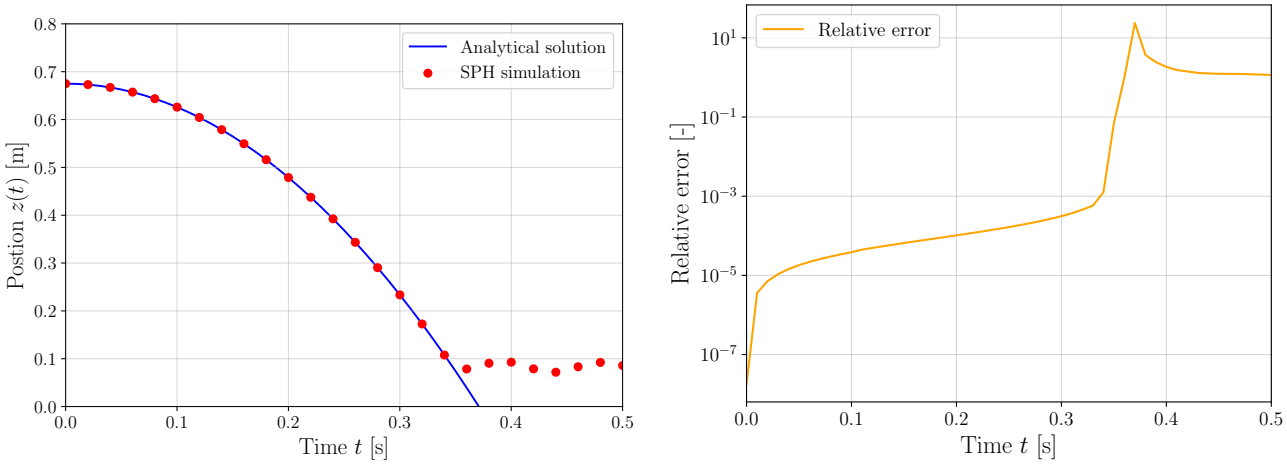
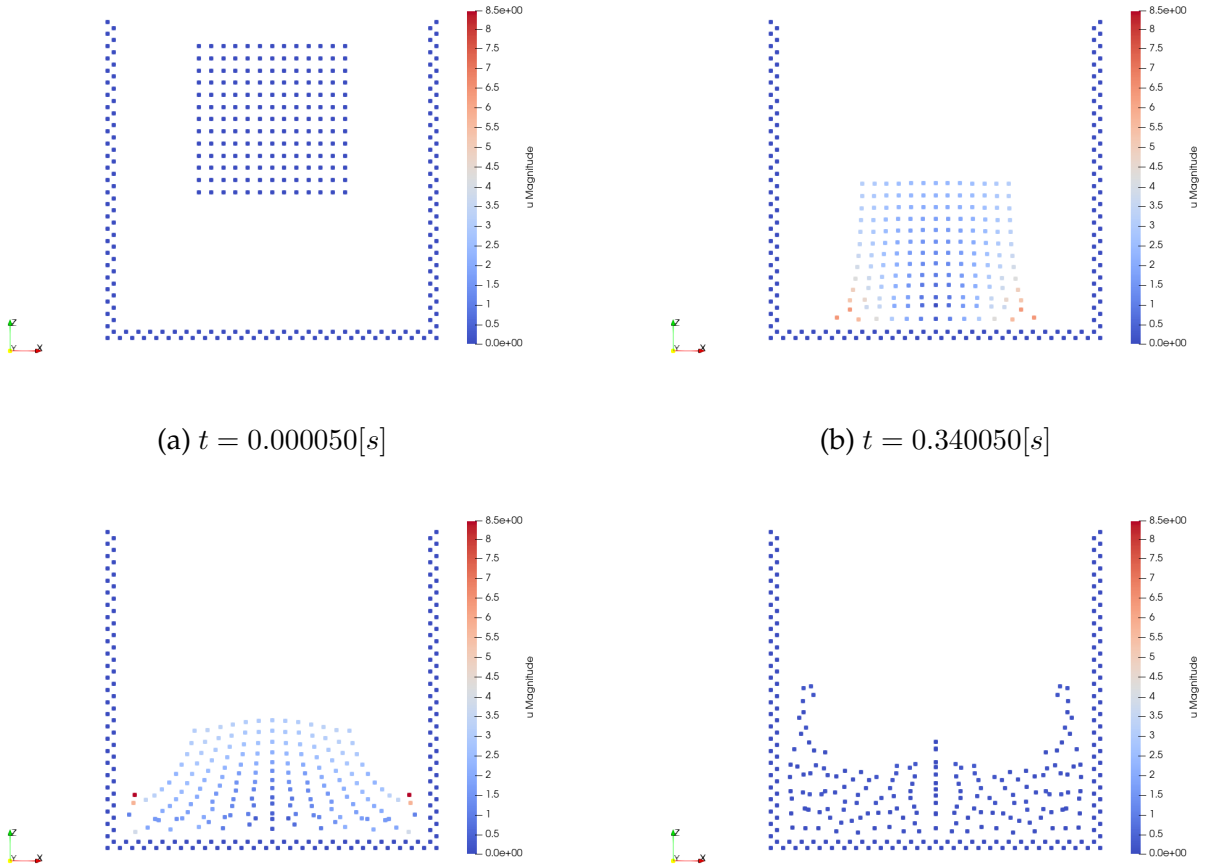


Figure 11: z position over time of a moving particle and comparison with the analytical solution (left). Relative error over time between the solution of the numerical simulation and the analytical solution (right) for the 2D simulation.

4 snapshots are represented in Fig.12 which illustrated the similar results between the 3D and 2D simulations. Moreover, the velocity is shown, for blueish particles the velocity is low while for reddish particles the velocity is higher.



(a) $t = 0.000050[s]$

(b) $t = 0.340050[s]$

(c) $t = 0.500050[s]$

(d) $t = 0.700050[s]$

Figure 12: Snapshots of the cube falling for different simulations time where blue particles represent low velocity and red particles represent high velocity for the 2D simulation.

III.2 Cube water at rest in a tank

In this second test, a tank is filled with water and is subject to gravity. The initial geometry is shown in Fig. 13 The aim of this test is to check if a liquid at rest reaches equilibrium under gravity and if a pressure gradient is observed and if it is linearly proportional to z.

Theoretical hydrostatic pressure

Reminding the momentum equation:

$$\frac{D\boldsymbol{u}}{Dt} = -\frac{\nabla p}{\rho} + \frac{\boldsymbol{F}}{\rho} = -\left(\nabla\left(\frac{p}{\rho}\right) + \frac{p}{\rho^2}\nabla\rho\right) + \frac{\boldsymbol{F}}{\rho}. \tag{III.2}$$

If the fluid is at rest, the left-hand-sided equation is null. Also, the single volume body force is the gravity. Hence, this latter equation reads:

$$\frac{\nabla p}{\rho} = g \Rightarrow p(z) = \rho q z \approx 9810 \times z, \tag{III.3}$$

where z is the depth of a given particle and where the density ρ is approximated constant at 1000 [kg/m^3] and by considering an initial condition where the pressure is zero at a zero z coordinate.

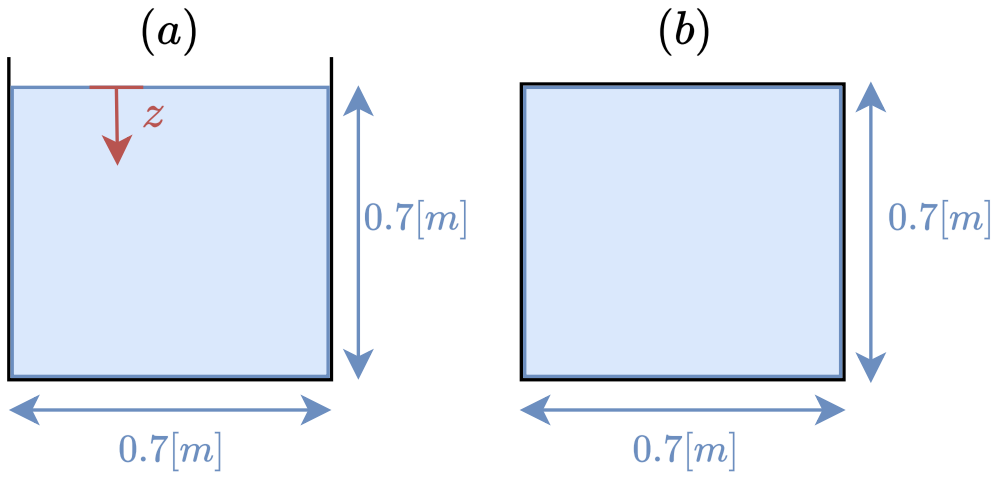


Figure 13: Initial geometry of the cube water at rest in a tank. Representation of the front view (a) and top view (b).

For the 3D case, the *3D_hydrostatic.json* file is used, and the characteristics of the simulation are provided in Table 4 below:

Simulation time [s]	Nb M.P.	Nb Part	s	α	c_0	Calculation time [s]
4.95005	24 389	33 818	0.025	0.5	30	13 428

Table 4: Different parameters of the 3D hydrostatic case.

Fig 14 (right) illustrates that a pressure gradient is observed in the liquid where a low pressure appears on the top and high pressure on the bottom of the tank.

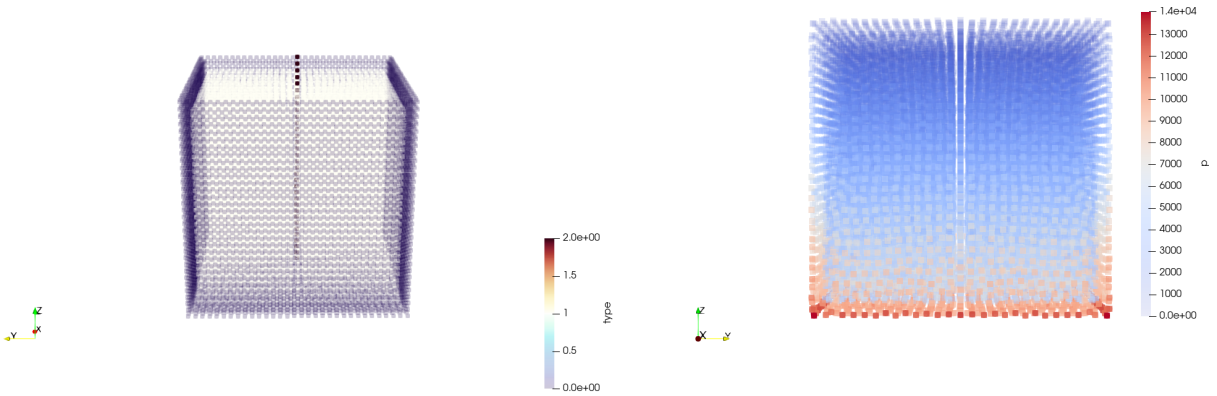


Figure 14: (left) Illustration of the tank fill with water, purple particles are the fixed particle, white particle are the moving particle and black particle are ghost particle used to obtain physical data. (right) Illustration of the pressure gradient in the tank at the end of the simulation. 3D case.

One thus expects a linear trend in the pressure from the SPH simulation. Thanks to the ghost particles (Sec. II.7), placed in a vertical line in the middle of the tank, the SPH-evaluated pressure is depicted in Fig. 15 (left), where it is compared with the theoretical pressure. One can see that the simulation yields good results compared to the theory. Moreover, it is noticeable that a few seconds are needed for the pressure to converge to the expected value. This damping effect is explained by the artificial viscosity, which can be adjusted by changing the value of α.

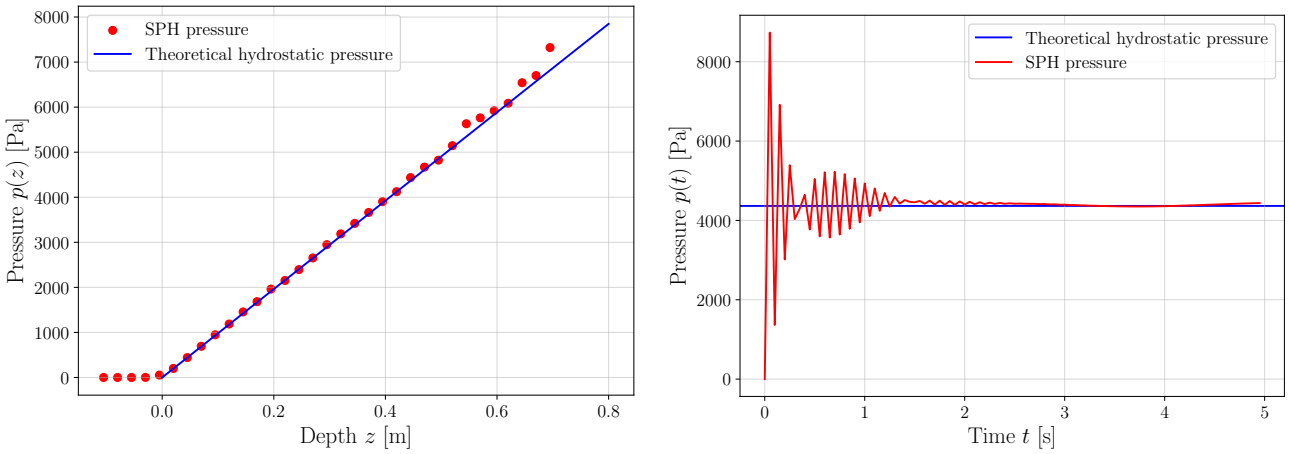


Figure 15: Pressure along the depth of liquid and comparison between the SPH pressure and the theorical pressure (left). Pressure over time of a ghost particle over time (right). For the 3D simulation.

Indeed, the expected behavior has been retrieved. A 2D simulation was also conducted, allowing the simulation to run faster and reducing the spacing, which increases the number of particles. Table 5 shows the parameters of the 2D simulation, and Fig. 17 illustrates the pressure over depth and the convergence. One can observe that the convergence requires more time. This is explained by Eq. (II.25), where the kinematic viscosity depends on the value of h and, consequently, on s.

Simulation time [s]	Nb M.P.	Nb Part	s	α	c_0	Calculation time [s]
7.485010	1 296	1 527	0.02	0.5	30	2 265

Table 5: Different parameters of the 2D hydrostatic case with Euler integration scheme.

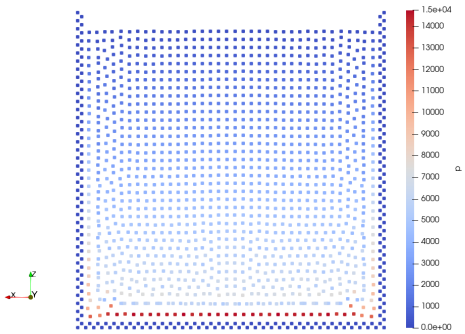


Figure 16: Illustration of the pressure gradient in the tank at the end of the simulation. 2D case.

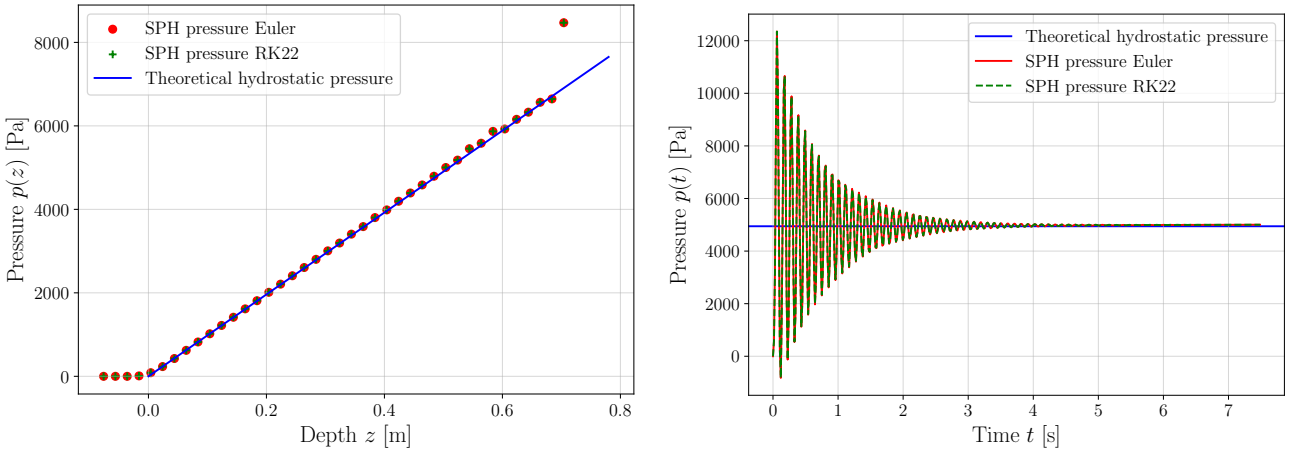


Figure 17: Pressure along the depth of liquid and comparison between the SPH pressure and the theorical pressure (left). Pressure over time of a ghost particle over time (right). For the 2D simulation. Simulation done with the Euler and RK22 scheme.

Moreover, Fig. 17 shows that both the Euler and RK22 schemes produce nearly identical results. However, the RK22 simulation took 3,323 seconds to complete, which is approximately 20 minutes longer than the Euler simulation (2,265 seconds). Given that the results of the two simulations are nearly the same and that the RK22 scheme requires more computation time, the decision was made to use the Euler scheme for this project.

III.3 Dam break

This third case is the dam break. The initial geometry of this problem is illustrated in Fig 18 and the different parameters are shown in Table 6. The json files used is *3D_dam_break.json* and *2D_dam_break.json*.

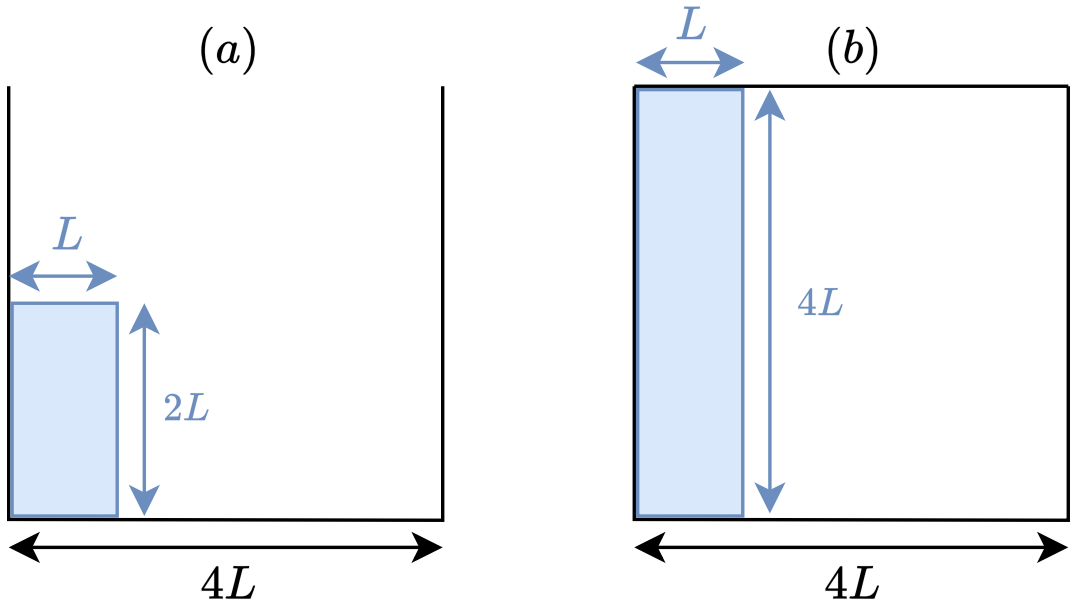


Figure 18: Initial geometry of the dam break test. Representation of the front view (a) and top view (b).

Simulation time [s]	Nb M.P.	Nb Part	s	α	c_0	Calculation time [s]
1.24501	9 471	27 027	0.05	0.1	30	2 265

Table 6: Different parameters of the 3D dam break case.

The aim of this test is to verify whether the wavefront produced by the SPH simulation matches the wavefront experimentally observed in the study by [Koshizuka] [5]. The experimental data points were manually extracted using IMAGEJ, so some errors may be present. In Fig. 19, the graph is shown in dimensionless form. The position of the wavefront x is divided by the characteristic length L, and the dimensionless time is obtained using the gravity g and L. The simulation closely follows the experimental dimensionless position. However, the numerical wavefront appears to be slightly faster than the experimental one. This discrepancy could be due to the viscosity being too low, resulting in a higher velocity.

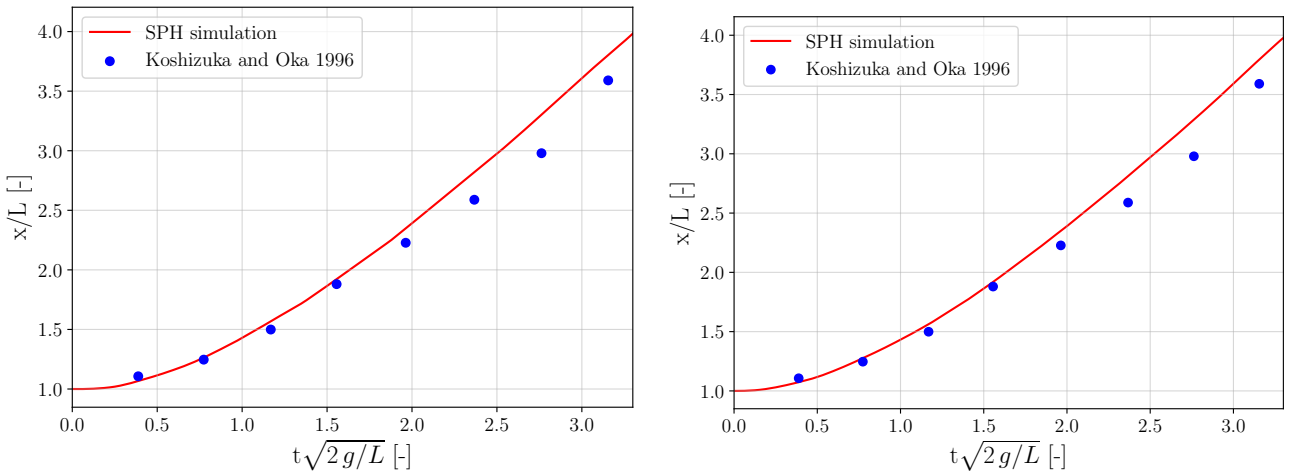
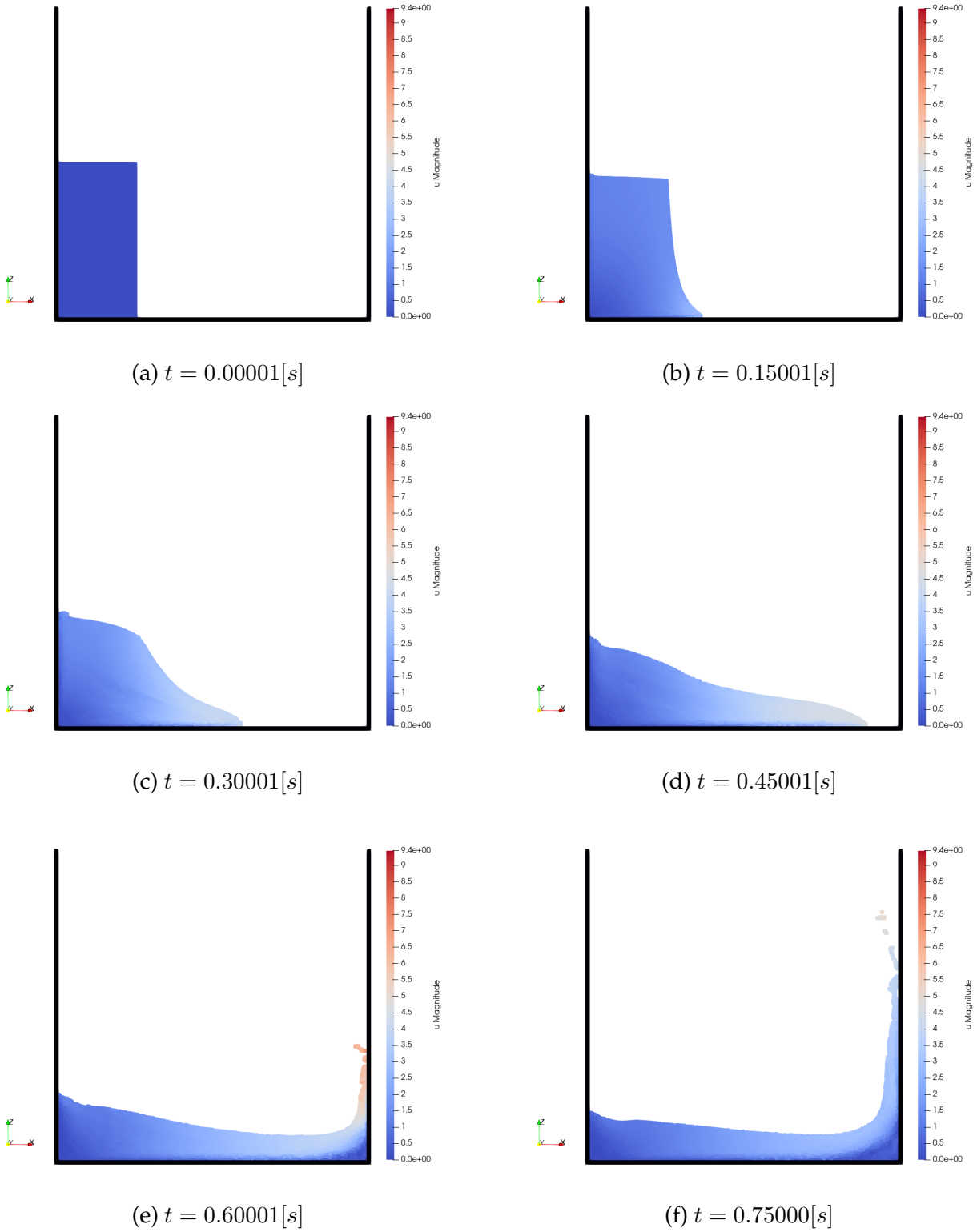


Figure 19: Position of the wave front over time in a dimensionless form where L = 0.5 m. Comparison between the numerical results and the experiment from [5]. 3D simulation (left), 2D simulation (right).

To illustrate the simulation, snapshots are presented in Fig 20 below where the velocity of particles is shown.



(a) $t = 0.00001[s]$

(b) $t = 0.15001[s]$

(c) $t = 0.30001[s]$

(d) $t = 0.45001[s]$

(e) $t = 0.60001[s]$

(f) $t = 0.75000[s]$

Figure 20: Snapshots of the 2D dam break simulation at different time.

IV Microfluidic phenomenon [Akinci et al.]

The main advantage of the SPH method is the mesh-free model. But this feature may become a drawback while surface tensions are involved, since the "boundary surfaces" have to be determined to apply surface tensions onto it.

Many papers use different methods to achieve this, our focus has been on a specific paper from Akinci et al. [1].

IV.1 Surface tension implementation

In their work, Akinci et al. introduce a cohesion method from a macroscopic perspective. Rather than solely focusing on cohesive forces, they also identify forces aimed at minimizing surface area. Their approach calculates the cohesive force of a particle as:

$$F_{\text{cohesion}_{i\leftarrow j}} = -\alpha_{\text{cohesion}} m_i m_j \frac{x_i - x_j}{\|x_i - x_j\|} W_{\text{cohesion}}(\|x_i - x_j\|), \tag{IV.1}$$

where i and j are neighbouring particles, α_{cohesion} a scaling factor and W_{cohesion} a special cohesion kernel which is defined by:

$$W_{\text{cohesion}}(r) = \frac{32}{\pi h^9} \begin{cases} (h-r)^3 r^3 & \text{if } \frac{h}{2} < r \leq h, \\ 2(h-r)^3 r^3 - \frac{h^6}{64} & \text{if } 0 < r \leq \frac{h}{2}, \\ 0 & \text{otherwise.} \end{cases} \tag{IV.2}$$

Where h represents the support radius. The cohesion kernel is represented in Fig 21:

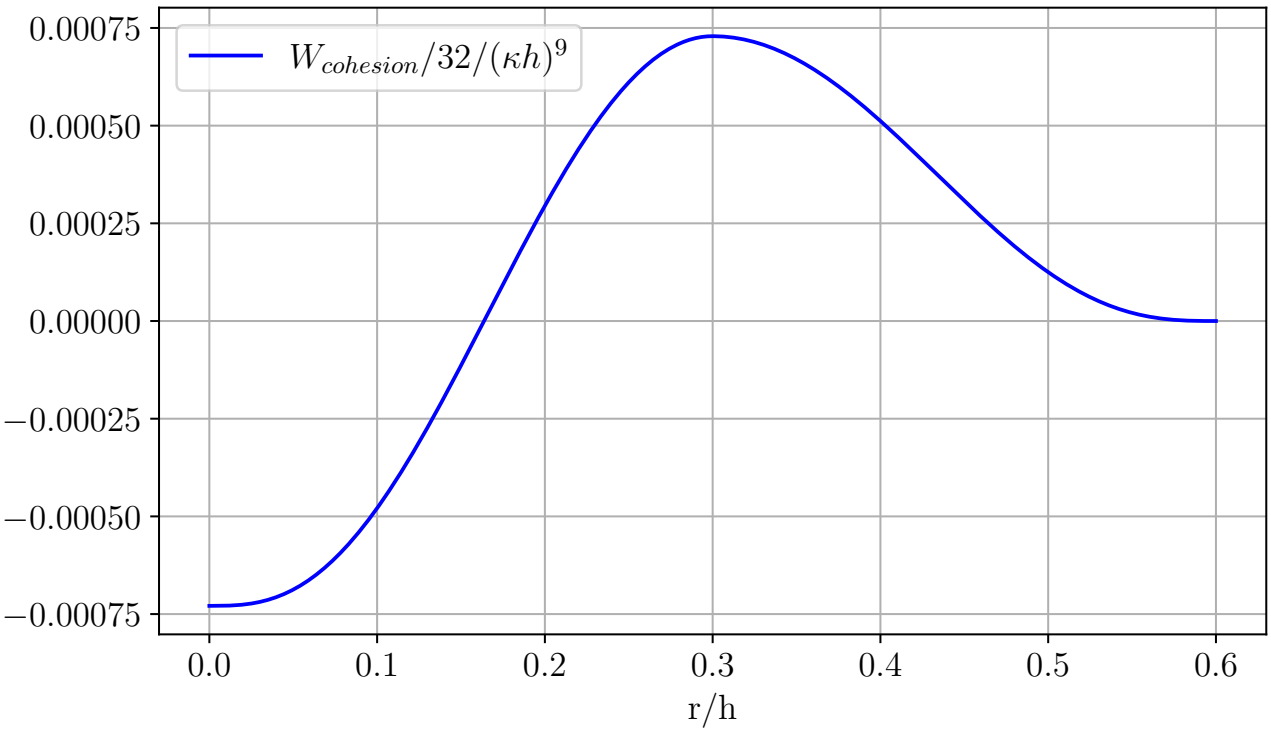


Figure 21: Cohesion kernel function with $\kappa = 1$ and $s = 0.025$.

Unlike the model proposed by Becker and Teschner [2], where cohesive forces are strictly positive, Akinci et al.'s approach allows for both positive and negative cohesive forces. This

feature generates repulsion forces between closely positioned particles, preventing undesirable clustering at the free surface.

Furthermore, Akinci et al. incorporate a force aimed at minimizing surface area. This supplementary force counteracts surface curvature, necessitating the computation of surface normals. These normals can be obtained using a technique called the colour field method. Essentially, this involves assigning a colour value of 1 to the particle and 0 elsewhere, then computing the gradient of the smoothed colour field:

$$\mathbf{n}_i = h \sum_j \frac{m_j}{\rho_j} \nabla W_{ij}, \tag{IV.3}$$

This calculation provides a surface normal directed inward into the fluid, with the factor h utilized to ensure normal scale independence. Within the fluid interior, the resulting vector's magnitude approaches zero, while at the free surface, it becomes proportional to curvature. Consequently, a symmetric force, opposing curvature, can be formulated as:

$$F_{\text{curvature}_{i \leftarrow j}} = -\alpha_{curv} m_i (\mathbf{n}_i - \mathbf{n}_j). \tag{IV.4}$$

where α_{curv} is a simple scaling factor (different from the previous α used). This force is used to minimize the surface area. It is important to notice that $\alpha_{\text{cohesion}} = \alpha_{\text{curv}} = \gamma$ in Akinci et al. model, since these two forces act together to generate the surface tensions. In the simulations of this study, one calls this parameter $\alpha_{\text{s.t.}}$.

Finally, both forces are combined as:

$$F_{\text{surface tension}_{i \leftarrow j}} = K_{ij} (F_{\text{cohesion}_i} + F_{\text{curvature}_i}), \tag{IV.5}$$

where, $K_{ij} = \frac{2\rho_0}{\rho_i + \rho_j}$ is as a symmetric factor enhancing surface tension forces at the free surface. Near the surface, ρ_i and ρ_j are underestimated due to particle deficiency, resulting in $K_{ij} > 1$. Conversely, for a particle with a complete neighbourhood, K_{ij} approximates 1.

IV.1.1 Analysis

In order to check if the surface tension equation given by Eq.(IV.5) can be used for microfluidic application. One simulates a cube of fluid particles (without any other fixed particles) that should turn into a sphere. Physically, surface tensions tend to minimize the surface area of the fluid, such that the sphere is the shape that allow this condition. Because of the curvature of the fluid, a pressure difference is induced between the outside and the inside, this pressure is called the *Laplace pressure* Δp, and is given by:

$$\Delta p = \sigma \left(\frac{1}{R_1} + \frac{1}{R_2} \right), \tag{IV.6}$$

where σ is the surface tension [Pa/m^2] and R_1, R_2 are the principal radii of curvature.

For a sphere, $R_1 = R_2 = R\,[m]$ and so Eq.(IV.6) becomes:

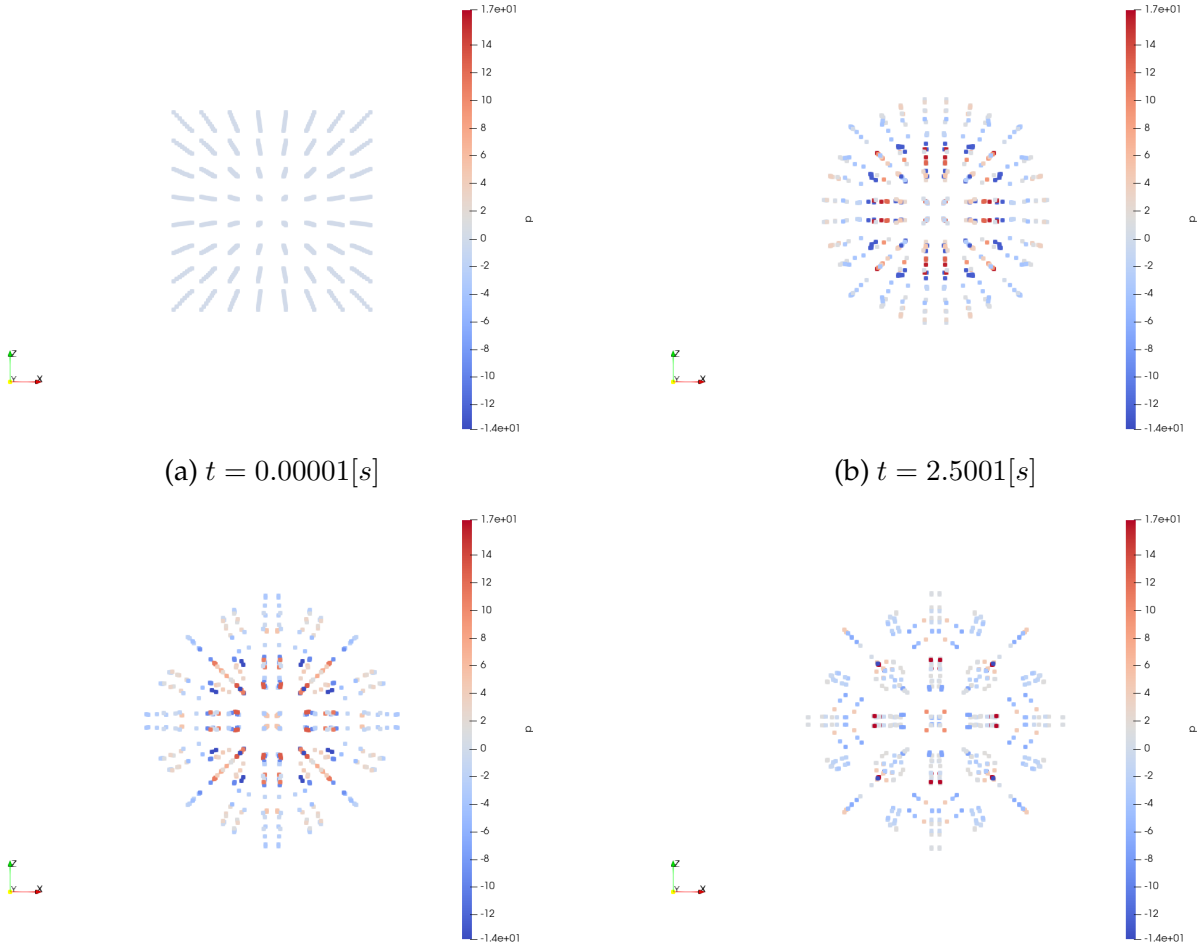
$$\Delta p = \sigma \frac{2}{R}. \tag{IV.7}$$

The final radius R can be found analytically. Indeed, surface tensions tends to minimize the surface area but the volume stays constant. With a cube of length L, the volume of the cube is L^3 and the volume of the sphere is $\frac{4}{3}\pi\, R^3$. By egalizing the two expressions, one obtains

$$\begin{aligned} \frac{4}{3}\pi\, R^3 &= L^3, \\ R &= \frac{L}{\sqrt[3]{\frac{4}{3}\pi}}, \end{aligned} \tag{IV.8}$$

for a 3D case. For a 2D simulation the pressure is expected to be $\Delta p = \frac{\sigma}{R}$, and the theoretical radius to be $R = \frac{L}{\sqrt{\pi}}$.

It represents a cube with a side length of L where only surface tension (excluding gravity) is considered. At the macro scale, the surface-to-volume ratio is so small that surface tensions are negligible compared to volume body forces like gravity and hydrostatic pressure. However, since one can manually choose to apply only the surface tension to the water cube, surface tension effects can be included regardless of the simulation scale. Additionally, the cube of water is assumed to flow in a vacuum, so no hydrostatic pressure is generated. Some time step of a simulation are shown in Fig 22. It is visible that the cube become a sphere after 0.25 seconds. However, it continues to deform and its shape completely changes. Some clusters appears and some of them are ejected from the sphere. It can come from a to large dt, so different initial time step was used. However, the simulation always diverges, let us think it may come from the method itself when a small number of particles simulate.



(a) $t = 0.00001[s]$ (b) $t = 2.5001[s]$

(c) $t = 3.75001[s]$ (d) $t = 6.25001[s]$

Figure 22: Snapshot of a simulation of a cube (blue points) with $L = 0.1$ [m] with a spacing s = 0.015 where the surface tension is applied with $\alpha_{\text{s.t.}}$ = 1.

From this, the pressure distribution can be illustrated at $t = 2.5$ s. On the one hand, a pressure difference (considering the outside pressure is zero) develops and is shown in Fig. 23. Although some fluctuations occur, the pressure distribution remains within the same order of magnitude as the theoretical prediction. On the other hand, regarding the radius, it starts at 0.5 and continues to increase, even surpassing the theoretical radius. The radius

seems to stabilized near 0.075 m, which is a radius that cannot be explain physically. The JSON file used for the different simulations is *3D_cube_to_sphere.json*.

Parameters	cube to sphere
s	0.015
L	0.1
$Nb\ MP$	512
α_{curv}	1
c_0	30
α	0.1
Simulation time	8 000 s

Table 7: Parameters of the cube to sphere simulation.

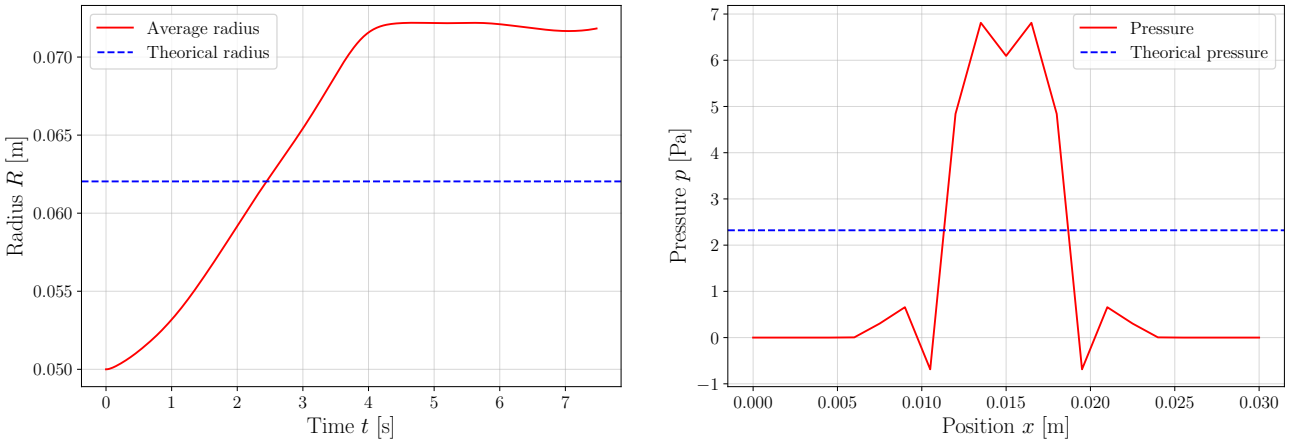


Figure 23: Radius over time for the simulation (left), pressure along a line passing trough the sphere (right).

In summary, this method is straightforward, and the results reflect this simplicity. However, several improvements can be made. Increasing the number of particles may enhance stability, as more particles would generate greater forces, potentially holding the sphere together and yielding more accurate physical results. However, this would significantly increase computation time. Additionally, the parameter α_{st} is a numerical rather than a physical parameter, making this method more challenging to apply to liquids other than water.

IV.2 Adhesion effect

To simulate the attractive forces between fluid particles and fixed particles, an adhesion force is introduced as:

$$\boldsymbol{F}_i^{\text{adhesion}} = -\beta_{adh} \sum_k m_i m_k \frac{\boldsymbol{x}_i - \boldsymbol{x}_k}{||\boldsymbol{x}_i - \boldsymbol{x}_k||} W_{\text{adhesion}}(||\boldsymbol{x}_i - \boldsymbol{x}_k||). \tag{IV.9}$$

where β is the adhesion coefficient and k denotes a neighbouring boundary particle.

Also, a specialized kernel is used to mimic the adhesion influence:

$$W_{\text{adhesion}}(r) = \frac{0.007}{h^{3.25}} \begin{cases} \sqrt[4]{-\frac{4r^2}{h} + 6r - 2h} & \text{if } \frac{h}{2} < r < h, \\ 0 & \text{otherwise.} \end{cases} \tag{IV.10}$$

Where h represents the support radius. This adhesion Kernel is illustrated in Fig 24.

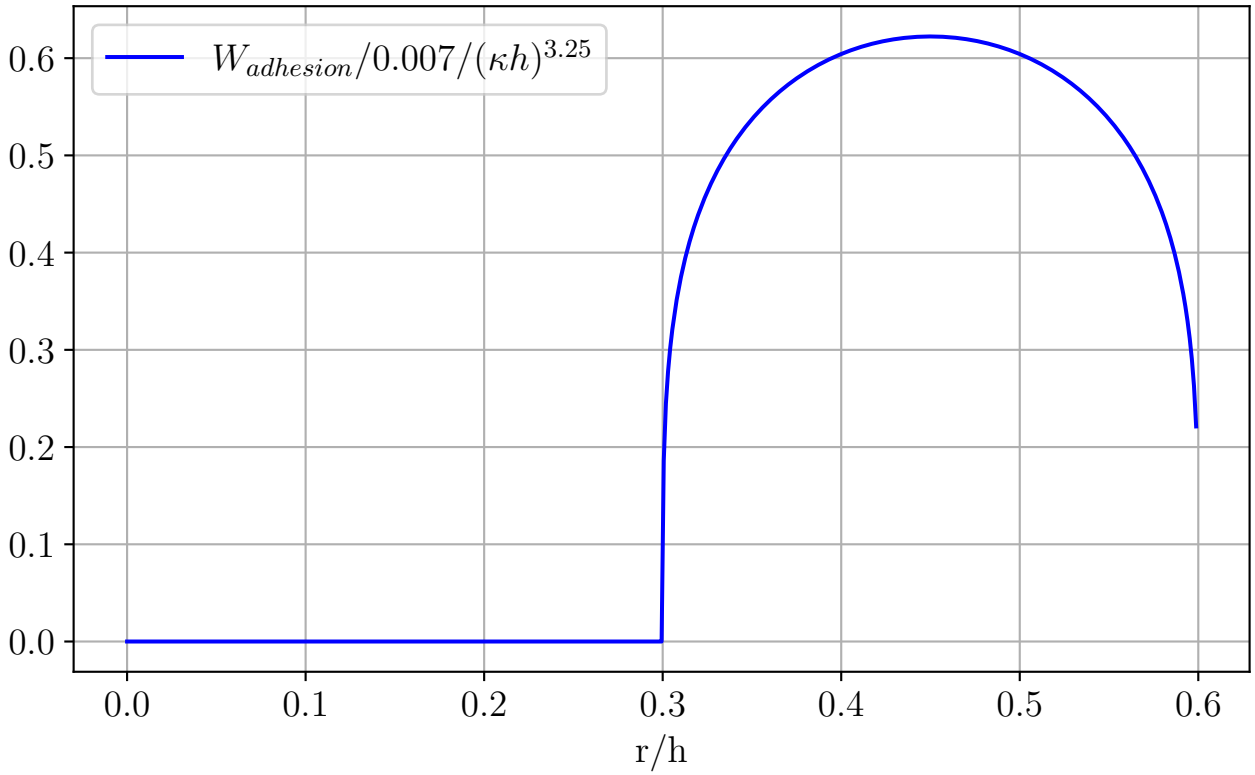


Figure 24: Adhesion kernel function for $\kappa = 1$ and $s = 0.25$.

The specific shape of this new kernel is designed to attract particles only at distances between $h/2$ and h, while strongly attracting almost all the particles in the neighbourhood of a given particle. The primary goal of this design is to ensure that all fixed neighbouring particles of a given particle play a significant role.

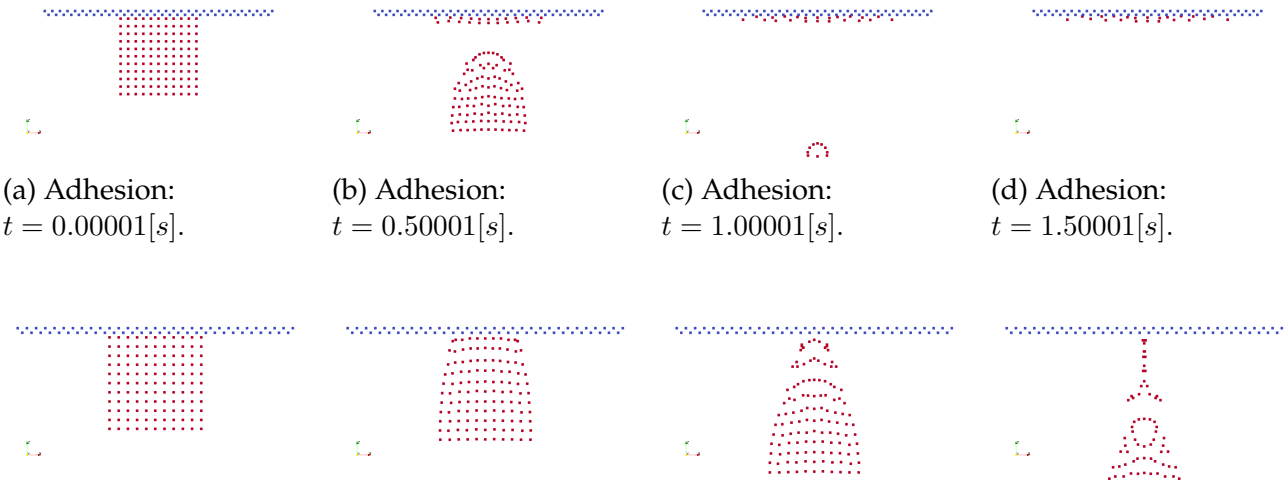
IV.2.1 Analysis

In order to understand the implication of the adhesion force, a simple simulation where a cube of water starts close to an upper boundary and in which gravity is applied. On the one hand, without adhesion, all particles of the cube should fall. On the other hand, with adhesion, particles of the cube close to the boundary should be attracted to and their fall should be slow down or even completely stop if the adhesion force surpasses gravity. Two simulation are compared on Fig 25, it can be seen that the particles near the upper boundary (ceiling) are attracted, and this force cancel out gravity. However, particles which are not close to the ceiling fall in the same manner, either adhesion is applied or not. As for $\alpha_{\text{s.t.}}$, β_{adh} is a numerical instead of a physical parameter and the same issues as before occurs. The JSON file *2D_adhesion.json* and *2D_no_adhesion.json* have been used. Table 8 contains several important characteristics of the simulation.

s	α	c_0	ρ_0	simulation time [s]	Nb M.P	Nb Part
0.1	0.05	15	1000	0.49501	121	182

Table 8: Characteristics of the 2D adhesion simulation.

In Fig. 25, it can be noticed when the adhesion force is added the bottom part of the cube fall earlier. It can be explained by the fact that the adhesion force acts on the two top layers of the cube and the other layers are only submitted to gravity. The top part tends to go up and the bottom part tends to go down, leading to separate the two parts easily.



(a) Adhesion: $t = 0.00001[s]$.
(b) Adhesion: $t = 0.50001[s]$.
(c) Adhesion: $t = 1.00001[s]$.
(d) Adhesion: $t = 1.50001[s]$.
(e) No adhesion: $t = 0.00001[s]$.
(f) No adhesion: $t = 0.50001[s]$.
(g) No adhesion: $t = 1.00001[s]$.
(h) No adhesion: $t = 1.50001[s]$.

Figure 25: 2D simulation of a fluid cube (red particles) near an upper boundary (blue particles), falling due to gravity and where adhesion is applied (a, b, c, d) and where is not (e, f, g, h). With a spacing $s = 0.01$, a length $L = 0.1$ and $\beta_{adh} = 0.5$.

V Microfluidic phenomenon [M. Ordoubadi et al.]

In order to remedy the lack of meaningful physical variables in the surface tensions implementation, one investigated another method implemented by M. Ordoubadi et al. [9].

V.1 Surface tension implementation

The essence of their approach lies in the fluid surface particle tracking algorithm. In particular, they introduce a mathematical surface tracking technique that allows for the rapid and efficient identification of particles at the free surface. Although their work used the *Wendland quintic kernel*, the results obtained with our standard cubic kernel were so comparable that the decision was made to use the same kernel for all simulations.

Mathematical Surface Tracking

This method is based on the principle that the absolute value of the divergence of a particle's position vector is approximately 2 if the particle is fully surrounded by other particles; this implies that it is not located on the free surface [6]. The divergence of the position vector is defined as:

$$(\mathbf{r} \cdot \nabla \mathbf{r})_i = \sum_{j=1}^{N} m_j \frac{\nabla_j W_{ij}}{\rho_j} \cdot (\mathbf{r}_{ij} - \mathbf{r}_i), \tag{V.1}$$

where W_{ij} represents the kernel function and ρ_j is the density at the location of particle j.

If:

$$|\mathbf{r} \cdot \nabla \mathbf{r}|_i \leq \epsilon, \tag{V.2}$$

then particle i is located on the free surface, with ϵ being a user-defined threshold value less than 2. In their study, ϵ is chosen to be 1.7, balancing accuracy and efficiency. Throughout

their study, this technique is referred to as the mathematical interface tracking method. Although a geometric free surface particle tracking algorithm is also considered in the paper, the mathematical approach alone produces reliable results, as demonstrated in Fig.26:

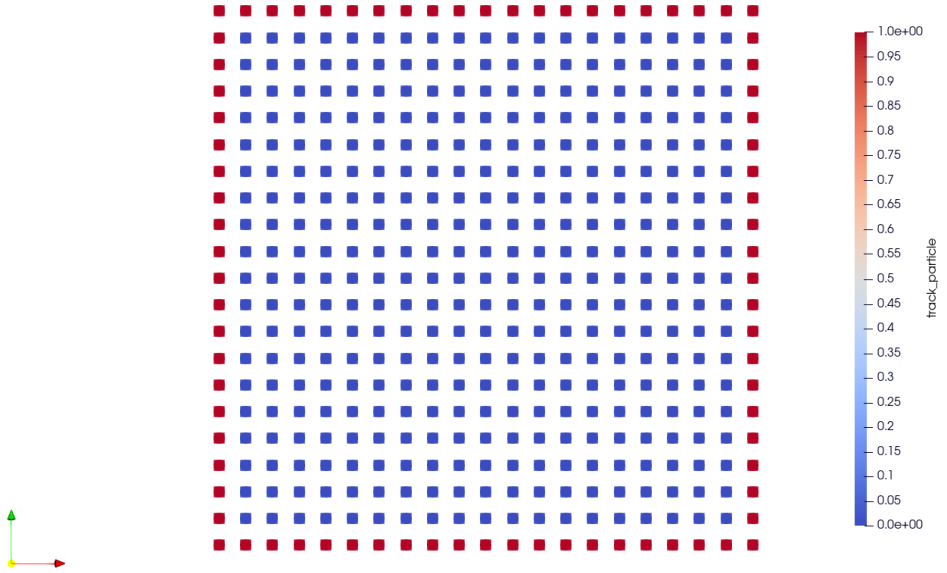


Figure 26: Application of the mathematical tracking surface, red particles represent free surface particles and blue particle are the inner ones.

Smoothed Color Function

In contrast to the method proposed by Akinci et al., the smoothed colour function is calculated differently depending on whether the particle is located at the free surface.

By applying this method, the smoothed colour function identifies a free surface wherever the particle density is low, independent of the interface tracking method described previously. In many impact scenarios, voids can form within the fluid due to particle distortion, leading to incorrect free surface detection. These nonphysical free surfaces can generate artificial surface tension forces, potentially causing the simulation to fail.

To avoid this issue, an integer variable N is introduced for each particle, which can be either 0 or 1. At the beginning of each time step, N is initialised to 0 for all particles. After interface tracking is completed, each particle is checked to see if it has any neighbouring free surface particles within its support domain. If a particle has a free surface neighbour, its N value is set to 1. Particles without free surface neighbours retain their initial value of 0.

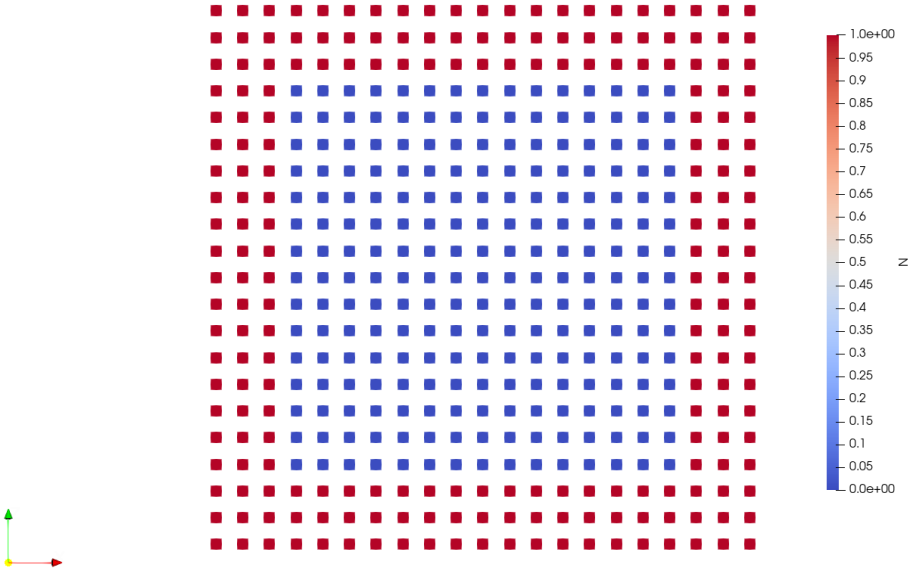


Figure 27: Fluid particles having at least one free surface particle as neighbour.

For all particles, the smoothed colour function is defined as follows:

$$c_i = \begin{cases} \sum_{j=1}^{N} m_j \frac{c_j^0}{\rho_j} W_{ij}, & \text{if } N = 1, \\ 1, & \text{otherwise.} \end{cases} \tag{V.3}$$

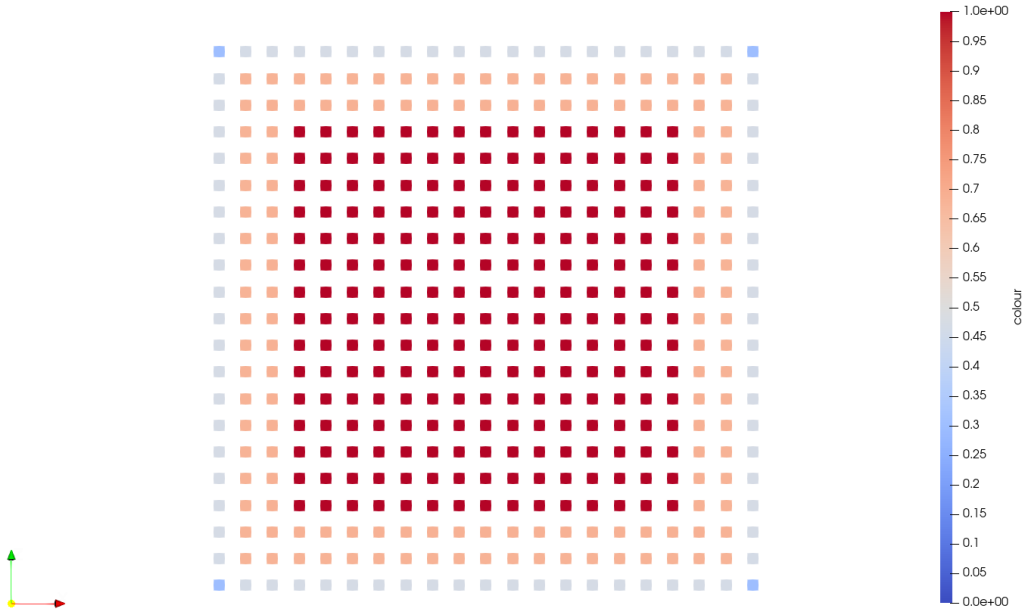


Figure 28: Smoother colour function applied onto the fluid cube.

Indeed, particles having their associated N value equal to 1 endure a smoother colour value whereas the other have a value of 1.

Normal Vector

The normal vector is calculated using the following expression:

$$\mathbf{n} = -\nabla c. \tag{25}$$

This expression is discretized according to the formulation by Morris [**?**] as follows:

$$\mathbf{n}_i = -\sum_{j=1}^{N} m_j \frac{(c_j - c_i)}{\rho_j} \nabla W_{ij}. \tag{V.4}$$

The variation in colour functions within the above equation ensures that the normal vector becomes exactly zero in areas far from the free surface. In many practical applications, significant curvatures can develop during the simulation (for example, at sharp corners). In these areas, the kernel's support domain may be severely truncated, resulting in very small normal vector values and consequently, reduced surface tension.

To overcome this limitation and enhance the accuracy of curvature and surface tension calculations, imaginary particles are introduced near the free surface. These particles contribute to the determination of the normal vector and curvature for particles close to the free surface. The procedure for identifying and implementing these imaginary particles is outlined below:

Consider particle i in Eq.(V.4) with its neighbours indexed by j. The procedure differs depending on whether particle i is located on the free surface.

Firstly, assume that particle i lies on the free surface. For any particle j within the support domain of i that is not on the free surface, the position vector between particles i and j is reflected in the opposite direction. This new vector, originating at i, indicates the position of an imaginary particle j'. This imaginary particle contributes to Eq.(V.4) for particle i, with a colour function value of 0, and density and mass equal to those of particle i. The kernel gradient $\nabla W_{ij'}$ equals $-\nabla W_{ij}$. In this situation, the imaginary particle j' is guaranteed to remain within the support domain of particle i.

Now, consider the case where particle i is not on the free surface but has at least one neighbouring particle on the surface (i.e., $N_i = 1$). For each surface particle j within the support domain of i, the vector $\mathbf{r}_{ij}$ is constructed. Next, the vector $\mathbf{r}_{jj'}$ is constructed with the same magnitude and direction as $\mathbf{r}_{ij}$, so that $\mathbf{r}_{ij'} = \mathbf{r}_{ij} + \mathbf{r}_{jj'}$. Since the length of vector $\mathbf{r}_{ij'}$ is twice that of $\mathbf{r}_{ij}$, it must be ensured that the imaginary particle j' stays within the support domain of particle i. If j' lies within the support domain, its contribution to Eq.(V.4) for particle i is included. In this case, unlike the previous one, the kernel gradient must be recalculated.

Surface Curvature and Surface Tension

The magnitude and direction of the normal vector are accurate near the free surface, but in other regions inside the fluid, the normal vector has very small values, leading to some error. To identify particles with reliable normal vectors, an integer variable R is defined as follows:

$$R_i = \begin{cases} 1 & \text{if } |\mathbf{n}_i| > \epsilon_0, \\ 0 & \text{if } |\mathbf{n}_i| \leq \epsilon_0, \end{cases} \tag{V.5}$$

where $\epsilon_0 = 0.01/h$.

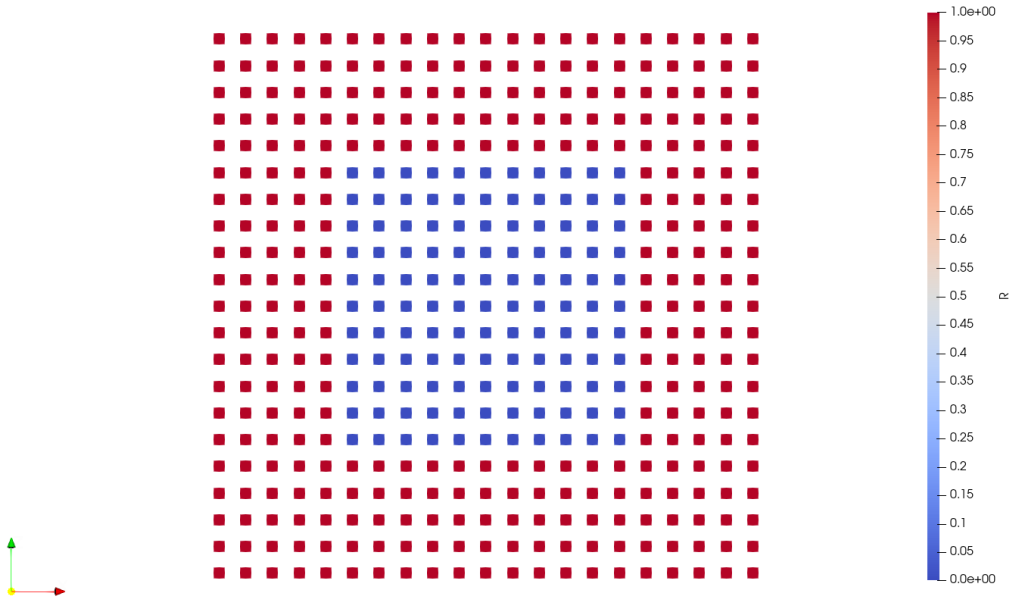


Figure 29: Truncation of the reliable normal vectors applied onto the fluid cube.

An intermediate approximation for curvature can then be obtained from:

$$\kappa_i = (\nabla \cdot \hat{\mathbf{n}})_i = \sum_{j=1}^{N} \min(R_i, R_j) \frac{m_j}{\rho_j} (\hat{\mathbf{n}}_j - \hat{\mathbf{n}}_i) \cdot \nabla W_{ij} \mathbf{r}_i \tag{V.6}$$

Due to the application of this filter, the kernel support domain of particles will be significantly truncated. To fix this, a correction to the kernel gradient is applied in Eq.(V.6) as follows:

$$\kappa_i = \frac{\kappa_i'}{L_i}, \tag{V.7}$$

where:

$$L_i = \sum_{j=1}^{N} \min(R_i, R_j) \frac{m_j}{\rho_j} W_{ij}. \tag{V.8}$$

In the curvature calculation using Eqs.(V.6)-(V.8), the imaginary particles introduced previously are also used to further enhance accuracy. The unit normal vector direction for the imaginary particles is calculated as follows:

If particle i is on the free surface and has a neighbour j not on the surface, an approximation for the unit normal vector of the imaginary particle j' is obtained from:

$$\hat{\mathbf{n}}_{j'} = 2\hat{\mathbf{n}}_i - \hat{\mathbf{n}}_j, \tag{V.9}$$

and the unit normal vector is:

$$\hat{\mathbf{n}}_{j'} = \frac{\hat{\mathbf{n}}_{j'}}{|\hat{\mathbf{n}}_{j'}|}. \tag{V.10}$$

Now, consider the case where particle i is not located on the free surface. An approximate unit normal vector for the imaginary particle j' is:

$$\hat{\mathbf{n}}_{j'} = 2\hat{\mathbf{n}}_j - \hat{\mathbf{n}}_i, \tag{V.11}$$

and the unit normal vector is again obtained from Eq.(V.10).

Below is represented of κ:

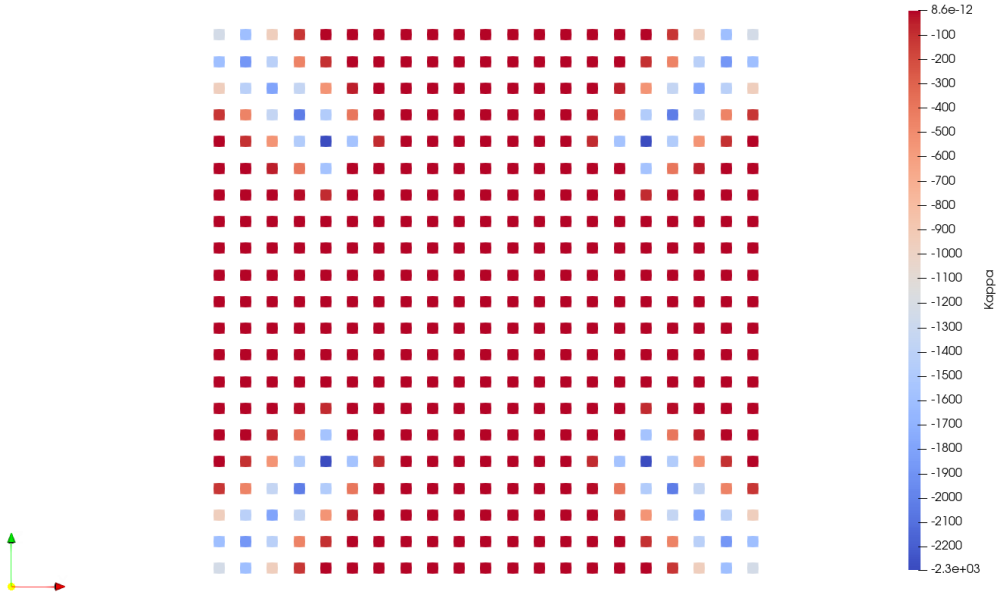


Figure 30: Curvature inside the cube for each particle.

Indeed, the particles, and more precisely near the corner (along the two diagonals of the cube) endure a negative curvature.

After calculating the surface normal vectors and curvature, the acceleration of each particle due to surface tension is given by:

$$(\mathbf{a}_s)_i = \frac{\sigma \kappa_i}{\rho_i}. \tag{V.12}$$

V.2 Analysis

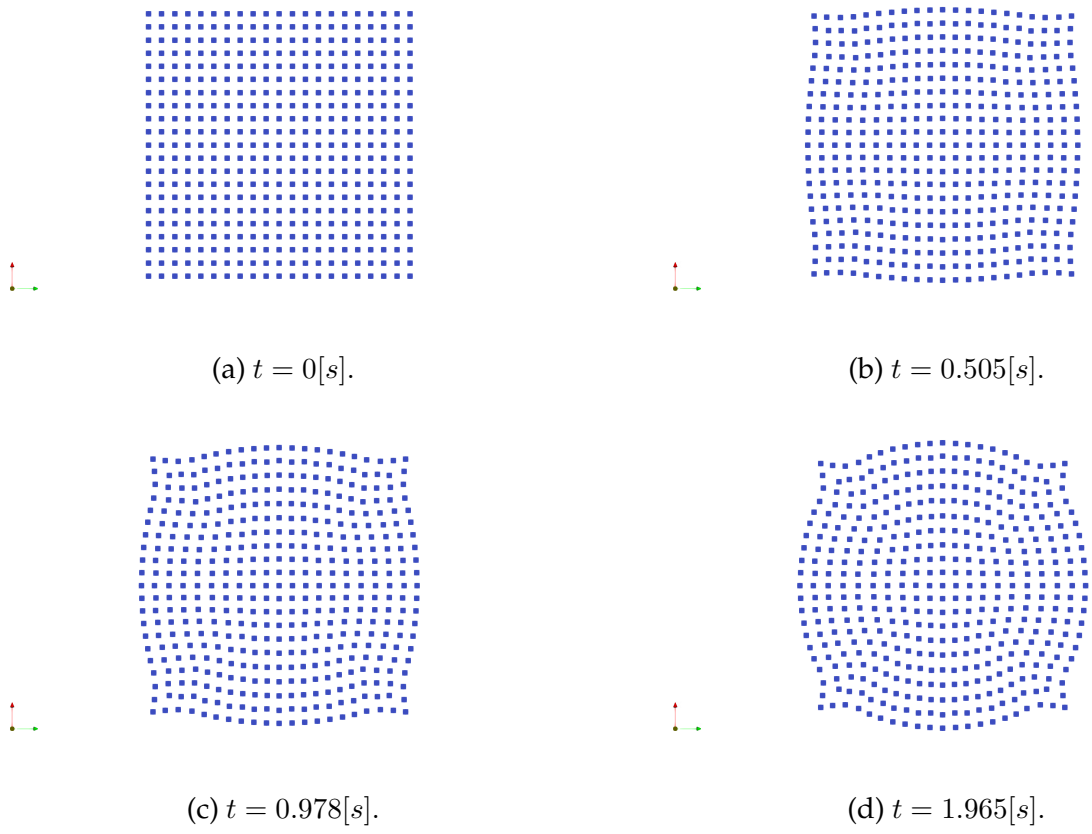
To compare this new method, the test case presented in Section IV was used. The main advantage of this method is that the σ used in further calculations physically represents the surface tension of a fluid.

The JSON file used for the different simulations is *2D_cube_to_sphere.json*.

Parameters	Cube to Sphere
s	5.10^{-4}
L	0.01
$Nb\ MP$	305
σ	0.072
c_0	30
dt_{init}	10^{-5}
α	1.5

Table 9: Parameters of the cube to sphere simulation (new surface tension).

First, the transformation of a cube into a sphere is analysed.



(a) $t = 0[s]$.

(b) $t = 0.505[s]$.

(c) $t = 0.978[s]$.

(d) $t = 1.965[s]$.

Figure 31: Snapshot of a simulation of a cube (blue points) with $L = 0.01$ [m] with a spacing s = 0.0005 where the surface tension is applied with γ = 0.072.

It is evident that the cube successfully transforms into a spherical shape. However, the sharp corners influence the overall particle arrangement, leading to convergence challenges. While the overall simulation appears symmetric, the particles along the diagonal of the initial cube seem less attracted compared to the others.

The impact of the imaginary particles seems to be minimal, which could suggest that the method is not fully optimised or that there may be an error in the algorithm. Nonetheless, the simulation is considerably less chaotic than the first method, with all particles moving smoothly and symmetrically.

In conclusion, this method offers several advantages, including simulation stability and the use of physical parameters. However, certain drawbacks, such as the high computational cost and the questionable results observed earlier, raise concerns about the method's overall utility. Further investigation is needed to establish greater confidence in this new approach.

VI Conclusion

In this study, we first introduced the three fundamental concepts of the Smoothed-particle hydrodynamics (SPH) method (Sec. I). Firstly, the SPH method is Lagrangian, meaning that the computational grid deforms with the object under study. Secondly, it is mesh-free, as it does not require a mesh for field variable interpolation, although mesh data may be needed for locating neighboring particles. Lastly, the method represents matter or fluid as a collection of particles that interact based on their distances (smoothing function), forming the core of the method's computational framework.

Next, the main mathematical concepts used in the study were introduced. The central concept is the *integral representation* (Eq. (II.1)), which allows for evaluating a function f(x) at any point in a domain Ω using the Dirac delta function. The second key concept is the smoothing of the Dirac delta function using a *smoothing function*, which represents the influence of a given neighborhood on a specific function evaluated at a point. This neighborhood support is described by a radius κh around a given location. The smoothing length h is kept constant throughout the simulation and is chosen as $h = 1.2s$ where s is the minimum spacing between particles. The smoothing function, also known as the *kernel function*, used in this study is the cubic spline kernel (Eq. (II.5)) with $\kappa = 2$, chosen for its renowned accuracy in classical numerical results. Spatial discretization is achieved using the *summation density* method (Eq. (II.7)) to represent a function at a point. For time discretization, the *Euler explicit method* (Eq. (II.12)) was chosen for its relatively low computational cost and the associated timestep threshold (Eq. (II.39)). Following this, the particle sorting algorithm, specifically the *Linked-list* algorithm, was discussed in Sec. II.5. This algorithm is more efficient than the *naive* algorithm for particle numbers exceeding one thousand. Finally, the Navier-Stokes equations adapted for the SPH method (Eq.(II.21) and Eq.(II.22)) were introduced, along with the *artificial viscosity* variables used to mimic the friction between particles (Eq.(II.23)).

Subsequently, two relationships between pressure and density were investigated (Sec.II.8): the *ideal gas law* (Eq.(II.27)) and the *quasi incompressible* equation (Eq.(II.30)). The focus was primarily on the second equation, as the study mainly dealt with water particles. These two state equations also influence how the density is initialized at the beginning of the simulation, depending on the desired physical behaviour. Three classical cases are studied. The first one consisting on a cube falling, the goal was to check if the uniformly accelerated movement was retrieved and it was the case. Moreover, an test in 2D was set up to verify that the code can be done in 2D too. Next, the hydrostatic pressure was correctly retrieved from the *cube of water at rest in a tank* case (Sec.III.2), a 2D simulation was done with a reasonably low number of particles, in order to compare the Euler and RK22 scheme, it concludes that the Euler scheme will be used during all the project. The last case was about a dam break, it measure the position of the wavefront over time in a dimensionless graph, and the numerical data are compered to experimental one.

Lastly, the performance improvement achieved through code parallelization using OpenMP was evaluated (Sec.II.11) based on the number of particles processed and the number of threads utilized. For particle counts below 10^5, it was observed that using more than 10 threads does enhance performance, but the gain is marginal compared to a simulation using 22 threads. All particle loops in the code were parallelized, as each particle operation is independent of the others.

Finally, microfluidic applications in this project were initially explored using the method proposed by Akinci et al. [1]. This approach incorporates both surface tension and adhesion

forces. Cohesion forces were modeled by introducing two sub-forces: the *cohesion forces* (Eq.(IV.1)) and a force that minimizes surface area (Eq.(IV.4)). These forces effectively replicate the attractive and repulsive interactions between fluid particles, and a specialized kernel was employed. A simulation was conducted, but it resulted in some non-physical outcomes due to sphere distortion. However, the pressure was calculated prior to the distortion and was found to be of the same order of magnitude as the theoretical value. It may be beneficial to increase the number of particles to improve pressure accuracy and achieve greater stability. Additionally, a basic simulation was run to observe the impact of the adhesion kernel on the system. Lastly, a more recent method proposed by Ordoubadi et al. for microfluidic applications, which adopts a more physical approach, was investigated but showed significant limitations due to the high computational time required and the lack of accurate results.

After many researches, it is clear that microfluidic applications can be derived from the SPH, but more elaborated algorithms need to be employed to hopefully retrieve some physical microfluidic variables.

After extensive research and numerous investigations, it has become evident that while microfluidic applications can indeed be derived from the Smoothed Particle Hydrodynamics (SPH) method, achieving accurate and reliable physical microfluidic variables requires the implementation of more sophisticated and elaborated algorithms. The basic SPH method, while versatile and powerful in many respects, often relies on numerical parameters that lack direct physical correlation, especially in the realm of microfluidic where precision and accurate representation of physical phenomena are paramount.

References

[1] Akinci, Nadir ; Akinci, Gizem ; Teschner, Matthias: Versatile surface tension and adhesion for SPH fluids. In: ACM Transactions on Graphics (TOG) 32 (2013), Nr. 6, S. 1–8

[2] Becker, Markus ; Teschner, Matthias: Weakly compressible SPH for free surface flows. In: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2007, S. 209–217

[3] F. Marcià, L. G. A. Souto-Iglesias: WCSPH viscosity diffusion processes in vortex flow. In: In:International Journal for Numerical Methods in Fluids (2011)

[4] Goffin, Louis: Development of a didactic SPH model, ULiège - Université de Liège, Diplomarbeit, June 2013

[5] Koshizuka, S. ; Oka, Y.: Moving-Particle Semi-Implicit MEthode for Fragmentation of Incommpressible Fluid. In: In: Nuclear and science and engineering 123 (1996), S. 421–434

[6] Lee, E-S ; Moulinec, Charles ; Xu, Rui ; Violeau, Damien ; Laurence, Dominique ; Stansby, Peter: Comparisons of weakly compressible and truly incompressible algorithms for the SPH mesh free particle method. In: Journal of computational Physics 227 (2008), Nr. 18, S. 8417–8436

[7] Liu, Gui-Rong ; Liu, Moubin B.: Smoothed particle hydrodynamics: a meshfree particle method. World scientific, 2003

[8] Monaghan, Joe J.: Smoothed particle hydrodynamics. In: In: Annual review of astronomy and astrophysics. Vol. 30 (A93-25826 09-90), p. 543-574. 30 (1992), S. 543–574

[9] Ordoubadi, M ; Yaghoubi, M ; Yeganehdoust, F: Surface tension simulation of free surface flows using smoothed particle hydrodynamics. In: Scientia Iranica 24 (2017), Nr. 4, S. 2019–2033