POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# Numerical simulations of two-phase non-Newtonian flows with OpenFOAM

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Author: **Lorenzo Galvan**

Student ID: 977175
Advisor: Prof. Nicola Parolini
Co-advisor: Emilia Capuano
Academic Year: 2022-23

# Abstract

Simulations are a critical aspect of fluid dynamics study and research, and a widely spread tool to understand many different scenarios and simulations. Despite a lot of development already made in this field, there is always a strive to reach better accuracy with fewer resources spent in every simulation. In this work, multi-phase models concerning Newtonian and non-Newtonian fluids will be studied and analyzed. Focusing on bubbles rising in a surrounding fluid, non-Newtonian effects and their relative differences in behaviour with respect to the Newtonian ones will be studied. Mathematical models will be presented and approximated using numerical methods through OpenFOAM, using applications capable of solving partial differential equations through finite volume schemes. Furthermore, an effort to reduce computational time and resources in simulations will be made and explored throughout the thesis, presenting different ways of impacting the cost of complex simulations.

**Keywords:** computational fluid dynamics, simulations, two-phase flow, non-Newtonian, grid refinement, computational resources.

# Abstract in lingua italiana

Le simulazioni sono un aspetto critico nello studio e nella ricerca in fluidodinamica e uno strumento ampiamente diffuso per comprendere numerosi scenari e simulazioni diverse. Nonostante i numerosi sviluppi già compiuti in questo campo, l'obiettivo è sempre raggiungere una migliore accuratezza con la minore spesa di risorse possibile in ogni simulazione. In questo lavoro verranno studiati e analizzati modelli multifase relativi a fluidi newtoniani e non newtoniani. Concentrandosi sulla risalita di bolle immerse in un fluido circostante, si studieranno gli effetti non newtoniani e le relative differenze di comportamento rispetto a quelli newtoniani. Saranno presentati dei modelli matematici poi approssimati con metodi numerici attraverso OpenFOAM, utilizzando applicazioni in grado di risolvere equazioni differenziali parziali attraverso metodi dei volumi finiti. Inoltre, nel corso della tesi si cercherà di ridurre i tempi e le risorse computazionali delle simulazioni, presentando diversi modi per incidere sui costi di simulazioni complesse.

**Parole chiave:** fluidodinamica computazionale, simulazioni, flusso bifase, non Newtoniano, raffinamento della griglia, risorse computazionali.

# Contents

# 1 | Introduction

Fluids are omnipresent in our world and are widely studied in an enormous amount of situations and phenomena. Research in this field has been conducted extensively in both academic and industrial fields, leading to the development of a variety of models in fluid dynamics.

Every model in fluid dynamics stems from equations called conservation laws, which describe the conservation of mass, linear momentum and energy during the flow. In particular, great importance is assumed by the Navier-Stokes equations, formulated over decades by the two scientists, which can be adapted in various cases of study, such as compressible or incompressible flow, viscous flow, and so on. These equations are still being actively and extensively researched, and have a prominent role for being even nominated as a Millennium Prize Problem for proving whether a smooth global solution always exists in a three-dimensional space [1].

Along these studies there is the clear presence of Computational Fluid Dynamics, or CFD, a branch in which numerical methods are used for simulating the phenomena, that may be too difficult or expensive to recreate in a controlled experiment. Here computers are used for obtaining a solution to the problem of interest in a specified computational domain following a model of the real phenomenon. While potent supercomputers may achieve solutions of extreme precision and detail even for very difficult problems, a comparison with results for reality should always be done to calibrate the model and ensure the solution reflects the one in the real world.

This work follows those steps in computational fluid dynamics research and proposes to analyze and study simulations of flows with two different fluids in contact with each other. In particular, a lot of emphasis is put on the behaviour of a particular family of fluids, the non-Newtonian fluids. A lot of research has already been conducted in this field since many of the fluids around us present non-Newtonian behaviours of various natures [2].

While primarily analyzing non-Newtonian behaviour, relevance will also be given to some Newtonian fluids, in order to compare their behaviour under similar conditions.

Research has also been done on computational cost reduction techniques, adopting an array of methods to try and achieve a solution in line with very expensive simulations with just a small fraction of the resources, with a focus on two-dimensional scenarios. Examples of these methods may include grid adaptivity, time adaptivity and different formulations of the problem. While methods and models have already been proposed [3, 4], new ones are always needed, due to changes and advancements in software and to obtain better efficiency.

## Structure of the thesis

This thesis is composed of five main chapters. **Chapter 2** provides a theoretical introduction to fluid dynamics, various kinds of flows and numerical methods useful in computational fluid dynamics. **Chapter 3** will focus on the numerical aspects of the work and provide a description of the newly introduced methods for reducing computational cost in two-dimensional simulations. In **Chapter 4** a first problem is simulated in a Newtonian fluid environment, on which the effect of the proposed methods will be analyzed. Along these lines, in **Chapter 5** a similar problem will be simulated, focusing on behaviour in non-Newtonian rheologies. In **Chapter 6** will see the simulation of a three-dimensional problem in a Newtonian environment, firstly to check the validity of the proposed methods, and then modified using a two-dimensional approach, to evaluate the different impact of cost reduction strategies. Finally, **Chapter 7** will exploit all the developed methods to see if three-dimensional non-Newtonian simulations can bear meaningful results.

# 2 | Mathematical models

This chapter will present all the mathematical models and formulations that will be explored in the thesis. In this work the main focus will be on multiphase flows, a particular case in which the flow is composed of at least two different fluids, that may be gases in a liquid, solid matter dispersed in a liquid or gas, two liquids and so on [5]. There are numerous instances of multiphase flows in the real world: in natural occurrences, such as rain, avalanches, mudslides and similar events, in cavitation effects or even in blood flow [6].

Multiphase flows are also of utmost importance in industrial applications, when dealing with transportation of solid substance in a liquid or gas carrier in pipes or when analyzing oil and gas reservoir drilling [7]. In particular, chapters 4 and 5 will see the study of a bubble column, in which a bubble of gas will be studied during its rise in a column of heavier fluid around it.

When modelling multiphase flows, the equations will reflect the different properties and parameters of the different fluids, and more will be added to describe the different forces that act on the interface between the fluids, such as added mass, drag, Basset, buoyancy and pressure forces [8]. Along these forces, there may also be equations regarding chemical reactions or thermal transfer between the different fluids [5].

This work will focus mainly on two-phase flow and the study of a rising bubble, in both Newtonian and non-Newtonian surrounding fluids, and will see the implementation of methods and techniques for reducing the time and computational cost of the simulations.

## 2.1. Navier-Stokes equations

Every problem in fluid dynamics can be described by some conservation equations, better known as Navier-Stokes equations, which will be described in this section.

First of all, it is important to note the difference between the two possible approaches to writing the equations. While observing the fluid, one can either focus on a precise element in the fluid or observe the flow as a whole. In this way, we can distinguish between a

Lagrangian approach, by studying the behaviour of fluid parcels, or an Eulerian approach, by studying the fields related to the flow, such as velocity or pressure [9].

In the mathematical formulation, it is then possible to distinguish a total material derivative, that is a derivative following the material particle in the flow. It is referred to as Lagrangian derivative and is written as:

$$\frac{D(.)}{Dt} = \frac{\partial(.)}{\partial t} + \mathbf{u} \cdot \nabla(.) \tag{2.1}$$

where $t$ is the time and $\mathbf{u}$ is the velocity vector. As the equation suggests, the total derivative of a quantity consists of a term representing its local derivative and a term regarding its change due to the movement in the flow.

## Continuity equation

The first equation is commonly referred to as continuity equation, and is an equation of mass conservation. The equation states that the difference in mass flow throughout the system is zero, for any arbitrary finite volume fixed in space. When considering an infinitesimally small volume, it can be written as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{2.2}$$

where $\rho$ is the density, $t$ is the time and $\mathbf{u}$ is the velocity vector.

In some particular cases, this equation can be further simplified. In fact, when considering an incompressible fluid with constant density along the particle trajectory, the equation reduces to:

$$\nabla \cdot \mathbf{u} = 0, \tag{2.3}$$

when the initial density $\rho$ is uniform in the whole domain.

## Momentum equation

The second equation relates to the forces acting in the control volume and is an expression of Newton's second law of motion:

$$F = \frac{\partial (m\mathbf{u})}{\partial t}$$

where $F$ represents the sum of the forces acting on the mass $m$. It states that momentum in the control volume is only modified through the action of the forces as described by Newton's second law. The momentum equation can be described for the component of $\mathbf{u}$ ($u$, $v$, $w$), or can be written in vector form as:

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla \cdot (\rho\mathbf{u} \otimes \mathbf{u} - \bar{\bar{\sigma}}) = \rho\mathbf{f} \tag{2.4}$$

where $\mathbf{f}$ is a force per unit mass acting on the volume, $\otimes$ represents the tensor product and $\bar{\bar{\sigma}}$ is the Cauchy stresses tensor, defined as $\bar{\bar{\sigma}} = -p\bar{\bar{\mathbf{I}}} + \bar{\bar{\tau}}$, where $p$ is the pressure, $\bar{\bar{\mathbf{I}}}$ is the identity tensor and $\tau$ is the viscous stress. For Newtonian fluids, the latter is defined as:

$$\bar{\bar{\tau}} = 2\mu D(\mathbf{u}) = 2\mu \frac{\nabla\mathbf{u} + (\nabla\mathbf{u})^T}{2}$$

where $\mu$ is the molecular viscosity and $D(\cdot)$ represents the symmetric part of rate of deformation tensor.

## Energy equation

The third and last equation states that energy is conserved in the system by the first law of thermodynamics. The energy equation can be written as:

$$\frac{\partial\rho e}{\partial t} + \nabla \cdot ((\rho e + p)\mathbf{u} - \bar{\bar{\tau}} - k\nabla T) = \rho\mathbf{f} \cdot \mathbf{u} + \rho s \tag{2.5}$$

where $e$ is the specific total energy, $k$ is the thermal conductivity and $s$ is a specific external heat source.

Finally, since the equations are described in seven variables ($\rho$, $p$, $u$, $v$, $w$, $T$, $e$) but only five are provided, they can be closed by adding the equation of state

$$p = \rho RT \tag{2.6}$$

with $R$ as the specific gas constant, and the caloric state equation

$$e = c_v T \tag{2.7}$$

with $c_v$ as the specific heat capacity at constant volume.

From now on, incompressible fluids will be considered with no heat or energy fluxes, thus only the continuity and momentum equation will be solved separately for obtaining the field of $p$ and $\mathbf{u}$.

## 2.2. Non-Newtonian fluids

Every fluid has a specific behaviour when reacting to stress. Some fluids follow Newton's law viscosity, where the shear stress is linearly dependent with the shear rate [10]:

$$\tau = -\mu(\nabla\mathbf{u} + (\nabla\mathbf{u})^T)) = -\mu\dot{\gamma}$$

where $\mu$ is the viscosity, $\tau$ is the stress tensor and $\dot{\gamma}$ is the rate of strain tensor.

In reality, many fluids are non-Newtonian, which means that their viscosity is dependent on the shear rate in a non-linear way or through deformation history. Different classes of non-Newtonian fluids are described in figure 2.1.

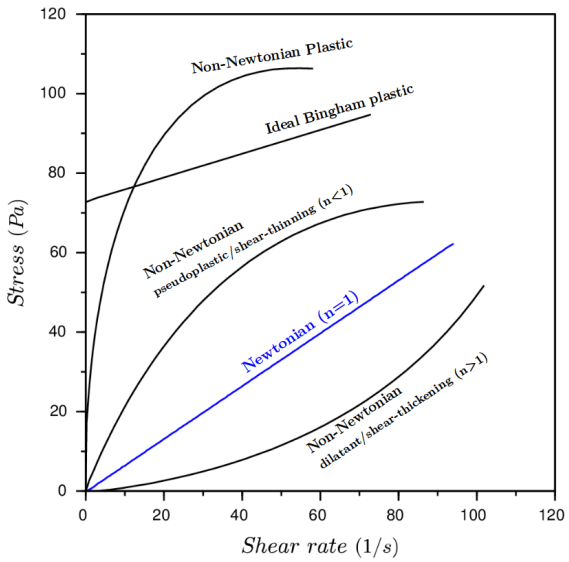



Figure 2.1: Classification of non-Newtonian fluids [11].

Some of them, such as shear-thinning and shear-thickening fluids, are described as generalised Newtonian fluids, in which $\mu$ can be replaced by an apparent viscosity $\eta(\dot{\gamma})$:

$$\tau = -\eta(\dot{\gamma})\dot{\gamma}. \tag{2.8}$$

Many empirical models have been proposed for studying different types of fluids, with

different governing equations:

- Power-law fluids:

$$\eta(\dot{\gamma}) = K(\dot{\gamma})^{n-1} \tag{2.9}$$

  where $K$ is the consistency index and $n$ is the flow behaviour index. In particular, for $n = 1$ the flow will be Newtonian, for $n < 1$ the flow will be shear-thinning (viscosity reduces with the shear rate) and for $n > 1$ the flow will be shear-thickening (viscosity increases with the shear rate) [10]. It is also possible to introduce a truncated power-law model, with $\eta$ restricted between two values, to avoid problems with zero or infinite viscosity.

- Cross fluids:

$$\eta(\dot{\gamma}) = \frac{\mu_0}{1 + (\lambda\dot{\gamma})^{1-n}} \tag{2.10}$$

  where $\lambda$ is the natural time. At a low shear rate it behaves like a Newtonian fluid, while at a high shear rate it behaves like a power-law fluid [12].

- Carreau fluids:

$$\eta(\dot{\gamma}) = \mu_\infty + (\mu_0 - \mu_\infty)(1 + (\lambda\dot{\gamma})^2)^{\frac{n-1}{2}} \tag{2.11}$$

  where $\mu_0$ and $\mu_\infty$ are the viscosity at zero and infinity shear rate and $\lambda$ is the relaxation time. At a low shear rate it behaves as a Newtonian fluid, while at a high shear rate it behaves like a power-law fluid [12].

- Herschel-Bulkley fluids:

$$\eta(\dot{\gamma}) = \begin{cases} \infty & \dot{\gamma} < \dot{\gamma}_0 \\ K\dot{\gamma}^{n-1} + \tau_0\dot{\gamma}^{-1} & \dot{\gamma} > \dot{\gamma}_0 \end{cases} \tag{2.12}$$

  where $K$ is a rheological parameter. In particular, the fluid does not deform until a certain stress is applied; this allows to capture the behaviour of the Bingham plastic fluids.

In this work, the shear-thinning and shear-thickening fluids will be considered. They will be studied in chapter 5 regarding a gas bubble rising in a non-Newtonian fluid.

## 2.3. Multi-phase flows

When dealing with multiphase flows, great importance is given to free surface flows, where we have two different fluids with a moving boundary between them. A great example is

the study of how surface waves in water interact with air or objects around and above the water, along with the study of open channels and rivers [13], [14]. In fluid dynamics and computational fluid dynamics, several models have been developed for studying this kind of problem.

For the purposes of this work, focused on various instances of a bubble rising in a surrounding fluid, the fluids will always be considered immiscible, so a clear interface can always be drawn between them.

## Numerical methods

- Arbitrary Lagrangian-Eulerian methods (ALE): they can combine the two different views in which the problem can be solved. In Lagrangian algorithms, each point in the computational grid moves with the flow, leading to great precision in tracking the fluid interface but needing constant re-meshing, as they are subject to large domain deformations. In Eulerian algorithms, a fixed domain is used to capture the interface moving with the flow: they lead to domain stability but may lack precision in determining the interface [15]. The ALE methods combine these two effects by allowing the computational domain to be fixed or moved as needed.
- Level-set methods (LS): they are a front-capturing approach for the interface. They define a function $\phi$ such that the zero level set $\phi = 0$ corresponds to the moving interface [16]. In this way, a much simpler equation may be solved for just obtaining the position of the interface at every time following the flow movements. During the flow evolution it may be needed to redefine the function as its shape may become too sensitive or almost discontinuous to correctly track the moving interface.
- Volume of fluid methods (VOF): they are a front-capturing approach based on previously developed marker-and-cell methods, which used markers to detect fluid position in a fixed grid. The VOF methods rely on a characteristic function or volume fraction function $C$ in the domain which identifies the fraction of fluid present in a particular cell. It varies from zero in cells with no fluid to one in cells full of fluid; in mixed cells it will obtain an intermediate value, defining the interface [17]. During the evolution, the function may change values in space, leading to capturing an interface not sharp enough, especially on coarser grids. This widely used method will see extensive use in this work, due to being the one exploited by the applications used for simulations in chapters 4, 5 and 6.

## 2.4. Flow model

This work will mainly focus on the rise of a bubble of light fluid immersed in a heavier fluid. For this kind of problem, the equations governing the flow are an adaptation of the Navier-Stokes equations in the case of two incompressible, immiscible and isothermal fluids. Thus, it is possible to only consider the momentum and continuity equations to obtain the fields of pressure $p = p(\mathbf{x}, t)$ and velocity $\mathbf{u} = \mathbf{u}(\mathbf{x}, t)$ depending on the position $\mathbf{x}$ and time $t$. Given a domain $\Omega$ divided into $\Omega_1(t)$ and $\Omega_2(t)$, with each corresponding to one of the two phases, such that $\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2$ and $\Omega_1 \cap \Omega_2 = \emptyset$, the equations read, for $i = 1, 2$:

$$\begin{cases} \rho_i(\mathbf{x}) \left( \dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = -\nabla p + \nabla \cdot (\mu_i(\mathbf{x})(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)) + \rho_i(\mathbf{x})\mathbf{g} & \text{in } \Omega \times [0, T] \\ \nabla \cdot \mathbf{u} = 0 & \text{in } \Omega \times [0, T] \end{cases} \tag{2.13}$$

where $T$ is the final time, $\rho_i$ are the density of the fluids, $\mu_i$ are the viscosities of the fluids and $\mathbf{g}$ is the gravitational force field.

The equations are supported by the coupling conditions on the interface $\Gamma$, defined as $\Gamma := \partial\Omega_1 \cap \partial\Omega_2$, $\Omega = \Omega_1 \cup \Gamma \cup \Omega_2$, $\partial\Omega \cap \Omega_2 = \emptyset$:

$$\begin{cases} [\mathbf{u}]_\Gamma = 0 & \text{at } \Gamma \times [0, T] \\ [-p\mathbf{I} + \mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)]_\Gamma = \sigma\kappa\hat{\mathbf{n}} & \text{at } \Gamma \times [0, T] \end{cases} \tag{2.14}$$

where $\sigma$ is the surface tension coefficient, $\kappa$ is the curvature of the interface and $\hat{\mathbf{n}}$ is the unit normal on $\Gamma$ pointing from $\Omega_2$ into $\Omega_1$.

# 3 | Numerical models

For simulating the proposed two-phase problems, this work will see the use of OpenFOAM (Open Source Field Operation and Manipulation) C++ libraries version 10 [18], which will be expanded for the goal of reducing time and computational complexity of the simulations.

## 3.1. interFoam

The main application present in the OpenFOAM library for solving the two-phase flows for immiscible, incompressible and isothermal fluids is interFoam, capable of solving those free surface flows.

As with every OpenFOAM application, interFoam relies on the use of dictionaries, files which allow the user to set every detail about the simulation. Among others, notable dictionaries are the `controlDict`, which sets every detail about simulation time-step and final time, file saving instructions, and user-defined post-processing function (some developed in this work will be described in appendix A), the `blockMeshDict`, which defines the computational mesh with details about cells, faces and boundary types, the `fvSchemes`, which defines all the numerical schemes for approximating the equations, providing discretization for the divergence, gradient, Laplacian and so on, the `fvSolution`, which provides information about the solvers needed for every field and which numerical methods to use, and so on.

The interFoam application relies on the volume of fluid (VOF) method already described, by defining a volume fraction or phase fraction $\alpha$ in each of the computational cells to identify the different fluids, ranging from zero (which identifies one fluid) to one (which identifies the other fluid). Cells in which this phase fraction obtains intermediate values will belong to the interface between the two fluids [18].

The solver introduces a new equation, solving for the phase fraction. Defining $\alpha$ the phase

fraction of the first fluid, the equation reads:

$$\frac{\partial \alpha}{\partial t} + \mathbf{u} \cdot \nabla \alpha = 0 \tag{3.1}$$

It is necessary to solve only one equation since it is known that, with two fluids, the sum of the volume fraction is equal to one, and the phase fraction for the second fluid can be obtained as $1 - \alpha$.

During the simulation, it is necessary to keep the interface (where $\alpha$ assumes values between 0 and 1) as sharp as possible, reducing the effect of numerical diffusion. interFoam exploits different features for a correct solving of this problem. Firstly, it uses the MULES algorithm, which is semi-implicit and second order in time, to keep the values of $\alpha$ between 0 and 1 [19]. Then, various models can be adopted to keep a sharp interface between the two fluids, such as the piecewise-linear interface calculation (PLIC), multicut piecewise-linear interface calculation (MPLIC) or a Van Leer convergence scheme [19]. In this work, a Van Leer convergence scheme is adopted, which is a composition of upwind and central differencing schemes, reaching second order in time.

This can be extended to more fluids by introducing $N - 1$ equations, where $N$ is the number of considered fluids, knowing that $\sum_{i=i}^{N} \alpha_i = 1$, with $\alpha_i$ phase fraction of the fluid $i$.

In this work, this scalar field will be referred to as $\alpha$ or `alpha.heavy`, identifying the heavy fluid when it assumes a value of 1, and identifying the bubble of light fluid when assuming a value of 0. During the simulations, isolines (when working in two dimensions) or isosurfaces (when working in three dimensions) will be saved with a value of $\alpha = 0.5$, in order to have a precise reference for the position and shape of the bubble during its rise.

The solver also includes modelling of surface tension parameters for different needs at the interface, such as contact angle with walls, and can support different kinds of rheology behaviour, such as non-Newtonian ones. It may also support adaptive re-meshing, which will see use in this work on problems better described in sections 3.4 and 3.5. All the transport models and simulation settings are described in OpenFOAM dictionaries.

There are other solvers which expand interFoam capabilities, such as compressibleInterFoam (possibility of compressible fluids) or compressibleMultiphaseInterFoam (allows for more than two fluids) but they are not needed in this work, as the focus will be on the incompressible, immiscible and isothermal flow of two fluids.

## 3.2. Finite volume method

The interFoam solver relies on the finite volume method, or FVM. This is a widely used discretization method for approximation of partial differential equations, such as the Navier-Stokes equations. As a first step, it is necessary to divide our computational domain in control volumes, which will be referred to as cells, of any shape, with just the conditions of planar faces. Then the equations will be discretized and solved algebraically through the use of matrices, with the final system transformed into the form $AU = B$, with $U$ as the vector of unknowns. The matrix system can then be solved through linear or iterative methods, which in OpenFOAM simulations are set case by case through the use of dictionaries.

Every quantity studied during the simulation is averaged over the control volume and stored in the centre of the cell, with the flow variables varying linearly across the cell. Then, the matrix and right-hand side vector values are obtained by adding every separately integrated term in the equation over each control volume [20]. Starting from the equations presented in section 2.1 and considering a single incompressible fluid with constant density in the domain, an outline of the discretization is now provided for the constant source term and the convection term, indicating the control volume with $V$ and its surface as $S$.

When integrating the source term due to gravitational force over the control volume of the cell one obtains:

$$\int_V \mathbf{g}\, dV = \mathbf{g} V_C \tag{3.2}$$

where $V_C$ is the volume of a single cell, since the force is constant in the domain and a fluid with constant density is considered. Thus, the gravity force is added as a term in the right-hand side vector $B$.

When integrating the convection term over the control volume one obtains, using the divergence theorem:

$$\int_V (\nabla \cdot \mathbf{uu})\, dV = \int_S \mathbf{u}(\mathbf{u} \cdot \hat{\mathbf{n}})\, dS \tag{3.3}$$

where $\hat{\mathbf{n}}$ is the normal to the cell surface pointing outward for every face. Now, $(\mathbf{u} \cdot \hat{\mathbf{n}})\, dS$ is known since it is the flow rate out of the cell surface, leaving only the unknown $\mathbf{u}$ at the cell faces to be found. Then, it is possible to move from an integral formulation to a

discrete one, thanks to [20]. For a cell composed of $N$ faces:

$$\sum_{i=1}^{N} \int_S \mathbf{u}_i(\mathbf{u}_i \cdot \hat{\mathbf{n}}_i)\, dS_i \approx \sum_{i=1}^{N} \int_S \mathbf{u}_{f_i}(\mathbf{u}_{f_i} \cdot \hat{\mathbf{n}}_{f_i})\, dS_i \tag{3.4}$$

where $\hat{\mathbf{n}}_{f_i}$ is the normal to the face $i$ and $\mathbf{u}_{f_i}$ is the velocity vector in the centre $f$ of the face $i$. These face values are obtained by interpolating the velocity values stored in the centre of two neighbouring cells, or by using a ghost cell in the case of boundary cells if needed. This could be done through a variety of schemes, such as Upwind, Central Differencing or QUICK schemes. Finally, all values are stored in the correct position in the matrix $A$, leading to diagonal terms due to contributions of cell centres and off-diagonal terms due to contributions of the neighbouring cells, reflecting the matrix connectivity.

An insight is also given for linear and non-linear source terms, which do not appear in the flow model proposed in this study, but may offer an interesting view on the method.

Considering a linear source term in the equation $F\mathbf{u}$, with $F$ scalar quantity and integrating over the control volume one obtains:

$$\int_V F\mathbf{u}\, dV = F_C \int_V \mathbf{u}\, dV \tag{3.5}$$

where $F_C$ is the scalar integrated over the volume. Then, considering $\mathbf{x} - \mathbf{x}_c$ the distance from the cell centre and $\mathbf{u}_c$ the value of the velocity in the centre of the cell, it is possible to expand the velocity:

$$F_C \int_V \mathbf{u}\, dV = F_C \int_V (\mathbf{u}_C + (\mathbf{x} - \mathbf{x}_C)\nabla\mathbf{u}_C)\, dV. \tag{3.6}$$

Then the integral can be split into two parts, leading to:

$$F_C \int_V (\mathbf{u}_c + (\mathbf{x} - \mathbf{x}_C)\nabla\mathbf{u}_C)\, dV = F_C\mathbf{u}_C \int_V dV + F_C \int_V (\mathbf{x} - \mathbf{x}_C)\, dV(\nabla\mathbf{u}_C) \tag{3.7}$$

and, since $\mathbf{x_C}$ is the cell centre, the integral $\int_V(\mathbf{x} - \mathbf{x}_C)\, dV$ will be equal to zero, giving the final term needed to add to our matrix system:

$$\int_V F\mathbf{u}\, dV = F_C\, \mathbf{u}_C\, V_C \tag{3.8}$$

This term can now either be added as an implicit term to the matrix $A$ or as an explicit term to the right-hand side $B$. In general, the better choice is the one that maximizes diagonal dominance of the matrix $A$.

Finally, non-linear source terms are linearized using values at previous iterations. For example, a source term of the form $F\mathbf{u}^2$ at iteration $i$ will be linearized as $(F\mathbf{u}^{i-1})\mathbf{u}$, exploiting the known value at the previous iteration $\mathbf{u}^{i-1}$. Then, following the discretization previously described for linear source term in equation 3.2, one can write:

$$\int_V F\mathbf{u}^2 \, dV = F_C \mathbf{u}_C^{i-1} \mathbf{u}_C \tag{3.9}$$

at every iteration $i$ and again can be added as an implicit term to the matrix $A$ or as an explicit term to the right-hand side $B$.

Of course, the quality of the results heavily depends on the mesh quality, which needs to be high enough (usually obtained through smaller and more refined cells) to guarantee a correct interpolation of the quantities. The information is transported through fluxes, which need to be conservative and consistent: a quantity moving between two neighbouring cells must have an opposite value on the cells, and the interpolation of a regular function needs to be continuous when the cell volume tends to zero.

As a final consideration, while the grid can be created in a structured or unstructured geometry and with cells of irregular shapes, for the purpose of this work the computational grid will always start as a structured mesh and the cells will always be considered cubic. When dealing with refinement, every newly created cell will still be cubic, as it is obtained by cutting existing cells along its midpoints. Some exceptions arise when working in two directions: in this setting, one of the directions is never considered by the solver or the refinement, and therefore the cells will be squares instead of cubes, maintaining the square property even when refined.

## 3.3. Computational mesh

The main focus will be on the computational mesh defined in OpenFOAM. Four different components define a mesh:

- Points: a location in 3D space, defined by a vector of its coordinates. They are compiled into a list and referred to with a label stating their position in the list. Every point must be part of at least one face.
- Faces: An ordered list of points such that two neighbouring points are connected by an edge. They are compiled into a list and referred to with a label stating their position in the list. They can be differentiated into internal faces and boundary faces based on their connectivity with cells.

- Cells: a list of faces in arbitrary order.
- Boundary: a list of patches, each of which is associated with a boundary condition. A patch is a list of face labels consisting only of boundary faces and no internal faces.

The mesh is represented by a `polyMesh` object, whose description is based on faces.

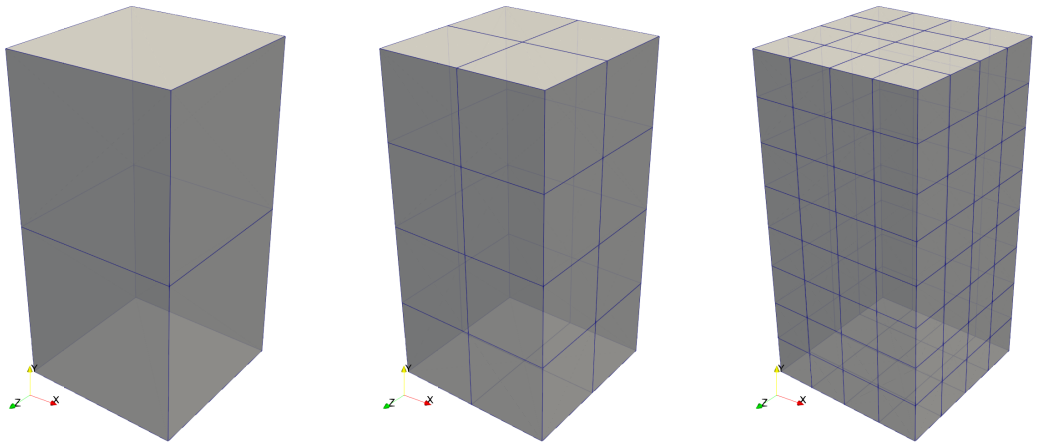
## 3.4. Native AMR

When performing simulations, particularly when analyzing different fluids in contact with each other, an adaptive method may be implemented to speed up the process. Adaptive Mesh Refinement, or AMR, is a case of $h$-adaptivity, where the mesh connectivity and shape are changed in order to decrease the computational cost, making possible solving problems otherwise too costly when working on a uniform grid.

The implementation in OpenFOAM is based on three main classes:

- `fvMeshTopoChangers::refiner`: the actual refiner which holds the mesh and gets called to do the refinement.
- `hexRef8`: used for cutting cells into 8 smaller ones.
- `refinementHistory`: contains the refinement tree, which is the history of the refinement.

When using AMR, during each iteration of the simulation the grid gets checked and cells may get refined or unrefined based on some condition. In this work, the focus will be on when a cell contains the interface between the two fluids; when this happens, the original cell gets split into 8 smaller ones to allow for a better capturing of the interface and a better accuracy in the results, as shown in figure 3.1. The process may also be done more than once if more accuracy is needed, resulting in more and even smaller cells around the interface.



(a) Original cells. (b) First level refinement. (c) Second level refinement.

Figure 3.1: Iterations of cell refinement

The cutting, handled by the `hexRef8` class, works by finding the middle point in each of the 6 faces of the cell that needs to be cut and connecting then with a new edge. In this way, two new cells are created in each of the three axis directions, resulting in 8 new smaller cells.

However, when working in a 2D framework, this method is not suitable for a correct simulation: in fact, even if the original mesh has a one-cell thickness in the direction orthogonal to the plane of interest, the cells are still split in eight, thus making the problem not two-dimensional anymore. So a 2D adaptation of the original AMR is needed.

### 3.4.1. 3D Refinement engine

The refinement is primarily handled by the `refiner` class, which is defined in the `fvMeshTopoChanger` namespace, a collection of all the other classes that move the mesh points, update the cell volumes and generate the corresponding mesh fluxes.

The main methods of the class are the `refine` and `unrefine`, which are responsible for the actual management of the cells and getting the information from the OpenFOAM dictionaries related to the refinement. Moreover, it can refine different regions in the same mesh, chosen in relation to the maximum level of refinement or field or condition of refinement. The cell cutting relies instead on the `hexRef8` class, and all the changes made during every iteration are finally applied to the mesh using the `update` method.

### 3.4.2. 3D Mesh cutter

The `hexRef8` is the one responsible for dividing every cell into 8 smaller ones. It can access the mesh and analyze the properties of every point, face and cell. When receiving the list of cells that need to be cut, it can create the new ones using the methods `addFace` and `createInternalFace`, along with checking and updating that all the conditions of a valid mesh are still met.

It also contains a variable of the `refinementHistory` class, needed to keep track of all the changes applied to the cells.

### 3.4.3. 3D Refinement history

It stores the history of all the cells that were refined. It is needed for the unrefinement, since every cell needs to know how many level of refinement has gone through in order to get back to its original state. It includes a further class `splitCell8`, which can handle the division of the cell into smaller ones.

## 3.5. 2D AMR

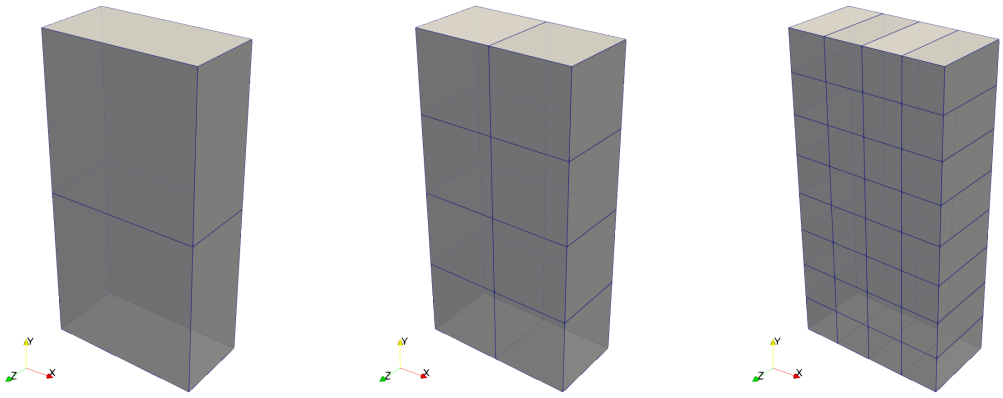
The adaptive mesh refinement implemented in the release version of OpenFOAM-10 is only correctly applicable in 3D cases, due to the inherent 3D structure of the application. For this reason, a 2D version of the AMR has been developed, starting from the already present structures, and will be depicted in this chapter.

When simulating a two-dimensional problem when working in a natural three-dimensional framework as OpenFOAM, some conditions need to be applied to the case, both in mesh generation and conditions on its boundary. In fact, the mesh is created with only one layer in the normal direction, on which an *empty* boundary condition is imposed, since no solution is required in the third dimension [18].

Given the interest that two-dimensional simulations still exert in the fluid dynamics field, many other instances of AMR in this environment for OpenFOAM have already been developed and studied, as shown in [3] and [4], but they rely on older versions of OpenFOAM and are no longer applicable in newer ones.

The newly implemented AMR 2D instead only cuts the selected cells into 4 smaller ones, disregarding the normal direction to the domain plane. In this way the mesh and the solution are kept in a 2D state, enabling the speedup in solution times due to fewer cells. Moreover, it is even more efficient than the classical AMR applied to the 2D cases,

since there in no extra layer added in the unwanted direction. The final result of the implementation will result in the refinement shown in figure 3.2.



(a) Original cells. (b) First level refinement. (c) Second level refinement.

Figure 3.2: Iterations of cell refinement

### 3.5.1. 2D Refinement engine

The first step in order to create a refinement engine capable of working in a full two-dimensional environment is creating a new refiner class, namely `refiner2D`, inspired from the original `refiner` class. The new class is still obtained with public inheritance from the `fvMeshTopoChanger`, just as the previous three-dimensional mentioned. Here all the attributes and methods are modified so they are able to handle a different number of newly created cells during every iteration.

In addition to the already present parameters, there are two new ones:

- `axis`: specifies which axis does not need to be refined, which is the direction normal to the plane of interest.
- `axisVal`: should represent the middle point of the plane of interest, which is used to keep the consistency in the unrefinement step.

The class also references a new mesh cutter, since `hexRef8` is only capable of cutting into 8 cells.

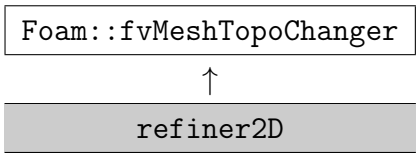



Table 3.1: Inheritance diagram for `refiner2D` class.

### 3.5.2. 2D Mesh cutter

The new class created for cutting in only two direction is `hexRef82D`, which serves the same goal as the original `hexRef8` for a different dimension space. As in the previous class, all the methods and attributes are modified to work with the new changes. In particular, it uses the previously mentioned `axis` and `axisVal` for determining which faces to cut, disregarding the third unused direction. When doing this, the cutter checks if a face is excluded from refinement by checking its `axis` value, and proceeds to cut only in the required directions.

When unrefining the mesh, the cutter then needs to stitch back together the smaller cells, making use of another class called `removeFaces2D`, obtained by modifying the already existing `removeFaces` and making it capable to apply topology changes in only two directions.

### 3.5.3. 2D Refinement history

Lastly, the class `refinementHistory2D` has been created to keep track of the history of the refinement. It contains information about:

- `visibleCells`: list of cells in the current mesh currently unrefined with label -1 or index into `splitCells`.
- `splitCells`: list of parents for every split cell and may contain the subsplit of 4 indices into `splitCells`.

Another new class, `splitCell4` has been created to take over the role of `splitCell8`, to be able to deal with the division of the small only in four smaller ones, updating a smaller list of parent and daughter cells.

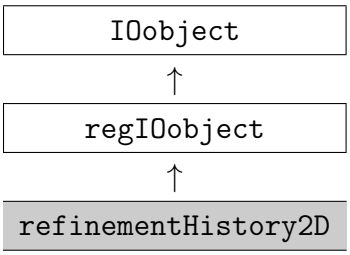



Table 3.2: Inheritance diagram for `refinementHistory2D` class.

# 4 | 2D Newtonian rising bubble

In this chapter, a first test case will be analyzed based on [21], studying a two-phase flow of two immiscible liquids. It consists of two-dimensional domain simulations of a bubble of lighter fluid rising while immersed in a heavier fluid. While the problem seems very simple, it actually introduces complex topology changes, due to the changing shape of the bubble during its rise in the domain.

First, some tests will be carried out based on the benchmark described [21], to check for consistency between the OpenFOAM model and the proposed ones, and then the newly proposed 2D AMR will be performed on the same cases to analyze its effect on computational cost and accuracy.

## 4.1. Definition of test cases

Two test cases will be analyzed, situated in the same domain as seen in figure 4.1 and starting conditions but with different fluid physical parameters as seen in table 4.1.

The domain $\Omega$ is composed by a rectangle [1x2] in which a bubble of the lighter fluid (with $\rho_2 < \rho_1$) of radius $r_0 = 0.25$ is positioned at $[0.5, 0.5]$. A no-slip boundary condition ($\mathbf{u} = 0$) will be imposed on the top and bottom sides and a free-slip boundary condition ($\mathbf{u} \cdot \hat{\mathbf{n}} = 0, \hat{\mathbf{t}} \cdot (\nabla \mathbf{u} + (\nabla \mathbf{u})^T) \cdot \hat{\mathbf{n}} = 0$, where $\hat{\mathbf{n}}$ is the normal vector and $\hat{\mathbf{t}}$ is the tangent one) will be imposed on the vertical sides. An initial condition $\mathbf{u}(0) = 0$ is assumed.

In particular, the first test case will see stronger tension forces applied on the rising bubble, keeping an ellipsoidal shape during the simulation, while the second case, with smaller tension forces, will see the possible breaking of the bubble during its rise [22].

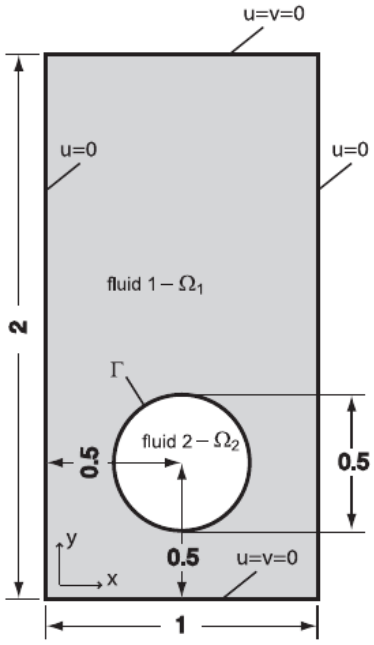



Figure 4.1: Initial configuration and boundary conditions for the test cases [21].

| Test case | $\rho_1$ | $\rho_2$ | $\mu_1$ | $\mu_2$ | $g$ | $\sigma$ |
|---|---|---|---|---|---|---|
| 1 | 1000 | 100 | 10 | 1 | 0.98 | 24.5 |
| 2 | 1000 | 1 | 10 | 0.1 | 0.98 | 1.96 |

Table 4.1: Physical parameters and dimensionless numbers defining the test cases [21].

Regarding the numerical aspects, in both cases three sets of grid and time-step will be used, with grid sizes $h$ of $\frac{1}{40}$, $\frac{1}{80}$ and $\frac{1}{160}$ with a time step $\Delta t$ of $\frac{h}{16}$, depending on the grid size. In both cases, the final time will be $T = 3$.

### 4.1.1. Quantities of interest

In order to keep consistency with the studies in the benchmarks corresponding to the simulations developed in this work ([21], [23], [24]), two quantities will be studied, in particular:

- Center of mass position: used to track the position of the bubble inside the domain, obtained as $\mathbf{X}_C = (x_c, y_c) = \frac{\int_{\Omega_2} \mathbf{x} dx}{\int_{\Omega_2} 1 dx}$.
- Rise velocity: reported in the $y$-direction, since the problem is symmetrical along the $x$-axis, intended as the mean velocity of the bubble, obtained as $\mathbf{U}_C = \frac{\int_{\Omega_2} \mathbf{u} dx}{\int_{\Omega_2} 1 dx}$.

These quantities are collected at run-time during the simulations, exploiting the functionalities of the `controlDict` OpenFOAM dictionary. This allows the user to collect data without needing to save more instances the simulation results than needed, as the solver will gather and compute the selected values at every specified time and save them in a dedicated file. A detailed computation of these quantities is described in A.2.

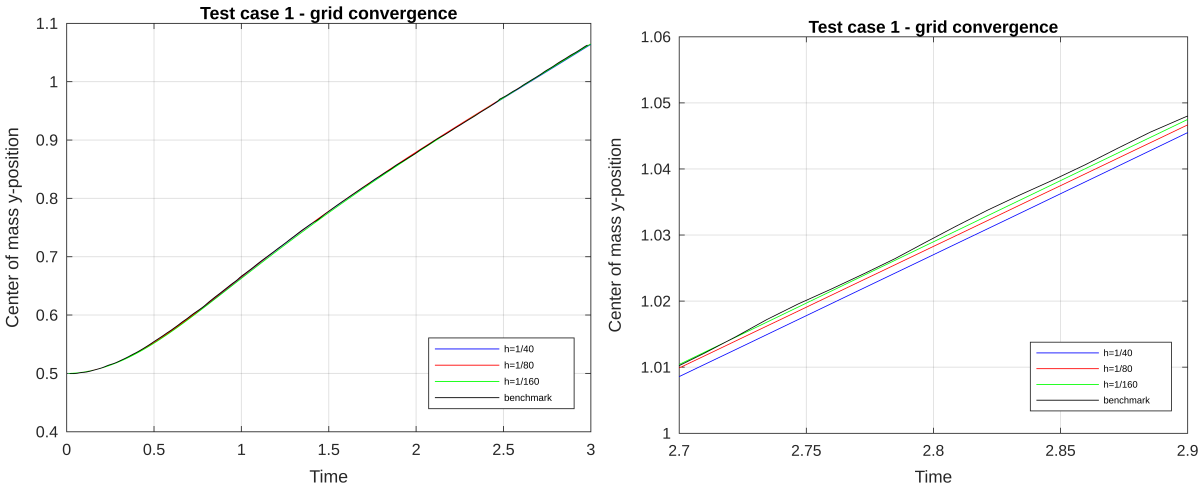
## 4.2. Results

### 4.2.1. Grid convergence

The two cases have been simulated in different mesh configurations as suggested in the benchmark [21] to check if the simulation converges to the reference solution. The different grid sizes, time-step sizes and number of cells are described in table 4.2.

| $h$ | $\Delta t$ | number of cells |
|---|---|---|
| $\frac{1}{40}$ | $\frac{1}{640}$ | 3200 |
| $\frac{1}{80}$ | $\frac{1}{1240}$ | 12800 |
| $\frac{1}{160}$ | $\frac{1}{2560}$ | 51200 |

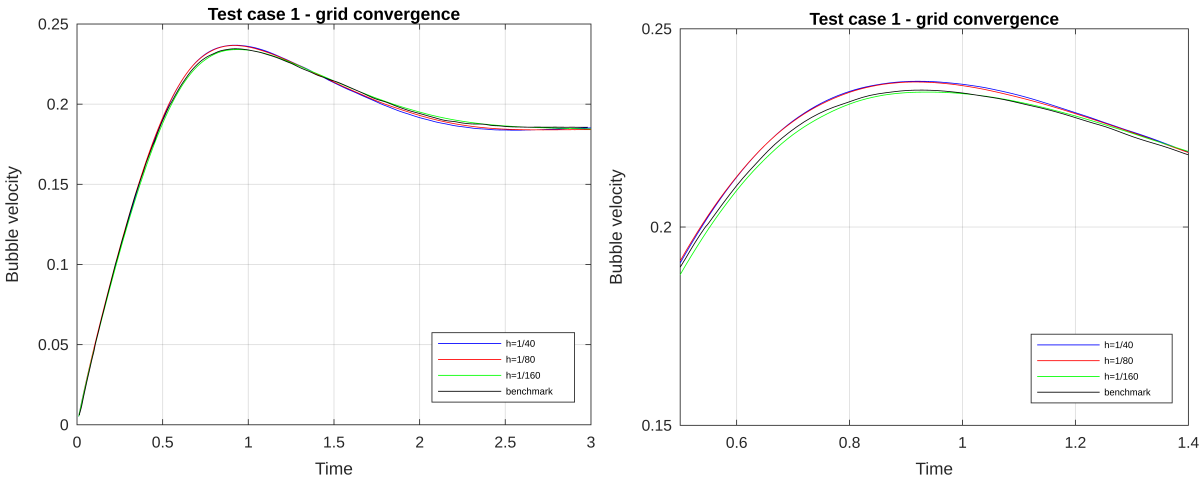
Table 4.2: Different settings for grid convergence simulations.




(a) Convergence of center of mass position evolution.

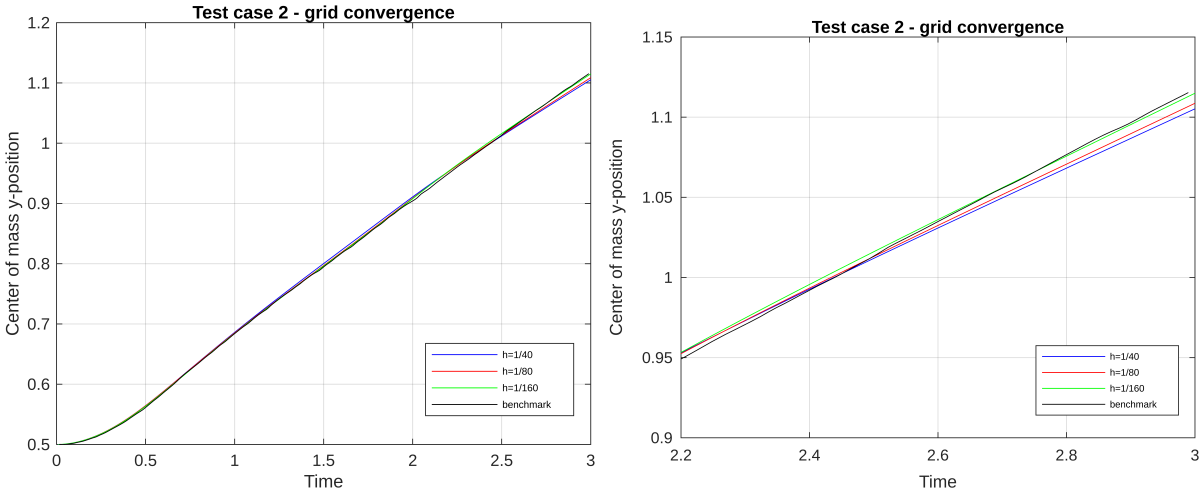
(b) Close-up of the results.

Figure 4.2: Center of mass convergence for the different grids in test 1. Benchmark results from [21].



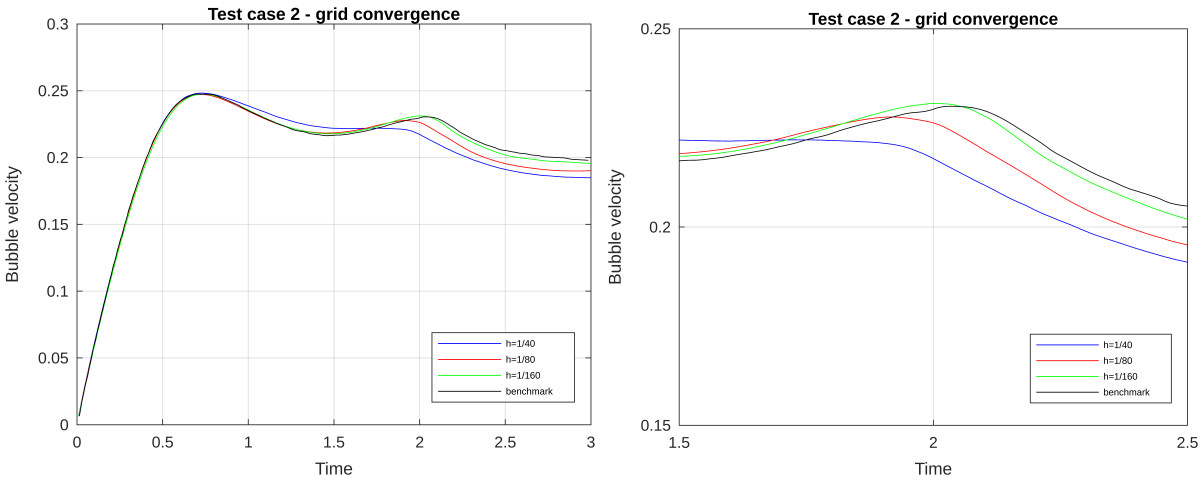

(a) Convergence of bubble rising velocity evolution. (b) Close-up of the results.

Figure 4.3: Bubble velocity convergence for the different grids in test 1. Benchmark results from [21].




(a) Convergence of center of mass position evolution. (b) Close-up of the results.

Figure 4.4: Center of mass convergence for the different grids in test 2. Benchmark results from [21].




(a) Convergence of bubble rising velocity evolution. (b) Close-up of the results.

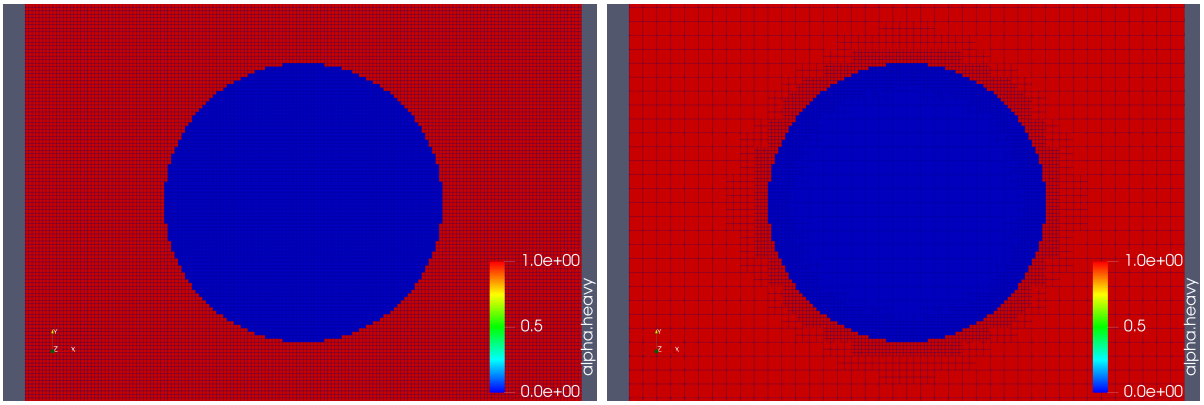
Figure 4.5: Bubble velocity convergence for the different grids in test 2. Benchmark results from [21].

The results presented in figures 4.2, 4.3, 4.4 and 4.5, including the comparison with the benchmark ones [21], show that the solution converges to the reference one as the grid gets refined. Thus, the finest grid size ($h = \frac{1}{160}$) will be used to analyze the effect of the dynamic refinement.

### 4.2.2. AMR effect

The two test cases have been simulated with the AMR described in chapter 3.5 to analyze the computational gain derived by the new refinement implementation. As stated before, the refinement only affects the grid in the plane of the domain and disregards the third direction. While this method leads to a great reduction in the number of cells required in the simulation, the speedup may be limited by the overhead computations introduced by the refinement itself.

In figure 4.6 the starting conditions for the two grids are presented. In 4.6a the grid is set for both the starting conditions and the evolution of the bubble, while in 4.6b the grid will change all during the simulation. As already stated, the grid size is set as $\frac{1}{160}$; in the case of the uniform grid, it is applied to the entire domain, while in dynamic conditions it will only apply to the bubble interface.

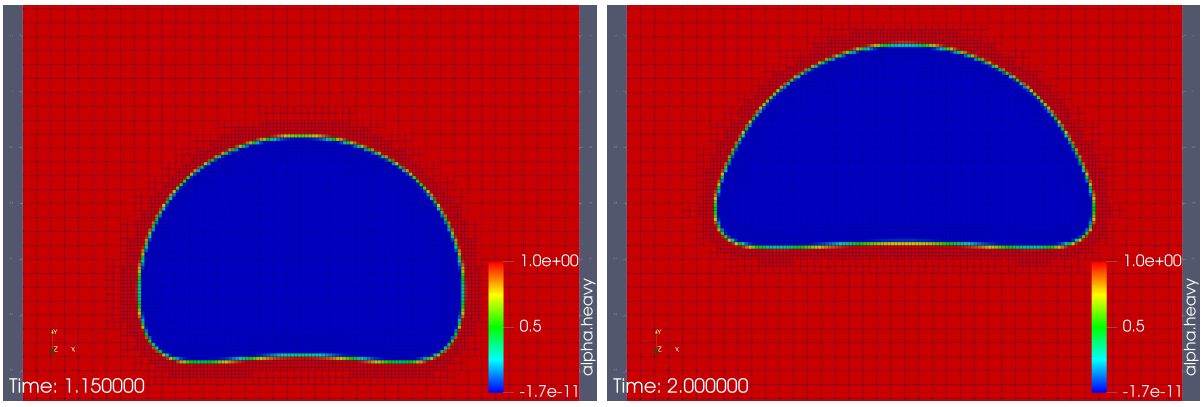



(a) Uniform mesh initial conditions. (b) Dynamic mesh initial conditions.

Figure 4.6: Close-up of initial conditions for the test cases. Red: heavy fluid. Blue: light fluid.

In figure 4.7 it is possible to observe an example of how the grid gets refined around the bubble and, at the same time, it gets unrefined where the bubble is no longer present during its rise. In this way, the bubble interface is captured in a very detailed manner, while all the other regions are studied more approximately. This is not a problem, since there are no effects on these other regions and are not needed in the study of the bubble.

The effect of AMR is mainly studied in the highest mesh resolution. Since the refinement adds some new levels in the grid, the starting mesh from which it is generated needs to be coarser, and the same resolution is to be intended near the interface, which is the main interest of the problem.




(a) Simulation snapshot at time=1.5s. (b) Simulation snapshot at time=2s.

Figure 4.7: Example of AMR in action during the simulation, with changes in mesh topology. Red: heavy fluid. Blue: light fluid.

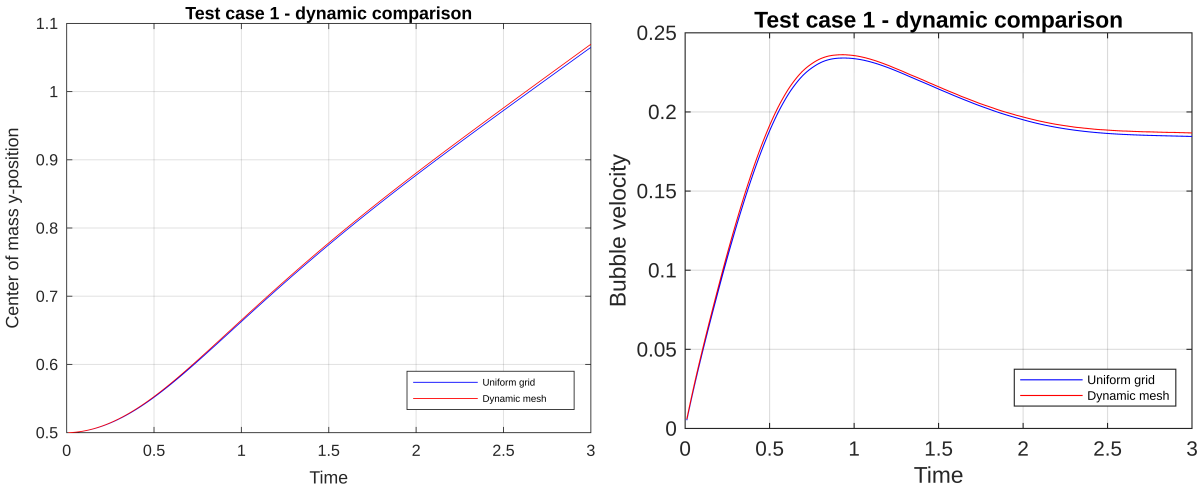
## Test case 1

In 4.3 the different computational costs for test 1 are displayed. Clearly, the number of cells is much smaller in the dynamic case when using AMR, with a peak of just 11.70% of those required for a uniform mesh when considering the same resolution around the interface. This also leads to a reduction in the disk space needed for saving simulation snapshots and data; while this is not extremely noticeable for this problem, it may be of interest when simulating larger and more intricate cases.

The execution time was obtained through the logs created by OpenFOAM applications and is obtained from equal machine conditions. In particular, both simulations were run on a single processor unit, to evaluate the effect of the newly developed 2D AMR without the effect of the added overhead from parallelization or other communications between processors. For this simulation, the execution time while considering the dynamic refinement was 15.23% of the one obtained from simulating with uniform grid, with nearly identical results as shown in figure 4.8.

| Test 1 - $h = \frac{1}{160}$ | uniform mesh | dynamic mesh |
|---|---|---|
| (Maximum) number of cells | 51 200 | 5 990 |
| Execution time | 3 251 s | 495 s |

Table 4.3: Computational resources required in uniform and dynamic simulations for test 1.




(a) Center of mass position evolution.

(b) Bubble rising velocity evolution.

Figure 4.8: Comparison between uniform grid and dynamic grid for test 1 (with $h = \frac{1}{160}$).

Comparing simulations with a higher grid size ($h = \frac{1}{40}$ and $h = \frac{1}{80}$) with dynamic refinement we find the solutions to be again very close to the ones obtained from a uniform grid size, with simulation resources needed reported in tables 4.4 and 4.5. In these cases, it is possible to see that the computational gain is not as evident as in the smaller grid size simulation. This is expected, since as cells get bigger, fewer and fewer cells are present in areas distant from the interface.

| Test 1 - $h = \frac{1}{40}$ | uniform mesh | dynamic mesh |
|---|---|---|
| (Maximum) number of cells | 3 200 | 1 190 |
| Execution time | 30 s | 14 s |

Table 4.4: Computational resources required in uniform and dynamic simulations for test 1.

| Test 1 - $h = \frac{1}{80}$ | uniform mesh | dynamic mesh |
|---|---|---|
| (Maximum) number of cells | 12 800 | 2 666 |
| Execution time | 324 s | 67 s |

Table 4.5: Computational resources required in uniform and dynamic simulations for test 1.

## Test case 2

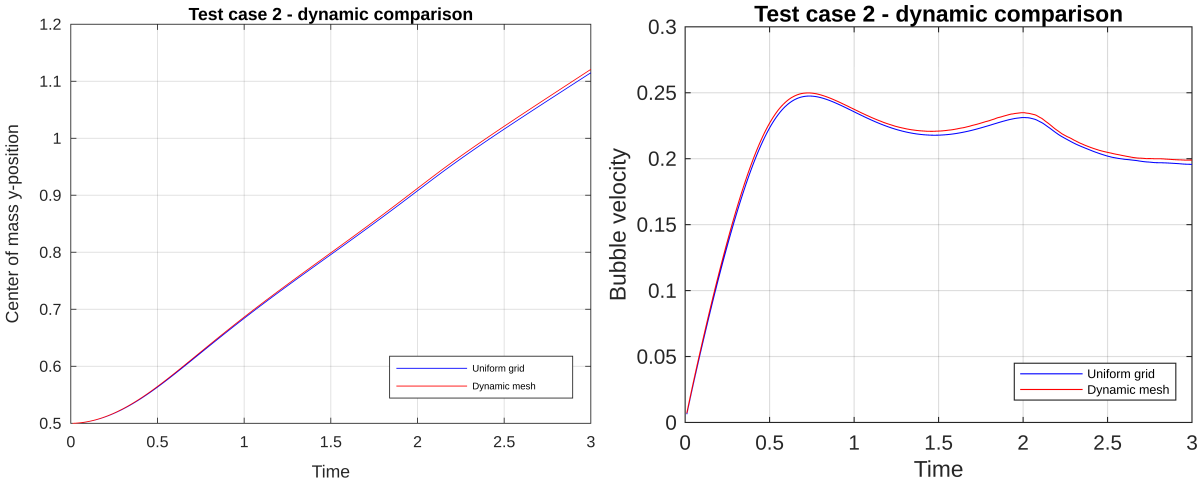
For the test 2 results, shown in figure 4.9, we obtain a similar situation when comparing the number of cells and the execution time in table 4.6. As in the previous test case, both simulations were run on a single processing unit in equal machine conditions. For this test, the simulation time of the dynamic case is just 16.60% of the uniform grid case.

| Test 2 - $h = \frac{1}{160}$ | uniform mesh | dynamic mesh |
|---|---|---|
| (Maximum) number of cells | 51 200 | 7 703 |
| Execution time | 3 446 s | 572 s |

Table 4.6: Computational resources required in uniform and dynamic simulations for test 2.

For this test case, the solution is very close to the benchmark one even when achieving

a great reduction of computational cost, validating the proposed 2D AMR as a reliable method for simulation of this kind of problem even in the case of interface breaking.




(a) Center of mass position evolution.

(b) Bubble rising velocity evolution.

Figure 4.9: Comparison between uniform grid and dynamic grid for test 2 (with $h = \frac{1}{160}$).

Again, when comparing the results from dynamic grid simulations at a bigger grid size, we find very close results but a less relative gain on computational cost, as reported in tables 4.7 and 4.8.

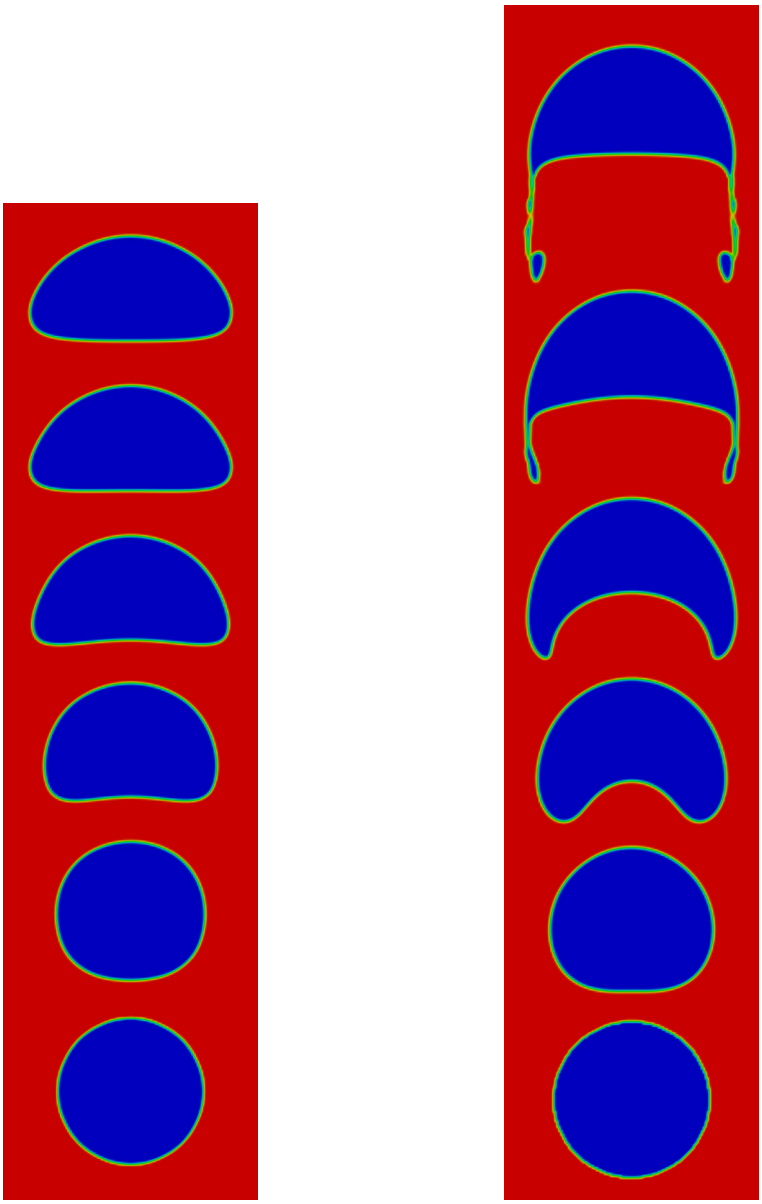
| Test 2 - $h = \frac{1}{40}$ | uniform mesh | dynamic mesh |
|---|---|---|
| (Maximum) number of cells | 3 200 | 1 346 |
| Execution time | 29 s | 15 s |

Table 4.7: Computational resources required in uniform and dynamic simulations for test 2.

| Test 2 - $h = \frac{1}{80}$ | uniform mesh | dynamic mesh |
|---|---|---|
| (Maximum) number of cells | 12 800 | 3 080 |
| Execution time | 336 s | 74 s |

Table 4.8: Computational resources required in uniform and dynamic simulations for test 2.

Finally, in figure 4.10 it is possible to observe the different shapes the bubble takes on in the different test cases. As already predicted, the bubble in the first case maintains an elliptical shape (4.10a), while in the second case the bubble breaks visible with streaks and smaller fragments leaning on both sides (4.10b). The snapshots are taken from the simulations at the same simulation time, once every 0.6 time units. However, they are rearranged to make the shape more visible, spacing them more than the actual simulations.



(a) Bubble shape evolution in test 1.

(b) Bubble shape evolution in test 2.

Figure 4.10: Shape evolution of the bubble in the two cases. Rearranged snapshots taken at every 0.6 time units. Red: heavy fluid. Blue: light fluid.

# 5 | 2D non-Newtonian rising bubble

The focus of this chapter is the study and simulation of non-Newtonian fluids. They play an important role in many different fields and environments, due to the variety of fluids that belong in this category. In particular, following the research of [24], bubbly flow is a critical point in many applications. The simulations proposed will picture the movement of a gas bubble rising in a non-Newtonian surrounding fluid.

A non-Newtonian fluid, as the name suggests, does not follow the Newtonian law of viscosity, that is its viscosity varies with the applied stress. The study considers two different types of non-Newtonian rheologies, shear-thinning and shear-thickening liquids, which respectively lower and increase their viscosity with the rate of shear strain.

Again, a 2D simulation model will be implemented. This is supported by literature studies ([25]) which highlight that non-Newtonian fluids tend to move in a 2D path rather than a 3D one, especially when considering high viscosity fluids.

## 5.1. Viscosity model

The starting model considered for describing the relation between shear strain rate and viscosity is the power law model:

$$\eta = K(\dot{\gamma})^{n-1} \tag{5.1}$$

where $\eta$ is the viscosity, $K$ is the flow consistency index, $n$ is the flow behaviour index and $\dot{\gamma}$ is the shear strain rate.

This formulation may incur in several problems in some cases, for example, the viscosity may become infinite in a stagnant shear-thinning liquid [24]. Thus, to keep the simulation stable near zero, a more appropriate formulation for the viscosity is the truncated power

law [26]:

$$\eta = \eta(\dot{\gamma}) = \begin{cases} \eta_0, & \dot{\gamma} < \dot{\gamma}_0 \\ K((\dot{\gamma}))^{n-1}, & \dot{\gamma}_0 \leq \dot{\gamma} \leq \dot{\gamma}_\infty \\ \eta_\infty, & \dot{\gamma} > \dot{\gamma}_\infty \end{cases} \tag{5.2}$$

which keeps the viscosity between two limited values $\eta_0$ and $\eta_\infty$ in relation of the thresholds on the shear strain rate $\dot{\gamma}_0$ and $\dot{\gamma}_\infty$.

## 5.2. Definition of test cases

Many different cases can be constructed by varying the parameters described in the truncated power law model. In fact, shear-thinning fluids will be considered with flow behaviour index $n = 0.8$, shear-thickening fluids with flow behaviour index $n = 1.2$, and for both cases the consistency index $K$ can vary between $\mu_{water}$, $10\mu_{water}$ and $100\mu_{water}$. Different bubble sizes will be considered, choosing from a diameter of 2 mm, 4 mm or 6 mm, leading to widely different behaviours in the various cases. All the parameters and sizes are chosen in line with the reference work [24].

The domain will be a rectangle with a height of 100 mm and a width of 50 mm, with a single air bubble of the chosen diameter situated at [25 mm, 10 mm] at the starting time. Regarding the grid size, uniform cells with size 0.25 mm x 0.25 mm will be used when considering a uniform grid, following the study in section 5.3.1. In section 5.3 the AMR will affect the number and dimension of the starting cells, keeping the size of the uniform grid along the interface.

In all cases, a final time $T = 0.3$ s is chosen, with a time-step $\Delta t = 10^{-5}$ s.

## 5.3. Results

### 5.3.1. Grid convergence

Firstly, one of the test cases has been simulated in various grid conditions to check if the solution converges. In particular, a bubble of diameter $d = 4$ mm rising in a shear-thickening fluid (flow behaviour index $n = 1.2$) with a consistency index $K = 100\mu_{water}$ has been chosen. The grid sizes and total cells in the simulations are reported in table 5.1, all with a time-step $\Delta t = 10^{-5}$ s.

| $h$ | number of cells |
|---|---|
| 0.5 mm | 20 000 |
| 0.25 mm | 80 000 |
| 0.125 mm | 320 000 |

Table 5.1: Different settings for grid convergence simulations.

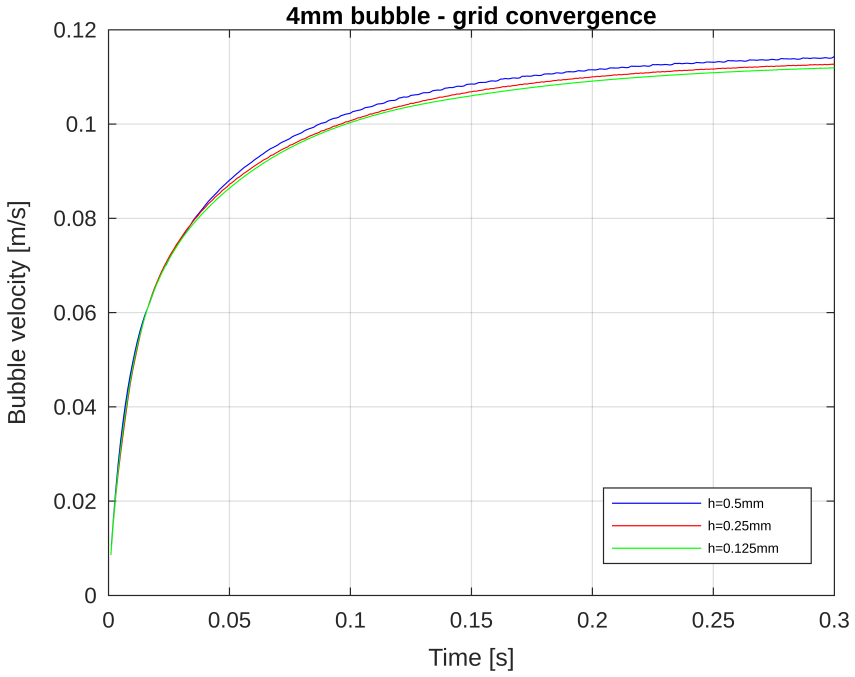



Figure 5.1: Results in different grid settings for a 4 mm bubble in shear-thickening fluid, $K = 100\mu_{water}$.

In figure 5.1 we can see that the solution converges to the one with a terminal velocity of around 0.11 m/s. Given the results, the difference between the grid with $h = 0.25$ mm and $h = 0.125$ mm is less than 1%, so the chosen computational mesh will be $h = 0.25$ mm, to save computational resources.

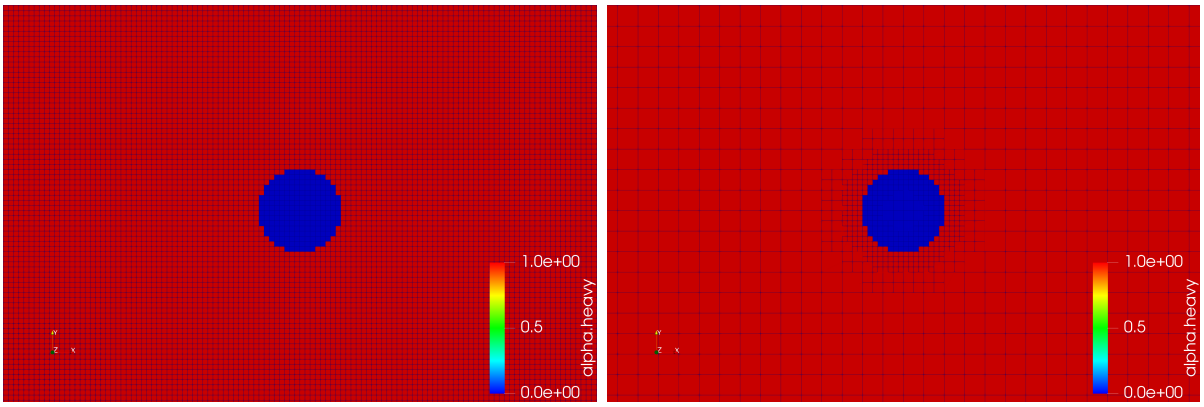
However, when comparing these results with benchmark values, a difference in terminal velocity is detected. In fact, these simulations present an overestimate of the terminal velocity of the bubble: 0.1119 m/s against 0.0811 m/s [24]. This difference could be due to a number of factors: possibly, the different applications used in the respective works (OpenFOAM-10 in this work, ANSYS Fluent 2019 R3 in [24]), or a different method used for the discretization of the equations.

It is important to note that for smaller bubbles with diameter $d = 2$ mm a finer grid is needed to keep the same interface accuracy, and a finer grid size $h = 0.125$ mm will be chosen.

### 5.3.2. AMR effect

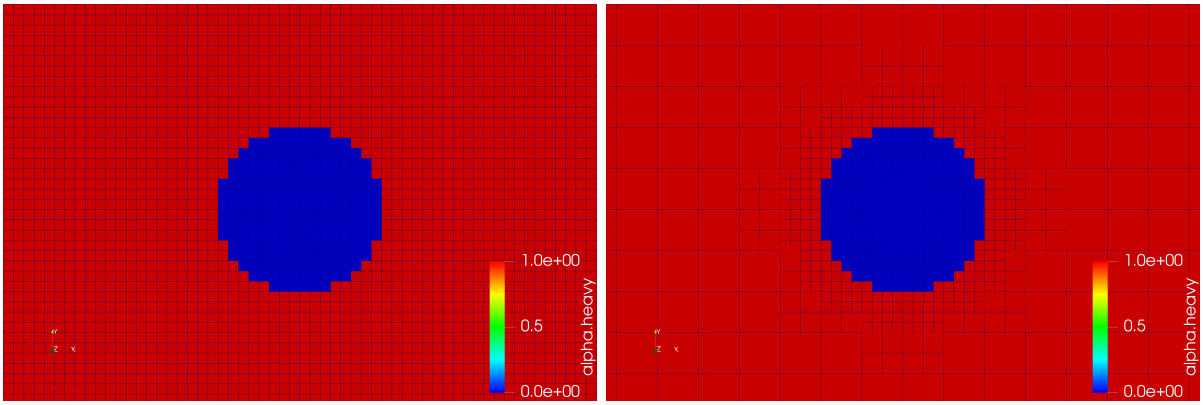
In this chapter both the effect of the grid-refining technique described in section 3.5 and the differences in Newtonian and non-Newtonian behaviour will be discussed.

The starting conditions for the uniform and dynamic refined grid are depicted in figures 5.2, 5.3 and 5.4. As in the previous dynamic simulations, the grid size of the dynamic grid is to be intended only around the interface of the bubbles, while in the outer region the mesh is allowed to be coarser. Again, the mesh will be unrefined once the interface has completely passed through the coarser cells.



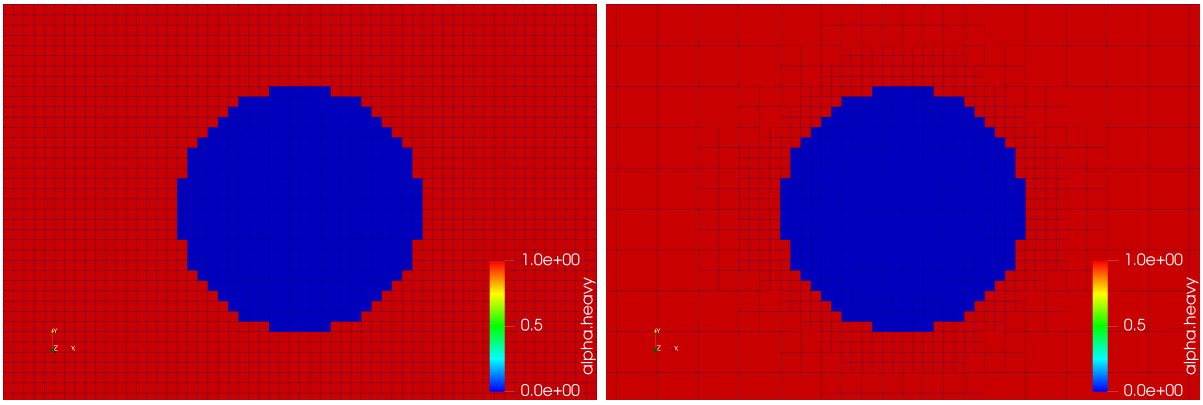

(a) Uniform mesh initial conditions. (b) Dynamic mesh initial conditions.

Figure 5.2: Close-up of initial conditions for the 2 mm diameter bubble cases in static and dynamic simulations.




(a) Uniform mesh initial conditions. (b) Dynamic mesh initial conditions.

Figure 5.3: Close-up of initial conditions for the 4 mm diameter bubble cases in static and dynamic simulations.




(a) Uniform mesh initial conditions. (b) Dynamic mesh initial conditions.

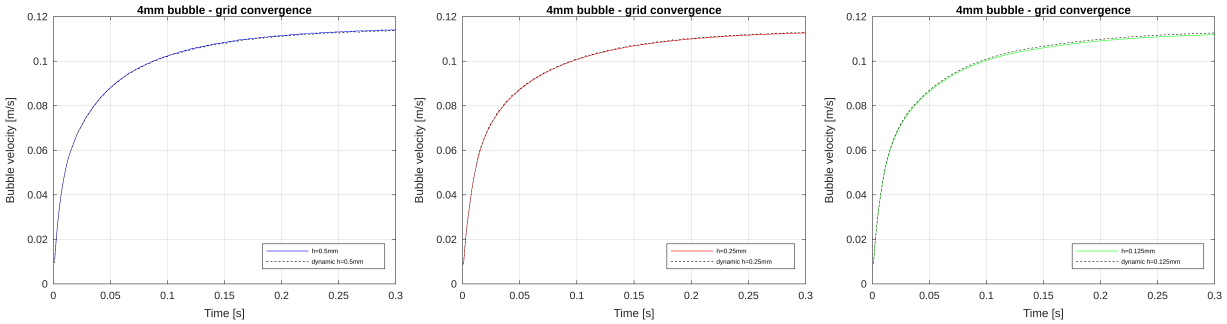
Figure 5.4: Close-up of initial conditions for the 6 mm diameter bubble cases in static and dynamic simulations. Grid size $h = 0.125$ mm.

## Test - 4mm diameter bubble in high-viscosity shear-thickening fluid

It is possible to compare how the dynamic simulations fare against the uniform grid ones by comparing the same case used for the grid convergence, that is a 4 mm diameter bubble in a shear-thickening fluid.

In figure 5.5 it is possible to observe how the dynamic solutions align with the ones obtained from the uniform grid simulations. In particular, there is no noticeable difference

between the two solutions in the case with grid size $h = 0.25$ mm chosen as a reference for every simulation.




(a) Grid size $h = 0.5$ mm. (b) Grid size $h = 0.25$ mm. (c) Grid size $h = 0.125$ mm.

Figure 5.5: Comparison of the rising velocity of a 4 mm diameter bubble in a shear-thickening fluid, $K = 100\mu_{water}$, in different grid settings.

| $h = 0.5$ mm | uniform mesh | dynamic mesh |
|---|---|---|
| Number of processors | 4 | 1 |
| (Maximum) number of cells | 20 000 | 1520 |
| Execution time | 708 s | 240 s |

Table 5.2: Computational resources required in uniform and dynamic simulations for a 4 mm diameter bubble in a shear-thickening fluid, $K = 100\mu_{water}$ in a coarse grid.

| $h = 0.25$ mm | uniform mesh | dynamic mesh |
|---|---|---|
| Number of processors | 4 | 1 |
| (Maximum) number of cells | 80 000 | 5 510 |
| Execution time | 3 653 s | 837 s |

Table 5.3: Computational resources required in uniform and dynamic simulations for a 4 mm diameter bubble in a shear-thickening fluid, $K = 100\mu_{water}$.

| $h = 0.125$ mm | uniform mesh | dynamic mesh |
|---|---|---|
| Number of processors | 16 | 1 |
| (Maximum) number of cells | 320 000 | 20 966 |
| Execution time | 7 426 s | 4 326 s |

Table 5.4: Computational resources required in uniform and dynamic simulations for a 4 mm diameter bubble in a shear-thickening fluid, $K = 100\mu_{water}$ in a fine grid.

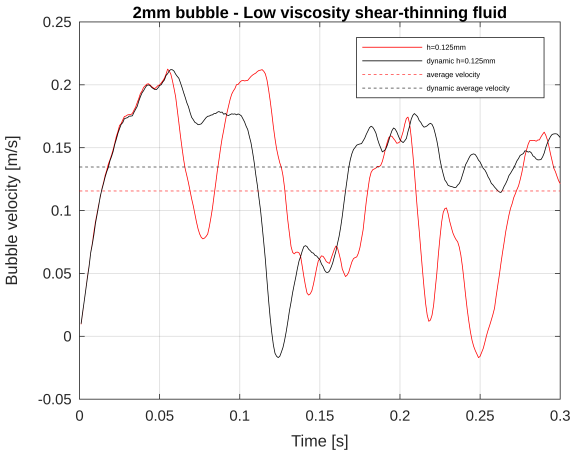
When comparing the different computational costs between the two approaches, in tables 5.2, 5.3 and 5.4, it is clear how much of an impact the 2D AMR has on the execution time and the memory saving due to the number of cells.

The decision on the number of processors was made for technical reasons: it was not possible to run the uniform grid cases on a single processor in a reasonable time, and it was not meaningful to simulate the dynamic cases on multiple processors due to the small number of cells they present. In any case, some meaningful insights can be gained by the cost comparison. In fact, even when running the simulations in parallel with a lot more processors, instead of just one used in the dynamic cases, the execution time is still greater than the one obtained with the dynamic refinement.

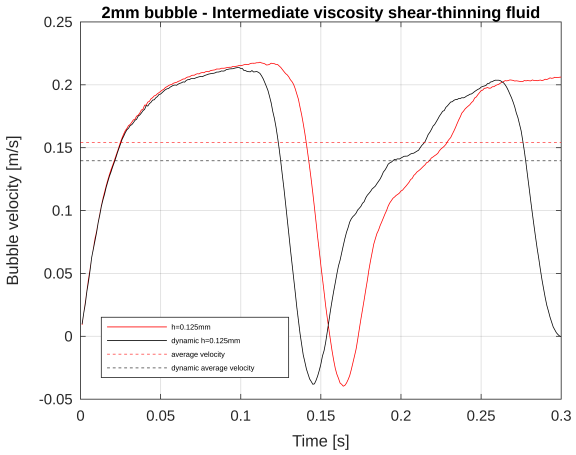
### Test - 2mm diameter bubble in shear-thinning fluid, various viscosities

Another case with different parameters has been simulated: a bubble of diameter $d = 2$ mm rising in a shear-thinning fluid (flow behaviour index $n = 0.8$) in different viscosities ($K = \mu_{water}$, $K = 10\mu_{water}$ and $K = 100\mu_{water}$) on both uniform grids and dynamically refined grids.

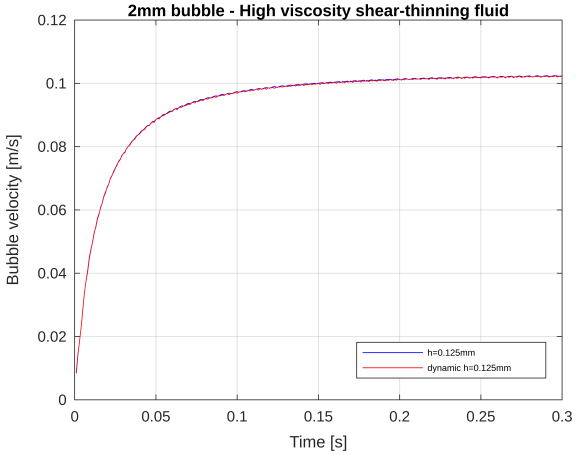
The results of these cases are depicted in figure 5.6. In particular, the low (5.6a) and intermediate (5.6b) viscosity cases present an oscillating behaviour. For these situations, also the average velocity of the bubble rise is reported. The most visible differences between the two approaches are in low viscosity environments, while in intermediate and high viscosity conditions the bubble follows the same trend both in trajectory and velocity.




(a) Low viscosity case $K = \mu_{water}$.




(b) Intermediate viscosity case $K = 10\mu_{water}$.




(c) High viscosity case $K = 100\mu_{water}$.

Figure 5.6: Comparison of the rising velocity of a 2 mm diameter bubble in a shear-thinning fluid, in different viscosity settings, with grid size $h = 0.125$ mm.

The cause of the oscillations is some movement of the bubble during its rise. In figure 5.7 it is possible to see the cause of those oscillations in figure 5.6b for the bubble of diameter $d = 2$ mm in a shear-thinning fluid in medium viscosity condition. The sudden drop in the velocity field is due to the horizontal movement and consequent loss in vertical velocity of the bubble. Due to this behaviour, the solution loses its symmetry even though there are no external forces applied on the bubble due to the non-Newtonian behaviour of the outer fluid. The image represents both the shape and the trajectory of the bubble during the simulation.

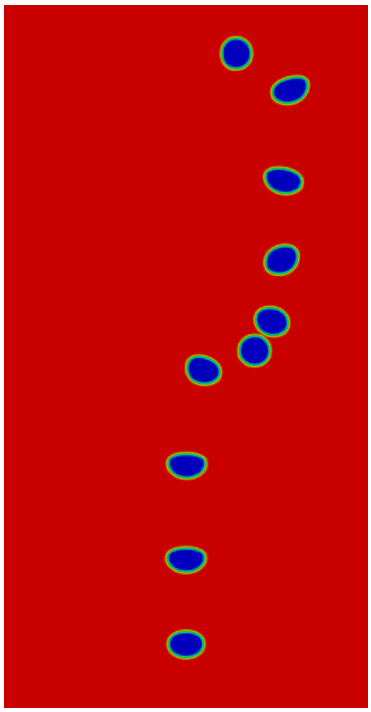


Figure 5.7: Bubble trajectory and shape evolution for $d = 2$ mm, shear-thinning fluid, $K = 10\mu_{water}$. Red: heavy fluid. Blue: light fluid. Snapshots taken every 0.3 s.

The behaviour in all three cases is coherent with what is observed by the reference benchmark [24]. In fact, small ($d = 2$ mm) bubbles follow a very oscillating path when rising in a low viscosity surrounding fluid, with the effect still noticeable at intermediate viscosities.

## Test - 4mm diameter bubble in shear-thinning and shear-thickening fluids

Finally, more tests were simulated with a bubble of diameter $d = 4$ mm in non-Newtonian fluids at different viscosities.

In figure 5.8 solutions for different viscosities of shear-thinning fluids are reported. In particular, a comparison between dynamic and uniform grids is done. During most of the simulations, the dynamic solution is coherent with the uniform one, with some differences only due to instabilities and oscillations of the bubble at lower and intermediate viscosities.

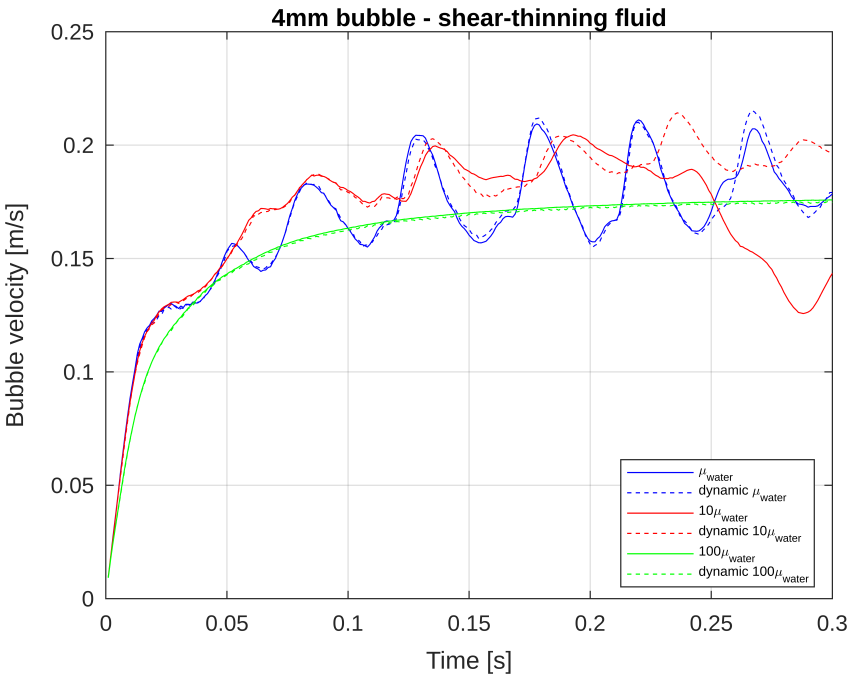



Figure 5.8: Rising velocity of a 4 mm diameter bubble in a shear-thinning fluid, in different viscosity settings, with grid size $h = 0.25$ mm. Comparison between uniform and dynamic simulations.

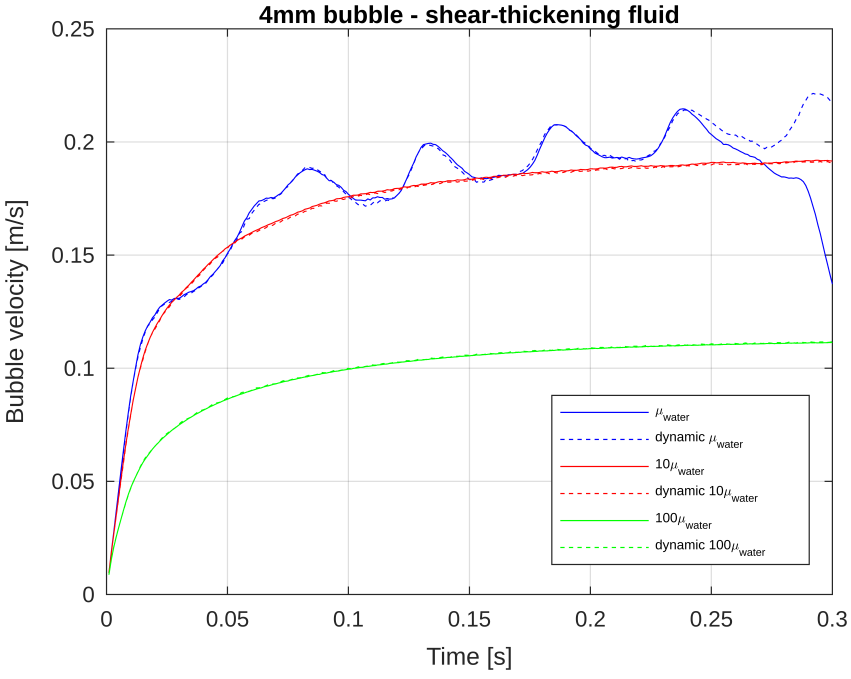



Figure 5.9: Rising velocity of a 4 mm diameter bubble in a shear-thickening fluid, in different viscosity settings, with grid size $h = 0.25$ mm. Comparison between uniform and dynamic simulations.

In figure 5.9 the same comparison is done for shear-thickening fluids. In this case, the dynamic solution is even closer to the uniform values, except for a small instability appearing at low viscosity.

In figure 5.10 it is possible to observe the oscillations in shape (reflected in the velocity graph) of a 4 mm diameter bubble in a low-viscosity shear-thickening fluid.

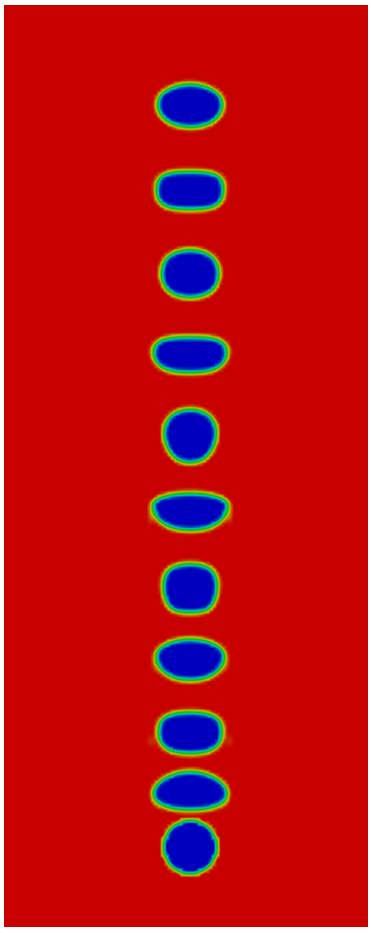


Figure 5.10: Bubble trajectory and shape evolution for $d = 4$ mm, shear-thickening fluid, $K = \mu_{water}$. Red: heavy fluid. Blue: light fluid. Snapshots taken every 0.3 s.

Again, the behaviour of the bubbles reflects what is observed in the reference benchmark [24]. In particular, oscillating effects are detected at low and intermediate viscosities in the shear-thinning case, while a more linear trajectory is observed in shear-thickening fluids. In the latter case, when considering a low viscosity environment, the oscillations are no longer due to oscillations in trajectory but rather in the deformation of the bubble during its rise.

# 6 | 3D Newtonian rising bubble

This chapter will focus on the study and analysis of a 3D flow, as opposed to a 2D flow of the previous chapters. It is based on the work of [23] and will once again see the rise of a bubble in a column of heavier fluid. Moreover, the setup will be similar to the one seen in chapter 4 and [21], as the work this section is based on originates from the same problem.

This study will be important to assess the efficiency of the newly proposed 2D AMR compared to the already present 3D one, and also a different approach using an axisymmetrical domain will be explored.

## 6.1. Definition of test cases

Two test cases will be analyzed, situated in the same domain and starting conditions but with different fluid physical parameters.

The domain $\Omega$ is depicted in figure 6.1. It is a cuboid of dimension [0,1] x [0,2] x [0,3] with a bubble of light fluid $\Omega_2$ centered in $\mathbf{x}_c = (0.5, 0.5, 0.5)^T$ of radius $r = 0.25$ immersed in a heavier fluid $\Omega_1$. An initial condition $\mathbf{u}(0) = 0$ is assumed in $\Omega$ and a no-slip condition ($\mathbf{u} = 0$) will be imposed on $\partial\Omega$.

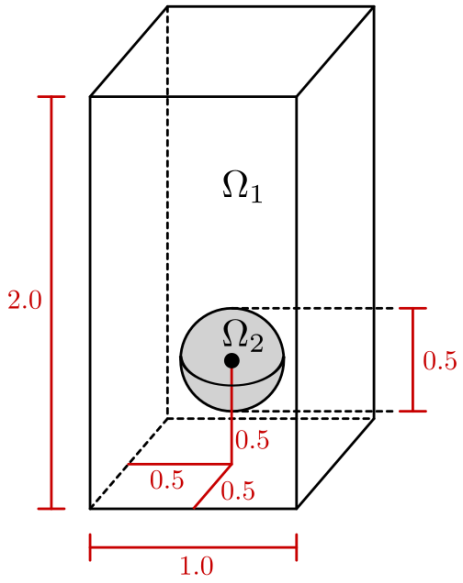



Figure 6.1: Initial configuration for the test cases [23].

The parameters will be the same as in the case of the 2D benchmark [21], and are reported in table 6.1. Again, the first test case will see an ellipsoidal bubble regime, while in the second one the bubble interface may break during its rise.

The time frame for the simulations will be different: for test case 1, a final time of $T = 3$ is considered, while for test case 2 the final time will be $T = 3.5$.

| Test case | $\rho_1$ | $\rho_2$ | $\mu_1$ | $\mu_2$ | $g$ | $\tau$ |
|---|---|---|---|---|---|---|
| 1 | 1000 | 100 | 10 | 1 | 0.98 | 24.5 |
| 2 | 1000 | 1 | 10 | 0.1 | 0.98 | 1.96 |

Table 6.1: Physical parameters and dimensionless numbers defining the test cases [23].

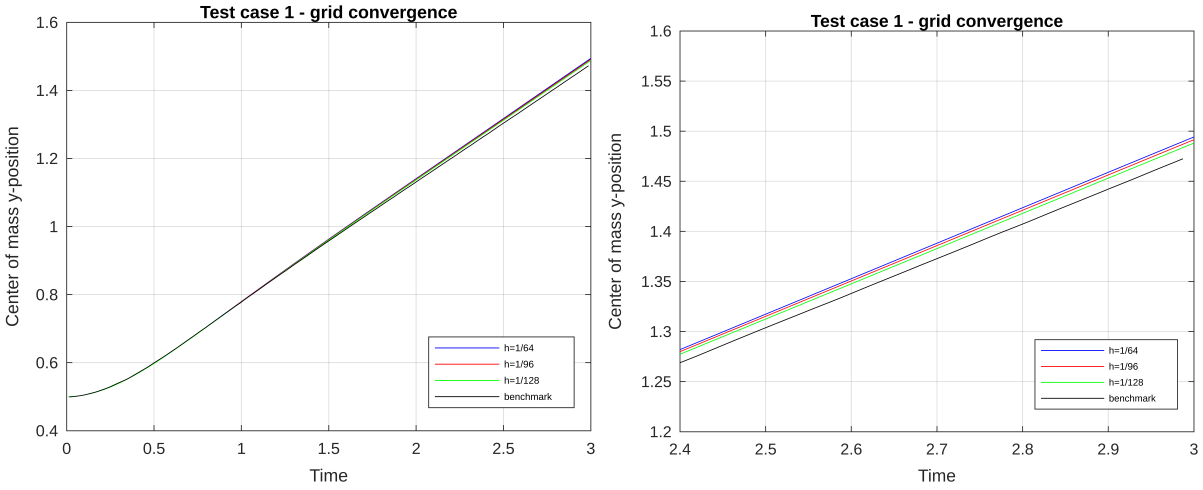
## 6.2. Results

### 6.2.1. Grid convergence

Firstly, the test cases have been simulated in various grid conditions to check if the solution converges to the one presented in the benchmark. The different grid sizes are reported in table 6.2 and have been simulated with a time-step of $\Delta t = 10^{-4}$. Of course, the number of cells increases much faster than in the 2D case, leading to heavier and longer simulations.

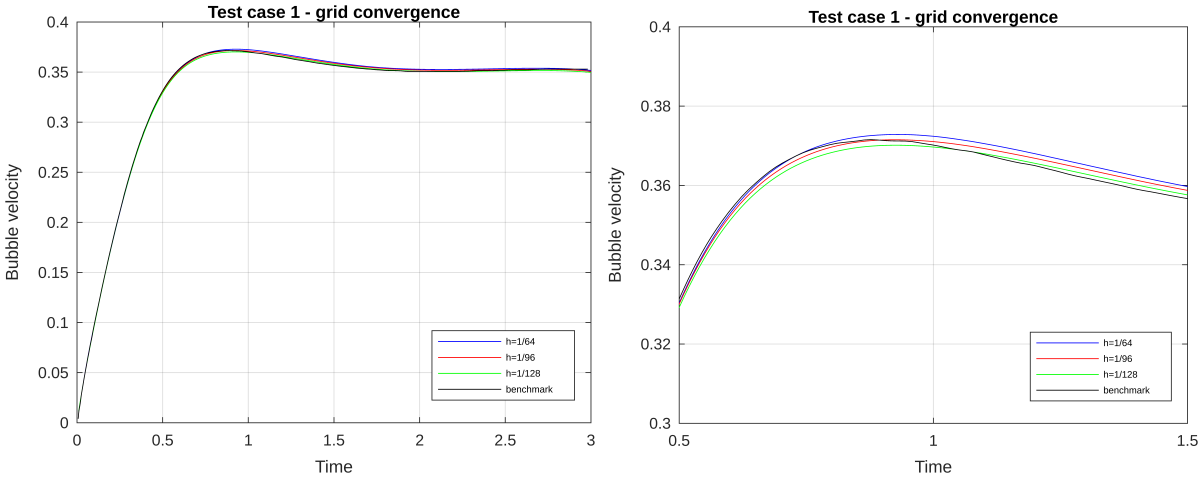
| $h$ | number of cells |
|---|---|
| $\frac{1}{64}$ | 524 288 |
| $\frac{1}{96}$ | 1 769 472 |
| $\frac{1}{128}$ | 4 194 304 |

Table 6.2: Different settings for grid convergence simulations.




(a) Convergence of center of mass position evolution. (b) Close-up of the results.

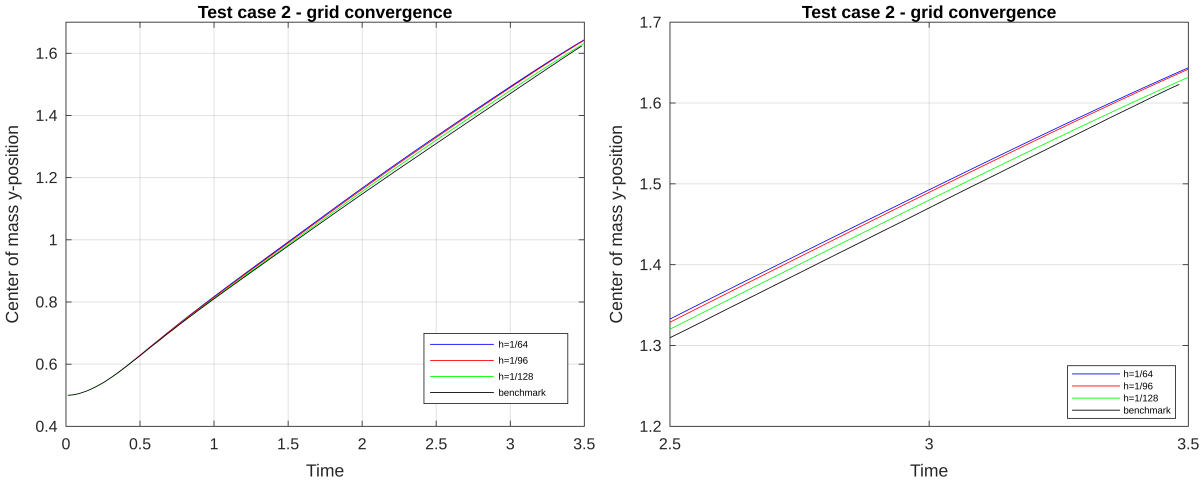
Figure 6.2: Center of mass convergence for the different grids in test 1. Benchmark results from [23].




(a) Convergence of velocity evolution. (b) Close-up of the results.

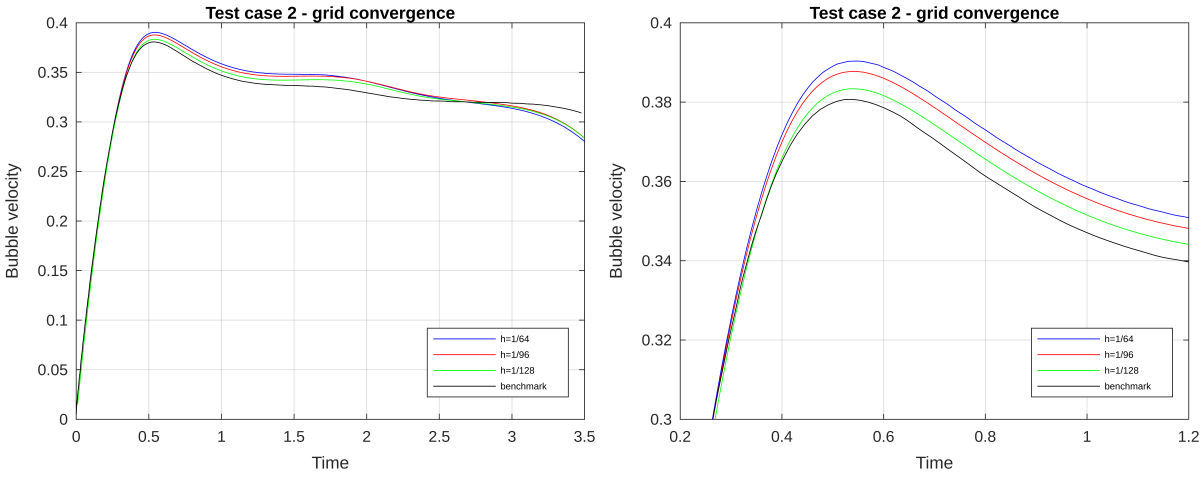
Figure 6.3: Bubble velocity convergence for the different grids in test 1. Benchmark results from [23].

As seen in figures 6.2 and 6.3 there is not an exact convergence to the benchmark solution. However, the results from the simulation seem to converge to a solution close to it in a monotone way. Possibly even more cells may be needed to reach the benchmark solution but reducing the grid size couldn't be done in this work due to hardware and simulation time constraints.




(a) Convergence of center of mass position evolution. (b) Close-up of the results.

Figure 6.4: Center of mass convergence for the different grids in test 2. Benchmark results from [23].




(a) Convergence of velocity evolution. (b) Close-up of the results.

Figure 6.5: Bubble velocity convergence for the different grids in test 2. Benchmark results from [23].

Simulations from test case 2 yield a similar result as seen in figures 6.4 and 6.5. In

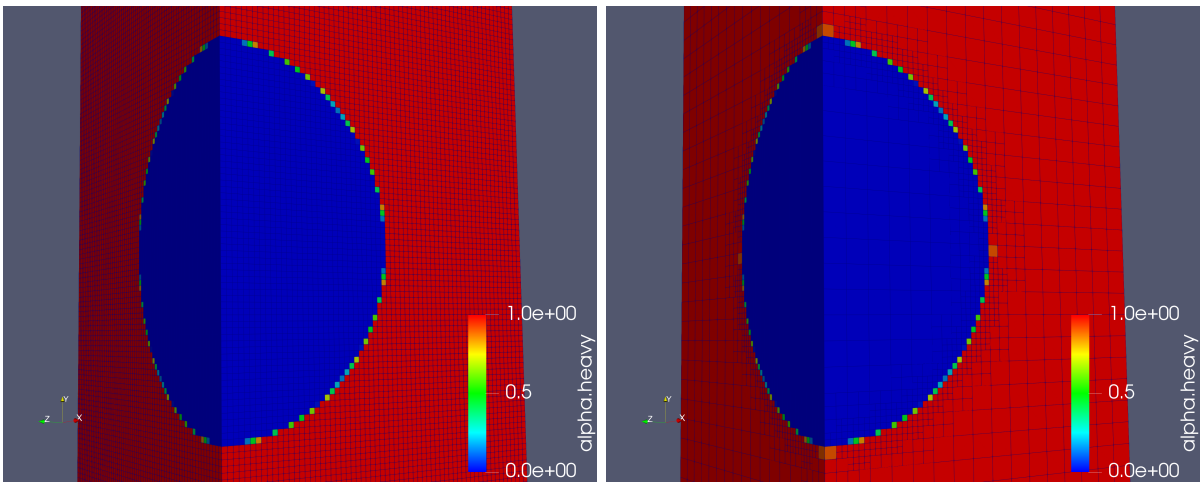
particular, since the bubbles rise faster, the effect of the upper wall in the domain, on which a no-slip condition is imposed, is much more noticeable in the final part of the simulation.

In any case, the reference for the simulated solution will be the one obtained with grid size $h = \frac{1}{128}$.

### 6.2.2. AMR effect

The effect of the native AMR will be studied in this chapter. This is done to assess the validity of the proposed 2D version by comparing if a similar decrease in computational cost is observed during the simulations.

Again, the refinement only happens around the interface of the bubble, creating more cells in the area needed for the study. For this reason, the grid size of the dynamic refined grid will have the same precision as the uniform one only around the bubble interface, the only region which requires more precision during the simulation. In figure 6.6 we can see the starting conditions for the bubbles in a quarter of the domain, to see the three-dimensional nature of the bubble. In figure 6.6a we can see conditions for the uniform grid and in figure 6.6b for the dynamic grid.




(a) Uniform mesh initial conditions. (b) Dynamic mesh initial conditions.

Figure 6.6: Section of the domain and close-up of initial conditions for the test cases. Red: heavy fluid. Blue: light fluid.

Simulations for both test cases were run in parallel on 12 processors on the same machine in equal conditions. The same number of processors was used for both uniform and dynamically refined cases. This is done also in the latter setting since, even when refining,

a lot of computational time is needed. However, the relatively low number of cells could undermine the parallelization effectiveness, and therefore fewer processors could yield a similar total execution time.

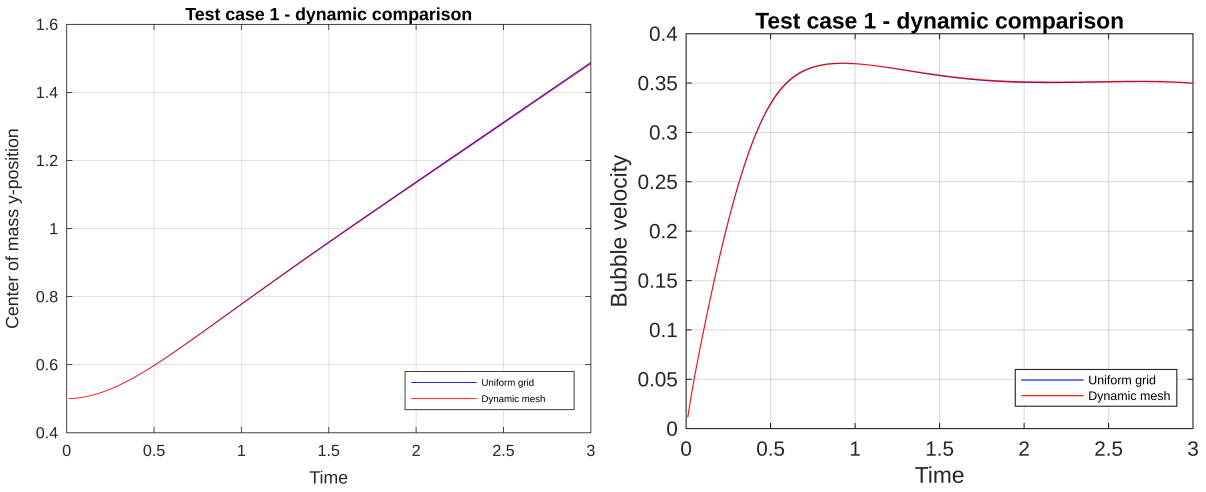
## Test case 1

In table 6.3 we can see the number of cells and computational time needed for the test case 1 simulation. The dynamic case was able to obtain results in 11.22% of the time the uniform grid case needed. It is very comparable with the time reduction experienced in the 2D case, confirming that the proposed method is a valid one.

| Test 1 | uniform mesh | dynamic mesh |
|---|---|---|
| (Maximum) number of cells | 4 194 304 | 197 045 |
| Execution time | 402 265 s | 45 143 s |

Table 6.3: Computational resources required in uniform and dynamic simulations for test 1.

By comparing the results between the uniform grid and dynamic one in figure 6.7 it is possible to see that they are extremely close in both centre of mass position (6.7a) and bubble velocity (6.7b), underlining that the AMR is a valid tool for reducing the computational cost of simulations.




(a) Center of mass position evolution.

(b) Bubble rising velocity evolution.

Figure 6.7: Comparison between uniform grid and dynamic grid for test 1 (with $h = \frac{1}{128}$).

## Test case 2

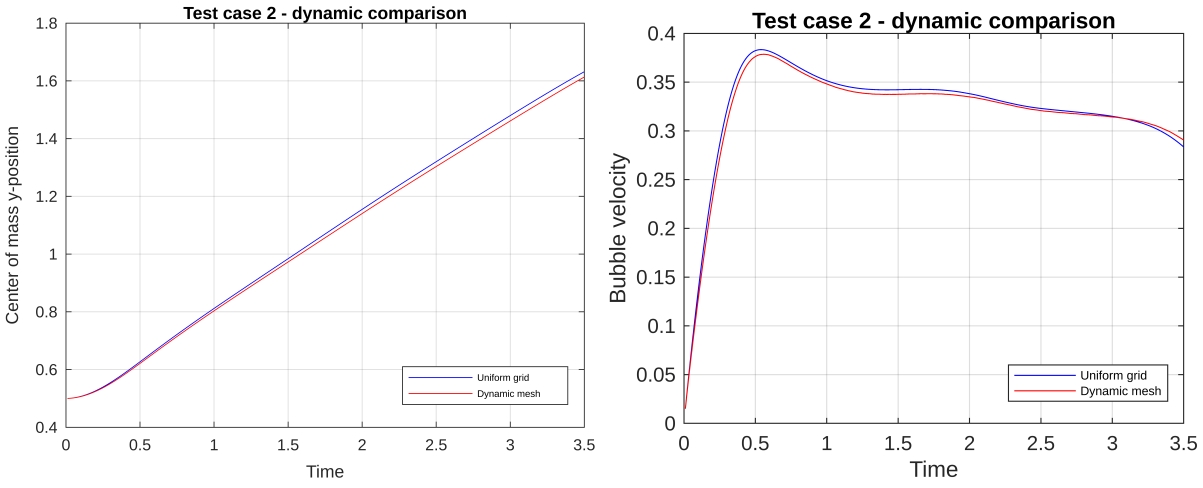
For this test case, an even more radical approach to saving computational costs was taken. While the simulations obtained with AMR show almost identical results as in test 1, more simulations were run with both the use of AMR and automatic adjustment of the time-step. This allows the solver to modify the time-step, in this case by using a larger one as the simulation runs, based on the Courant number [18].

In table 6.4 we can see the number of cells and computational time needed for the test case 2 simulation, recalling that the low execution time of the dynamic test is due to both AMR and adjustable time-step effects.

| Test 1 | uniform mesh | dynamic mesh and adjustable time-step |
|---|---|---|
| (Maximum) number of cells | 4 194 304 | 282 697 |
| Execution time | 429 438 s | 6 509 s |

Table 6.4: Computational resources required in uniform and dynamic simulations for test 2.

In this case, the results for the centre of mass position (figure 6.8a) and bubble velocity (figure 6.8b) present some differences, due to the adjustment of the time-step which was too aggressive. However, they are still close enough to give a correct representation, even if not exact, of the behaviour of the bubble.




(a) Center of mass position evolution.

(b) Bubble rising velocity evolution.

Figure 6.8: Comparison between uniform grid and dynamic grid with adjustable time-step for test 2 (with $h = \frac{1}{128}$).

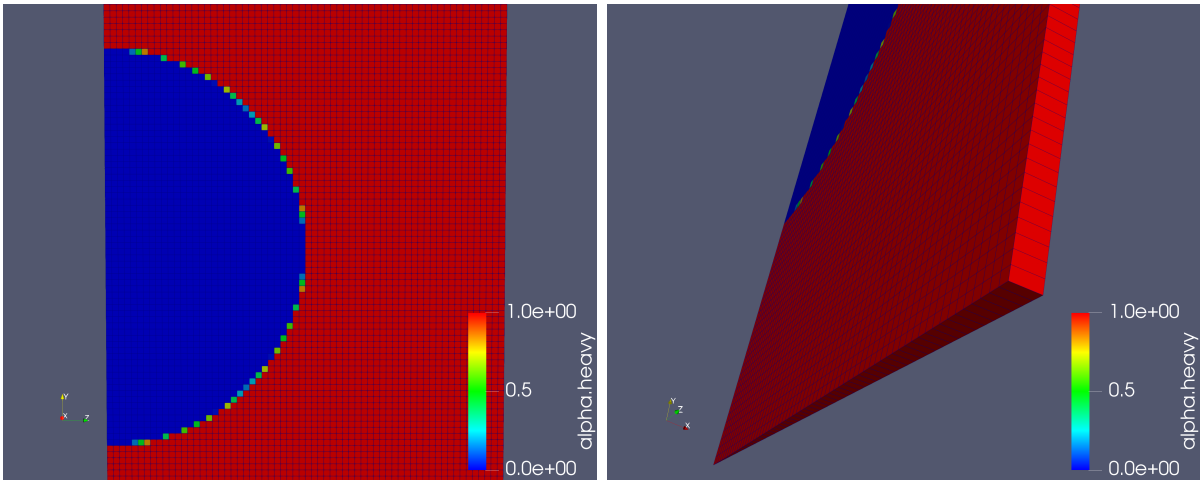
## 6.3. Axisymmetrical domain

Finally, a different approach to solving the problem was taken. Instead of solving the problem in a 3D domain, an axisymmetric one will be chosen, resulting in a wedge-shaped slice. The problem is obviously symmetric around a vertical axis passing through the centre of the bubble, since we have a centred sphere as initial condition and there are no other sources of movement or instabilities during the simulation.

The axisymmetric condition is already supported in OpenFOAM and modifies the 3D problem to a 2D one in axial and radial directions by considering a single slice of trapezoidal cells. This will greatly reduce the computational cost of the simulation, thanks to the much smaller number of cells in the domain. Of course, an error is introduced since the cells in the normal direction to the radial one are flat. However, this error is negligible for angles of around 1° [27].

The domain was created starting from the shape of the 3D test cases, taking just a 2° slice from it (with an angle width of 1° for each side). When visualizing the results, the bubble shape can be reconstructed by replicating the domain around its axis until the whole round angle has been completed.

The grid parameters are the same as the 3D case, with a grid size $h = \frac{1}{128}$ and a time-step $\Delta t = 10^{-4}$, to keep consistency with the previous cases. The domain and starting conditions are depicted in figure 6.9.




(a) Wedge domain initial conditions. (b) Wedge shape.

Figure 6.9: Close-up of initial conditions and domain shape for the wedge domain test cases. Red: heavy fluid. Blue: light fluid.

## Test case 1

In figures 6.10 and 6.11 it is possible to see how the solution in these conditions compares to both the benchmark one and the three-dimensional one for test 1.

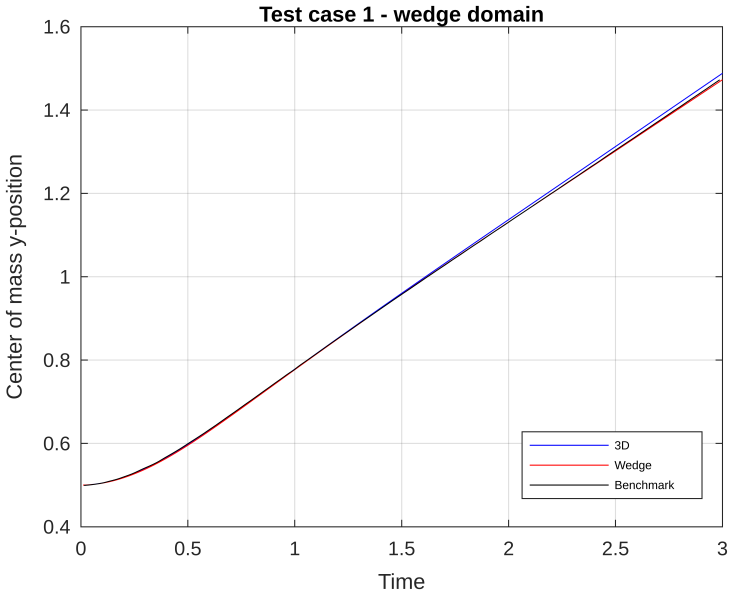



Figure 6.10: Center of mass position evolution. Comparison between uniform grid and wedge case for test 1 (with $h = \frac{1}{128}$). Benchmark values from [23].

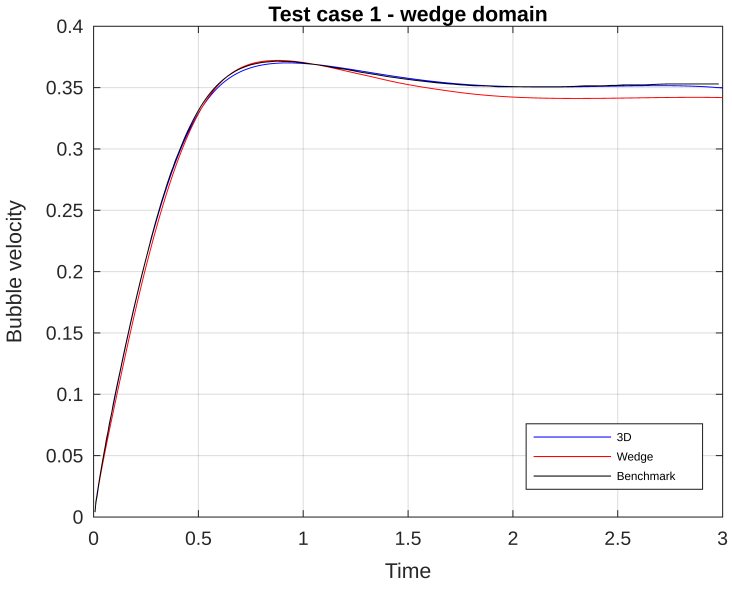



Figure 6.11: Bubble velocity evolution. Comparison between uniform grid and wedge case for test 1 (with $h = \frac{1}{128}$). Benchmark values from [23].

The solution is close to the benchmark one, though a small difference in position and

velocity is notable. The difference with the purely 3D case might be due to the differences between the two domains: in the 3D case it is a cuboid, while in the axisymmetric case it is a cylinder. Hence, the walls on which a no-slip condition is applied may have a non-negligible effect on the bubble and its evolution during the simulation.

## Test case 2

In figures 6.12 and 6.13 it is possible to see how the solution in these conditions compares to both the benchmark one and the three-dimensional one for test 2.

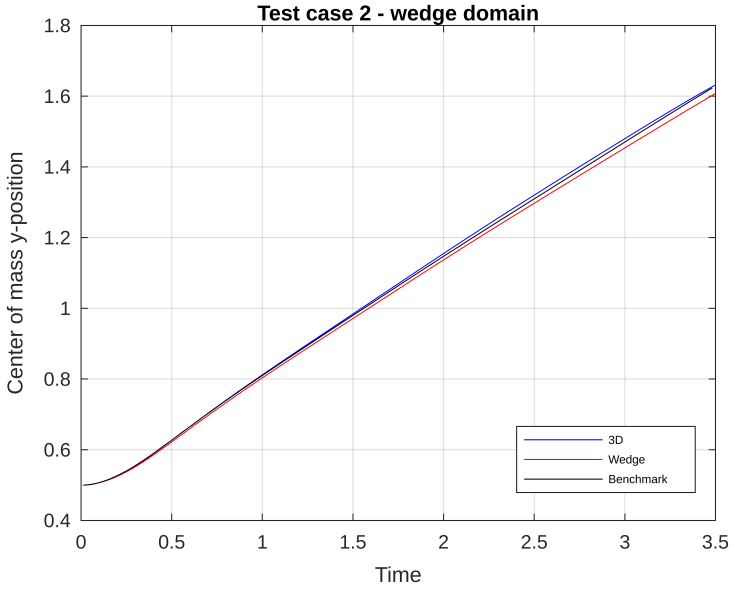



Figure 6.12: Center of mass position evolution. Comparison between uniform grid and wedge case for test 2 (with $h = \frac{1}{128}$). Benchmark values from [23].

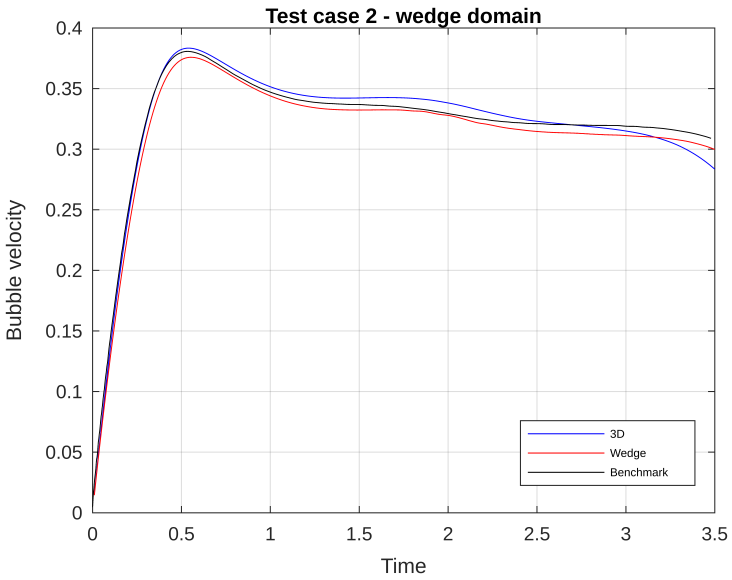



Figure 6.13: Bubble velocity evolution. Comparison between uniform grid and wedge case for test 2 (with $h = \frac{1}{128}$). Benchmark values from [23].

As in the previous test case, the difference between this solution and the one obtained from the 3D simulation may be explained through a different effect of the walls on the bubble. Again, the solution is close to the benchmark and follows the same behaviour, even if the bubble reaches slightly lower velocities.

It is possible to say that a wedge domain simulation may help in studying any axisymmetric 3D problem if a reduction in computational time is needed. In fact, in this case, the solution obtained is very close to the expected one provided by the benchmark by [23].

# 7 | 3D non-Newtonian rising bubble

Finally, given the previous experience with 3D cases, two non-Newtonian simulations have been developed. Starting from the 2D non-Newtonian cases presented in chapter 5, the same bubble has been adapted in a 3D setting.

In particular, a bubble with diameter $d = 4$ mm has been simulated in both a shear-thinning and a shear-thickening in a medium viscosity environment $K = 10\mu_{water}$.

Moreover, given the previous encouraging results from dynamic refinement, the simulations will only be run in dynamic conditions, to save on the expensive computational cost that they would provide in a uniform setting.

## 7.1. Shear-thickening fluid

First, a bubble rising in a shear-thickening fluid is considered. In figure 7.1 it is possible to observe a comparison between the results of 2D and 3D simulations for the bubble in corresponding conditions. In the 3D simulation, there is a noticeable increase in velocity, but the bubble follows a very similar behaviour in both trajectory and shape. This is not unexpected, as the same behaviour is observed when comparing the results of 2D and 3D in the Newtonian case discussed in previous chapters.

The full rise of the bubble is depicted in figure 7.2.

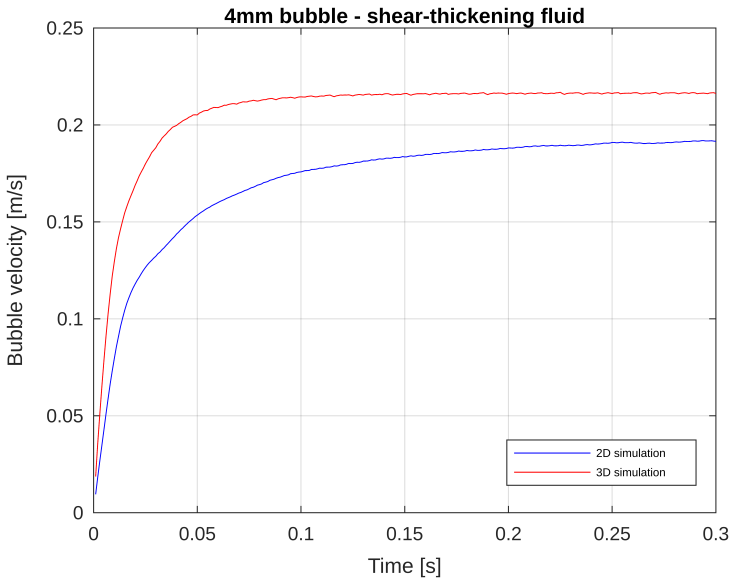



Figure 7.1: Comparison between 2D and 3D bubble in shear-thickening fluid. $d = 4$ mm, $K = 10\mu_{water}$

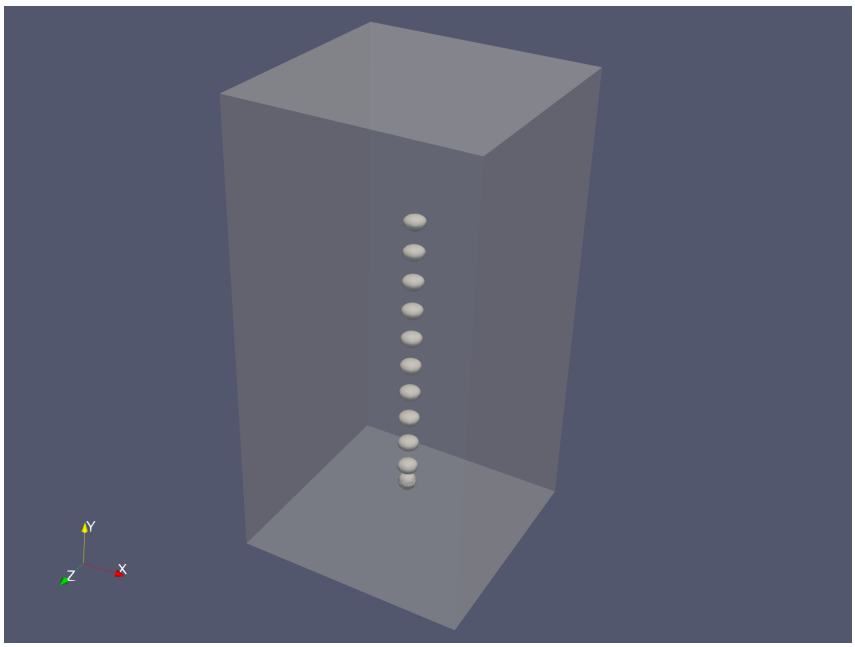


Figure 7.2: Bubble shape and trajectory for $d = 4$ mm, shear-thickening fluid, $K = 10\mu_{water}$. Snapshots taken every 0.3 s.

As noted from previous 2D simulations and [24], the bubble does not present any meaningful oscillations in both trajectory and shape, rising linearly towards the upper side of the domain. Moreover, due to the larger acceleration the bubble has in the 3D case, it reaches its terminal velocity much faster than the 2D one.

## 7.2. Shear-thinning fluid

Finally, a bubble rising in a shear-thinning fluid is considered. In figure 7.3 it is possible to observe a comparison between the results of 2D and 3D simulations for the bubble in corresponding conditions. As in the previous test, a difference in velocity is observed, but the behaviour of the bubble is the same in the two simulations.

In particular, as seen in figure 7.3, the same oscillating behaviour is present in both bubbles. This is again due to the lateral movement of the bubble during its rise observable in the 3D domain, with an impact on the vertical velocity of the bubble.

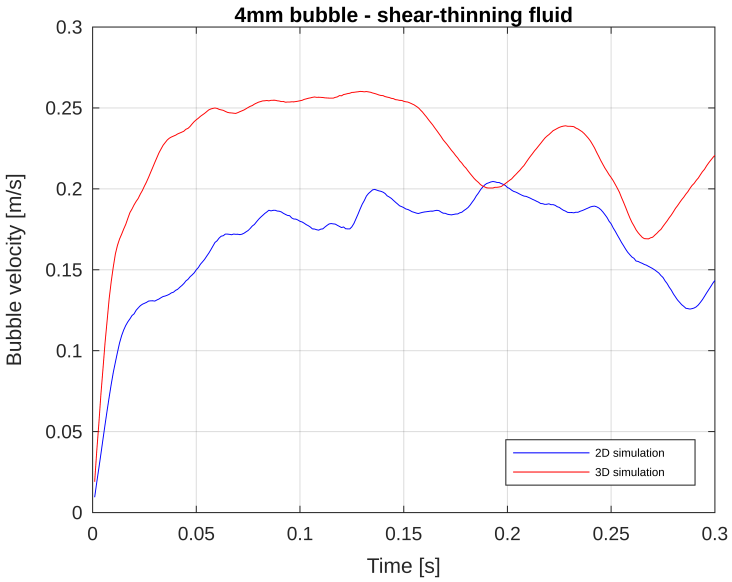



Figure 7.3: Comparison between 2D and 3D bubble in shear-thinning fluid. $d = 4$ mm, $K = 10\mu_{water}$

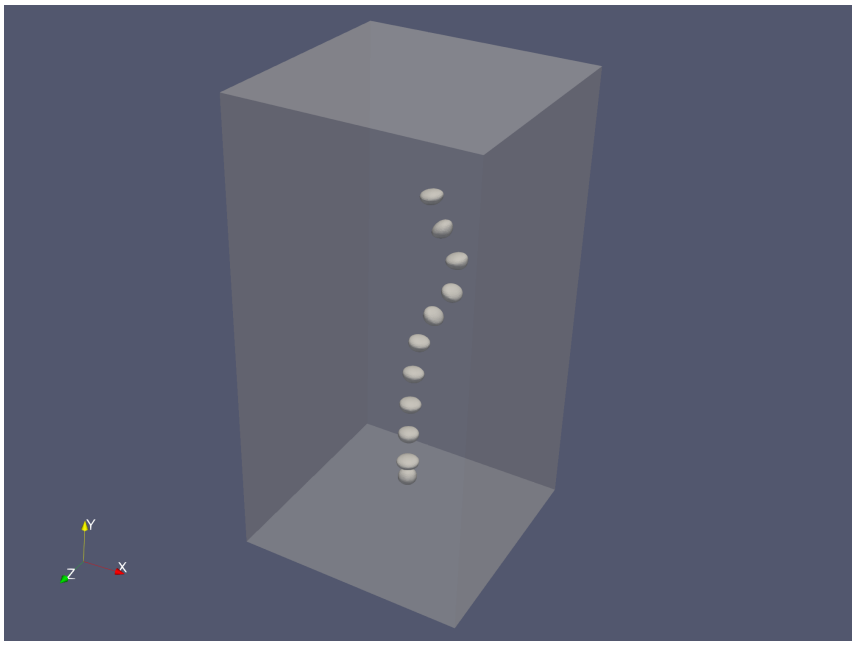



Figure 7.4: Bubble shape and trajectory for $d = 4$ mm, shear-thinning fluid, $K = 10\mu_{water}$. Snapshots taken every 0.3 s.

Looking at the bubble trajectory during its rise, as suggested by the velocity comparison, some oscillations are present. These movements share an evident similarity with the 2D bubble case, as displayed in figure 5.7.

# 8 | Conclusion and future developments

This work presents a set of numerical simulations of flows with multiple fluids involved and inspects them through the use of newly developed cost reduction techniques. As a first step, it analyzes the flow of a bubble rising in a Newtonian fluid in a 2D framework, to set the groundwork for more complex rheologies. During the simulations, it was clear how even simple-looking problems may present challenges, for example, due to the interface between the two fluids breaking during the rise.
In the second problem again we focus on a gas bubble rising in a heavy fluid in a 2D environment. In this case, the proposed surrounding fluids are non-Newtonian fluids, considered in different cases. It is possible to observe different behaviours of the bubble when rising in a shear-thickening or a shear-thinning fluid. Other aspects appear during the simulation, such as oscillations and non-linear behaviour of the gas bubble related to the viscosity or rheology of the surrounding fluid.
Finally, in the third model, a step towards a 3D model is made, considering bubbles rising in both Newtonian and non-Newtonian surrounding fluids, and observing their relation with the 2D previously simulated cases.

In all these cases an effort has been made to reduce the computational time and resources needed for the simulations. While many methods have been exploited, such as adaptive time-step adjustment or moving an axisymmetrical problem from a 3D environment to a 2D one, the main tool was adaptive mesh refinement, a method in which the computational grid gets more refined and precise in the areas of interest and less importance is given to areas not related to the considered phenomenon.
While in the 3D case many software, such as OpenFOAM, already come with a built-in adaptive mesh refinement method, for the 2D case a new one was developed, following the structure of the existing one.
The simulations developed in this work demonstrate that the proposed refinement technique works extremely well in the two-dimensional environment, especially in conditions of a stable solution. Some differences have been observed when studying oscillating or

unstable cases, but the general behaviour of the bubble evolution is conserved.
Therefore there is reason to believe that this proposed method may be useful, if not for obtaining exact results, for gaining a general idea of the solution at a very small fraction of the computational cost.

This work paves the way for several future developments. A first and obvious extension would be to simulate more and varied cases using the proposed two-dimensional adaptive mesh refinement, to check for its validity in a larger array of situations. Other works may focus on improving the computational gains of this method, especially in very complex and large problems, such as employing a method for balancing the grid cells among different processors when using parallelization methods.

# Bibliography

[1] Charles Fefferman. Existence and smoothness of the navier–stokes equation. *American Mathematical Society and Clay Mathematics Institute*, 2006.

[2] Wei-Tao Wu and Mehrdad Massoudi. Recent advances in mechanics of non-newtonian fluids. 2020.

[3] Jonas Karlsson. Implementing anisotropic adaptive mesh refinement in openfoam. 2012.

[4] Ahmad Baniabedalruhman. Dynamic meshing around fluid-fluid interfaces with applications to droplet tracking in contraction geometries. 2015.

[5] Clayton T. Crowe. *Multiphase Handbook*. Taylor & Francis, 2006.

[6] Christopher E. Brennen. *Fundamentals of Multiphase Flows*. Cambridge University Press, 2005.

[7] Baojiang Sun. *Multiphase flow in oil and gas well drilling*. John Wiley & Sons, 2016.

[8] D. Darmana, N. Deen, J. Kuipers, W. Harteveld, and R. Mudde. Numerical study of homogeneous bubbly flow: Influence of the inlet conditions to the hydrodynamic behavior. 2009.

[9] William Smyth. *All Things Flow: Fluid Mechanics for the Natural Sciences*. 2019.

[10] A. Battistella, S.J.G. van Schijndel, M.W. Baltussen, I. Roghair, and M. van Sint Annaland. On the terminal velocity of single bubbles rising in non-newtonian power-law liquids. 2020.

[11] Simscale resources. `https://simscale.com`.

[12] Giovanni Elvio Langella. Single and two-phase flows of shear-thinning fluids: analytical and computational fluid dynamics modelling. 2014.

[13] Jean-Marc Vanden-Broeck. Water waves and related free-surface flows. 2001.

[14] William Finnegan and Jamie Goggins. Numerical simulation of linear water waves and wave–structure interaction. 2012.

[15] Jean Donea, Antonio Huerta, J-Ph. Ponthot1, and A. Rodrıguez-Ferran. *The Encyclopedia of Computational Mechanics*. Wiley, 2004.

[16] J. A. Sethian and Peter Smereka. Level set methods for fluid interfaces. 2003.

[17] Grétar Tryggvason, Ruben Scardovelli, and Stéphane Zaleski. *Direct Numerical Simulations of Gas–Liquid Multiphase Flows*. Cambridge University Press, 2011.

[18] OpenFOAM foundation. OpenFOAM user guide, version 10. 2022.

[19] CFD Direct. Openfoam resources. `https://cfd.direct/openfoam/`.

[20] Hrvoje Jasak. Error analysis and estimation for the finite volume method with applications to fluid flows. 1996.

[21] S. Hysing, S. Turek, N. Parolini, and al. Quantitative benchmark computations of two-dimensional bubble dynamics. 2008.

[22] Roland Clift, John R. Grace, and Martin E. Weber. *Bubbles, Drops, and Particles*. Academic Press: New York, 1978.

[23] J. Adelsberger, P. Esser, M. Griebel, S. Groß, M. Klitz, and A. Rüttgers. 3d incompressible two-phase flow benchmark computations for rising droplets. 2014.

[24] Sadra Mahmoudi, Farshid Hemmatian, Kaveh Padasht Dahkaee, Mark W. Hlawitschka, and Apostolos Kantzas. Detailed study of single bubble behavior and drag correlations in newtonian and non-newtonian liquids for the design of bubble columns. 2022.

[25] Yefeng Zhou, Panxing Kang, Zhengliang Huang, Pan Yan, Jingyuan Sun, Jingdai Wang, and Yongrong Yang. Experimental measurement and theoretical analysis on bubble dynamic behaviors in a gas-liquid bubble column. 2019.

[26] Susana Gabbanelli, German Drazer, and Joel Koplik. Lattice boltzmann method for non-newtonian (power-law) fluids. 2005.

[27] Christopher Greenshields and Henry Weller. *Notes on Computational Fluid Dynamics: General Principles*. CFD Direct Ltd, Reading, UK, 2022.

[28] OpenFOAM. *Documentation*. `https://cpp.openfoam.org/v10/`.

# A | Appendix A

## A.1. AMR2D code

The code developed for this thesis mostly relates to the adaptive mesh refinement, as described in chapter 3.5.

Here the definition of the the static members of the new class `refiner2D` is reported and its integration with the other OpenFOAM namespaces.

```
namespace Foam {
namespace fvMeshTopoChangers {
defineTypeNameAndDebug(refiner2D, 0);
addToRunTimeSelectionTable(fvMeshTopoChanger, refiner2D, fvMesh);
} // namespace fvMeshTopoChangers
} // namespace Foam
```

In the following sections, an insight on the specifics used for the simulations is given.

## A.2. Quantities of interest computation

To obtain the results depicted in the figure throughout the thesis, post-processing has been made at run-time during the simulation. This is possible in OpenFOAM by adding some pre-defined or user-defined functions in the `controlDict` file, the dictionary used for governing the simulation details, such as solvers, simulation time and file management.

Here an extract of the `controlDict` file for computing the quantities described in section 4.1.1 and for obtaining a file describing the bubble position and velocity is reported:

```
functions{
includeFunc isoSurface    // Generates an interpolated surface of all
                                    // points at the same alpha.heavy level.
qoi
{       // Definition of libraries and function name.
        libs        ("libutilityFunctionObjects.so"),("libfieldFunctionObjects.so");
```

```
        type coded;
        executeControl  writeTime;
        writeControl    adjustableRunTime;
        writeInterval   0.001;
        name write_qoi;

        codeWrite
        #{
        // Obtain the quantities needed: time, cell centres, cell volumes,
        // alpha values defining the fluids and velocity field.
        const double time = mesh().time().value();
        const vectorField C = mesh().C();
        const scalarField V = mesh().V();
        const volScalarField alpha =
                    mesh().lookupObject<volScalarField>("alpha.heavy");
        const vectorField U = mesh().lookupObject<vectorField>("U");
        // Integrals of position and velocity on bubble domain across all processors.
        Foam::Vector<double> posSum = gSum((1-alpha)*V*C);
        Foam::Vector<double> velSum = gSum((1-alpha)*V*U);

        // Integral of volume of bubble domain.
        scalar volalpha = 0;
        forAll(alpha,i)
                volalpha += V[i]*(1-alpha[i]);
        reduce(volalpha, sumOp<scalar>());

        // Obtain bubble velocity and center of mass position.
        Foam::Vector<double> velAvg = velSum/volalpha;
        Foam::Vector<double> posAvg = posSum/volalpha;

        // Save time, center of mass position and bubble vertical velocity in a file.
        std::ofstream os;
        os.open("postProcessing/qoi.dat", std::fstream::app);
        if(Pstream::master())
        {
                if(time == 0)
                {
                        os << "Time: " << " \t" << " y" << "\t\t" << " Uy ";
                }
                else
                {
                        os <<"\n" << time << "\t" << posAvg[1] << "\t" << velAvg[1];
                }
        }
        os.close();
```

```
        #};
        codeInclude
        #{
                #include <fstream>
        #};
}
}
```

## A.3. dynamicMeshDict refinement dictionary

Here an example of a dictionary for 2D adaptive mesh refinement is presented:

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  10
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
topoChanger
{
    type            refiner2D;
    libs            ("libfvMeshTopoChangers.so" "libdynamicFvMeshUser.so");
    mover           none;
    // Axis on which refinement should not be applied.
    axis            2;
    // Midpoint of cells in the specified axis direction.
    axisVal         0.05;
    // How often to refine.
    refineInterval  1;
    // Field to be refinement on.
    field           alpha.heavy;
    // Refine field in between lower..upper.
    lowerRefineLevel 0.001;
    upperRefineLevel 0.999;
```

```
32      // Have slower than 2:1 refinement.
33      nBufferLayers   3;
34      // Refine cells only up to maxRefinement levels.
35      maxRefinement   2;
36      // Stop refinement if maxCells reached.
37      maxCells        200000;
38      // Flux field and corresponding velocity field. Fluxes on changed
39      // faces get recalculated by interpolating the velocity. Use 'none'
40      // on surfaceScalarFields that do not need to be reinterpolated.
41      correctFluxes
42      (
43          (phi none)
44          (nHatf none)
45          (rhoPhi none)
46          (alphaPhi.heavy none)
47          (ghf none)
48      );
49      // Write the refinement level as a volScalarField.
50      dumpLevel       true;
51  }
52  // ************************************************************************* //
```

# List of Figures

# List of Tables

# Acknowledgements

This thesis marks the end of a beautiful six-year-long journey, and I would like to thank all the people who provided guidance, help and encouragement throughout it, without whom I would not have succeeded.

Firstly, I would like to express my deepest appreciation to prof. Nicola Parolini, who has been my advisor in this work. He has always assisted me with dedication and willingness to help, and this work couldn't exist without him. I'm extremely grateful to my co-advisor Emilia Capuano, who provided exceptionally valuable advice and guidance. Their constant presence and help have given me this amazing learning and growth opportunity.

A huge thanks to all of my friends who accompanied me along the way. Thanks to Luca, invaluable companion of study, adventures and games during all these years. Thanks to Giulia and Caterina, always helpful in the time of need and ready to bring joy to my heart. Thanks to Federico and Marco, I have always felt at home with you. Thanks to everyone else, who made these years a fantastic adventure.

Finally, an immense thanks to my grandparents, who I could always feel close to me, my mum and dad, who always supported me with trust and appreciation, and all of my family. Thank you for all the love you gave me, during both easy and difficult times. All of you are a great source of inspiration, and will always hold a special place in my heart.

Thank you everyone, from the bottom of my heart.