Travail de Fin d'Etudes : Development of a compressible flow solver for PFEM fluid simulations

Auteur : Février, Simon
Promoteur(s) : Ponthot, Jean-Philippe; Boman, Romain
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil physicien, à finalité approfondie
Année académique : 2019-2020
URI/URL : http://hdl.handle.net/2268.2/9010

LIÈGE université
Sciences Appliquées

ATFE9007-1 - Final thesis
(included an introduction to research methodology)

Development of a 3D compressible flow solver for PFEM fluid simulations

Simon Février

Abstract

A solver using the Particle Finite Element Method (PFEM) for incompressible and weakly-compressible Navier–Stokes equations is developed in order to simulate quasi incompressible 3D flows. The equations describing these types of flows are recalled in their analytical form. The PFEM method and the discretized version of the equations are described in 3D, using an implicit scheme for the incompressible equations and an explicit scheme for the compressible ones. The problem of 3D triangulation and alpha shape algorithm is solved by using the Computational Geometry Algorithms Library (CGAL). The results are compared to the solver developed during the PhD thesis of M.-L. Cerquaglia as well as analytical results when available. The solver, although able to simulate flows with fluids near incompressibility quite well, seems to have more difficulties simulating fluids which are too compressible.

Supervisors
Pr. Jean-Phillipe Ponthot
Dr. Romain Boman

Jury Member
Pr. Vincent Terrapon

Jury President
Pr. Benoît Vanderheyden

Master of Science (MSc) in Engineering Physics

Academic year 2019-2020

Table of contents

Notations

$\mathbb{E}$	A set.
q	A scalar (zero-th order tensor).
$\vec{v}$	A vector (first order tensor).
$\vec{v}_1 \cdot \vec{v}_2$	A standard dot product between two vector.
$\|\vec{v}\|$	The standard norm of a vector.
$\hat{n}$	A unit vector (symbolized by the hat).
$(\hat{e}_x, \hat{e}_y, \hat{e}_z)$	A Cartesian basis.
$(\hat{e}_r, \hat{e}_\theta, \hat{e}_z)$	A cylindrical basis.
$(\hat{e}_r, \hat{e}_\theta, \hat{e}_\phi)$	A spherical basis.
$\vec{\nabla}\vec{v}$	The gradient of $\vec{v}$.
$\vec{\nabla} \cdot \vec{v}$	The divergence of $\vec{v}$.
$\vec{\nabla} \times \vec{v}$	The curl of $\vec{v}$.
$\vec{\vec{T}}$	A tensor (second order tensor).
$\vec{\vec{I}}$	The identity tensor.
$\vec{\vec{T}} \cdot \vec{v}$	A standard dot product between a tensor and a vector.
$\vec{\vec{T}}_1 : \vec{\vec{T}}_2$	A standard double dot product between two tensors.
$\vec{\vec{T}}^{\mathrm{T}}$	The transpose tensor associated to this tensor.
$\mathrm{I}_{\vec{\vec{T}}}$, $\mathrm{II}_{\vec{\vec{T}}}$, $\mathrm{III}_{\vec{\vec{T}}}$	Tensor invariants $\mathrm{tr}\vec{\vec{T}}$, $\frac{1}{2}\left(\mathrm{tr}\vec{\vec{T}}^2 - (\mathrm{tr}\vec{\vec{T}})^2\right)$, and $\det\vec{\vec{T}}$.
$\mathrm{dev}\vec{\vec{T}}$	Deviatoric part of a tensor.
$\vec{\nabla} \cdot \vec{\vec{T}}$	The divergence of $\vec{\vec{T}}$.
$\mathrm{tr}\vec{\vec{T}}$	The trace of $\vec{\vec{T}}$.
$\det\vec{\vec{T}}$	The determinant of $\vec{\vec{T}}$.
$\mathbf{A}$	A matrix.
$\mathbf{v}$	A column-matrix.
$\mathbf{A}^{\mathsf{T}}$	The transpose of a matrix.
$\mathbf{v}^{\mathsf{T}}$	A row-matrix.
$[f]$	The matrix representing f in a cartesian basis.

Introduction

In addition to its existing solid mechanics solver, the MN2L (*Non-Linear Computational Mechanics*) lab of the University of Liège has gained access to free surface flow simulations and FSI (*Fluid Structure Interaction*) problems simulations thanks to the development of its PFEM (*Particle Finite Element Method*) incompressible Newtonian flow solver as well as the CUPyDO partitioned coupler. The PFEM is a mesh-based Langrangian method, giving it the ability to follow the free surface quite easily; while CUPyDO is an integrated Python environment able to couple multiple fluid or solid solvers for FSI problems. Both programs were developed in collaboration with the MTFC (*Multiphysic and Turbulent Flow Compilation*) lab at Uliège during a previous work by Marco-Lucio Cerquaglia (Cerquaglia [2019]) and David Thomas (Thomas et al. [2019]). The main problem that the lab currently encounters is the simulation of real 3D flows, as the current existing code is only able to simulate 2D flows. Moreover, simulating a 3D incompressible flow might turn out to be extremely computationally expensive due the need to use an implicit solver for such kind of equations, since such solvers do not scale well when increasing the number of unknowns, and memory consumption.

Cerquaglia's solver also uses an external executable (named *triangle*, Shewchuk [1996]) to perform the Delaunay triangulation of the remeshing step inherent to PFEM simulations. This means that some data has to be exchanged between the PFEM program and the external executable through the hard drive, which has an impact on performance. This external program is also unable to perform 3D Delaunay triangulations.

To remedy this situation, an explicit solver which should scale far much better with the size of the problems of interest will be developed in this master thesis. To do so, the incompressible Navier–Stokes equations will be replaced by the weakly-compressible Navier–Stokes equations, based on the work in Meduri [2019], but still with the goal of simulating incompressible flows. An incompressible implicit solver has also been developed alongside the explicit solver.

Moreover, both implicit and explicit solvers support 3D flows simulations. All the 2D and 3D Delaunay triangulations as well as alpha-shape algorithm are performed using CGAL (*Computational Geometry Algorithms Library*, The CGAL Project [2020]) such that the remeshing does not need to do input-output access on the drives anymore.

This report is divided in five parts. The first part will recall the continuum mechanics equations involved in Newtonian incompressible and weakly-compressible flows. The second part will describe the method used in this report and the particular problems encountered in 3D with spurious element generation. The third part will compare the implicit incompressible 2D solver developed in this work to Cerquaglia's solver. The fourth part will compare the incompressible implicit 2D solver and the weakly compressible explicit 2D solver developed in this work. The final part will then illustrate the 3D capabilities of the solvers developed in this master thesis.

Part I

Reminder of continuum mechanics

Contents

In this short part, the equations describing the behaviour of a linear incompressible or weakly-compressible fluid will be recalled, both in their strong and weak form. A basic knowledge of continuum mechanics as well as FEM (*Finite Element Method*) is assumed all along the manuscript.

1 Conservation equations

We consider the evolution of a volume of matter $V(t)$ through time as seen in Figure 1.

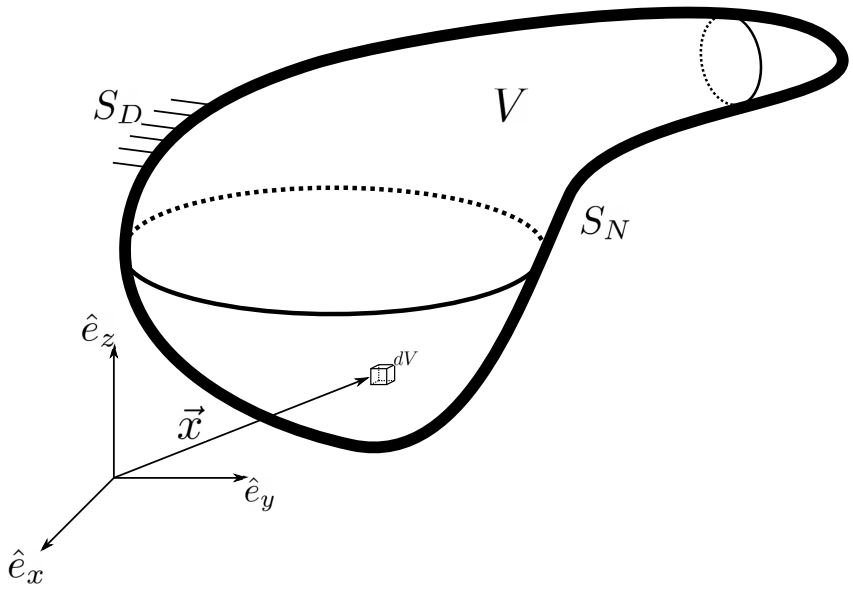


Figure 1: A continuum of matter.

The boundary of this fluid domain $\partial V(t)$ is assumed to be partitioned in $S_D(t)$ and $S_N(t)$, which are respectively the boundary on which are applied Dirichlet and Neumann boundary conditions, such that $S_D \cup S_N = \partial V$ and $S_D \cap S_N = \emptyset$.

Under continuum hypothesis, the equations describing the behaviour of a homogeneous material without electromagnetic effect are:

$$\begin{cases} \dfrac{d\rho}{dt} + \rho\vec{\nabla}\cdot\vec{v} = 0 \\ \rho\dfrac{d\vec{v}}{dt} = \vec{\nabla}\cdot\vec{\vec{\sigma}} + \rho\vec{b} \\ \vec{\vec{\sigma}} = \vec{\vec{\sigma}}^{\mathrm{T}} & ,\forall\vec{x}\in V(t)\,, \\ \rho\dfrac{du}{dt} = \vec{\vec{\sigma}}\colon\vec{\vec{D}} + \rho r - \vec{\nabla}\cdot\vec{q} \\ \mathcal{D} = \vec{\vec{\sigma}}\colon\vec{\vec{D}} - \rho\dfrac{du}{dt} + \rho T\dfrac{d\eta}{dt} - \dfrac{\vec{q}\cdot\vec{\nabla}T}{T} \geq 0 \end{cases} \tag{I.1}$$

where ρ is the density, $\vec{v}$ the velocity, $\vec{\vec{\sigma}}$ the Cauchy stress tensor, $\vec{b}$ the body forces, u the internal energy, T the absolute temperature, $\vec{\vec{D}}$ the strain rate tensor, r the heat source, $\vec{q}$ the heat flux and η the entropy at any point $\vec{x}(t)$ in the considered volume $V(t)$.

These equations represent the conservation of mass, linear momentum, angular momentum and energy respectively; while the inequation represents the second principle of thermodynamics where $\mathcal{D}$ is called the dissipation.

The initial and boundary conditions for this domain are considered to be of the form:

$$\begin{cases} \vec{v}(\vec{X}, t=0) = \vec{v}_0 \text{ and } \rho(\vec{X}, t=0) = \rho_0, & \forall\vec{X}\in V(0) \\ \vec{v}(\vec{x}, t) = \vec{v}_b, & \forall\vec{x}\in S_D(t)\ , \\ \vec{\vec{\sigma}}\cdot\hat{n} = \vec{t}_b, & \forall\vec{x}\in S_N(t) \end{cases} \tag{I.2}$$

where $\vec{X}$ denotes the reference configuration (taken here as the initial one).

Alternatively, the mass conservation equations can be written as:

$$\rho_0 = \rho J\,, \tag{I.3}$$

where ρ_0 is the the initial density field and J is the determinant of the deformation gradient tensor:

$$\vec{\vec{F}} = \frac{\partial\vec{x}}{\partial\vec{X}}\,, \tag{I.4}$$

such that $J = \det\vec{\vec{F}}$.

2 Navier–Stokes equations

Assuming the stress tensor depends only linearly on the strain rate tensor, the constitutive equations for a so-called Newtonian fluid reads:

$$\vec{\vec{\sigma}} = -p\vec{\vec{I}} + \lambda\vec{\nabla}\cdot\vec{v}\vec{\vec{I}} + 2\mu\vec{\vec{D}}\,, \tag{I.5}$$

where μ and λ are the so-called first and second viscosity of the considered fluid. If the Stokes hypothesis is assumed ($\lambda + \frac{2}{3}\mu = 0$), this constitutive equation becomes:

$$\vec{\vec{\sigma}} = -p\vec{\vec{I}} + 2\mu\ \mathrm{dev}\vec{\vec{D}} = -p\vec{\vec{I}} + \mu\left(\vec{\nabla}\vec{v} + \left(\vec{\nabla}\vec{v}\right)^{\mathrm{T}} - \frac{2}{3}\vec{\nabla}\cdot\vec{v}\vec{\vec{I}}\right)\,, \tag{I.6}$$

Further assuming that all properties of the fluid are independent of temperature, the system of equations from I.1&I.2 becomes:

$$\begin{cases} \dfrac{d\rho}{dt} + \rho\vec{\nabla}\cdot\vec{v} = 0, & \forall\vec{x}\in V(t) \\ \rho\dfrac{d\vec{v}}{dt} = \vec{\nabla}\cdot\vec{\vec{\sigma}} + \rho\vec{b} = -\vec{\nabla}p + \mu\vec{\nabla}^2\vec{v} + \dfrac{\mu}{3}\vec{\nabla}(\vec{\nabla}\cdot\vec{v}) + \rho\vec{b}, & \forall\vec{x}\in V(t) \\ \vec{v}(\vec{X}, t=0) = \vec{v}_0 \text{ and } \rho(\vec{X}, t=0) = \rho_0, & \forall\vec{X}\in V(0) \\ \vec{v}(\vec{x}, t) = \vec{v}_b, & \forall\vec{x}\in S_D(t) \\ \vec{\vec{\sigma}}\cdot\hat{n} = \vec{t}_b, & \forall\vec{x}\in S_N(t) \end{cases}. \tag{I.7}$$

The energy equation can be dropped, as thermal effects will not affect fluid flow since every parameter is assumed independent of temperature and that its evolution will not be considered in this work. The second principle of thermodynamics imposes that μ and $\lambda + \frac{2}{3}\mu$ are both positive quantities.

This system of four equations with five unknowns (v_x, v_y, v_z, ρ and p) is missing an equation to link the density and the pressure. Two solutions to this will studied hereafter: incompressible and weakly compressible fluid description.

2.1 Incompressible fluid

An incompressible fluid is characterised by

$$\frac{d\rho}{dt} = 0 \Rightarrow \vec{\nabla}\cdot\vec{v} = 0\,, \tag{I.8}$$

so that the system of equations from I.7 is reduced to

$$\begin{cases} \vec{\nabla}\cdot\vec{v} = 0, & \forall\vec{x}\in V(t) \\ \rho\dfrac{d\vec{v}}{dt} = \vec{\nabla}\cdot\vec{\vec{\sigma}} + \rho\vec{b} = -\vec{\nabla}p + \mu\vec{\nabla}^2\vec{v} + \dfrac{\mu}{3}\vec{\nabla}(\vec{\nabla}\cdot\vec{v}) + \rho\vec{b}, & \forall\vec{x}\in V(t) \\ \vec{v}(\vec{X}, t=0) = \vec{v}_0 \text{ and } \rho(\vec{X}, t=0) = \rho_0, & \forall\vec{X}\in V(0) \\ \vec{v}(\vec{x}, t) = \vec{v}_b, & \forall\vec{x}\in S_D(t) \\ \vec{\vec{\sigma}}\cdot\hat{n} = \vec{t}_b, & \forall\vec{x}\in S_N(t) \end{cases}. \tag{I.9}$$

One can see from the above system that there is no explicit equation for the time evolution of the pressure unknown.

2.2 Weakly-compressible fluid

In this case an equation of state will be used to link the density and the pressure. The bulk modulus of the fluid is defined as:

$$K = \rho \left.\frac{\partial p}{\partial \rho}\right|_T . \tag{I.10}$$

Assuming that the bulk modulus depends linearly of the pressure:

$$K = K_0 + K_0' p \,, \tag{I.11}$$

and that the temperature remains constant, one can derive:

$$\begin{aligned}
&\rho \frac{dp}{d\rho} = K_0 + K_0' p \\
&\Rightarrow \frac{1}{K_0 + K_0' p} \frac{dp}{d\rho} = \frac{1}{\rho} \\
&\Rightarrow \int_{\rho_0}^{\rho} \frac{1}{K_0 + K_0' p} \frac{dp}{d\rho^\star} \, d\rho^\star = \int_{\rho_0}^{\rho} \frac{1}{\rho^\star} \, d\rho^\star \\
&\Rightarrow \frac{1}{K_0'} \ln \left| \frac{K_0 + K_0' p(\rho)}{K_0 + K_0' p(\rho_0)} \right| = \ln \left| \frac{\rho}{\rho_0} \right| \\
&\Rightarrow \frac{K_0 + K_0' p(\rho)}{K_0 + K_0' p(\rho_0)} = \left(\frac{\rho}{\rho_0} \right)^{K_0'} .
\end{aligned} \tag{I.12}$$

Assuming $p(\rho_0) = 0$, the Tait-Murnagham state equation (Macdonald [1966]) reads:

$$p = \frac{K_0}{K_0'} \left[\left(\frac{\rho}{\rho_0} \right)^{K_0'} - 1 \right] , \tag{I.13}$$

and the system of equations from I.7 becomes:

$$\begin{cases}
\dfrac{d\rho}{dt} + \rho \vec{\nabla} \cdot \vec{v} = 0 & \forall \vec{x} \in V(t) \\
\rho \dfrac{d\vec{v}}{dt} = -\vec{\nabla} p + \mu \vec{\nabla}^2 \vec{v} + \dfrac{\mu}{3} \vec{\nabla} (\vec{\nabla} \cdot \vec{v}) + \rho \vec{b} & \forall \vec{x} \in V(t) \\
p = \dfrac{K_0}{K_0'} \left[\left(\dfrac{\rho}{\rho_0} \right)^{K_0'} - 1 \right] & \forall \vec{x} \in V(t) \\
\vec{v}(\vec{X}, t=0) = \vec{v}_0 \text{ and } \rho(\vec{X}, t=0) = \rho_0, & \forall \vec{X} \in V(0) \\
\vec{v}(\vec{x}, t) = \vec{v}_b, & \forall \vec{x} \in S_D(t) \\
\vec{\vec{\sigma}} \cdot \hat{n} = \vec{t}_b, & \forall \vec{x} \in S_N(t)
\end{cases} . \tag{I.14}$$

3 Galerkin formulation

The above derived systems of equations I.7, I.9 or I.14 are designated as *strong formulation* of the system of equations, because of the derivability requirements on the different

unknown fields. A so-called *weak formulation*, which has lower derivability requirements will be introduced hereafter.

The spaces $\mathbb{L}^2(\Omega)$ and $\mathbb{H}^1(\Omega)$ are first introduced (Adams and Fournier [2003]):

$$\mathbb{L}^2(\Omega) = \left\{ f : \Omega \to \mathbb{R} \colon \int_\Omega f^2 \, \mathrm{d}\Omega < +\infty \right\}, \tag{I.15}$$

$$\mathbb{H}^1(\Omega) = \left\{ f : \Omega \to \mathbb{R} \colon f \in \mathbb{L}^2(\Omega) \text{ and } \vec{\nabla} f \in \mathbb{L}^2(\Omega) \right\}. \tag{I.16}$$

The *space of admissible or test functions* for the velocity, for the velocity with homogeneous boundary conditions and for the pressure, denoted by $\mathbb{S}^v$, $\mathbb{S}_0^v$ and $\mathbb{S}^p$ respectively, are defined:

$$\begin{aligned} \mathbb{S}^v &= \left\{ \vec{w} \in \mathbb{H}^1(V(t)) \colon \vec{w} = \vec{w}_b \text{ on } S_D(t) \right\} \\ \mathbb{S}_0^v &= \left\{ \vec{w} \in \mathbb{H}^1(V(t)) \colon \vec{w} = 0 \text{ on } S_D(t) \right\} \\ \mathbb{S}^p &= \left\{ q \in \mathbb{L}^2(V(t)) \right\} \end{aligned} . \tag{I.17}$$

The weak form can then be derived by integrating the product of the strong form I.7 and a test function over the considered volume. The natural boundary conditions are also handled by the weak form of the momentum equation, while essential boundary conditions still need to be applied. By choosing $\vec{w} \in \mathbb{S}_0^v$ as the test function for the momentum equation and $q \in \mathbb{S}^p$ as the test function for the continuity equation, one can write:

$$\begin{cases} \displaystyle\int_{V_0} q \, \rho J \, dV_0 = \int_{V_0} q \, \rho_0 \, dV_0, & \forall \vec{x} \in V(t) \\ \displaystyle\int_{V(t)} \vec{w} \cdot \rho \frac{d\vec{v}}{dt} \, dV = \int_{V(t)} \vec{w} \cdot \left(\vec{\nabla} \cdot \vec{\vec{\sigma}} \right) dV + \int_{V(t)} \vec{w} \cdot \rho \vec{b} \, dV \\ \qquad\qquad - \displaystyle\int_{S_N(t)} \vec{w} \cdot (\vec{\vec{\sigma}} \cdot \hat{n} - \vec{t}_b) \, dS_N \quad , & \forall \vec{x} \in V(t) \\ \vec{v}(\vec{X}, t = 0) = \vec{v}_0 \text{ and } \rho(\vec{X}, t = 0) = \rho_0, & \forall \vec{X} \in V(0) \\ \vec{v}(\vec{x}, t) = \vec{v}_b, & \forall \vec{x} \in S_D(t) \end{cases} \tag{I.18}$$

where J is the determinant of the deformation gradient tensor. This finally gives (since $\rho J \, dV_0 = \rho \, dV$ and $\vec{w} = 0$ on $S_D(t)$):

$$\begin{cases} \displaystyle\int_{V(t)} q \, \rho \, dV = \int_{V_0} q \, \rho_0 \, dV_0, & \forall \vec{x} \in V(t) \\ \displaystyle\int_{V(t)} \vec{w} \cdot \rho \frac{d\vec{v}}{dt} \, dV = - \int_{V(t)} \vec{\nabla} \vec{w} \colon \vec{\vec{\sigma}} \, dV + \int_{V(t)} \vec{w} \cdot \rho \vec{b} \, dV \\ \qquad\qquad + \displaystyle\int_{S_N(t)} \vec{w} \cdot \vec{t}_b \, dS_N \quad , & \forall \vec{x} \in V(t) \quad . \\ \vec{v}(\vec{X}, t = 0) = \vec{v}_0 \text{ and } \rho(\vec{X}, t = 0) = \rho_0, & \forall \vec{X} \in V(0) \\ \vec{v}(\vec{x}, t) = \vec{v}_b, & \forall \vec{x} \in S_D(t) \end{cases} \tag{I.19}$$

The weak form of the continuity equation can also be written differently if the continuity equation with explicit time derivation is used:

$$\int_{V(t)} q \left(\frac{d\rho}{dt} + \vec{\nabla} \cdot \vec{v} \right) dV = 0 \quad \forall \vec{x} \in V(t) \,. \tag{I.20}$$

One can see from equation I.19 that the spatial derivability requirements on the stress tensor have been lowered, hence the name weak form. While a solution to the strong form is a solution to the weak form, the inverse is not always true. Such a solution is called a *generalized solution*.

Now that the required equations have been introduced, the numerical method used in this master thesis, as well as the discretized equations will be described in the next part.

Part II
The Particle Finite Element Method

Contents

In this part, the particle finite element method is introduced. After a brief overview of the method, a discussion about the remeshing procedure is made. The numerical scheme is then derived and finally the actual implementation is described. again it is assumed that the reader has some knowledge about the FEM and discretization methods.

1 Overview of the method

As mentioned in the introduction of this report, this work focuses on numerically solving problems involving free surface flows. Being able to track the free surface is then of particular importance.

Various methods exist in computational fluid dynamics nowadays. They can be separated in two big categories, based on the employed formalism:

- Eulerian methods, which use a fixed grid/mesh where the fluid is moving through it.
- Lagrangian methods, where the grid/mesh/nodes are attached to the fluid in motion.

Some examples of methods to solve free surface flows will be described hereafter.

1.1 Eulerian methods

Examples of free surface tracking Eulerian methods are the Volume Of Fluid method and the Level Set method.

The Volume Of Fluid method (Noh and Woodward [1976]) solves multiphase flows by tracking the content of each cell using the volume fraction of each phase in the cell. The main problem of this method is that the free surface is not sharp and can be quite spread out in space.

The Level Set method (Osher and Sethian [1988]; Sussman et al. [1994]) uses a spatial set function, which has positive sign on a certain side and a negative sign on the other side of the free surface, and is equal to zero at the free surface . While the free surface is sharp, the method is more difficult to implement and does not conserve mass really well like the Volume of Fluid method.

The main problem the necessity to discretize each phase of the multiphase flow, which can be very expensive and is not useful at all in some cases, for example when modelling the behaviour of waves in which the surrounding flow of air has little impact. Those methods also contains a convective term in their discretized equations and are complex to implement.

1.2 Lagrangian methods

An example of Lagrangian method is the SPH method (Gingold and Monaghan [1977]) which discretizes the fluid as particles, without a mesh. The properties of the fluid at a certain position are evaluated using a kernel function of a certain kernel length which allows a weighted average of the properties of the nearby particles. The particles are attached to the fluid in motion, but imposing boundary conditions might be extremely difficult. On the other, this method is simple to implement and parallelize.

1.3 The PFEM

Correctly representing the boundaries of the fluid is really important in all problems involving surface tension or solid-fluid interaction, in order for example to correctly impose boundary conditions or compute the curvature. The idea is then to use a standard FEM mesh to discretize the fluid. The mesh is attached to the fluid in motion. At each time step, a FEM-like computation is performed based on this mesh to compute the time-evolution of the fluid. As the mesh deforms with the fluid, a new mesh is sometimes needed in order to ensure accurate computation since the previous one might be too deformed. For this remeshing procedure to be as fast as possible, linear shapes functions are used and the fluid can thus be represented by a cloud of nodes which stores all of the fluid properties at a certain time, nodes which are the basis of the mesh generation. Using a FEM mesh allows the tracking of a sharp free surface. This is the basics of the *Particular Finite Element Method* (PFEM), first described in Idelsohn et al. [2004].

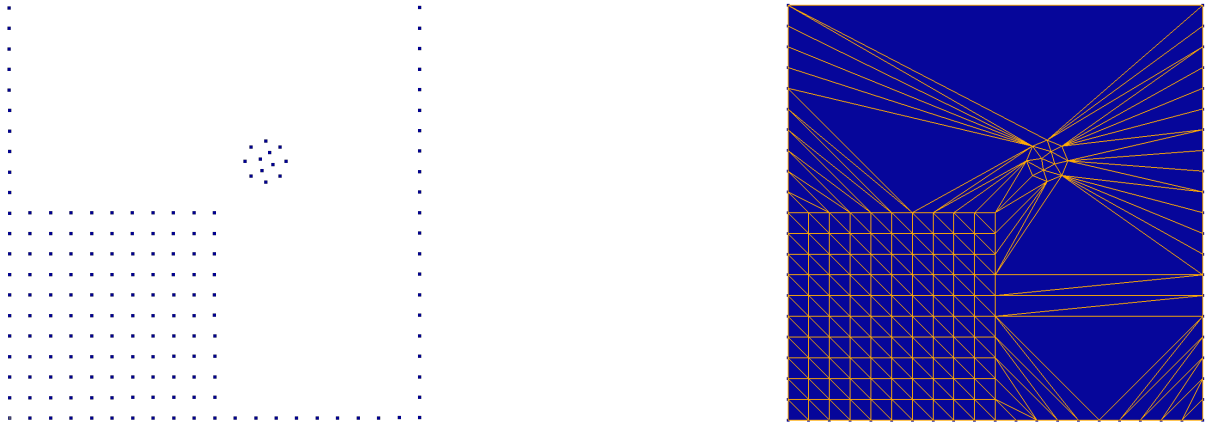
2 Remeshing Procedure

A mesh composed of elements (triangles in 2D or tetrahedrons in 3D) needs to be generated from the cloud of nodes in order to perform the FEM computations. The generation of the initial set of points is performed by the `Gmsh` library Geuzaine and Remacle [2009], which has been chosen for its simplicity and the knowledge of this library in a previous project by the author of this report, while the generation of the mesh during the solver computation is performed by the *Computational Geometry Algorithms Library* (hereafter denominated CGAL (The CGAL Project [2020])), because this library contains everything needed regarding the triangulation and extra elements deletion that are needed for the remeshing in the PFEM.

2.1 Delaunay triangulation

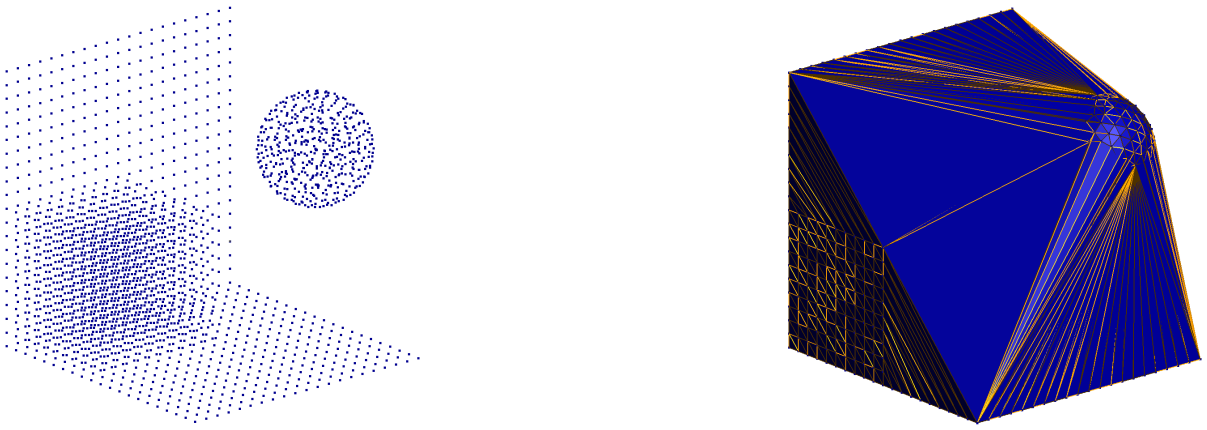
A Delaunay triangulation (Delaunay [1934]) is first performed on the set of points, which has the remarkable defining property that the circumcirle/circumsphere of any element does not contain any other node in the 2D/3D set. This triangulation has also the property, in 2D, of maximizing the minimum angle of in all elements with respect to other triangulations (Meduri [2019]), which is indeed great for the computing quality of the mesh.

An example of such a triangulation can be seen in Figure 2 in 2D and in Figure 3 in 3D.



(a) Before triangulation. (b) After triangulation.

Figure 2: 2D Delaunay triangulation of a cloud of nodes using CGAL. The square box is of 10 units length and the cube is of 5 units length. The disk is of 1 unit diameter and is placed at $\sqrt{2}$ unit length from the cube top right corner along the square diagonal.



(a) Before triangulation. (b) After triangulation.

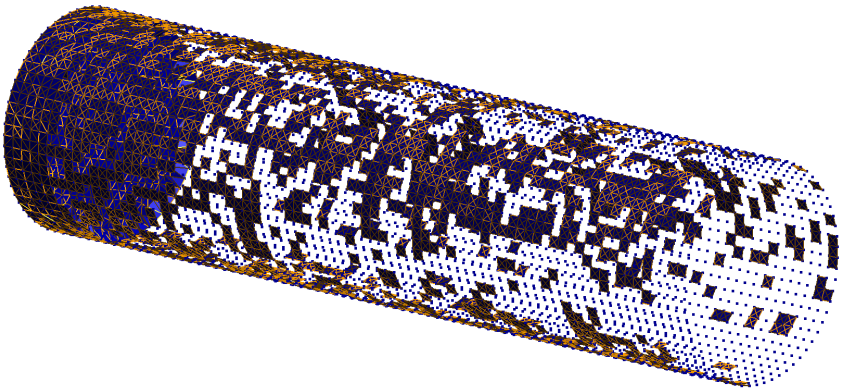
Figure 3: 3D Delaunay triangulation of a cloud of nodes using CGAL. The square planes are of 10 units length and the cube is of 5 units length. The sphere is of 4 unit diameter and is placed at $2.5\sqrt{3}$ unit length from the cube corner along the cube diagonal.

As it can be seen on both figures, both the square and the circle in 2D and both the cube and the sphere in 3D are meshed, but also all the space around. Triangulating the set of points is thus insufficient to correctly identify the fluid domains. Some triangles/tetrahedrons need to be discarded. This operation is performed using the alpha-shape algorithm (Edelsbrunner et al. [1983]) (using CGAL).

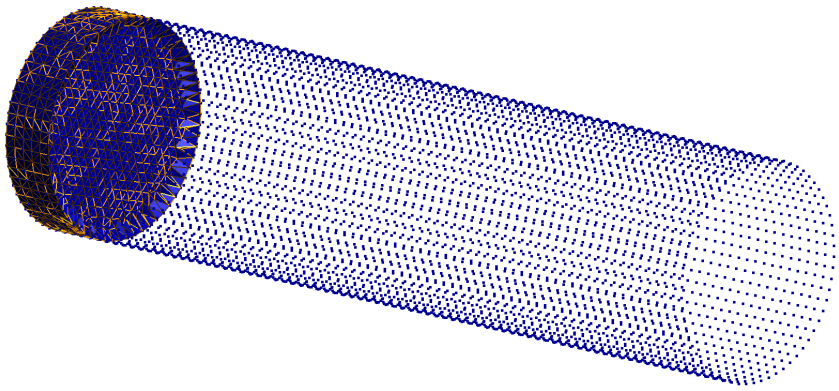
Since it will be needed in various part of this report, the *characteristic size* (h_{char}) of the elements in a mesh is defined as the initial spacing between the nodes, which is the value used by `Gmsh` to generated the initial cloud of nodes.

In order not to surcharge the figures in the next section, we will anticipate a discussion on 3D mesh generation here. As it can be seen in figure 4, near-zero-volume tetrahedrons

are generated by CGAL in some 3D problems. Those elements (called *slivers*) are simply deleted from the mesh, by deleting all elements whose volume are inferior to $10^{-4} \times h_{char}^3$. The impact on the "real" fluid mesh should be negligible, as multiple strategies are used alongside the alpha-shape in order to conserve relatively constant element size through the simulation as it will be seen later in this part. This effect only appears in 3D.



(a) Mesh generated by CGAL without slivers deletion. As it can be seen, non fluid elements are generated and looks like planar tetrahedrons.



(b) Mesh generated by CGAL with sliver deletion. The problem is now correctly meshed.

Figure 4: Mesh generation for a cylinder of fluid lying at the entrance of a cylinder pipe.

2.2 Alpha-shape algorithm

The alpha-shape algorithm was first introduced in Edelsbrunner et al. [1983]. This technique allows to reconstruct the overall shape formed by a set of points, and is performed in this work thanks to CGAL (Edelsbrunner and Mücke [1994]; Da [2020]; Da et al. [2020]).

To understand how this algorithm works, some mathematical definitions are needed (Spanier [1981]). A simplex can be seen as the generalization of the concept of triangle to any dimension:

- a 0-simplex is a point.

- a 1-simplex is a line segment.
- a 2-simplex is a triangle.
- a 3-simplex is a tetrahedron.

A k-simplex is composed of $k+1$ points. A face of a simplex is the convex hull of any not empty subset of those $k+1$ points. For a triangle (2-simplex), the faces are the 3 line segments and the 3 points forming the triangle.

A simplicial complex $\mathcal{K}$ is a set of simplicies such that:

- Every face of a simplex in $\mathcal{K}$ is also in $\mathcal{K}$.
- For every pair of simplicies s_1 and s_2 in $\mathcal{K}$, if they have a non empty intersection $i = s_1 \cap s_2 \neq \emptyset$, this intersection is a face of s_1 and s_2.

A simplex is said α-exposed if there is an open disk/open sphere of radius $\sqrt{\alpha}$ through the vertices of the simplex that does not contain any other point of the set. For a triangle in 2D, this open disk is unique as there is only one circle that passes by 3 points in 2D, such that there is only one α value at which a triangle could be α-exposed: $\alpha = r^2_{circumcircle}$. For a line segment in 2D, the α value is generally comprised between two limits. An exemple of the α-value of a segment and a triangle in 2D can be seen in figure 5.

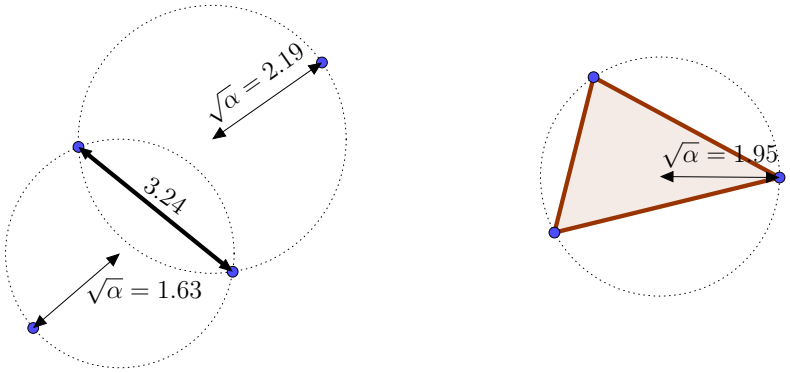


Figure 5: Minimum and maximum α value of a 1-simplex (on the left) and a 2-simplex (on the right) for which they remains α-exposed in 2D.

An α_{char}-complex is a simplicial complex, subcomplex of a triangulation, containing all α-exposed k-simplicies ($k \in [0; \dim(\text{set})] \subset \mathbb{N}$) of α-value smaller than α_{char}. The alpha-shape is then the interior of the α_{char}-complex. In summary, a triangle is removed from the mesh if:

$$r_{circumcirle} > \sqrt{\alpha_{char}}\,. \tag{II.1}$$

The version of the alpha-shape algorithm used in Cerquaglia [2019] and Meduri [2019] is similar to the one used in CGAL, but the *alpha* value is defined differently. In Cerquaglia

[2019]; Meduri [2019], a triangle is removed from the mesh if its circumcircle radius is greater than a certain constant:

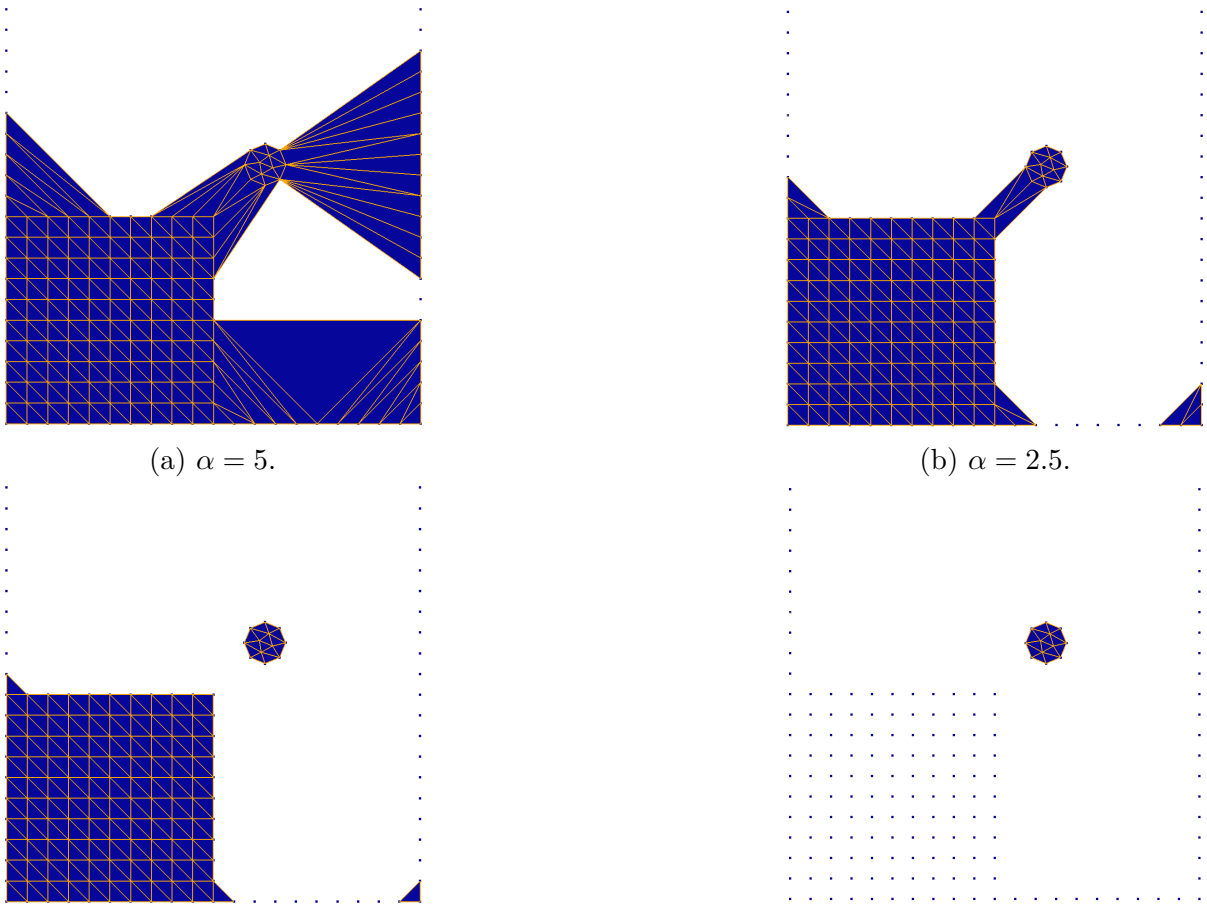
$$r_{circumcircle} > \alpha h \,. \tag{II.2}$$

The definition of h in Cerquaglia [2019]; Meduri [2019] is the average of all the h_e, h_e being the minimal distance between two points in the element e. The two definitions can be reconciled by:

$$\alpha_{CGAL} = (\alpha_{C,M} h_{char})^2 \,. \tag{II.3}$$

In this work, the parameter α that the user inputs to the solvers is indeed the parameter $\alpha_{C,M}$ in the above equation, in order to use an α value relatively independent of the mesh density. The conversion to the CGAL definition is obviously done inside the program. Please note than even with this, the alpha-shape algorithm will remain different because of the differences between h_{char}, given by the user in the present work, and h, compute as described above by Cerquaglia [2019] and Meduri [2019].

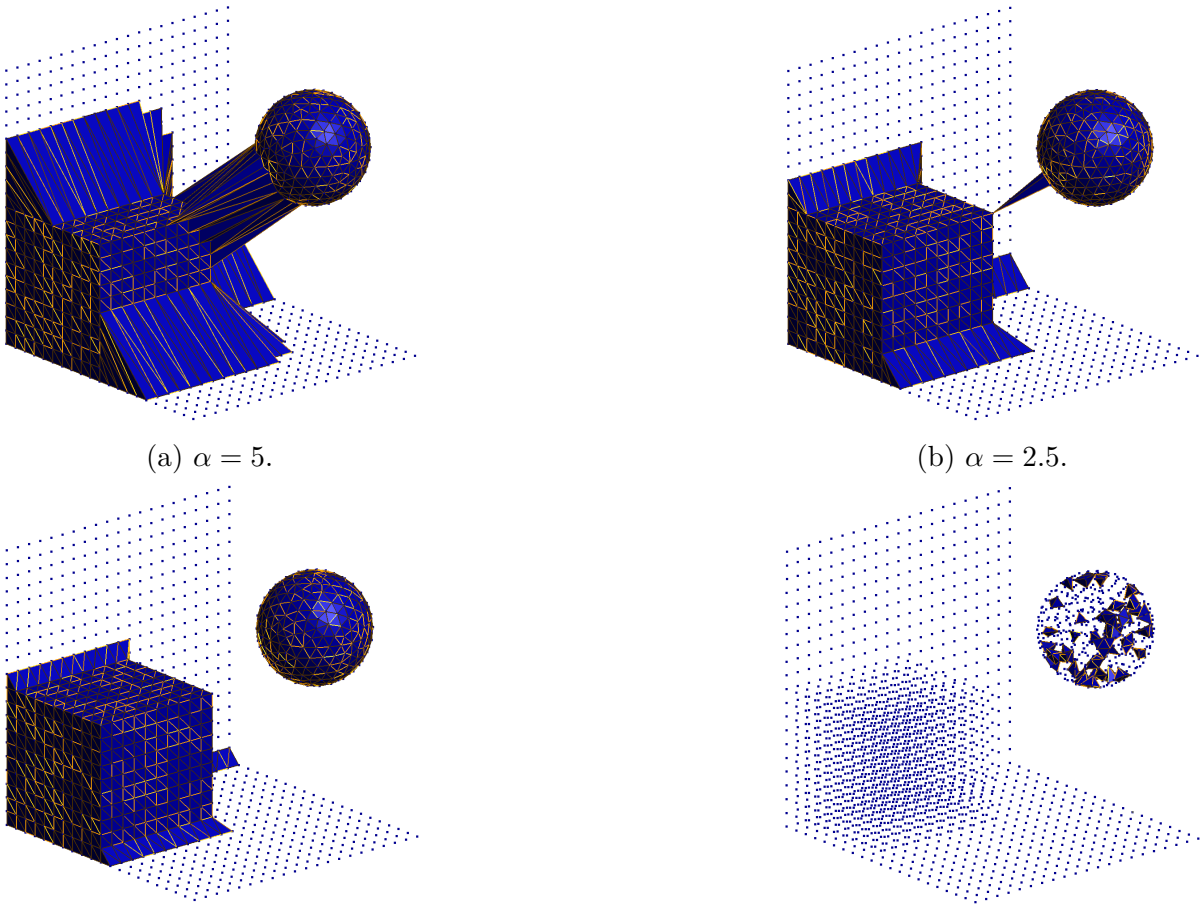
The result of the alpha shape algorithm on the Delaunay triangulation from Figure 2 and Figure 3 can be seen in Figure 6 in 2D and Figure 7 in 3D for various $\alpha_{C,M}$ values.



(a) $\alpha = 5$. (b) $\alpha = 2.5$.

(c) $\alpha = 1.25$. (d) $\alpha = 0.625$.

Figure 6: Alpha-shape algorithm results on a Delaunay triangulation see Figure 2 for the initial mesh) for various $\alpha_{C,M}$ values using CGAL in 2D ($h_{char} = 1$ unit of length).



(a) $\alpha = 5$.

(b) $\alpha = 2.5$.

(c) $\alpha = 1.25$.

(d) $\alpha = 0.625$.

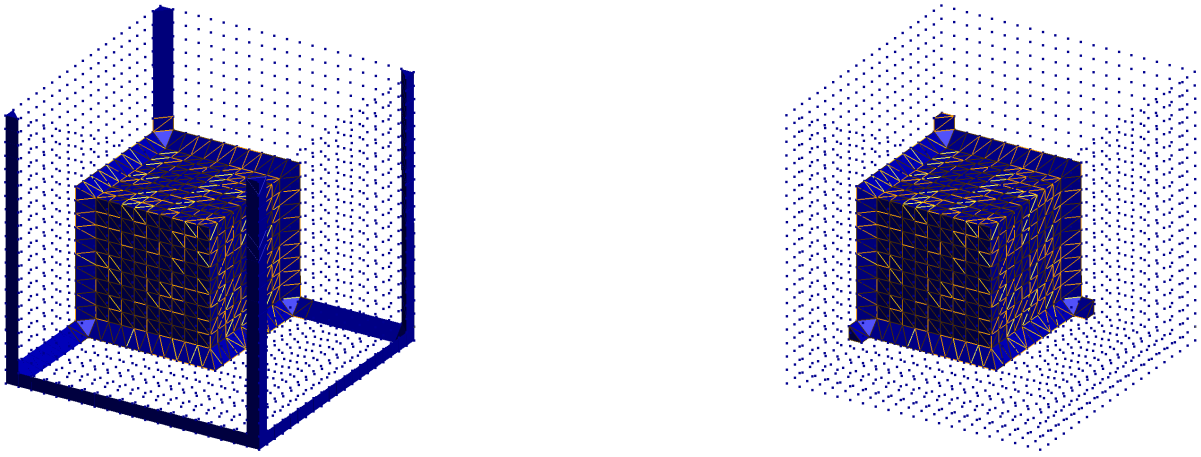
Figure 7: Alpha-shape algorithm results on a Delaunay triangulation (see Figure 3 for the initial mesh) for various $\alpha_{C,M}$ values using CGAL in 3D ($h_{char} = 1$ unit of length).

As a conclusion, the alpha value should be large enough to capture all of the actual fluid elements, but small enough to discard non-fluid elements. Alpha values between 1.1 an 1.3 generally appear to be a good choice.

However, extra fluid elements are still present in those cases in the box corners (see Figure 6c and Figure 7c), as there is no way for CGAL to differentiate them from an actual same-sized fluid element. Those kind of superfluous elements have two characteristics:

- They are only composed of boundary nodes.
- The nodes of those elements do not have neighbour fluid nodes.

Thanks to those characteristics, those kind of spurious elements can be deleted quite easily. The final results can be seen in Figure 8 in 3D.



(a) Before extra elements deletion.

(b) After extra elements deletion.

Figure 8: Exemple of extra elements deletion for a fluid cube in a box in 3D.

It can be noted however that the cube shape in Figure 8 is still not perfectly identified. There is indeed no way for CGAL to differentiate those elements from the other "real" elements. However, the size of those elements will decrease as the number of nodes is increased, such that the cube will be better and better approximated as the number of nodes is increased.

3 Other mesh-improvement algorithms

The density of nodes should be controlled while the simulation in running in order to maintain good mesh quality, and prevent the apparition of bad fluid behaviour in the simulation conditions. To do so, one could consider adding or removing nodes from the simulation when it is needed. Two examples of why adding or removing nodes could be useful are presented hereafter.

It could happen that some nodes become too close from each others (for example when nodes approach a wall boundary), which can lean to higher numerical errors and physical errors (e.g. nodes crossing a boundary). The only solution is to delete some of the nodes which are too close from each other. This deletion unfortunately removes information from the solution.

The opposite problem could appear if at some part of the mesh some elements became too big, such that the alpha-shape algorithm introduces holes in the mesh were there should not be any. To prevent that, a solution is to add nodes at the center of those too big elements.

Two strategies are thus implemented:

- if some nodes become too close to each other, one of the nodes is deleted if the distance between the two nodes is inferior to a user-defined parameter times the mesh characteristic size (h_{char}). This parameter is denoted as γ.

- if an element becomes too big, a node is added at the center of the element if the element volume is greater than a user-defined parameter times the mesh characteristic element volume (h^3_{char}). This parameter is denoted as ω.

In order to avoid simulating fluid that goes out of the spatial region of interest of a simulation, a bounding box mechanism is implemented such that fluid nodes which go out of that box are deleted.

4 Frequency and performance of remeshing

The overall remeshing algorithm used in this work is not really optimized, as the bounding box checking, nodes removing and adding part of the algorithm do not feed back their modification of the cloud of nodes to the list of elements, such that a triangulation plus an alpha-shape algorithm are performed in between each of these step, which is rather inefficient and could be optimized by simply asking to those parts of the remeshing to also check the elements when they delete or add new nodes.

For the implicit incompressible solver developed in this work, the remeshing is performed at each time step. As it will be seen in section III.6, the time taken by the remeshing is really small in comparison to the time taken to solve the equations, so that this is not currently a problem.

For the explicit compressible solver developed in this work, the time taken by the remeshing is in the same order of magnitude as the time taken to solve the equation, as it will be seen in section in section IV.6. Moreover, since explicit time steps are generally really small, it would be a waste of computing power to remesh at every time step like for the incompressible solver, because the mesh does not change that much between two close time steps. That is why the remeshing is only performed at a certain frequency, which can be set in the solver using the "maximum Δt" user-defined variable, which also represents an upper bound for the time step in both solvers when adaptive time step is used. It can finally be noted that in 3D the time to perform the remeshing is greater that the time to actually solve the equations, as it will be seen in V.3. The remeshing takes here a big part of the computational time, even if it is not triggered at every time step.

5 Space Discretization

Now that a mesh is available, the discretization of the system of equations I.19 can be described. This will be done in 3D, the 2D case being similar.

We consider that the mesh represents an approximation of the actual fluid domain:

$$V_h(t) = \bigcup_{e=1}^{N_e} V_h^e(t), \quad \lim_{h\to 0} V_h(t) = V(t)\,, \tag{II.4}$$

where V_h^e is the volume of the element e, and h is a characteristic element size. The unknown fields are then approximated by:

$$\begin{aligned} \vec{v}(\vec{x},t) &\simeq \mathbf{N}^v(\vec{x})\mathbf{v}(t) \\ p(\vec{x},t) &\simeq \mathbf{N}^p(\vec{x})\mathbf{p}(t)\,, \\ \rho(\vec{x},t) &\simeq \mathbf{N}^p(\vec{x})\boldsymbol{\rho}(t) \end{aligned} \tag{II.5}$$

where $\mathbf{N}^v$ and $\mathbf{N}^p$ are the linear shape function matrices for the velocity and the pressure/density ; and $\mathbf{v}$, $\mathbf{p}$ and $\boldsymbol{\rho}$ the vectors of nodal unknowns:

$$\begin{aligned} \mathbf{N}^v &= \begin{bmatrix} N_1^v & N_2^v & N_3^v & N_4^v & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & N_1^v & N_2^v & N_3^v & N_4^v & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & N_1^v & N_2^v & N_3^v & N_4^v \end{bmatrix} \\ \mathbf{N}^p &= \begin{bmatrix} N_1^p & N_2^p & N_3^p & N_4^p \end{bmatrix} \\ \mathbf{v} &= \begin{pmatrix} u_1 & u_2 & u_3 & u_4 & v_1 & v_2 & v_3 & v_4 & w_1 & w_2 & w_3 & w_4 \end{pmatrix}^\mathsf{T} \\ \boldsymbol{\rho} &= \begin{pmatrix} \rho_1 & \rho_2 & \rho_3 & \rho_4 \end{pmatrix}^\mathsf{T} \\ \mathbf{p} &= \begin{pmatrix} p_1 & p_2 & p_3 & p_4 \end{pmatrix}^\mathsf{T}\,. \end{aligned} \tag{II.6}$$

The gradient of shape function matrices for the velocity and the pressure/density are

defined as:

$$\mathbf{B}^v = \begin{bmatrix} \frac{\partial N_1^v}{\partial x} & \frac{\partial N_2^v}{\partial x} & \frac{\partial N_3^v}{\partial x} & \frac{\partial N_4^v}{\partial x} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial N_1^v}{\partial y} & \frac{\partial N_2^v}{\partial y} & \frac{\partial N_3^v}{\partial y} & \frac{\partial N_4^v}{\partial y} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial N_1^v}{\partial z} & \frac{\partial N_2^v}{\partial z} & \frac{\partial N_3^v}{\partial z} & \frac{\partial N_4^v}{\partial z} \\ \frac{\partial N_1^v}{\partial y} & \frac{\partial N_2^v}{\partial y} & \frac{\partial N_3^v}{\partial y} & \frac{\partial N_4^v}{\partial y} & \frac{\partial N_1^v}{\partial x} & \frac{\partial N_2^v}{\partial x} & \frac{\partial N_3^v}{\partial x} & \frac{\partial N_4^v}{\partial x} & 0 & 0 & 0 & 0 \\ \frac{\partial N_1^v}{\partial z} & \frac{\partial N_2^v}{\partial z} & \frac{\partial N_3^v}{\partial z} & \frac{\partial N_4^v}{\partial z} & 0 & 0 & 0 & 0 & \frac{\partial N_1^v}{\partial x} & \frac{\partial N_2^v}{\partial x} & \frac{\partial N_3^v}{\partial x} & \frac{\partial N_4^v}{\partial x} \\ 0 & 0 & 0 & 0 & \frac{\partial N_1^v}{\partial z} & \frac{\partial N_2^v}{\partial z} & \frac{\partial N_3^v}{\partial z} & \frac{\partial N_4^v}{\partial z} & \frac{\partial N_1^v}{\partial y} & \frac{\partial N_2^v}{\partial y} & \frac{\partial N_3^v}{\partial y} & \frac{\partial N_4^v}{\partial y} \end{bmatrix}$$

$$\mathbf{B}^p = \begin{bmatrix} \frac{\partial N_1^v}{\partial x} & \frac{\partial N_2^v}{\partial x} & \frac{\partial N_3^v}{\partial x} & \frac{\partial N_4^v}{\partial x} \\ \frac{\partial N_1^v}{\partial y} & \frac{\partial N_2^v}{\partial y} & \frac{\partial N_3^v}{\partial y} & \frac{\partial N_4^v}{\partial y} \\ \frac{\partial N_1^v}{\partial z} & \frac{\partial N_2^v}{\partial z} & \frac{\partial N_3^v}{\partial z} & \frac{\partial N_4^v}{\partial z} \end{bmatrix}, \tag{II.7}$$

such that:

$$\begin{aligned} \mathbf{B}^v\mathbf{v} &\simeq \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial y} & \frac{\partial w}{\partial z} & \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) & \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right) & \left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) \end{pmatrix}^\mathsf{T} \\ \mathbf{B}^p\mathbf{p} &\simeq \begin{pmatrix} \frac{\partial p}{\partial x} & \frac{\partial p}{\partial y} & \frac{\partial p}{\partial z} \end{pmatrix}^\mathsf{T} \\ \mathbf{B}^p\boldsymbol{\rho} &\simeq \begin{pmatrix} \frac{\partial \rho}{\partial x} & \frac{\partial \rho}{\partial y} & \frac{\partial \rho}{\partial z} \end{pmatrix}^\mathsf{T}. \end{aligned} \tag{II.8}$$

The test functions pair $(q, \vec{w})$ is approximated the same way as the unknown field. Putting those linear approximations into I.19 and developing leads to the discretized system. The results are summarized hereafter.

5.1 Weakly Compressible fluid

The system of equation I.19 is discretized as:

$$\begin{aligned} \mathbf{M}\frac{d\mathbf{v}}{dt} &= \mathbf{D}^\mathsf{T}\mathbf{p} - \mathbf{K}\mathbf{v} + \mathbf{f}_{\text{ext}} \\ \mathbf{M}^\rho\frac{d\boldsymbol{\rho}}{dt} + \mathbf{D}^\rho\mathbf{v} &= 0 \quad \text{or} \quad \mathbf{M}^\rho\boldsymbol{\rho} = \mathbf{M}_0^\rho\boldsymbol{\rho}_0 \end{aligned}, \tag{II.9}$$

where:

$$\begin{aligned}
\mathbf{M} &= \int_{V_h(t)} \rho^\star \mathbf{N}^{v\mathsf{T}} \mathbf{N}^v \, dV_h \qquad \mathbf{D} = \int_{V_h(t)} \mathbf{N}^{p\mathsf{T}} \mathbf{m}^\mathsf{T} \mathbf{B}^v \, dV_h \\
\mathbf{K} &= \int_{V_h(t)} \mathbf{B}^{v\mathsf{T}} \mathbf{d}_{\text{dev}} \mathbf{B}^v \, dV_h \qquad \mathbf{f}_{\text{ext}} = \int_{V_h(t)} \rho^\star \mathbf{N}^{v\mathsf{T}} \left[\vec{b}\right] \, dV_h + \int_{S_{Nh}(t)} \mathbf{N}^{v\mathsf{T}} \left[\vec{t_b}\right] \, dS_{Nh} \\
\mathbf{M}^\rho &= \int_{V_h(t)} \mathbf{N}^{\rho\mathsf{T}} \mathbf{N}^\rho \, dV_h \qquad \mathbf{D}^\rho = \int_{V_h(t)} \rho^\star \mathbf{N}^{\rho\mathsf{T}} \mathbf{m}^\mathsf{T} \mathbf{B}^v \, dV_h \\
\mathbf{m}^\mathsf{T} &= \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{d}_{\text{dev}} = \begin{bmatrix} 4/3 & -2/3 & -2/3 & 0 & 0 & 0 \\ -2/3 & 4/3 & -2/3 & 0 & 0 & 0 \\ -2/3 & -2/3 & 4/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \quad , \tag{II.10}$$

where $\rho^\star = \mathbf{N}^\rho \boldsymbol{\rho}$. The index 0 below the matrix $\mathbf{M}^\rho$ and the vector $\boldsymbol{\rho}$ indicates they are computed on a reference configuration. As expected two versions of the continuity equation are available. The second one has the advantage that it does not contains any time derivation. However some problems cannot be simulated using this equation as it will be seen in part IV.

The equation of state still needs to be added:

$$[\mathbf{p}]_i = \frac{K_0}{K_0'} \left[\left(\frac{[\boldsymbol{\rho}]_i}{\rho_0} \right)^{K_0'} - 1 \right], \quad 1 \leq i \leq N_{\text{nodes}} \tag{II.11}$$

5.2 Incompressible fluid

The system of equations I.19 is discretized as:

$$\begin{aligned}
&\mathbf{M} \frac{d\mathbf{v}}{dt} = \mathbf{D}^\mathsf{T} \mathbf{p} - \mathbf{K}\mathbf{v} + \mathbf{f}_{\text{ext}} \\
&\mathbf{D}\mathbf{v} = 0
\end{aligned} \quad , \tag{II.12}$$

The matrices are all defined the same way as in II.9 except the $\mathbf{d}_{\text{dev}}$ matrix which can be simplified as:

$$\mathbf{d}_{\text{dev}} = \text{diag}(2,\ 2,\ 2,\ 1,\ 1,\ 1) \tag{II.13}$$

and $\rho^\star = \rho$ is a constant.

There is no explicit time evolution equation for the pressure in II.12. The incompressible Navier–Stokes equations form a saddle point problem ([Donéa and Huerta, 2003]) in which the pressure acts as a Lagrange multiplier of the incompressibility equation. The system of equations II.12 does not satisfies the Ladyzhenskaya-Babuška-Brezzi (LBB) condition (Donéa and Huerta [2003]; Sani et al. [1981a,b]) when linear elements for the velocity and pressure are used. A stability procedure, such as the PSPG described hereafter, is thus applied to the equation.

In the Pressure Stabilizing Petrov Galerkin (PSPG) procedure (Hughes et al. [1986]), the pair of test functions $(q, \vec{w})$ is replaced by $(q, \vec{w} + \tau_{\text{PSPG}} \vec{\nabla} q)$. At the end, this acts by

adding a term proportional to the momentum equation (which is equal to zero for the exact solution) to the continuity equation. The final equation for the mass conservation reads:

$$\mathbf{C}\frac{d\mathbf{v}}{dt} - \mathbf{D}\mathbf{v} + \mathbf{L}\mathbf{p} = \mathbf{h}\,, \tag{II.14}$$

with:

$$\begin{aligned} \mathbf{C} &= \int_{V_h(t)} \tau_{\mathrm{PSPG}} \mathbf{B}^{p\mathsf{T}} \mathbf{N}^v \, dV_h \qquad \mathbf{L} = \int_{V_h(t)} \tau_{\mathrm{PSPG}} \mathbf{B}^{p\mathsf{T}} \mathbf{B}^p \, dV_h \\ \mathbf{h} &= \int_{V_h(t)} \tau_{\mathrm{PSPG}} \mathbf{B}^{p\mathsf{T}} \left[\vec{b}\right] \, dV_h \end{aligned}\,, \tag{II.15}$$

the value of τ_{PSPG} being computed for all elements as (Tezduyar [1991]):

$$\tau_{PSPG}^{(e)} = \sqrt{\frac{1}{\left(\frac{2}{\Delta t}\right)^2 + \left(\frac{2\left\|\vec{U}^{(e)}\right\|}{\tilde{h}^{(e)}}\right)^2 + \left(\frac{4\mu}{(\tilde{h}^{(e)})^2\rho}\right)^2}}\,, \tag{II.16}$$

where Δt is the time step, $\tilde{h}^{(e)}$ is an element characteristic size computed as the circumcircle diameter of the element and $\left\|\vec{U}^{(e)}\right\|$ is the norm of the fluid velocity in the element.

6 Time integration

Having the space discretization done, the time integration methods for both types of fluids can now be derived.

6.1 Weakly Compressible fluid

As every unknown can be computed explicitly in II.9, an explicit time integration scheme will be employed in this case. More precisely, a central difference scheme will be used (Meduri [2019]).

The mass matrices will be lumped (i.e. for each row of the matrix, the elements are summed up on the diagonal) as soon as they are in the left hand side of the equations. No global matrices are assembled during the computations. The summary of the algorithm used can be found in Algorithm 1.

Algorithm 1: Central Difference scheme

Previous time step data: $(\mathbf{x}_n, \mathbf{v}_n, \mathbf{p}_n, \boldsymbol{\rho}_n, \mathbf{a}_n, \Delta t_n)$

Half-step velocity: $\mathbf{v}_{n+1/2} = \mathbf{v}_n + \frac{1}{2}\Delta t_n \mathbf{a}_n$

Position update: $\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta t_n \mathbf{v}_{n+1/2}$

Density update: $\boldsymbol{\rho}_{n+1} = (\mathbf{M}^{\rho}_{n+1})^{-1}\mathbf{f}^{\rho}_{n+1}$

Pressure update: $[\mathbf{p}^{n+1}]_i = \frac{K_0}{K_0'}\left[\left(\frac{[\boldsymbol{\rho}^{n+1}]_i}{\rho_0}\right)^{K_0'} - 1\right] \quad \forall i$

Acceleration update: $\mathbf{a}_{n+1} = \mathbf{M}^{-1}_{n+1}\mathbf{f}_{n+1}$

Velocity update: $\mathbf{v}_{n+1} = \mathbf{v}_{n+1/2} + \frac{1}{2}\Delta t_n \mathbf{a}_{n+1}$

Time step update: $\Delta t_{n+1} = C \min_e \left(\frac{h_e}{\max(\|\vec{v}_e\|, \sqrt{K_e/\rho_e})}\right)$

The index $(n+1)$ of the matrices indicates that they have been computed using the updated position $\mathbf{x}_{n+1}$. The vector $\mathbf{f}_{n+1}$ is defined as $\mathbf{D}^{\mathsf{T}}_{n+1}\mathbf{p}_{n+1} - \mathbf{K}_{n+1}\mathbf{v}_{n+1/2} + f_{\text{ext}\,n+1}$ and the vector $\mathbf{f}^{\rho}_{n+1}$ is either $\mathbf{M}^{\rho}_0\boldsymbol{\rho}_0$ or $\mathbf{M}^{\rho}_{n+1}\boldsymbol{\rho}_n - \Delta t_n \mathbf{D}^{\rho}_{n+1}\mathbf{v}_{n+1/2}$ depending on the form of the continuity equation chosen. The mass matrix present in the right hand side of the continuity equation is not lumped in order to stabilize the equations (this procedure is called *Consistent Lumping Stabilization* Meduri [2019]; Ryzhakov et al. [2010]). For the first form mentioned here, the reference configuration can be any previous configuration and in practice is chosen to be the precedent time step configuration.

As this explicit scheme is only conditionally stable, a CFL-like condition is used to compute the next time step (Courant et al. [1967]), using as speed the maximum of the fluid speed and the pressure wave speed in the fluid. The parameter $C \in [0; 1]$ in the time step update is a user-defined safety coefficient. In sub-sonic condition, which is always the condition met in this report, the pressure wave speed is always the greatest and thus the parameter K_0 will strongly impact the size of the time steps and the performance of the explicit solver.

6.2 Incompressible fluid

As there is no explicit equation for the pressure in II.12, an implicit backward-Euler scheme is used. This leads to a system of the form:

$$\begin{bmatrix} \frac{1}{\Delta t}\mathbf{M}_{n+1} + \mathbf{K}_{n+1} & -\mathbf{D}^{\mathsf{T}}{}_{n+1} \\ \frac{1}{\Delta t}\mathbf{C}_{n+1} - \mathbf{D}_{n+1} & \mathbf{L}_{n+1} \end{bmatrix} \begin{pmatrix} \mathbf{v}_{n+1} \\ \mathbf{p}_{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{extn+1} + \frac{1}{\Delta t}\mathbf{M}_{n+1}\mathbf{v}_n \\ \mathbf{h}_{n+1} \end{pmatrix}, \tag{II.17}$$

where the vector $(\mathbf{v} \quad \mathbf{p})^{\mathsf{T}}$ will be denoted $\mathbf{q}$, the left hand side global matrix as $\mathbf{A}$ and the right hand side vector as $\mathbf{b}$. Since the matrix $\mathbf{A}$ is a function of the solution (through the nodes position, which are updated thanks to the velocity), a non-linear algorithm is needed to solve this system. The Picard (fixed-point) algorithm has been chosen for its simplicity. The summary of the algorithm is summarised in Algorithm 2.

Algorithm 2: Implicit Picard algorithm

Previous time step data: $(\mathbf{x}_n, \mathbf{q}_n)$
$k = 0$
Initial guess: $\mathbf{x}_k = \mathbf{x}_n$
while *Convergence not reached* **do**
  Solve $\begin{bmatrix} \frac{1}{\Delta t}\mathbf{M}_k + \mathbf{K}_k & -\mathbf{D}^{\mathsf{T}}{}_k \\ \frac{1}{\Delta t}\mathbf{C}_k - \mathbf{D}_k & \mathbf{L}_k \end{bmatrix} \mathbf{q}_k = \begin{pmatrix} \mathbf{f}_{\mathrm{ext}\,k} + \frac{1}{\Delta t}\mathbf{M}_k \mathbf{v}_n \\ \mathbf{h}_k \end{pmatrix}$ for $\mathbf{q}_k$
  $\mathbf{x}_{k+1} = \mathbf{x}_n + \Delta t \mathbf{v}_{k+1}$
  $k = k + 1$
end
$\mathbf{q}_{n+1} = \mathbf{q}_k$
$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta t\, \mathbf{v}_{n+1}$

The convergence criterion is a criterion on the velocity part of the solution:

$$\frac{\|\mathbf{v}_k - \mathbf{v}_{k-1}\|}{\|\mathbf{v}_{k-1}\|} < \varepsilon\,, \tag{II.18}$$

ε being a user defined parameter. It can happen that for a certain time step the Picard algorithm converges really slowly, or does not seem to converge at all. When the number of Picard algorithm iterations exceeds at certain number (user-defined), the current time step size is decreased. On the other end, if the number of Picard algorithm iterations is too small, the time step increases, up to a user-defined maximum value (called the maximum time step). The coefficients to increase or decrease the time step in those cases are also user-defined.

7 Boundary conditions

The solvers developed in this work currently supports the following kind of boundary conditions:

- No-slip boundary condition. The velocity of the nodes are imposed to a certain value. This is done by modifying the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ in the implicit solver, and by modifying the matrix $\mathbf{M}$ and the vector $\mathbf{f}$ in the explicit solver (in this case, it is the acceleration which is imposed to control the velocity). While the solver can handle non zero velocities for the boundary nodes (and thus moving boundaries), this specific case has not been tested extensively.

- Dirichlet boundary condition on the velocity. The velocity of the nodes are imposed like in the previous case but the nodes will not move. If a patch of fluid is connected to the nodes where this boundary condition is imposed, the elements nearby those nodes will extend and the remeshing algorithm will add nodes at the center of those elements when they are too big, thus simulating an entrance of fluid. Any velocity profile can be created that way.

- Dirichlet boundary condition on the pressure at the free surface. Indeed, the only value that the pressure can take in this work is the value 0. It can be strongly imposed using similar method as the previous boundary conditions, and will be denoted *Pressure imposed at free surface* in the tables describing the parameters used in the different case studies.

8 Implementation

All the code developed can be found at the address:

https://github.com/ImperatorS79/PFEM

The code is written using the C++17 programming language (Stroustrup et al. [2020]), at the exception of the initial and boundary conditions code which is written in Lua v5.1 (de Figueiredo et al. [2020]). The interface between Lua and C++ is performed using the sol2 v3.0 library (Meneide et al. [2020]). Some informations about how boundary conditions can be set-up in Lua are provided in Annex B.

The initial set of nodes generation as well as the results writing is performed thanks to the Gmsh library (Geuzaine and Remacle [2009]). Remeshing and alpha shape algorithm are performed thanks to CGAL (The CGAL Project [2020]). Some informations about how the input mesh file should be built are provided in Annex A.

The linear algebra operations are performed using the Eigen v3.3.7 library (Guennebaud et al. [2010]). This library provides multiple linear solvers for sparse matrices which can be used to solve the $\mathbf{Aq} = \mathbf{b}$ problem in the non-linear algorithm of the implicit solver. The list of those solvers can be found in https://eigen.tuxfamily.org/dox/group__TopicSparseSystems.html. Both direct and iterative solvers exist. In this work, the SparseLU solver has been chosen because it turned out that it was the fastest. That solver is however not parallel.

The parallel threading in the code is implemented using the OpenMP 4.5 standard (OpenMP Review Board [2020]). Finally the parameters reading in the executable is performed using the JSON for Modern C++ library (Lohmann et al. [2020]). However, a SWIG binding (Beazley et al. [2020] is available to execute the code from Python 3 (Python Software Foundation [2020]). Some informations about how the JSON parameters file should look like are provided in Annex C; and about how the python script should look like in Annex D.

The code is divided in two main parts. The libpfemMesh library is responsible for storing a mesh and handling all mesh related operations, like the remeshing. The libpfemSolver library contains the two classes SolverIncompressible and SolverCompressible which can be used to do a simulation. The code provides the ability to extract some data:

- The states (example: v_x, v_y, p,..) at one specific point.
- The total mass of the fluid.
- The maximum/minimum position of the fluid nodes.
- The states for all elements using Gmsh.

The pfem executable is just a wrapper around the libpfemSolver library that reads the parameters from a json file and then launches the simulation.

The simulations have either been run on a personal computer or on the Fabulous aerospace and mechanical engineering department cluster at Uliège.

After having introduced the numerical methods and the discretization, the next part will compare this work's solver to Cerquaglia's solver and analytical solutions in order to verify that the implementation is correct.

Part III
2D Incompressible solver case studies

Contents

In this part, the implicit incompressible solver developed in this work will be compared to analytical solutions when they are available, and to Cequaglia's solver Cerquaglia [2019] when they are not, in order to verify if the implementation is correct. Unless stated differently, the same parameters are used for this work's solver and for Cerquaglia's solver (including the use of a Picard non-linear algorithm).

1 Hydrostatic case

A 2D fluid lying in a box is considered. The purpose of this test is to see if the solver is able to correctly find the pressure field of a static fluid. The geometry of this problem can be found in Figure 9.

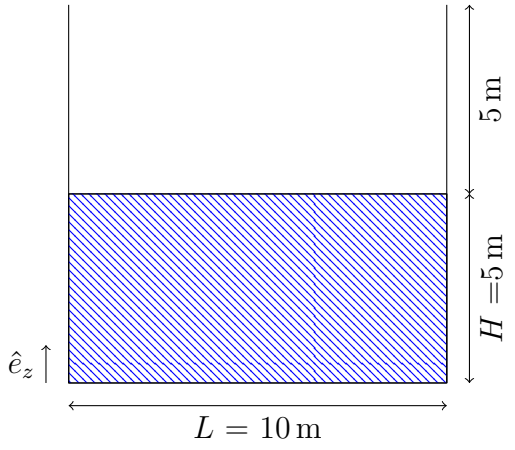


Figure 9: Inital geometry of the hydrostatic problem.

The analytical solution for the unkowns in this problem is simply:

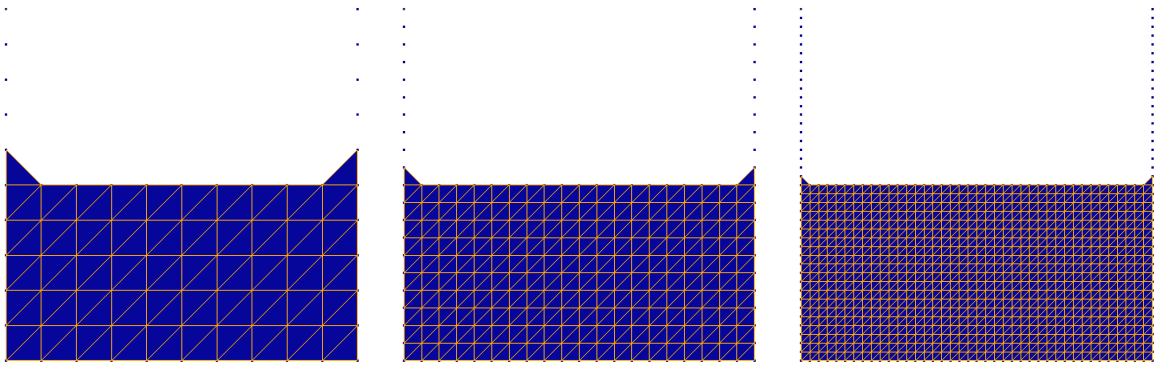
$$\begin{aligned} p(z) &= \rho g(5 - z) \\ \vec{v} &= \vec{0} \end{aligned}, \tag{III.1}$$

assuming the pressure is zero at the free surface. Three simulations were run using three different element characteristic sizes. The parameters used for those simulations are summarized in Table 1. The materials properties are typical of water.

h_{char}	1, 0.5 and 0.25 m
α (alpha-shape)	1.2
γ (nodes removing)	0.7
ω (nodes adding)	0.7
Bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$	$[-2, -1, 12, 100]$
Gravity	$9.81\ \mathrm{kg\,m\,s^{-2}}$
Pressure imposed at free surface	yes
Adaptive time step	yes
Coefficient when increasing Δt	1.5
Coefficient when decreasing Δt	2
Maximum Δt	0.01 s
Initial Δt	0.01 s
Simulation span	4 s
Picard algorithm relative tolerance	5×10^{-6}
Picard algorithm maximum iterations number	10
Density	$1000\ \mathrm{kg\,m^{-3}}$
Dynamic viscosity	0.001 Pa s

Table 1: Parameters used for solving an incompressible hydrostatic problem whose geometry is described in Figure 9.

The initial meshes can be found in Figure 10.

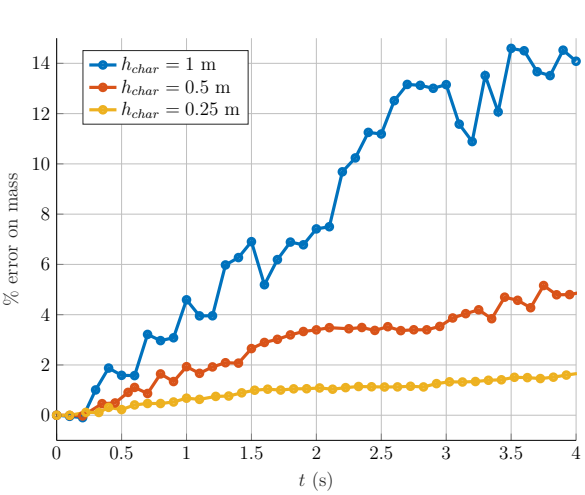


(a) $h_{char} = 1\,\mathrm{m}$, 76 nodes. (b) $h_{char} = 0.5\,\mathrm{m}$, 251 nodes. (c) $h_{char} = 0.25\,\mathrm{m}$, 901 nodes.

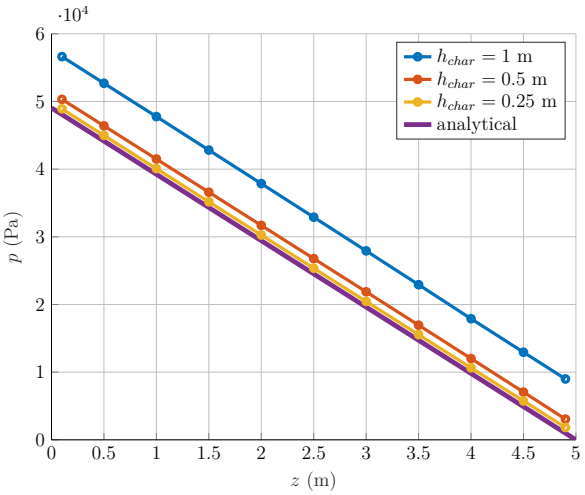
Figure 10: Initial mesh for three different mesh densities for hydrostatic problem whose geometry is described in Figure 9 and parameters given in Table 1.

As it can be seen in Figure 10, two supplementary triangles lie on top of the actual free surface. Those triangles are generated by CGAL which cannot erase them since they have the same size and form as the others triangles and are connected to the main fluid domain. Since the size of those triangles decreases with the h_{char} parameters, this is does not generate an inconsistency/problem in the formulation.

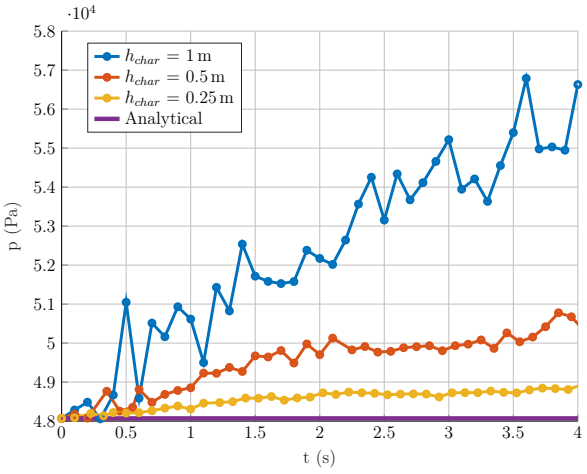
The evolution of the total mass with respect to time and the pressure profile along the vertical axis in the middle of the box at time $t = 4\,\mathrm{s}$ are considered. The results of the simulations can be found in Figure 11.



(a) Error on mass as a function of time.



(b) Pressure along the z axis at the middle of the water column at $t = 4\,\mathrm{s}$.

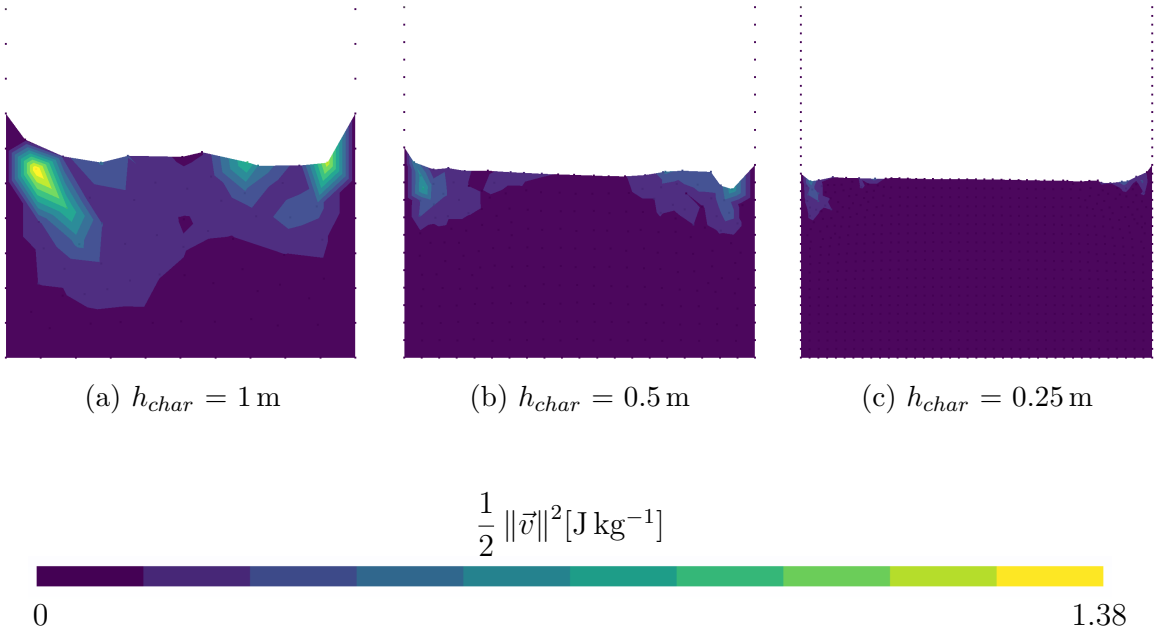


(c) Pressure at the point $(5, 0.1)$ (middle of the water column, near the bottom) as a function of time.

Figure 11: Simulation results for the hydrostatic case whose geometry is described in Figure 9 and parameters in Table 1.

It can be seen that the error on the pressure decreases when the average element size decreases. However, the mass is constantly increasing in all the three cases. This increase of mass over time is correlated to an increase of pressure over time at the bottom middle point of the box. This is not expected for such an hydrostatic case where the total

mass and the pressure at that point should remain constant. This generation of mass is maybe due to non-zero velocities induced by the two spurious triangles, which have been identified in Figure 10. An insight on those velocities is provided in Figure 12.



(a) $h_{char} = 1\,\mathrm{m}$ (b) $h_{char} = 0.5\,\mathrm{m}$ (c) $h_{char} = 0.25\,\mathrm{m}$

Figure 12: Specific kinetic energy for three different mesh size at $t = 4\,\mathrm{s}$ in the hydrostatic case whose geometry is described in Figure 9 and parameters in Table 1.

Since the error on the free surface position and shape as well as the total kinetic energy generated in the fluid decreases when h_{char} decreases, this again does not generate an inconsistncy/problem in the formulation. One could be tempted to delete the two spurious triangles in order to see if the errors disappear. Unfortunately, this leads to an infinite loop in the Picard algorithm, with the relative error not changing between the iterations. As a conclusion here, one should be careful when simulating flows containing an hydrostatic part for too long. One could try to delete those two spurious triangle in order to see if the fluid is then motionless. Unfortunately, in this case the Picard algorithm seems unable to converge to a solution.

2 Flow in between two plates

A flow between two plates is considered, whose geometry can be found in Figure 13. There is no gravity in this problem. A no-slip boundary condition with $\vec{0}$ velocity is applied on the two plates. At the "entrance" of the plates, on the left, a Dirichlet boundary condition with $\vec{v} = 1\hat{e}_x\ \mathrm{m\,s^{-1}}$ is applied. An initial surface of fluid of length $0.2\,\mathrm{m}$ is also placed at the "entrance" of the plates, near the Dirichlet nodes, and the same initial condition is applied on the nodes of this surface. All those nodes which have an initial velocity different from $\vec{0}$ will start to move to the right. As described in the previous part, the Dirichlet nodes will not move, and thus the remeshing algorithm will add nodes at center of the deforming elements, which will simulate an input of fluid (the time step size should be inferior to $h_{char}/\|\vec{v}\|$ to be sure that the fluid does not detach from the Dirichet boundary and that a fluid input is correctly simulated).

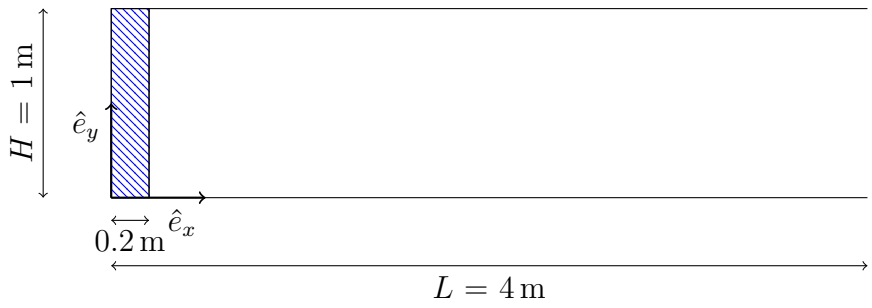


Figure 13: Initial geometry of the flow between two plates problem.

If the fluid flows long enough in between long enough plates, one could expect that the transient solution converges to a steady-state solution, which is:

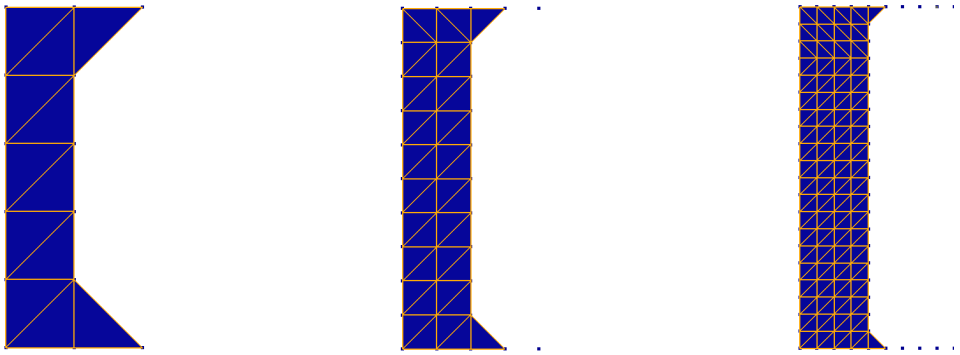
$$v_x(y) = \frac{6Q}{H^3} y(H-y)\,, \quad \frac{dp}{dx} = -\frac{12\mu Q}{H^3}\,, \tag{III.2}$$

where Q is the volume flow rate and μ the dynamic viscosity. A high viscosity has been chosen in this test in order for the fully-developed region of the flows to appear soon enough. The flow at 3 m from the entrance will be compared to the steady-sate solutions, even if a perfect matching is not expected, relatively the same behaviour and order of magnitude of the solution are expected. Multiple simulations were run using three different element characteristic sizes, comparing the profile of the horizontal velocity with respect to y at 3 m from the entrance and the profile of the pressure field along the horizontal axis at time $t = 4\,\mathrm{s}$. A parabola initial condition on the velocity following the formula in equation III.2 has also been tested for one of the h_{char}. The parameters used for those simulations are summarized in Table 2.

h_{char}	0.2, 0.1 and 0.05 m
α (alpha-shape)	1.2
γ (nodes removing)	0.7
ω (nodes adding)	0.7
Bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$	$[-1, -0.25, 5, 1.25]$
Gravity	$0\ \mathrm{kg\,m\,s^{-2}}$
Pressure imposed at free surface	yes
Adaptive time step	yes
Coefficient when increasing Δt	1.5
Coefficient when decreasing Δt	2
Maximum Δt	0.005 s
Initial Δt	0.005 s
Simulation span	5 s
Picard algorithm relative tolerance	5×10^{-6}
Picard algorithm maximum iterations number	10
Density	$1000\ \mathrm{kg\,m^{-3}}$
Dynamic viscosity	200 Pa s

Table 2: Parameters used for solving an incompressible pipe problem whose geometry is described in Figure 13.

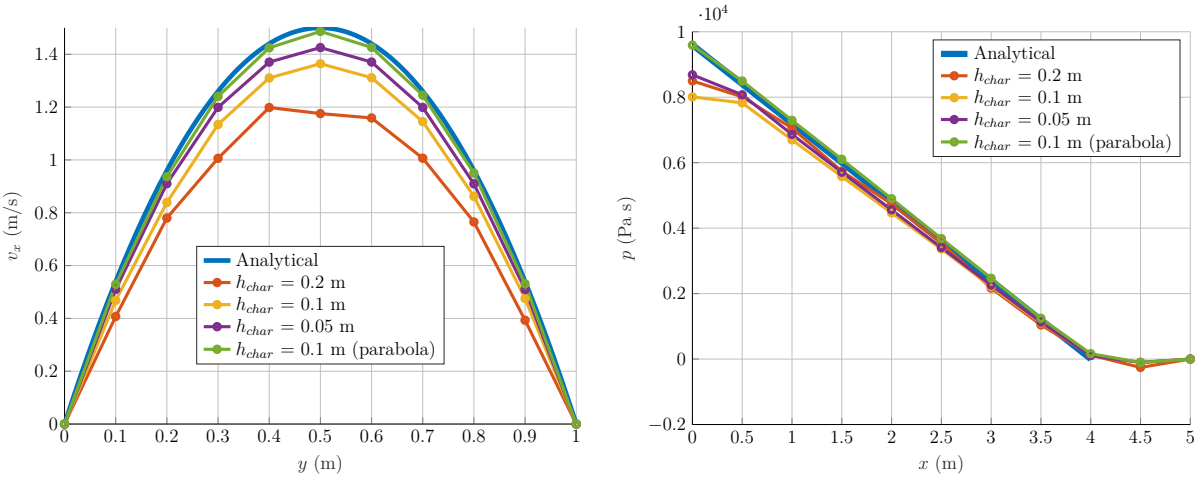
For the case $h_{char} = 0.1\,\text{m}$, a initial condition equal to the analytical results presented in III.2 is also considered. The initial meshes can be found in Figure 14.



(a) $h_{char} = 0.2\,\text{m}$, 50 nodes. (b) $h_{char} = 0.1\,\text{m}$, 109 nodes. (c) $h_{char} = 0.05\,\text{m}$, 257 nodes.

Figure 14: Left part of the initial mesh for three different mesh densities for the problem whose geometry is described in Figure 9 and parameters are given in Table 1.

Like previously, two supplementary triangles lie on front of the actual free surface. The results of the simulations can be found in Figure 15.



(a) Horizontal velocity profile along $x = 3\,\text{m}$. (b) Pressure profile along $y = 0.5\,\text{m}$.

Figure 15: Simulation results for three different mesh size for the pipe case whose geometry is described in Figure 13 and parameters in Table 2 at $t = 4\,\text{s}$.

As it can be seen, the velocity profile converges to the analytical steady state solution as the mesh characteristic size decreases. The pressure profile is also following well the steady state solution in the half end of the pipe, which is expected since the beginning of the pipe is experiencing a transient behaviour. The gradient of pressure seems to change when $x > 4$ m such that the pressure does not evolve anymore. This is simply due to the fact that the plates are only 4 m long. When an initial condition following equation III.2 is imposed as a Dirichlet boundary condition, the matching between the numerical results and the steady-state analytical solution is even better.

A graphical evolution of the fluid and its horizontal velocities at three time steps for $h_{char} = 0.05\,\text{m}$ can be seen in Figure 16.

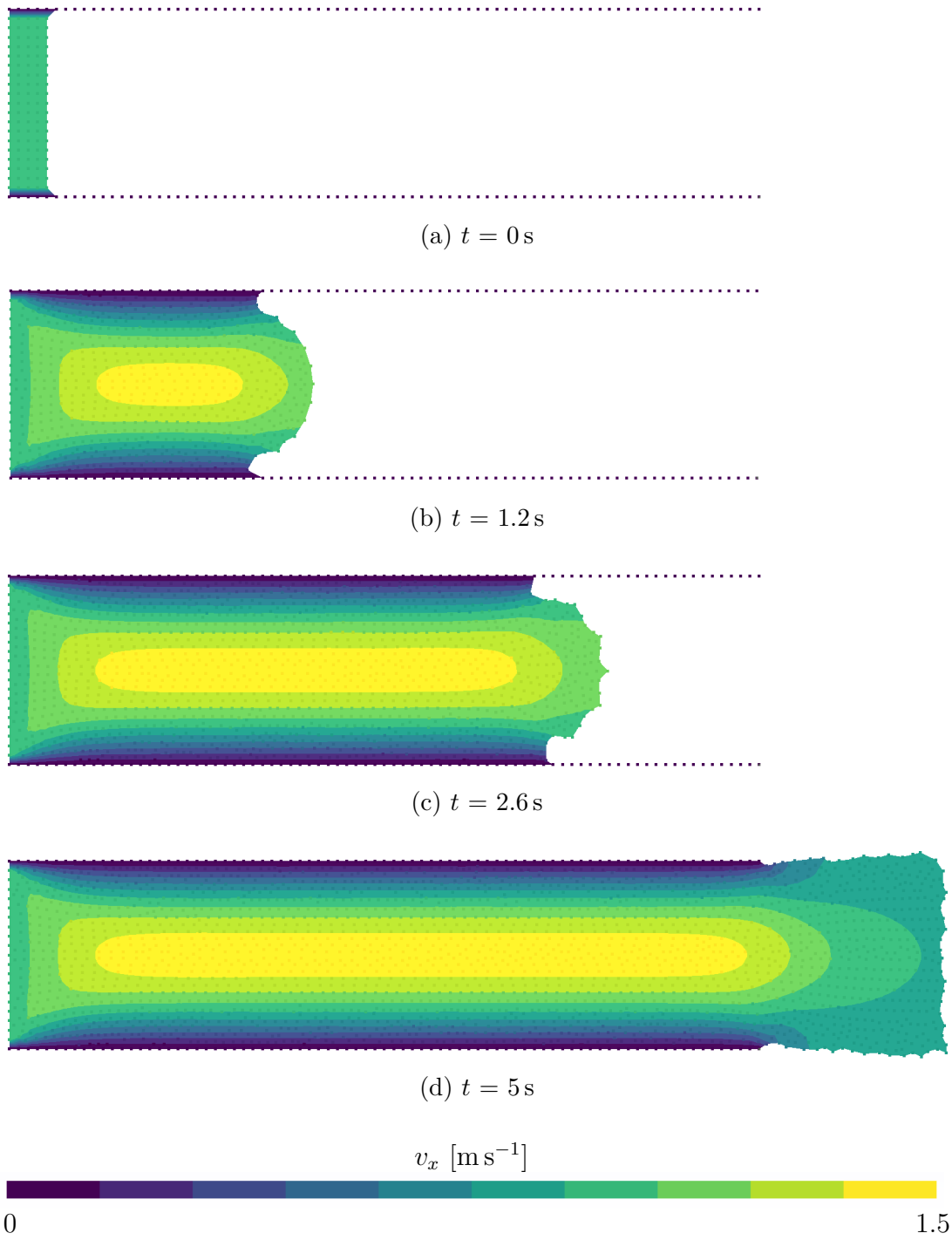


Figure 16: Horizontal component of the velocity at different times for $h_{char} = 0.05\,\mathrm{m}$ in the case whose geometry is described in Figure 9 and parameters are given in Table 1.

It can be seen on Figure 16d that when the fluid has reached the end of the plates, the behaviour of the solution change and switch from a parabola-like profile to a more uniform profile.

3 Sloshing

A sloshing problem considers the movement of a fluid with a free surface inside an object. In the problem studied hereafter, a fluid whose free surface initial position is given by:

$$y(x) = 1 + 0.1\sin(\pi x - \frac{\pi}{2}), \tag{III.3}$$

is lying in a box and no-slip boundary conditions are applied to the wall of that box. The resulting geometry is presented in Figure 17.

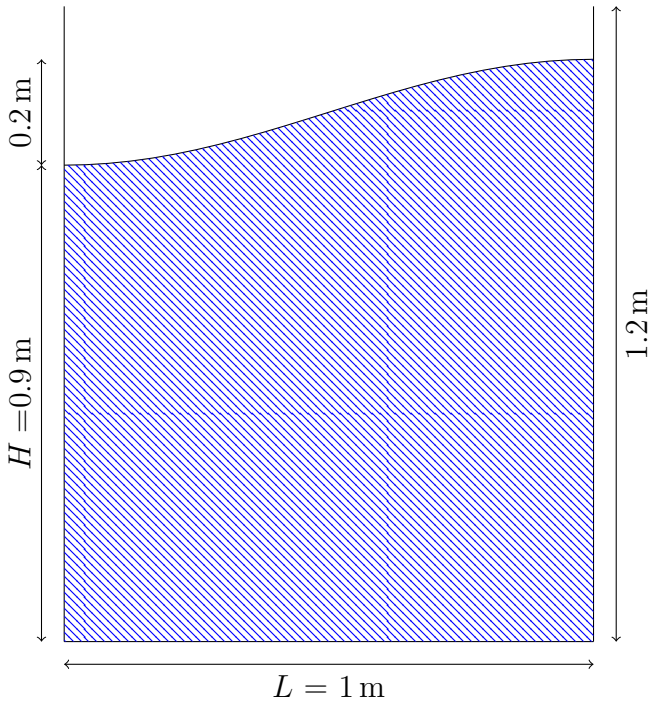


Figure 17: Initial geometry of the sloshing problem .

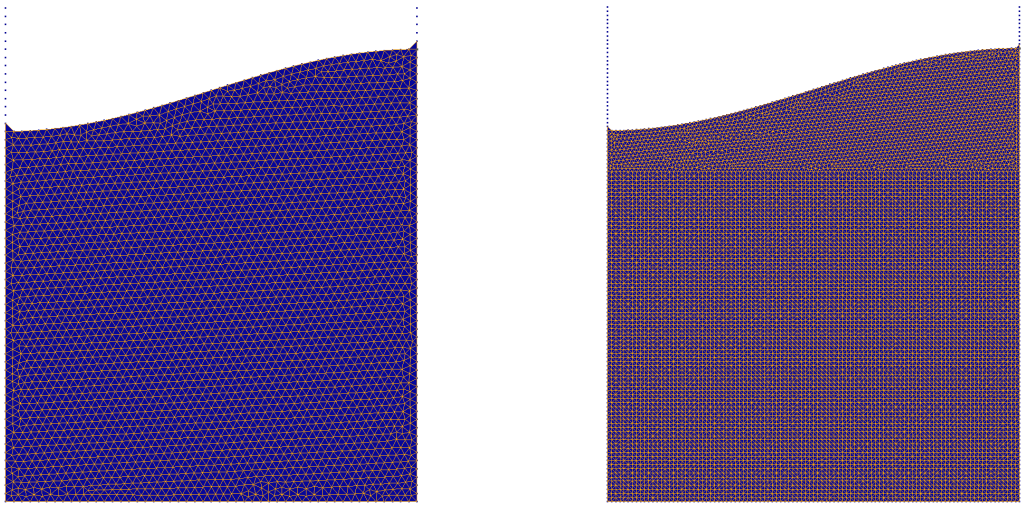
Two simulations are run using two different h_{char} from both the present solver and the one proposed by M.-L. Cerquaglia. The parameters used for those simulations are summarized in Table 2.

An analytical solution exists for this case when a small amplitude of the free surface perturbation and zero friction with the wall are considered (Wu et al. [2001]). The solvers developed in this work do not support such kind of boundary conditions. However, a damped oscillation of the free surface is still expected.

h_{char}	0.02 and 0.01 m
α (alpha-shape)	1.2
γ (nodes removing)	0.7
ω (nodes adding)	0.7
Bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$	$[-2, -1, 12, 100]$
Gravity	9.81 kg m s^{-2}
Pressure imposed at free surface	yes
Adaptive time step	yes
Coefficient when increasing Δt	1.5
Coefficient when decreasing Δt	2
Maximum Δt	0.01 s
Initial Δt	0.01 s
Simulation span	4 s
Picard algorithm relative tolerance	5×10^{-6}
Picard algorithm maximum iterations number	10
Density	1 kg m^{-3}
Dynamic viscosity	0.01 Pa s

Table 3: Parameters used for solving an incompressible sloshing problem whose geometry is described in Figure 17.

The initial meshes can be found in Figure 18.



(a) $h_{char} = 0.02\,\text{m}$, 3005 nodes. (b) $h_{char} = 0.01\,\text{m}$, 10555 nodes.

Figure 18: Initial mesh for two different mesh density for the sloshing problem whose geometry is described in Figure 17 and parameters in Table 3.

The maximum of the fluid height will be tracked. The results are presented in Figure 19.

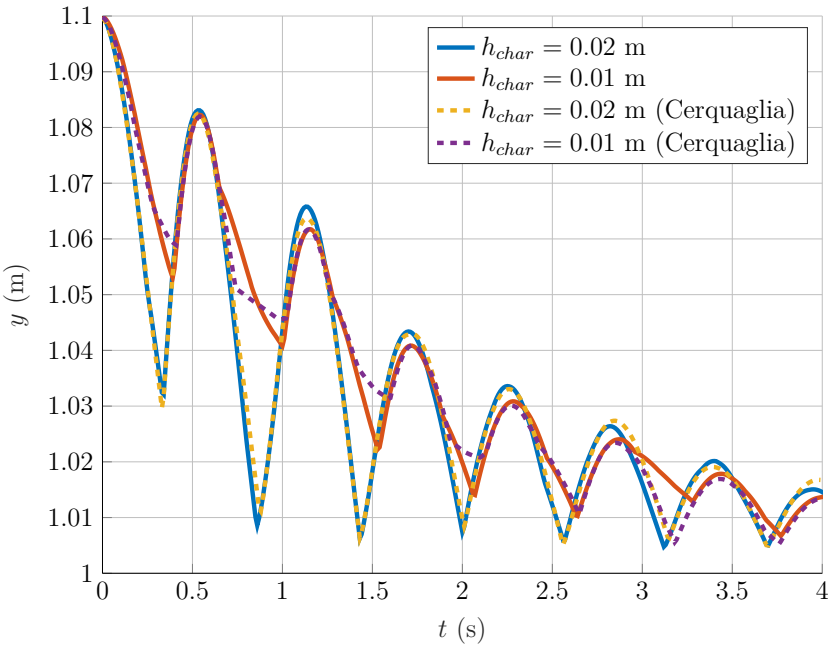


Figure 19: Simulation results for three different mesh size for the pipe case whose geometry is described in Figure 13 and parameters in Table 2 at $t = 4\,\mathrm{s}$.

The damped oscillations of the free surface are recovered. In the case $h_{char} = 0.02\,\mathrm{m}$, this work's solver and Cerquaglia solvers gives really close solutions up to $t = 2.5\,\mathrm{s}$ where some differences appear. In the case $h_{char} = 0.01\,\mathrm{m}$, this work's solver and Cerquaglia solvers gives really close solutions when near the different local maximum of the solution. However, in between those maximum, the "filling" is different. As the algorithm tracks the maximum of the free surface height, it happens than a particle lying near the wall will move more slowly due to the friction with the wall than a particle less close to the wall, which explains why the maximum of the free surface height suddenly seems to slow down. The "filling" behaviour remains slightly different between the two solvers, but it can be recalled to the reader that those two solvers are not the same (e.g. the remshing and the linear algebra operations are not performed by the same software !).

4 Dam break

A "dam break" problem is considered. It consists of a rectangle surface of fluid initially resting which will start to move when one of the wall maintaining it is instantaneously removed at time $t = 0\,\mathrm{s}$. Its geometry can be found in Figure 20. This particular geometry has been chosen in order to compare to solution to experimental results in Koshizuka and Oka [1996] later on.

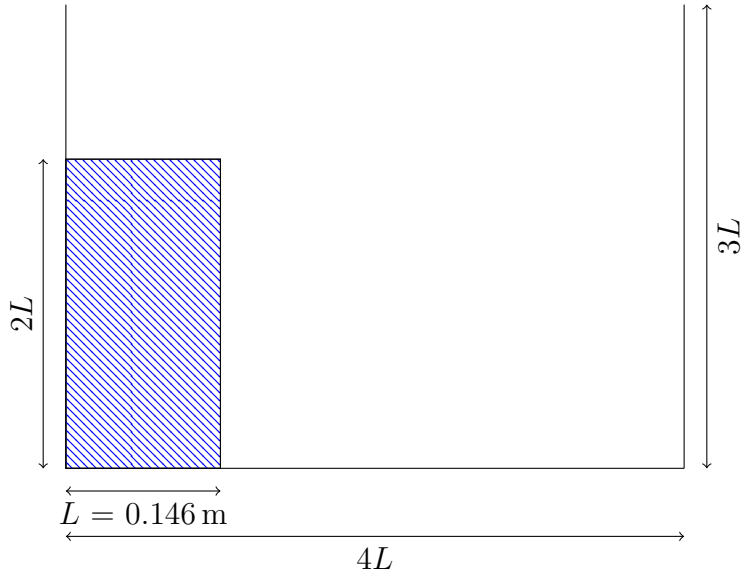


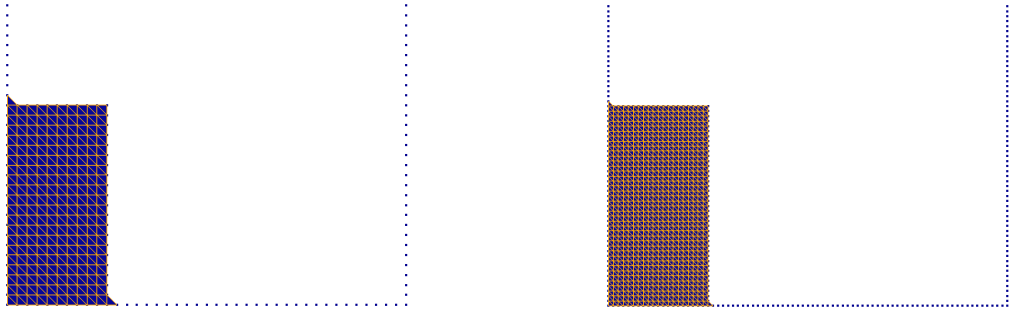
Figure 20: Initial geometry of the dam break problem.

Since there is no analytical solution to this problem, two simulations on both the code developed in this thesis and Cerquaglia's code were run. The parameters for this problem can be found in Table 4 and the mesh used in Figure 21.

h_{char}	0.0146 and 0.0073 m
α (alpha-shape)	1.2
γ (nodes removing)	0.7
ω (nodes adding)	0.7
Bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$	$[-0.01, -0.01, 0.594, 100]$
Gravity	9.81 kg m s^{-2}
Pressure imposed at free surface	yes
Adaptive time step	yes
Coefficient when increasing Δt	1.5
Coefficient when decreasing Δt	2
Maximum Δt	0.01 s
Initial Δt	0.01 s
Simulation span	2 s
Picard algorithm relative tolerance	5×10^{-6}
Picard algorithm maximum iterations number	10
Density	1000 kg m^{-3}
Dynamic viscosity	0.001 Pa s

Table 4: Parameters used for solving an incompressible dam break problem whose geometry is described in Figure 20

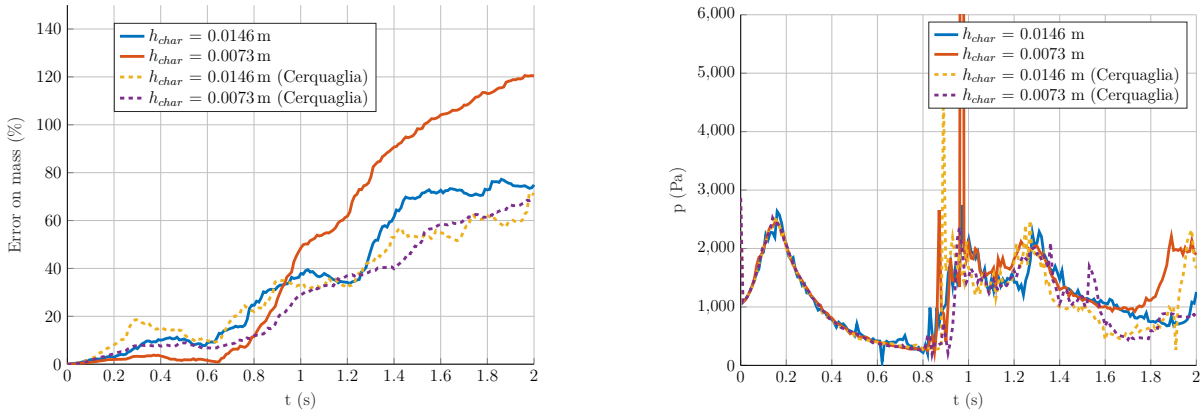
.



(a) $h_{char} = 0.0146\,\text{m}$, 301 nodes. (b) $h_{char} = 0.0073\,\text{m}$, 1001 nodes.

Figure 21: Initial mesh for two different mesh density for the dam break problem whose geometry is described in Figure 20 and parameters in Table 4.

The total mass of the fluid as well as the evolution of the pressure in the left corner will be studied. The results of the simulations can be found in Figure 22.



(a) Error on mass as a function of time. (b) Pressure at $(x, y) = (0, 0)$ m as a function of time.

Figure 22: Simulation results for two different mesh size for the dam break case whose geometry is described in Figure 20 and parameters in Table 4.

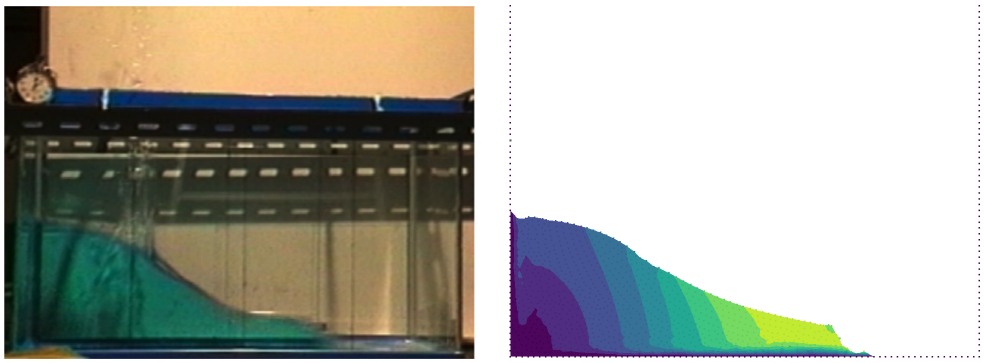
The first comment which should be done is that the mass increases a lot in those simulations (it doubles after 2 s). The adding and the removing constant should be checked to prevent that from happening. One could also note that in the case $h_{char} = 0.0073$ m for the solver developed in this work, the mass surprisingly suffers the greatest increase. As stated in subsection II.2.2, the two definitions of the alpha-value are slightly different and thus could also explains the difference of results between the two solvers. However such a great increase of mass (two times more mass after two seconds) is problematic and a study of the relation between the remeshing parameters and the mass evolution could be interesting.

Regarding the pressure evolution at the left corner, all the four curves seems to provide the same solution before 0.8 s, but the fluid attains the right corner at approximatively 0.27 s. The local maximum near 0.9 s corresponds to the come back of the closing from the right to the left. The interpretation for the rest of the curve is difficult since the big

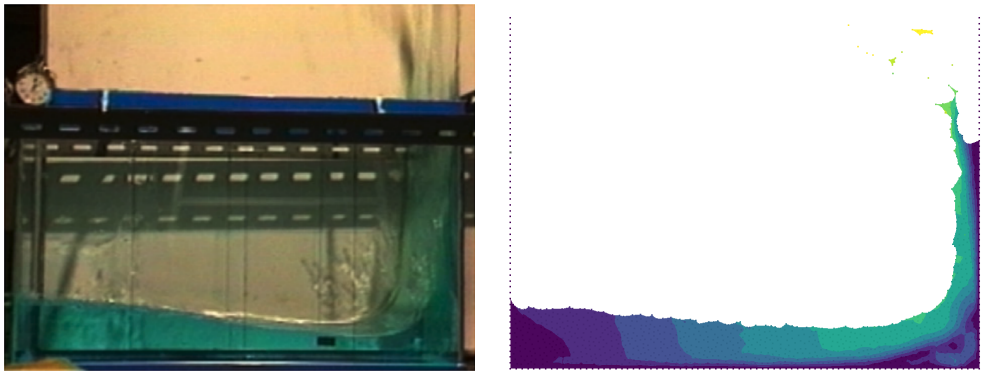
increase of mass that the fluid has encountered, but both Cerquaglia's and this work's solver remains in the same order of magnitude.

A lot of "spikes" can be seen on the pressure field. A plausible hypothesis is that they appear because of the adding/removing nodes algorithm. Indeed one could imagine that deleting a node makes the local velocity field too much not divergent-free so that a spike appears in the pressure afterwards. However, completely deactivating the addition and deletion of nodes is not feasible as it would cause nodes to cross the boundaries during the flow of the fluid. This should however be investigated further.

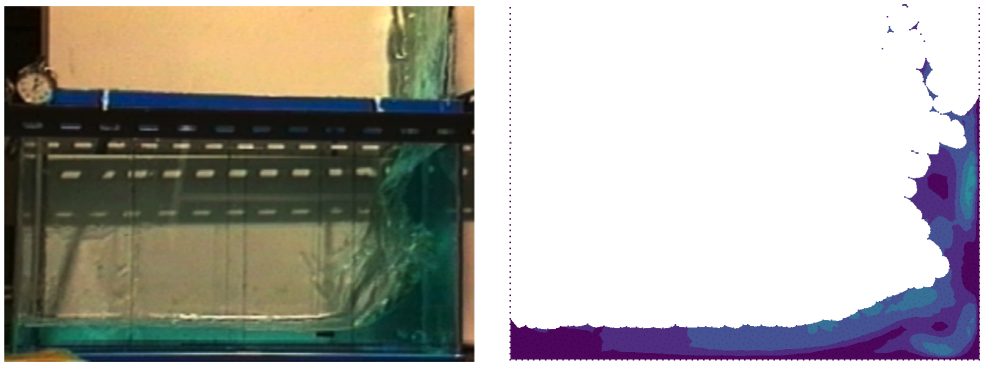
The flow can also be compared to experimental results from Koshizuka and Oka [1996]. This comparison is done in Figure 23. The simulation matches the experimental results quite nicely, even though as time advances in the simulation, the probability of a chaotic behaviour to appear increases, deteriorating the results.



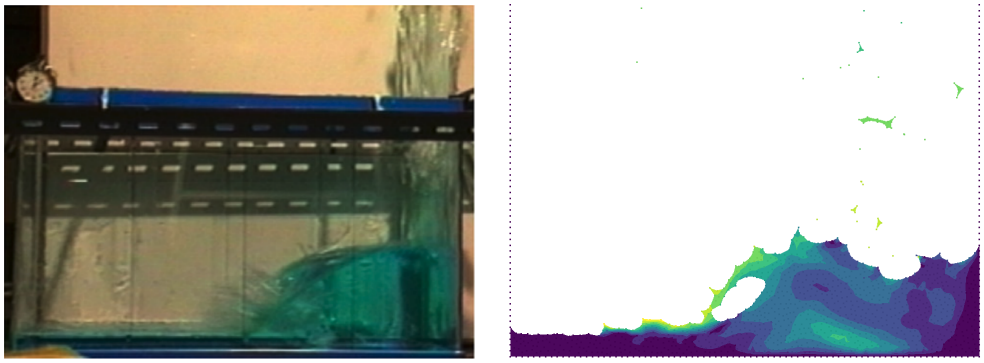
(a) $t = 0.2\,\mathrm{s}$



(b) $t = 0.4\,\mathrm{s}$



(c) $t = 0.6\,\mathrm{s}$



(d) $t = 0.8\,\mathrm{s}$

$\frac{1}{2}\|\vec{v}\|^2 [\mathrm{J\,kg^{-1}}]$

0 1.35

Figure 23: Simulation results for $h_{char} = 0.0073\,\mathrm{m}$ for the dam break case whose geometry is described in Figure 20 and parameters in Table 4, compared to experimental results in Koshizuka and Oka [1996].

5 Dam break with obstacle

The same dam break as in the previous section with a rigid obstacle in the fluid's way is now considered and the geometry of this problem is presented in Figure 24.

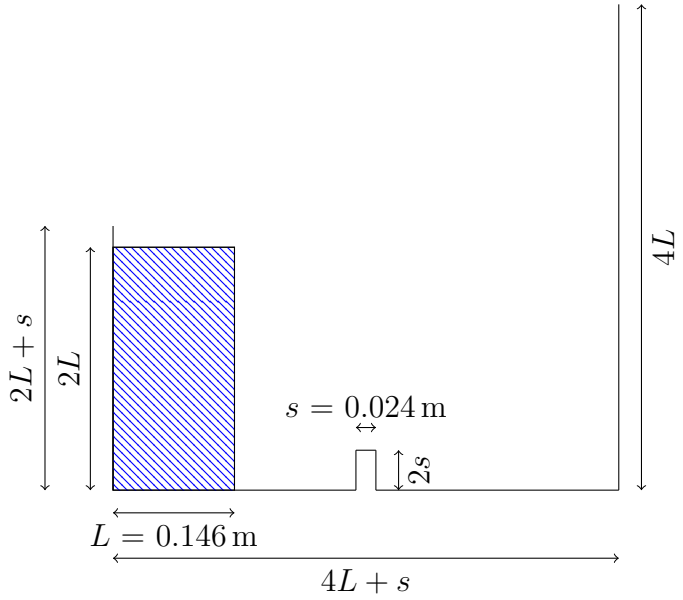


Figure 24: Initial geometry of the dam break with obstacle problem.

The initial mesh used can be seen in Figure 25. The parameters are the same as in 4, except that $h_{char} = 0.006$ m and the maximum time step is 0.0005 s.

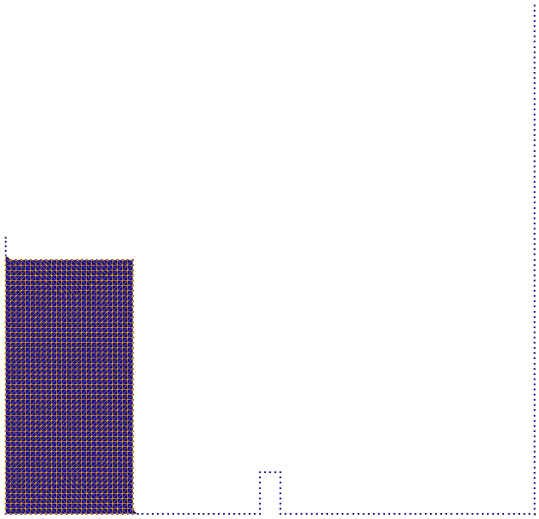
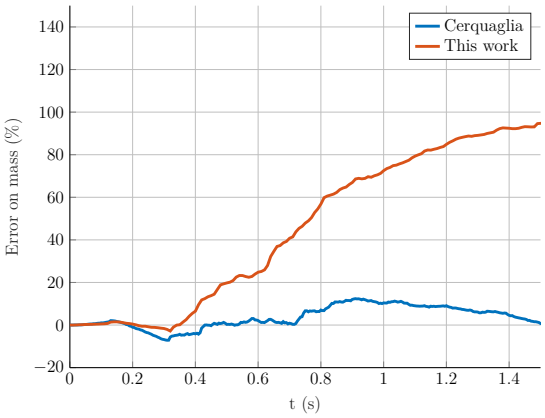
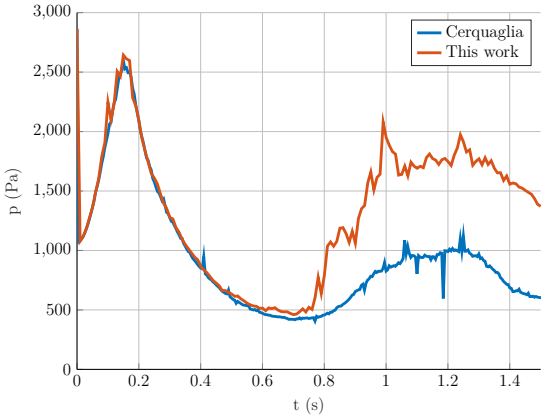


Figure 25: Initial mesh for the dam break with obstacle problem whose geometry is described in Figure 24 and parameters in Table 4, $h_{char} = 0.006$ m, 1497 nodes.

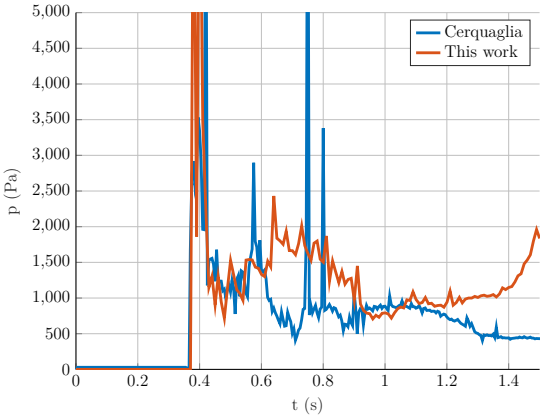
The evolution of the total mass and the pressure at the leftmost and rightmost corners over time will be studied. The results are presented in Figure 26



(a) Error on mass as a function of time.



(b) Pressure at the leftmost corner as a function of time.



(c) Pressure at the rightmost as a function of time.

Figure 26: Simulation results for the dam break with obstacle case whose geometry is described in Figure 24 and parameters in Table 4.

The same conclusion can been drawn here than for the dam break problem in the previous section. The difference between Cerquaglia's solver and this work's solver is even greater (Cerquaglia's solver is conserving the mass quite well). Again there are differences between the two solvers in the remeshing part and a study of the influence of the remeshing parameters on the mass evolution should be done. Regarding the pressure at the leftmost corner, the two solvers give nearly the same result until 0.8 s. After that, the curve from this work's solver seems to be shifted up with respect to the curve from Cerquaglia's solver. Regarding the pressure at the rightmost corner, the time required by the fluid to reach that point (0.3 s) seems to be the same for both solvers, the pressure remaining in the same order of magnitude afterwards.

A graphical comparison between the two solvers is provided in figure 27. Globally, the two solutions behave quite the same way in the range displayed. However it seems that there is more mass detaching from the tail which is appearing when the fluid hit the obstacle in this work's solver than in Cerquaglia's solver.

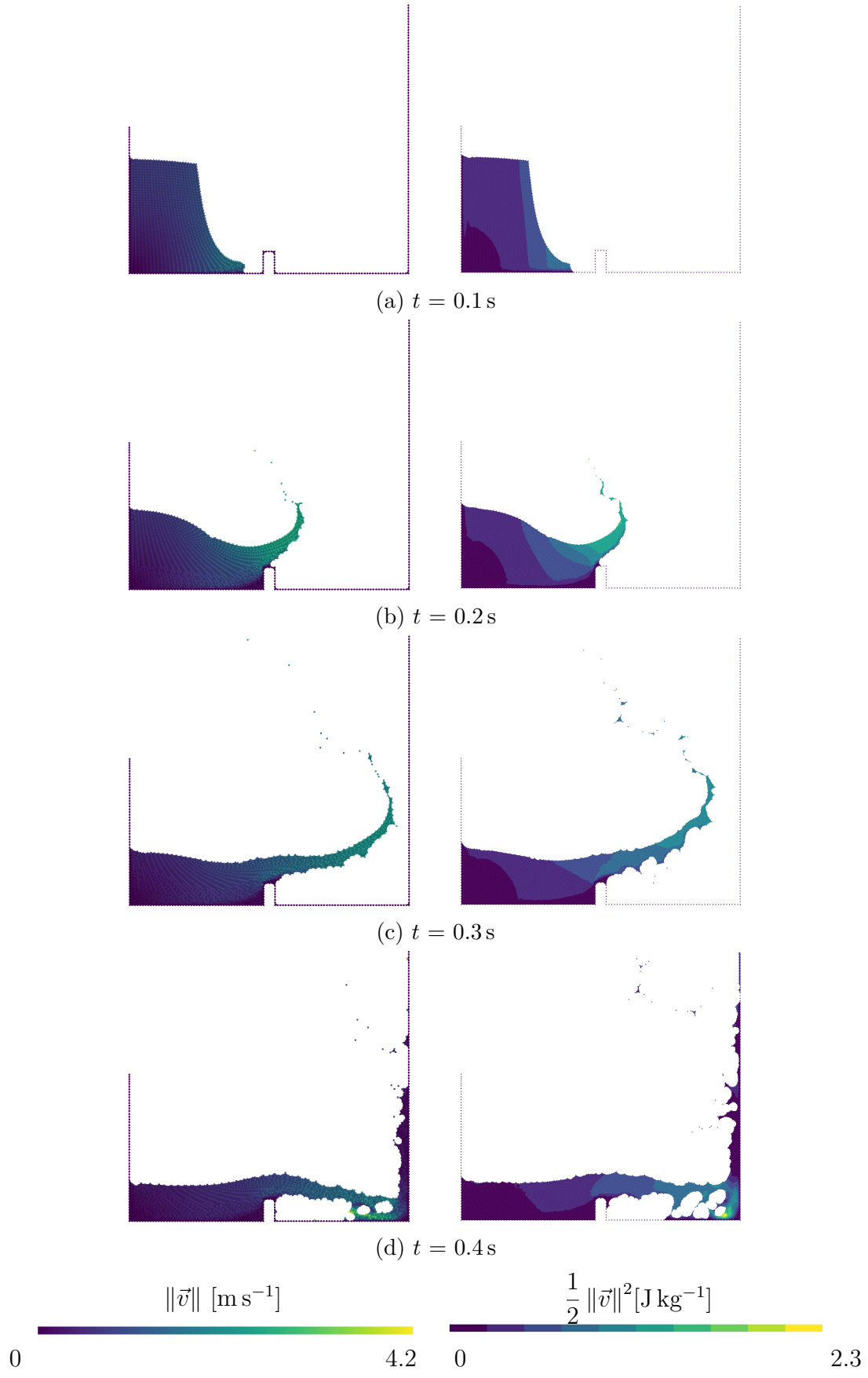


Figure 27: Comparison of this work's solver results (on the right) and Cerquaglia's solver results (on the left) for the dam break with obstacle problem.

6 Performance considerations

A comparison of the typical time to compute the solution and the typical time of remeshing is provide in Figure 28. The problem used in this figure is the dam break problem from section 4, except that the time step used was 0.0001 s. Only one thread is used.

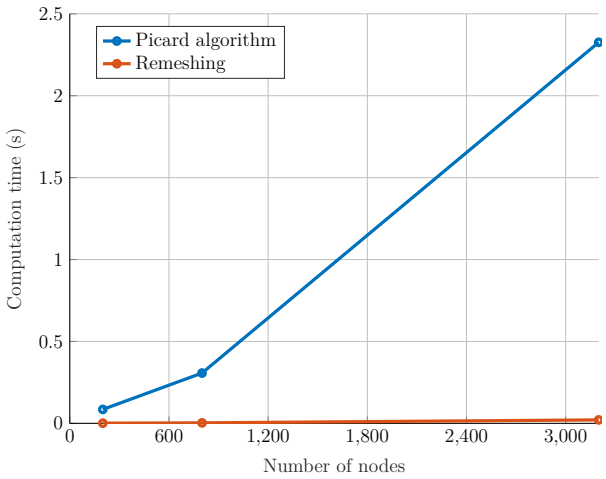


Figure 28: Comparison of the remeshing computation time and the computation time of the Picard algorithm in the dam break problem described in Figure 20 and in Table 4 for one typical time step.

It can be seen that the computation time for the remeshing is always smaller that the time to compute the solution. More over, the time to compute the remeshing increases more slowly than the time to compute the solution when the number of nodes initially present increases. The remeshing is thus not blocking regarding the computation time for the incompressible implicit solver developed in this work.

6.1 Mesh size

The computation time for various initial number of nodes are compared in Figure 29 for this work's solver and Cerquaglia's solver. The used problem is again the problem from 4, except that the time step used was 0.0001 s. Only one thread is used for both solvers. The solver developed is this work seems to be always faster. However, this result has to be taken with caution as it is never impossible that an error in the parameters of the different solvers could have been made.

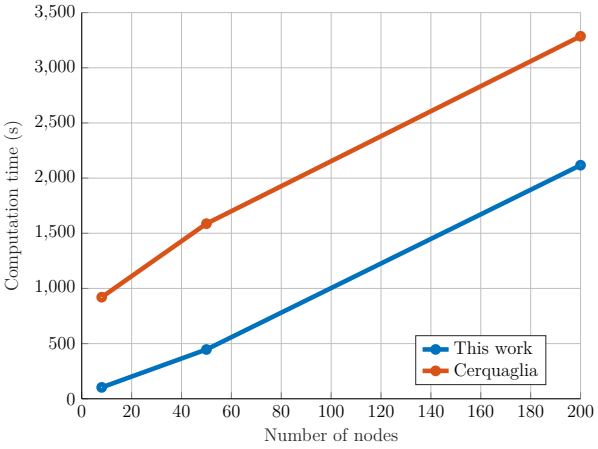


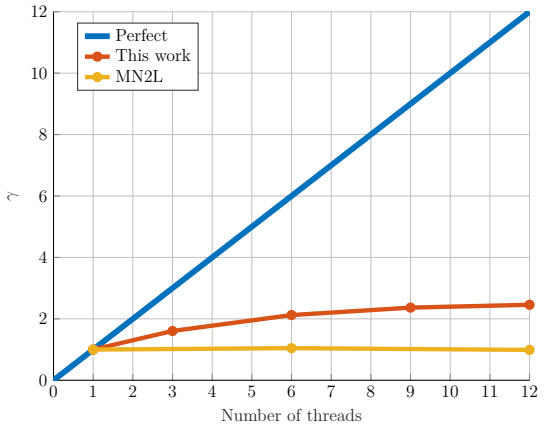
Figure 29: Comparison of the computation time between this work's solver and Cerquaglia's solver for various initial number of nodes.

6.2 Parallelization

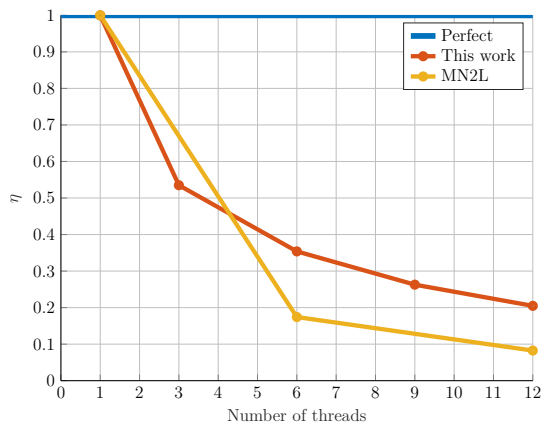
Defining τ_N the execution time of the program using N threads, using `OpenMP` (OpenMP Review Board [2020]) in this work and Intel Thread Building Blocks (Intel [2020]) in Cerquaglia's solver, the speed-up ratio and the parallel efficiency are defined as:

$$\gamma = \frac{\tau_1}{\tau_N} \quad \text{and} \quad \eta = \frac{\tau_1}{N\tau_N} \tag{III.4}$$

The two solvers are compared in Figure 30. The used problem is again the problem from III4, except that the time step used was 0.0001 s.



(a) Speed-up ration as a function of the number of threads.



(b) Parallel efficiency as a function of the number of threads.

Figure 30: Comparison of the parallelization of Cerquaglia's solver and this work's solver.

The parallel performance of both solvers is really bad. Cerquaglia's solver also seems not to speed-up at all when changing the number of threads, which is strange and maybe due to an error from the author of this report while using Cerquaglia's solver.

As explained in section II.8, this work's solver uses `Eigen` (Guennebaud et al. [2010]) to solve the linear system involved in the Picard non-linear algorithm. In particular, it uses

its `SparseLU` solver, because it appears in early test that it was the fastest. This solver is however a serial solver and thus cannot be accelerated using threads. A similar method is used in the Picard algorithm implemented in Cerquaglia's solver, through the `gmm` library (Renard et al. [2020]). This could explain a part of the bad threading performance of both solvers.

Other ways of improvements could be explored:

- The use of another more efficient non-linear algorithm (Newton-Raphson could be tried).
- The use of an iterative algorithm to solve the linear system. Early test showed a degradation of performance using `Eigen`'s iterative solvers but the knowledge of them by the author of this report was quite limited and it should be worth to try again.

This work's implicit incompressible solver has been compared to Cerquaglia's solver and analytical solutions. While it seems that this work's solver is able to correctly simulate incompressible flows, a problem of too big increase of mass during the simulations was identified. In the next part, the explicit weakly-compressible solver developed in this work will be compared to the implicit incompressible solver developed in this work in order to verify that the implementation is correct.

Part IV
2D Compressible solver case studies

Contents

In this part, we will focus on the compressible solver implemented in this work by looking at various case studies. When an analytical solution is not available, the results will be compared to this work's incompressible solver. The geometries and meshes used are the same as in the previous part so they will not be recalled here. In the following, "strong" continuity will denote the use of the continuity equation without explicit time derivative (i.e. $\mathbf{M}^{\rho}\boldsymbol{\rho} = \mathbf{M}_0^{\rho}\boldsymbol{\rho}_0$).

1 Hydrostatic case

The same case as in the previous part is considered using the compressible solver developed in this work. The geometry and the meshes are the same as in Figure 9 and 10 and the parameters used can be found in Table 5.

The analytical solution for the unknowns in this problem is:

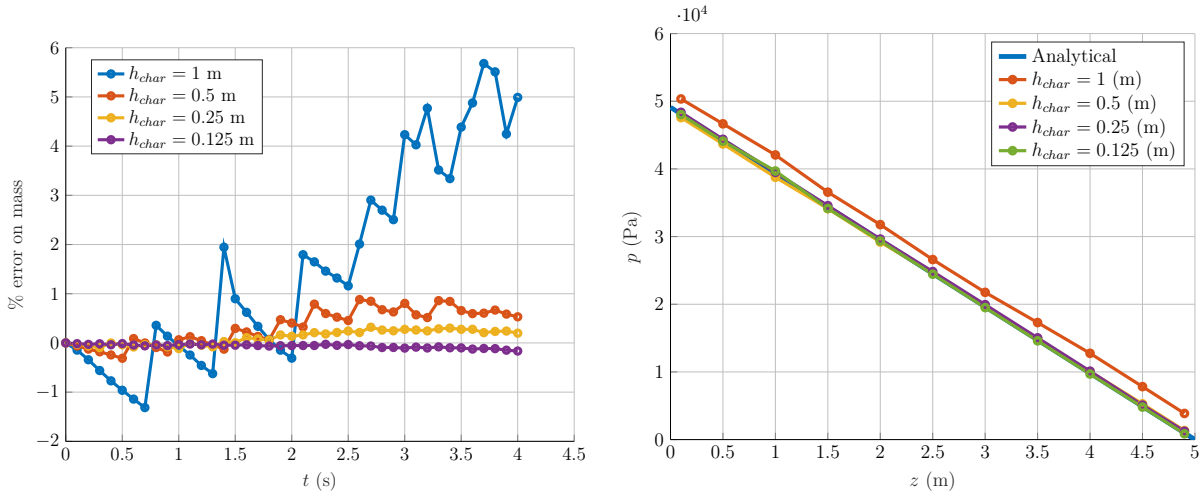
$$\begin{aligned} \rho &= \rho_0 \left(\frac{K_0' - 1}{K_0} \rho_0 g (5 - z) + 1 \right)^{\frac{1}{K_0' - 1}} \\ p &= \frac{K_0}{K_0'} \left[\left(\frac{\rho}{\rho_0} \right)^{K_0'} - 1 \right] \\ \vec{v} &= \vec{0}\,, \end{aligned} \tag{IV.1}$$

The purpose of this test is to verify the quality of the numerical solution with respect to the analytical solution in a certain range of the parameters K_0 and K_0'.

h_{char}	1, 0.5, 0.25 and 0.125 m
α (alpha-shape)	1.2
γ (nodes removing)	0.7
ω (nodes adding)	0.7
Bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$	$[-0.1, -0.1, 10.1, 100]$
Gravity	9.81 kg m^2 s^{-1}
Pressure imposed at free surface	yes
"Strong" continuity	yes
Adaptive time step	yes
Security coeffeficent	0.2
Maximum Δt	0.001 s
Initial Δt	10^{-8} s
Simulation span	4 s
Density	1000 kg m^{-3}
Dynamic viscosity	0.001 Pa s
K_0	2.2×10^7, 2.2×10^6 and 2.2×10^5 Pa
K_0'	1.6, 7.6 and 10.6

Table 5: Parameters used for solving a compressible hydrostatic problem whose geometry is described in Figure 9.

The first test will simply compare different numerical solutions computed using different mesh element characteristic sizes at $K_0 = 2.2 \times 10^7$Pa, value at which the difference between the incompressible analytical solution and the weakly-compressible analytical solution is negligible. The results of this test can be found in figure 31. The value of K_0' chosen here is 7.6 (which is the value for water at ambient temperature).



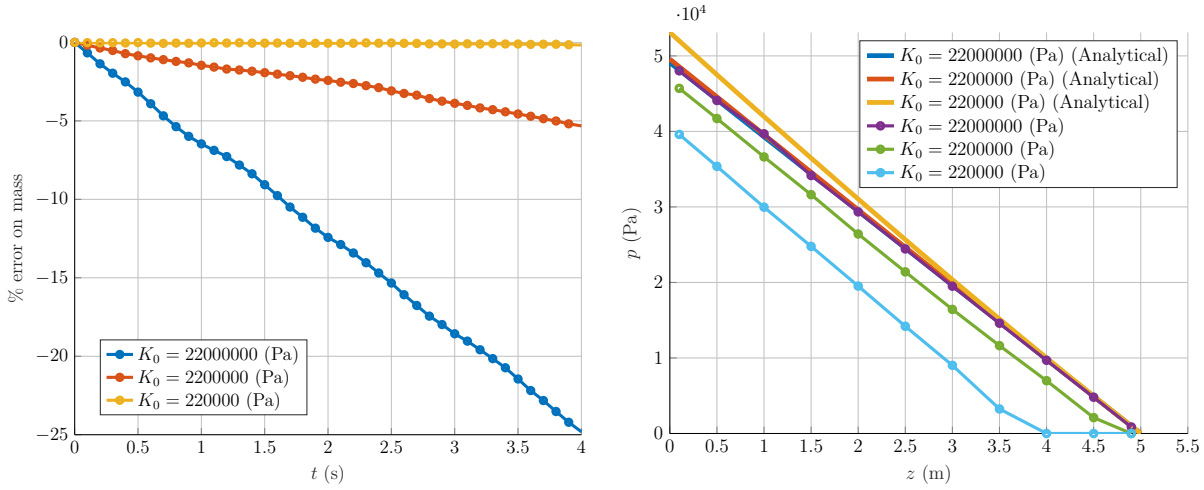
(a) Error on mass in function of time.

(b) Pressure along the y axis at the middle of the water column at $t = 4$ s.

Figure 31: Simulation results for four different mesh sizes in the hydrostatic case whose geometry is described in Figure 9 and parameters in Table 5 at $t = 4$ s, $K_0 = 2.2 \times 10^7$Pa and $K_0' = 7.6$.

The first comment to be made is that the mass is far better conserved than in the incompressible case. For $h_{char} = 1\,\text{m}$, after 4 s, the incompressible solver encountered an error on mass of 14% while the compressible solver only encountered an error on mass of 5 %. Secondly, the pressure is also far better approximated, the curve for $h_{char} = 1\,\text{m}$ is closer to the analytical curve in the compressible case than in the incompressible case, and the other curves are so close to the analytical curve in the compressible case that it is difficult to distinguish them.

The second test will compare the behaviour of the solution for multiple value of K_0, in order to know the range of values at which the solver gives a good solution (for $h_{char} = 0.125$ m). The results are presented in figure 32.



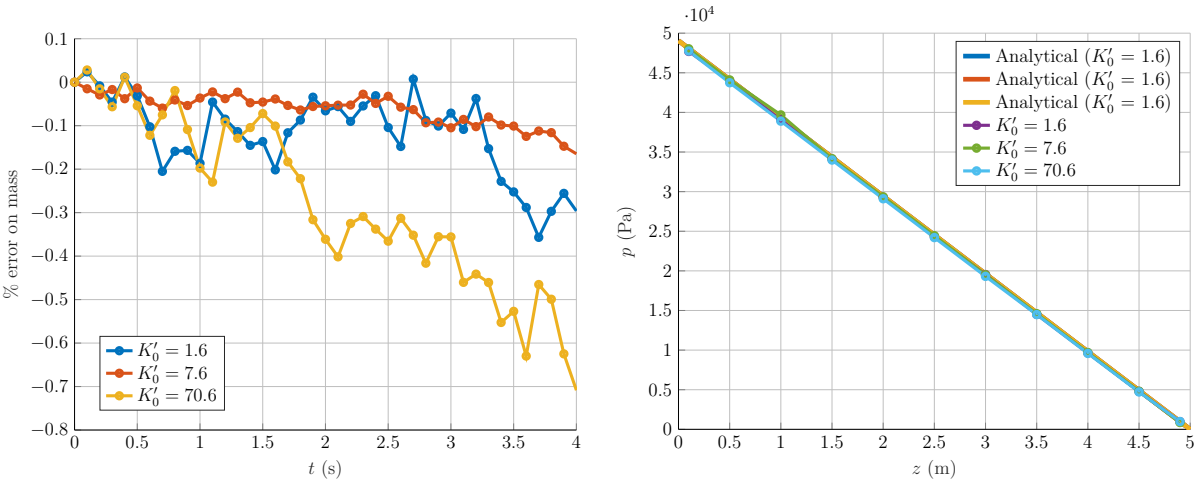
(a) Error on mass in function of time.

(b) Pressure along the y axis at the middle of the water column at $t = 4\,\text{s}$.

Figure 32: Simulation results for three different K_0 values in the hydrostatic case whose geometry is described in Figure 9 and parameters in Table 5 at $t = 4\,\text{s}$, $h_{char} = 0.125$ and $K_0' = 7.6$.

As it can be seen, it is difficult for the compressible solver to represent fluid with $K_0 = 2.2 \times 10^6$ and 2.2×10^5 Pa, while the curve of $K_0 = 2.2 \times 10^7$ matches its analytical curve as expected. Indeed the level of fluid decreases in the two lower K_0 cases, as it can be deducted from the mass figure and from the pressure figure which references 0 Pa when the fluid is on the free surface or if there is no fluid. The current approach seems able to correctly represent fluid not too far from incompressibility, which is the original objective in this work. The behaviour at lower K_0 should be explored further to extend the solver capabilities to more compressible fluid.

The last test will study the influence of the last parameter K_0^p on the solution. Since it had been identified that only fluid with high K_0 are represented correctly, this test will use a value of $K_0 = 2.2 \times 10^7$Pa. The results are presented on figure 33.



(a) Error on mass in function of time.

(b) Pressure along the y axis at the middle of the water column at $t = 4\,\mathrm{s}$.

Figure 33: Simulation results for three different K_0' values in the hydrostatic case whose geometry is described in Figure 9 and parameters in Table 5 at $t = 4\,\mathrm{s}$, $h_{char} = 0.125$ and $K_0 = 2.2 \times 10^7 \mathrm{Pa}$.

This last parameter does not seem to influence the mass evolution and the different pressure curves in Figure 33 are undistinguishable. It will thus be fix at $K_0' = 7.6$ for the rest of this work, value near 7 being the value used by the empirical Tait equation (Meduri [2019]).

2 Flow between two plates

The same problem as described in the previous part is considered, with the same geometry as in Figure 13. The purpose of this test is to see if the compressible solver developed in this work is able to correctly represent the parabola profile of the horizontal velocity between the two plates. Two values of K_0 will also be tested. The parameters used for the simulations can be found in Table 6.

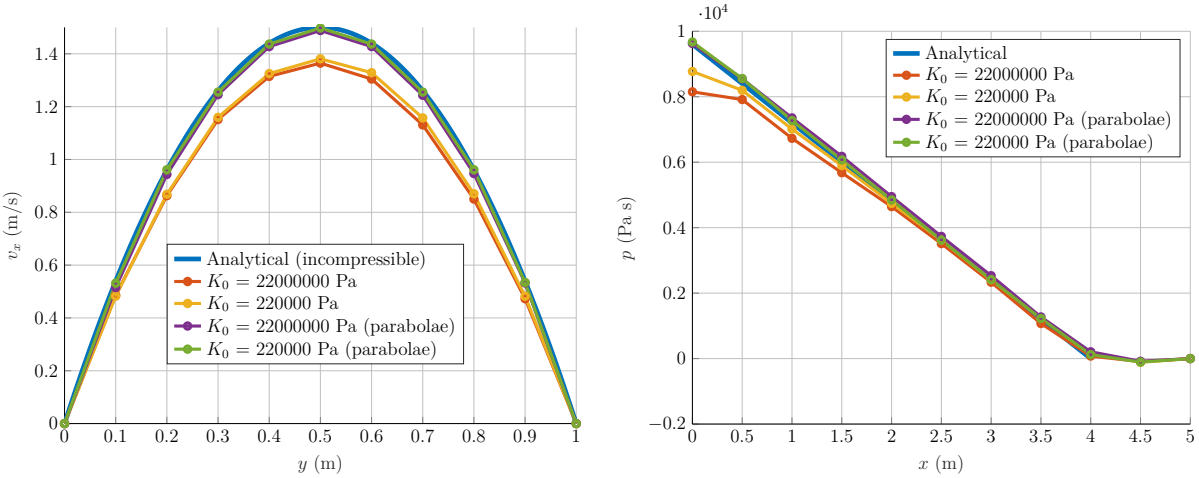
It should be noted that the continuity equation with explicit time derivative has to be used in this case, otherwise the elements near the Dirichlet boundary will not deform enough due to the presence of the boundary for the remeshing algorithm to add nodes at the center of those elements in order to simulate a flow input (the continuity equation without explicit time derivative seems too "strong" at conserving mass).

In the same way as in the previous part, both a uniform and a parabola profile for the Dirichlet boundary condition are tested. In the same way as in the previous part, if the fluid has enough time to flow, the velocity profile should converge to a steady-state solution. Here the horizontal velocity profile and the pressure profile in the middle of the plates are compared to the analytical solution of the incompressible steady-state problem.

h_{char}	0.1 m
α (alpha-shape)	1.2
γ (nodes removing)	0.7
ω (nodes adding)	0.7
Bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$	$[-1, -0.25, 5, 1.25]$
Gravity	0 kg m s^{-2}
Pressure imposed at free surface	yes
"Strong" continuity	no
Adaptive time step	yes
Security coefficient	0.2
Maximum Δt	0.005 s
Initial Δt	10^{-8} s
Simulation span	5 s
Density	1000 kg m^{-3}
Dynamic viscosity	200 Pa s
K_0	2.2×10^7, and 2.2×10^5 Pa
K_0'	7.6

Table 6: Parameters used for solving a compressible pipe problem whose geometry is described in Figure 13.

The results can be found in Figure 34. The analytical curve correspond to the steady-state solution of the incompressible problem.



(a) Horizontal component of the velocity at the function of y at $x = 3$ m.

(b) Pressure at $y = 0.5$ m.

Figure 34: Simulation results for the compressible pipe case whose geometry is described in Figure 13 and parameters in Table 6 at $t = 4$ s.

The pressure profile matches quite well the analytical solution of the incompressible problem for both K_0 when using a parabola Dirichlet boundary condition. When using a uniform boundary condition, the correspondence is good at the end of the pipe as

expected. The pressure does not evolve anymore out of the pipe as expected too. Regarding the horizontal velocity, a parabola-like profile is recovered in all the cases, and the numerical solution matches the solution of the incompressible steady-state problem when a parabola Dirichlet boundary condition is used as expected.

The most interesting results is that the velocity profile does not notably changes when dividing K_0 by 100. This means that one could use a lower K_0 in some cases and not obtain degraded results as it has been remarked in the previous section, which is great for compute time since, as said in subsection II.6.1, the K_0 parameter is the main parameter influencing the time step size in sub-sonic condition. However this parameter influences the pressure profile at the beginning of the plates in the case where a uniform Dirichlet boundary condition is used.

3 Sloshing

The same problem as described in the previous part is considered, with the same geometry as in Figure 17. The parameters used for the simulations can be found in Figure 7. Again the maximum height of the free surface is tracked. The parameter K_0 is set to 2.2×10^6 Pa, and not to 2.2×10^7 Pa in order to spare some computing time.

h_{char}	0.02 and 0.01 m
α (alpha-shape)	1.2
γ (nodes removing)	0.7
ω (nodes adding)	0.7
Bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$	$[-1, -0.25, 5, 1?25]$
Gravity	9.81 kg m s^{-2}
Pressure imposed at free surface	yes
"Strong" continuity	yes
Adaptive time step	yes
Security coefficient	0.2
Maximum Δt	0.01 s
Initial Δt	10^{-8} s
Simulation span	4 s
Density	1 kg m^{-3}
Dynamic viscosity	0.01 Pa s
K_0	2.2×10^6 Pa
K_0'	7.6

Table 7: Parameters used for solving a compressible sloshing problem whose geometry is described in Figure 17.

The results are presented in Figure 35.

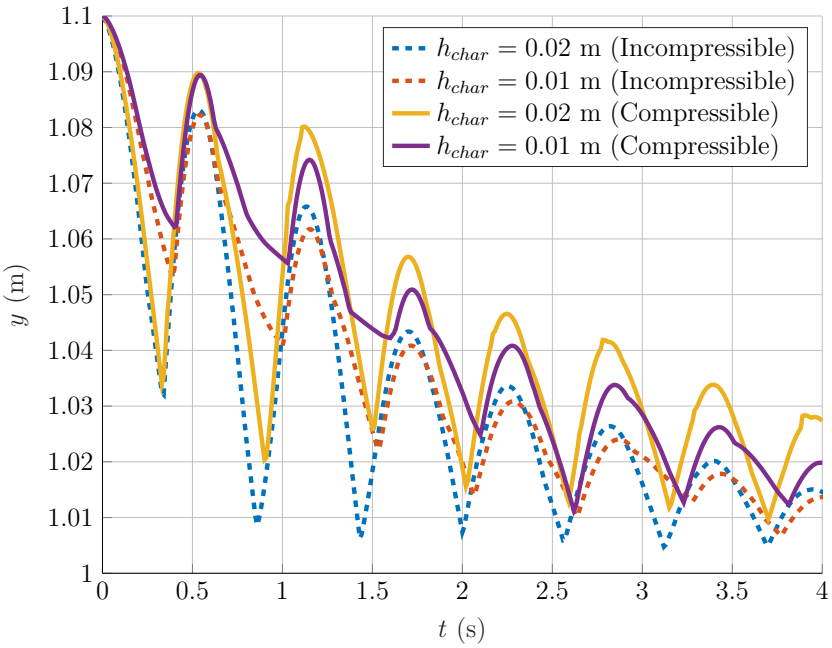


Figure 35: Simulation results for the compressible sloshing case whose geometry is described in Figure 17 and parameters in Table 7.

While the positions of the local maximum seems to be the same for both solvers, the maximum amplitude decreases slower for the compressible solver than for the incompressible solver. Testing the behaviour of this problem for varying value of K_0 greater that 2.2×10^6 could be really interesting, this could unfortunately not been done in time for this report.

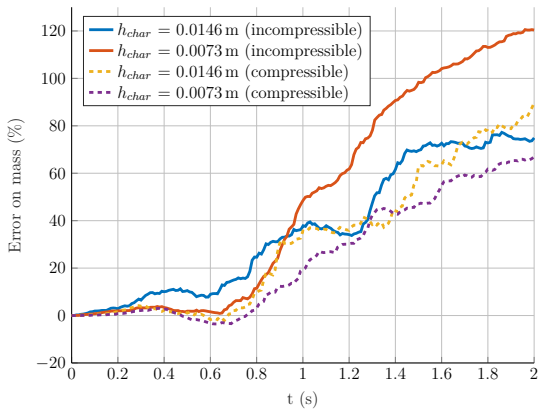
4 Dam break

The same problem as described in the previous part is considered, with the same geometry as in Figure 20. The parameters used for the simulations can be found in Figure 8.

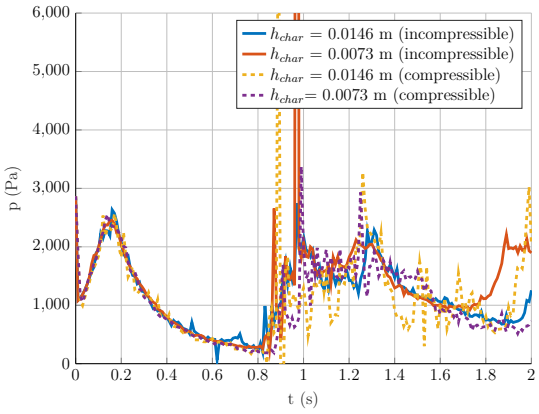
h_{char}	0.02 and 0.01 m
α (alpha-shape)	1.2
γ (nodes removing)	0.7
ω (nodes adding)	0.7
Bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$	$[-0.01, -0.01, 0.594, 100]$
Gravity	9.81 kg m s^{-2}
Pressure imposed at free surface	yes
"Strong" continuity	yes
Adaptive time step	yes
Security coefficient	0.1
Maximum Δt	0.005 s
Initial Δt	10^{-8} s
Simulation span	2 s
Density	1000 kg m^{-3}
Dynamic viscosity	0.001 Pa s
K_0	2.2×10^6 Pa
K_0'	7.6

Table 8: Parameters used for solving a compressible dam break problem whose geometry is described in Figure 20.

The total error on mass as well as the pressure at the left corner with respect to time are studied. The results are presented on Figure 36, and a graphical comparison between the incompressible and compressible solver developed in this work can be found in Figure 37.



(a) Error on mass as a function of time.



(b) Pressure at the leftmost as a function of time.

Figure 36: Simulation results for the compressible dam break case whose geometry is described in Figure 20 and parameters in Table 8.

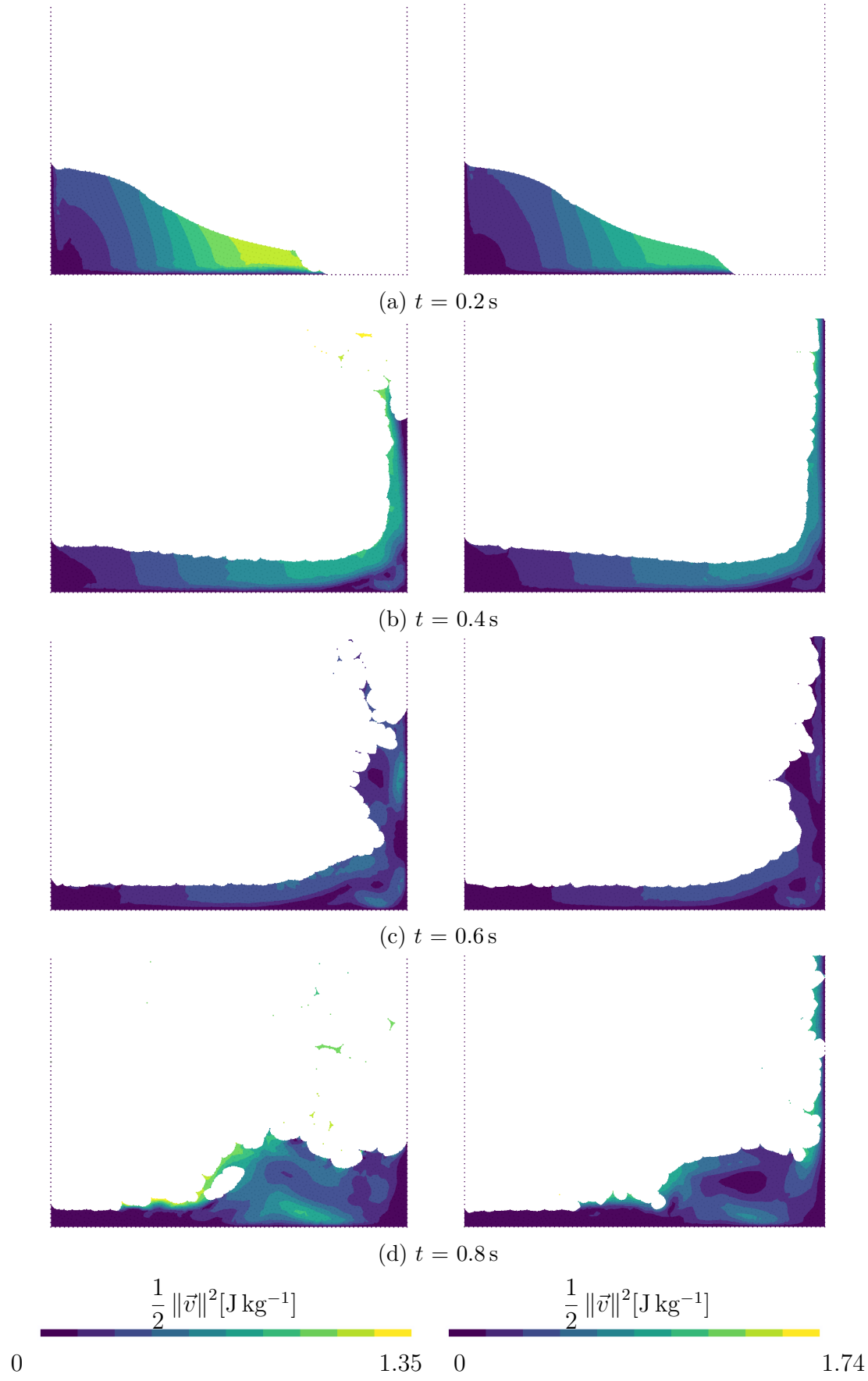


Figure 37: Comparison of the dam break flow between the incompressible solver (left) and the compressible solver (right). The geometry is described in Figure 20 and the parameters in Table 4 and 8.

Firstly the increase of mass is smaller in the compressible solver (up to 80 % of increase) than in the incompressible solver (up to 120 % of increase). Concerning the pressure results, those from the compressible solver seems more noisy in the second part of the flow (after 0.8 s while the results in the first part of the flow are really similar). A more deep study of this test should be done with varying K_0 to see if this parameter has the same impact on the solution as for the flow between two plates problem.

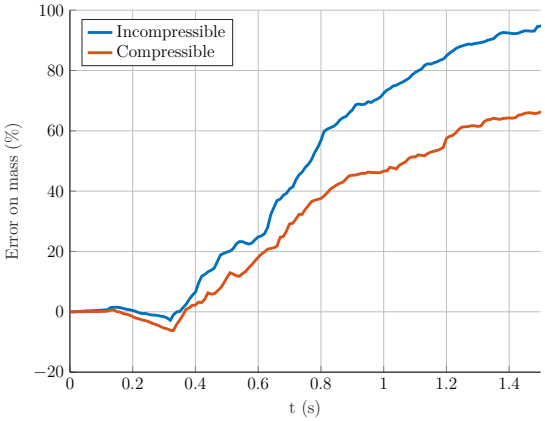
Concerning the graphical results in Figure 37, the overall shape of the solution seems the same for both solvers. It should be noted however that the solution from the compressible solver seems to stick more to the right wall than for the incompressible solver. Indeed at time $t = 0.4$ and 0.6 s the fluid keeps going up along the wall for the compressible solver instead of being partially projected in an oblique direction for the incompressible solver. At 0.9 s no more fluid sticks to the wall for the incompressible solver while some fluid remains up on the wall for the compressible solver. The closing wave is also further to the left at that time for the incompressible solver.

5 Dam break with obstacle

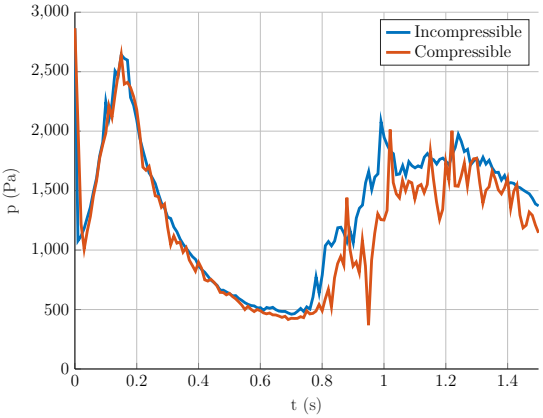
This problem is again the same as in the incompressible part. The geometry is described in Figure 24 and the parameters in Table 8, but $h_{char} = 0.006$ m and the maximum time step is 0.0005 s. The error on mass and the pressure at the leftmost corner and rightmost corner with respect to time will be studied. The results are presented in Figure 38.

The error on mass is smaller for the compressible solver (up to 66 % of increase) than for the incompressible solver (up to 95 % of increase). There is a notable decrease of mass of 6.2 % at 0.33 s for the compressible solver. Regarding the pressure at the leftmost corner, the results are the same, but one can remark that the solution from the compressible solver is more noisy than the solution from the incompressible solver, and that the curve of the first solver seems a little bit below the curve of the second solver after 0.8 s. Regarding the pressure at the rightmost corner, one can see that the fluid seems to arrive at this corner at the same time for both solvers around 0.4 s. After that time, it is clearly visible in the figure than the solution from the compressible solver is really noisy is comparison to the solution from the incompressible one, but is remaining in the same order of magnitude.

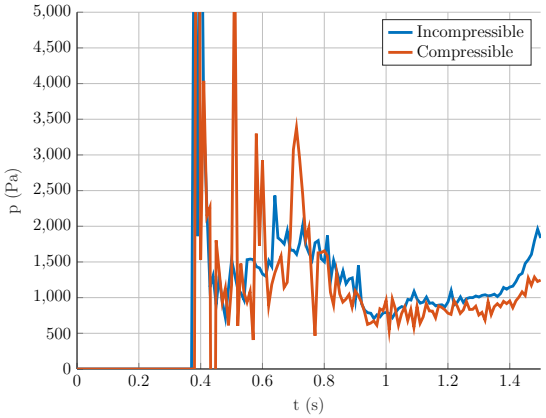
A graphical comparison between the solution from this work's compressible solver and this work's incompressible solver is presented in figure 39. Globally, the two solutions behave quite the same way in the range displayed. There are some differences in the shape of the free surface of the tail that is appearing when the the fluid hits the obstacle. Those differences comes certainly from the difference of solver used conjugated with the remeshing algorithm



(a) Error on mass as a function of time.



(b) Pressure at the leftmost corner as a function of time.



(c) Pressure at the rightmost corner as a function of time.

Figure 38: Simulation results for the dam break with obstacle case whose geometry is described in Figure 24 and parameters in Table 4.

.

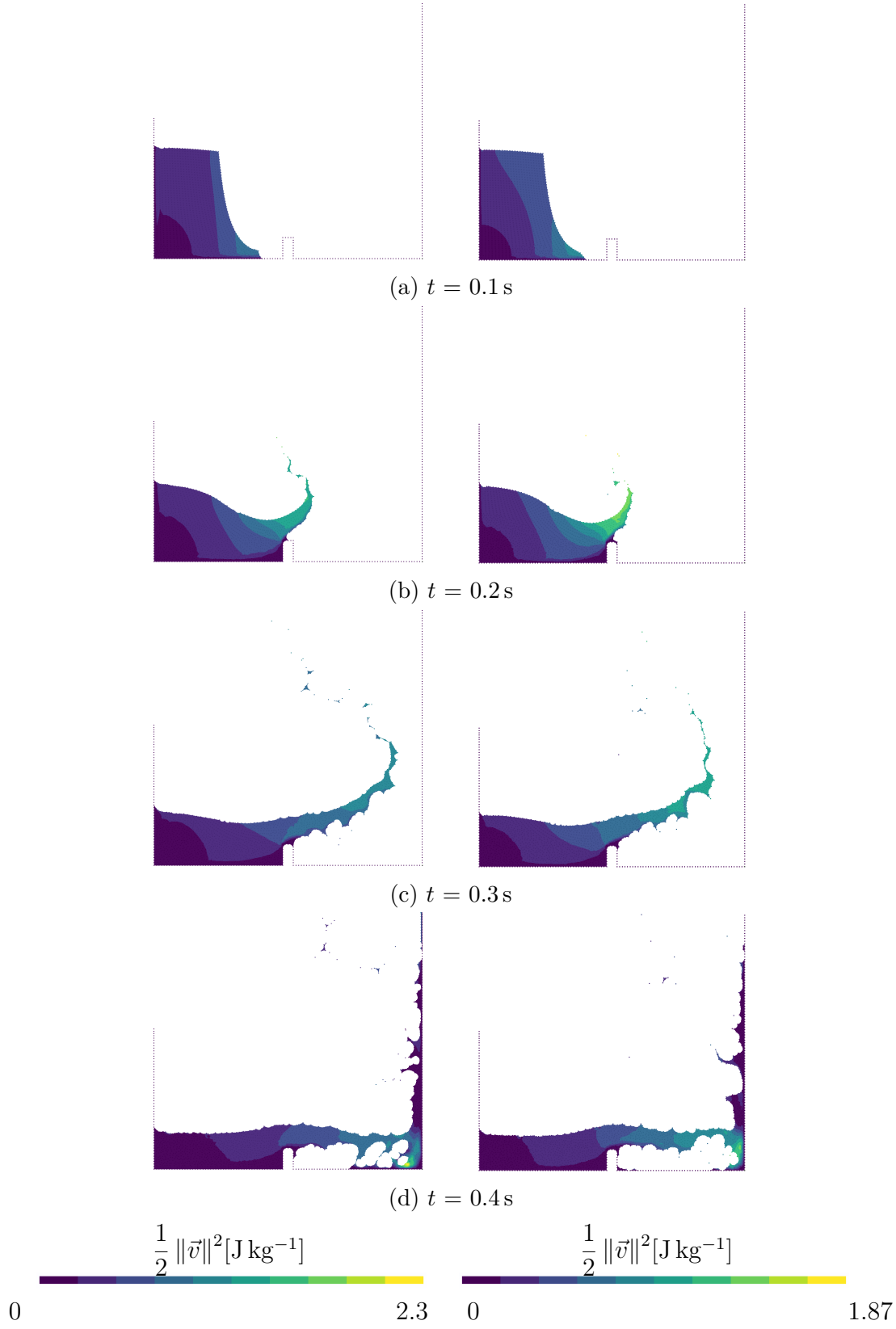


Figure 39: Comparison between the incompressible solver and the compressible solver for the dam break with obstacle case. The geometry can be found in 20 and the parameters in 4 and 8.

.

6 Performance considerations

A comparison of the typical time to compute the solution for one time step in the compressible solver and the time of remeshing (when it happens) is provided in figure 40. The problem used in this figure is the dam break problem from section 4, except that the time step used was 0.0001 s. Only one thread is used.

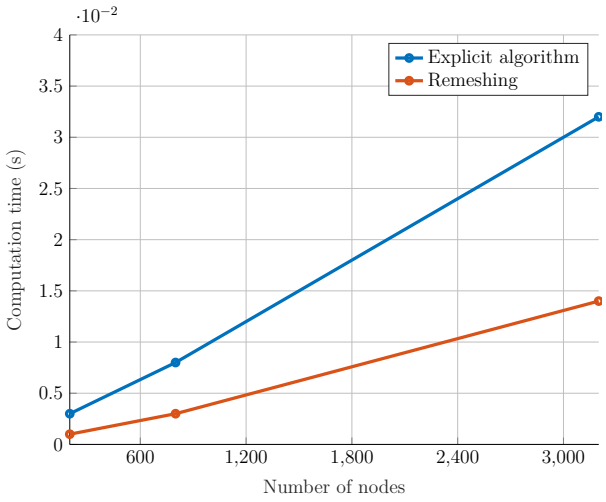


Figure 40: Comparison of the computation time between the explicit algorithm and the remeshing (when it is triggered) for the dam break problem described in Figure 20 and Table 8.

As expected the time to compute a time step in explicit is really smaller than the time to compute a time step in implicit, which makes the computation time of the explicit time step of the same order of magnitude as the time to perform the remeshing.

6.1 Incompressible and compressible solving time comparison

A comparison of the time to compute the solution for this work's incompressible solver and this work's compressible solver is provided in Figure 40. The problem used in this figure is the dam break problem from section 4, except that the time step used was 0.0001 s. Only one thread is used. The compressible solver seems faster and it also seems that the computation time grows faster with the number of nodes for the incompressible solver than for the compressible solver. It should be noted however that a lot of parameters can influence the computation time of both solvers such that those results should be taken with caution.

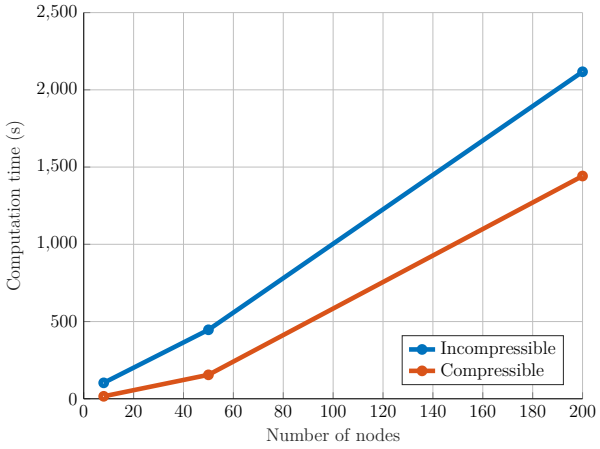
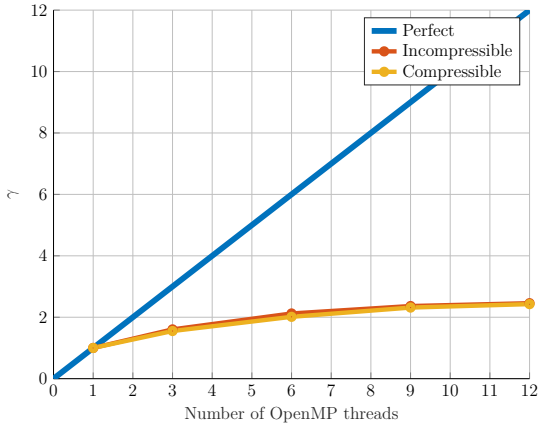


Figure 41: Comparison of the computation time of the incompressible and compressible solver in the dam break problem described in Figure 20 and in Table 4 8.

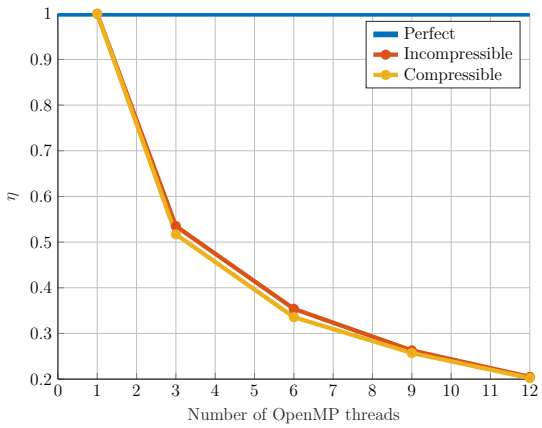
6.2 Parallelization

The incompressible and compressible solvers are compared in the same way as in the previous part in Figure 30.

Surprisingly, the two solvers behave the same way regarding to parallel performance. This not expected as the compressible solver does not contain a part not parallelized as the incompressible solver. No explanation can be currently provided for this for the compressible solver, such that it should be investigated further.



(a) Speed-up ration as a function of the number of threads.



(b) Parallel efficiency as a function of the number of threads.

Figure 42: Comparison of the parallelization of the incompressible and compressible solver.

This work's implicit compressible solver has been compared to this work's explicit compressible solver and analytical solutions. Both solver seems overall to give the same

results. The study has brought to light however that the compressible solver sometimes gives more noisy solution than the incompressible and has highlighted the particular problem of simulating fluid too far from incompressibility with the current method. This could be a particular subject of interest if one is interested in simulating such kind of flow.

Part V

3D case studies

Contents

In this part, two 3D tests are performed. Even if the program is able to use both the incompressible and compressible solver for 3D simulations, all the presented simulations use the compressible solver. This is because the incompressible solver takes far too much computation time to solve those problems. For example, in the pipe case showcased below, the incompressible solver took 48 h to solve 0.9 s of flow, while the compressible solver manage to solve all the 5 s of the flow during that time, which justifies the development of a compressible explicit solver in order to compute 3D problems in a reasonable amount of time.

1 Fluid in cylindrical pipe

A cylindrical pipe is considered and its geometry can be found in Figure 43. A Dirichlet boundary condition with $v_x = 1\,\mathrm{m\,s^{-1}}$ is placed at the left entrance of fluid. A small initial cylindrical volume of fluid is also placed next to this boundary, in order to allow the addition of nodes by the remeshing algorithm as the elements of this volume will be deformed due to the presence of the Dirichlet boundary.

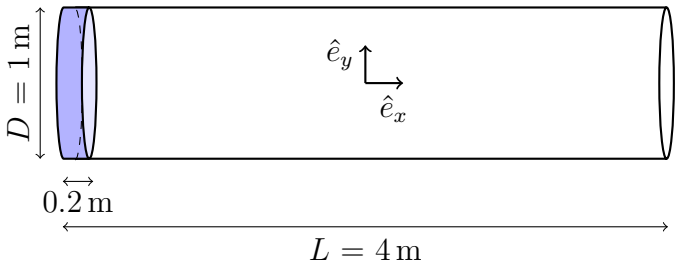


Figure 43: Initial geometry of the pipe problem.

If the fluid flows for long enough in a long enough pipe, one could expect that the transient solution converges to a steady-state solution. For an incompressible fluid, it is:

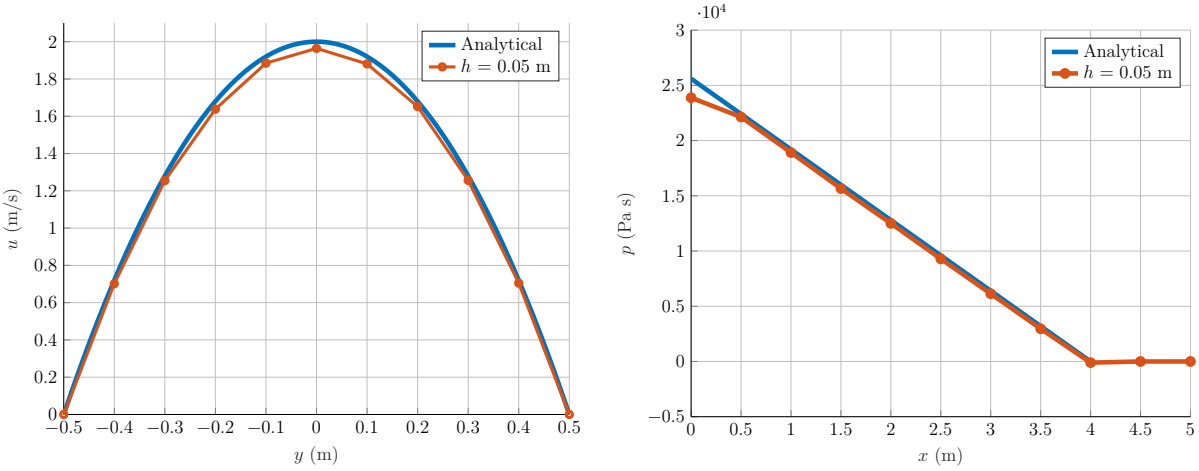
$$\begin{aligned} u &= \frac{2Q}{\pi R^4}(R^2 - r^2) \\ \frac{dp}{dx} &= -\frac{8\mu Q}{\pi R^4} \end{aligned}, \tag{V.1}$$

where Q is the volume flow rate, R is the pipe radius, μ is the dynamic viscosity and r is the radial distance from the center of the pipe. The parameters are summed up in Table 9. A sufficiently high value of the dynamic viscosity has been chosen in order for the flow to become fully developed soon enough.

h_{char}	0.05 m
α (alpha-shape)	1.4
γ (nodes removing)	0.7
ω (nodes adding)	0.35
Bounding box $(x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max})$	$[-1, -1, -0.5, 5, 1, 1.5]$
Gravity	0 kg m^2 s^{-1}
Pressure imposed at free surface	yes
"Strong" continuity	false
Adaptive time step	yes
Security coeffeficent	0.05
Maximum Δt	0.01 s
Initial Δt	10^{-8} s
Simulation span	5 s
Density	1000 kg^3 m^{-1}
Dynamic viscosity	0.001 Pa s
K_0	2.2×10^6 Pa
K_0'	7.6

Table 9: Parameters used for solving a compressible 3D pipe problem whose geometry is described in Figure 43.

The results of the simulation can be seem in Figure 44.

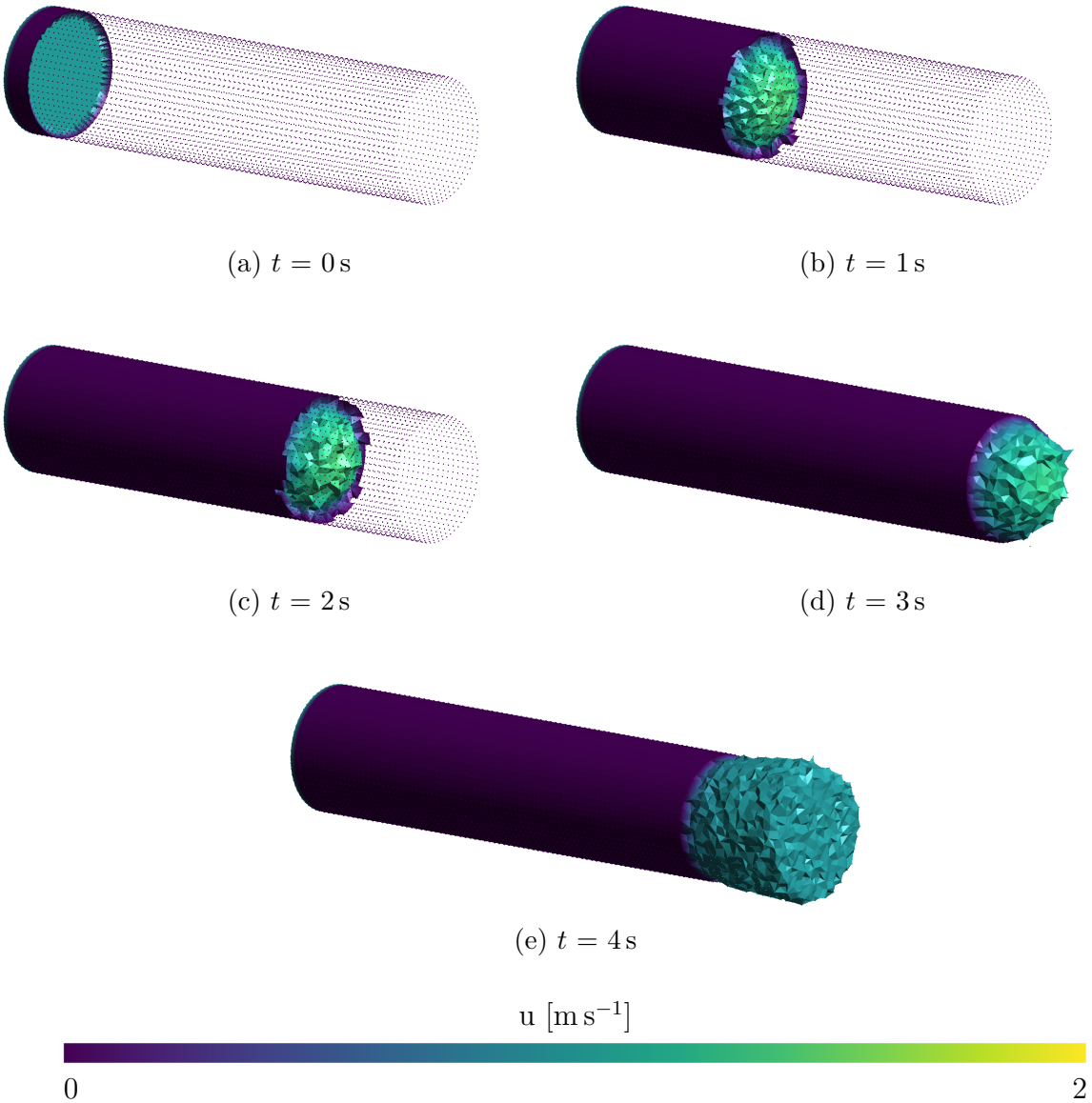


(a) Horizontal component of the velocity as a function of y at $x = 3$ m.

(b) Pressure along the pipe center.

Figure 44: Simulation results for the 3D compressible pipe case whose geometry is described in Figure 43 and parameters in Table 9 at $t = 2.7$ s.

The velocity profile matches well the incompressible analytical solution. The pressure profile also matches well that solution, excepted at the pipe entrance, due to the transient behaviour of the flow there. Outside of the pipe, pressure does not evolve anymore as expected. A graphical representation of the solution is provided in Figure 45 and Figure 46.



(a) $t = 0\,\mathrm{s}$ (b) $t = 1\,\mathrm{s}$

(c) $t = 2\,\mathrm{s}$ (d) $t = 3\,\mathrm{s}$

(e) $t = 4\,\mathrm{s}$

Figure 45: Horizontal component of the velocity at multiple time stations at $h_{char} = 0.05\,\mathrm{m}$ in the 3D pipe case whose geometry is described in Figure 43 and parameters in Table 9.

As it can be seen in Figure 46, the parabola velocity profile switches to a more uniform velocity profile when the fluid goes out of the pipe, as friction in the fluid smooths out the gradient of velocity since the pipe walls are not there anymore.

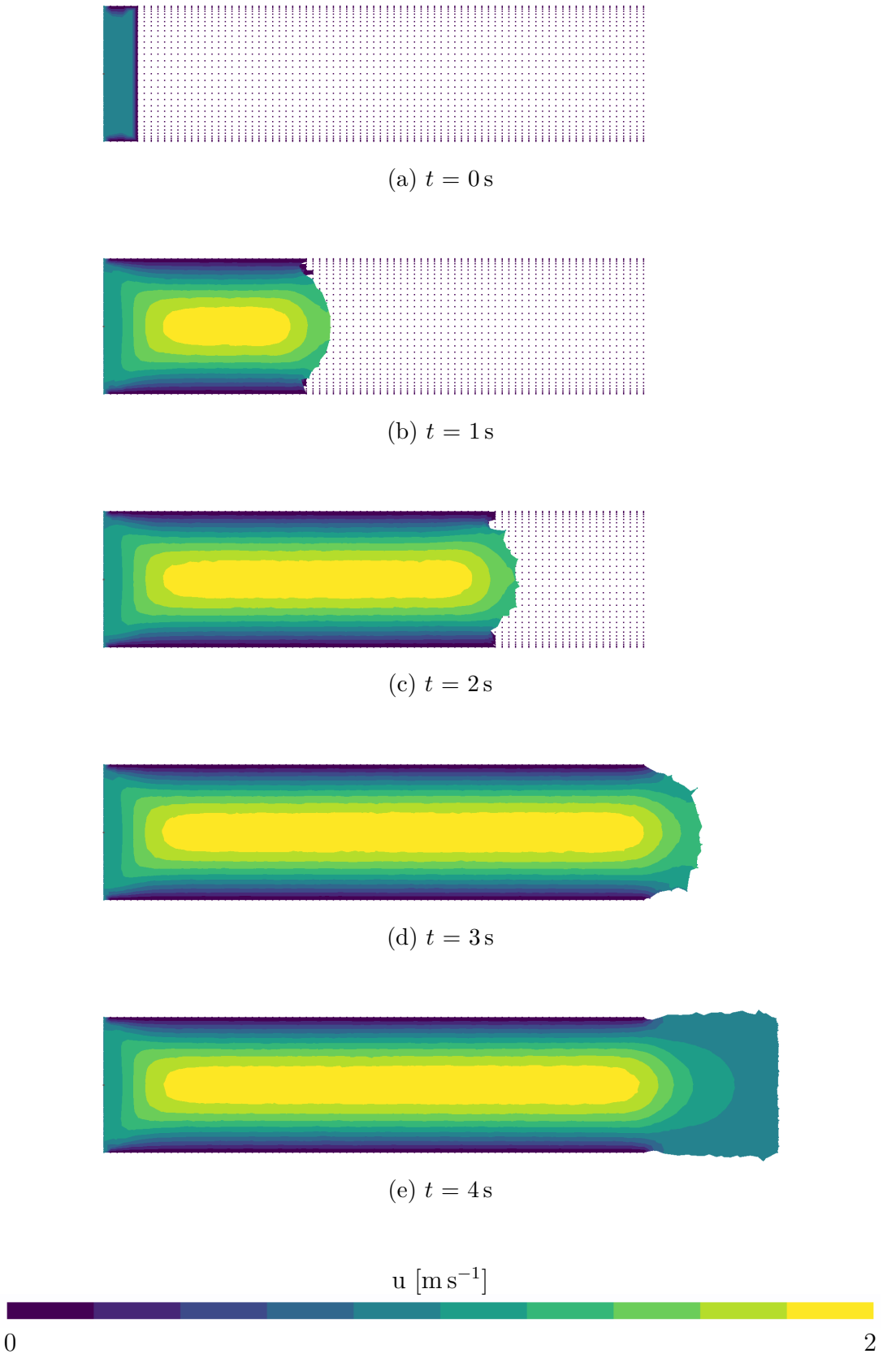


Figure 46: Horizontal component of the velocity at multiple time stations in the 3D pipe case whose geometry is described in Figure 43 and parameters in Table 9, cutted view at half of the pipe.

2 Dam break

A 3D dam break similar to the 2D dam break presented before is considered. Its geometry can be found in Figure 47 and the parameters used in Table 10. The results are presented in Figure 48 and 49 .

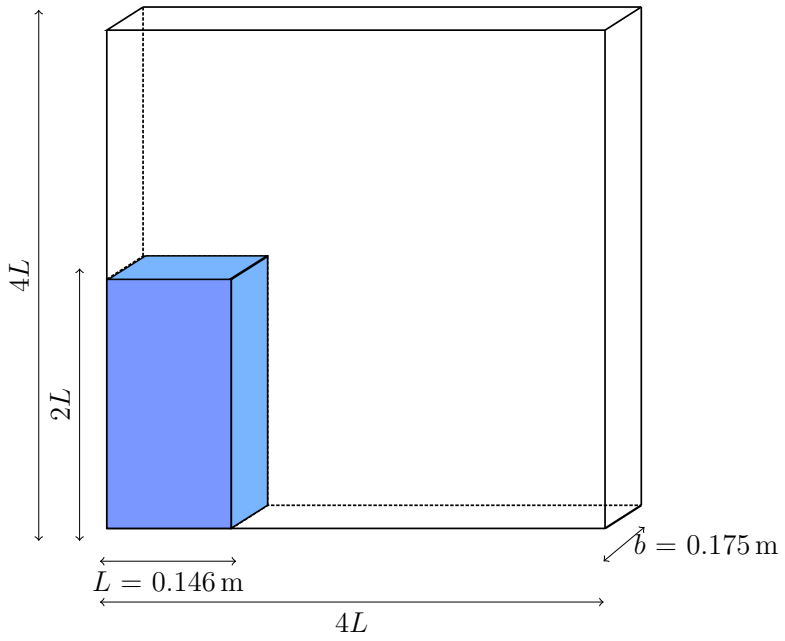


Figure 47: Initial geometry of the 3D dam break problem.

h_{char}	0.0073 m
α (alpha-shape)	1.2
γ (nodes removing)	0.7
ω (nodes adding)	0.37
Bounding box $(x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max})$	$[-1, -1, -1, 1.584, 1.175, 100]$
Gravity	9.81 $\mathrm{kg\,m^2\,s^{-1}}$
Pressure imposed at free surface	yes
"Strong" continuity	yes
Adaptive time step	yes
Security coefficient	0.05
Maximum Δt	0.01 s
Initial Δt	10^{-8} s
Simulation span	1 s
Density	1000 $\mathrm{kg^3\,m^{-1}}$
Dynamic viscosity	200 Pa s
K_0	2.2×10^6 Pa
K_0'	7.6

Table 10: Parameters used for solving a compressible 3D dam break problem whose geometry is described in Figure 47

.

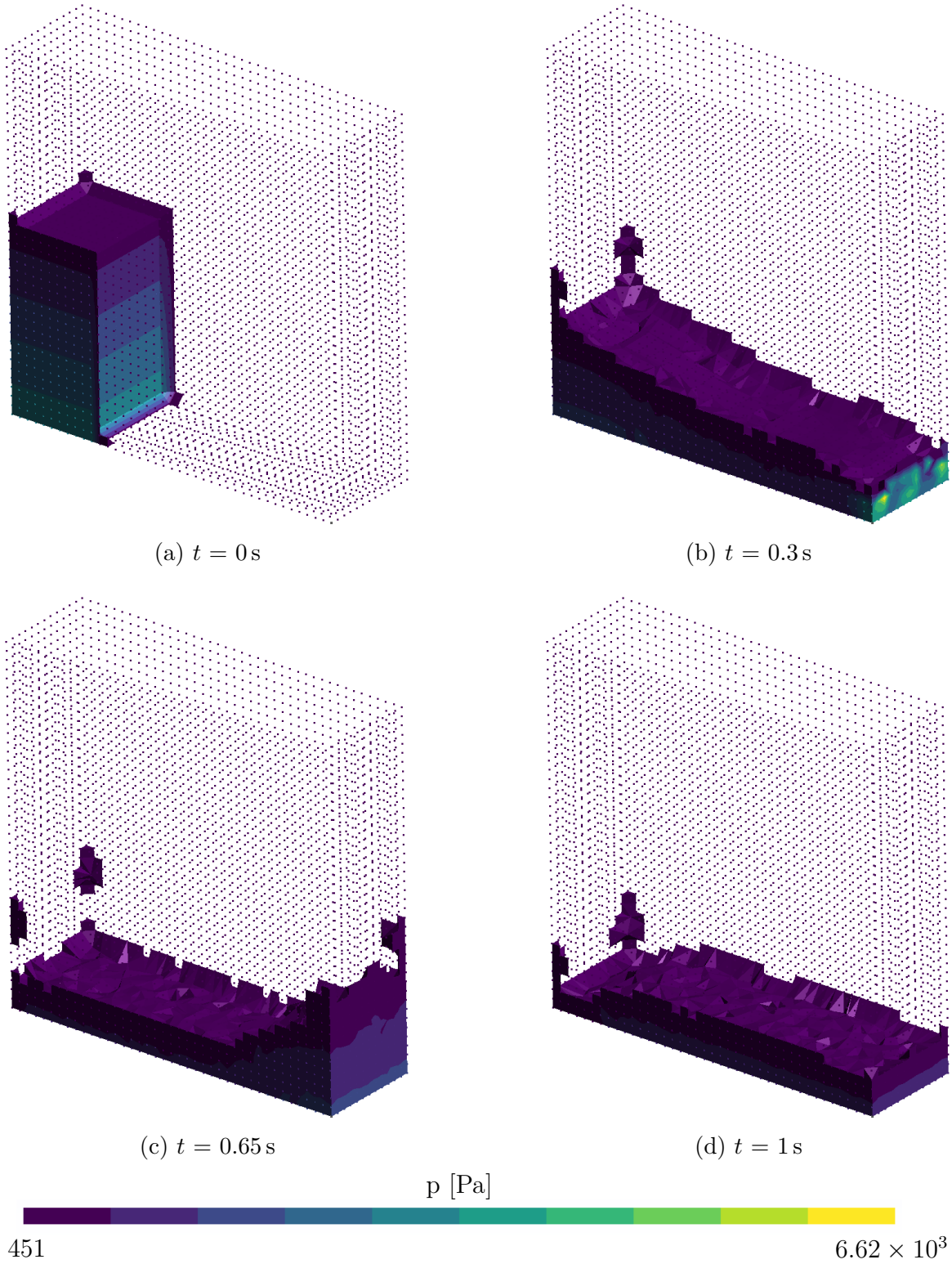


Figure 48: Pressure at multiple time stations in the 3D dam break case whose geometry is described in Figure 47 and parameters in Table 10

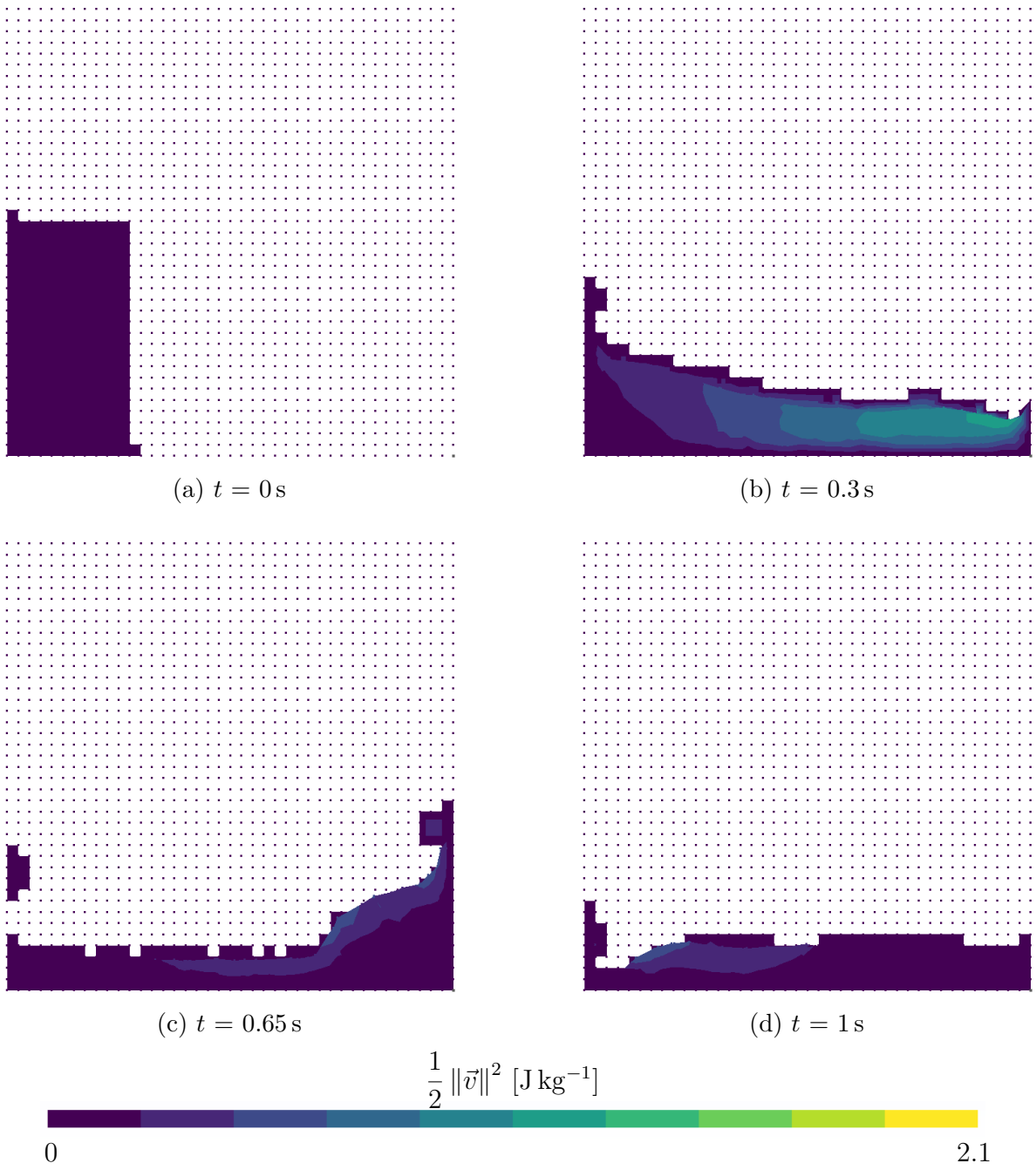


Figure 49: $\frac{1}{2}\|\vec{v}\|^2$ at multiple time step in the 3D dam break case whose geometry is described in Figure 47 and parameters in Table 10

While the global expected behaviour of a dam break is presented, the discretization in this example is not fine enough to capture all the features of the flow, as such a simulation could not be done in time for this work.

3 Performance considerations

A comparison of the typical time to compute the solution for one time step in the compressible solver and the time of remeshing (when it happens) is provided in figure 50. The problem used is the pipe problem firstly described in this part

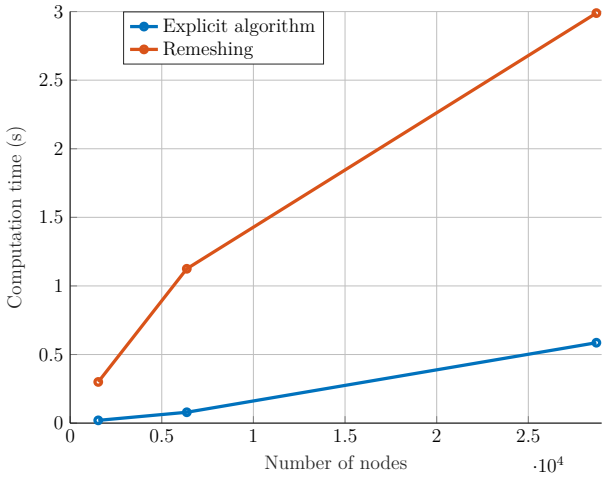


Figure 50: Repartition of the computation time in the 3D pipe problem described in Figure 43 and in Table 9 for one time step.

As it can be seen, in 3D the time to perform the remeshing is greater than the time to compute one explicit time step and seems to grow faster with the number of nodes. This should thus be a major concern if simulating 3D problems with a high number of nodes is needed.

In this part it has been showed that the compressible solver developed in this work is able to simulate accurately 3D flows. However there is still some research work to do, notably on the acceptable range of remeshing parameters in 3D as well as the time of remeshing in 3D which could be a major bottleneck for further uses.

Conclusions and future work

In this work, a weakly-compressible solver for the Navier–Stokes equations has been developed in order to simulate 3D flows. To do so, the strong form and weak form of the equations in both the compressible case and the incompressible case have been recalled from continuum mechanics. The PFEM method has then been introduced, and a solution to perform the Delaunay triangulation and the alpha-shape algorithm in 3D, necessary steps to produce a mesh on which computations can be made, has been found by using CGAL. Based on the work in Cerquaglia [2019] and Meduri [2019], the spatial discretized equations and the time integration methods used have then been introduced.

In a second time, the 2D incompressible solver developed in this work has been compared to analytical solutions and to the solver developed by Marco-Lucio Cerquaglia (Cerquaglia [2019]). Although the solver seems able to correctly simulate those flows, a problem of too big increase of mass has been identified. After that the 2D compressible solver developed in this work has been compared to the 2D incompressible solver from this work. The compressible solver is able to reproduce the results from the incompressible solver. However, the results are in some case more noisy for the compressible solver, and this solver seems unable to correctly simulate fluid too far from incompressibility.

At last, the 3D compressible solver has been tested using a cylindrical pipe problem. The numerical solution being close to the analytical one, this show that the solver is able to simulate 3D flows.

There is still a lot of room for improvements in the solvers developed in this work, and some ideas to do so are presented hereafter:

- The parameter h_{char} has to be specified in the parameters file, while it could be computed from the `.msh` file in order to minimize the probability of an error from mismatching h_{char} between that file and the parameters file.
- A space varying h_{char} could be introduced easily using `Lua` (de Figueiredo et al. [2020]) in order to tune the density of nodes where the solution is the more interesting to compute.
- The remeshing algorithm currently performs three triangulations per remeshing, because the functions which add, delete nodes or check if some nodes are outside the bounding box do not ensure that the state of the `Mesh` class remains consistent after being executed. While it is not really a problem in the incompressible 2D solver, it can be really problematic in the compressible 3D solver where the time to perform the remeshing becomes a major concern.
- The Delaunay triangulation and alpha-shape algorithm are currently not parallelized (because it seems CGAL does not have parallelized algorithm for them). Since the 3D remeshing takes a lot time, it could be interesting to investigate if those algorithm inside CGAL could benefit from parallelization.

- The problem of generation of slivers has been solved in a quite abrupt way. It is not impossible that some really small actual fluid elements might be deleted. The problem of slivers should thus be investigated further. Some more 3D specific mesh optimization algorithms might also be needed.

- A more precise study should be done on the remeshing constants (α, γ and ω) to see their influence on the mass conservation during the simulations (in both 2D and 3D).

- The solver does not currently handle imposed surface traction at all. This is mainly due to the fact that currently the mesh does not store any geometrical information about the edges (Jacobian determinant, and especially curvature computation). This prevents taking into account any surface tension effect but could be easily added.

- The origin of the "spikes" appearing in the pressure graph in the dam break problems should investigated. The addition and removal of nodes by the remeshing algorithm could be a candidate as a cause of this effect.

- Technically speaking, there is nothing in the current code that prevents one to use moving wall, the `Lua` code of boundary conditions allowing to set any velocity for wall nodes . However, this has not been tested in this work due to lack of time. It could thus be interesting to see if it works or not.

- The solver does not support free slip boundary conditions, which can be useful at high Reynolds number (Cerquaglia [2019]).

- Other time integration schemes could be used in the incompressible and compressible solvers, which currently use a backward-Euler and a central difference scheme respectively.

- Other convergence criteria from Cerquaglia [2019] could be implemented for the non-linear algorithm in the incompressible solver.

- The poor parallel performance of the compressible solver should be investigated. In the incompressible solver, different non-linear algorithms could be tested (e.g. Newton-Raphson) and one could try to use an iterative solver instead of a solver using LU decomposition to solve the algebraic system.

- The compressible solver does not behave really well for too small value of the parameter K_0. While this was not a problem in this work as the compressible solver was still used for simulating flows with fluid near incompressibility, this should be investigated if someone is interested in solving more compressible flows.

- Different equations of state could be added in the compressible solver. However this has an interest only if the low K_0 problem is solved first.

Annexes

A Mesh file

The two solvers developed in this work use a `.msh` as input file for the initial set of nodes. This file should be generated by the `4.5` version of `Gmsh` (Geuzaine and Remacle [2009]).

In 2D, each curve and surface must have a *physical curve* and a *physical surface*, which is a tool provided by `Gmsh` to identify from which curves or surfaces the nodes come from once inside the solver (but multiple curves/surfaces can be in the same *physical curve*. The same should be done in 3D for surfaces and volumes, using *physical surfaces* and *physical volumes*.

The name of those now denominated physical objects will be used by the solver to know which `Lua` function to call to get the boundary and initial conditions.

B Boundary conditions implementation

A typical `Lua` file for the boundary and initial conditions for the problem from section III.2 can be seen in Figure 51.

```
function initFluid(pos)
    return {-6*pos[2]*(pos[2]-1), 0, 0}, false
end

function initBoundary(pos)
    return {0, 0, 0}, false
end

function initFluidInput(pos)
    return {-6*pos[2]*(pos[2]-1), 0, 0}, true
end

function Boundary(pos, initPos, t)
    return {0, 0}
end

function FluidInput(pos, initPos, t)
    return {-6*pos[2]*(pos[2]-1), 0}
end
```

Figure 51: Example of a Lua boundary and initial condition file for an incompressible problem.

Each physical object must provide a function named `initNameOfPhysicalObject`(e.g. `initFluid`, `initFluidInput`, `initBoundary` in the example of Figure 51) , which takes the position of the nodes inside the physical object as argument. This functions is then

used to initialize the states of those nodes (e.g. v_x, v_y, p, etc) at the beginning of the program.

Moreover, each boundary must provide a function named `nameOfPhysicalObject`, which takes the current position of nodes, the initial position of the nodes of the boundary and the current time of the simulation as argument. Those functions are used by the solver to impose the correct value of the for velocity for the wall and Dirichlet boundary conditions. In order to differentiate a wall condition from a fluid input, a boolean which is true in the last case is returned in the `initNameOfPhysicalObject` functions.

It can be seen in the example that the velocity of the *physical curve* "FluidInput" and *physical surface* "Fluid" are initialized to a parabola profile, respectively by function `initFluidInput` and `initFluid`. The "FluidInput" boundary is a Dirichlet boundary, as denoted by the returned value `true` in the function `initFluidInput`. On the other hand the *physical curve* "Boundary" represents a fixed wall, as denoted by the returned value `false` in the function `initBoundary`.

C How to use the executable and its JSON parameter file ?

The command to execute the program is :

```
pfem path_to_json_file path_to_msh_file
```

The `JSON` file will contain all the parameters needed for the simulation. A typical example of this file for the problem described in section IV.2 can be found in Figure 52

```
{
    "ProblemType": "Compressible",
    "Remeshing": {
        "hchar": 0.1,
        "alpha": 1.2,
        "omega": 0.7,
        "gamma": 0.7,
        "boundingBox": [-1, -0.25, 5, 1.25]
    },
    "Solver": {
        "gravity": 0,
        "strongPAtFS": true,
        "strongContinuity": false,
        "Time": {
            "adaptDT": true,
            "securityCoeff": 0.2,
            "maxDT": 0.005,
            "initialDT": 1e-8,
            "endTime": 5.1
        },
        "Fluid": {
            "rho0": 1000,
            "mu": 200,
            "K0": 2200000,
            "K0prime": 7.6,
            "pInfty": 0
        },
        "IBCs": "../../examples/2D/pipe/IBC_Comp.lua",
        "Extractors" : [
            {
                "type": "Point",
                "outputFile": "line_h_0.025.txt",
                "timeBetweenWriting": 0.1,
                "stateToWrite": 0,
                "points": [[3, 0.5], [3, 0.6]]
            },
            {
                "type": "Point",
                "outputFile": "line_p.txt",
                "timeBetweenWriting": 0.1,
                "stateToWrite": 2,
                "points": [[1.5, 0.5], [2, 0.5], [2.5, 0.5]]
            },
            {
                "type": "GMSH",
                "outputFile": "results.msh",
                "timeBetweenWriting": 0.2,
                "whatToWrite": ["u"],
                "writeAs": "NodesElements"
            }
        ]
    },
    "verboseOutput": false
}
```

Figure 52: Example of a JSON parameters file for a compressible problem.

The writing of such a parameter file is quite easy, and the potential errors can be found quite easily.

D How to use the project from python ?

The project generates a python module thanks to SWIG (Beazley et al. [2020]). The library can then be called and the parameters set by filling python objects which are sent to the solver's constructors. An example of such a python file launching a simulation for the problem from section III.2 can be seen in Figure 53

```
from pfemSolverw import MeshCreateInfo
from pfemSolverw import SolverCreateInfo
from pfemSolverw import SolverIncompCreateInfo
from pfemSolverw import SolverIncompressible
from pfemSolverw import VectorDouble
from pfemSolverw import VectorString
from pfemSolverw import VectorVectorDouble


meshInfos = MeshCreateInfo()
meshInfos.hchar = 0.1
meshInfos.alpha = 1.2
meshInfos.gamma = 0.7
meshInfos.omega = 0.7
boundingBox = VectorDouble(4)
boundingBox[0] = -1
boundingBox[1] = -1
boundingBox[2] = 5
boundingBox[3] = 2
meshInfos.boundingBox = boundingBox
meshInfos.mshFile = "../../examples/2D/pipe/geometry.msh"

solverInfos = SolverCreateInfo()
solverInfos.gravity = 0
solverInfos.strongPatFS = True
solverInfos.adaptDT = True
solverInfos.maxDT = 0.01
solverInfos.initialDT = 0.01
solverInfos.endTime = 2
solverInfos.IBCfile = "../../examples/2D/pipe/IBC_incomp.lua"
solverInfos.meshInfos = meshInfos

solverIncompInfos = SolverIncompCreateInfo()
solverIncompInfos.rho = 1000
solverIncompInfos.mu = 1e-3
solverIncompInfos.picardRelTol = 1e-6
solverIncompInfos.picardMaxIter = 10
solverIncompInfos.coeffDTincrease = 1.5
solverIncompInfos.coeffDTdecrease = 2
solverIncompInfos.solverInfos = solverInfos

solver = SolverIncompressible(solverIncompInfos)

whatToWrite = VectorString(2)
whatToWrite[0] = "u"
whatToWrite[1] = "p"
solver.addGMSHExtractor("results.msh", 0.1, whatToWrite,"NodesElements")
solver.solveProblem(False)
```

Figure 53: Example of a python file to launch a simulation.

The first part of the file imports the required python objects. Firstly a `MeshCreateInfo` object is filled and will contain the parameters about remeshing. Then a `SolverCreateInfo` object is filled, which will contain general informations common to both the incompressible and compressible solvers. Finally a `SolverIncompCreateInfo` object is filled, containing the parameters specific to the incompressible solver. This object is passed to the constructor of a `SolverIncompressible` object, which represents the solver. The method `addGMSHExtractor` adds an extractor to the solver so that the results are exported to a `.msh` file. Finally, the solver is launched using the method `solveProblem`, which takes a boolean controlling the verbosity of the console output.

References

Books

R. Adams and J. Fournier. *Sobolev Spaces, Volume 140 (Pure and Applied Mathematics)*. Academic Press, jul 2003. URL https://www.xarg.org/ref/a/0120441438/.

R. Courant, K. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM Journal of Research and Development*, 11(2):215–234, Mar. 1967. URL https://doi.org/10.1147/rd.112.0215.

J. Donéa and A. Huerta. *Finite Element Methods for Flow Problems*. Wiley, Apr. 2003. URL https://doi.org/10.1002/0470013826.

E. H. Spanier. *Algebraic Topology*. Springer New York, 1981. URL https://doi.org/10.1007/978-1-4684-9322-1.

Thesis

M. L. Cerquaglia. *Development of a fully-partitioned PFEM-FEM approach for fluid-structure interaction problems characterized by free surfaces, large solid deformations, and strong added-mass effects*. PhD thesis, Université de Liège, 2019.

S. Meduri. *A fully explicit Lagrangian Finite Element Method for highly nonlinear Fluid-Structure Interaction problems*. PhD thesis, Politecnico di Milano, 2019.

Articles

B. Delaunay. Sur la sphère vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72, Jan. 1994. URL https://doi.org/10.1145/174462.156635.

H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, July 1983. URL https://doi.org/10.1109/tit.1983.1056714.

R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181(3):375–389, Dec. 1977. URL https://doi.org/10.1093/mnras/181.3.375.

T. J. Hughes, L. P. Franca, and M. Balestra. A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuška-Brezzi condition: a stable Petrov-Galerkin formulation of the Stokes problem accommodating equal-order interpolations. *Computer Methods in Applied Mechanics and Engineering*, 59(1):85–99, Nov. 1986. URL https://doi.org/10.1016/0045-7825(86)90025-3.

S. Idelsohn, E. Oñate, and F. D. Pin. The particle finite element method: a powerful tool to solve incompressible flows with free-surfaces and breaking waves. *International Journal for Numerical Methods in Engineering*, 61(7):964–989, Sept. 2004. URL https://doi.org/10.1002/nme.1096.

S. Koshizuka and Y. Oka. Moving-particle semi-implicit method for fragmentation of incompressible fluid. *Nuclear Science and Engineering*, 123(3):421–434, July 1996. URL https://doi.org/10.13182/nse96-a24205.

J. R. Macdonald. Some simple isothermal equations of state. *Reviews of Modern Physics*, 38(4):669–679, Oct. 1966. URL https://doi.org/10.1103/revmodphys.38.669.

W. F. Noh and P. Woodward. SLIC (simple line interface calculation). In *Proceedings of the Fifth International Conference on Numerical Methods in Fluid Dynamics June 28 – July 2, 1976 Twente University, Enschede*, pages 330–340. Springer Berlin Heidelberg, 1976. URL https://doi.org/10.1007/3-540-08004-x_336.

S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79 (1):12–49, Nov. 1988. URL https://doi.org/10.1016/0021-9991(88)90002-2.

P. B. Ryzhakov, R. Rossi, S. R. Idelsohn, and E. Oñate. A monolithic lagrangian approach for fluid–structure interaction problems. *Computational Mechanics*, 46(6):883–899, Aug. 2010. URL https://doi.org/10.1007/s00466-010-0522-0.

R. L. Sani, P. M. Gresho, R. L. Lee, and D. F. Griffiths. The cause and cure (?) of the spurious pressures generated by certain FEM solutions of the incompressible Navier-Stokes equations: Part 1. *International Journal for Numerical Methods in Fluids*, 1 (1):17–43, Jan. 1981a. URL https://doi.org/10.1002/fld.1650010104.

R. L. Sani, P. M. Gresho, R. L. Lee, D. F. Griffttths, and M. Engelman. The cause and cure (!) of the spurious pressures generated by certain FEM solutions of the incompressible Navier-Stokes equations: Part 2. *International Journal for Numerical Methods in Fluids*, 1(2):171–204, Apr. 1981b. URL https://doi.org/10.1002/fld.1650010206.

M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114(1):146–159, Sept. 1994. URL https://doi.org/10.1006/jcph.1994.1155.

T. Tezduyar. Stabilized finite element formulations for incompressible flow computations. In *Advances in Applied Mechanics*, pages 1–44. Elsevier, 1991. URL https://doi.org/10.1016/s0065-2156(08)70153-4.

D. Thomas, M. Cerquaglia, R. Boman, T. Economon, J. Alonso, G. Dimitriadis, and V. Terrapon. CUPyDO - an integrated python environment for coupled fluid-structure simulations. *Advances in Engineering Software*, 128:69–85, Feb. 2019. URL https://doi.org/10.1016/j.advengsoft.2018.05.007.

G. Wu, R. E. Taylor, and D. Greaves. The effect of viscosity on the transient free-surface waves in a two-dimensional tank. *Journal of Engineering Mathematics*, 40(1):77–90, 2001. URL https://doi.org/10.1023/a:1017558826258.

Software

D. M. Beazley et al. *Simplified Wrapper and Interface Generator*. 2020. URL http://www.swig.org/.

T. K. F. Da. 2D alpha shapes. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.2 edition, 2020. URL https://doc.cgal.org/5.0.2/Manual/packages.html#PkgAlphaShapes2.

T. K. F. Da, S. Loriot, and M. Yvinec. 3D alpha shapes. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.2 edition, 2020. URL https://doc.cgal.org/5.0.2/Manual/packages.html#PkgAlphaShapes3.

L. H. de Figueiredo, R. Ierusalimschy, and W. Celes. *Lua 5.1 language*. 2020. URL https://www.lua.org/.

C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, Sept. 2009. URL https://doi.org/10.1002/nme.2579.

G. Guennebaud, B. Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org, 2010.

Intel. *Intel Threading Building Blocks*. 2020. URL https://github.com/oneapi-src/oneTBB.

N. Lohmann et al. *JSON for Modern C++* . 2020. URL https://github.com/nlohmann/json.

J. Meneide et al. *Sol3 (sol2 v3.0) - a C++ <-> Lua API wrapper with advanced features and top notch performance*. 2020. URL https://github.com/ThePhD/sol2.

OpenMP Review Board. *OpenMP 4.5 standard*. 2020. URL https://www.openmp.org/.

Python Software Foundation. *Python 3.8*. 2020. URL http://www.python.org/.

Y. Renard, J. Pommier, M. Fournie, and B. Schleimer. *Gmm++: a generic C++ template library for sparse, dense and skyline matrices*. 2020. URL http://getfem.org/gmm.html.

J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied Computational Geometry Towards Geometric Engineering*, pages 203–222. Springer Berlin Heidelberg, 1996. doi: 10.1007/bfb0014497. URL https://doi.org/10.1007/bfb0014497.

B. Stroustrup et al. *The C++17 language*. 2020. URL https://www.isocpp.org.

The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.2 edition, 2020. URL https://doc.cgal.org/5.0.2/Manual/packages.html.