2015

DYNAMIC MESHING AROUND FLUID-FLUID INTERFACES WITH APPLICATIONS TO DROPLET TRACKING IN CONTRACTION GEOMETRIES

Ahmad Baniabedalruhman
Michigan Technological University

Recommended Citation

Baniabedalruhman, Ahmad, "DYNAMIC MESHING AROUND FLUID-FLUID INTERFACES WITH APPLICATIONS TO DROPLET TRACKING IN CONTRACTION GEOMETRIES", Dissertation, Michigan Technological University, 2015.
https://doi.org/10.37099/mtu.dc.etds/1005

Follow this and additional works at: https://digitalcommons.mtu.edu/etds

 Part of the Applied Mathematics Commons

DYNAMIC MESHING AROUND FLUID-FLUID INTERFACES WITH APPLICATIONS TO DROPLET TRACKING IN CONTRACTION GEOMETRIES

By

Ahmad Baniabedalruhman

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Mathematical Sciences

MICHIGAN TECHNOLOGICAL UNIVERSITY

2015

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Mathematical Sciences.

Department of Mathematical Sciences

Dissertation Advisor: *Prof. Feigl, Kathleen A*

Committee Member: *Prof. Tanner, Franz X*

Committee Member: *Prof. Xu, Zhengfu*

Committee Member: *Prof. Yang, Song L*

Department Chair: *Prof. Gockenbach, Mark S*

Contents

List of Figures

List of Tables

Acknowledgments

I wish to express my gratitude to my advisors, Prof. Kathleen Feigl and Prof. Franz Tanner, for their continuous support, understanding, guidance, and encouragement throughout my research and writing the thesis. Also, I would like to thank the other committee members, Prof. Song Yang and Prof. Zhengfu Xu, for their valuable comments and taking time to serve on my committee.

Furthermore, I would like to acknowledge the collaboration and interesting discussion with my friends and colleagues in the CFD group: Dr. Abdallah Al-Habahbeh, William Case, Chao Liang, Samer Alokaily, and Olabanji Shonibare. Also, the department of mathematical sciences at Michigan technological university is gratefully acknowledged for their financial support.

Finally, I would like to thank my father, mother, brothers, and sisters for their support and encouragement. My special thanks to my wife Ruba for her assistance, patience, motivation, and encouragement. I appreciate her help and comments throughout writing the thesis and I will always remember her.

ABBREVIATIONS

Acronyms

CV	Control Volume
FVM	Finite Volume Method

Greek symbols

$\boldsymbol{T}$	Cauchy stress tensor
τ	viscous tensor
$\dot{\gamma}$	shear rate
η_0	viscosity at zero shear rate
η_∞	viscosity at infinity shear rate

ρ	Density

Roman symbols

$\boldsymbol{g}$	Body force per unit mass
P	Pressure
$\boldsymbol{v}$	Fluid velocity

Abstract

The dynamic meshing procedure in an open source three-dimensional solver for calculating immiscible two-phase flow is modified to allow for simulations in two-dimensional planar and axisymmetric geometries. Specifically, the dynamic mesh refinement procedure, which functions only for the partitioning of three-dimensional hexahedral cells, is modified for the partitioning of cells in two-dimensional planar and axisymmetric flow simulations. Moreover, the procedure is modified to allow for computing the deformation and breakup of drops or bubbles that are very small relative to the mesh of the flow domain. This is necessary to avoid mass loss when tracking small drops or bubbles through flow fields. Three test cases are used to validate the modifications: the deformation and breakup of a two-dimensional drop in a linear shear field; the formation and detachment of drops in a two-dimensional micro T-junction channel; and an axisymmetric bubble rising from a pore into a static liquid. The tests show that the modified code performs very well, giving accurate results for much less computational time when compared to corresponding simulations without dynamic meshing.

The modified code is then applied to study drop breakup conditions inside a spray nozzle when an emulsion is sprayed to produce a powder. This is done by tracking droplets of various sizes through the flow field within the nozzle and determining conditions under which they break up. The particular interest is in determining the largest drop sizes for which breakup does not occur. The effects of viscosity ratio, capillary number, shear rate,

and fluid rheology on the critical drop sizes are determined.

Although the code modifications performed for this research were implemented for dynamic mesh refinement of cells close to fluid-fluid interfaces, they may be adapted to other regions in the domain and for other types of flow problems.

Chapter 1

Introduction

Two-phase flow is a flow of a fluid system composed of two different kinds of matter, e.g., solid particles in a gas or liquid, gas bubbles in a liquid or liquid droplets in a gas stream or another immiscible liquid. An interface is a surface separating the two phases of the fluid system. The study of two-phase flows is very important because of their widespread applications in industry. Their applications includes lubrication, spray processes, fluid-particle transport, food stuff processing (emulsions, foams), nuclear reactor cooling and material manufacturing.

An emulsion is a mixture of two immiscible liquids of which one is dispersed in another. The dispersed phase can be either droplets of a single fluid, in which case the fluid system is called a simple emulsion, or the dispersed phase can itself be an emulsion, in which case the fluid system is called double emulsion or multiple emulsion. The most

common types of emulsions are water-in-oil (w-o) and oil-in-water (o-w). Hydrophilic or lipophilic surfactants are encapsulated to produce stable emulsions by reducing the interfacial tension between the phases. Emulsions are inherently unstable due in part to coalescence and compositional ripening [1], [2], [3]. Coalescence is the process by which droplets merge with each other to form larger droplets, whereas compositional ripening occurs by diffusion and/or permeation of the surfactants components across the disperse phase. Hence their stability phenomena and the production of stable emulsions are studied by many researchers [4, 5, 6, 7, 8]. Producing powders from emulsions reduces the problem of stability and increases the shelf life. Many researchers have worked on spraying of emulsions where they mostly studied simple emulsions rather than multiple emulsions [9, 10, 11, 12]. Producing powders by spraying multiple emulsions is more complex in terms of preserving its structure [13]. Many studies have looked at the effect of the spraying process and the viscosity ratio on the spray drop size [14, 15, 16, 17, 18, 19]. A uniform drop size distribution is desirable with drop radii on the order of microns.

The processing of emulsions can be studied computationally by numerically solving a two-phase flow problem in which the location of the fluid-fluid interface must be computed along with the flow variables, such as velocity and pressure. There are different numerical approaches to solve two-phase problems. Two popular approaches are interface tracking and interface capturing methods. In the interface tracking method, a mesh to track the interface is needed and mesh points lie on the interface. In this method, the interface is explicitly described by the computational mesh and the mesh is updated if the interface is

moved so that mesh points remain on the interface [20]. On the other side, the interface capturing methods implicitly describe the interface by an artificial function where the mesh is fixed. The most popular interface capturing methods are the level set [21, 22] and volume-of-fluid [23, 24] methods. In the level set method, the signed distance function ϕ is used to describe the interface, where ϕ is zero at the interface, positive in the dispersed phase, and negative in the continuous phase. In addition, the level set function is smooth, allowing for an accurate calculation for the curvature κ. The volume-of-fluid method uses a discontinuous volume fraction function α instead of the level set function. The volume fraction function α is one in the dispersed phase and zero in the continuous phase. Once a mesh is introduced, the value of α in a cell is the volume fraction of the dispersed phase in the cell. Therefore, $0 < \alpha < 1$ in cells that contains the interface. The volume-of-fluid approach has much better mass conserving properties than the level set approach, but a major challenge is accurately calculating the curvature κ. In order to obtain accurate two-phase flow calculations, a sufficiently refined mesh around the interface is required. Instead of refining the mesh throughout the whole domain, dynamic mesh refinement can be used.

Dynamic mesh refinement allows an accurate solution with low costs by having high mesh resolution in specific regions, for example, around the interface in two-phase flow problems. It reduces the costs in terms of computational time and storage compared to a refined uniform mesh. Dynamic mesh refinement was studied on structured grids by Berger et al. [25]. In general, there are two methods for the adaptive mesh refinement. The first is

r-refinement in which the number of grid points and cells are fixed, and the grid points are redistributed on the mesh to produce high resolution (i.e. small cells) in particular places. The second method is h-refinement in which new points and cells are added to the mesh in order to have sufficient resolution in desired regions. The h-refinement is achieved by dividing a set of cells into smaller cells. Many scholars have worked on the dynamic mesh refinement, including Mavriplis [26, 27] who formulated an adaptive mesh refinement for an unstructured mesh; Pizadeh [28] who developed an unstructured grid adaption using different adaptive techniques; and Anderson [29] who developed an algorithm to solve Euler equations by combining the staggered grid arbitrary Lagrangian-Eulerian techniques with structured local adaptive mesh refinement. In addition, Coirier [30] developed an adaptively-refined, Cartesian, cell-based scheme for the Euler and Navier-Stokes equations, while Hunt [31] developed a code to solve three dimensional equations using adaptive refinement; and Qingluan [32] developed an adaptive mesh refinement algorithm for engine spray simulations where the refinement is required in the spray region. Also, the `interDyMFoam` solver for two-phase flow in OpenFOAM® uses a dynamic mesh refinement for three dimensional hexahedral meshes.

OpenFOAM® stands for Open Field Operation And Manipulation. It is an open source code using C++ libraries and serves as a modeling and computational fluid dynamic (CFD) platform for the research community. The mesh generation, equations discretization, and matrix manipulations can be accomplished using applications of source codes and libraries in OpenFOAM® [33]. The software contains solvers for many computational fluid

dynamics problems ranging from fluid flow including heat transfer, chemical reactions, and turbulence to solid dynamics and electromagnetic. New solvers or modifications to existing solvers or libraries can be constructed by the user to meet the needs of his/her specific application.

OpenFOAM® has the ability to study multi-phase flows, specifically through the `interFoam` and `interDyMFoam` solvers. In this thesis, a modification of the `interDyMFoam` solver is achieved for application to 2D planar and axisymmetric flows. Two cases are studied to validate the modification for the 2D planar simulations, specifically (1) a droplet in a planar linear shear flow, and (2) droplets detaching from a pore into a shear flow field. Furthermore, a bubble rising in water is used to validate the axisymmetric simulations.

The modified `interDyMFoam` is then used to study breakup conditions of drops inside a nozzle when an emulsion is sprayed to produce a powder. Dynamic meshing around the interface is necessary in this application since the drop sizes can be very small relative to the geometry. Moreover, due to the large number of drops that need to be tracked, the simulations are performed in two dimensions. The effect of capillary number, viscosity ratio, shear rate, and fluid rheology on the critical break up radius is studied.

Contributions of this thesis

This dissertation makes several contributions to the field of Computational Fluid Dynamics and the understanding of drop breakup conditions in complex geometries. The major contributions are:

1. The dynamic meshing capabilities of a popular open source CFD software package used in the research community has been improved.

2. The dynamic meshing in a two-phase flow solver has been modified to allow for dynamic meshing around fluid-fluid interfaces in two-dimensional planar and axisymmetric geometries. The modification to these geometries serves to reduce computational time and allows for application to problems in which many small drops or bubbles must be tracked.

3. The modified dynamic meshing code was applied to three test problems:

   (a) Drop deformation and breakup in linear shear flow (two-dimensional planar).

   (b) Drop formation and detachment from micro T-channels (two-dimensional planar).

   (c) Bubble rising from a pore into static liquid (axisymmetric).

   The performance of the modified code was evaluated on these three test problems in terms of computational time, mesh independence, and mass conservation. Comparisons were made with the two-phase flow solver without dynamic meshing.

4. The effect of the dynamic meshing parameters on the results was determined for the micro T-channel test problem.

5. The dynamic meshing code was further modified to improve the refinement around droplets as they move in a flow field. This was necessary to prevent mass loss when

tracking droplets that are very small relative to the flow domain length scale and corresponding mesh.

6. The modified dynamic meshing code was used to track droplets in a contraction geometry representing a spraying nozzle. From this, in-nozzle drop breakup conditions were investigated when spraying an emulsion. Of particular interest to spray engineers are critical drop sizes, that is, the largest drops that will not break up within the nozzle.

7. The effect of shear rate, rheology of the continuous phase fluid, nozzle length, capillary number, and viscosity ratio (i.e., drop viscosity relative to continuous phase viscosity) on critical drop sizes was determined.

8. Grace curves for this geometry, which give the critical capillary number as a function of viscosity ratio, were determined. Results of the above parameter study were interpreted in terms of these Grace curves.

9. The modified dynamic meshing procedure may be used for other two-dimensional and axisymmetric problems solved with OpenFOAM®.

Chapter 2

Mathematical Model

Fluid dynamics is the science which studies the motion of liquids and gases and how they interact with the environment. It has applications in many fields and its uses include calculating forces and moments on aircraft, determining the mass flow rate of petroleum through pipelines and predicting weather patterns. The solution of a fluid dynamics problem involves calculating various properties of the fluid, such as velocity, density, pressure and temperature, as a function of space and time. Continuum mechanics treats the material as a continuous mass instead of discrete particles. The basic conservation laws of continuum mechanics are the conservation of mass, conservation of momentum and conservation of energy. From these conservation laws we can derive the differential equations that describe the properties of the fluid and flow.

Two-phase flow is best described as the flow of a fluid system composed of two different

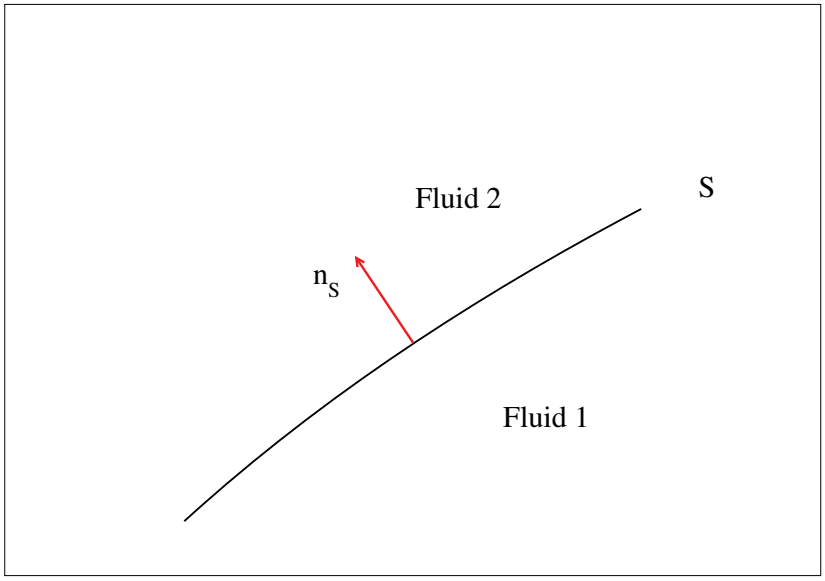

Figure 2.1: Schematic diagram of a two-phase flow

kinds of matter, e.g., solid particles in a gas or liquid, gas bubbles in a liquid, or liquid droplets in a gas stream or another immiscible liquid. An interface is a surface separating the two phases of the fluid system. Two-phase flow has many industrial applications such as lubrication, spray processes, fluid-particle transport, food stuff processing (emulsions, foams), nuclear reactor cooling and material manufacturing.

2.1 Governing Equations

Two-phase flow is a flow of two fluids separated from each other by interface S as shown in Figure 2.1, where $\boldsymbol{n}_S$ is the unit normal vector on the interface S directed to fluid 2. In this thesis, we are primarily interested in the two-phase flow where both fluids are liquids.

From the conservation of mass and momentum principles, the differential form of the continuity equation is

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = 0, \tag{2.1}$$

and the momentum equation is

$$\frac{\partial (\rho \boldsymbol{v})}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}\boldsymbol{v}) = \nabla \cdot \boldsymbol{T} + \rho \boldsymbol{g} + \boldsymbol{f} \tag{2.2}$$

where ρ is the density, $\boldsymbol{v}$ is the velocity, $\boldsymbol{T}$ is the Cauchy stress tensor, $\boldsymbol{g}$ is the gravity and $\boldsymbol{f}$ is a force per unit volume. The stress tensor $\boldsymbol{T}$ can be expressed as:

$$\boldsymbol{T} = -PI + \boldsymbol{\tau} \tag{2.3}$$

where P is the pressure and $\boldsymbol{\tau}$ is the viscous stress tensor. Using Eq. (2.3), the momentum equation becomes

$$\frac{\partial (\rho \boldsymbol{v})}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}\boldsymbol{v}) = -\nabla P + \nabla \cdot \boldsymbol{\tau} + \rho \boldsymbol{g} + \boldsymbol{f}. \tag{2.4}$$

If changes in ρ are negligible, as in the case of incompressible flow, the equations become

$$\nabla \cdot \boldsymbol{v} = 0, \tag{2.5}$$

$$\rho[\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v}\cdot\nabla)\boldsymbol{v}] = -\nabla P + \nabla\cdot\boldsymbol{\tau} + \rho\boldsymbol{g} + \boldsymbol{f}. \tag{2.6}$$

An equation is also needed to describe the evolution of the interface. This equation depends on the numerical approach that is used to solve the two-phase flow problem, and will be described later. Interface conditions must also be specified to describe the behavior at the fluid-fluid interface. There are two conditions on the interface, given below, where $[m]_S$ denote a jump across the interface S of a function m, i.e., $[m]_s = m_1 - m_2$, where the subscripts denote the fluids.

1. Continuous velocity: $[\boldsymbol{v}]_S = \boldsymbol{0}$ (there is no jump in the velocity across the interface).

2. Jump in surface traction: $[\boldsymbol{T}\cdot\boldsymbol{n}]_S = \Delta\boldsymbol{f}$, where $\Delta\boldsymbol{f}$ represents a surface force due to interfacial tension. A constitutive equation is needed for $\Delta\boldsymbol{f}$. A common one is $\Delta\boldsymbol{f} = \sigma\kappa\boldsymbol{n}_S$, where $\kappa = \nabla\cdot\boldsymbol{n}_S$ is the local mean curvature, $\boldsymbol{n}_S$ is the unit normal vector on the interface, and σ is the interfacial tension. This is a generalization of the Young-Laplace equation [34] that gives the capillary pressure difference across the interface S between two static fluids.

There are several techniques for calculating the surface tension force. Some of those methods are the Continuum Surface Stress method (CSS) [35, 36], ghost fluid method (GFM) [37, 38], Meier's method [39, 40] and The Continuum Surface Force method (CSF) [41]. In the CSF method, these conditions, in particular, the jump in $\boldsymbol{T}\cdot\boldsymbol{n}$ is accounted for

in the momentum equation as

$$\boldsymbol{f}_S = \sigma \kappa \boldsymbol{n}_S \delta(\boldsymbol{x} - \boldsymbol{x}_S) \tag{2.7}$$

where $\boldsymbol{f}_S$ is the volumetric surface tension force, $\kappa = \nabla \cdot \boldsymbol{n}_S$ is the local mean curvature, $\boldsymbol{n}_S$ is the unit normal vector on the interface, $\delta(\boldsymbol{x} - \boldsymbol{x}_s)$ is the Dirac delta function, and σ is the interfacial tension. The calculation of the normal vector and curvature is discussed in Section 2.2.

2.2 Equations to Describe Interface

There are two main approaches for describing the evolution of the interface: Interface tracking methods and interface capturing methods. In interface tracking methods, the moving interface is explicitly described by the nodes of the computational mesh. The mesh must be adjusted so that the nodes lie on the interface. In interface capturing methods, the location of the moving interface is implicitly described by a scalar function. These methods are Eulerian in which the mesh is stationary or moving in a given manner. The most popular interface capturing methods are the level set method (LSM), volume of fluid method (VOF), and coupled level set-volume-of-fluid (CLSVOF) method. The basic LSM and VOF method are described below.

2.2.1 Level Set Method (LSM)

The level set method captures the motion of an interface by embedding the interface as the zero level set of the level set function φ [22]. The level set function is defined as:

$\varphi(\boldsymbol{x},t) < 0$ if $\boldsymbol{x}$ is in fluid 1,

$\varphi(\boldsymbol{x},t) = 0$ if $\boldsymbol{x}$ is on the interface and

$\varphi(\boldsymbol{x},t) > 0$ if $\boldsymbol{x}$ is in fluid 2

The level set equation is

$$\frac{\partial \varphi}{\partial t} + \boldsymbol{v} \cdot \nabla \varphi = 0. \tag{2.8}$$

Physically, this equation means that the value φ does not change with time along a particle path since the left hand side of this equation is the material derivative. That is, the interface is convected with the flow fluid. For example, on the interface the value of φ will be the same at each time. Initially, $\varphi(\boldsymbol{x},0)$ is the signed distance function to the interface. The momentum equation can be written as:

$$\rho(\varphi)(\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v} \cdot \nabla \boldsymbol{v}) = -\nabla P + \nabla \cdot \boldsymbol{\tau} + \rho(\varphi)\boldsymbol{g} + \sigma\kappa(\varphi)\boldsymbol{n}_S(\varphi)\delta(\varphi) \tag{2.9}$$

where

$\rho(\varphi) = \rho_2 + (\rho_1 - \rho_2)H(\varphi)$

$\mu(\varphi) = \mu_2 + (\mu_1 - \mu_2)H(\varphi)$

$\kappa = \nabla \cdot \mathbf{n}_S$ and

$\mathbf{n}_S = \frac{\nabla\varphi}{|\nabla\varphi|}$

where H is the Heaviside function and the subscript in ρ and μ indicate the fluid phase. Also, μ is the dynamic viscosity, which is used in the constitutive equation for τ as discussed in Section 2.3.

2.2.2 Volume of Fluid Method (VOF)

The volume-of-fluid method uses a volume fraction function α, instead of level set function, to describe the location of the interface [23]. The volume fraction function α is a discontinuous function such that $\alpha = 0$ in the continuous fluid (fluid 2) and $\alpha = 1$ in the dispersed fluid (fluid 1). On a computational mesh, this discontinuity is smoothed by letting

$\alpha = 0$ in cells that contain only the fluid 2,

$\alpha = 1$ in cells that contain only the fluid 1 and

$0 < \alpha < 1$ in cells where the interface passes such that $\alpha = \frac{V_{Fluid1}}{V}$, where V_{Fluid1} is the volume of fluid 1 in the cell and V is the volume of the cell.

The volume fraction equation is

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha \boldsymbol{v}) = 0. \tag{2.10}$$

As in the level set method, we can write the surface force term as: $\boldsymbol{f}_S = \sigma\kappa\delta(\boldsymbol{x} - \boldsymbol{x}_S)\boldsymbol{n}_S$. In VOF, the CSF $\boldsymbol{f}_S = \sigma\kappa\delta(\boldsymbol{x} - \boldsymbol{x}_S)\boldsymbol{n}_S$ is reformulated as $\boldsymbol{f}_S = \sigma\kappa\nabla\alpha$ [23]. The momentum equation can be written as:

$$\rho(\alpha)(\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v} \cdot \nabla \boldsymbol{v}) = -\nabla P + \nabla \cdot \boldsymbol{\tau} + \rho(\alpha)\boldsymbol{g} + \sigma\kappa(\alpha)\nabla\alpha \tag{2.11}$$

where

$\rho(\alpha) = \alpha\rho_1 + (1-\alpha)\rho_2$,

$\mu(\alpha) = \alpha\mu_1 + (1-\alpha)\mu_2$, and

$\kappa = -\nabla \cdot \frac{\nabla\alpha}{|\nabla\alpha|}$ and

where the subscript in ρ and μ indicate the fluid phase.

The level set function is continuous, making the calculation of the unit normal and curvature accurate, however the method does not guarantee the conservation of mass. On the other hand, the volume-of-fluid is mass conserving but the volume fraction function is discontinuous, making it less effective in calculating the unit normal and curvature as surface tension force increases. In this thesis, we use the VOF method as implemented in `interFoam`.

2.3 Rheology

To close the system of the governing equations in Section 2.1, an expression for the stress tensor $\boldsymbol{\tau}$ must be specified. This stress depends on the deformation and strain rate experienced by the fluid.

Rheology is the science that studies the flow and deformation of materials. It describes the relationship between stress and deformation (strain). The mathematical form of this relationship is called the constitutive equation.

The common rheology terms are stress $\boldsymbol{\tau}$ which is the force acting on an area divided by that area, strain rate $\dot{\boldsymbol{\gamma}}$ which is the rate of change in shape of a deformed material with respect to time and mathematically defined as the rate-of-strain tensor $\dot{\boldsymbol{\gamma}} = \nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T$ where $\boldsymbol{v}$ is the velocity, shear rate $\dot{\gamma}$ which is the magnitude of the rate-of-strain tensor, and viscosity which is the quantity that describes a fluid's resistance to flow.

A Newtonian fluid is a fluid in which the stress is linear in the rate-of-strain tensor. The constitutive equation for an incompressible Newtonian fluid is given by

$$\boldsymbol{\tau} = \mu \dot{\boldsymbol{\gamma}} = \mu [\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T], \tag{2.12}$$

where μ is the constant dynamic viscosity.

A non-Newtonian fluid is a fluid whose stress is not linear in $\dot{\boldsymbol{\gamma}}$. There are two types of non-Newtonian fluids: time-dependent fluids in which the relation between stress and

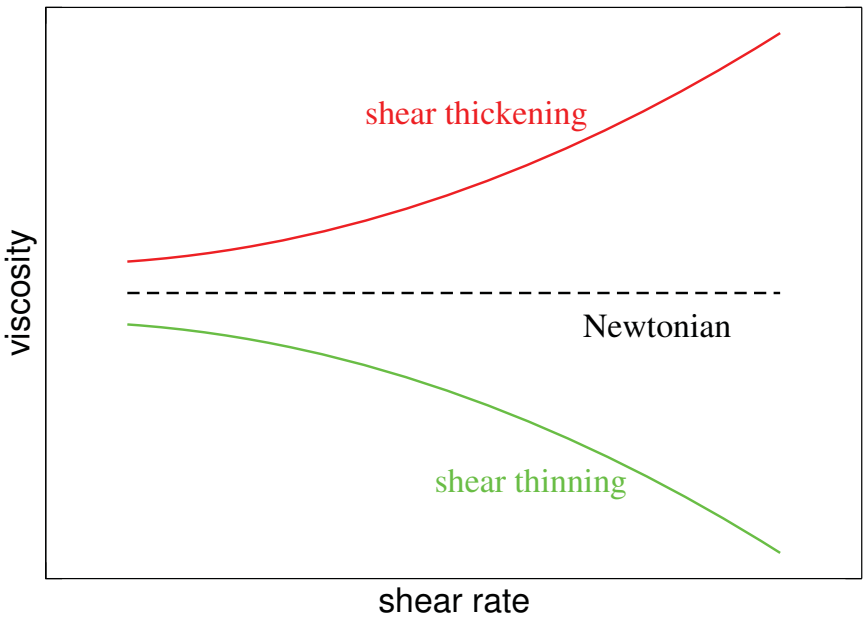

Figure 2.2: Viscosity vs shear rate for different fluids.

strain depends on how long the shear has been active, and time-independent fluids in which the relation does not depend on the time. The time-independent non-Newtonian fluids are the most popular and in this thesis only this type of non-Newtonian fluid is considered. This type of fluid is classified into shear-thinning or pseudo-plastic fluids in which the viscosity decreases when the shear rate increases, shear-thickening or dilatant fluids in which the viscosity increases when the shear rate increases, and yield stress fluids in which a minimum stress is required before the material will flow. Figure 2.2 shows a graph of viscosity and shear rate for shear thinning, shear thickening and Newtonian fluids.

Generalized Newtonian fluid models assume a simple constitutive equation like the one for the Newtonian fluid but here the viscosity is a function of the shear rate. The general

form of the constitutive equation for the generalized Newtonian fluid models is

$$\tau = \eta(\dot{\gamma})\dot{\gamma} \tag{2.13}$$

where $\eta(\dot{\gamma})$ is the viscosity function. The resulting values of the viscosity at very low and high shear rates are known as the zero-shear-rate viscosity η_0 and the infinite-shear-rate viscosity η_∞ respectively. Two popular models for generalized Newtonian fluid will be described here.

2.3.1 Power Law Model

The model describes a power-law relation between the viscosity η and shear rate $\dot{\gamma}$, and is given by:

$$\eta = K\dot{\gamma}^{n-1} \tag{2.14}$$

where K is the consistency coefficient (units of $Pa \cdot s^n$) which reflects the vertical shift in the viscosity curve on a log-log plot, and the dimensionless n is the power-law index such that $n-1$ represents the slope of the viscosity curve on a log-log plot and reflects how close the fluid is to Newtonian. For a Newtonian fluid, $n = 1$ and the consistency index K is equal to the viscosity of the fluid. If $n < 1$, then the fluid is shear-thinning and if $n > 1$, then the fluid is shear-thickening. This model is popular because most fluids have a linear relation

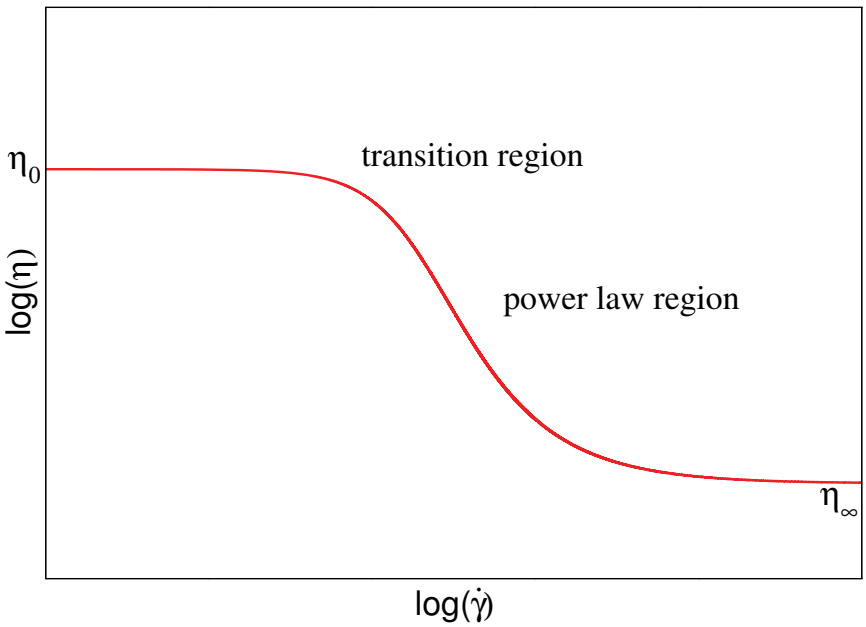

Figure 2.3: Viscosity vs shear rate for a Carreau-Yasuda model.

in some region of the log-log graph but it does not describe the zero- and infinite-shear-rate viscosities.

2.3.2 Carreau-Yasuda Model

A model which does describe the upper and lower shear rate regions is the Carreau-Yasuda model. The relationship between viscosity and shear rate is given by:

$$\frac{\eta - \eta_\infty}{\eta_0 - \eta_\infty} = [1 + (m\dot{\gamma})^a]^{\frac{n-1}{a}} \tag{2.15}$$

The parameter m is a constant with units of time, where $\frac{1}{m}$ is the critical shear rate at which viscosity begins to decrease or increase, a is a dimensionless constant which affects

the shape of the transition region (e.g., increasing a sharpens the transition), and n is a dimensionless constant which describes the slope in the power law region as show in Figure 2.3. The Bird-Carreau model is given by $a = 2$ in Eq. (2.15), therefore the equation becomes

$$\frac{\eta - \eta_\infty}{\eta_0 - \eta_\infty} = [1 + (m\dot{\gamma})^2]^{\frac{n-1}{2}}. \tag{2.16}$$

2.4 Numerical Methods

The governing system of equations for two-phase flow involving Newtonian and non-Newtonian fluids is solved using the finite volume method (FVM) described in this section. Since the momentum equation is a time-dependent convection-diffusion equation, we first describe the finite volume method for a general time-dependent convection-diffusion equation, we then discuss its application to incompressible flow problems.

2.4.1 Finite Volume Method Discretization

The finite volume method (FVM) is a popular method to solve numerically the governing equations of fluid dynamics. In FVM, the computational domain is divided into a finite

number of control volumes (CVs) and the governing equations are integrated on the CV to get the integral form for the equations. The description of the basic FVM below follows in part that given in the thesis of Jasak [42].

The boundary of a CV contains a number of faces and, it is assumed here that each face in the domain share at most two CVs. Figure 2.4 shows a CV, where V_P is the volume of the CV, P is a computational point at the centroid of the CV, f is a computational point at the center of a face, S_f is the area of the face f, $\boldsymbol{n}_f$ is the face outward unit normal vector, N is a computational point of a neighboring CV, $\boldsymbol{d}_f$ is the vector between P and N and $\boldsymbol{r}_P$ is the vector between the origin and P.

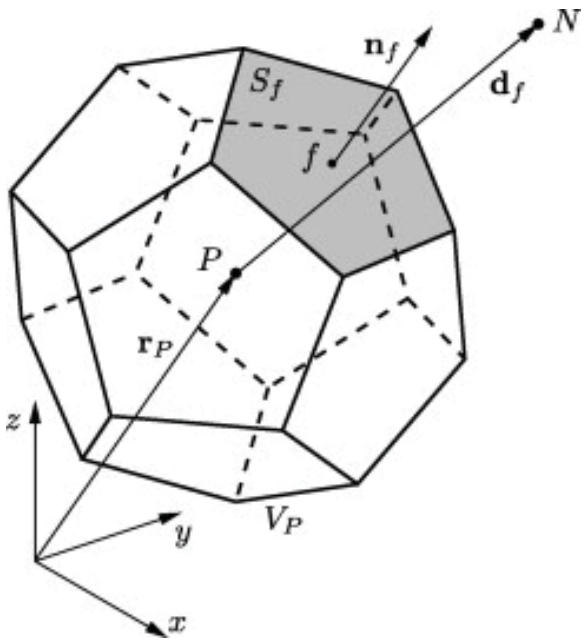

Figure 2.4: Arbitrary control volume.

The coordinates of the centroid of the CV, $\boldsymbol{x}_P$, and the face, $\boldsymbol{x}_f$, are given by:

$$\int_{V_P} (\boldsymbol{x} - \boldsymbol{x}_P) dV = \mathbf{0}, \tag{2.17}$$

$$\int_{f} (\boldsymbol{x} - \boldsymbol{x}_f) dS = \mathbf{0}. \tag{2.18}$$

The general convection-diffusion equation in fluid dynamics has the form:

$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{\text{temporal derivative}} + \underbrace{\nabla\cdot(\rho\boldsymbol{v}\phi)}_{\text{convective term}} = \underbrace{\nabla\cdot(\rho\Gamma_\phi\nabla\phi)}_{\text{diffusion term}} + \underbrace{q_\phi(\phi)}_{\text{source term}} \tag{2.19}$$

where ϕ is a general property and Γ_ϕ is the diffusion coefficient. The key step of FVM is the integration of Eq. (2.19) over a CV yielding

$$\int_{V_P} \frac{\partial(\rho\phi)}{\partial t} dV + \int_{V_P} \nabla\cdot(\rho\boldsymbol{v}\phi) dV = \int_{V_P} \nabla\cdot(\rho\Gamma_\phi\nabla\phi) dV + \int_{V_P} q_\phi(\phi) dV, \tag{2.20}$$

By applying the Gauss divergence theorem, Eq. (2.20) can be written as follows:

$$\int_{V_P} \frac{\partial(\rho\phi)}{\partial t} dV + \oint_{\partial V_P} \boldsymbol{n}\cdot(\rho\boldsymbol{v}\phi) dS = \oint_{\partial V_P} \boldsymbol{n}\cdot(\rho\Gamma_\phi\nabla\phi) dS + \int_{V_P} q_\phi(\phi) dV, \tag{2.21}$$

where $\boldsymbol{n}$ is the outward-pointing unit normal vector. By integrating Eq. (2.21) with respect to time t over a small interval, we get the most general form

$$\int_t^{t+\delta t}\left[\int_{V_P}\frac{\partial(\rho\phi)}{\partial t}dV+\oint_{\partial V_P}\boldsymbol{n}\cdot(\rho\boldsymbol{v}\phi)dS-\oint_{\partial V_P}\boldsymbol{n}\cdot(\rho\Gamma_\phi\nabla\phi)dS\right]dt = \int_t^{t+\delta t}\int_{V_P}q_\phi(\phi)dVdt. \tag{2.22}$$

In the FVM discretization, the linear variation is used to approximate the function ϕ around the point P. The approximation is a second-order accurate and it is given by:

$$\phi(\boldsymbol{x}) \quad = \quad \phi_P+(\boldsymbol{x}-\boldsymbol{x}_P)\cdot(\nabla\phi)_P, \tag{2.23}$$

where $\phi_P=\phi(\boldsymbol{x}_P)$. In the below sections some of the discretization methods are described.

2.4.1.1 Discretization of Convection Term

Since each CV is bounded by a number of faces, then

$$\oint_{\partial V_P} (\rho \boldsymbol{v} \phi \cdot \boldsymbol{n}) dS = \sum_f \left(\int_f (\rho \boldsymbol{v} \phi \cdot \boldsymbol{n}_f) dS \right). \tag{2.24}$$

By using the assumption of linear variation for ϕ around the point f, the term $\rho \boldsymbol{v} \phi$ is written as:

$$\rho \boldsymbol{v} \phi(\boldsymbol{x}) = (\rho \boldsymbol{v} \phi)_f + (\boldsymbol{x} - \boldsymbol{x}_f) \cdot (\nabla(\rho \boldsymbol{v} \phi))_f \tag{2.25}$$

Therefore, the integral inside the sum above is approximated as following:

$$\int_f (\rho \boldsymbol{v} \phi \cdot \boldsymbol{n}_f) dS = (\rho \boldsymbol{v} \phi)_f \cdot \int_f \boldsymbol{n}_f dS + (\nabla(\rho \boldsymbol{v} \phi))_f : \int_f (\boldsymbol{x} - \boldsymbol{x}_f) \boldsymbol{n}_f dS, \tag{2.26}$$

where $\boldsymbol{g}_f$ stands for the value of $\boldsymbol{g}$ at the center of the face f. Assuming $\boldsymbol{n}_f$ is constant on face f (i.e., that face is a plane surface) and using Eq. (2.18), Eq. (2.26) becomes

$$\int_f (\rho \boldsymbol{v} \phi \cdot \boldsymbol{n}) dS = (\rho \boldsymbol{v} \phi)_f \cdot \int_f \boldsymbol{n}_f dS = (\rho \boldsymbol{v} \phi)_f \cdot \boldsymbol{S}, \tag{2.27}$$

where $\boldsymbol{S} = S_f \boldsymbol{n}_f$ is the outward area vector of a face and S_f is the face area. The right-hand side of Eq. (2.24) can be approximated using Eq. (2.27), so that Eq. (2.24) becomes

$$\begin{aligned} \oint_{\partial V_P} (\rho \boldsymbol{v} \phi \cdot \boldsymbol{n}) dS &= \sum_f (\rho \boldsymbol{v} \phi)_f \cdot \boldsymbol{S} \\ &= \sum_f \boldsymbol{S} \cdot (\rho \boldsymbol{v})_f \phi_f \\ &= \sum_f F \phi_f, \end{aligned} \tag{2.28}$$

where

$$F = \boldsymbol{S} \cdot (\rho \boldsymbol{v})_f \tag{2.29}$$

is the convective mass flux through the face f. To find F, the values of ρ and $\boldsymbol{v}$ have to be found at the faces by interpolating from the values at the centroids. A weighted average is used to estimate ϕ at the face as in the equation below:

$$\phi_f = b_f \phi_P + (1 - b_f) \phi_N \tag{2.30}$$

as illustrated in Figure 2.5. Different values for b_f gives different methods. Three basic methods are presented below.

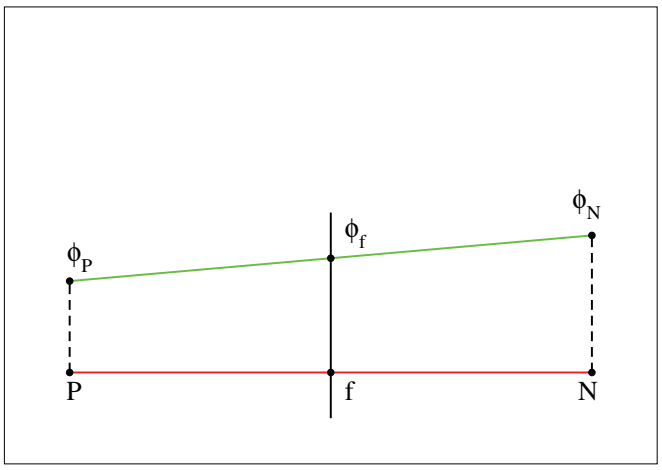

Figure 2.5: Face interpolation.

1. **Central Differencing CD**

   In CD, b_f in Eq. (2.30) is defined as:

$$b_f = \frac{\overline{fN}}{\overline{PN}} \tag{2.31}$$

   where $\overline{fN}$ is the distance between the face and the computational point N and $\overline{PN}$ is the distance between the computational points P and N as shown shown in Figure 2.5. If the mesh is uniform then $b_f = \frac{1}{2}$. The method is second-order but unphysical oscillations appear in the solution for convection-dominated problems, which often makes the solution unbounded. More details are found in Chapter 14 of [43] and Chapter 4 of [44].

2. **Upwind Differencing UD**

In UD, b_f in Eq. (2.30) is defined as:

$$b_f = \begin{cases} 1 & \text{if } F \geq 0 \\ \\ 0 & \text{if } F < 0. \end{cases}$$

where $F = \boldsymbol{S} \cdot (\rho \boldsymbol{v})_f$ is the flux. The unphysical oscillations are removed in this method because it depends on the flux direction. Also, it is bounded and stable but it is a first-order accurate because it uses the first-order backward differencing, (see [42]).

3. **Blended Differencing BD**

The BD is a combination between CD and UD and defined as:

$$\phi_f = (1 - k_f)(\phi_f)_{UD} + k_f(\phi_f)_{CD} \tag{2.32}$$

where $(\phi_f)_{UD}$ is the value from the UD, $(\phi_f)_{CD}$ is the value from CD and k_f is a blending factor between 0 and 1. This method attempts to preserve the accuracy and boundedness.

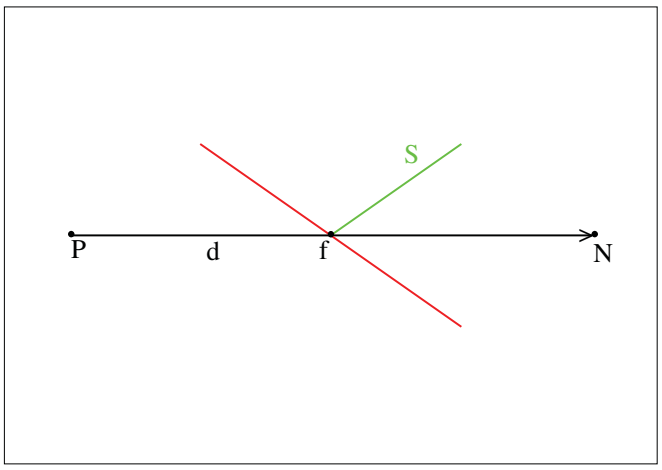

Figure 2.6: Vectors S and d on a non-orthogonal mesh.

2.4.1.2 Discretization of Diffusion Term

Following the approach used for the convection term, we get the approximation for the diffusion term

$$\begin{aligned} \oint_{\partial V_P} (\rho \Gamma_\phi \nabla \phi) \cdot \boldsymbol{n} dS &= \sum_f (\rho \Gamma_\phi \nabla \phi)_f \cdot \boldsymbol{S} \\ &= \sum_f (\rho \Gamma_\phi)_f \boldsymbol{S} \cdot (\nabla \phi)_f. \end{aligned} \tag{2.33}$$

If the mesh is orthogonal then the estimation for $\boldsymbol{S} \cdot (\nabla \phi)_f$ can be defined as:

$$\boldsymbol{S} \cdot (\nabla \phi)_f = \mid \boldsymbol{S} \mid \frac{\phi_N - \phi_P}{\mid \boldsymbol{d} \mid}. \tag{2.34}$$

If the mesh is non-orthogonal, as in Figure 2.6, then $\boldsymbol{S}\cdot(\nabla\phi)_f$ can be written as:

$$\boldsymbol{S}\cdot(\nabla\phi)_f = \underbrace{\boldsymbol{\Delta}\cdot(\nabla\phi)_f}_{\text{orthogonal contribution}} + \underbrace{\boldsymbol{K}\cdot(\nabla\phi)_f}_{\text{non-orthogonal contribution}}, \tag{2.35}$$

where $\boldsymbol{\Delta}$ is parallel to the vector $\boldsymbol{d}$ and $\boldsymbol{S} = \boldsymbol{\Delta} + \boldsymbol{K}$. The estimation in Eq. (2.34) can be used to approximate the orthogonal contribution and the non-orthogonal contribution can be approximated by approximating $(\nabla\phi)_f$ using the weighted average as:

$$(\nabla\phi)_f = b_f(\nabla\phi)_P + (1-b_f)(\nabla\phi)_N \tag{2.36}$$

where b_f is the same as in Eq. (2.31) and $(\nabla\phi)_P$ can be approximated using the second-order approximation to the Gauss divergence theorem as follows

$$\int_{V_P}\nabla\phi dV = \oint_{\partial V_P}\phi\cdot\boldsymbol{n}dS \tag{2.37}$$

$$(\nabla\phi)_P V_P = \sum_f\left(\int_f\phi\cdot\boldsymbol{n}_f dS\right) \tag{2.38}$$

$$(\nabla\phi)_P = \frac{1}{V_P}\sum_f\boldsymbol{S}\phi_f. \tag{2.39}$$

The integral on the left in Eq. (2.37) is approximated by multiplying the value of the function at the centroid of the CV by its volume. The integral on the face f is approximated by using the linear variation of ϕ on the interface $\phi(\boldsymbol{x}) = \phi_f + (\boldsymbol{x}-\boldsymbol{x}_f)\cdot(\nabla\phi)_f$. There

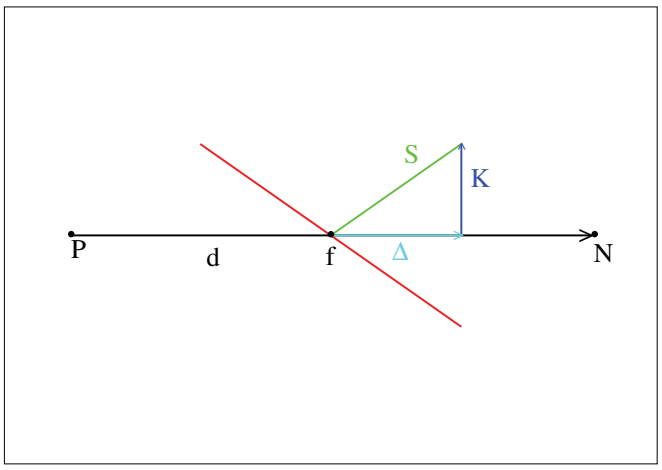

Figure 2.7: Vectors $\boldsymbol{\Delta}$ and $\boldsymbol{K}$ in the minimum correction approach.

are many ways to find the $\boldsymbol{\Delta}$ and $\boldsymbol{K}$ vectors. Here, two methods are described.

1. **Minimum correction approach**

   In this method, we choose $\boldsymbol{K}$ to be orthogonal to the vector $\boldsymbol{\Delta}$ to keep the non-orthogonal contribution as small as possible as shown in Figure 2.7. Also, $\boldsymbol{\Delta}$ can be written as:

$$\boldsymbol{\Delta} = \frac{\boldsymbol{d}\cdot\boldsymbol{S}}{\boldsymbol{d}\cdot\boldsymbol{d}}\boldsymbol{d}. \tag{2.40}$$

2. **Over-relaxed approach**

   In this method $\boldsymbol{\Delta}$ is defined as:

$$\boldsymbol{\Delta} = \frac{\boldsymbol{d}}{\boldsymbol{d}\cdot\boldsymbol{S}} \mid \boldsymbol{S} \mid^2 = \frac{\boldsymbol{S}\cdot\boldsymbol{S}}{\boldsymbol{d}\cdot\boldsymbol{S}}\boldsymbol{d}. \tag{2.41}$$

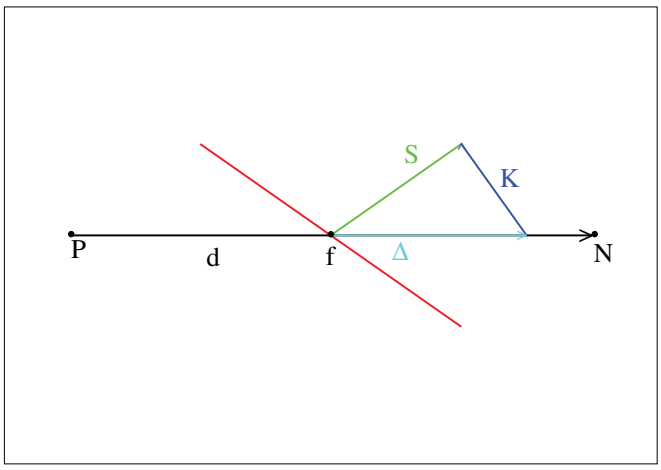

Figure 2.8: Vectors $\boldsymbol{\Delta}$ and $\boldsymbol{K}$ in the over-relaxed approach.

Figure 2.8 shows the vectors $\boldsymbol{\Delta}$ and $\boldsymbol{K}$ in the over-relaxed approach. In this approach, the importance of the term in ϕ_P and ϕ_N is caused to increase with the increase in non-orthogonality $\boldsymbol{\Delta}$.

The non-orthogonal correction possibly creates unboundedness, especially when the non-orthogonality is high. Therefore, if the boundedness is more important than accuracy, then the non-orthogonal correction has to be reduced or discarded. However this will result in reducing the order of accuracy [42]. According to Jasak [42], the over-relaxed approach is the best approach that treats the non-orthogonality from the aspect of stability, convergence, and computational time. The converged solution is obtained, even if the non-orthogonality is severe, when the other approaches cause divergence.

2.4.1.3 Discretization of Source Term

The source term $q_\phi(\phi)$ can be a function of ϕ and it is approximated by the linear expression

$$q_\phi(\phi) = q_u + q_p\phi, \tag{2.42}$$

where q_u and q_p can also depend on ϕ. This allows the implicit treatment of the source term. The integral form of the source term can be approximated as follows

$$\int_V q_\phi(\phi)dV = (q_u + q_p\phi)_P V_P \tag{2.43}$$

$$= q_u V_P + q_p V_P \phi_P. \tag{2.44}$$

2.4.1.4 Temporal Discretization

Using the previous discretization for the convection, diffusion, and source term, Eq. (2.22) can be written as

$$\int_t^{t+\delta t} \left[\left(\frac{\partial \rho \phi}{\partial t} \right)_P V_P + \underbrace{\sum_f F \phi_f}_{\text{convection term}} \underbrace{- \sum_f (\rho \Gamma_\phi)_f \boldsymbol{S} \cdot (\nabla \phi)_f}_{\text{diffusion term}} \right] dt = \int_t^{t+\delta t} \underbrace{(q_u V_P + q_p V_P \phi_P)}_{\text{source term}} dt \tag{2.45}$$

where $\int_{V_P} \frac{\partial \rho \phi}{\partial t} dV$ is approximated by the one-point centroid such as in the other terms. To finish the discretization the following approximations are used

$$\left(\frac{\partial \rho \phi}{\partial t} \right)_P = \frac{\rho_P^n \phi_P^n - \rho_P^o \phi_P^o}{\delta t} \tag{2.46}$$

$$\int_t^{t+\delta t} \phi(t) dt = (w\phi^o + (1-w)\phi^n)\delta t, \tag{2.47}$$

where $\phi^n = \phi(t+\delta t)$, $\phi^o = \phi(t)$ and w is a constant.

By using the previous equations, assuming the density and diffusivity do not change over

time and dividing by δt, Eq. (2.45) becomes

$$\begin{aligned}\frac{\rho_P\phi_P^n-\rho_P\phi_P^o}{\delta t}V_P &+ \sum_f\Big[(1-w)F\phi_f^n+wF\phi_f^o\Big]\\ &- \sum_f\Big[(1-w)(\rho\Gamma_\phi)_f\boldsymbol{S}\cdot(\nabla\phi)_f^n+w(\rho\Gamma_\phi)_f\boldsymbol{S}\cdot(\nabla\phi)_f^o\Big]\\ &= q_uV_P+(1-w)q_pV_P\phi_P^n+wq_pV_P\phi_P^o. \end{aligned}\tag{2.48}$$

For different w, various time integration methods can be obtained. For example, the first-order explicit Euler method is obtained if $w=1$, the first-order bounded Euler method is obtained if $w=0$, and the second-order Crank-Nicholson method is obtained if $w=\frac{1}{2}$. The values of ϕ_f and $(\nabla\phi)_f$ depend on the values of ϕ in the neighboring cells, therefore for any CV whose centroid is $\boldsymbol{x}_P$, Eq. (2.48) can be written as

$$a_P\phi_P^n+\sum_N a_N\phi_N^n=R_P. \tag{2.49}$$

The summation in Eq. (2.49) is over the neighboring cells of the cell with centroid P. From Eq. (2.48), the coefficients a_f and a_N result from the coefficients of ϕ_P and ϕ_N and the coefficients of these functions resulting from approximating ϕ_f and $\nabla\phi_f$ using ϕ_P and ϕ_N. For the whole geometry, this produces a linear system of algebraic equations of the

form

$$\boldsymbol{B}\boldsymbol{y} = \boldsymbol{R} \tag{2.50}$$

where $\boldsymbol{B}$ is a sparse matrix with coefficients a_P on the diagonal and a_N off the diagonal, $\boldsymbol{y}$ is the vector with the unknown values of ϕ on all CVs, and $\boldsymbol{R}$ is the source vector which contains firstly, the values of the constant part of the source term and secondly, the parts of convection term, diffusion term and temporal derivative at the old time level. Numerical approaches to solve the resulting equations, will be discussed later.

The momentum equation is a convection-diffusion equation with the pressure gradient as a source term. Therefore, it can be discretized using the same methods as in Sections 2.4.1.1–2.4.1.4. However, there are some additional complexities that must be addressed. Some of those are: (1) there are multiple equations and multiple unknowns such as $\boldsymbol{v}_x$, $\boldsymbol{v}_y$, $\boldsymbol{v}_z$, and pressure P; (2) there are nonlinear terms such as the convection $\nabla \cdot (\boldsymbol{v}\boldsymbol{v})$ and the viscous stress tensor τ for a non-Newtonian fluid; and (3) the equations are coupled and a specific treatment is required in order to handle the pressure-velocity coupling. This is discussed in the next section.

2.4.2 Pressure-Velocity Coupling

For incompressible non-Newtonian fluids, using the generalized Newtonian models (see Section 2.3) the mass and momentum equations have the form

$$\nabla \cdot \boldsymbol{v} = 0 \tag{2.51}$$

$$\frac{\partial(\rho \boldsymbol{v})}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}\boldsymbol{v}) - \nabla \cdot \eta(\dot{\gamma})(\nabla \boldsymbol{v} + \nabla \boldsymbol{v}^T) = -\nabla P \tag{2.52}$$

where η is the viscosity and $\dot{\gamma}$ is the shear rate. In the momentum equation, the nonlinear term $\nabla \cdot (\boldsymbol{v}\boldsymbol{v})$ appears. This issue is solved by solving a non-linear system, or by linearization which is the chosen option to reduce the computational time. The non-linear term is linearized as follows

$$\begin{aligned} \int_{V_P} \nabla \cdot (\rho \boldsymbol{v}\boldsymbol{v}) dV &= \int_{\partial V_P} (\rho \boldsymbol{v}\boldsymbol{v} \cdot \boldsymbol{n}) dS \\ &= \sum_f \boldsymbol{v}_f (\rho \boldsymbol{v})_f^o \cdot \boldsymbol{S} \\ &= \sum_f F^o \boldsymbol{v}_f \\ &= a_P \boldsymbol{v}_P + \sum_N a_N \boldsymbol{v}_N, \end{aligned} \tag{2.53}$$

where $\boldsymbol{v}^o$ is the velocity from the previous time step and $F^o = \boldsymbol{S} \cdot (\rho \boldsymbol{v})_f^o$ is the flux from the previous time step.

Another issue here is the incompressibility where the continuity equation does not involve density because it is constant which thus results in no explicit equation for pressure. In this system we have the same number of unknowns and equations. In the spirit of the Rhie and Chow procedure [45], a pressure equation can be derived from the continuity and momentum equations as follows. As in Section (2.4.1), the continuity equation can be discretized as

$$0 = \int_{V_P} \nabla \cdot \boldsymbol{v} dV = \int_{\partial V_P} \boldsymbol{v} \cdot \boldsymbol{n} dS = \sum_f \boldsymbol{S} \cdot \boldsymbol{v}_f \tag{2.54}$$

and the momentum equation as

$$a_P \boldsymbol{v}_P = \boldsymbol{H}(\boldsymbol{v}) - \nabla P \tag{2.55}$$

where

$$\boldsymbol{H}(\boldsymbol{v}) = -\sum_N a_N \boldsymbol{v}_N + \frac{\boldsymbol{v}^o}{\delta t}. \tag{2.56}$$

From Eq. (2.55), we have

$$\boldsymbol{v}_P = \frac{\boldsymbol{H}(\boldsymbol{v})}{a_P} - \frac{1}{a_P} \nabla P \tag{2.57}$$

and using interpolation, $\boldsymbol{v}_f$ can be written as

$$\boldsymbol{v}_f = (\frac{\boldsymbol{H}(\boldsymbol{v})}{a_P})_f - (\frac{1}{a_P}\nabla P)_f. \tag{2.58}$$

Substituting Eq. (2.58) into Eq. (2.54) yields

$$0 = \sum_f \boldsymbol{S} \cdot \left[(\frac{\boldsymbol{H}(\boldsymbol{v})}{a_P})_f - (\frac{1}{a_P}\nabla P)_f \right] = \sum_f \boldsymbol{S} \cdot (\frac{\boldsymbol{H}(\boldsymbol{v})}{a_P})_f - \sum_f \boldsymbol{S} \cdot (\frac{1}{a_P}\nabla P)_f \tag{2.59}$$

and hence

$$\sum_f \boldsymbol{S} \cdot (\frac{1}{a_P}\nabla P)_f = \sum_f \boldsymbol{S} \cdot (\frac{\boldsymbol{H}(\boldsymbol{v})}{a_P})_f. \tag{2.60}$$

The pressure gradient can be found by interpolating the pressure field to the cell faces and Eq. (2.55) can be written as

$$a_P \boldsymbol{v}_P = \boldsymbol{H}(\boldsymbol{v}) - \sum_f \boldsymbol{q}(P)_f. \tag{2.61}$$

Note that the flux F can be calculated as following

$$F = \boldsymbol{S} \cdot \left[(\rho \frac{\boldsymbol{H}(\boldsymbol{v})}{a_P})_f - (\rho \frac{1}{a_P}\nabla P)_f \right]. \tag{2.62}$$

Eqs. (2.60) and (2.61) are the discrete pressure and velocity equations. Both equations have two unknowns (P, $\boldsymbol{v}$) and the following three predictor-corrector methods are the

most commonly used to solve them.

2.4.2.1 The Semi-Implicit Method for Pressure-Linked Equation (SIMPLE) algorithm

The SIMPLE [46] algorithm is a predictor-corrector procedure for calculating the pressure P and velocity $\boldsymbol{v}$. It is a solver for steady-state (no time derivative) incompressible single phase fluid. The (implicit) under-relaxed form of the momentum equation, Eq. (2.61), can be written as

$$\frac{a_P}{\alpha_{\boldsymbol{v}}}\boldsymbol{v}_P^n + \sum_N a_N \boldsymbol{v}_N^n = R_P + \frac{1-\alpha_{\boldsymbol{v}}}{\alpha_{\boldsymbol{v}}} a_P \boldsymbol{v}_P^o, \tag{2.63}$$

where $\alpha_{\mathbf{v}}$ is the velocity under-relaxation factor ($0 < \alpha_{\mathbf{v}} \leq 1$) and $\boldsymbol{v}^o$ is the velocity from the previous iteration.

The SIMPLE algorithm can be outlined as following:

1. Start with a guessed value of pressure P^* in the first step and afterwards the resulted pressure from previous step is then used.

2. Solve the under-relaxed momentum Eq. (2.63) to find the velocity $\boldsymbol{v}^*$ by using the guessed pressure P^* to find R_P. This step is called the momentum predictor.

3. Compute the mass fluxes at the cells faces $F^* = \boldsymbol{S} \cdot \left[(\rho \frac{\boldsymbol{H}(\boldsymbol{v}^*)}{a_P})_f \right]$, which is needed in

the right hand side of the pressure Eq. (2.60).

4. Solve Eq. (2.60) to find the new value for the pressure P^{**}.

5. Correct the mass fluxes at the cells faces Eq. (2.62) using the new value for the pressure P^{**}, $F = F^{*} - (\rho \frac{1}{a_P} \nabla P^{**})_f \cdot \boldsymbol{S}$.

6. Apply some explicit pressure under-relaxation factor $0 < \alpha_P \leq 1$ to find the new pressure $P^{new} = P^{*} + \alpha_P (P^{**} - P^{*})$.

7. Calculate the corrected velocity $\boldsymbol{v}^{new}$ using Eq. (2.57) and the new pressure value P^{new}.

8. For a non-Newtonian fluid, update the viscosity from a generalized Newtonian constitutive equation using the corrected velocity.

9. Test for convergence and repeat the steps from step 2 assuming the new pressure P^{new} as the guessed pressure P^{*} if not converged.

If there are nonorthogonal cells in a mesh, then it may be desired to repeat step 4 for a specific number of iterations. In OpenFOAM®, this number is called `nNonOrthogonalCorrectors` and is specified in the file `<case>\system\fvSolution`. If this number is zero, then step 4 is performed one time. The recommended values for the under-relaxation factors according to [47] are $\alpha_P = 0.2$ and $\alpha_{\boldsymbol{v}} = 0.8$. In OpenFOAM® [33], the default values are $\alpha_P = 0.3$ and $\alpha_{\boldsymbol{v}} = 0.7$. The convergence is checked by the residual values of the velocity and pressure.

If each residual is below a specific tolerance then the solver will stop. In OpenFOAM®, the SIMPLE algorithm residuals are found in the file `<case>\system\fvSolution` under the name of `residualControl`.

2.4.2.2 Pressure Implicit with Splitting of Operators (PISO) algorithm

The PISO [48] algorithm was developed originally for a non-iterative computation of unsteady compressible flows, but it was further developed for steady calculation and for incompressible flow. The algorithm uses more than one corrector rather than one like in SIMPLE. In each time step, the algorithm can be described as follows:

1. Start with a guessed value of pressure P^* in the first step and afterwards the resulted pressure from previous step is then used.

2. Obtain an approximation for the velocity by solving the momentum equation Eq. (2.61), using the pressure from the previous time step.

3. Approximate the mass fluxes at the cell faces $F^* = \boldsymbol{S} \cdot \left[(\rho \frac{\boldsymbol{H}(\boldsymbol{v}^*)}{a_P})_f \right]$, which is needed in the right hand side of the pressure Eq. (2.60).

4. Using the approximated velocity, solve the pressure equation Eq. (2.60).

5. Find the final flux correcting the approximated flux by the pressure effect using

Eq. (2.62).

6. Correct the velocity using the new pressure value, where this is an explicit correction and is achieved using Eq. (2.57).

7. For a non-Newtonian fluid, update the viscosity from a generalized Newtonian constitutive equation using the corrected velocity.

The last five steps (3-7) are iterated a fixed number of times before moving to the next time step. In OpenFOAM®, this number is called `nCorrectors` and is specified in the file `<case>\system\fvSolution`. Also, as in the SIMPLE algorithm, the `nNonOrthogonalCorrectors` should be defined. This value determines how many times step 4 should be repeated. Note that no under-relaxation is performed for pressure or velocity, and there are no residual controls.

2.4.2.3 Merged PISO-SIMPLE (PIMPLE) algorithm

The PIMPLE algorithm uses the SIMPLE and PISO algorithms combined. It is a good algorithm to use for transient calculations. At each time step, the algorithm combines the SIMPLE algorithm and then uses the PISO algorithm to adjust the pressure correction. In each time step, the PIMPLE algorithm can be summarized as:

1. Calculate the velocity $\boldsymbol{v}^*$ using Eq. (2.63) and pressure P^* from previous time step.

2. Approximate the face flux $F^* = \boldsymbol{S} \cdot \left[(\rho \frac{\boldsymbol{H}(\boldsymbol{v}^*)}{a_P})_f \right]$, which is needed in the right hand side of the pressure Eq. (2.60).

3. Calculate the corrected pressure P^{**} using Eq. (2.60) and the approximated flux .

4. Correct the face fluxes using the new pressure value P^{**} via Eq. (2.62).

5. Apply an explicit under-relaxation for the pressure as in the SIMPLE algorithm. $P^{new} = P^* + \alpha_P(P^{**} - P^*)$.

6. Correct the velocity from the new pressure value P^{new} using Eq. (2.57).

7. Repeat steps 2-6 `nCorrectors` more times.

8. For a non-Newtonian fluid, update the viscosity from a generalized Newtonian constitutive equation using the corrected velocity.

9. Test for convergence using residual controls. If satisfied, move to next time step. If not, then repeat steps 1-8 at most `nOuterCorrectors` more times.

The convergence is controlled by `residualControl` as in the SIMPLE algorithm. If the `nOuterCorrectors` is equal to one, then the PIMPLE algorithm will be operating in PISO mode. The `nCorrectors`, `nOuterCorrectors`, and `residualControl` numbers are defined in the file `<case>/system/fvSolution`. The next section describes some of the iterative methods used to solve the linear systems encountered in the solution algorithms.

2.4.3 Linear Solvers

Each discretized momentum equation, pressure equation, and pressure correction equation results in a linear system of the form $\boldsymbol{By} = \boldsymbol{R}$. There are many methods to solve these linear systems. In this section, some of those methods are discussed.

1. **Generalized Geometric-Algebraic Multi-Grid (GAMG) Method**

   The multi-grid method is a fast method. The idea is to accelerate the convergence of an iterative method by correcting the solution from time to time. If the approximated solution to the linear system is $\boldsymbol{y}_h$ then the error is $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{y}_h$ and the residual is $\boldsymbol{r} = \boldsymbol{R} - \boldsymbol{B}\boldsymbol{y}_h$. The error $\boldsymbol{e}$ satisfies

$$\begin{aligned} \boldsymbol{Be} &= \boldsymbol{B}(\boldsymbol{y} - \boldsymbol{y}_h) && (2.64) \\ &= \boldsymbol{By} - \boldsymbol{By}_h && (2.65) \\ &= \boldsymbol{r}. && (2.66) \end{aligned}$$

   The multi-grid method solves the equation $\boldsymbol{Be} = \boldsymbol{r}$ on a coarser grid and then interpolates the solution to the fine grid. Then it adds the approximated error to the approximated solution. The method is achieved by defining the following matrices:

   - A restriction matrix $\boldsymbol{T}$ which transfers a vector from the fine grid to the coarse

grid

- An interpolation (prolongation) matrix $\boldsymbol{P}$ which returns the vector to the fine grid

The method can be summarized as follows:

(a) Solve the system $\boldsymbol{By} = \boldsymbol{R}$ by a few iterations to find $\boldsymbol{y}_h$

(b) Find the residual on the coarse grid by $\boldsymbol{r}_c = \boldsymbol{Tr}$

(c) Solve the system $\boldsymbol{B}_c\boldsymbol{e}_c = \boldsymbol{r}_c$

(d) Interpolate the error to the fine grid via $\boldsymbol{e}_h = \boldsymbol{Pe}_c$

(e) Add the error to the approximated solution $\boldsymbol{y}_{new} = \boldsymbol{y}_h + \boldsymbol{e}_h$

(f) Repeat steps (a)-(e) until convergence is reached

In Step (c), $\boldsymbol{B}_c = \boldsymbol{TBP}$ and the subscript c represents the coarse grid and the cell size in c is twice of the cell size in the original grid. The above multi-grid method is called v-cycle, meaning that we have only two grids, as shown in Figure 2.9 on the left, where f denotes the fine grid. The multi-grid method can be applied on more than two grids in the same way as in the v-cycle. Figure 2.9 on the right shows a multi-grid method on four grids. Multi-grid method can be used to find a good initial guess by finding the solution on the coarsest grid and interpolating it to the fine grid. The geometric multi-grid method uses the geometry to find the restriction matrix $\boldsymbol{T}$ and prolongation matrix $\boldsymbol{P}$. Algebraic multi-grid constructs the matrices from the

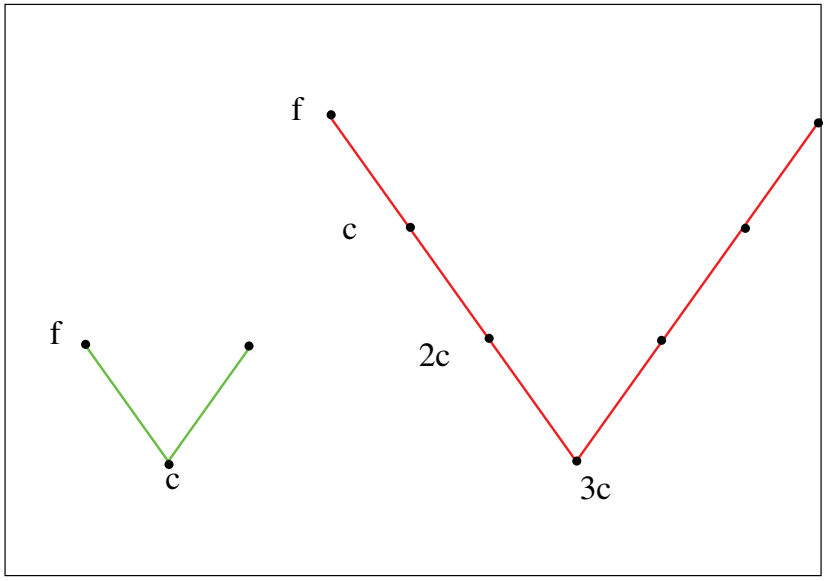

Figure 2.9: v-cycle and V-cycle.

matrix $\boldsymbol{B}$ and does not use the geometry. For that reason, it is a good choice for unstructured grids. The multi-grid method can be used as a preconditioner.

2. **Gauss Seidel Method**

The Gauss Seidel method uses the decomposition of the matrix $\boldsymbol{B} = \boldsymbol{D} - \boldsymbol{U} - \boldsymbol{L}$, where $\boldsymbol{B}$ is a symmetric positive-definite matrix, $\boldsymbol{D}$ is a diagonal matrix containing the diagonal entries of $\boldsymbol{B}$, $-\boldsymbol{U}$ is the upper triangular part of $\boldsymbol{B}$, and $-\boldsymbol{L}$ is the lower triangular part of $\boldsymbol{B}$. The linear system can be written as

$$(\boldsymbol{D} - \boldsymbol{L})\boldsymbol{y} = \boldsymbol{U}\boldsymbol{y} + \boldsymbol{R}. \tag{2.67}$$

The Gauss Seidel solves this linear system by using the value of $\boldsymbol{y}$ from the previous iteration on the right hand side of Eq. (2.67). The new value for $\boldsymbol{y}$ can be written as:

$$\boldsymbol{y}_{k+1} = (\boldsymbol{D}-\boldsymbol{L})^{-1}\boldsymbol{U}\boldsymbol{y}_k + (\boldsymbol{D}-\boldsymbol{L})^{-1}\boldsymbol{R}. \tag{2.68}$$

3. **Conjugate Gradient (CG) Method**

If the matrix $\boldsymbol{B}$ is symmetric positive-definite, then minimizing the quadratic function $f(\boldsymbol{y}) = \frac{1}{2}\boldsymbol{y}^T\boldsymbol{B}\boldsymbol{y} - \boldsymbol{R}^T\boldsymbol{y}$ is equivalent to solving the linear system $\boldsymbol{B}\boldsymbol{y} = \boldsymbol{R}$. Also, note that $\boldsymbol{r} = \boldsymbol{R} - \boldsymbol{B}\boldsymbol{y} = -\nabla f(\boldsymbol{y})$. The solution is updated iteratively via

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \alpha_k \boldsymbol{p}_k. \tag{2.69}$$

The idea here is to start with an initial guess $\boldsymbol{y}_0$ and then, at each step, walk in a direction such that $f(\boldsymbol{y}_{k+1}) < f(\boldsymbol{y}_k)$. The conjugate gradient method chooses the set of search direction vectors $\{\boldsymbol{p}_0, \boldsymbol{p}_1, ..., \boldsymbol{p}_n\}$ such that the set is $\boldsymbol{B}$-conjugate ($\boldsymbol{p}_i^T\boldsymbol{B}\boldsymbol{p}_j = 0, \forall i \neq j$). The step length α_k and search direction $\boldsymbol{p}_k$ are defined as

$$\alpha_k = \frac{\boldsymbol{r}_k^T\boldsymbol{r}_k}{\boldsymbol{p}_k^T\boldsymbol{B}\boldsymbol{p}_k} \tag{2.70}$$

$$\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_k \boldsymbol{p}_{k-1} \tag{2.71}$$

where $\beta_k = \frac{\boldsymbol{r}_k^T\boldsymbol{r}_k}{\boldsymbol{r}_{k-1}^T\boldsymbol{r}_{k-1}}$. The conjugate gradient method starts with an initial residual $\boldsymbol{r}_0 = \boldsymbol{R} - \boldsymbol{B}\boldsymbol{y}_0$ and calculates the initial guess for the search direction $\boldsymbol{p}_0 = \boldsymbol{r}_0$. It

then repeats the following steps from $k = 0$ until the residual gets below a specified tolerance:

(a) Calculate step length $\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T \boldsymbol{B} \boldsymbol{p}_k}$

(b) Calculate $\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \alpha_k \boldsymbol{p}_k$

(c) Calculate the new residual $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{B} \boldsymbol{p}_k$

(d) Calculate $\boldsymbol{\beta}_k = \frac{\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{r}_k}$

(e) Calculate the new direction $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$.

4. **Bi-Conjugate Gradient (BiCG) method**

The bi-conjugate gradient method is applicable for non-symmetric matrices. It uses both matrices $\boldsymbol{B}$ and $\boldsymbol{B}^T$. The method makes the two sets of search direction vectors $\{\boldsymbol{p}_0, \boldsymbol{p}_1, ..., \boldsymbol{p}_n\}$ for $\boldsymbol{B}$ and $\{\boldsymbol{q}_0, \boldsymbol{q}_1, ..., \boldsymbol{q}_n\}$ for $\boldsymbol{B}^T$ mutually orthogonal ($\boldsymbol{q}_i^T \boldsymbol{B} \boldsymbol{p}_i = 0$). As in the conjugate gradient method, the BiCG method starts with an initial guess $\boldsymbol{R} - \boldsymbol{B}\boldsymbol{y}_0 = \boldsymbol{r}_0 = \boldsymbol{p}_0 = \boldsymbol{s}_0 = \boldsymbol{q}_0$ and then repeat the following steps from $k = 0$ until convergence:

(a) Calculate step length $\alpha_k = \frac{\boldsymbol{s}_k^T \boldsymbol{r}_k}{\boldsymbol{q}_k^T \boldsymbol{B} \boldsymbol{p}_k}$

(b) Calculate $\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \alpha_k \boldsymbol{p}_k$

(c) Calculate the new residual of $\boldsymbol{B}$ as $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{B} \boldsymbol{p}_k$

(d) Calculate the new residual of $\boldsymbol{B}^T$ as $\boldsymbol{s}_{k+1} = \boldsymbol{s}_k - \alpha_k \boldsymbol{B}^T \boldsymbol{q}_k$

(e) Calculate $\boldsymbol{\beta}_k = \frac{\boldsymbol{s}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{s}_k^T \boldsymbol{r}_k}$

(f) Calculate the new direction $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$

(g) Calculate the new direction $\boldsymbol{q}_{k+1} = \boldsymbol{s}_{k+1} + \beta_k \boldsymbol{q}_k$.

Preconditioners

If the matrix $\boldsymbol{M}$ is nonsingular, then $\boldsymbol{By} = \boldsymbol{R}$ and $\boldsymbol{M}^{-1}\boldsymbol{By} = \boldsymbol{M}^{-1}\boldsymbol{R}$ have the same solution. The preconditioner $\boldsymbol{M}$ of the matrix $\boldsymbol{B}$ is a matrix such that $\boldsymbol{M}^{-1}\boldsymbol{B}$ has a smaller condition number than $\boldsymbol{B}$, where cond($\boldsymbol{B}$) $= \|\boldsymbol{B}\|\|\boldsymbol{B}^{-1}\|$.

Diagonal Incomplete-Cholesky (DIC)

This method can be used to find the preconditioner matrix $\boldsymbol{M}$ by using diagonal incomplete Cholesky decomposition. For the matrix $\boldsymbol{B}$, the diagonal incomplete Cholesky of a matrix $\boldsymbol{B}$ has the form $\boldsymbol{LDL}^T$ where $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{D}$ is a diagonal matrix. The preconditioner matrix $\boldsymbol{M}$ defined to be $\boldsymbol{M} = \boldsymbol{LDL}^T$.

Diagonal Incomplete Lower Upper (DILU)

The choice of the preconditioner matrix $\boldsymbol{M}$ in this method is defined to be $\boldsymbol{M} = (\boldsymbol{L} + \boldsymbol{D})\boldsymbol{D}^{-1}(\boldsymbol{D} + \boldsymbol{U})$ where $\boldsymbol{L}$ is the lower part of the matrix $\boldsymbol{B}$, $\boldsymbol{U}$ is the upper part of the matrix $\boldsymbol{B}$, and $\boldsymbol{D}$ is a diagonal matrix such that diagonal of $\boldsymbol{M}$ equal to the diagonal of $\boldsymbol{B}$.

Chapter 3

Dynamic Meshing For Two-Phase Flows

This chapter describes the two-phase (VOF) flow solvers in OpenFOAM®, namely `interFoam` and `interDyMFoam`. We start with some comments on the meshes in OpenFOAM®. All meshes in OpenFOAM® are 3D Cartesian meshes, even for 2D simulations. In 2D simulations, the computational domain in one of the coordinate directions is always one cell thick. In OpenFOAM® the projection of the computational domain boundary in the other two directions are called empty patches. However, in the axisymmetric calculations, they are called wedge patches. Each cell in the mesh is assigned a designated number which contains a number of points and faces. There are two kinds of faces: an internal face that connects two cells, an owner and a neighbor cell, and a boundary face that belonging to one owner cell. Each face is also assigned a designated number where an internal face has an owner cell with the lower number and a neighbor cell with the higher

number.

Dynamic mesh refinement allows us to refine the cells in a coarse mesh at specific regions that requires smaller cells. It is a good way to get accurate results with reduced computational time in comparison to a refined static mesh. The coarse mesh should be refined enough to give an accurate result outside the region that requires small cells. In two-phase flow, the refinement should be on the interface between the fluids because of the steep gradients in the volume fraction function and potentially the material properties of the fluid system. In the following section, the discretization of the volume fraction equation is described since it is needed in the two-phase flow calculation.

3.1 Discretization of Volume Fraction Equation

In this section, the discretization of the volume fraction Eq. (2.10) is described. Because the conservation of the phase fraction is important to give accurate physical properties, such as density, especially for the fluids with high density ratio, an artificial compression term is added to Eq. (2.10) and the volume fraction equation becomes [49]

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha \boldsymbol{v}) + \nabla \cdot ((\alpha(1-\alpha)\boldsymbol{v}_r) = 0 \tag{3.1}$$

where $\boldsymbol{v}_r = \boldsymbol{v}_2 - \boldsymbol{v}_1$ and the subscripts represent the fluid phase. Note that the artificial compression term is nonzero only in a thin region around the interface due to the factor of $\alpha(1-\alpha)$; therefore it does not affect the solution outside the interface region. The compression term reduces the numerical diffusion, thus allowing sharp interface resolution. The integral form of Eq. (3.1) is

$$\int_t^{t+\delta t} \int_{V_P} \frac{\partial \alpha}{\partial t} dV dt + \int_t^{t+\delta t} \int_{V_P} \nabla \cdot (\alpha \boldsymbol{v}) dV dt + \int_t^{t+\delta t} \int_{V_P} \nabla \cdot ((\alpha(1-\alpha)\boldsymbol{v}_r) dV dt = 0. \tag{3.2}$$

By applying the Gauss divergence theorem the equation become

$$\int_t^{t+\delta t} \left[\int_{V_P} \frac{\partial \alpha}{\partial t} dV + \oint_{\partial V_P} \boldsymbol{n} \cdot (\alpha \boldsymbol{v}) dS + \oint_{\partial V_P} \boldsymbol{n} \cdot ((\alpha(1-\alpha)\boldsymbol{v}_r) dS \right] dt = 0. \tag{3.3}$$

The discretization of the terms in Eq. (3.3) is done in the same way as the discretization of the terms in the general convection-diffusion equation in Section 2.4.1. The discretized equation has the form

$$\begin{aligned} \frac{\alpha_P^n - \alpha_P^o}{\delta t} V_P \quad &+ \quad \sum_f \left[(1-w)\psi \alpha_f^n + w \psi \alpha_f^o \right] \\ &+ \quad \sum_f \left[(1-w)\Psi(\alpha(1-\alpha))_f^n + w \Psi(\alpha(1-\alpha))_f^o \right] = 0 \end{aligned} \tag{3.4}$$

where S_f is the face area, $\boldsymbol{S} = S_f \boldsymbol{n}_f$ is the outward area vector, $\psi = \boldsymbol{S} \cdot \boldsymbol{v}_f$ is the face flux of the linear term, and Ψ is the face flux of the non-linear term and is calculated based on the maximum velocity magnitude at the interface region and its direction as:

$$\Psi = \boldsymbol{n_f} \min \left[C_\alpha \frac{|\,\boldsymbol{\phi}\,|}{|\,\boldsymbol{S}\,|}, \max \left(\frac{|\,\boldsymbol{\phi}\,|}{|\,\boldsymbol{S}\,|} \right) \right], \tag{3.5}$$

where $\boldsymbol{\phi} = \boldsymbol{S} \cdot \boldsymbol{v}_f$ is face volume flux and C_α is an adjustable coefficient which determines the magnitude of the compression. In OpenFOAM®, C_α is defined in the `<case>/system/fvSolution` file. As in the previous section, different values of w give different numerical methods. In OpenFOAM®, the explicit method is used ($w = 1$). The fluxes are calculated from the previous time step.

In the next two sections, the implementation of the governing equations for two-phase flow problems is described for the `interFoam` and `interDyMFoam` solvers in `OpenFOAM`. After that, a brief comparison between them is presented for a 3D test problems before describing the modifications for 2D and axisymmetric flows.

3.2 Description of interFoam Solver

The `interFoam` solver for two immiscible incompressible fluids uses a VOF (volume of fluid) phase-fraction based interface capturing approach. It uses an adaptive time step depending on the Courant number $Co = \frac{|v|\delta t}{\delta x}$. To choose the new time step, a maximum

Courant number Co^o is calculated from the flow conditions, using $\boldsymbol{v}$ and δt from the previous time step. The new time step δt^n is then calculated using the following expression [50]

$$\delta t^n = \min\left\{\frac{Co_{max}}{Co^o}\delta t^o; (1+0.1\frac{Co_{max}}{Co^o})\delta t^o; 1.2\delta t^o; \delta t_{max}\right\} \tag{3.6}$$

where δt^o is the old time step, Co_{max} is the pre-set maximum Courant number, and δt_{max} is the pre-set maximum time step. The values of Co_{max} and δt_{max} are specified in the `<case>/system/controlDict` file.

It is critical to have the volume fraction α value accurate because it affects other physical properties such as the density and viscosity, as well as the interface curvature. Therefore, the volume fraction equation is solved in sub-cycles within each time step. The new sub-cycle time step is calculated from the time step for the flow

$$\delta t_{sc} = \frac{\delta t}{nAlphaSubCycles}, \tag{3.7}$$

where `nAlphaSubCycles` is the number of sub-cycles defined in the file `<case>/system/fvSolution`. The flux $F_{sc} = \boldsymbol{S}\cdot(\rho\boldsymbol{v})_f$ is calculated at each δt_{sc} and the total flux, which is needed in the momentum equation, is calculated as $F = \sum_1^{nAlphaSubCycles}\frac{\delta t_{sc}}{\delta t}F_{sc}$. The use of sub-cycles speeds up the calculations by allowing smaller time steps for the evolution of α, while retaining larger steps for solving the other equations.

The specification of the pressure boundary conditions is simplified if the modified pressure $\widehat{P}$ is used which is obtained by removing the hydrostatic pressure from the pressure P. It defined as

$$\widehat{P} = P - \rho \boldsymbol{g} \cdot \boldsymbol{x}, \tag{3.8}$$

where ρ is the density, $\boldsymbol{g}$ is the gravity, and $\boldsymbol{x}$ is the cell center. Using Eq. (3.8), the pressure gradient is

$$\nabla P = \nabla \widehat{P} + \rho \boldsymbol{g} + \boldsymbol{g} \cdot \boldsymbol{x} \nabla \rho. \tag{3.9}$$

In addition to the advantage of a simpler specification of the pressure boundary condition, this treatment enables efficient numerical treatment of the steep density jump at the interface.

The pressure gradient in the momentum equation, Eq. (2.11), is replaced by the pressure gradient in Eq. (3.9). Hence, the momentum equation becomes

$$\rho(\alpha)(\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v} \cdot \nabla \boldsymbol{v}) = -\nabla \widehat{P} - \boldsymbol{g} \cdot \boldsymbol{x} \nabla \rho + \nabla \cdot \eta(\dot{\gamma})(\nabla \boldsymbol{v} + \nabla \boldsymbol{v}^T) + \sigma \kappa(\alpha) \nabla \alpha. \tag{3.10}$$

Using the same technique as in Section 2.4.2, the pressure-velocity coupling equations are: the momentum equation

$$a_P \boldsymbol{v}_P = \boldsymbol{H}(\boldsymbol{v}) - \nabla \widehat{P} - \boldsymbol{g} \cdot \boldsymbol{x} \nabla \rho + \sigma \kappa \nabla \alpha, \tag{3.11}$$

the pressure equation

$$\sum_f \boldsymbol{S} \cdot (\frac{1}{a_P} \nabla \widehat{P})_f = \sum_f \boldsymbol{S} \cdot \left[(\frac{\boldsymbol{H}(\boldsymbol{v})}{a_P})_f - (\frac{\boldsymbol{g} \cdot \boldsymbol{x} \nabla \rho}{a_P})_f + (\frac{\sigma \kappa \nabla \alpha}{a_P})_f \right], \tag{3.12}$$

and the flux equation

$$F = \boldsymbol{S} \cdot \left[(\frac{\boldsymbol{H}(\boldsymbol{v})}{a_P})_f - (\frac{\boldsymbol{g} \cdot \boldsymbol{x} \nabla \rho}{a_P})_f + (\frac{\sigma \kappa \nabla \alpha}{a_P})_f - (\frac{1}{a_P} \nabla \widehat{P})_f \right]. \tag{3.13}$$

The `interFoam` was validated on several benchmarks tests [23]. The solver initiates the variables and then starts the time loop. Refer to the source code of `interFoam` solver which is located in the folder `OpenFOAM/OpenFOAM2-1-0/application/solvers/multiphase/interFoam`. In each time step, the solver is outlined as follows:

1. Calculate the Courant number by calling the `CourantNo.H` library and adjust the time step by calling the `setDeltaT.H` library.

2. Correct the phase properties, such as density and viscosity, using the new volume fraction α by calling the function `twoPhaseProperties.correct()`.

3. Solve the volume fraction equation (Eq. 3.4) as described in Section 3.1 to find α using the fluxes from the previous time step by calling the `alphaEqnSubCycle.H` library. The α value is iteratively corrected via Eq. (3.4) a number of times equal to `nAlphaCorr`, which is specified in the `<case>/system/fvSolution` file.

The cAlpha keyword specified in the file <case>/system/fvSolution is a factor that controls the compression of the interface, where 0 corresponds to no compression. After solving the volume fluid equation, the density and viscosity are modified using the new values of α.

4. Start the PIMPLE loop to solve for the pressure and velocity as described in Section 2.4.2.3. In this step, the volume fraction function α from the previous step is used to calculate the CSF term $\sigma\kappa\nabla\alpha$.

The solver repeats these steps until a pre-set time which is specified in the <case>/system/controlDict file under the name endTime.

3.3 Dynamic Mesh Refinement in interDyMFoam Solver

The interDyMFoam solver is the same as the interFoam solver but with the ability of mesh motion and dynamic mesh refinement. In this study, we will concentrate on the dynamic mesh refinement. The interDyMFoam solver can do the refinement only for the 3D hexahedral cells by partitioning the cells equally in all three directions . The mesh refinement in interDyMFoam is achieved by using the dynamicFvMesh and dynamicMesh libraries. The mesh refinement is initialized by calling the function mesh.update() (refer to the source code of interDyMFoam solver). This function is defined in the dynamicFvMesh library and can be found in the file

OpenFOAM/OpenFOAM2-1-0/src/dynamicFvMesh/dynamicRefineFvMesh/dynamicRefineFvMesh.C. This function can carries out the refinement as follows:

1. Reads the dynamicMeshDict file which is located in the constant folder of the case directory. This file has some values that are needed to conduct the refinement, which are further explained at the end of this section.

2. Determines the candidate cells that can be refined by calling the function selectRefineCandidates() which is located in dynamicRefineFvMesh.C file. The cells are chosen based on three bases. First, a field is specified, which can be the magnitude of the velocity or the volume fraction function alpha1. Second, a maximum number of refinement is specified. The first and second bases are determined in the dynamicMeshDict file. Finally, the cell must have an nAnchors value of 8. The nAnchors is defined in the dynamicRefineFvMesh.C file. If the value of nAnchors for the cell is not 8, then the cell can not be refined. To find the value of nAnchors for a given cell, a loop is taken over all the points of the cell and if the pointLevel is less than or equal to cellLevel for a point, then this point is added to the nAnchors. In the dynamicRefineFvMesh file, the cellLevel and pointLevel are defined such that each cell has a cellLevel starting with 0 for the original cell and if the cell is refined once then this number becomes 1 for each new cell and so on. The value of pointLevel is similarly defined.

3. Selects a subset of candidate cells for refinement by calling the function `selectRefineCells()` which is located in the `dynamicRefineFvMesh.C` file. The subset is chosen based on the maximum number of cells allowed, which is defined in the `<case>/constant/dynamicMeshDict` file.

4. Perform the refinement by calling the function `refine()` which is located in the `dynamicFvMesh.C` file. The function `refine()` calls the function `setRefinement()` which is defined in the `dynamicMesh` library and can be found in the file `OpenFOAM/OpenFOAM2-1-0/src/dynamicMesh/polyTopoChange/polyTopoChange/hexRef8.C`. This function can do the refinement as follows:

   (a) For each cell to be refined a point is added in the center of the cell as shown in Figure 3.1.

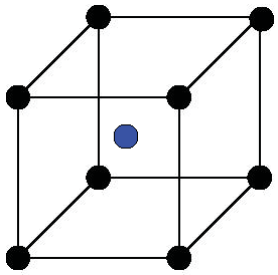

Figure 3.1: A hexahedral cell with a point in the middle

   (b) For each cell to be refined, a point is then added in the center of each face of the cell, one of which is illustrated in Figure 3.2.

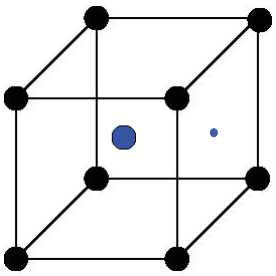

Figure 3.2: A point in the middle of a face

(c) For each cell to be refined, a point is added in the middle of each edge, as illustrated in Figure 3.3.

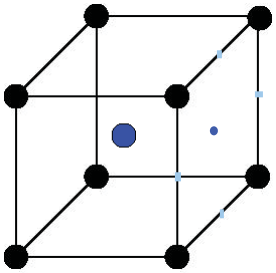

Figure 3.3: A points in the middle of an edges

(d) Each face is divided into four new faces as shown in Figure 3.4 and each new face assigned an owner and a neighbor cell.

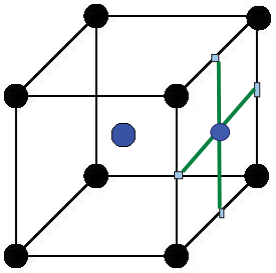

Figure 3.4: Divide a face into four faces

(e) Internal faces are added to the cell by connecting the points in the center of two neighboring faces, the point in the center of the edge that connects the faces, and the point in the center of the cell, as shown in Figure 3.5. Therefore, a neighbor and an owner cell are assigned to each face.

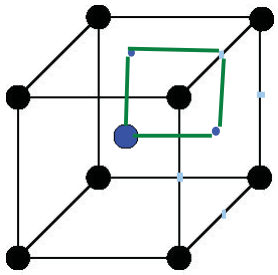

Figure 3.5: Internal face added to the cell

(f) The fields are mapped from the old mesh to the new mesh as an initial condition to speed up the computational process. The field value at the centroid of a cell in the original mesh is transfered to the new cells by assigning them the same values.

5. Determines the points that can be unrefined by calling the function `selectUnrefinePoints()` which is located in the `dynamicRefineFvMesh.C` file. The points are chosen based on the `PointLevel` and `nBufferLayers` numbers which are defined in the file `<case>/constant/dynamicMeshDict`. If the `PointLevel` is greater than 0, then the point can be unrefined. The `nBufferLayers` number will be discussed later in this section.

6. The unrefinement can be done by calling the function `unrefine()` located in the `dynamicFvMesh.C` file. The unrefinement is achieved by calling the function `setUnrefinement()` defined in the `dynamicMesh` library and can be found in the file `OpenFOAM/OpenFOAM2-1-0/src/dynamicMesh/polyTopoChange/polyTopoChange/hexRef8.C`. This function removes the points selected by the function `selectUnrefinePoints()` and their connected faces and points.

7. Finally, the fields are mapped from the old to the new mesh. The values at the centroid are mapped by taking the average for the small cells.

After the refinement is done, the fluxes are corrected in the solver using the new values of the velocity. Figure 3.6 shows the `dynamicMeshDict` file. In this file, the refine interval should be one or greater. This number indicates the time step occurrence of the refinement. For example, if the refine interval is two, the refinement will be operated every second time step. The field that is used to determine the cells requiring refinement is specified to be `alpha1`. The lower and upper refine levels determine the range of the field such that each cell having the field value in this range will be a candidate for the refinement. The unrefinement level is the number that controls the points which can be unrefined, and the number of buffer layers is used to find the buffer layers that should be extended for unrefinement. Each cell can be refined up to the maximum refinement number and if the total cells exceed the maximum number of cells, the refinement stops. The fluxes that needs

to be corrected is defined in this file as well. Finally, the dumpLevel is true to write the refinement level.

```
dynamicFvMesh    dynamicRefineFvMesh;

dynamicRefineFvMeshCoeffs
{
    refineInterval    1;

    field             alpha1;

    lowerRefineLevel  0.001;

    upperRefineLevel  0.999;

    unrefineLevel     10;

    nBufferLayers     1;

    maxRefinement     2;

    maxCells          200000;

    correctFluxes
    (
        (phi  Urel)
        (phiAbs  U)
        (phiAbs_0  U_0)

    );

    dumpLevel         true;
}
```

Figure 3.6: Example of a dynamicMeshDict file.

3.4 Test of interFoam and interDyMFoam in 3D

In this section, we will compare the `interFoam` and `interDyMFoam` solvers from the aspects of CPU time, mesh independence, and cell size around the interface. The test case is a three-dimensional liquid drop in a linear shear flow. The computational domain is a

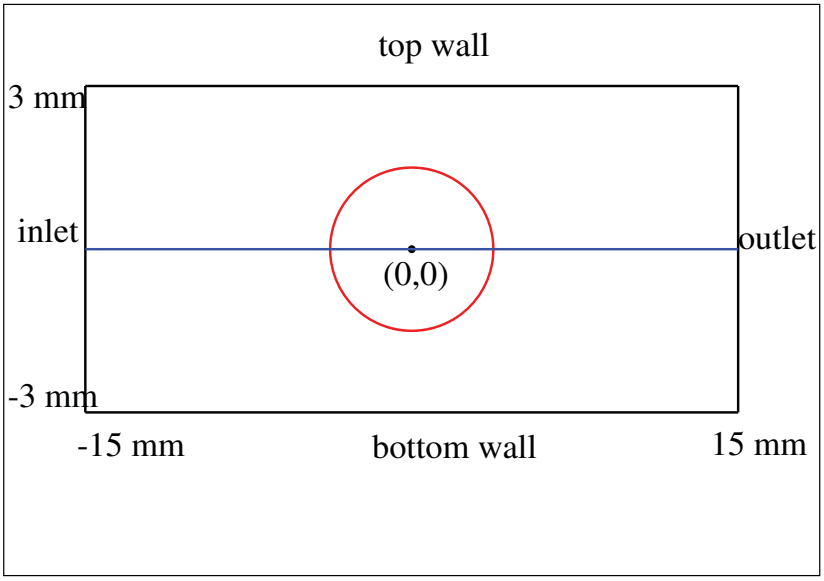

Figure 3.7: Schematic diagram of a drop of radius 1 mm centered in a channel. The x-axis and y-axis are horizontally and vertically, respectively, and the positive z-axis point out of the paper.

channel of length 30 mm, height 6 mm, and depth 6 mm. The origin of the coordinate system is placed at the center of the domain, so that $-15 \leq x \leq 15$, $-3 \leq y \leq 3$, and $-3 \leq z \leq 3$. The drop is a sphere of radius 1 mm and center at $(0,0,0)$. At time $t = 0$, the upper wall (at $y = 3$ mm) moves at a constant speed of u in the positive x-direction, and the lower wall (at $y = -3$ mm) moves at a constant speed of u in the negative x-direction. The side walls (at $z = \pm 3$ mm) remain stationary. The geometry is illustrated in Figure 3.7. The continuous and disperse phases are taken to be Newtonian fluids. The transport properties are as follows:

$\mu_c = \mu_d = 1.06 \times 10^{-1} Pas$

$\rho_c = \rho_d = 10^3 kg/m^3$

$\sigma = 0.0415 N/m$

where μ is the dynamic viscosity, ρ is the density, and σ is the interfacial tension. The

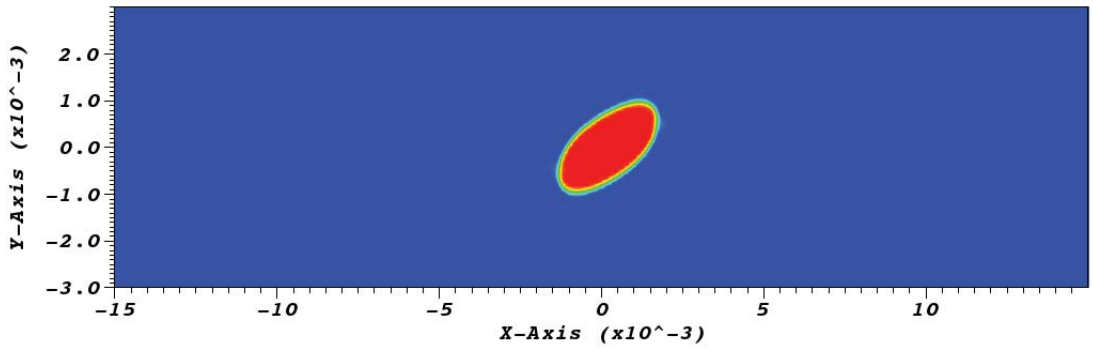

Figure 3.8: A droplet at steady-state for the 3D drop in shear flow test case ($Ca = 0.3$).

viscosity ratio and density ratio are $\lambda = \frac{\mu_d}{\mu_c} = 1$ and $\frac{\rho_d}{\rho_c} = 1$.

The capillary number is a dimensionless number that characterizes the ratio of viscous forces to interfacial tension forces. It is defined as

$$Ca = \frac{a\mu_c\dot{\gamma}}{\sigma}, \tag{3.14}$$

where $\dot{\gamma}$ is the shear rate, and a is the radius of the undeformed droplet. The subscripts, c and d, denote the continuous and disperse phases respectively. The critical capillary number Ca_{crit} is the value of Ca above which drop breakup occurs ($Ca > Cacrit$) and below which breakup does not occur ($Ca < Cacrit$). The critical capillary number in a given type of flow depends on the viscosity ratio.

Table 3.1 shows the boundary conditions for the test case. The `zeroGradient` condition means that the derivative normal to the boundary is zero. For example, the condition of the

Table 3.1
Boundary conditions for the 3D drop in shear flow test case, where Ca is the capillary number

boundary	velocity	$p\_\rho gh$	alpha (α)
inlet	zeroGradient	zeroGradient	zeroGradient
outlet	zeroGradient	0	zeroGradient
bottom	(-1.175$(m/s)*Ca$ 0 0)	zeroGradient	zeroGradient
top	(1.175$(m/s)*Ca$ 0 0)	zeroGradient	zeroGradient
front and back	(0 0 0)	zeroGradient	zeroGradient

velocity on the inlet boundary means $\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}} = 0$ since x-direction is the normal for the inlet. The velocities in the x-direction for the upper and lower wall boundaries are given in terms of the capillary number. This dependence comes from the formula of the capillary number above and the formula of the shear rate $\dot{\gamma} = \frac{u}{H}$, where H is the half distance between the bottom and top (here $H = 3$ mm). We take $Ca = 0.3$ which is a sub-critical capillary number for a viscosity ratio of $\lambda = \frac{\mu_d}{\mu_c} = 1$. The end time for all cases is 1 s. The parameters used in the `interDyMFoam` solver, defined in the `<case>/constant/dynamicMeshDict` file, are shown in Table 3.2.

For this test case, steady-state is reached. The stationary drop shape (at time $t = 0.99$ s) is shown in Figure 3.8. The steady-state results in the following sections are represented, in part, by graphing data along the horizontal line that passes through the origin ($y = 0$) as shown in Figure 3.7.

Table 3.2
DynamicMeshDict parameters for the 3D drop in shear flow test case

refineInterval	field	lowerRefInterval	upperRefInterval
1	alpha (α)	0.1	0.9

unrefineLevel	nBufferLayers	maxRefinement	maxCells
10	1	2	400000

3.4.1 Mesh Independence Study

This section studies the mesh independence for the `interFoam` and `interDyMFoam` solvers. The study is achieved by comparing the solutions of the velocity and pressure on three different meshes, described in Table 3.3. Figure 3.9 shows the velocity along the centerline ($y = 0$) for the three different meshes using `interFoam`. As the mesh is refined, the difference between the velocity curves becomes smaller, and the velocity curves become smoother. Similar results were found for the pressure shown in Figure 3.10. Figures 3.11 and 3.12 show the velocity and pressure for the `interDyMFoam` solver on the three different meshes and, for comparison, for the `interFoam` solver on the fine mesh. The graphs for the velocity and pressure show almost identical results on the three meshes where the difference between the curves is negligible. The `interFoam` results on the fine mesh agree better with those from `interDyMFoam` in terms of velocity than pressure.

The pressure graphs in Figures 3.10 and 3.12 show a jump in pressure within the droplet. The jump in pressure is almost the same for the three meshes using `interDyMFoam`

solver. It is clear that the coarse mesh used with `interFoam` is insufficient since the solution curves have a lot of oscillation and they are inaccurate compare to the solution curves from the standard and fine meshes. Figures 3.9–3.12 show that `interDyMFoam` performs better than `interFoam` in terms of producing smooth mesh independence results for this test case. As seen in Table 3.4, this is due in large part to the cell size around the interface. The cell size around the interface using the coarse mesh of `interDyMFoam` is smaller than the cell size using `interFoam` with fine mesh. More refinement around the interface is needed to improve the results of `interFoam`. This would require a more global refinement of the mesh compared to `interDyMFoam`, and would therefore increase the CPU time.

Table 3.3
Initial mesh and number of cells using interFoam and interDyMFoam for the 3D drop in a shear flow test case

Solver	Initial mesh	Number of cells
interFoam(Coarse)	$125\times25\times25$	78125
interFoam(Standard)	$165\times33\times33$	179685
interFoam(Fine)	$218\times43\times43$	403082
interFoam(non-uniform)	$50,65,50\times33\times33$	179685
interDyMFoam(Coarse)	$76\times\ 15\times15$	$17100-27000$
interDyMFoam(Standard)	$100\times\ 20\times20$	$40000-53000$
interDyMFoam(Fine)	$132\times\ 26\times26$	$89232-115000$

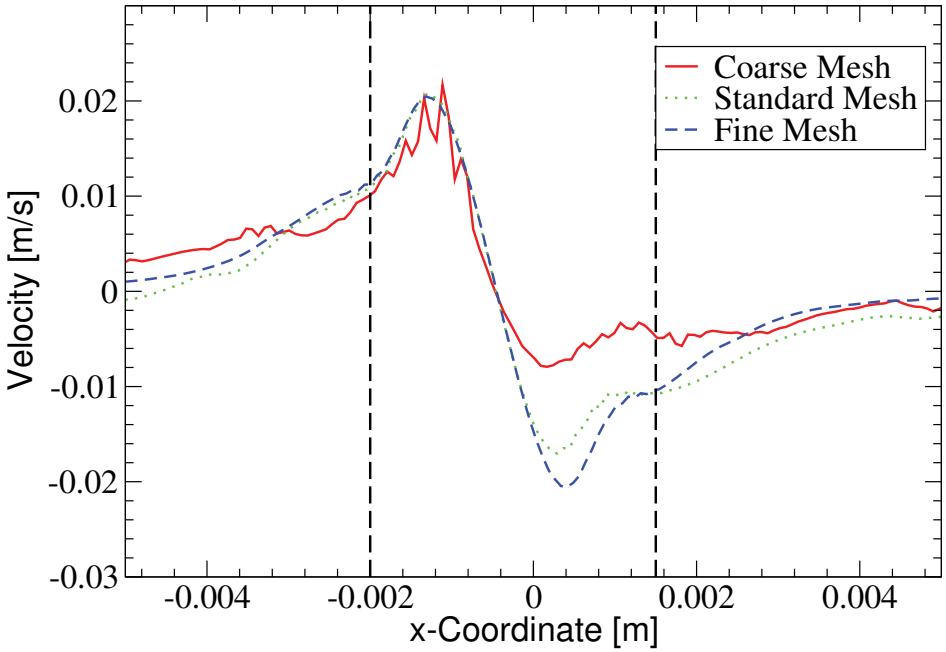

Figure 3.9: Velocity at steady-state along line $y = 0$ using interFoam for the 3D drop in shear flow test case $Ca = 0.3$. The vertical lines indicate the boundary of the drop along $y = 0$.

3.4.2 Comparison Between interFoam And interDyMFoam Using Serial Calculations

This section compares the two solvers from the aspects of the CPU time, the number of cells, and cell size around the interface. For the `interFoam` solver there are two cases. In the first case, a uniform mesh is considered and in the second case (non-uniform), the computational domain is divided into three blocks in the x-direction to have smaller cells around the interface. The minimum and maximum x-coordinate for the blocks are $-15, -3$; $-3, 3$; and $3, 15$ started from block one to block three respectively. Table 3.4 shows the difference in CPU time, number of cells, and cell size around the interface. The

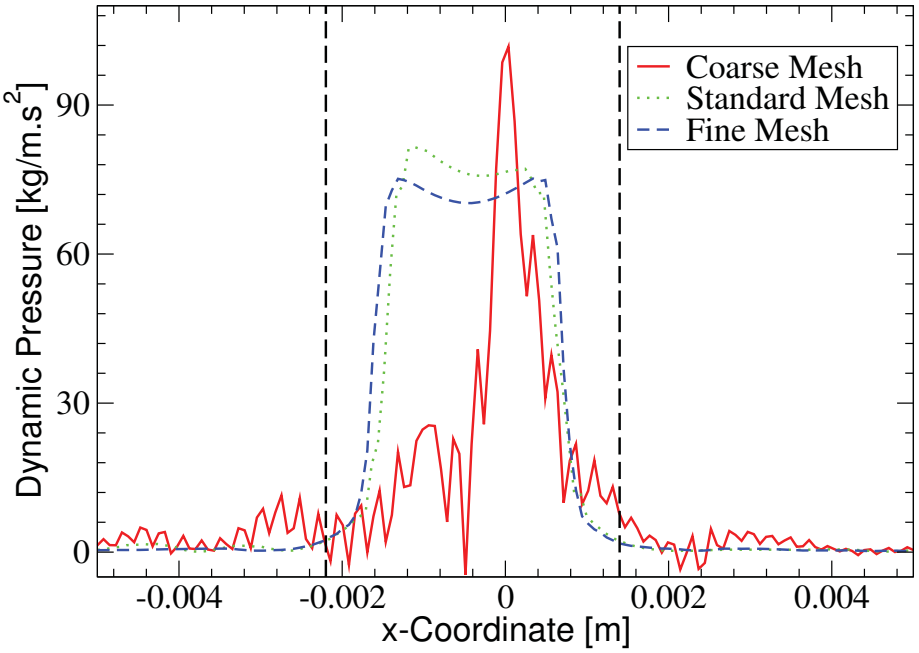

Figure 3.10: Pressure at steady-state along line $y = 0$ using interFoam for the 3D drop in shear flow test case $Ca = 0.3$. The vertical lines indicate the boundary of the drop along $y = 0$.

`interDyMFoam` has a lower overall CPU time, even when comparing its fine mesh with `interFoam`'s standard and non-uniform meshes, keeping in mind that the cell size around the interface is smaller for `interDyMFoam`. This is due in part to the fewer number of cells in the fine mesh of `interDyMFoam` (maximum of 115000 cells) compared with the standard and non-uniform meshes of `interFoam` (179685 cells). Note that although the standard uniform mesh and non-uniform mesh of `interFoam` have the same number of cells, the CPU time is almost doubled when using the non-uniform mesh. This is because the cell size in the x-direction for the non-uniform mesh is $\delta = 0.09$ mm which is half of the cell size in the uniform mesh (0.18 mm), making the time step size smaller in the non-uniform mesh since it depends on the Courant number $Co = \frac{|v|\delta t}{\delta x}$. Specifically, if the cell size (δx) is decreased then the time step will decrease to have the Courant number

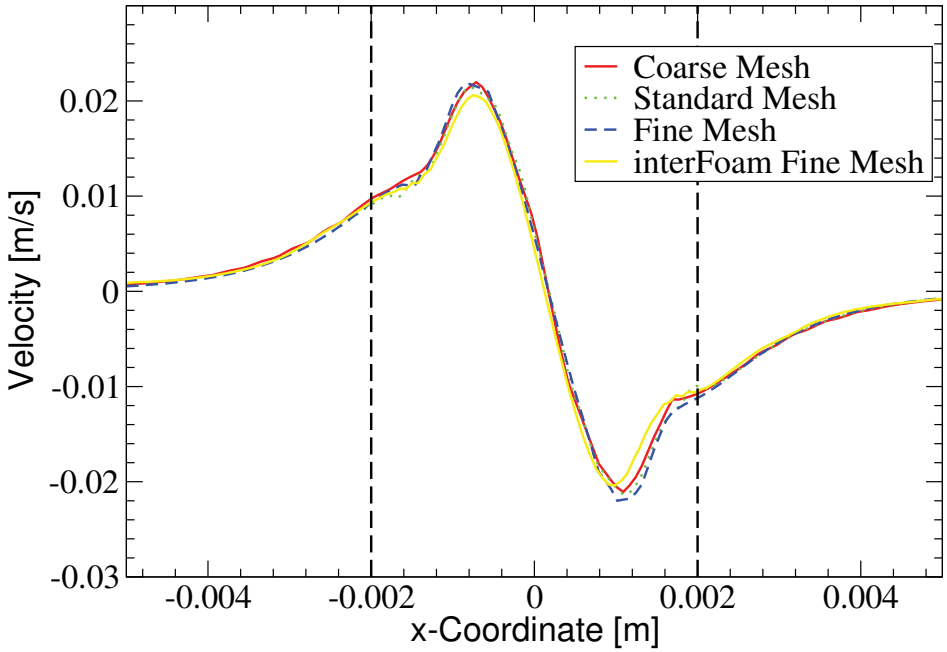

Figure 3.11: Velocity at steady-state along line $y = 0$ using interDyMFoam for the 3D drop in shear flow test case $Ca = 0.3$. The vertical lines indicate the boundary of the drop along $y = 0$.

below a specific number defined in the `<case>/system/controlDict` file. Indeed, the typical time step was nearly doubled on the standard uniform mesh (2.7×10^{-4} s) compared to the non-uniform mesh (1.4×10^{-4} s). The smaller cells on the non-uniform mesh were located in the center part of the domain, where we know the drop remains. In general, the location of the interface is unknown as time passes. Thus, when using a static mesh, it is hard to refine the relevant region of the computational domain. Therefore, in most cases, by using a static mesh, the whole mesh has to be refined, causing an increase in the CPU time dramatically. Therefore, `interDyMFoam` will be a better option to do the refinement around the interface with a much reduced CPU time.

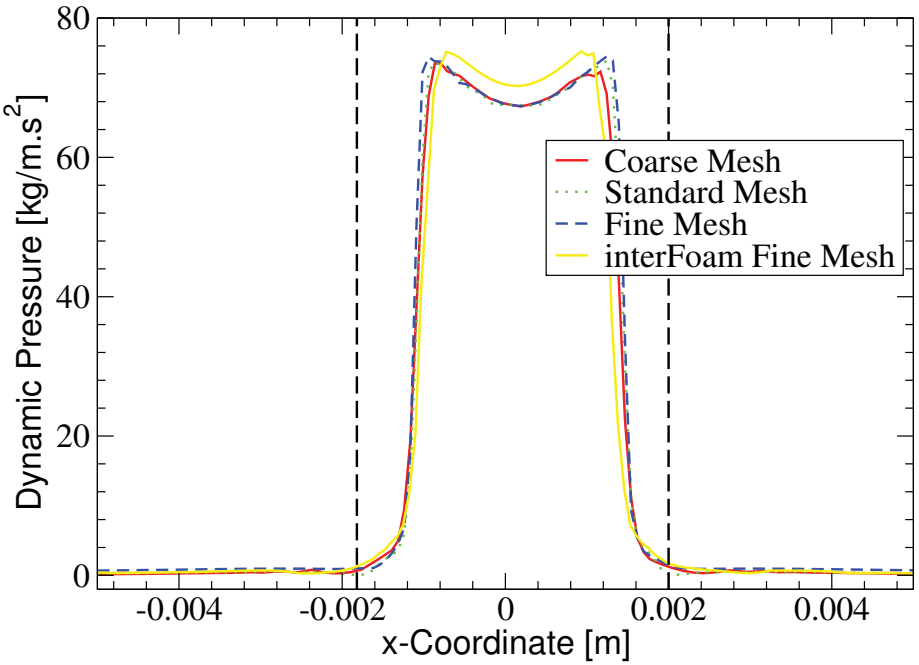

Figure 3.12: Pressure at steady-state along line $y = 0$ using interDyMFoam for the 3D drop in shear flow test case $Ca = 0.3$. The vertical lines indicate the boundary of the drop along $y = 0$.

Table 3.4
CPU time and cell size around the interface using interFoam and interDyMFoam for the 3D drop in a shear flow test case ($Ca = 0.3$)

Solver	CPU Time (s)	Number of cells	Cell size around the interface(mm)
interFoam(Coarse)	8773	78125	0.24× 0.24×0.24
interFoam(Standard)	23423	179685	0.18× 0.18×0.18
interFoam(Fine)	62367	403082	0.14× 0.14×0.14
interFoam(non-uniform)	45232	179685	0.09× 0.18×0.18
interDyMFoam(Coarse)	3151	17100 – 27000	0.10× 0.10×0.10
interDyMFoam(Standard)	8671	40000 – 53000	0.075× 0.075×0.075
interDyMFoam(Fine)	23254	89232 – 115000	0.057× 0.057×0.057

Table 3.5
interDyMFoam in parallel for the 3D drop in a shear flow test case
($Ca = 0.3$)

Solver	CPU Time (s)	Initial Mesh	Cell size around the interface(mm)
interDyMFoam (8 1 1)	3059	100×20×20	0.075× 0.075×0.075
interDyMFoam (4 2 1)	2802	100×20×20	0.075× 0.075×0.075
interDyMFoam (2 2 2)	2917	100×20×20	0.075× 0.075×0.075
interDyMFoam (Scotch)	3068	100×20×20	0.075× 0.075×0.075
interDyMFoam(Serial)	8671	100× 20×20	0.075× 0.075×0.075

3.4.3 Effect of Parallelization on Efficiency of interDyMFoam

In general, to run a case in parallel, the `decomposePar` has to be run before running the solver directly. When running the `decomposePar`, it chooses the number of processors and the method of how to split the computational domain. For more information, refer to the `decomposeParDict` file, located in the system folder of the case. Table 3.5 summarizes the CPU time and the cell size around the interface using `interDyMFoam` solver in parallel on 8 processors and in serial on the standard mesh. There are four different parallel cases using two different methods. In the first three cases, the `simple` method is used, whereas, in the fourth case the `scotch` method is used. In the simple method, the number of sub-domains in each direction must be specified. For example, (4 2 1) means that the domain is divided into four sub-domains in x direction, two subdomains in y direction, and one subdomain in z direction. In the scotch method, the solver chooses how to split the domain in the best way to minimize the number of processor boundaries. As shown in the table in our case, the best method to use is the simple method with directions (4

2 1) since the CPU time is lowest. This method divides the domain around the interface into four small subdomains. Thus, the refined regions of the domain can be distributed onto more processors while keeping the number of cells on the processor boundaries acceptable. In general, if the interface is unknown, the scotch method will be the best option to use. By comparing between the serial and parallel calculations, the CPU time increases by a factor ranging $2.8 - 3.1$ when the serial calculation is used. Therefore, using the parallel calculation will be a good option to reduce the CPU time and get the same solution.

3.5 Modifications to the interDyMFoam Solver

In this section, we will describe the modifications done for `interDyMFoam` to allow dynamic mesh refinement in 2D simulations. The 2D `interDyMFoam` uses the same functions that the 3D `interDyMFoam` uses. However, a cell is divided into four cells instead of eight cells as the case in 3D. For the selected cells to be refined, the dynamic mesh refinement in 2D simulations works as follows:

1. A point is added to the center of each face that belongs to empty patches as shown in Figure 3.13. In 2D, no point should be added to cell centroid because in one direction the number of cells must remain one. Furthermore, no point is added to the center of other faces for the same reason.

2. A point is added to the center of each edge that lies on the empty patches as shown in Figure 3.13. The points should not be added to the other edges since this is a 2D simulation.

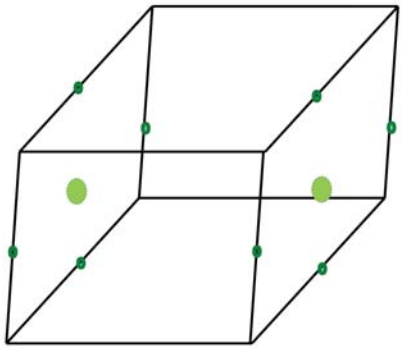

Figure 3.13: The points in the center of the faces and edges

3. Each face that has a point added to its center is divided into four new faces as shown in Figure 3.14. Each new face is assigned a new owner and neighbor cell. Therefore, a face of an adjacent unrefined cell has four neighboring cells.

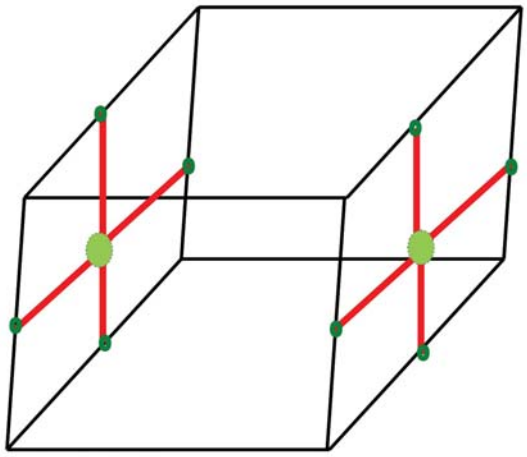

Figure 3.14: The faces are divided into four new faces

4. Each face that does not have a point added to its center is divided into two new faces as shown in Figure 3.15.

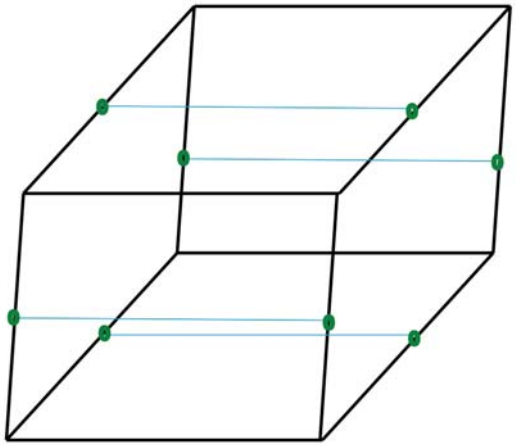

Figure 3.15: The faces are divided into two new faces

5. Finally, four new internal faces are added to the cell by connecting the points that were added to the center of the faces and the points that were added to the center of the edges as shown in Figure 3.16.

The rest of the process is similar to the `interDyMFoam` solver. The `interDyMFoam` for axisymmetric simulations works similarly as the 2D `interDyMFoam` by replacing the empty patches with wedge patches. However, the cells on the center line should be treated separately since they are wedges instead of hexahedral. Those cells have faces that contains three vertices instead of four. Such triangular faces are divided into two faces, one

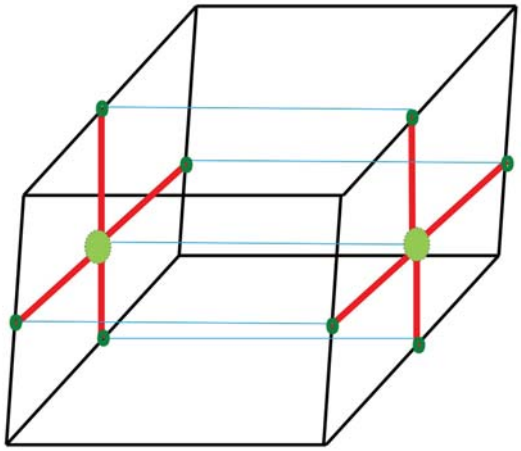

Figure 3.16: Internal faces are added to the cell

containing four vertices and the other containing three vertices. Also, we should add an internal face that contains three vertices, where one of the vertices is the middle point of the center line edge. The partition of a wedge is illustrated in Figure 3.17.

In these solvers, we added a new values to the `dynamicMeshDict` file, `axis`, `axisVal`, and `nBufferLayersR` (see Figure 3.18). The `axis` and `axisVal` are the numbers that controls the points which can be unrefined. If the empty or wedge faces are perpendicular on the x-axis, then the `axis` number should be 0 and the `axisVal` number should be between the minimum and maximum values of the x-component in the geometry. If the empty or wedge faces are perpendicular on the y-axis, then the `axis` number should be 1 and the `axisVal` number should be between the minimum and maximum values of the y-component in the geometry. If the empty or wedge faces are perpendicular on the z-axis, then the `axis` number should be 2 and the `axisVal` number should be between the minimum and maximum values of the z-component in the geometry. The `nBufferLayersR` value works the same as `nBufferLayers`, which

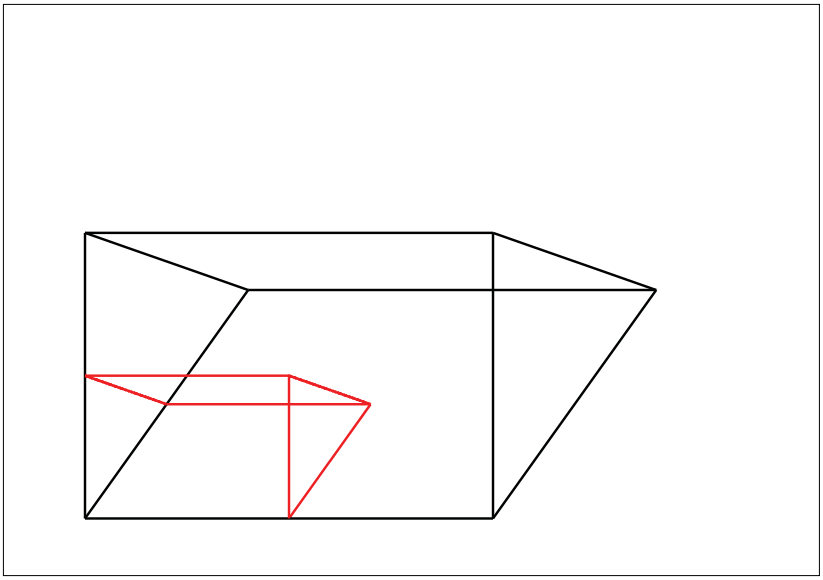

Figure 3.17: A cell with divided face and two internal faces added in axisymmetric case

find the number of buffer layers that should be extended for the unrefinement, but for the refinement instead of unrefinement. This value was added to extend the number of buffer layers around the interface during the refinement step, since it was found that additional resolution is often needed in this step, particularly when tracking small drops in a flow field. If the resolution around the interface is extended, then the mass will be conserved for a small droplet in a large geometry, whereas, without extending the resolution, the droplet loses mass and may even vanish. The modifications are made via libraries, specifically, the `dynamicRefineFvMesh` and `polyTopoChange` libraries. The modifications are further explained in the Appendix B. To validate the modifications, three cases are presented in the next sections.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMesh   dynamicRefineFvMesh2D;

dynamicRefineFvMesh2DCoeffs
{
    // How often to refine
    refineInterval  1;
    // Field to be refinement on
    field           alpha1;
    axis             2;  //AB
    axisVal          0;  //AB
    // Refine field inbetween lower..upper
    lowerRefineLevel 0.1;
    upperRefineLevel 0.9;
    // If value < unrefineLevel unrefine
    unrefineLevel   10;
    nBufferLayersR   1;  //AB
    // Have slower than 2:1 refinement
    nBufferLayers   1;
    // Refine cells only up to maxRefinement levels
    maxRefinement   4;
    // Stop refinement if maxCells reached
    maxCells        200000;
    // Flux field and corresponding velocity field. Fluxes on changed
    // faces get recalculated by interpolating the velocity. Use 'none'
    // on surfaceScalarFields that do not need to be reinterpolated.
    correctFluxes
    (
        (phi Urel)
        (phiAbs U)
        (phiAbs_0 U_0)
        (nHatf none)
        (rho*phi none)
        (ghf none)
    );
    // Write the refinement level as a volScalarField
    dumpLevel       true;
}


// ************************************************************************* //
```

Figure 3.18: Example of a dynamicMeshDict for 2D simulations

3.6 Test of interFoam and interDyMFoam in 2D Planar Geometry

In order to validate the modifications of the code for 2D planar simulations, the solutions from the modified code are compared with the solutions from the `interFoam` solver for two test problems: drop deformation and breakup in simple shear flow, and drop formation and detachment from a micro T-channel.

3.6.1 Drop Deformation and Break Up in Simple Shear Flow

In this section, we test `interDyMFoam` in 2D planar geometry for the test case of a two-dimensional liquid drop in a simple shear flow. This is the two-dimensional version of the 3D test case considered in Section 3.4. The computational domain is a channel of length 30 mm and height 6 mm. The origin of the coordinates system is placed in the center of the domain, such that $-15 \leq x \leq 15$ and $-3 \leq y \leq 3$. The drop is a circle of radius 1 mm and center $(0,0)$ (see Figure 3.7). The boundary conditions are shown in Table 3.6. The fluids are taken to be Newtonian and the material properties are $\mu_c = \mu_d = 1.06 \times 10^{-1} Pa.s$, $\rho_c = \rho_d = 10^3 kg/m^3$, and $\sigma = 0.0415 N/m$. The coarse and fine meshes are produced by decreasing and increasing the number of cells in the standard mesh by a factor of 1.5 in each direction respectively (see Table 3.7). We first consider the case of $Ca = 0.3$, and then

the case of $Ca = 0.4$.

Table 3.6
Boundary conditions for the 2D drop in a shear flow test case where Ca is the capillary number

boundary	velocity	$p\_\rho gh$	alpha (α)
inlet	zeroGradient	zeroGradient	zeroGradient
outlet	zeroGradient	0	zeroGradient
bottom	(-1.175$(m/s)*Ca$ 0 0)	zeroGradient	zeroGradient
top	(1.175$(m/s)*Ca$ 0 0)	zeroGradient	zeroGradient
front and back	empty	empty	empty

Table 3.7
Initial mesh and number of cells for the 2D drop in a shear flow test case

Solver	Initial Mesh	Number of Cells
interFoam (Coarse)	300×60	18000
interFoam (Standard)	450×90	40500
interFoam (Fine)	675×135	91125
interDyMFoam (Coarse)	80×16	$1280-2100$
interDyMFoam (Standard)	120×24	$2880-3900$
interDyMFoam (Fine)	180×36	$6480-8000$

3.6.1.1 Test Case Using $Ca = 0.3$

For $Ca = 0.3$, the drop reaches a stationary shape after a while and no break up occurs. Figures 3.19 and 3.20 show the refinement around the interface. Figure 3.19 illustrates the coarsest mesh (top figure), the refinement in the x, y directions at $t = 0.005$ s (middle figure), and the refinement in the x, y directions at steady-state $t = 0.99$ s (bottom figure). Figure 3.20 shows the number of cells in the z-direction equal one. Note that

the refinements are shown as diagonals, although the cells are actually partitioned into rectangles.

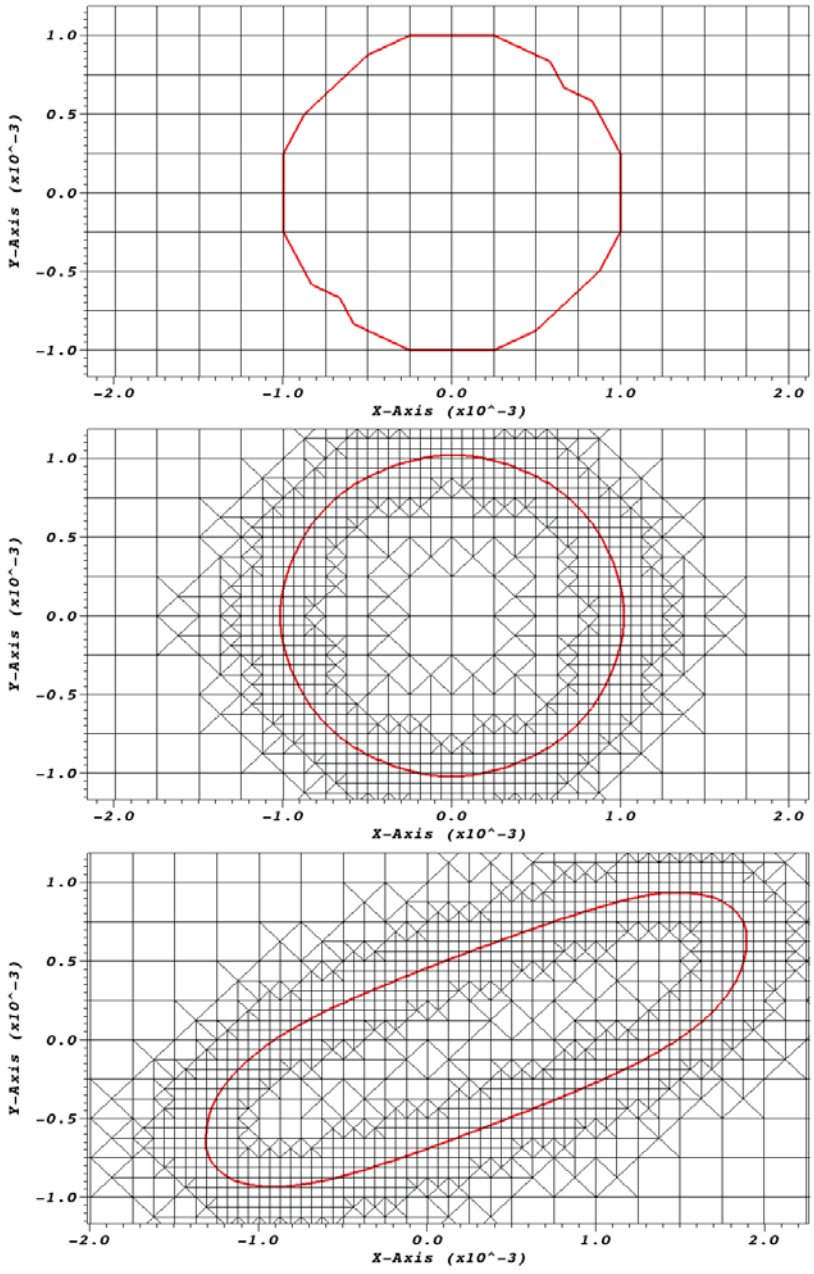

Figure 3.19: Dynamic refinement in 2D at $t = 0$ s (top), $t = 0.005$ s (middle), and $t = 0.99$ s (bottom) for the 2D drop in a shear flow test case

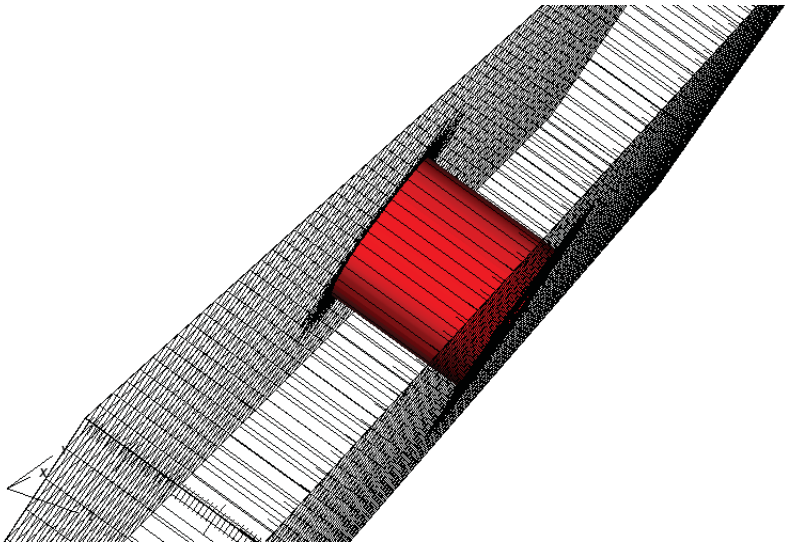

Figure 3.20: Dynamic refinement in 2D at $t = 0.005$ s for the 2D drop in a shear flow test case

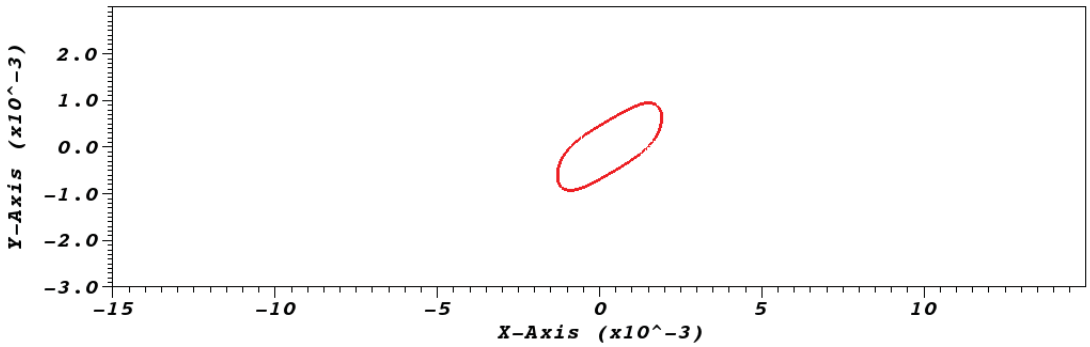

Figure 3.21: Drop at steady-state $t = 0.99$ s for the 2D drop in a shear flow test case

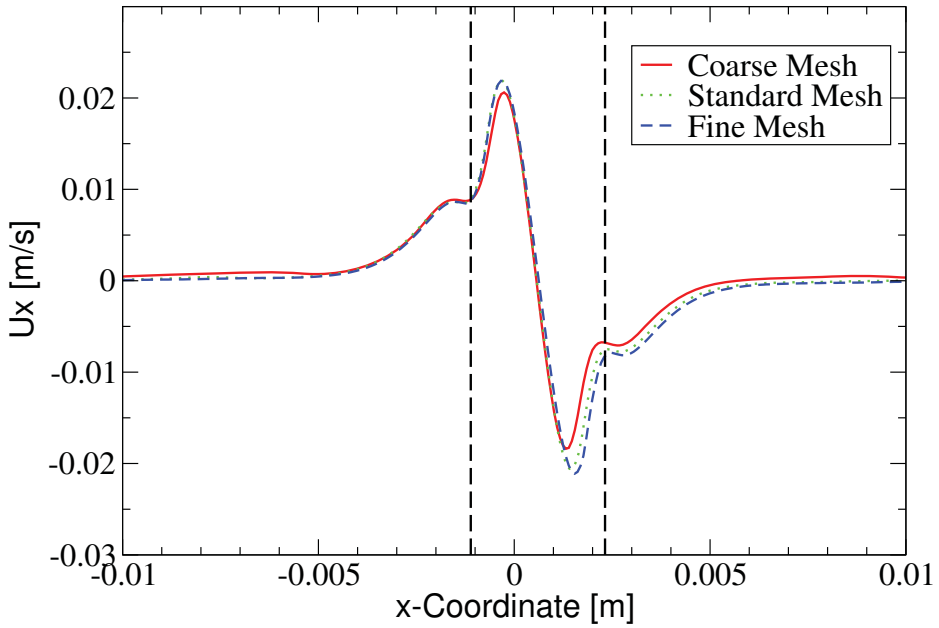

Figure 3.22: Velocity using interFoam on three different meshes $Ca = 0.3$ for the 2D drop in a shear flow test case

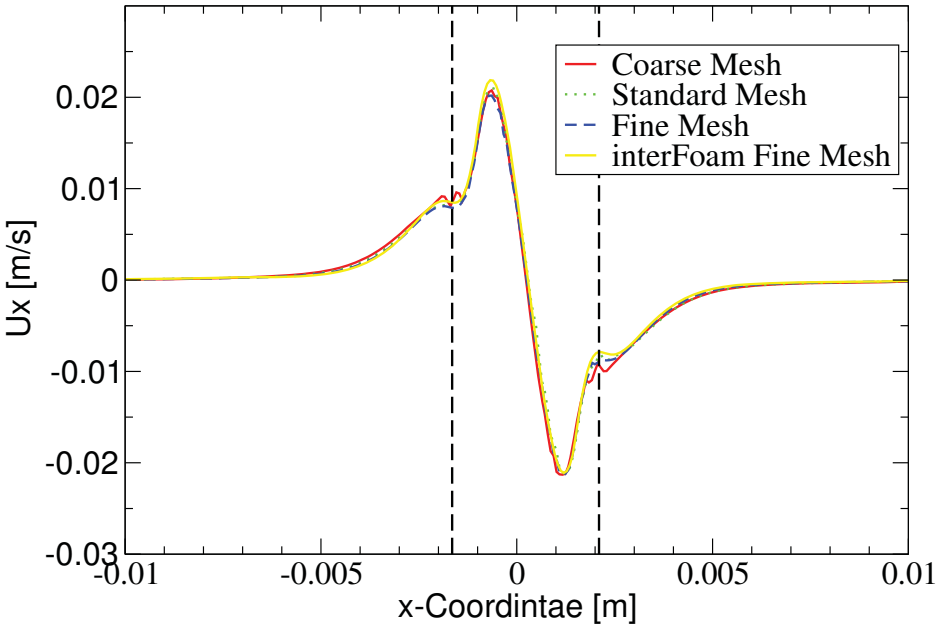

Figure 3.23: Velocity using interDyMFoam on three different meshes $Ca = 0.3$ for the 2D drop in a shear flow test case

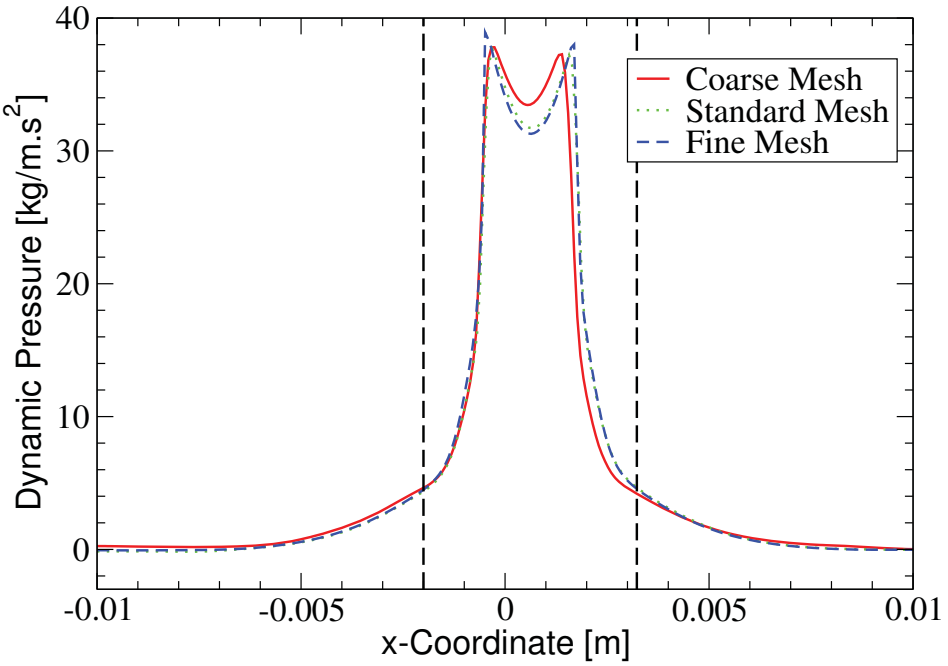

Figure 3.24: Pressure using interFoam on three different meshes $Ca = 0.3$ for the 2D drop in a shear flow test case

The velocity and pressure from `interFoam` and `interDyMFoam` for this 2D problem are compared along the horizontal line $y = 0$ at time $t = 0.99$ s, when a stationary drop shape is reached (see Figure 3.21). Figure 3.22 shows the velocity graphs for the same case with three different meshes: coarse, standard, and fine mesh using the `interFoam` solver. The vertical lines indicate the boundary of the drop. The solutions are almost identical outside the droplet and behave similarly inside the droplet. Also, the graphs show a zero velocity outside the droplet which is expected since the top and bottom walls are moving at the same speed in the x-component with opposite directions. Inside the droplet, the velocity increased and then decreased. Similarly, in Figure 3.23, the same behavior is found using the `interDyMFoam` solver. The velocities agree with the `interFoam` solution on the refined mesh. Similarly, the pressure graphs, Figures 3.24 and 3.25, show

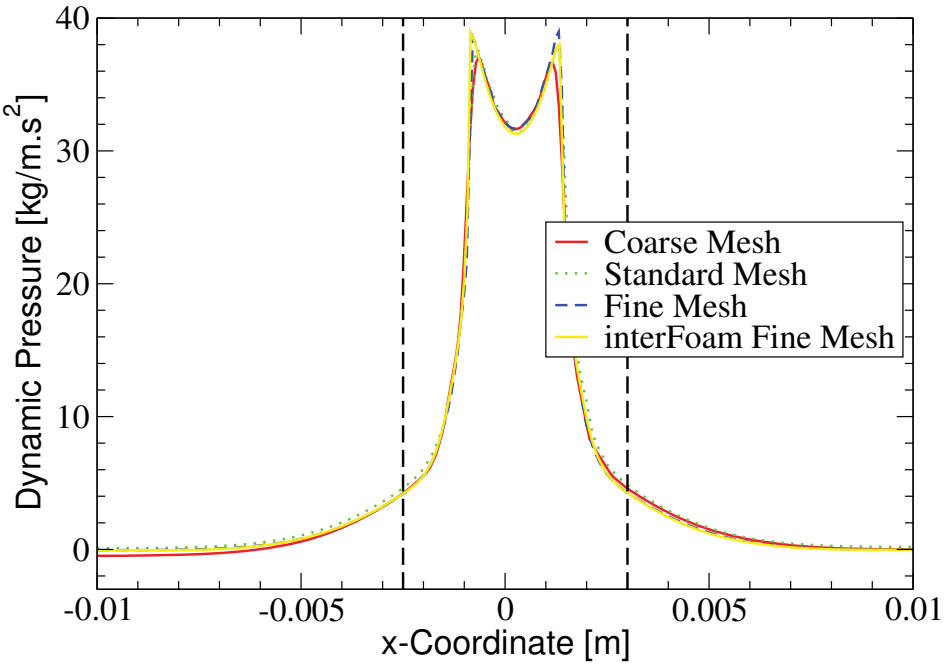

Figure 3.25: Pressure using interDyMFoam on three different meshes $Ca = 0.3$ for the 2D drop in a shear flow test case

the same convergence. They have the same behavior overall, as the jump in pressure is decreasing by increasing the mesh resolution using `interFoam` solver. However, the solutions from `interDyMFoam` solver are accurate even on the coarse mesh. Table 3.8

Table 3.8
CPU time, cell size around the interface, and relative change in radius for the 2D drop in a shear flow test case ($Ca = 0.3$)

Solver	Cell CPU Time (s)	Initial Mesh	Cell size around the interface(mm)	relative change in R
interFoam (Coarse)	1164	300×60	0.1 × 0.1	0.034
interFoam (Standard)	6409	450×90	0.067 × 0.067	0.008
interFoam (Fine)	21611	675×135	0.044× 0.044	0.002
interDyMFoam (Coarse)	107	80×16	0.09× 0.09	0.030
interDyMFoam (Standard)	355	120×24	0.063× 0.063	0.008
interDyMFoam (Fine)	918	180×36	0.042× 0.042	0.003

shows the difference in CPU time, number of cells, cell size around the interface, and

the relative change in radius. Here, the radius is calculated by considering the drop as a circle using the $\alpha = 0.05$ contour, so if the volume fraction in a cell less than 0.05, then the cell does not contribute to the calculation of the radius. The `interDyMFoam` has a much lower overall CPU time, even when comparing its fine mesh with the `interFoam`'s coarse mesh as the CPU time increased by a factor of 1.2, keeping in mind that the cell size around the interface is almost the same between the two solvers. The relative change in the radius is calculated using the formula $\frac{R_0 - R_{0.99}}{R_0}$, where R_0 is the radius at $t = 0$ s and $R_{0.99}$ is the radius at $t = 0.99$ s. The relative change decreases with decreasing cell size around the interface, however it is almost identical in comparison between the two solvers. Furthermore, the numbers are small in all cases especially in the fine mesh which indicates that the mass inaccuracy due to the mesh is negligible. As a result, the two solvers give similar results with a big difference in the CPU time.

3.6.1.2 Test Case Using $Ca = 0.4$

This section compares the two solvers on the 2D drop in shear flow for $Ca = 0.4$. This is a super-critical Ca, where break up occurs. In the previous case using $Ca = 0.3$, the break up does not occur. In this case, however, the droplet breaks up into two daughter droplets using both solvers, `interFoam` and `interDyMFoam`. Figure 3.26 shows the daughter droplets using `interFoam` (top) and `interDyMFoam` (bottom). The dynamic refinement in the x, y directions is shown in Figure 3.27. In the figure, the refinement appears around the

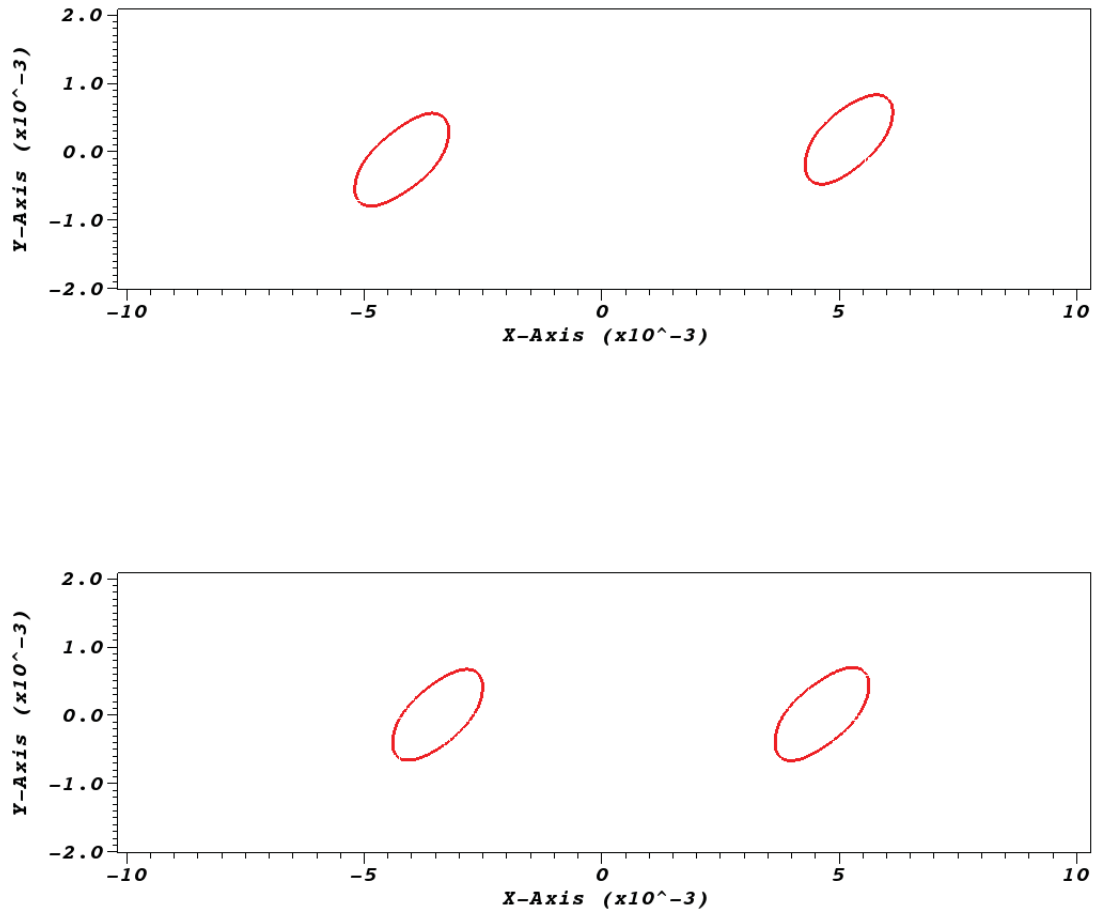

Figure 3.26: Drop breakup in 2D using $Ca = 0.4$ at $t = 0.99$ s using interFoam (top) and interDyMFoam (bottom) for the 2D drop in a shear flow test case

interface and there is no refinement between the droplets. That is, after the initial droplet break up and the droplets move apart, the unrefinement occurs for the refined cells far from the interfaces. The number of cells in the z-direction is one as shown in Figure 3.28. Table 3.9 summarizes the CPU time, cell size around the interface, the initial radius R_0, the radius of the first daughter droplet R_1, the radius of the second daughter droplet R_2, the

Table 3.9
CPU time, cell size around the interface, and relative change in radius for the 2D drop in a shear flow test case ($Ca = 0.4$)

Solver	CPU Time (s)	Cell size around the interface(mm)	$R_0(mm)$	$R_1(mm)$	$R_2(mm)$	$\frac{R_0^2-(R_1^2+R_2^2)}{R_0^2}$	break up time (s)
interFoam	1578	0.1×0.1	1.003	0.667	0.688	0.09	0.115
interFoam	6811	0.067×0.067	1.006	0.715	0.688	0.03	0.21
interFoam	26068	0.044×0.044	1.002	0.703	0.708	0.008	0.36
interDyMFoam	242	0.09×0.09	1.036	0.695	0.694	0.1	0.15
interDyMFoam	661	0.063×0.063	1.017	0.699	0.719	0.03	0.225
interDyMFoam	1438	0.042×0.042	1.030	0.722	0.728	0.008	0.43

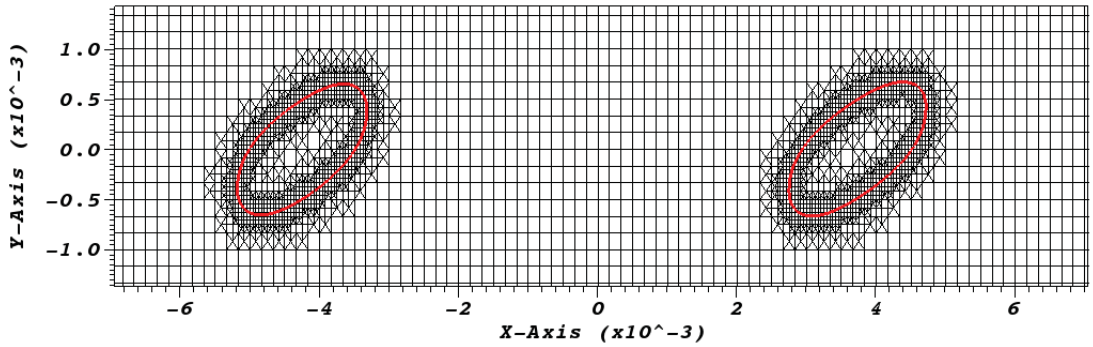

Figure 3.27: Dynamic refinement in 2D with $Ca = 0.4$ at $t = 0.99$ s for the 2D drop in a shear flow test case

relative change in mass, and the breakup time. The efficiency of `interDyMFoam` relative to `interFoam` increases with mesh refinement, i.e. with decreased cell size around the interface. For the three `interDyMFoam` meshes, the CPU time for `interDyMFoam` decreases by a factor of 6.5, 10.3, and 18.1 relative to the corresponding `interFoam` mesh. The droplet is broken into two droplets, thus a refinement is needed around the interface for each droplet, (see Figures 3.27 and 3.28). The radius of each droplet depends

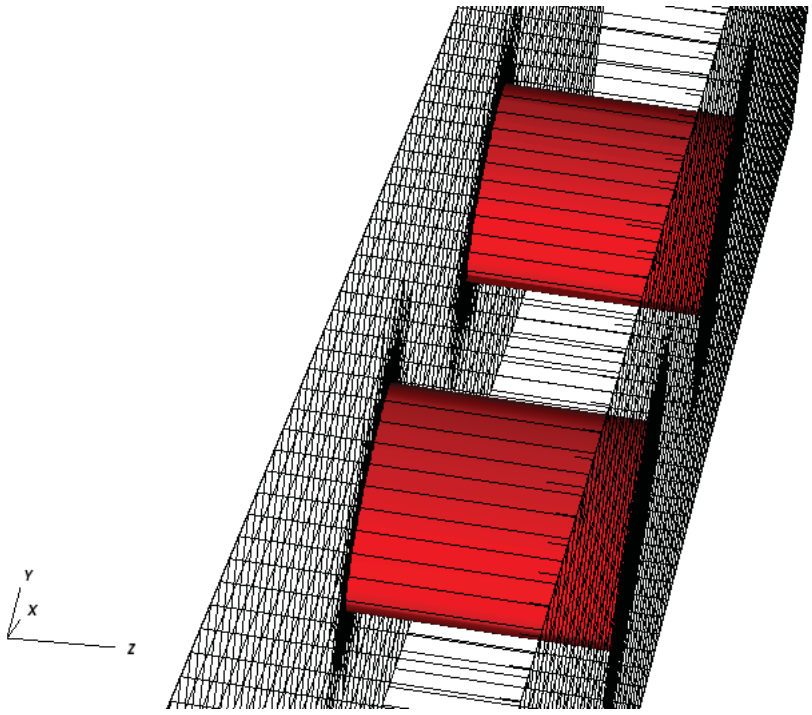

Figure 3.28: Dynamic refinement in 2D with $Ca = 0.4$ at $t = 0.99$ s for the 2D drop in a shear flow test case

of the cell size around the interface and the initial radius of the droplet. The initial radius depends on the initial mesh, especially for the `interDyMFoam` solver since the solver starts with a coarse mesh compared to `interFoam` solver. As expected, each droplet is broken into two daughter droplets with almost identical radius particularly on the fine mesh. In principle, due to mass conservation, the area of the initial droplet must equal the sum of the areas of the daughter drops. Mathematically, this can be written as

$$\pi R_0^2 = \pi R_1^2 + \pi R_2^2, \tag{3.15}$$

where R_0 is the radius of the initial drop. From Eq. (3.15), we get $R_0^2 = R_1^2 + R_2^2$. Therefore, the relative change in R can be calculated using the formula $\frac{R_0^2 - (R_1^2 + R_2^2)}{R_0^2}$. In the table, the

relative change is small for all cases especially on the refined mesh. Also, the relative change is almost the same if we compare between the solvers. The break up time is dependent on the mesh resolution as the droplet breaks up on the coarse mesh faster than on the finer mesh. Physically, the breakup time should not differ. However, in the simulations, breakup time depends on the cell size, thus the difference in breakup time using different meshes.

In conclusion, the modified `interDyMFoam` solver performs well compared with `interFoam` on this 2D planar test problem. It produces similar results at much lower CPU times.

3.6.2 A Drop Detachment From a Micro T-channel

In this section, the modified `interDyMFoam` is evaluated for the problem of a 2D micro T-channel. In particular, the performance of the modified code is compared with `interFoam`, and the effect of the parameters in `dynamicMeshDict` is investigated. The micro T-channel flow problem consists of a disperse phase fluid which is transported through a pore and into a gap containing a continuous phase fluid. Drops of the disperse phase are then detached by the shear flow field of the continuous phase. The domain used for the pore is a channel of length 25 micrometers and height of 200 micrometers. The domain used for the gap is a channel of length 2250 micrometers and height of 500 micrometers The origin of the coordinate system is placed such that $-250 \leq x \leq 2000$

and $0 \leq y \leq 500$ for the gap and $-25 \leq x \leq 25$, $-200 \leq y \leq 0$ for the pore. The geometry is divided into four blocks as shown in Figure 3.29. Table 3.10 summarizes the number of cells in each block for the three meshes in both solvers. The coarse and fine meshes are obtained from the standard mesh by decreasing and increasing the number of cells in each direction by a factor of 1.4, respectively. Table 3.11 shows the boundary conditions used in the simulations. The `fixedFluxPressure` boundary means the pressure gradient is adjusted such that the flux is specified using the velocity boundary condition. The balancing of interfacial tension forces σ_{ls} (liquid-solid), σ_{fs} (fluid-solid), and σ_{lf} (liquid-fluid) produces the equilibrium or static contact angle. The balance of these surface tension can express in Young's relation [51], $\sigma_{fs} - \sigma_{ls} - \sigma_{lf}\cos(\theta) = 0$, where θ is the angle between the tangent line of the liquid at the triple point, where the three phases meet, and the solid from the liquid side. Wetting is the ability of liquid to maintain contact with a solid surface, resulting from intermolecular interaction when the two are brought together. Contact angle $\theta = 0$ is a perfectly wetting case and $\theta = 180^o$ is a perfect non-wetting case. If the contact angle is larger than 90^o then the surface is non-wetting. On the other hand, if the angle is below than 90^o the material is wetting the surface. In the simulations, a non-wetting behavior is assumed by taking the static contact angle to be 180^o. The end time for all cases is 0.1 s. The continuous and disperse phases are taken to be Newtonian fluids. The velocity set to be $0.3m/s$ directed to the positive x-axis on the inlet of the shear flow channel and about 0.01 m/s directed to the positive y-axis on the inlet of the pore. The transport properties are as follows:

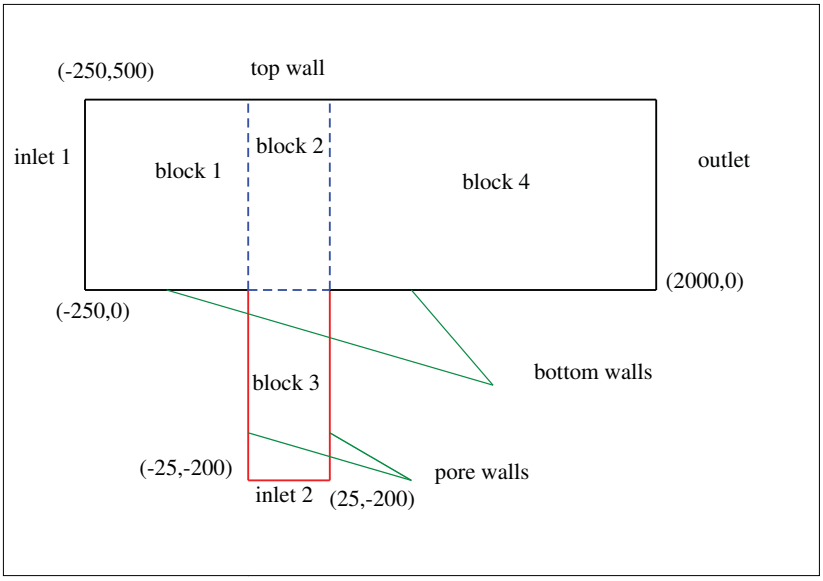

Figure 3.29: Geometry sketch for a drop detachment from a micro T-channel test case where the units are in micrometers

$\mu_c = 1.056 \times 10^{-1} Pa.s$

$\mu_d = 1 \times 10^{-3} Pa.s$

$\rho_c = 960 kg/m^3$

$\rho_d = 10^3 kg/m^3$

$\sigma = 0.0415 kg/s^2$

Table 3.10
Number of cells in each block for a drop detachment from a micro T-channel test case

Solver	block 1	block 2	block 3	block 4	number of cells
interFoam (Coarse)	56×125	13×125	13×50	494×125	71081
interFoam (Standard)	78×175	18×175	18×70	692×175	139160
interFoam (Fine)	109×245	25×245	25×98	969×245	272685
interDyMFoam (Coarse)	19×42	5×42	5×17	165×42	8023
interDyMFoam (Standard)	27×59	7×59	7×24	231×59	15803
interDyMFoam (Fine)	38×82	10×82	10×33	323×82	30752

Table 3.11
Boundary conditions for a drop detachment from a micro T-channel test case

boundary	velocity	$p\_\rho gh$	alpha1
inlet 1	$(0.3,0,0)$	zeroGradient	zero
inlet 2	$(0,0.011,0)$	zeroGradient	1
outlet	zeroGradient	0	zeroGradient
bottom walls	$(0,0,0)$	fixedFluxPressure	constantAlphaContactAngle(180^o)
top wall	$(0,0,0)$	fixedFluxPressure	constantAlphaContactAngle(180^o)
pore wall	$(0,0,0)$	fixedFluxPressure	constantAlphaContactAngle(180^o)
front and back	empty	empty	empty

where μ is the dynamic viscosity, ρ is the density, σ is the interfacial tension, and the subscripts c and d stands for continuous and disperse phases respectively. The capillary number $Ca = \frac{\mu_c u}{\sigma}$, where u is the characteristic velocity, is about 0.76 and viscosity ratio $\lambda = \frac{\mu_d}{\mu_c}$ is about 0.0095. The Reynolds number is the ratio of inertial forces to viscous forces and mathematically defined as $Re = \frac{\rho u L}{\mu}$, where L is the characteristic length and u is the characteristic velocity. The Reynolds number for the continuous phase Re_c is about 1.37 and for the disperse phase Re_d is about 0.55. Figures 3.30 and 3.31 show the first drop detachment using `interDyMFoam` and `interFoam` solver, respectively. In the simulations, several droplets are produced of nearly uniform size. The drop size reported in the following is the average of those sizes. In both cases, the radius of the detached drops are larger than the pore radius. The coarsest mesh and refinement around the interface after first drop detach are show in Figure 3.32.

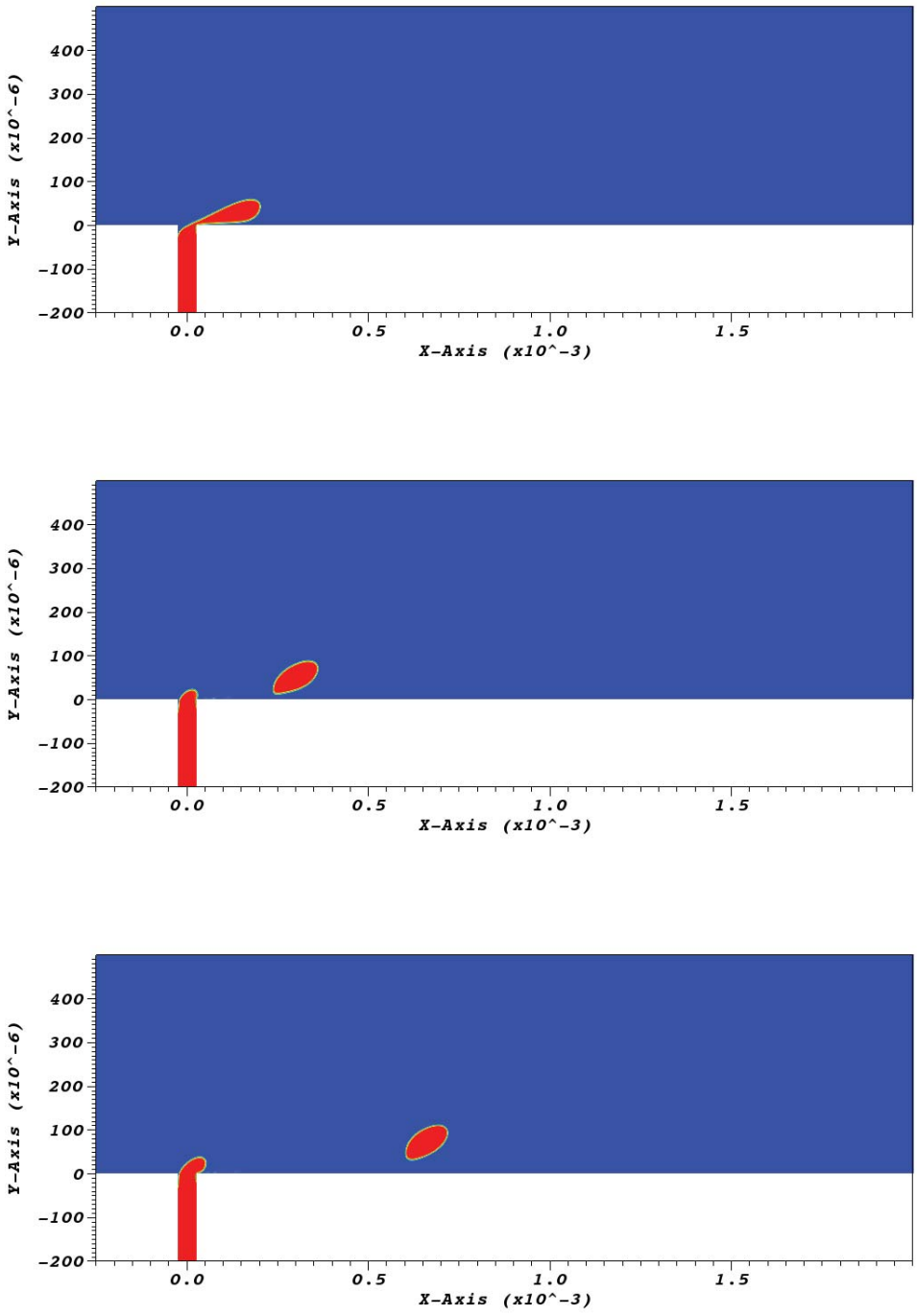

Figure 3.30: Drop deformation and detachment at $t = 0.01, 0.012$, and 0.014 s using interDyMFoam for a drop detachment from a micro T-channel test case

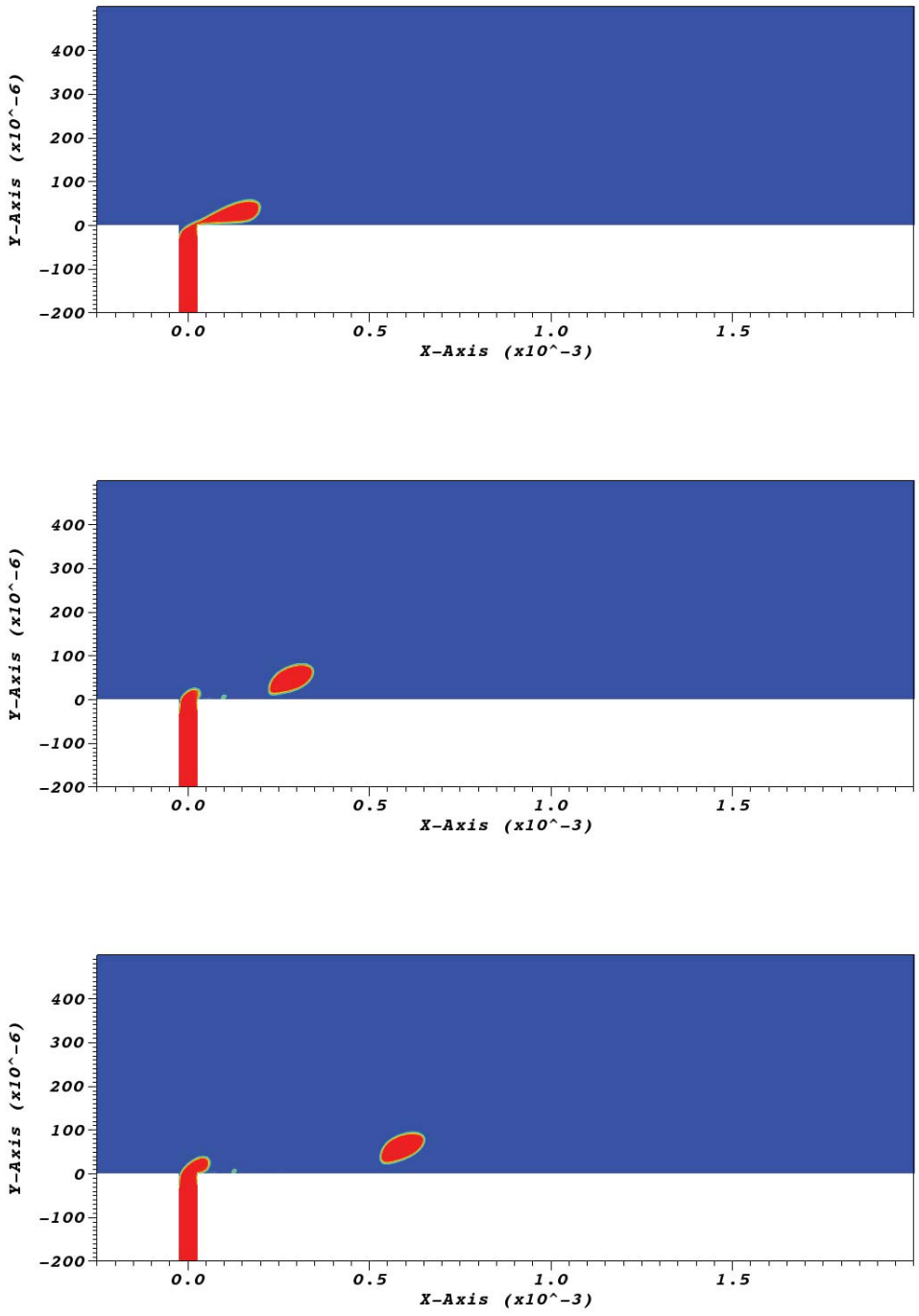

Figure 3.31: Drop deformation and detachment at $t = 0.01, 0.012,$ and 0.014 s using interFoam for a drop detachment from a micro T-channel test case

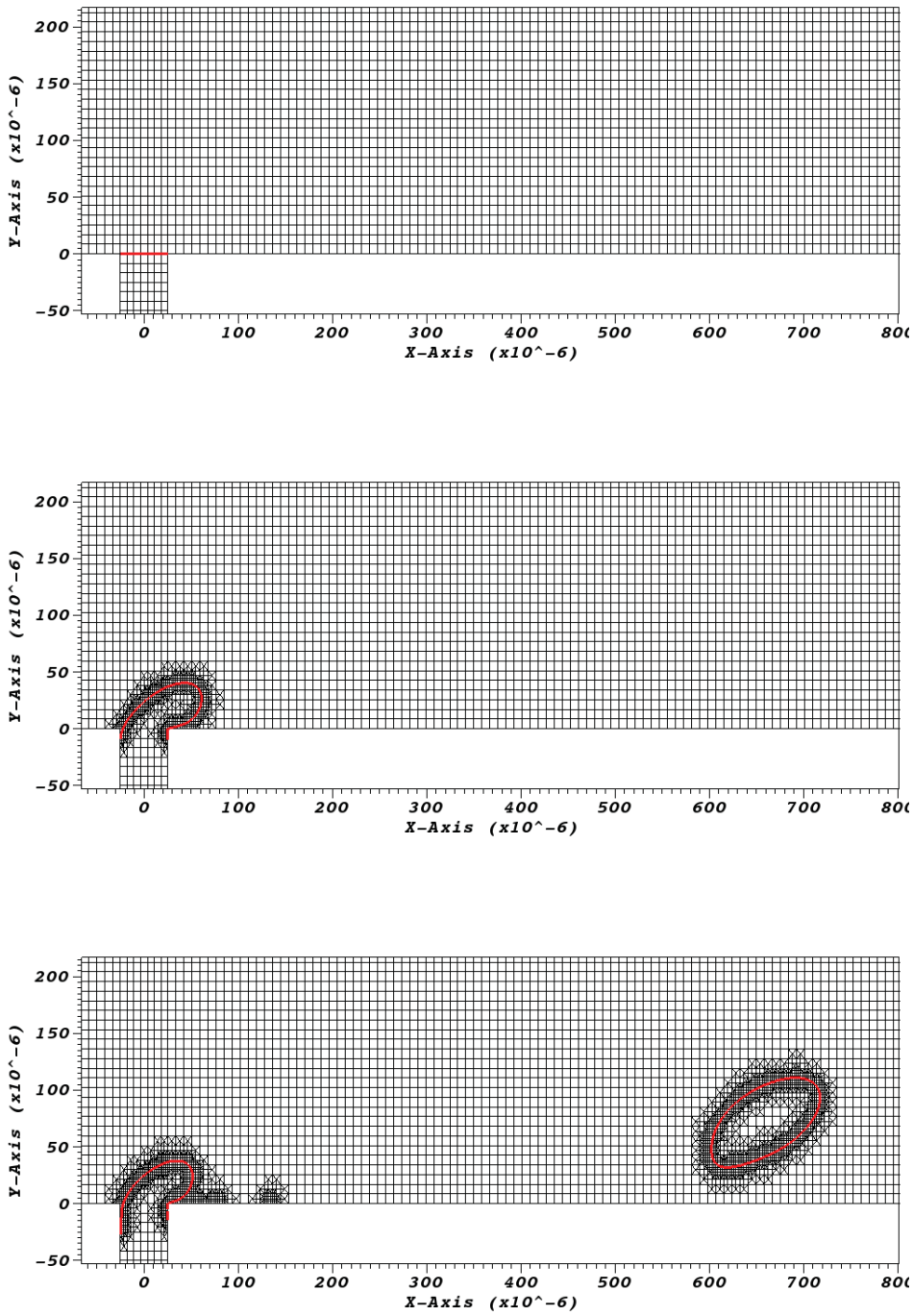

Figure 3.32: Dynamic refinement at $t = 0, 0.004$, and 0.014 s for a drop detachment from a micro T-channel test case

3.6.2.1 Mesh Independence Study

This section compare the predictions of `interFoam` and the modified `interDyMFoam`, each on three different meshes. In particular, we compare the pressure curves and drop sizes. Figure 3.33 shows the pressure curves along a horizontal line through the center of a detached droplet for three different meshes using `interFoam`. The graphs are almost identical outside the droplet. The graphs indicate mesh independence for the standard and fine meshes. The same behavior is noted in Figure 3.34, where the `interDyMFoam` solver is used. The `interDyMFoam` appears to perform better than `interFoam` for this case because the pressure does not undershoot close to the interface as observed in the `interFoam` solution.

Next, we compare the detached drop size and the CPU time for the two solvers on the

Table 3.12
CPU time and ratio of a drop radius to the pore radius using maximum refinement 1 for a drop detachment from a micro T-channel test case

Solver	CPUTime (s)	R/PR	Relative Change
interFoam (Coarse)	9056	1.54826	
interFoam (Standard)	41817	1.6903	0.0840312
interFoam (Fine)	90305	1.77856	0.0496268
interDyMFoam (Coarse)	3798	1.62489	
interDyMFoam (Standard)	9457	1.70793	0.0486203
interDyMFoam (Fine)	26159	1.77832	0.0395823

different meshes. This is summerized in Table 3.12. The drop size is given as the ratio of the radius of the detached drop to the pore radius. The table shows that for both solvers the

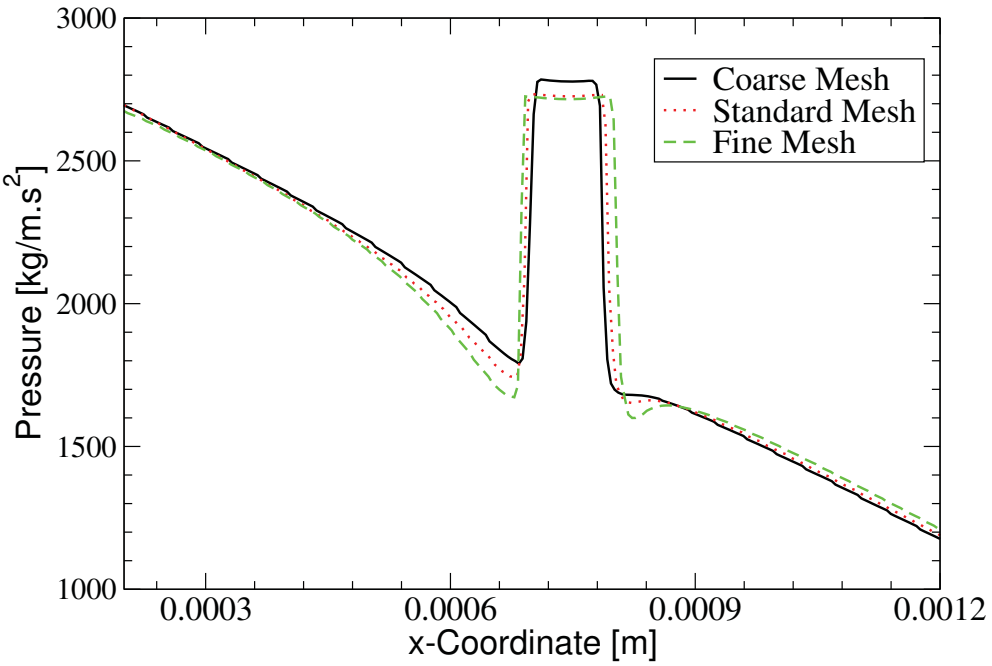

Figure 3.33: Pressure using interFoam for a drop detachment from a micro T-channel test case

drop size increases as the cell size around the interface decreases. However, the relative change in drop size decreases with mesh refinement, and is smaller in `interDyMFoam`. Moreover, the standard mesh of each solver produces nearly the same drop sizes (1.6903 vs 1.70793), as does the fine mesh of each solver (1.77856 vs 1.77832). In addition, the CPU time using `interDyMFoam` is much smaller than the CPU time using `interFoam` by a factor of 2.4 and 3.4 for the coarse and fine mesh respectively.

3.6.2.2 dynamicMeshDict Parameters Study

This section investigates the effect of the `dynamicMeshDict` parameters, `refineInterval`, `maxRefinement`, and `nBufferLayers`, on the CPU

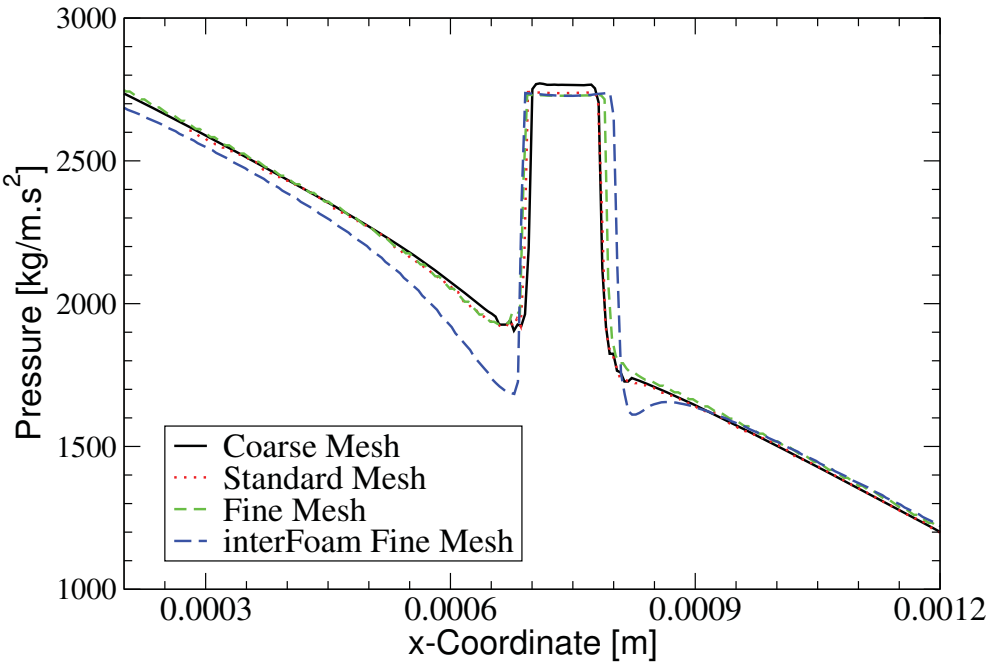

Figure 3.34: Pressure using interDyMFoam for a drop detachment from a micro T-channel test case

times and drop sizes when using the `interDyMFoam` solver. The CPU time and

Table 3.13
CPU time and Ratio of a Droplet to the radius of the Pore with Different Refine Interval Numbers using interDyMFoam solver with max. refinement equal to 3 and buffer layer equal to one for a drop detachment from a micro T-channel test case

refine interval	CPUTime (s)	CPUTime Ratio to 1	R/PR	Relative Change to 1
1	10954		1.75554	
3	5930	0.54	1.75498	0.000312
5	3718	0.34	1.75595	0.000233
7	3387	0.31	1.75047	0.002887
9	3231	0.30	1.73890	0.009481

normalized drop radius for different values of `refineInterval` is shown in Table 3.13. Recall that the refine interval specifies how often the mesh should be refined, so that `refineInterval` $= n$ means that the mesh is dynamically refined every n time steps.

Assuming the solution is most accurate using refine interval equal to one, it will be used as a standard to compare with other refine interval numbers. By comparing the refine interval equal to 1 with the other refine interval numbers, the relative change in drop size is negligible. Although there is a significant difference in the relative change in drop size between refine interval equal to 7 and 9 compared to other refine interval numbers, the relative change is still negligible. On the other hand, the time ratio is almost the same using refine interval equal to 5, 7, and 9 which is around 0.3 but it is 0.5 when using refine interval equal to 3. Thus, we suggest the refine interval to be between 5 and 9 because it saves CPU time without compromising the accuracy of the results.

In Table 3.14, the effect of the maximum refinement number on the drop size and the

Table 3.14
CPU time and Ratio of a Droplet to the radius of the Pore with Different Maximum Refinement Numbers using refine interval equal and buffer layer equal to one for a drop detachment from a micro T-channel test case

max. refinement	CPUTime (s)	CPUTime Ratio	R/PR	Relative Change
2	3798		1.62489	
3	10954	2.9	1.75554	0.074
4	33500	3.0	1.85210	0.052

CPU time is shown. Recall that the maximum refinement parameter, `maxRefinement`, indicates how many times a given cell may be refined. The table shows that the CPU time increases by a factor of 3 when the maximum refinement number is increased by 1. However, the relative change in drop size is reduced from 0.074 to 0.052, which means that the maximum refinement number has an effect on the CPU time and the drop size.

Table 3.15 outlines the effect of the number of buffer layers on the CPU time and drop

Table 3.15
CPU time and Ratio of a Droplet to the radius of the Pore with Different Number of Buffer Layers using refine interval equal to one and max. refinement equal to 2 and 4 for a drop detachment from a micro T-channel test case

Solver	CPU time (s)	R/PR
interDyMFoam (max. 2, Buf1)	3798	1.6249
interDyMFoam (max. 2, Buf3)	2888	1.6293
interDyMFoam (Max. 4, Buf1)	33500	1.8521
interDyMFoam (Max. 4, Buf3)	28260	1.8566

size for two different cases, one with maximum refinement 2 and the other with maximum refinement 4. Recall that the number of buffer layers is used to find the buffer layers that should be extended for unrefinement. Here, we take the number of buffer layers equal to 1 and 3. The table shows that for each value of `maxRefinement` (2 or 4), the CPU time decreases slightly as the number of buffer layers increases from 1 to 3, while the drop sizes remain essentially the same. Therefore, the results imply that the number of buffer layers has little or insignificant effect on the CPU time and drop size.

3.7 Test of interFoam and interDyMFoam in 2D Axisymmetric Geometry

A bubble rising in water is analyzed to test the modified `interDyMFoam` solver in axisymmetric geometry. Figure 3.35 shows the geometry used for the simulations. The computational domain is a small wedge channel open into a large wedge channel. The small channel has width of 1 mm and height of 5 mm and contains the disperse phase (air). The large channel has width of 40 mm and height of 80 mm and contains the continuous phase (water). The origin is placed such that $0 \leq x \leq 1$, $-5 \leq y \leq 0$, $-0.01 \leq z \leq 0.01$ for the small channel and $0 \leq x \leq 40$, $0 \leq y \leq 80$, $-0.4 \leq z \leq 0.4$ for the large channel. The center line of the axisymmetric geometry is placed such that $-5 \leq y \leq 80$, $x = 0$, and $z = 0$. The number of cells in each block for the all cases is outlined in Table 3.16. The coarse and fine meshes are produced by decreasing and increasing the standard mesh by a factor of 1.5, respectively . Also, the boundary conditions are summarized in Table 3.17. The end time for all cases is 0.5 s. The two fluids in the simulation are Newtonian. The transport properties are as follows:

$\mu_c = 1 \times 10^{-3} Pa.s$

$\mu_d = 1.8 \times 10^{-5} Pa.s$

$\rho_c = 1000 Kg/m^3$

$\rho_d = 1 Kg/m^3$

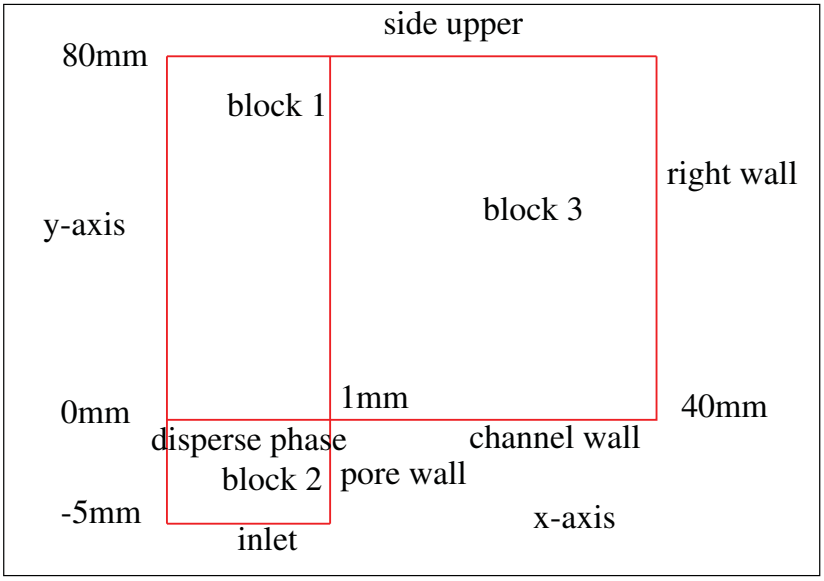

Figure 3.35: Geometry sketch of a bubble rising in a water axisymmetric case

Table 3.16
Number of cells in each block for the standard mesh of a bubble rising in a water axisymmetric case

Solver	block 1	block 2	block 3	number of cells
interFoam (Coarse)	5×400	5×25	200×400	82125
interFoam (Standard)	8×600	8×38	300×600	185104
interFoam (Fine)	12×900	12×57	450×900	416484
interDyMFoam (Coarse)	1×130	1×08	66×130	8718
interDyMFoam (Standard)	2×195	2×12	99×195	19719
interDyMFoam (Fine)	3×293	3×18	149×293	44590

$\sigma = 0.072 Kg/s^2$

where μ is the dynamic viscosity, ρ is the density, σ is the interfacial tension, and the subscripts c and d stands for continuous and disperse phases respectively. The capillary number Ca is 0.375×10^{-3}, viscosity ratio λ is 0.018, and Reynolds number Re for the disperse phase is 1.5. Figures 3.36 and 3.37 show the bubble deformation and detachment using `interDyMFoam` and `interFoam` solver, respectively. The disperse

Table 3.17
Boundary conditions of a bubble rising in a water axisymmetric test case

boundary	velocity	$p\_\rho gh$	alpha1
inlet	$(0, 0.0265258, 0)$	zeroGradient	1
side upper	zeroGradient	zeroGradient	zeroGradient
right wall	$(0, 0, 0)$	0	zeroGradient
channel wall	$(0, 0, 0)$	fixedFluxPressure	constant angle (110^o)
pore wall	$(0, 0, 0)$	zeroGradient	zeroGradient
center line	empty	empty	empty
front and back	wedge	wedge	wedge

phase produced a bubble that is larger in size than the small channel length. The refinement around the interface and unrefinement between the bubble and the inlet are shown in Figure 3.38.

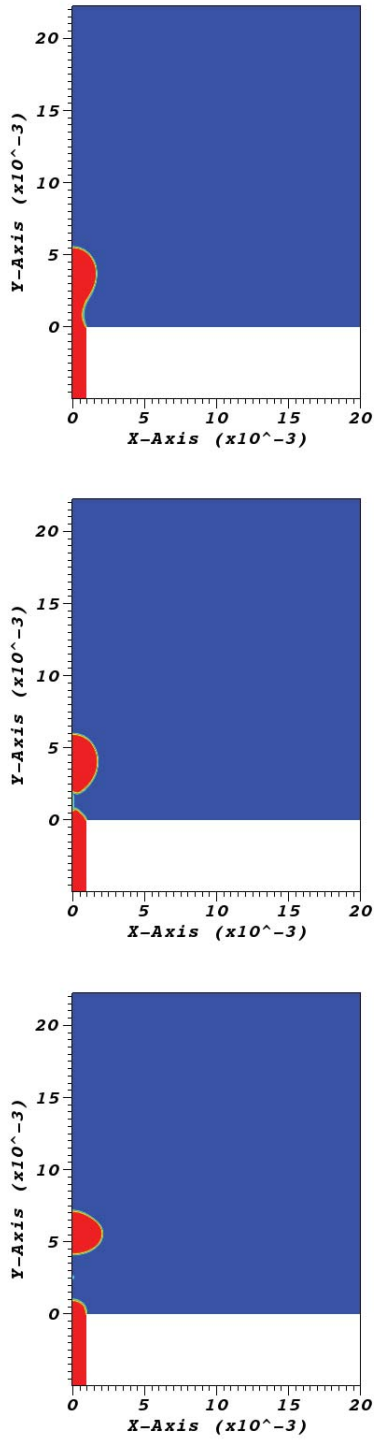

Figure 3.36: Drop deformation and detachment at $t = 0.335, 0.34$, and 0.35 s using interDyMFoam for a bubble rising in water test case

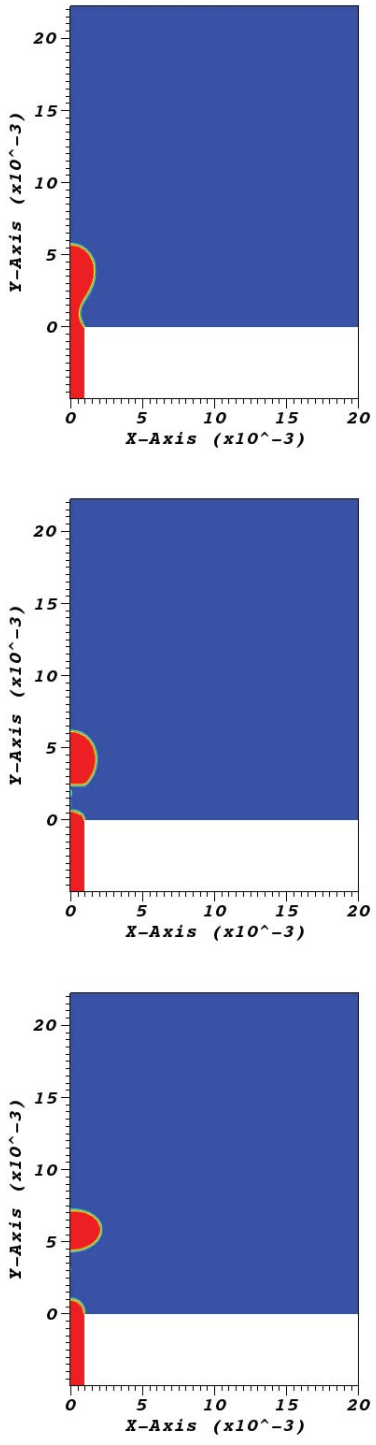

Figure 3.37: Drop deformation and detachment at $t = 0.35, 0.355,$ and 0.365 s using interFoam for a bubble rising in water test case

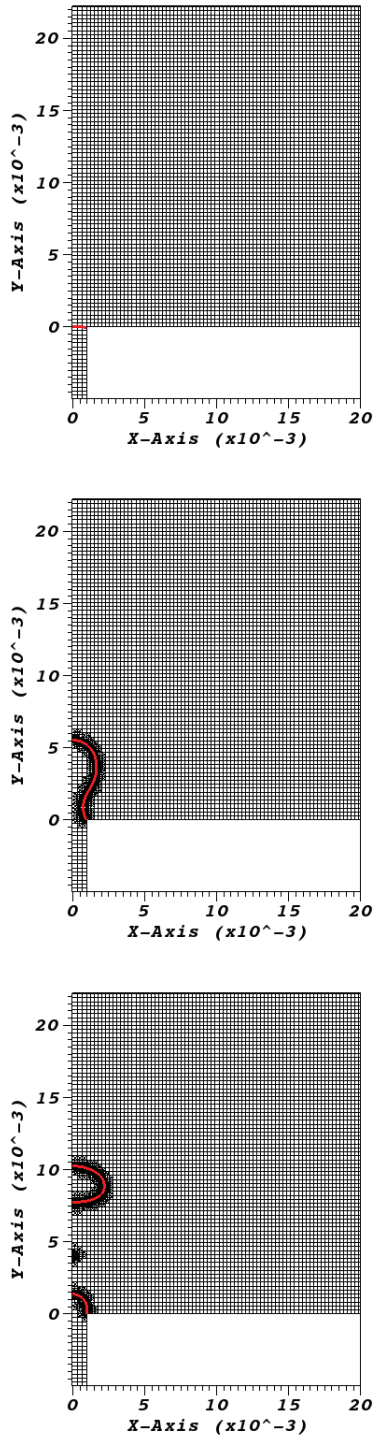

Figure 3.38: Dynamic refinement around the interface for a bubble rising in water test case

Figure 3.39 shows the pressure along the centerline predicted by `interFoam`. The jump in the pressure decreases with mesh refinement, but the difference in pressure jump between consecutive meshes decreases as well. This indicates the jump in pressure is converging with mesh refinement. The mesh independence of the centerline pressure predicted by the modified `interDyMFoam` is shown in Figure 3.40. This figure also shows that the pressure jump predicted by both solvers agrees on the fine meshes.

Finally, Table 3.18 presents the CPU time, bubble radius, and relative change in the radius for the two solvers. The CPU time is much less using `interDyMFoam`. For example, on the fine mesh, using `interFoam`, the CPU time increased by a factor of 4.6 compared to `interDyMFoam`. The table also shows that the bubble radius increases by increasing the mesh resolution but the difference becomes smaller for both solvers. Moreover, the drop sizes predicted by the two solvers compare well. As a result, the modified `interDyMFoam` solver gives results similar to `interFoam` but for much less CPU time.

Table 3.18
CPU time, bubble radius, and relative change using interFoam and interDyMFoam for a bubble rising in a water axisymmetric test case

Solver	CPU Time (s)	R	relative change
interFoam (Coarse)	9932	1.767	
interFoam (Standard)	30025	1.837	0.038
interFoam (Fine)	122298	1.865	0.015
interDyMFoam (Coarse)	1767	1.785	
interDyMFoam (Standard)	7624	1.830	0.025
interDyMFoam (Fine)	26507	1.845	0.008

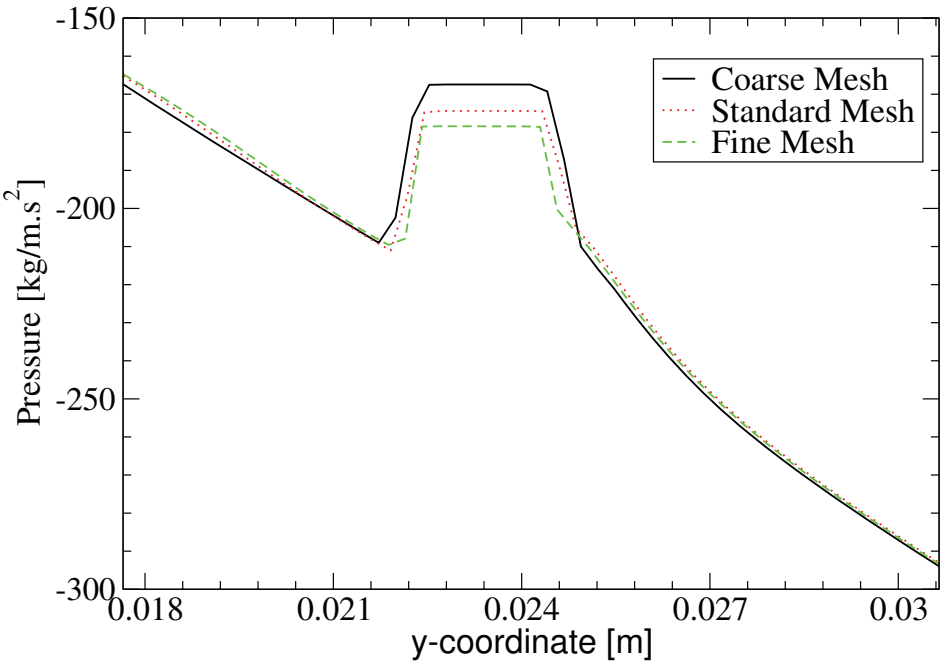

Figure 3.39: Pressure using interFoam for a bubble rising in a water axisymmetric test case

3.8 Summary and Conclusion

In summary, the `interDyMFoam` solver for 3D geometry is modified to work for 2D planar and axisymmetric geometries. Also, the solver is modified to allow for computing the deformation and breakup of drops or bubbles that are very small relative to the mesh of the flow domain. To validate the modified `interDyMFoam`, three test problems are considered: two in 2D planar geometry and one in axisymmetric geometry. The solutions from the modified `interDyMFoam` are compared with the solutions from `interFoam` from the aspects of mass accuracy, CPU time, and cell size around the interface. The modified `interDyMFoam` gives accurate solutions compared to `interFoam` with much

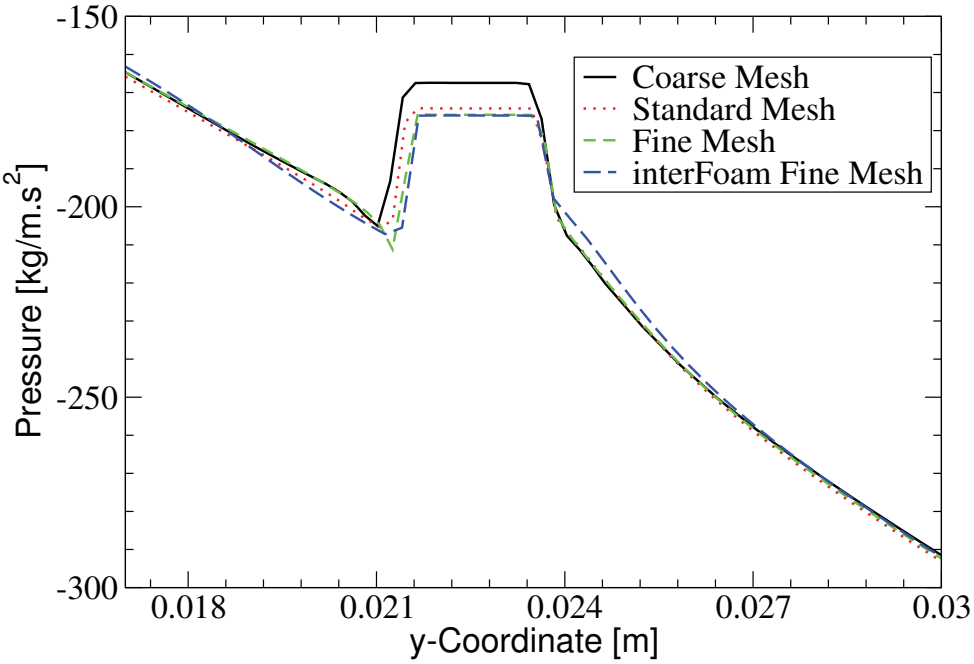

Figure 3.40: Pressure using interDyMFoam for a bubble rising in a water axisymmetric test case

less CPU time.

Chapter 4

Break up Conditions Inside a Spray Nozzle

An emulsion is a mixture of two immiscible phases of which one is dispersed in another. If the disperse phase itself contains drops of another liquid, then the emulsion is called a double emulsion. The most common types of double emulsions are the water-in-oil-in-water (W-O-W) and oil-in-water-in-oil (O-W-O). Their applications are found in many industries, including the food and pharmaceutical industries. Hydrophilic and lipophilic surfactants are encapsulated to produce a stable emulsion since they reduce the interfacial tension between phases. It is desirable for emulsions and double emulsions to have mechanical strength so they can stand against the mechanical forces generated by the manufacturing process such as mixing and pumping. After production, they should be

weak enough to release their encapsulations in the desired manner, such as the controlled release time during digestion and the release rate of flavor by chewing.

Double emulsions are inherently unstable due in part to coalescence and compositional ripening [1], [2], [3]. Coalescence is the process by which droplets merge with each other to form larger droplets, whereas, compositional ripening occurs by diffusion and/or permeation of the surfactants components across the disperse phase. The solid powder of emulsions produced by spraying is more stable than the liquids emulsions since the coalescence and ripening are reduced [11], [52]. Therefore, emulsion powders tend to have a longer shelf life which is desirable for economical reasons. In addition to these advantages, powder emulsions reduces the amount of the stabilizer needed.

It is important for double emulsions to retain their structure during and after the spray processing. For example, drops should not break up since this would change the structure, and hence properties of emulsions. Therefore, it is important to study droplet break up conditions inside and outside the nozzle.

In spraying emulsions, the atomization in a controlled manner is necessary to maintain its structure such as droplet size and droplet distribution. In their experiments, Dubey et al. [53] produced solid particles by spraying and studied the influences of the spray process parameters on the structure of double emulsions. Uddin et al. [54] investigated the effect of insoluble surfactants on the breakup of rotating liquid jets. Drop breakup during spraying of emulsions was also investigated by other researchers experimentally and computationally [55, 56] and emulsions with a good particle size distributions were produced.

In this chapter, the droplet breakup conditions inside a spray nozzle is analyzed for a simple emulsion. The study is achieved by the simulation of two-phase flow where the droplet is a Newtonian fluid and the outer phase is either Newtonian or non-Newtonian. Specifically, drops of different sizes are tracked through the flow field of the continuous phase fluid. The goal is to determine the effects of shear rate, capillary numbers, viscosity ratio, and fluid rheology (Newtonian or non-Newtonian) on the droplet breakup. Of particular interest is the critical drop size, that is, the largest drop that does not break up within the nozzle. To determine these critical drop sizes, many drops of different sizes must be tracked along different particle tracks for a given fluid system. Since this would be computationally very expensive in three dimensions, even with the use of dynamic meshing, the simulations are performed in two dimensions. This allows us to study the qualitative behavior and functional relationships, and will help determine appropriate three-dimensional simulations.

A first step, single phase flow calculations are performed to study the mesh independence for the outer phase fluid. This allows us to determine a mesh suitable to describe the outer phase flow field. Then, the two-phase flow is solved using dynamic refinement mesh to have accurate resolution around the interface. Using dynamic meshing around the drop interface as it moves through the flow field is necessary since the drops are most often very small relative to the dimensions of the geometry. The refined mesh from the single phase calculations is used as a basic mesh for the two-phase flow calculations.

4.1 Problem Description

The geometry and material properties in the simulations are taken from the experiments of Dubey et al. [11]. A cylindrical tapered die geometry is used in their experiments, see Figure 4.1. The upstream cylinder had radius $R_u = 3$ mm and length $L_u = 22.5$ mm, and the downstream cylinder had radius $R_d = 0.5$ mm and length $L_d = 1.5$ mm, giving the contraction ratio, R_u/R_d, of 6:1. Also, the emulsion (which is non-Newtonian) is produced from two Newtonian fluids. The transport properties of the fluids used to produce the emulsion are $\mu_d = 0.0634 Pa.s$, $\rho_d = 918 Kg/m^3$, $\mu_c = 0.0209 Pa.s$, $\rho_c = 1018 Kg/m^3$, and interfacial tension is $\sigma = 0.00575 Kg/s^2$. The computational domain is taken to be the two dimensional version of the actual nozzle geometry, with symmetry assumed along the centerplane. Moreover, in the simulations, the length of the downstream channel is extended from 1.5 mm to 20 mm, which allows us to consider the effect of nozzle channel length on drop breakup. The computational domain and coordinate system are also given in Figure 4.1. The transport properties for the droplets are taken to be the same as above:

$\eta_d = 0.0634$ Pa.s $\qquad \rho = 918 Kg/m^3 \qquad \sigma = 0.00575 Kg/s^2$.

The continuous phase is taken to be either the (non-Newtonian) emulsion or a Newtonian fluid. The non-Newtonian fluid has a slightly shear-thinning behavior, whose viscosity function followed the Bird-Carreau model

$$\frac{\eta - \eta_\infty}{\eta_0 - \eta_\infty} = [1 + (m\dot{\gamma})^2]^{\frac{n-1}{2}} \tag{4.1}$$

with parameters $\eta_0 = 0.113 Pa.s$, $\eta_\infty = 0.08 Pa.s$, $\rho = 977 Kg/m^3$, $m = 0.0049$ s, and $n = 0.01323$. In this model, η_0 and η_∞ represent the zero-shear-rate and infinite-shear-rate viscosities, respectively, m is a time constant whose reciprocal gives the shear rate at which the fluid begins to shear thin, and n is the dimensionless power-law index which controls the rate at which the fluid shears thin. The values of ρ and η_0 are used for the density and viscosity of the Newtonian continuous phase fluid. Table 4.1 summarizes the material parameters used in the simulations. The relation between the viscosity and shear rate for the non-Newtonian continuous phase fluid, given by the Bird-Carreau model, is shown in Figure 4.2. The viscosity ratio $\lambda = \eta_d/\eta_c$ corresponding to the Newtonian continuous phase was $\lambda = 0.56$, while λ ranged from 0.56 to 0.79 for the non-Newtonian continuous phase. The boundary conditions are outlined in Table 4.2. The centerline boundary ($y = 0$)

Table 4.1
Fluid parameters used in the simulations. The parameters for the non-Newtonian fluid correspond to the Bird-Carreau viscosity model, Eq. (4.1).

Fluid Phase	Density ρ [kg/m^3]	Viscosity η [Pas]
Dispersed	$\rho_d = 918$	$\eta_d = 0.0634$
Newtonian Continuous	$\rho_c = 977$	$\eta_c = 0.113$
Non-Newtonian Continuous	$\rho_c = 977$	$\eta_0 = 0.113$ $\eta_\infty = 0.08$ $m = 0.0049$ s $n = 0.01323$ [-]

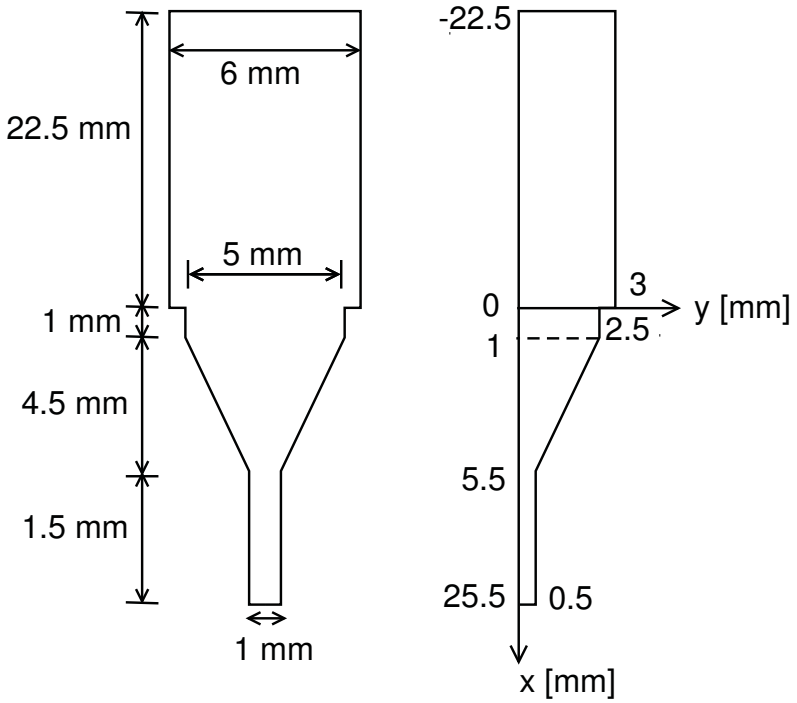

Figure 4.1: Schematic diagram of the nozzle geometry used in experiments (left) and the computational domain used in the simulations (right).

.

is specified as symmetry boundary. The velocity is set to be zero on the walls, zero gradient on the outlet, and constant $\boldsymbol{u} \cong (0.03m/s, 0, 0)$ at the inlet, where this velocity corresponds to one flow rate used in the experiments. The boundary conditions for the volume fraction function α are taken to be zero gradient, where this function is used in the two-phase calculations. Finally, the pressure is taken to be zero gradient along the inlet and walls, and zero on the outlet. The Reynolds number for the Newtonian flow is $Re = 0.76$ and for the non-Newtonian flow varies in the range $0.76 \leq Re \leq 1.1$.

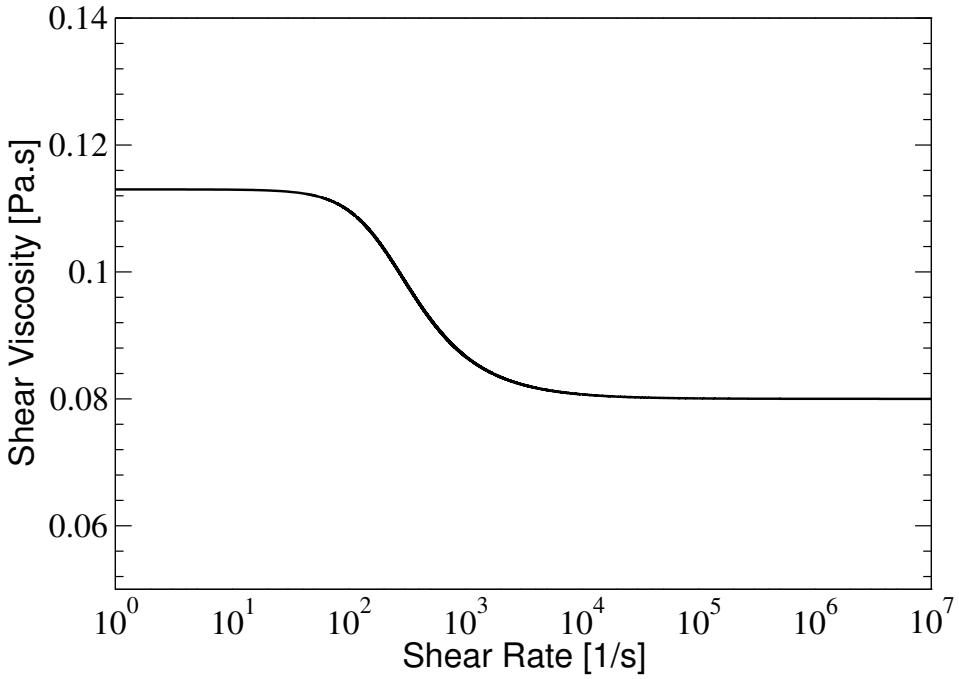

Figure 4.2: Viscosity vs shear rate of the non-Newtonian continuous phase fluid predicted by the Bird-Carreau model.

.

Table 4.2
Boundary conditions for the nozzle.

boundary	velocity	p	alpha1
inlet	$(0.0294731, 0, 0)$	zeroGradient	zeroGradient
outlet	zeroGradient	0	zeroGradient
walls	$(0, 0, 0)$	zeroGradient	zeroGradient
centerline	symmetryPlane	symmetryPlane	symmetryPlane
front and back	empty	empty	empty

4.2 Single Phase Flow Calculations

The computational domain is divided into five blocks as shown in Figure 4.3. Table 4.3 summarizes the number of cells in each block for the coarse, standard, and fine mesh. Note

that all meshes are non-uniform. The coarse and fine mesh are obtained by decreasing and increasing the number of cells in the standard mesh by a factor of 1.5, respectively. The single phase simulations are achieved using the `simpleFoam` solver of OpenFOAM®. In this section, the mesh independence and the convergence are discussed for both continuous phase fluids. In all graphs in this section, the dashed curves represent the non-Newtonian fluid and the solid ones represent the Newtonian fluid. The residual convergence for the fine mesh is illustrated in Figure 4.4 where the `residualControl` are 10^{-4} and 10^{-5} for the pressure and velocity respectively. That is, the velocity-pressure iterations terminate when the current solution to each discrete system of equations for velocity components and pressure produces a residual that meet these criteria simultaneous. The number of iterations needed for convergent is a little less in the case of Newtonian fluid.

The velocity along the center line is shown in Figure 4.5. The figure shows mesh independence for both fluids. By comparing the Newtonian and non-Newtonian graphs, the figure also shows almost identical velocities until $x = 5.5$ mm and then in the small channel ($x \geq 5.5$ mm), the velocity is greater in the Newtonian case. This is due to the shear-thinning behavior of the non-Newtonian fluid. The calculated pressure field for each fluid is also mesh independent. This is illustrated in Figure 4.6 where the centerline pressure is plotted. For both fluids, pressure decreases very slightly in the large channel, while the pressure gradient is much larger in magnitude in the small channel. This is consistent with the analytical expression for the constant pressure gradient in fully-developed pressure-driven flow, where it is seen that $\frac{dp}{dx}$ is inversely proportional to a power of channel height

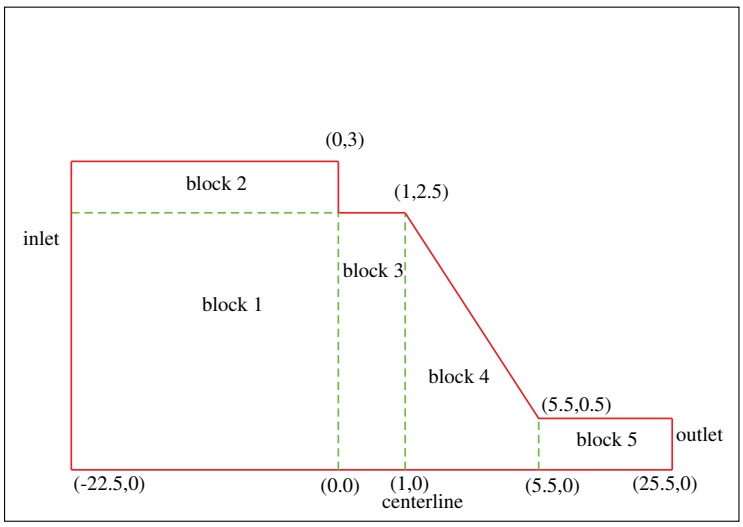

Figure 4.3: Computational domain and number of blocks for the nozzle.

H. Actually, for a Newtonian fluid $\frac{dp}{dx} = \frac{-3}{2}\mu\frac{Q}{H^3}$ where Q is the flow rate, while for a power-law, where $\eta(\dot{\gamma}) = K\dot{\gamma}^{n-1}$, $\frac{dp}{dx} = -\kappa\left[\frac{Q}{2H}(\frac{1}{H})^{\frac{1}{n}+1}(\frac{1}{n}+2)\right]^n$. Figure 4.6 also shows that the non-Newtonian fluid has lower pressure and lower pressure gradients. This is due to shear-thinning behavior of the fluid.

We choose to perform the two-phase calculations using the fine mesh which is the most suitable for tracking small drops as described in the next section.

Table 4.3
Number of cells in each block for the nozzle.

mesh	block 1	block 2	block 3	block 4	block 5	number of cells
Coarse	107×12	107×2	5×12	21×12	95×12	2950
Standard	160×18	160×3	7×18	32×18	142×18	6618
Fine	240×27	240×5	11×27	48×27	231×27	15510

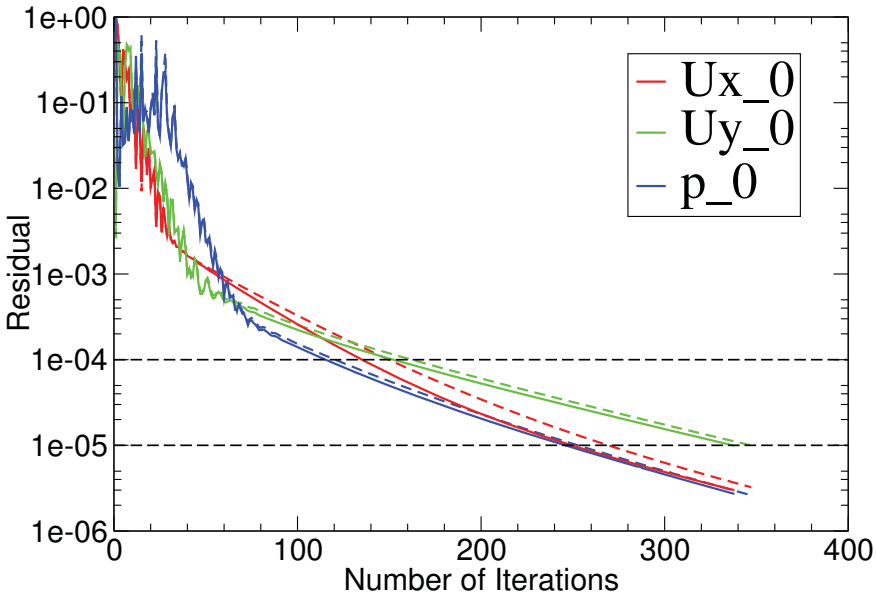

Figure 4.4: Residual using Newtonian (solid curves) and non-Newtonian (dashed curves) fluids for the refined mesh for the nozzle.

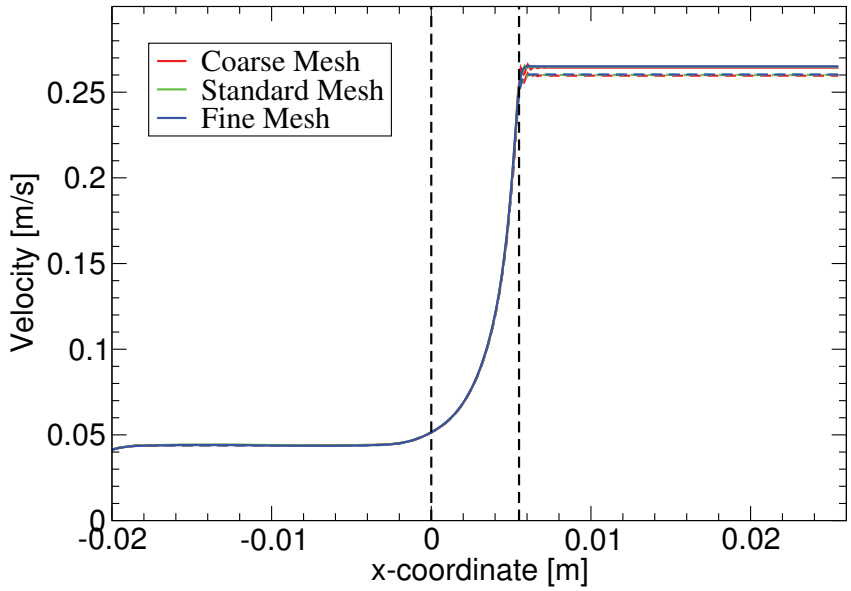

Figure 4.5: Velocity along the centerline for the single phase calculations using Newtonian (solid curves) and non-Newtonian (dashed curves) for the nozzle. The vertical dashed lines at $x = 0$ and $x = 5.5$ mm indicate the contracting part of the domain.

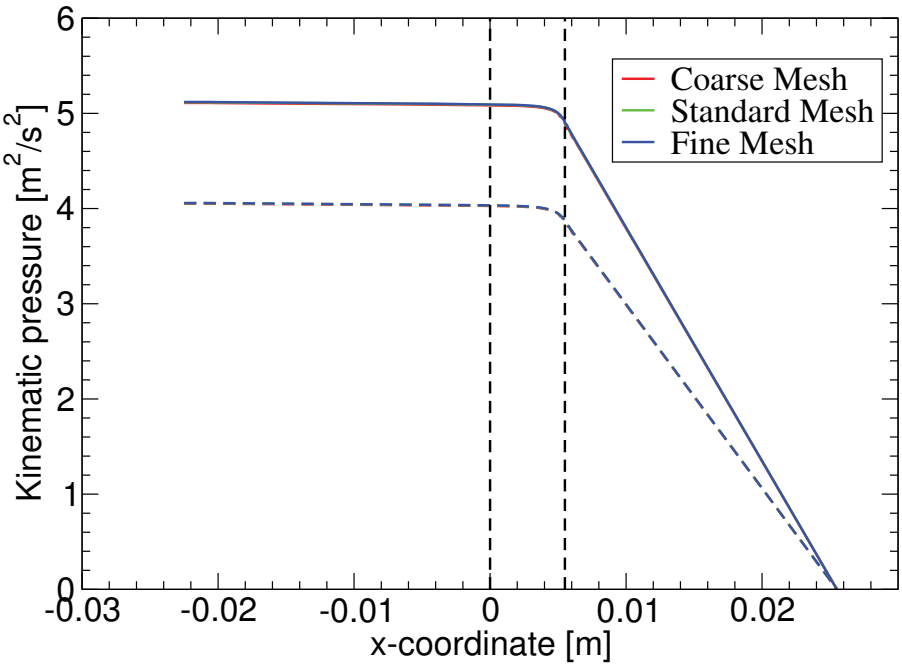

Figure 4.6: Pressure along the centerline for the single phase calculations using Newtonian (solid curves) and non-Newtonian (dashed curves) for the nozzle. The vertical dashed lines at $x = 0$ and $x = 5.5$ mm indicate the contracting part of the domain.

4.3 Drop Tracking Along Streamlines

In this section, the break up investigations are performed for droplets moving along different streamlines, see Figure 4.7. Each drop started in the fully developed flow in the upstream channel, and the streamlines were labeled by the initial y-coordinate of the drop's center. The streamlines ranged from $y = 0$ (centerline) to $y = 2.7$ mm (close to the upstream channel wall, located at $y = 3$ mm). Figure 4.8 shows the shear rates, $\dot{\gamma} = |\dot{\boldsymbol{\gamma}}|$ where $\dot{\boldsymbol{\gamma}} = \nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T$ is the rate-of-strain tensor, along various streamlines for both the Newtonian fluid (solid curves) and non-Newtonian fluid (dashed curves) as a function of particle transit time along the streamline. For each streamline, $t = 0$ corresponds to the beginning of the contraction section of the domain (at $x = 1$ mm). Characteristic of shear-thinning fluids, their shear rates are smaller than those for the Newtonian fluid close to the centerline and larger than those for the Newtonian fluid close to the wall.

Dynamic meshing is used in the simulations to have accurate resolution around the

Table 4.4
DynamicMeshDict parameters for the nozzle.

refineInterval	field	lowerRefInterval	upperRefInterval	
1	alpha (α)	0.1	0.9	
unrefineLevel	nBufferLayersR	nBufferLayers	maxRefinement	maxCells
10	$1-2$	1	$3-6$	400000

interface. The parameters used in `dynamicMeshDict` are as follows: refine interval equal to one, field `alpha` α, lower refine interval equal to 0.1, upper refine interval equal

to 0.9, number of buffer layers for unrefinement equal to 1, number of buffer layers for the refinement ranging between $1-2$ where one is used for large drops and two is used for small drops, and maximum refinement ranging between $3-6$ to establish the mesh independence of drop breakup. Table 4.4 summaries these parameters. Figure 4.9 shows the mesh refinement around a drop in two regions of the domain: in the upstream portion of the domain, where the shear rates are relatively low and the drop remains nearly circular, and in the downstream portion of the domain, where the shear rates are high and the drop elongates dramatically before breaking up. Recall that the refinements are shown as diagonals, although the cells are actually partitioned into rectangles.

Before the 2D planar simulations were performed, axisymmetric simulations were accomplished, keeping in mind that the axisymmetric simulation are valid only for droplets along the center line (see Figure 4.10). From the simulation, break up did not occur along the center line and the same results are observed in the 2D planar simulation. Moreover, no breakup occurred on streamlines $y < 0.75$ in the 2D simulations since the maximum drop radius that can be placed in the nozzle is 3 mm and larger drops are needed to get the breakup along these streamlines.

The critical drop size is defined to be either the largest drop size for which a drop does not break up in the domain, or the smallest drop size for which breakup occurs. Figure 4.11 shows a drop deformation and breakup along the streamline $y = 1.5$ for the non-Newtonian continuous phase fluid. Figure 4.12 is presented to show the critical breakup radius at different streamlines (top) and shear rate in the downstream channel (bottom) for both

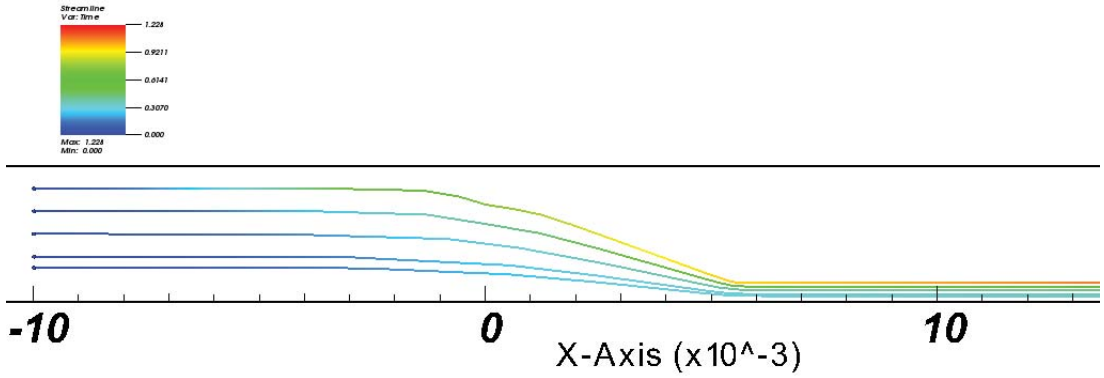

Figure 4.7: Nozzle streamlines at y equal to 0.75, 1, 1.5, 2, and 2.5 millimeter.

continuous phase fluids. Two curves are shown for each of these fluids. In each case, the lower and upper curves correspond to no breakup and breakup, respectively, so that the critical drop size lies somewhere between these two curves. For both fluids, the critical breakup radius is smaller when the streamline is farther from the centerline until about streamline $y = 1.5$ or $y = 2.0$ mm. Moreover, there is a rapid decreasing in the critical breakup radius for streamlines less than or equal to $y = 1.5$, corresponding to the downstream shear rate of $\dot{\gamma} \approx 500$ 1/s, while it slows down after that remaining almost the same. Note that the critical radius agrees between the two fluids after streamline $y = 1.5$. Therefore, the fluid rheology has an effect on the critical breakup radius along streamlines close to the center line, i.e $y < 1.5$. This effect makes the critical breakup radius larger in the case of Newtonian fluid where the opposite was expected since the shear rate is larger. A possible theory that could explain the results is because the non-Newtonian fluid has different viscosity ratios, affecting the critical breakup radius as will be discussed later. For

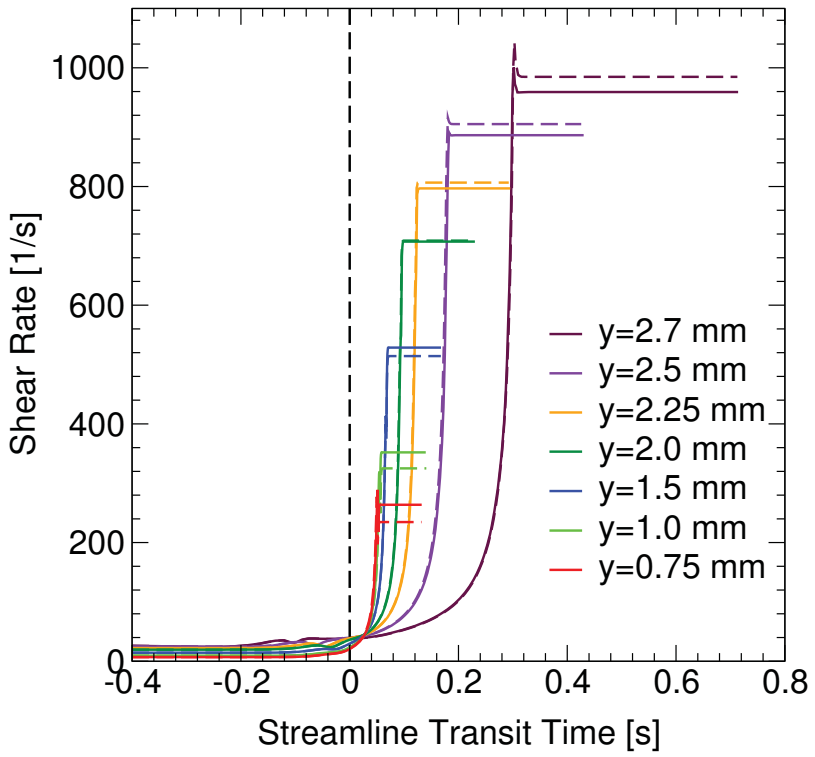

Figure 4.8: Shear rates as a function of transit time along a set of streamlines for the Newtonian (solid curves) and non-Newtonian (dashed curves) continuous phase fluid. Along each streamline, $t = 0$ corresponds to beginning of the contraction at $x = 1$ mm.

example, for the non-Newtonian fluid, the viscosity ratio $\lambda = \frac{\mu_d}{\mu_c}$ ranges from $\lambda = 0.62$ along $y = 0.75$ mm to $\lambda = 0.65$ along $y = 1.0$ mm, while for the Newtonian continuous phase, the viscosity ratio is somewhat smaller, remaining at $\lambda = 0.56$.

The capillary number represents the ratio of viscous forces to interfacial forces and is calculated using the formula $Ca = \frac{r\dot{\gamma}\eta_c}{\sigma}$, where r is the radius of the droplet, $\dot{\gamma}$ is the shear rate of the continuous phase, η_c is the continuous phase viscosity, and σ is the interfacial tension. The critical capillary number is a number for which the droplet breaks up if the capillary number is greater than the critical one ($Ca > Cacrit$) and does not break up if

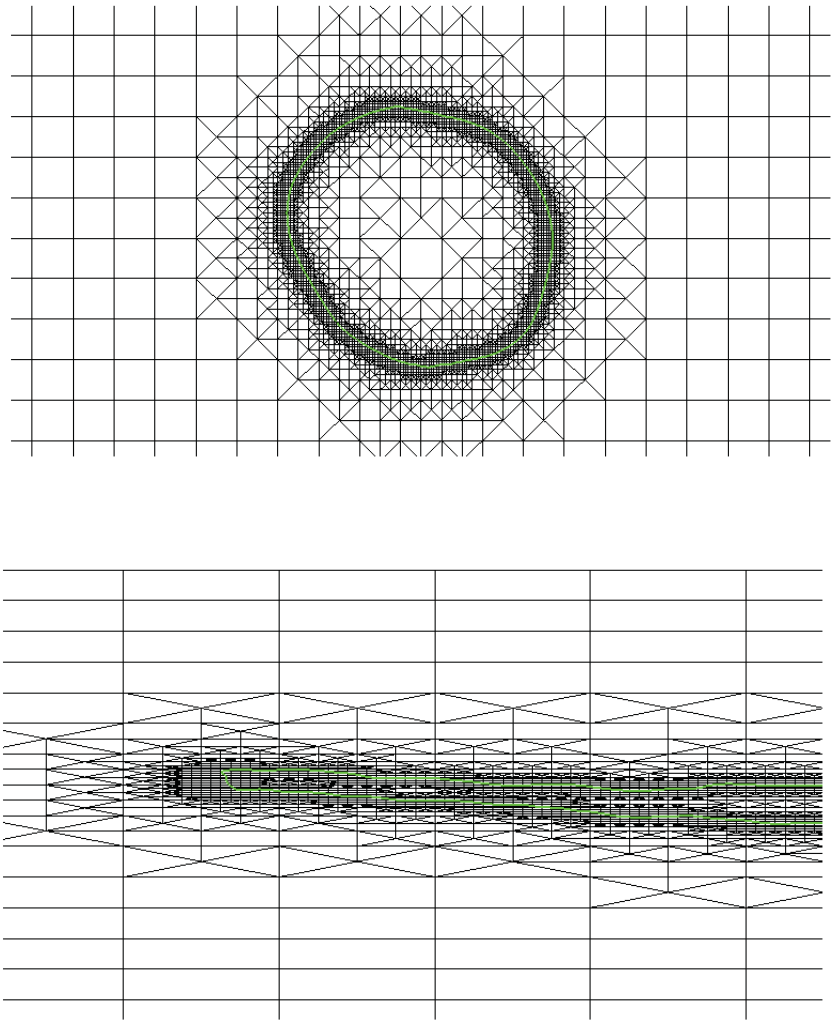

Figure 4.9: Mesh around the drop interface in the low-shear-rate upstream (top) and high-shear-rate downstream (bottom) portions of the domain for the nozzle.

the capillary number is less than the critical one ($Ca < Cacrit$). The general behavior seen for the critical radius is also observed for the critical capillary number *Cacrit* as shown in Figure 4.13, where the critical capillary number *Cacrit* is calculated using the constant shear rate and viscosity values in the downstream channel. The critical capillary numbers decrease with distance from the centerline before becoming approximately constant. Also, the the critical capillary numbers for the Newtonian fluid are larger than those for the

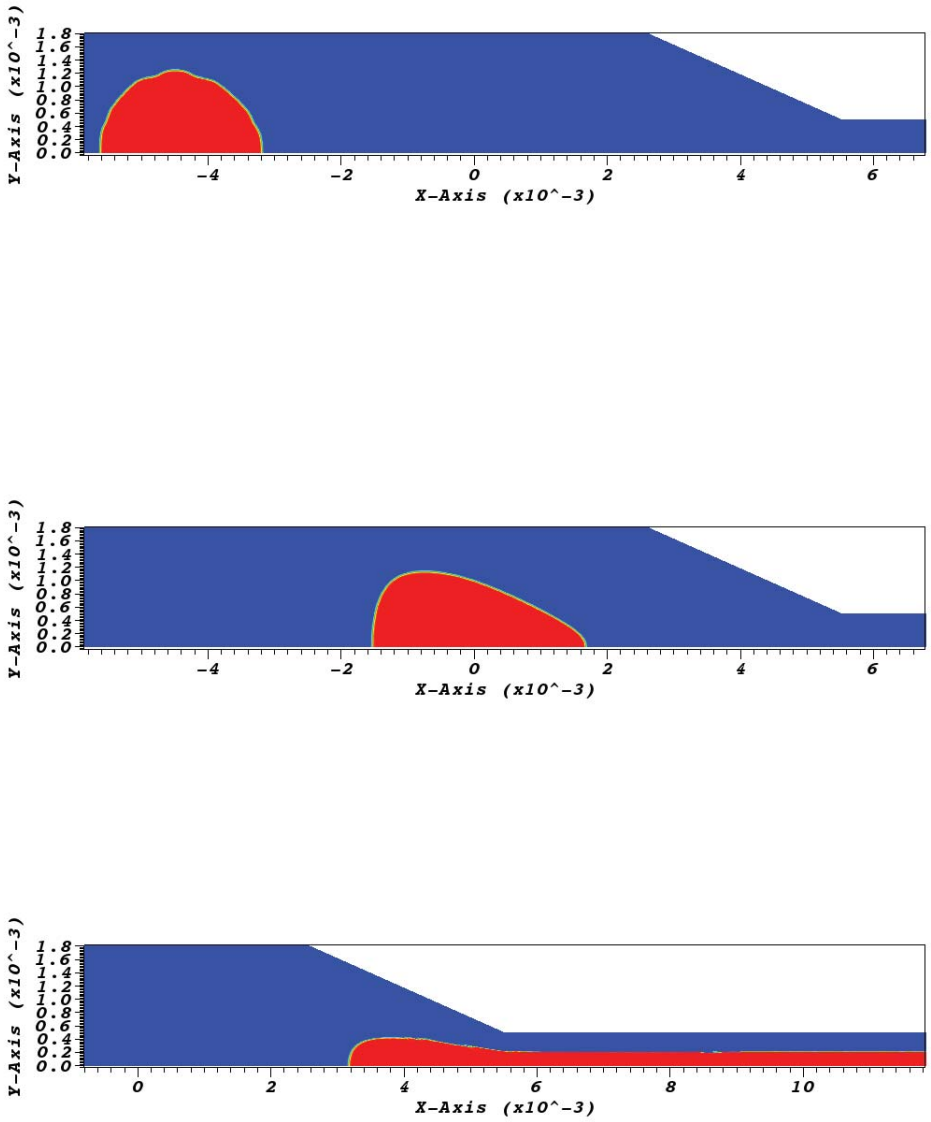

Figure 4.10: Drop deformation at $t = 0.01, 0.08$, and 0.13 s for the nozzle.

non-Newtonian fluid for streamlines close to the centerline.

The shear rate and drop breakup location along different streamlines for both fluids is shown in Figure 4.14. The location is given by the x-coordinate of the center of the elongated drop and is indicated by a circle along the streamline. The upper graph

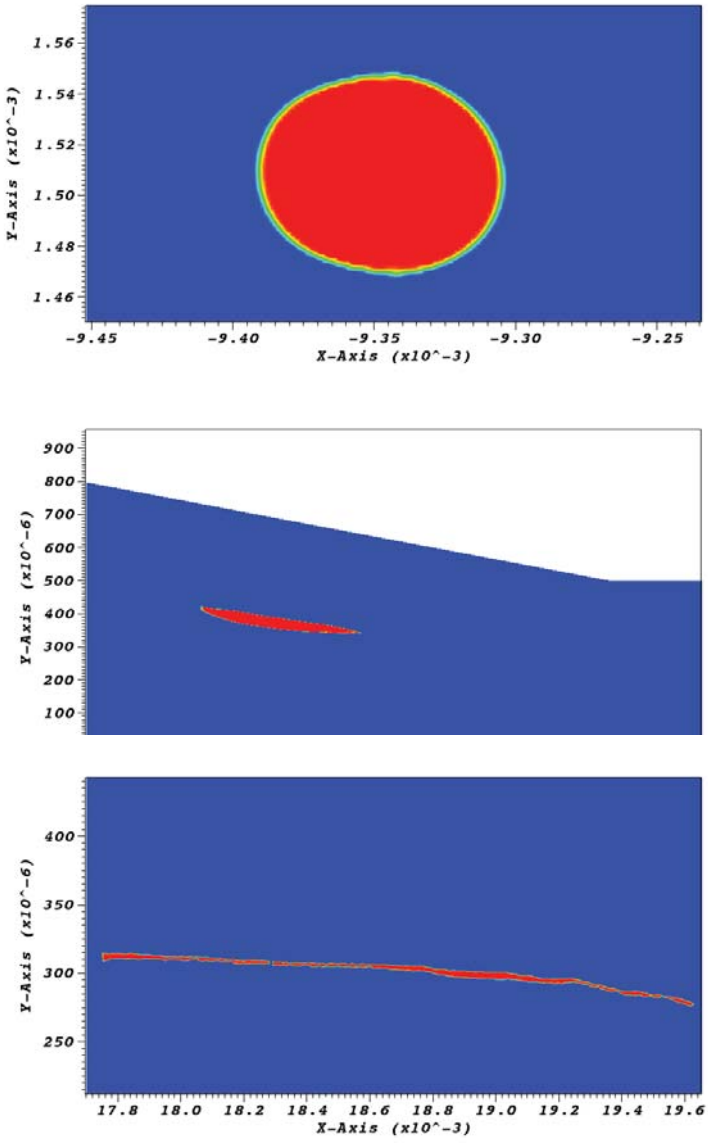

Figure 4.11: Drop deformation and breakup for streamline $y = 1.5$ at $t = 0.02, 0.42$, and 0.5 s for the non-Newtonian continuous phase for the nozzle.

corresponds to the Newtonian continuous phase and the lower graph corresponds to the non-Newtonian continuous phase. The vertical dashed lines at $x = 0$ and $x = 5.5$ mm indicate the contracting part of the domain (see Figure 4.1), while the vertical line at

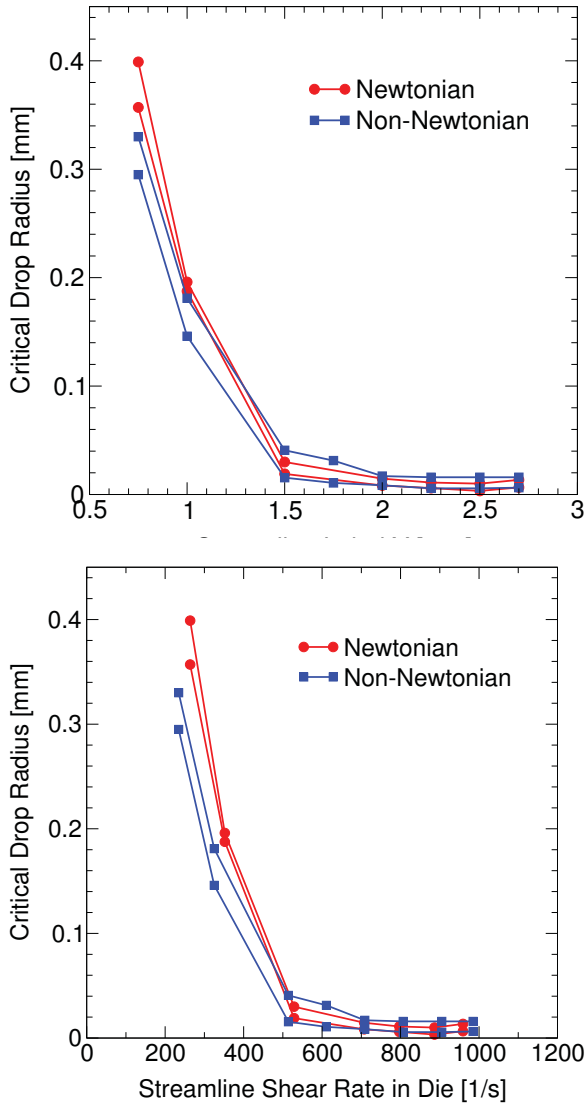

Figure 4.12: Critical drop sizes as a function of the streamline position (top) and the downstream shear rate (bottom) for the Newtonian and non-Newtonian continuous phase for the nozzle.

$x = 7$ mm indicates the end of the nozzle used in experiments. The breakup occurs near the beginning of the downstream channel for streamlines far from the centerline $y \geqslant 2.25$ mm, but it occurs closer to the end of the downstream channel for streamlines near the centerline. The breakup position gives an idea on how long the nozzle should be to produce break up along different streamlines. The simulations indicate that within the length of the

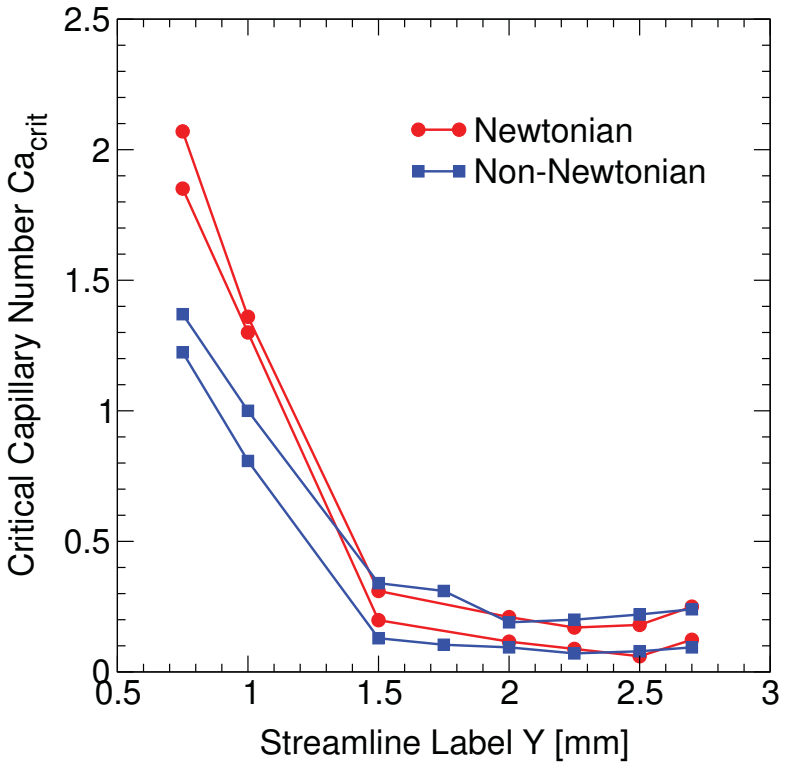

Figure 4.13: Critical Capillary number as a function of the streamline position for the Newtonian and non-Newtonian continuous phase for the nozzle.

actual nozzle used in experiments breakup would occur only along streamlines close to the wall $y \geq 2.25$. Table 4.5 gives the breakup locations for super-critical drop sizes along streamlines $y = 1.5$ and $y = 2.0$ for the non-Newtonian continuous phase. There appears to be little effect of super-critical drop size on breakup location, for drop sizes relevant to this geometry.

Grace [57] has constructed a plot of the critical capillary number as a function of the viscosity ratio $\lambda = \frac{\mu_d}{\mu_c}$ for Newtonian/Newtonian fluid systems in unbounded simple shear flow as shown in Figure 4.15. In the graph, the droplet does not breakup if the capillary number is under the curve and the breakup occurs when the capillary number is above the

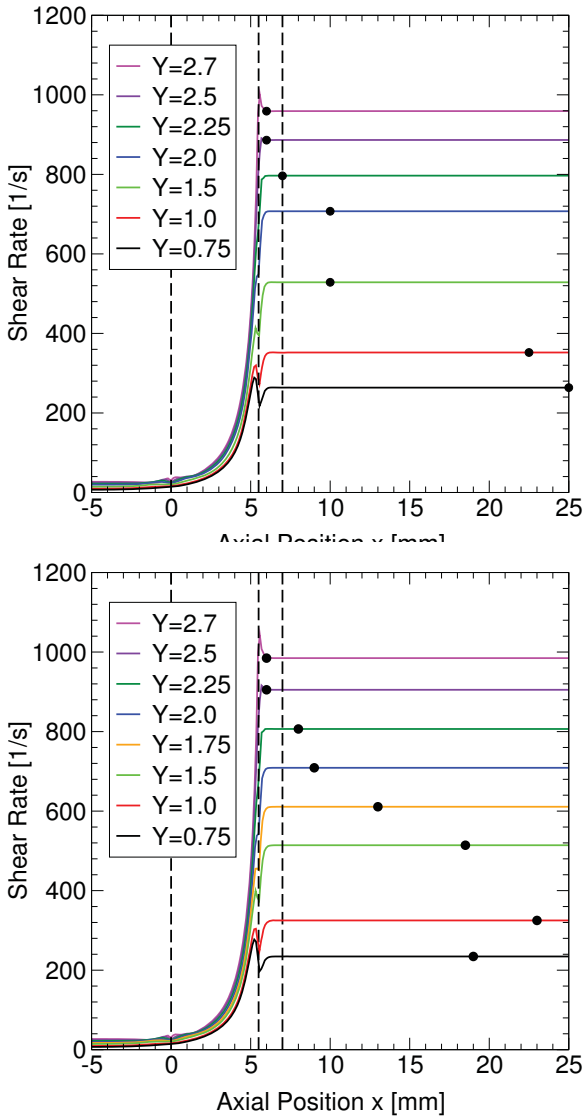

Figure 4.14: Breakup position of a drop along a given streamline in the Newtonian continuous phase (top) and non-Newtonian continuous phase (bottom) for the nozzle.

curve. The droplet does not breakup if the viscosity ratio is greater than a number around 4. Instead the large viscosity of the disperse phase relative to the continuous phase causes the droplet to rotate. Moreover, the minimum critical capillary number is when the viscosity ratio somewhere between about 0.6 and 1. This is because the viscosity of the disperse is

Table 4.5
Drops breakup location for streamlines $y = 1.5$ and $y = 2$ in the non-Newtonian continuous phase for the nozzle.

Streamline	drop radius (mm)	breakup location (mm)	breakup drop center (mm)
$y = 1.5$	0.041	17-20	18.5
$y = 1.5$	0.090	14-18	16.0
$y = 1.5$	0.148	11-16	13.5
$y = 1.5$	0.196	11-18	14.5
$y = 1.5$	0.245	10-17	13.5
$y = 2.0$	0.017	08-10	09.0
$y = 2.0$	0.090	09-13	11.0
$y = 2.0$	0.157	08-13	10.5
$y = 2.0$	0.202	08-16	12.0
$y = 2.0$	0.245	07-16	11.5

less than the viscosity of the continuous phase which makes the deformed droplet horizontal and aligned with the flow field. As a result, the viscosity ratio has an effect on the critical capillary number which implies that it affects the droplet breakup radius.

To study the relation between the critical capillary number and viscosity ratio for the Newtonian fluid at streamlines $y = 1$ and $y = 2$, Figure 4.16 is presented. Many droplets are tracked along these streamlines with different viscosity ratios to get the curves. The critical capillary number for each case is between the lower and upper curve for each case. The shear rate used in the computation of the capillary number is the streamwise constant shear rate in the downstream channel, namely $\dot{\gamma} = 352\ \text{s}^{-1}$ and $710\ \text{s}^{-1}$ for streamline $y = 1.0$ mm and 2.0 mm, respectively. The dashed vertical lines in the figure represent the range of viscosity values reached for the original drop in the non-Newtonian continuous phase fluid. The critical capillary number reaches the minimum when the viscosity ratio

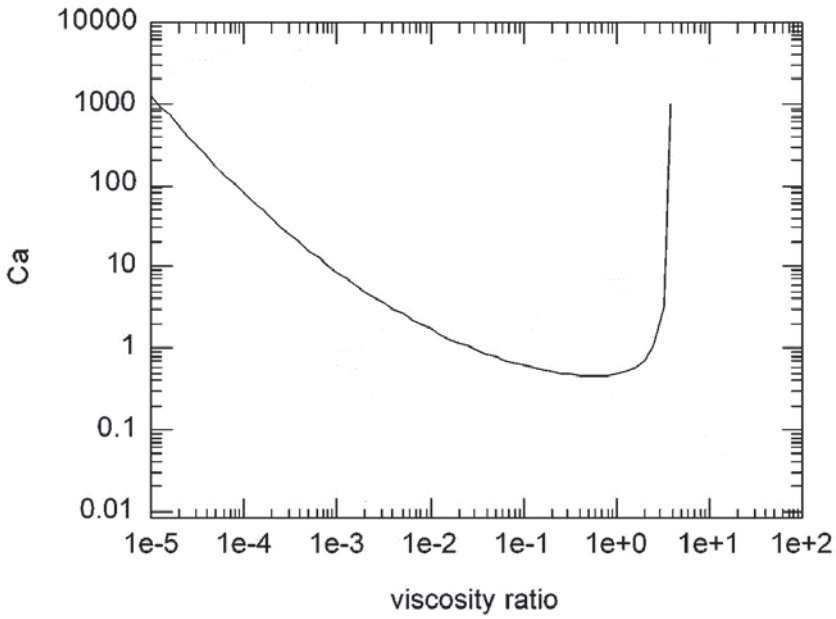

Figure 4.15: Critical capillary number vs viscosity ratio (Grace curve).

is between 0.56 and 1 and they show a similar behavior to the Grace plot where small changes are observed at streamline $y = 2$ due in part to small critical breakup radius. The critical capillary number decreases when the viscosity ratio increases from 0.56 to 0.75 at streamline $y = 1$ and remains constant at streamline $y = 2$. In particular, the viscosity ratio has an effect on the critical capillary and radius numbers where low ($\lambda \leq 0.56$) and high ($\lambda \geq 1$) viscosity ratio increase these numbers. The decrease in the critical capillary number at streamline $y = 1$, helps explain why the critical drop radius (Figure 4.12) and critical capillary number (Figure 4.13) are larger for the Newtonian continuous phase than for the non-Newtonian continuous phase for streamlines closer to the centerline, such as $y = 1$ mm, even though the shear rates are larger for the Newtonian case. Likewise, the nearly constant value of Ca_{crit} between $\lambda = 0.56$ and $\lambda = 0.79$ along streamline $y = 2.0$ mm helps explain why there is little or no difference between the critical drop radius (Fig. 4.12)

and critical capillary number (Fig. 4.13) along streamlines farther from the centerline, such as $y = 2.0$ mm.

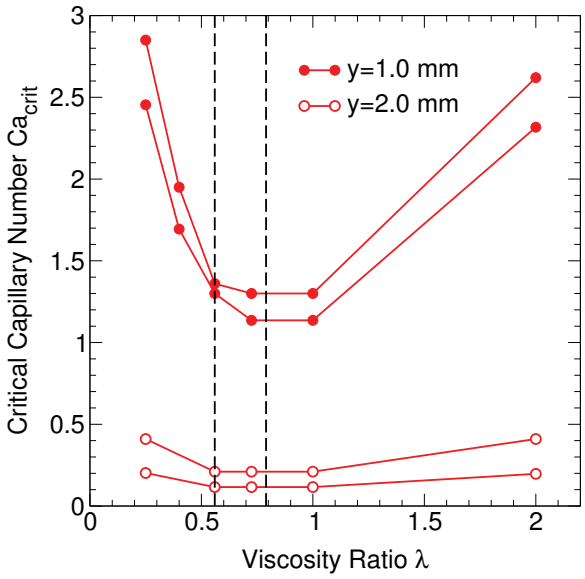

Figure 4.16: Critical capillary number as a function of viscosity ratio along two streamlines in the Newtonian continuous phase for the nozzle. The dashed vertical lines represent the range of viscosity ratios encountered for the original drop viscosity (see Table 4.1).

4.4 Summary and Conclusions

In summary, the droplet breakup conditions inside a spray nozzle is analyzed for a simple emulsion. The study is achieved by the simulation of two-phase flow where the droplet is a Newtonian fluid and the outer phase is either Newtonian or non-Newtonian. Because a large number of drops must be tracked in order to determine breakup conditions, the simulations were performed in two dimensions using the modified `interDyMFoam` solver. A first

step, single phase flow calculations are performed to study the mesh independence for the outer phase fluid. Then, the two-phase flow is solved using dynamic refinement mesh to have an accurate resolution around the interface. The simulations on a single phase flow revealed a mesh independence after analyzing velocity and pressure on three different meshes. The refined mesh is used as a basic mesh for the two-phase flow.

For both continuous phase fluids, there was an initial rapid decrease in critical drop size as distance from the centerline of the nozzle increased, i.e., as the shear rates experienced by the drop increased. Starting at approximately half-way between the centerline and the wall, the critical drop radius became approximately constant or decreased only slightly.

By noting the location of breakup within the nozzle, it was determined that drops near the centerline break up only for very long dies. Within the length of the nozzle used in experiments, only drops closer to the wall broke up for the flow rate considered.

It was also found that close to the centerline of the nozzle, critical drop sizes were larger for the Newtonian continuous phase than for the non-Newtonian continuous phase, even though the shear rates were larger along these streamlines for the Newtonian fluid. The explanation for this is partly due to the viscosity ratios reached in the simulations. This was illustrated by determining critical capillary numbers for a range of viscosity ratios along two streamlines in the Newtonian continuous phase. The resulting Ca_{crit}-vs-λ curves resembled the well-known Grace curve for steady simple shear flow.

From these simulations we can get general idea about the droplet breakup conditions inside a spray nozzle. Additional simulations and experimental validation are needed.

Chapter 5

Summary and Future Work

In summary, this study focused on the dynamic meshing in a two-phase flow solver. The 3D solver has been modified to allow for dynamic meshing around fluid-fluid interfaces in two-dimensional planar and axisymmetric geometries. Moreover, the procedure is modified to allow for computing the deformation and breakup of drops or bubbles that are very small relative to the mesh of the flow domain. This is necessary to avoid mass loss when tracking small drops or bubbles through flow fields.

To validate the modifications, the modified dynamic meshing code for two-dimensional planar geometry was applied to two test problems: drop deformation and breakup in linear shear flow, and drop formation and detachment from a micro T-channel. The modified dynamic meshing code for axisymmetric geometry was applied to a bubble rising from a pore into a static liquid. In these test problems, we studied computational time, mesh

independence, and mass accuracy. Comparisons were made with the two-phase flow solver without dynamic meshing.

To investigate the validity of the modified code, simulation and performances have been discussed in detail. It was found that the modified code produces accurate results with much less CPU time in comparison to simulations without dynamic meshing. A summary of the conclusions from the validation tests are given in Section 3.8.

The modified code was then applied to study the droplet breakup conditions inside a spray nozzle for a single emulsion, where droplets of various sizes were tracked through the flow field within the nozzle to determine the conditions under which they break up. The goal was to determine the largest drop sizes for which the breakup does not occur. The critical radius and capillary number at different streamlines with various shear rates were found along with the breakup location for both Newtonian and non-Newtonian continuous phase fluids. Furthermore, the effect of viscosity ratio on the critical capillary number in a Newtonian continuous phase fluid was determined for two streamlines.

The simulations showed that the critical drop sizes decreased rapidly as distance from the centerline increased, before becoming approximately constant. The simulations also showed that the fluid rheology has an effect on the critical drop size at streamlines near the centerline, however, there was no effect far from the centerline. This behavior was attributed to the range of viscosity ratios reached in the simulations and the Grace curves which where produced for this geometry. A summary of conclusions can be found in Section 4.4.

Although the modified dynamic meshing procedure was applied to fluid-fluid interfaces in two-phase flow problems, it can be adapted to other regions in the domain and for other types of flow problems in 2D planar and axisymmetric geometries.

Future Work

This thesis provides a preliminary investigation into the breakup conditions of emulsion droplets inside a spraying nozzle. The study used a nozzle with diameter 1 mm, downstream cylinder of length 20 mm, and flow rate corresponding to velocity $\boldsymbol{v} = (0.03, 0, 0)$ m/s. To get more detailed information about the droplet breakup conditions inside a spray nozzle, additional simulations are needed. The effect of different parameters on the droplet breakup can be study in the future. Some of those parameters are:

1. Geometric parameters such as the nozzle diameter and length of the contraction region.

2. Flow parameters, such as the inlet velocity or flow rate.

3. Material parameters, such as the viscosity of the continuous phase μ_c, interfacial tension σ, infinity shear rate η_∞, and n in the Bird-Carreau model.

An additional improvement to the code can be realized by replacing the volume of fluid (VOF) method with the coupled level set-volume fluid method (CLSVOF). The coupled level set-volume fluid method uses (1) the VOF method to calculate the volume fraction function α since it is mass conservative and (2) the level set method to calculate the

curvature κ since it is smoother. Also, the breakup conditions can be further studied when the droplet exits the nozzle. Finally, the breakup conditions can be investigated using 3D simulations to get more realistic results but this will require significantly more computational time, even when using dynamic meshing. However, the information obtained in our 2D simulations will serve to guide our choice of 3D simulations.

References

[1] A. Aserin, *Multiple emulsions: technology and applications*. Hoboken, New Jersery: Wiley & Sons, Inc, 2007.

[2] K. Pays, J. Giermanska-Kahn, B. Pouligny, J. Bibette, and F. Leal-Calderon, "Coalescence in surfactant-stabilized double emulsions," *Langmuir*, vol. 17(25), pp. 7758–7769, 2001.

[3] J. Sander, L. Isa, P. Rühs, P. Fischer, and A. Studart, "Stabilization mechanism of double emulsions made by microfluidics," *Soft Matter*, vol. 8, pp. 11471–11477, 2012.

[4] M. Ficheux, L. Bonakdar, F. Leal-Calderon, and J. Bibette, "Some stability criteria for double emulsions," *Langmuir*, vol. 14, pp. 2702–2706, 1998.

[5] S. Frasch-Melnik, F. Spyropoulos, and I. Norton, "W1/o/w2 double emulsions stabilised by fat crystals - formulation, stability and salt release," *Journal of Colloid and Interface Science*, vol. 350(1), pp. 178–185, 2010.

[6] N. Garti, "Progress in stabilization and transport phenomena of double emulsions in food applications," *Lebensmittel-Wissenschaft and Technologie*, vol. 30, pp. 222–235, 1997.

[7] Y. B. Li, S. G. Zangh, and J. G. Li, "perimental and theoretical approaches on uniform droplets formation from a rationed rotating membrane system," *Chemical Engineering Science*, vol. 66, pp. 788–796, 2011.

[8] R. Lutz, A. Aserin, L. Wicker, and N. Garti, "uble emulsions stabilized by a charged complex of modified pectin and whey protein isolate," *Colloids and Surfaces B: Biointerfaces*, vol. 72, pp. 121–127, 2009.

[9] M. Aghbashlo, H. Mobli, A. Madadlou, and S. Rafiee, "The correlation of wall material composition with flow characteristics and encapsulation behavior of fish oil emulsion," *Food Research International*, vol. 49, pp. 379–388, 2012.

[10] H. C. Carneiro, R. V. Tonon, C. R. Grosso, and M. D. Hubinger, "Encapsulation efficiency and oxidative stability of flaxseed oil microencapsulated by spray drying using different combinations of wall materials," *Journal of Food Engineering*, vol. 115, pp. 443–451, 2013.

[11] B. N. Dubey and E. J. Windhab, "Iron encapsulated microstructured emulsion particle formation by prilling process and its release kinetics," *Journal of Food Engineering*, vol. 115(2), pp. 198–206, 2013.

[12] C. Tang and X. Li, "Microencapsulation properties of soy protein isolate and storage stability of the correspondingly spray-dried emulsions," *Food Research International*, vol. 52, pp. 419–428, 2013.

[13] B. N. Dubey, M. R. Duxenneuner, , and E. J. Windhab, "Synthesis of functional food powder of simple and multiple emulsions through prilling process," *Proceedings of 11th International Congress on Engineering and Food*, 2011a.

[14] J. M. Ballester, N. Fueyo, and C. Dopazo, "Combustion characteristics of heavy oil-water emulsions," *Fuel*, vol. 75(6), pp. 695–705, 1996.

[15] C. D. Bolszo, A. A. Narvaez, V. G. McDonell, D. R. Dunn, and W. A. Sirignano, "Pressure-swirl atomization of water-in-oil emulsions," *Atomization and Sprays*, vol. 20(12), pp. 1077–1099, 2010.

[16] L. Broniarz-Press, M. Ochowiak, J. Rozanski, and S. Woziwodzki, "The atomization of water-oil emulsion," *Experimental Thermal and Fluid Science*, vol. 33, pp. 955–962, 2009.

[17] W. Kim, T. Yu, and W. Yoon, "Atomization characteristics of emulsified fuel oil by instant emulsification," *Journal of Mechanical Science and Technology*, vol. 26, pp. 1781–1791, 2012.

[18] J. Schroeder, A. Kleinhans, Y. Serfert, S. Drusch, H. P. Schuchmann, and V. Gaukel, "Viscosity ratio: A key factor for control of oil drop size distribution in effervescent atomization of oil-in-water emulsions," *Journal of Food Engineering*, vol. 111(2), pp. 265–271, 2012.

[19] A. Tratnig, G. Brenn, T. Strixner, P. Fankhauser, N. Laubacher, and M. Stranzinger, "Characterization of spray formation from emulsions by pressureswirl atomizers for spray drying," *Journal of Food Engineering*, vol. 95(1), pp. 126–134, 2009.

[20] Y. Sun and C. Beckermann, "Sharp interface tracking using the phase-field equation," *Journal of Computational Physics*, vol. 220(1), pp. 626–653, 2007.

[21] J. Sethian and P. Smereka, "Level set methods for fluid interface," *Annual Review of Fluid Mechanics*, vol. 35(1), pp. 341–372, 2003.

[22] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. New York: Springer-Verlag New York, Inc., 2003.

[23] S. S. Deshpande, L. Anumolu, and M. F. Trujillo, "Evaluating the performance of the two-phase flow solver interfoam," *Computational science and discovery*, vol. 5, 014016, 2012.

[24] W. Rider and D. Kothe, "Reconstructing volume tracking," *comput. Phys.*, vol. 141, p. 112, 1998.

[25] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *Comp. Phy.*, vol. 53, pp. 484–512, 1984.

[26] D. J. Mavriplis and A. D. Gosman, "Adaptive meshing techniques for viscous flow calculations on mixed element unstructured meshes," *Int. J. Num. Meth. Fluids*, vol. 34, pp. 93–111, 2000.

[27] D. J. Mavriplis, "Accurate multigrid solution of the euler equations on unstructured and adaptive meshes," *AIAA J.*, vol. 28, pp. 213–221, 1990.

[28] S. Z. Pizadeh, "An adaptive unstructured grid method by grid subdivision, local remeshing and grid movement," *AIAA*, vol. 99, p. 3255, 1999.

[29] R. W. Anderson, R. B. Pember, and N. S. Elliott, "An arbitrary lagrangian-eulerian method with adaptive mesh refinement for the solutions of the euler equations," *Comp. Phys.*, vol. 199, pp. 86–617, 2004.

[30] W. J. Coirier, "An adaptively-refined, cartesian, cell-based scheme for the euler and navier-stokes equations," *NASA Technical Memorandum*, p. 106754, 1994.

[31] J. D. Hunt, *An Adaptive 3D Cartesian Approach for the Parallel Computation of Inviscid Flow About Static and Dynamic Configurations*. PhD thesis, The University of Michigan, 2004.

[32] Q. Xue, *Development of adaptive mesh refinement scheme and conjugate heat transfer model for engine simulations*. PhD thesis, Iowa State University, 2009.

[33] *OpenFOAM User Guide*, 2.1.0 ed., December 2011.

[34] M. D. and M. R., *Drops and bubbles in interfacial research*. The Netherlands: Elsevier, 1998.

[35] B. Lafaurie, C. Nardone, R. Scardovelli, S. Zaleski, and G. Zanetti, "Modelling merging and fragmentation in multiphase flows with surfer," *Comput. Phys.*, vol. 113, pp. 134–147, 1994.

[36] S. Popinet and S. Zaleski, "A front-tracking algorithm for accurate representation of surface tension," *Int. J. Numer. Methods Fluids*, vol. 30, pp. 775–793, 1999.

[37] L. X.-D., F. R.P., and K. M., "A boundary condition capturing method for poissonâĂŹs equation on irregular domains," *Comput. Phys.*, vol. 160, pp. 151–178, 2000.

[38] K. M., F. R.P., and L. X.-D., "A boundary condition capturing method for multiphase incompressible flow," *Sci. Comput.*, vol. 15, pp. 323–360, 2000.

[39] M. Meier, G. Yadigaroglu, and B. Smith, "A novel technique for including surface tension in plic-vof methods," *Eur. J. Mech. B Fluids*, vol. 21, pp. 61–73, 2002.

[40] M. Meier, *Numerical and experimental study of large steam-air bubbles injected in a water pool*. PhD thesis, Swiss Federal Institute of Technology, 1999.

[41] J. U. Brackbill, D. B. Kothe, and C. Zemach, "A continuum method for modeling surface tension," *Computational Physics*, vol. 100, p. 335, 1992.

[42] H. Jasak, *Error analysis and estimation in the Finite Volume method with applications to fluid flows*. PhD thesis, Imperial College, University of London, 1996.

[43] J. D. Hoffman, *Numerical Methods for Engineers and Scientists*. New York: McGrawHill, 1992.

[44] P. Wesselin, *Principles of Computational Fluid Dynamics*. Heidelberg: Springer, 2001.

[45] C. Rhie and W. Chow, "A numerical study of the turbulent Âřow past an isolated airfoil with trailing edge separation," in *AIAA-82-0998, AIAA/ASME 3rd Joint Thermophysics, Fluids, Plasma and Heat Transfer Conference*, (St.Louis, Missouri), 1982.

[46] S. Patankar, *Numerical heat transfer and fluid flow*. Taylor & Francis, 1980.

[47] M. Peric, *A finite volume method for the prediction of three-dimensional fluid flow in complex ducts.* PhD thesis, Imperial College London (University of London), 1985.

[48] R. I. Issa, "Solution of the implicitly discretised fluid flow equations by operator-splitting," *Journal of Computational Physics*, vol. 62, pp. 40–65, 1986.

[49] H. Rusche, *Computational Fluid Dynamics of Dispersed Two-Phase Flows at High Phase Fractions*. PhD thesis, mperial College of Science, Technology and Medicine, 2002.

[50] S. M. Damián, *An Extended Mixture Model for the Simultaneous Treatment of Short and Long Scale Interfaces*. PhD thesis, Universidad Nacional del Litoral, 2013.

[51] T. Young, "An essay on the cohesion of fluids," *Phil. Trans. R. Soc. Lond.*, vol. 65, p. 95, 1805.

[52] B. Dubey, M. Duxxenneuner, C. Küchenmeister, P. Fischer, and E. Windhab, "Influences of rheological behavior of emulsions on the spraying process," in *24rd Annual Conference on Liquid Atomization and Spray Systems*, (Estoril, Portugal), September 2011.

[53] B. Dubey, M. Duxenneuner, and E. Windhab, "Prilling process: an alternative for atomization and producing solid particles of emulsions," *23rd Annual Conference on Liquid Atomization and Spray Systems, Brno, Czech Republic*, 2010.

[54] J. Uddin, S. Decent, and M. Simmons, "The effect of surfactants on the instability of a rotating liquid jet," *Fluid Dynammics Research*, vol. 40, pp. 827–851, 2008.

[55] L. Broniarz-Press, M. Ochowiak, J. Rozanski, and S. Woziwodzki, "The atomization of waterâĂŞoil emulsions," *Experimental Thermal and Fluid Science*, vol. 33, pp. 955–962, 2009.

[56] F. Tanner, S. Srinivasan, T. Althaus, K. Feigl, and E. Windhab, "Modeling and validation of the crystallization process in food sprays," *1th Triennial International Conference on Liquid Atomization and Spray System, Vail, Colorado, USA*, 2009.

[57] H. P. GRACE, "ispersion phenomena in high viscosity immiscible fluid systems and application of static mixers as dispersion devices in such system," *Chemical Engineering Communications*, vol. 14, pp. 225–277, 1982.

[58] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*. No. 43, Society for Industrial and Applied Mathematics, 1987.

[59] F. A. Morrison, *Understanding Rheology*. New York: Oxford University Press, Inc., 2001.

[60] J.-D. Yu, S. Sakai, and J. A. Sethian, "A coupled level set projection method applied to ink jet simulation," *Interface and Free Boundaries*, vol. 5, p. 459, 2003.

[61] D. Wong, M. Simmons, S. Decent, E. Parau, and A. King, "Break-up dynamics and drop size distributions created from spiralling liquid jets," *International Journal of Multiphase Flow*, vol. 30, pp. 499–520, 2004.

[62] R. B. Bird, R. C. Armstrong, and O. Hassager, *Dynamics of Polymeric Liquids*. New York: John Wiley and Son Inc, 2nd ed., 1987.

[63] W. Shyy, *Computational Modeling for Fluid Flow and Interfacial Transport*. Amsterdam: Elsevier, corrected ed., 1997.

[64] B. N. Datta, *Numerical linear algebra and applications*, vol. 116. Society for Industrial and Applied Mathematics, 2010.

[65] I. Demirdžić and M. Perić, "Space conservation law in finite volume calculations of fluid flow," *International journal for numerical methods in fluids*, vol. 8, no. 9, pp. 1037–1050, 1988.

[66] J. H. Ferziger and M. Perić, *Computational methods for fluid dynamics*, vol. 3. Springer Berlin etc, 2001.

[67] L. Hogben, *Handbook of linear algebra*. Chapman & Hall, 2007.

[68] K. Hutter and K. Jöhnk, *Continuum methods of physical modeling: continuum mechanics, dimensional analysis, turbulence*. Springer Verlag, 2004.

[69] H. Jasak, A. Jemcov, and J. Maruszewski, "Preconditioned linear solvers for large eddy simulation," in *CFD 2007 Conference, CFD Society of Canada*, 2007.

[70] H. Jasak and H. Rusche, "Dynamic mesh handling in openfoam," in *Proceeding of the 47th Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition, Orlando, Florida*, 2009.

[71] H. Jasak and Z. Tuković, "Dynamic mesh handling in openfoam applied to fluid-structure interaction simulations," in *Proceedings of the V European Conference on Computational Fluid Dynamics (ECCOMAS CFD 2010)(Lisbon, Portugal, 14-17 June 2010), JCF Pereira AS, Pereira JMC,(Eds.).(27)*, 2010.

[72] C. Kleinstreuer, *Biofluid dynamics: Principles and selected applications*. CRC, 2006.

[73] A. Krishnamoorthy and D. Menon, "Matrix inversion using cholesky decomposition," *arXiv preprint arXiv:1111.4144*, 2011.

[74] T. Lucchini, "Running openfoam tutorials."

[75] E. W. Merrill, "Rheology of blood," *Physiol Rev*, vol. 49, no. 4, pp. 863–88, 1969.

[76] S. V. Patankar and D. B. Spalding, "A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows," *International Journal of Heat and Mass Transfer*, vol. 15, no. 10, pp. 1787–1806, 1972.

[77] Y. Saad, *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003.

[78] J. F. Steffe, *Rheological methods in food process engineering*. Freeman Press, 1996.

[79] M. Stranzinger, *Numerical and Experimental Investigations of Newtonian and Non-Newtonian Flow in Annular Gaps with Scraper Blades*. PhD thesis, Swiss Federal Institute of Technology (ETH), 1999.

[80] Y. Takeda, "Velocity profile measurement by ultrasound doppler shift method," *International journal of heat and fluid flow*, vol. 7, no. 4, pp. 313–318, 1986.

[81] L. N. Trefethen and D. Bau III, *Numerical linear algebra*. No. 50, Society for Industrial and Applied Mathematics, 1997.

[82] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics*. USA: Pearson Education Limited, 2nd ed., 2007.

Appendix A

interFoam and interDyMFoam solvers

- The source code for the `interFoam` solver is

```
 /*---------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open
   \\    /   O peration     | Source CFD Toolbox
    \\  /    A nd           | Copyright (C) 2011
     \\/     M anipulation  | OpenFOAM Foundation
-------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it
    and/or modify it under the terms of the GNU General
    Public License as published by the Free Software
    Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be
    useful, but WITHOUT ANY WARRANTY; without even the
    implied warranty of MERCHANTABILITY or FITNESS FOR
    A PARTICULAR PURPOSE.  See the GNU General Public
    License for more details.

    You should have received a copy of the GNU General
    Public License along with OpenFOAM.  If not,
    see <http://www.gnu.org/licenses/>.
```

```
Application
    interFoam

Description
    Solver for 2 incompressible, isothermal immiscible
    fluids using a VOF (volume of fluid) phase-fraction
    based interface capturing approach.

    The momentum and other fluid properties are of the
    "mixture" and a single momentum equation is solved.

    Turbulence modelling is generic, i.e. laminar, RAS
    or LES may be selected.

    For a two-fluid approach see twoPhaseEulerFoam.

\*---------------------------------

#include "fvCFD.H"
#include "MULES.H"
#include "subCycle.H"
#include "interfaceProperties.H"
#include "twoPhaseMixture.H"
#include "turbulenceModel.H"
#include "interpolationTable.H"
#include "pimpleControl.H"
#include "ker.H"

// * * * * * * * * * * * * * * * * *

int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"

    pimpleControl pimple(mesh);

    #include "initContinuityErrs.H"
    #include "createFields.H"
    #include "readTimeControls.H"
    #include "correctPhi.H"
```

```
  #include "CourantNo.H"
  #include "setInitialDeltaT.H"


  // * * * * * * * * * * * * * * * *

Info<< "\nStarting time loop\n" << endl;

while (runTime.run())
  {
      #include "readTimeControls.H"
      #include "CourantNo.H"
      #include "alphaCourantNo.H"
      #include "setDeltaT.H"

      runTime++;

      Info<< "Time = " << runTime.timeName()
       << nl << endl;

      twoPhaseProperties.correct();

      #include "alphaEqnSubCycle.H"

      while (pimple.loop())
      {
          #include "UEqn.H"

          // --- Pressure corrector loop
          while (pimple.correct())
          {
              #include "pEqn.H"
          }

          if (pimple.turbCorr())
          {
              turbulence->correct();
          }
      }

      runTime.write();
```

```
        Info<< "ExecutionTime = " <<
         runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " <<
         runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}


// ************************************************//
```

- The source code of `interDyMFoam` solver is

```
 /*---------------------------------------------------*\
  =========                 |
 \\        /  F ield        | OpenFOAM: The Open
  \\      /                  | Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011
     \\/     M anipulation  | OpenFOAM Foundation
-------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it
    and/or modify it under the terms of the GNU General
    Public License as published by the Free Software
    Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be
    useful, but WITHOUT ANY WARRANTY; without even the
    implied warranty of MERCHANTABILITY or FITNESS FOR A
    PARTICULAR PURPOSE.  See the GNU General Public
    License for more details.

    You should have received a copy of the GNU General
    Public License along with OpenFOAM.  If not,
    see <http://www.gnu.org/licenses/>.
```

```
Application
    interDyMFoam

Description
    Solver for 2 incompressible, isothermal immiscible
    fluids using a VOF (volume of fluid) phase-fraction
    based interface capturing approach, with optional
    mesh motion and mesh topology changes including
    adaptive re-meshing.

\*-----------------------------------*/

#include "fvCFD.H"
#include "dynamicFvMesh.H"
#include "MULES.H"
#include "subCycle.H"
#include "interfaceProperties.H"
#include "twoPhaseMixture.H"
#include "turbulenceModel.H"
#include "pimpleControl.H"

// * * * * * * * * * * * * * * * * * * * *//

int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createDynamicFvMesh.H"
    #include "initContinuityErrs.H"
    #include "createFields.H"
    #include "readTimeControls.H"

    pimpleControl pimple(mesh);

    surfaceScalarField phiAbs("phiAbs", phi);
    fvc::makeAbsolute(phiAbs, U);

    #include "correctPhi.H"
    #include "CourantNo.H"
    #include "setInitialDeltaT.H"
    // * * * * * * * * * * * * * * * * * * * * //
    Info<< "\nStarting time loop\n" << endl;
```

```
while (runTime.run())
{
    #include "readControls.H"
    #include "alphaCourantNo.H"
    #include "CourantNo.H"

    #include "setDeltaT.H"

    runTime++;

    Info<< "Time = " << runTime.timeName()
     << nl << endl;

    scalar timeBeforeMeshUpdate =
     runTime.elapsedCpuTime();

    {
        volVectorField Urel("Urel", U);

        if (mesh.moving())
        {
            Urel -= fvc::reconstruct(fvc::meshPhi(U));
        }

        // Do any mesh changes
        mesh.update();
    }

    if (mesh.changing())
    {
        Info<< "Execution time for mesh.update() = "
            << runTime.elapsedCpuTime() -
                   timeBeforeMeshUpdate
            << " s" << endl;

        gh = g & mesh.C();
        ghf = g & mesh.Cf();
    }

    if (mesh.changing() && correctPhi)
    {
        #include "correctPhi.H"
```

```
        }

        if (mesh.changing() && checkMeshCourantNo)
        {
            #include "meshCourantNo.H"
        }

        twoPhaseProperties.correct();

        #include "alphaEqnSubCycle.H"

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            #include "UEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }

            if (pimple.turbCorr())
            {
                turbulence->correct();
            }
        }

        runTime.write();

        Info<< "ExecutionTime = " <<
            runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " <<
            runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}
```

```
// ********************************** //
```

Appendix B

Modifications to interDyMFoam

In this section, if a new line is added to the code, then the initial "AB" is added to the right of the line. If multiple consecutive lines are added, then the initial "AB...." is added to the first line and "AB" to the last line.

- Modifications to `dynamicRefineFvMesh` library.
  1. Create a new library called `dynamicRefineFvMesh2D` in `src/dynamicFvMesh` library by using the command
     cp -r `dynamicRefineFvMesh dynamicRefineFvMesh2D`
  2. Change the name in files from `dynamicRefineFvMesh` to `dynamicRefineFvMesh2D` by using the command
     "sed -i s/dynamicRefineFvMesh/dynamicRefineFvMesh2D/g file name" where the file name can be `dynamicRefineFvMesh2D.C` and `dynamicRefineFvMesh2D.H`
  3. Add `dynamicRefineFvMesh2D/dynamicRefineFvMesh2D.C` to the file `src/dynamicFvMesh/Make/files`
  4. To extend the number of buffer layers for refinement, part of the following is added to the `dynamicRefineFvMesh2D.C` source code;

```
const label nBufferLayersR =
readLabel(refineDict.lookup("nBufferLayersR")); //AB
PackedBoolList refineCell(nCells());

   if (globalData().nTotalCells() < maxCells)
    {
```

```
        selectRefineCandidates
        (
          lowerRefineLevel,
          upperRefineLevel,
          vFld,
         refineCell
           );
for (label i = 0; i < nBufferLayersR; i++)  //AB ....
        {
           extendMarkedCells(refineCell)
        }                                          //AB
```

- Modifications to `dynamicMesh` library.
  The modification to this library is done by modifying the `hexRef8.C` source code file which is located in the directory `src/dynamicMesh/polyTopoChange/polyTopoChange`.

  1. Create new files, `hexRef82D.C` and `hexRef82D.H` by using the command
     cp `hexRef8.C hexRef82D.C`
     cp `hexRef8.H hexRef82D.H`
  2. Change the name in files from `hexRef8` to `hexRef82D`
  3. Add `polyTopoChange/polyTopoChange/hexRef82D.C` to the file `src/dynamicMesh/Make/files`
  4. The lines added to the source code `hexRef82D.C` are given below:
     (a) Give the cells that are chosen for refinement a number greater than zero (1 is used here)

```
labelList cellMidPoint(mesh_.nCells(), -1);

  forAll(cellLabels, i)

   {
       label cellI = cellLabels[i];
       cellMidPoint[cellI] = 1;   //AB
   }
```

     (b) The empty faces and the edges on the empty faces are the only faces and edges visible for partitioning

```
for (label faceI = mesh_.nInternalFaces();
    faceI < mesh_.nFaces(); faceI++)  //AB....
    {
const label & patchID =
```

```
     mesh_.boundaryMesh().whichPatch(faceI);
if (isA<emptyPolyPatch>(mesh_.
                     boundaryMesh()[patchID]))
{
     isDivisibleFace[faceI] = true;
const labelList& fEdges = mesh_.faceEdges(faceI);
       forAll(fEdges, i)
            {
              label edgeJ = fEdges[i];

              isDivisibleEdge[edgeJ] = true;

            }
           }
 }                                          //AB
```

(c) Give the visible edges a number (1234 is used here)

```
forAll(cellMidPoint, cellI)

  {

if (cellMidPoint[cellI] >= 0)

   {
const labelList& cEdges = mesh_.cellEdges(cellI);

 forAll(cEdges, i)

    {
     label edgeI = cEdges[i];

      const edge& e = mesh_.edges()[edgeI];

  if
    (
      pointLevel_[e[0]] <= cellLevel_[cellI]
   && pointLevel_[e[1]] <= cellLevel_[cellI]
   && isDivisibleEdge[edgeI]              //AB
     )
   {
 edgeMidPoint[edgeI] = 12345;
   }
```

```
    }

    }
```

(d) Add a point in the middle of the visible faces

```
forAll(faceMidPoint, faceI)
{
if (faceMidPoint[faceI] >= 0
 && isDivisibleFace[faceI])       //AB
  {
 const face& f = mesh_.faces()[faceI];

 faceMidPoint[faceI] = meshMod.setAction

  (
   polyAddPoint
    (
     (
   faceI < mesh_.nInternalFaces()

   ? mesh_.faceCentres()[faceI]

   : bFaceMids[faceI-mesh_.nInternalFaces()]

       ),                      // point
    f[0],                     // master point
      -1,                    // zone for point
     true                   // supports a cell

        )

      );
// Determine the level of the corner points
// and midpoint will be one higher.

newPointLevel(faceMidPoint[faceI])
            = faceAnchorLevel[faceI]+1;

    }
  }
```

(e) The corner points

```
labelListList cellAnchorPoints(mesh_.nCells());
{
```

```
labelList nAnchorPoints(mesh_.nCells(), 0);

forAll(cellMidPoint, cellI)
 {
  if (cellMidPoint[cellI] >= 0)
   {
  cellAnchorPoints[cellI].setSize(8);

    }
  }

forAll(cellMidPoint, cellI)           //AB....

{
const cell& cFaces = mesh_.cells()[cellI];

forAll(cFaces, i)
 {
 label faceI = cFaces[i];

 const face& f = mesh_.faces()[faceI];

forAll(f, fp)

 {
  label pointI = f[fp];

  if
   (
    isDivisibleFace[faceI]            //AB

 && cellMidPoint[cellI] >= 0

 && pointLevel_[pointI] <= cellLevel_[cellI]

    )
     {
  if (nAnchorPoints[cellI] == 8)

    {
     dumpCell(cellI);
```

```
      FatalErrorIn

      (
    "hexRef82D::setRefinement(const labelList&"

    ", polyTopoChange&)"

      )   << "cell " << cellI
     << " of level " << cellLevel_[cellI]

     << " uses more than 8 points of equal or"

     << " lower level" << nl

     << "Points so far:" << cellAnchorPoints[cellI]

     << abort(FatalError);

      }

  cellAnchorPoints[cellI][nAnchorPoints[cellI]++]
      = pointI;

      }

      }
      }
```

(f) Split the empty faces to four new faces and the other faces to two new faces

```
forAll(faceMidPoint, faceI)
{
if (faceMidPoint[faceI] >= 0
 && affectedFace.get(faceI))
 {
// Face needs to be split and hasn't yet been
//done in some way (affectedFace - is impossible
//since this is first change but just for
//completeness)

const face& f = mesh_.faces()[faceI];

// Has original faceI been used (three faces
//added, original gets modified)
```

```
bool modifiedFace = false;
label anchorLevel = faceAnchorLevel[faceI];

if (isDivisibleFace[faceI])          //AB

   {
   face newFace(4);
forAll(f, fp)

  {
label pointI = f[fp];

if (pointLevel_[pointI] <= anchorLevel)

  {
// point is anchor. Start collecting face.

DynamicList<label> faceVerts(4);

faceVerts.append(pointI);

// Walk forward to mid point
// - if next is +2 midpoint is +1
// - if next is +1 it is midpoint
// - if next is +0 there has
//   to be edgeMidPoint

 walkFaceToMid
  (
     edgeMidPoint,
     anchorLevel,
     faceI,
     fp,
     faceVerts
   );

faceVerts.append(faceMidPoint[faceI]);

 walkFaceFromMid
  (
      edgeMidPoint,
      anchorLevel,
```

```
        faceI,
        fp,
        faceVerts
    );

// Convert dynamiclist to face.

newFace.transfer(faceVerts);

// Get new owner/neighbour

label own, nei;

    getFaceNeighbours
        (
          cellAnchorPoints,
          cellAddedCells,
          faceI,
          pointI,       // Anchor point
          own,
          nei
         );


if (debug)
 {
if (mesh_.isInternalFace(faceI))
 {
label oldOwn = mesh_.faceOwner()[faceI];
label oldNei = mesh_.faceNeighbour()[faceI];
   checkInternalOrientation
   (
     meshMod,
     oldOwn,
     faceI,
     mesh_.cellCentres()[oldOwn],
     mesh_.cellCentres()[oldNei],
     newFace
    );

   }
```

```
else
  {
label oldOwn = mesh_.faceOwner()[faceI];
   checkBoundaryOrientation
   (
     meshMod,
     oldOwn,
     faceI,
     mesh_.cellCentres()[oldOwn],
     mesh_.faceCentres()[faceI],
     newFace
    );
   }
  }

if (!modifiedFace)
  {
modifiedFace = true;
modFace(meshMod, faceI, newFace, own, nei);
  }

else
  {
addFace(meshMod, faceI, newFace, own, nei);
  }
  }
 }
}

else                            //AB....

 {
 face newFace(2);

forAll(f,fp)
 {
label pointI = f[fp];

label nextpointI = f[f.fcIndex(fp)];

label edgeI = meshTools::findEdge
      (mesh_, pointI, nextpointI);
```

```
if (edgeMidPoint[edgeI] >=0)

  {
   DynamicList<label> faceVerts(4);

   label pointJ = f[f.rcIndex(fp)];

   faceVerts.append(pointI);

   walkFaceToMid
    (
     edgeMidPoint,
     anchorLevel,
     faceI,
     fp,
     faceVerts
    );

   walkFaceFromMid
    (
     edgeMidPoint,
     anchorLevel,
     faceI,
     f.rcIndex(fp),
     faceVerts
    );

   faceVerts.append(pointJ);

   newFace.transfer(faceVerts);

   label own, nei;

    getFaceNeighbours
    (
     cellAnchorPoints,
     cellAddedCells,
     faceI,
     pointI,
     own,
     nei
    );
```

```
if (debug)
 {
 if (mesh_.isInternalFace(faceI))
  {
label oldOwn = mesh_.faceOwner()[faceI];
label oldNei = mesh_.faceNeighbour()[faceI];

checkInternalOrientation
  (
    meshMod,
    oldOwn,
    faceI,
    mesh_.cellCentres()[oldOwn],
    mesh_.cellCentres()[oldNei],
    newFace
   );
   }

else
  {
label oldOwn = mesh_.faceOwner()[faceI];

checkBoundaryOrientation
  (
   meshMod,
   oldOwn,
   faceI,
   mesh_.cellCentres()[oldOwn],
   mesh_.faceCentres()[faceI],
   newFace
   );
  }
  }

if (!modifiedFace)

 {
modifiedFace = true;

modFace(meshMod, faceI, newFace, own, nei);
```

```
  }

else
 {
addFace(meshMod, faceI, newFace, own, nei);
 }
 }
 }
 }                                        //AB

// Mark face as having been handled

  affectedFace.unset(faceI);

   }

 }
```

(g) Get the anchor cell in 2D case where the corresponding vertices on the empty faces have the same anchor cell

```
Foam::label Foam::hexRef82D::getAnchorCell
(
const labelListList& cellAnchorPoints,
const labelListList& cellAddedCells,
const label cellI,
const label faceI,
    const label pointI
) const
{
if (cellAnchorPoints[cellI].size())
{
label index = findIndex(cellAnchorPoints
                        [cellI], pointI);
if (index != -1)
 {
if (index >= 4)     //AB....
 {
if (index == 4)
 {
   index = 8;
 }
index = 8 - index;
 }                   //AB
```

```
return cellAddedCells[cellI][ index];
 }
// pointI is not an anchor cell.
// Maybe we are already a refined face so
// check all the face vertices.

const face& f = mesh_.faces()[faceI];
forAll(f, fp)
{
label index = findIndex(cellAnchorPoints
                      [cellI], f[fp]);

if (index != -1)
  {
if (index >= 4)     //AB....
  {
if (index == 4)
  {
    index = 8;
  }
index = 8 - index;
  }                  //AB
return cellAddedCells[cellI][index];
  }
  }

// Problem.

dumpCell(cellI);
Perr<< "cell:" << cellI << " anchorPoints:"
 << cellAnchorPoints[cellI] << endl;

FatalErrorIn("hexRef82D::getAnchorCell(..)")
<< "Could not find point " << pointI
<< " in the anchorPoints for cell "
<< cellI << end
<< "Does your original mesh obey the 2:1
    constraint and"
<< " did you use consistentRefinement to
     make your cells to refine"
<< " obey this constraint as well?"
<< abort(FatalError);
```

```
            return -1;
        }
    else
        {
            return cellI;
        }
    }
```

(h) Add internal faces

```
    if (faceMidFnd == midPointToFaceMids.end())
    {
    midPointToFaceMids.insert(edgeMidPointI,
                edge(faceMidPointI, -1));
    }

    else
     {
    edge& e = faceMidFnd();

    if (faceMidPointI != e[0])
    {
    if (e[1] == -1)
    {
    e[1] = faceMidPointI;

    changed = true;
    }
    }
    if (e[0] != -1 && e[1] != -1)

    {
    haveTwoFaceMids = true;
    }

    // Pout << "face edge " << e << endl;

    }

    // Check if this call of storeMidPointInfo
    // is the one that completed all the
    // nessecary information.

    if (changed && haveTwoAnchors)    //AB....
```

```
 {
const cell& cFaces = mesh_.cells()[cellI];

label face1 = -1;

forAll(cFaces, i)
 {
label faceJ = cFaces[i];

if (cellMidPoint[faceJ] != faceMidPointI

    && cellMidPoint[faceJ] >= 0

    && cellMidPoint[faceJ] != 123456789 )

    {
  face1  = faceJ;
    }
   }

const edge& anchors = midPointToAnchors
                    [edgeMidPointI];

label index = findIndex(cellAnchorPoints
              [cellI], anchorPointJ);

if (findIndex(cellAnchorPoints[cellI],
              anchorPointJ) == 0)
 {
  index = 4;
 }

if (findIndex(cellAnchorPoints[cellI],
              anchorPointJ) == 4)
  {
   index = 8;
  }

label point1 = cellAnchorPoints
               [cellI][8 - index];

label edgeMidPointJ = -1;
```

```
const face& f = mesh_.faces()[face1];

const labelList& fEdges =
             mesh_.faceEdges(face1);

DynamicList<label> newFaceVerts(4);

if (faceOrder == (mesh_.faceOwner()[faceI]
                                == cellI))

{
label anch = findIndex(f, point1);
if (pointLevel_[f[f.rcIndex(anch)]]
    <= cellLevel_[cellI])

{
label edgeJ = fEdges[f.rcIndex(anch)];

edgeMidPointJ = edgeMidPoint[edgeJ];

}

else

{
label edgeMid = findLevel(face1, f, f.rcIn
  dex(anch), false, cellLevel_[cellI] +1);

edgeMidPointJ = f[edgeMid];

}                                    //AB

newFaceVerts.append(faceMidPointI);

// Check & insert edge split if any

insertEdgeSplit

 (
 edgeMidPoint,
 faceMidPointI,  // edge between faceMid
 edgeMidPointI,  // and edgeMid
```

```
 newFaceVerts
 );

newFaceVerts.append(edgeMidPointI);

insertEdgeSplit
(
 edgeMidPoint,
 edgeMidPointI,
 edgeMidPointJ,      //AB
 newFaceVerts
);

newFaceVerts.append(edgeMidPointJ);
newFaceVerts.append(cellMidPoint[face1]);

   }

else
{
label anch = findIndex(f, point1);

if (pointLevel_[f[f.fcIndex(anch)]]
    <= cellLevel_[cellI])
{
label edgeJ = fEdges[anch];
edgeMidPointJ = edgeMidPoint[edgeJ];
 }

else
{
label edgeMid = findLevel(face1, f, f.fcIn
  dex(anch), true, cellLevel_[cellI] + 1);

edgeMidPointJ = f[edgeMid];

}

newFaceVerts.append(edgeMidPointJ);

insertEdgeSplit
 (
```

```
 edgeMidPoint,
 edgeMidPointJ,      //AB
 edgeMidPointI,
 newFaceVerts
 );

newFaceVerts.append(edgeMidPointI);

insertEdgeSplit
 (
 edgeMidPoint,
 edgeMidPointI,
 faceMidPointI,
 newFaceVerts
 );

newFaceVerts.append(faceMidPointI);

newFaceVerts.append(cellMidPoint[face1]);

  }

face newFace;

newFace.transfer(newFaceVerts);
```

(i) get the points that can be unrefined

```
Foam::labelList Foam::hexRef82D::
getSplitPoints(
const label axis, const scalar axisVal)//AB
const
{
if (debug)

{

checkRefinementLevels(-1, labelList(0));

}


if (debug)
```

```
 {

 Pout<< "hexRef82D::getSplitPoints :"

 << " Calculating unrefineable points"
 << endl;

   }


 if (!history_.active())

 {

FatalErrorIn("hexRef82D::
getSplitPoints()")<< "Only call if
constructed with history capability"

   << abort(FatalError);

  }



 // Master cell

 // -1 undetermined

 // -2 certainly not split point

 // >= label of master cell

labelList splitMaster(mesh_.nPoints(), -1);
labelList splitMasterLevel(mesh_.nPoints()
  , 0);

 // Unmark all with not 8 cells


for (label pointI = 0; pointI <
 mesh_.nPoints(); pointI++)
```

```
 {
const labelList& pCells =
        mesh_.pointCells(pointI);

vector  coord = mesh_.points()[pointI];


if (pCells.size()!=4||
    coord[axis]>axisVal) //AB
{
  splitMaster[pointI] = -2;

 }

 }

// Unmark all with different master cells

const labelList& visibleCells =
        history_.visibleCells();



forAll(visibleCells, cellI)

 {

const labelList& cPoints =
     mesh_.cellPoints(cellI);



if (visibleCells[cellI] != -1 &&
   history_.parentIndex(cellI) >= 0)

{

label parentIndex =
      history_.parentIndex(cellI);
```

```
 // Check same master.

 forAll(cPoints, i)

 {

label pointI = cPoints[i];



label masterCellI = splitMaster[pointI];



if (masterCellI == -1)

 {


splitMaster[pointI] = parentIndex;

splitMasterLevel[pointI] =
       cellLevel_[cellI] - 1;

 }

else if (masterCellI == -2)

 {

   }

else if

 (

(masterCellI != parentIndex)

|| (splitMasterLevel[pointI] !=
    cellLevel_[cellI] - 1)

  )
```

```
 {

splitMaster[pointI] = -2;

 }

 }

  }

else

{

forAll(cPoints, i)

{

label pointI = cPoints[i];


splitMaster[pointI] = -2;

            }

        }

    }
```

After the modifications are done, run `wmake libso` to compile the `dynamicMesh` and `dynamicFvMesh` libraries.
The modifications for axisymmetric geometry will be the same but by replacing `dynamicFvMesh2D` with `dynamicFvMeshAxi` and `hexRef82D` with `hexRef8axi`. The cells on the centerline should have special treatment as follows;

1. Split the empty faces to four new faces and the other faces to two new faces

   ```
   forAll(faceMidPoint, faceI)
   ```

```
{
if (faceMidPoint[faceI] >= 0 &&
    affectedFace.get(faceI))
 {
const face& f = mesh_.faces()[faceI];

bool modifiedFace = false;
label anchorLevel = faceAnchorLevel[faceI];

if (isDivisibleFace[faceI])           //AB
 {

face newFace(4);

forAll(f, fp)
 {
label pointI = f[fp];

if (pointLevel_[pointI] <= anchorLevel)
 {
// point is anchor. Start collecting face.

DynamicList<label> faceVerts(4);

faceVerts.append(pointI);

// Walk forward to mid point.
// - if next is +2 midpoint is +1
// - if next is +1 it is midpoint
// - if next is +0 there has to be
//   edgeMidPoint

walkFaceToMid
 (
 edgeMidPoint,
 anchorLevel,
 faceI,
 fp,
 faceVerts
 );

faceVerts.append(faceMidPoint[faceI]);
```

```
walkFaceFromMid
 (
 edgeMidPoint,
 anchorLevel,
 faceI,
 fp,
 faceVerts
 );

// Convert dynamiclist to face.

newFace.transfer(faceVerts);

label own, nei;

getFaceNeighbours
 (
 cellAnchorPoints,
 cellAddedCells,
 faceI,
 pointI,          // Anchor point
 own,
 nei
 );

if (debug)
{

if (mesh_.isInternalFace(faceI))
 {
label oldOwn = mesh_.faceOwner()[faceI];

label oldNei = mesh_.faceNeighbour()[faceI];

checkInternalOrientation
 (
 meshMod,
 oldOwn,
 faceI,
 mesh_.cellCentres()[oldOwn],
 mesh_.cellCentres()[oldNei],
 newFace
```

```
 );
}

else
{
label oldOwn = mesh_.faceOwner()[faceI];

 checkBoundaryOrientation
 (
 meshMod,
 oldOwn,
 faceI,
 mesh_.cellCentres()[oldOwn],
 mesh_.faceCentres()[faceI],
 newFace
 );
 }
}

if (!modifiedFace)
{
modifiedFace = true;

modFace(meshMod, faceI, newFace, own, nei);
}

else
{
addFace(meshMod, faceI, newFace, own, nei);
}
 }
 }
 }

else                              //AB....
{
face newFace(2);

forAll(f,fp)
{
label pointI = f[fp];
label nextpointI = f[f.fcIndex(fp)];
```

```
label edgeI = meshTools::findEdge(mesh_,
              pointI, nextpointI);
if (edgeMidPoint[edgeI] >=0)
{
label pointJ = f[f.rcIndex(fp)];
label prevpointJ = f[f.rcIndex
                   (f.rcIndex(fp))];
label edgep = meshTools::findEdge(mesh_,
               pointI, pointJ);

if (edgeMidPoint[edgep] >=0)
{
DynamicList<label> faceVerts(3);
faceVerts.append(pointI);

walkFaceToMid
(
 edgeMidPoint,
 anchorLevel,
 faceI,
 fp,
 faceVerts
);

walkFaceFromMid
(
 edgeMidPoint,
 anchorLevel,
 faceI,
 fp,
 faceVerts
);

newFace.transfer(faceVerts);
 }

else
{
DynamicList<label> faceVerts(4);

faceVerts.append(pointI);
```

```
walkFaceToMid
(
 edgeMidPoint,
 anchorLevel,
 faceI,
 fp,
 faceVerts
);

walkFaceFromMid
(
 edgeMidPoint,
 anchorLevel,
 faceI,
 f.rcIndex(fp),
 faceVerts
);

faceVerts.append(pointJ);

newFace.transfer(faceVerts);
 }

label own, nei;

getFaceNeighbours
(
 cellAnchorPoints,
 cellAddedCells,
 faceI,
 pointI,
 own,
 nei
);

if (debug)
{
if (mesh_.isInternalFace(faceI))
 {
label oldOwn = mesh_.faceOwner()[faceI];

label oldNei = mesh_.faceNeighbour()[faceI];
```

```
checkInternalOrientation
 (
  meshMod,
  oldOwn,
  faceI,
  mesh_.cellCentres()[oldOwn],
  mesh_.cellCentres()[oldNei],
  newFace
  );
 }

else
{
label oldOwn = mesh_.faceOwner()[faceI];

checkBoundaryOrientation
(
  meshMod,
  oldOwn,
  faceI,
  mesh_.cellCentres()[oldOwn],
  mesh_.faceCentres()[faceI],
  newFace
);
 }
}

if (!modifiedFace)
 {
modifiedFace = true;

modFace(meshMod, faceI, newFace, own, nei);
 }

else
{
addFace(meshMod, faceI, newFace, own, nei);
}
 }
}
 }                                        //AB
```

```
affectedFace.unset(faceI);
}
}
```

2. Add internal faces

```
if (changed && haveTwoAnchors)     //AB....
{
const cell& cFaces = mesh_.cells()[cellI];

label face1 = -1;

forAll(cFaces, i)
{
label faceJ = cFaces[i];

if (cellMidPoint[faceJ] != faceMidPointI

 && cellMidPoint[faceJ] >= 0
 && cellMidPoint[faceJ] != 123456789 )

{
face1  = faceJ;
}
}

const edge& anchors = midPointToAnchors
                     [edgeMidPointI];

label index = findIndex(cellAnchorPoints
                 [cellI], anchorPointJ);

if (findIndex(cellAnchorPoints[cellI],
    anchorPointJ) == 0)
{
index = 4;
}

if (findIndex(cellAnchorPoints[cellI],
    anchorPointJ) == 4)
{
index = 8;
}
```

```
label point1 = cellAnchorPoints[cellI]
                          [8 - index];

label edgeMidPointJ = -1;

const face& f = mesh_.faces()[face1];

const labelList& fEdges = mesh_.
                    faceEdges(face1);

 DynamicList<label> newFaceVerts(4);

if (faceOrder == (mesh_.faceOwner()[faceI]
              == cellI))
{
label anch = findIndex(f, point1);

if (pointLevel_[f[f.rcIndex(anch)]]
 <= cellLevel_[cellI])
{
label edgeJ = fEdges[f.rcIndex(anch)];
edgeMidPointJ = edgeMidPoint[edgeJ];
}

else
{
label edgeMid = findLevel(face1, f, f.rcIn
  dex(anch), false, cellLevel_[cellI] +1);

edgeMidPointJ = f[edgeMid];
}                                    //AB

newFaceVerts.append(faceMidPointI);

insertEdgeSplit
(
 edgeMidPoint,
 faceMidPointI,  // edge between faceMid
 edgeMidPointI,  // and edgeMid
 newFaceVerts
);
```

```
newFaceVerts.append(edgeMidPointI);

if (edgeMidPointI!=edgeMidPointJ)
 {
insertEdgeSplit
 (
 edgeMidPoint,
 edgeMidPointI,
 edgeMidPointJ,          //AB
 newFaceVerts
 );

newFaceVerts.append(edgeMidPointJ);
}

newFaceVerts.append(cellMidPoint[face1]);
 }

else
{
label anch = findIndex(f, point1);

if (pointLevel_[f[f.fcIndex(anch)]]
 <= cellLevel_[cellI])
{
label edgeJ = fEdges[anch];

edgeMidPointJ = edgeMidPoint[edgeJ];
}

else
{
label edgeMid = findLevel(face1, f, f.fcIn
  dex(anch), true, cellLevel_[cellI] + 1);

edgeMidPointJ = f[edgeMid];    //AB
}

if (edgeMidPointI!=edgeMidPointJ)
{
newFaceVerts.append(edgeMidPointJ);
```

```
insertEdgeSplit
 (
  edgeMidPoint,
  edgeMidPointJ,   //AB
  edgeMidPointI,
  newFaceVerts
  );
}

newFaceVerts.append(edgeMidPointI);

insertEdgeSplit
(
  edgeMidPoint,
  edgeMidPointI,
  faceMidPointI,
  newFaceVerts
);

newFaceVerts.append(faceMidPointI);

newFaceVerts.append(cellMidPoint[face1]);
}

face newFace;

newFace.transfer(newFaceVerts);
```

Appendix C

nozzle

The actual values in the nozzle simulations for both fluids

Table C.1
Critical breakup radius, capillary number, viscosity ratio, and the breakup position in the nozzle for the non-Newtonian fluid

y	MaxRef	drop radius(mm)	ca	viscosity ratio	break up position (mm)
0.75	3,4	0.2950-0.3300	1.224-1.37	0.623	15-23
1.00	3,4	0.1460-0.1810	0.808-1.00	0.649	20-26
1.50	5,6	0.0155-0.0408	0.129-0.34	0.686	17-20
1.75	5,6	0.0107-0.0313	0.104-0.31	0.700	12-14
2.00	5,6	0.0085-0.0170	0.094-0.19	0.709	08-10
2.25	5,6	0.0057-0.0159	0.071-0.20	0.718	07-09
2.50	5,6	0.0057-0.0159	0.079-0.22	0.725	05-07
2.70	5,6	0.0063-0.0159	0.094-0.24	0.729	05-07

Table C.2
Critical breakup radius, capillary number, viscosity ratio, and the breakup position in the nozzle for the Newtonian fluid

y	MaxRef	drop radius (mm)	ca	viscosity ratio	break up position (mm)
0.75	3,4	0.3570-0.3990	1.851-2.07	0.56	24-26
1.00	3,4	0.1876-0.1960	1.300-1.36	0.56	20-25
1.50	5,6	0.0190-0.0300	0.198-0.31	0.56	09-11
2.00	5,6	0.0083-0.0146	0.116-0.21	0.56	09-11
2.25	5,6	0.0059-0.0110	0.088-0.17	0.56	06-08
2.50	5,6	0.0033-0.0100	0.060-0.17	0.56	05-07
2.70	5,6	0.0065-0.0135	0.123-0.25	0.56	05-07

Table C.3
Critical breakup radius, capillary number, viscosity ratio, and the breakup position in the nozzle for the Newtonian fluid at stream line $y = 1$

y	MaxRef	drop radius (mm)	ca	viscosity ratio	break up position (mm)
1.00	3,4	0.3543-0.4120	2.454-2.85	0.25	12-17
1.00	3,4	0.2446-0.2811	1.694-1.95	0.4	16-22
1.00	3,4	0.1876-0.1960	1.300-1.36	0.56	20-25
1.00	3,4	0.1640-0.1875	1.136-1.30	0.725	20-25
1.00	3,4	0.1640-0.1875	1.136-1.30	1	16-22
1.00	3,4	0.3345-0.3780	2.317-2.62	2	22-25.5

Table C.4
Critical breakup radius, capillary number, viscosity ratio, and the breakup position in the nozzle for the Newtonian fluid at stream line $y = 2$

y	MaxRef	drop radius(mm)	ca	viscosity ratio	break up position (mm)
2.00	5,6	0.0145-0.0295	0.202-0.41	0.25	09-11
2.00	5,6	0.0083-0.0146	0.116-0.21	0.56	09-11
2.00	5,6	0.0083-0.0146	0.116-0.21	0.725	09-11
2.00	5,6	0.0083-0.0146	0.116-0.21	1	11-13
2.00	5,6	0.0141-0.0294	0.197-0.41	2	09-11