2016

COMPUTATIONAL METHODS FOR THE INVESTIGATION OF LIQUID DROP PHENOMENA IN EXTERNAL GAS FLOWS

Chao Liang
chaolian@mtu.edu

COMPUTATIONAL METHODS FOR THE INVESTIGATION OF LIQUID DROP PHENOMENA IN EXTERNAL GAS FLOWS

By

Chao Liang

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Mathematical Sciences

MICHIGAN TECHNOLOGICAL UNIVERSITY

2016

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 Co-advisor:	*Dr. Franz X. Tanner*
Dissertation Co-advisor:	*Dr. Kathleen A. Feigl*
Committee Member:	*Dr. Zhengfu Xu*
Committee Member:	*Dr. Song-Lin Yang*
Department Chair:	*Dr. Mark Gockenbach*

Contents

List of Figures

List of Tables

Preface

This dissertation is submitted for the degree of Doctor of Philosophy at Michigan Technological University. The research described herein was conducted under the supervision of Prof. Franz X. Tanner and Prof. Kathleen A. Feigl in the Department of Mathematical Sciences, Michigan Technological University, between September 2011 and April 2016. This work is to the best of my knowledge original, except where references are made to previous work. Neither this, nor any substantially similar dissertation has been or is being submitted for any other degree, diploma or other qualification at any other university.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisors Prof. Franz X. Tanner and Prof. Kathleen A. Feigl for their continuous support of my PhD study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having better advisors for my PhD study.

Besides my advisors, I would like to thank the rest of my thesis committee: Prof. Zhengfu Xu and Prof. Song-Lin Yang, for their insightful comments and encouragement in my PhD study.

My sincere thanks also go to Department of Mathematical Sciences, which provided me with teaching assistantship. Without their support it would not be possible to complete my PhD study.

I thank my fellows in the group: Dr. Abdallah Al-Habahbeh, Dr. Ahmad Baniabedalruhman, William Case, Olabanji Shonibare and Samer Alokaily for sharing literature and assistance. Especially, I thank William Case for sharing his post-processing drop size calculation routine.

I would also like to thank my families, especially my mother. They were always

supporting me and encouraging me with their best wishes.

List of Abbreviations

Acronyms

BiCG	Bi-conjugate Gradient
CG	Conjugate Gradient
CD	Central Differencing
UD	Upwind Differencing
CV	Control Volume
FVM	Finite Volume Method
SIMPLE	Semi-Implicit Method for Pressure-Linked Equations
PISO	Pressure-Implicit with Splitting of Operators

Non-dimensional parameters

Co	Courant number
Re	Reynolds number
We	Weber number
Oh	Ohnesorge number
Nu	Nusselt number
Pr	Prandtl number

Greek symbols

μ	dynamic viscosity
ν	kinematic viscosity
ρ	density
β	thermal expansion coefficient

Roman symbols

p	pressure
$\mathbf{v}$	velocity
T	temperature
e	specific energy
$\mathbf{g}$	gravitational constant
c_p	specific heat capacity at constant pressure
k	thermal conductivity
L_f	latent heat of fusion

Abstract

Computational methods for the investigation of drop deformation, drop breakup and drop solidification are developed and implemented in the open source computing environment OpenFOAM® [63]. The goal of this research is to simulate these drop phenomena by resolving all the relevant time and length scales in order to obtain correlations and statistical information which then can be used in the modeling of dispersed multiphase flows such as sprays. The use of such models is central for the simulation of dispersed multiphase flows because present-day computational technology does not have the capacity to resolve millions of droplets, as is typically encountered in sprays.

There are three aspects to this thesis. Three types of simulations are performed. First, the two-phase flow solver, `interFoam`, is modified to allow for simulating a moving droplet in an external airflow within a fixed computational domain. The modified solver is then used for the modeling and simulation of deformation and breakup of drops in axisymmetric and three dimensional symmetric flows, and the results are utilized to validate drop deformation and breakup theories, and to derive statistical information for the product drop size distributions in the various flow regimes. The Taylor Analogy Breakup (TAB) model has been modified, and this modified TAB model is also presented and validated.

Second, the feasibility and accuracy of calculating convective heat transfer coefficients using CFD is studied for two test cases: the flow between parallel flat plates and the flow past a cylinder. The feasibility and accuracy is demonstrated and achieved by comparing the results with the literature.

Third, for the investigation of the solidification of drops, an enhanced enthalpy-porosity model [6] is presented and implemented into OpenFOAM® to form a new solver, `modPolyMeltFoam`. Two test cases are used to validate the model and its implementation: the pure natural convection of water in a cavity and the solidification of water in a cavity. These tests show that the code performs very well comparing with results from the literature. The code is then coupled with the conjugate heat transfer solver, `chtMultiRegionFoam`, to create the solver `modFluidFluidChtMultiRegionFoam` which is able to simulate a stationary water droplet solidifying in a cold airflow. The results of the solidification studies are used to obtain correlations for the convective heat transfer coefficients, which in turn can be utilized in the modeling of freezing sprays where the solidification of millions of droplets needs to be described.

Chapter 1

Introduction

1.1 Background

Two-phase flows occur in many industrial and manufacturing applications, including spray-related applications, the formation of polymer blends and the creation of emulsions. In these applications, it is desirable to disperse one fluid in another, either to increase the rate of heat and mass transfer, or to form an emulsion with specific properties. This thesis focuses on sprays. One of the difficulties of simulating sprays is that one cannot resolve the length and time scales sufficiently due to the exceptionally high computational costs. However, modeling can be used to overcome this difficulty. Information needed for spray modeling can be provided by

using CFD. Specifically, in this thesis drop deformation and breakup is investigated to gain insight into the breakup mechanisms and to obtain statistical information on the product drop size distributions. Also, in order to obtain correlations used in the modeling of heat transfer in freezing sprays, solidification processes are simulated by resolving all the relevant length and time scales.

One method for spray freezing (also called spray chilling or prilling) processes is by contacting with a cold gas. It involves several mechanisms: (i) the deformation and breakup of individual droplets, (ii) the heat transfer between gas and droplets, and (iii) the solidification under natural convection inside droplets in a cold airflow.

There are three parts to this research. Background of these three parts is given in the following three subsections.

1.1.1 Droplet Deformation and Breakup

In sprays, droplet deformation and breakup play an important role in increasing the liquid-gas interface area and, consequently, in the enhancement of heat and mass transfer between the liquid and the surrounding gas. One of the difficulties of simulating sprays is that one cannot resolve the length and time scales sufficiently due to the exceptionally high computational costs. To overcome this difficulty, there are several drop breakup models in use, including the TAB model of O'Rourke and Amsden

[64], the Surface Wave Instability Atomization (Wave) model of Reitz [71], the Drop Deformation and Breakup (DDB) model of Ibrahim et al. [40], the Enhanced TAB (ETAB) model of Tanner [82], Tanner and Weisser [85] and its improved Cascade Atomization and Drop Breakup (CAB) model of Tanner [83].

Drop breakup of liquid-in-gas systems is categorized into four regimes, based on experimental observations, as reported by Guildenbecher et al. [30]. When the Ohnesorge number (Oh) is small, i.e., $Oh < 0.1$, the effects of drop viscosity can be neglected and the breakup depends only on the Weber number (We) of the droplets. The Ohnesorge number and the Weber number are defined as $Oh = \mu/\sqrt{\rho\sigma D}$, $We = \rho v^2 D/\sigma$, where μ is the drop dynamic viscosity, ρ is the drop density, σ is the surface tension, D is the characteristic length (typically the drop diameter), and v is the relative velocity between the drop and the surrounding air. For smaller Weber numbers ($We < 11$), breakup does not occur and only deformation takes place. For larger Weber numbers, drop breakup takes place in the following four breakup regimes: the bag breakup, $11 < We < 35$; the stamen breakup, $35 < We < 80$; the sheet-thinning/stripping breakup, $80 < We < 350$; and the catastrophic breakup, $We > 350$.

Computer simulations are an effective tool for the investigation of the dynamics of drop deformation and the mechanisms of drop breakup in different regimes. Various researchers have simulated the drop deformation and breakup numerically. Zaleski et al. [96] simulated the deformation and breakup of two dimensional drops. Their

simulations were able to capture details of the nonlinear interaction between the deforming droplets and the vortical structures in the droplets' wake. Meng and Colonius [55] performed two dimensional symmetric simulations of breakup of water cylinders in the flow behind normal shocks. They found the existence of recirculation regions and an upstream jet in the wake. The steady motion of deformable axisymmetric drops was simulated by Dandy and Leal [15] at several Reynolds and Weber numbers using finite difference methods. These investigations showed that at smaller Reynolds numbers the shape of the drop tends toward a spherical cap, but at larger Reynolds numbers the drop becomes more disk shaped. Bozzi et al. [9] used finite element methods to simulate the steady motion of axisymmetric drops. One of their findings was that external recirculation zones can be attached to or disjoint from the drop, depending on the Reynolds number. Liang et al. [50] presented simulations of axisymmetric drop breakups using the volume of fluid (VOF) method for a limited number of cases. Their results showed fair qualitative agreement with experiments and theoretical correlations in terms of the droplet shape, breakup time, and drag coefficients. Han and Tryggvason [32] presented simulations of axisymmetric drop breakups due to acceleration by a constant body force. Their numerical results are summarized by "breakup maps" where the different breakup modes are shown in the Eötvös number-Ohnesorge number (Eo-Oh) diagram for different values of the viscosity and the density ratios. Later, the same authors, Han and Tryggvason [33], presented simulations of axisymmetric drop breakups caused by impulsive acceleration using the finite

difference front tracking method. Their results are summarized by "breakup maps" where the different breakup modes are shown in the Weber number-Reynolds number (We-Re) diagram for different values of the density ratios.

In the DDB, TAB, ETAB and CAB models, the drop deformation and onset of breakup are predicted using the Taylor Oscillator. Various modifications to the Taylor Oscillator have been made by several researchers, and the modified models have been tested by simulating sprays. Liu et al. [52] accounted for the effects of drop distortion and oscillation due to the relative motion between the drop and the gas, and let the drag coefficient vary between a rigid sphere (no distortion) and a disk (maximum distortion). They tested the modified drop drag model with the TAB and the WAVE model to obtain improved predictions of diesel sprays. Experiments investigating the microscopic structure of the drop breakup process have been conducted by Hwang et al. [39], and the results show that the drop flattening significantly affects the drop's drag coefficient. They found that the drop trajectories could be modeled adequately using a modified dynamic drag model that accounts for drop distortion. Liu and Reitz [53] used a dynamic drag model that is a modified version of the DDB model and accounts for the increase of both the drop's frontal area and its drag coefficient, as a function of its distortion. They analyzed the drop trajectory and its distortion during the initial stage of the drop breakup process.

Investigations have also been conducted to model product drop size distribution after

drop breakup. One method for modeling drop size distributions is empirical. A simple mathematical expression is chosen that fits the data collected for a range of atomizers, resulting in various distributions including Rosin-Rammler [73], Nukiyama-Tanasawa [61], log-normal, root-normal [86], log-hyperbolic [94]. However, with an empirical approach it is difficult to extrapolate the data to regimes outside the experimental range [4]. An alternative to the empirical approach is the maximum entropy (ME) method, pioneered by Sellens and Brzustowski [75] and Li and Tankino [49], which determines the most likely drop size distribution as the one that maximizes an entropy function under a set of physical constraints [4, 20]. A drawback of the ME method is its low convergence rate [47]. Another alternative to the empirical approach is the Discrete Probability Function (DPF) method, developed by Sivathanu and Gore [76], which divides the spray formation process into deterministic and non-deterministic portions by assuming that the spray formation involves a series of breakup stages where a fluid mechanics instability analysis can be used. The DPF method was first applied to modeling drop size distributions in Newtonian sprays by Sovani et al. [78, 79]. The DPF method has not been validated extensively due to the difficulties of obtaining experimental data. In addition, the DPF method requires the probability density function (PDF) of the fluctuating initial conditions as input, which can be provided by methods of Computational Fluid Dynamics (CFD) [4].

1.1.2 Convective Heat Transfer Coefficient Calculation

Convective heat transfer coefficients (CHTCs) are required in practically all heat transport calculations and they depend on material and flow properties. CHTCs are generally not easily calculated analytically and are difficult to derive from experimental measurements. More often, they are determined using empirical correlations based on measurements of different geometry and flows with great costs. Recently, CFD was used to determine CHTCs on building surfaces [8, 16]. An advantage of this technique is that detailed information on the thermal flow field is available. CFD can be used to predict CHTCs, but the model must always be validated with experimental data in order to verify the accuracy of the solution.

1.1.3 Solidification under Natural Convection

Melting and Solidification are phase change processes in which a moving boundary separates the two phases. They are of importance in a lot of practical applications including purification of metals, welding and many other technologies. Phase change with pure heat conduction is rarely encountered in the real world. A phase change process is necessarily associated with temperature and/or concentration gradients in the liquid phase where convection arises under the action of buoyancy forces due to

these gradients. The convection flow can have a very significant influence on the phase change process. A number of researchers [11, 36, 69, 81] have reported that the convection affects not only the rate of melting or solidification but also the resulting structure and distribution of the solutes in the liquid phase of a multicomponent system.

Mathematical models for solving phase change with natural convection can be divided into two categories, the multi-domain formulation (transformed grid method) and the single-domain formulation (fixed grid method). In the multi-domain formulation, the governing equations are solved in each phase domain separately [24, 31, 90]. This approach requires a continuous update of the two domains due to the time dependent interface position. In the single-domain formulation, the governing equations are solved in the entire physical domain [68, 74, 91]. The main advantage of this approach is that the interface is not explicitly computed and the energy balance condition is automatically satisfied at the interface.

1.2 Contributions of this Thesis

This thesis makes several contributions to the field of Computational Multi-phase Flows. The major contributions are:

1. The two-phase incompressible flow solver, `interFoam`, has been modified to form a new solver, `interSEAFoam`. This new solver is based on the Shifted Eulerian Adaption (SEA) method to keep the droplet within the fixed computational domain.

2. The solver `interSEAFoam` was used to investigate the:

   (a) Deformation and breakup of drops in axisymmetric flows.

   (b) Deformation and breakup of drops in three dimensional symmetric flows.

3. The Taylor Analogy Breakup (TAB) model has been modified to account for the change in the cross-sectional area of the drop to predict the drop deformation more accurately. Numerical results from a drop in axisymmetric flows are compared with the TAB and the modified TAB models, and show better agreement with the modified TAB model. The types of breakup are found to be in good qualitative agreement with experimental observations.

4. Numerical simulations of three dimensional symmetric flows at different Weber numbers corresponding to different breakup regimes agree with experimental observations. The product drop size distribution of each breakup regime is quantified and is found to be consistent with experimental observations. This statistical information can be used to develop and improve spray models.

5. The feasibility and accuracy of calculating convective heat transfer coefficients using CFD was tested in two cases:

(a) Calculation of convective heat transfer coefficients along parallel flat plates in the laminar flow regime with imposed constant heat flux and with imposed constant wall temperature.

(b) Calculation of convective heat transfer coefficients along a cylinder wall in the laminar flow regime with imposed constant heat flux and with imposed constant wall temperature.

Comparisons of the results with reference values from the literature show good accuracy and performance of calculating convective heat transfer coefficients using CFD.

6. An enhanced enthalpy-porosity model [6] has been implemented into OpenFOAM® to form a new solver, `modPolyMeltFoam`. This enhanced enthalpy-porosity model takes different thermophysical properties of solid and liquid phases into account. The code was tested for two cases:

   (a) Pure natural convection of water in a cavity.

   (b) Solidification of water in a cavity.

   The performance of the code was evaluated on these two test cases. Comparisons with results from the literature show good agreement.

7. The `chtMultiRegionFoam` solver in OpenFOAM® has been modified to form a new solver, `modFluidFluidChtMultiRegionFoam`, to simulate a water drop solidifying in a cold airflow. Specifically, the part of the code for the fluid in the

original `chtMultiRegionFoam` is replaced by `icoFoam` coupled with the temperature equation, while the code for the solid in the original `chtMultiRegionFoam` is replaced by `modPolyMeltFoam`.

The codes are documented in the appendix.

Chapter 2

Computational Fluid Dynamics and Numerical Methods

Computational fluid dynamics (CFD) uses numerical methods to solve the fundamental nonlinear differential equations that describe fluid flows. There are many advantages of employing CFD, e.g., 1) CFD provides insight into flow mechanisms which are difficult to get from experiments; 2) CFD reduces time and costs greatly in new designs compared to experiments; 3) CFD makes it possible to analyze problems whose experiments are very difficult or dangerous to carry out; 4) CFD offers the possibility of studying systems under conditions over its limits.

Fluid flow is typically caused by external forces. These driving forces consist of surface

forces and body forces. Surface forces include viscous force, surface tension, etc. while body forces include gravity, aerodynamic drag, buoyancy force, etc.

2.1 Conservation Principles

In fluid dynamics, the general form of convection-diffusion equation in the Eulerian frame of reference takes the form

$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{\text{temporal derivative}} + \underbrace{\nabla\cdot(\rho\mathbf{v}\phi)}_{\text{convection term}} = \underbrace{\nabla\cdot(\rho\Gamma_\phi\nabla\phi)}_{\text{diffusion term}} + \underbrace{q_\phi(\phi)}_{\text{source term}}\ , \tag{2.1}$$

where ϕ is a general property and Γ_ϕ is the diffusion coefficient. Taking $\phi = 1$ in Eq. (2.1) leads to the equation for conservation of mass. Taking $\phi = \mathbf{v}$ in Eq. (2.1) leads to the equation for conservation of momentum. Taking $\phi = e$ in Eq. (2.1) leads to the equation of conservation of energy.

2.2 Numerical Methods

2.2.1 Finite Volume Method

The finite volume method (FVM) is a method for approximating partial differential equations in the form of algebraic equations [48]. In the FVM, the computational domain is divided into a finite number of non-overlapping control volumes (CVs) which completely cover the computational domain. The governing equations are integrated on each CV to get the integral form of the equations. The description of the FVM below follows in part that given in the thesis of Jasak [42].

Each control volume, except the ones adjacent to the boundaries, of a computational domain is a convex polytope bounded by a set of flat faces and each face is shared with only one neighboring control volume. For a CV, suppose 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, $\mathbf{n}_f$ is the outward unit normal vector to the face, N is a computational point of a neighboring CV, $\mathbf{d}_f$ is the displacement vector between P and N, and $\mathbf{r}_P$ is the displacement vector between the origin and P.

The coordinates of the centroid of the CV and the center of a face, $\mathbf{x}_P$ and $\mathbf{x}_f$, are

given by

$$\int_{V_P} (\mathbf{x} - \mathbf{x}_P)\, dV = \mathbf{0}, \tag{2.2}$$

$$\int_f (\mathbf{x} - \mathbf{x}_f)\, dS = \mathbf{0}. \tag{2.3}$$

The key step of the FVM is the integration of Eq. (2.1) over a CV yielding

$$\int_{V_P} \frac{\partial(\rho\phi)}{\partial t}\, dV + \int_{V_P} \nabla \cdot (\rho \mathbf{v}\phi)\, dV = \int_{V_P} \nabla \cdot (\rho \Gamma_\phi \nabla\phi)\, dV + \int_{V_P} q_\phi(\phi)\, dV. \tag{2.4}$$

After applying the Gauss divergence theorem, Eq. (2.4) becomes

$$\int_{V_P} \frac{\partial(\rho\phi)}{\partial t}\, dV + \int_{\partial V_P} \mathbf{n} \cdot (\rho \mathbf{v}\phi)\, dS = \int_{\partial V_P} \mathbf{n} \cdot (\rho \Gamma_\phi \nabla\phi)\, dS + \int_{V_P} q_\phi(\phi)\, dV, \tag{2.5}$$

where $\mathbf{n}$ is the outward unit normal vector to the face. Integrating Eq. (2.5) with respect to time t over a small interval yields

$$\int_t^{t+\Delta t} \left[\int_{V_P} \frac{\partial(\rho\phi)}{\partial t}\, dV + \int_{\partial V_P} \mathbf{n} \cdot (\rho \mathbf{v}\phi)\, dS - \int_{\partial V_P} \mathbf{n} \cdot (\rho \Gamma_\phi \nabla\phi)\, dS \right] dt = \int_t^{t+\Delta t} \int_{V_P} q_\phi(\phi)\, dV\, dt. \tag{2.6}$$

In the FVM, ϕ is assumed to have a linear variation both in space and time around the computational point P. This gives a second-order discretization method in space and time, which is accurate since Eq. (2.6) is a second-order integral equation. The

linear variations of ϕ are given by

$$\phi(\mathbf{x}) = \phi_P + (\mathbf{x} - \mathbf{x}_P) \cdot (\nabla\phi)_P, \tag{2.7}$$

$$\phi(t + \Delta t) = \phi^o + \Delta t \left(\frac{\partial \phi}{\partial t}\right)^o, \tag{2.8}$$

where $\phi_P = \phi(\mathbf{x}_P)$, $(\nabla\phi)_P = \nabla\phi(\mathbf{x}_P)$, $\phi^o = \phi(t)$ and $\left(\frac{\partial\phi}{\partial t}\right)^o = \frac{\partial\phi}{\partial t}(t)$.

The discretization methods in space and time are discussed next.

2.2.1.1 Discretization of Convection Term

Each CV is bounded by a number of faces, so the surface integral can be written as

$$\int_{\partial V_P} (\rho\mathbf{v}\phi \cdot \mathbf{n})\, dS = \sum_f \left(\int_f (\rho\mathbf{v}\phi \cdot \mathbf{n}_f)\, dS \right). \tag{2.9}$$

Applying the assumption of linear variation of ϕ around the point f, the term $\rho\mathbf{v}\phi$ in Eq. (2.9) can be written as

$$\rho\mathbf{v}\phi(\mathbf{x}) = (\rho\mathbf{v}\phi)_f + (\mathbf{x} - \mathbf{x}_f) \cdot (\nabla(\rho\mathbf{v}\phi))_f. \tag{2.10}$$

Therefore, the integral inside the sum in Eq. (2.9) is approximated as

$$\int_f (\rho \mathbf{v}\phi \cdot \mathbf{n}_f)\, dS = (\rho \mathbf{v}\phi)_f \cdot \int_f \mathbf{n}_f\, dS + (\nabla(\rho \mathbf{v}\phi))_f : \int_f (\mathbf{x} - \mathbf{x}_f)\mathbf{n}_f\, dS, \tag{2.11}$$

where : is the double dot product between two tensors. Assuming $\mathbf{n}_f$ is constant on face f, i.e., the face is a planar surface, and using Eq. (2.3), Eq. (2.9) becomes

$$\begin{aligned} \int_{\partial V_P} (\rho \mathbf{v}\phi \cdot \mathbf{n})\, dS &= \sum_f (\rho \mathbf{v}\phi)_f \cdot \mathbf{S} \\ &= \sum_f \mathbf{S} \cdot (\rho \mathbf{v})_f \phi_f \\ &= \sum_f F\phi_f, \end{aligned} \tag{2.12}$$

where $\mathbf{S} = \int_f \mathbf{n}_f\, dS$ is the area vector, and $F = \mathbf{S} \cdot (\rho \mathbf{v})_f$ is the convective mass flux through the face f. ρ, $\mathbf{v}$ and ϕ are found at the face f by interpolating from the values at the centroids. In the basic approach, a variate of interpolation schemes can be used for the convection term (see [62] for details). A weighted average is used to calculate ρ, $\mathbf{v}$ and ϕ at the face f as

$$\rho_f = b_f \rho_P + (1 - b_f)\rho_N, \tag{2.13}$$

$$\mathbf{v}_f = b_f \mathbf{v}_P + (1 - b_f)\mathbf{v}_N, \tag{2.14}$$

$$\phi_f = b_f \phi_P + (1 - b_f)\phi_N, \tag{2.15}$$

There are several methods to compute the weight factor b_f. The most common ones are given below.

1. **Central Differencing (CD)**

   The weight factor, b_f, in Eq. (2.13) - (2.15) is defined as

$$b_f = \frac{\overline{fN}}{\overline{PN}}, \tag{2.16}$$

   where $\overline{fN}$ is the distance between the face and the centroid point N in the neighbouring CV, $\overline{PN}$ is the distance between the centroid point P in the CV and the centroid point N in the neighbouring CV. This method is second-order accurate but sometimes makes the solution unbounded, i.e., ϕ can take values outside its physically meaningful range. More details are found in Chapter 14 of Hoffmanand and Frankel [38] and Chapter 4 of Wesseling [92].

2. **Full Upwind Differencing (UD)**

   The weight factor, b_f, in Eq. (2.13) - (2.15) is defined as

$$b_f = \begin{cases} 1, & \text{if } F \geq 0 \\ 0, & \text{if } F < 0 \end{cases} \tag{2.17}$$

   where $F = \mathbf{S} \cdot (\rho \mathbf{v})_f$ is the flux. The value of b_f in this method depends on the flux direction, therefore the solution is bounded and the method is stable.

However, this method is only first-order accurate because it uses the first-order backward differencing. More details are found in [42].

3. **Blended Differencing (BD)**

   This method is a combination of the CD and the UD, and it is defined as

$$\phi_f = (1 - k_f)(\phi_f)_{UD} + k_f(\phi_f)_{CD}, \tag{2.18}$$

   where $(\phi_f)_{UD}$ is the value from the UD, $(\phi_f)_{CD}$ is the value from the CD, and k_f is a blending factor between 0 and 1. The blending factor k_f controls how much numerical diffusion will be introduced. This method is developed to preserve the accuracy and the boundedness. More details are found in [42].

2.2.1.2 Discretization of Diffusion Term

By using a similar approach as above, the diffusion term is discretized as

$$\begin{aligned}\int_{\partial V_P} (\rho\Gamma_\phi \nabla\phi) \cdot \mathbf{n}\, dS &= \sum_f (\rho\Gamma_\phi \nabla\phi)_f \cdot \mathbf{S} \\ &= \sum_f (\rho\Gamma_\phi)_f \mathbf{S} \cdot (\nabla\phi)_f. \end{aligned} \tag{2.19}$$

If the computational mesh is orthogonal, i.e., vectors $\mathbf{d}$ and $\mathbf{S}$ shown in Fig. 2.1 are parallel, then the estimation for $\mathbf{S} \cdot (\nabla\phi)_f$ can be defined as

$$\mathbf{S} \cdot (\nabla\phi)_f = |\mathbf{S}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}. \tag{2.20}$$

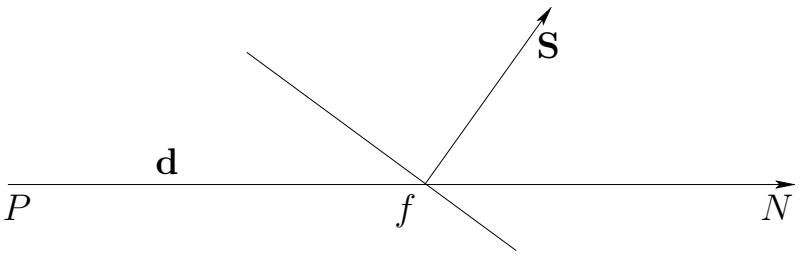


Figure 2.1: Vectors $\mathbf{S}$ and $\mathbf{d}$ on a non-orthogonal mesh.

If the computational mesh is non-orthogonal, i.e., vectors $\mathbf{d}$ and $\mathbf{S}$ shown in Fig. 2.1 are not parallel, then the estimation for $\mathbf{S} \cdot (\nabla\phi)_f$ can be defined as

$$\mathbf{S} \cdot (\nabla\phi)_f = \underbrace{\boldsymbol{\Delta} \cdot (\nabla\phi)_f}_{\text{orthogonal contribution}} + \underbrace{\mathbf{K} \cdot (\nabla\phi)_f}_{\text{non-orthogonal contribution}}, \tag{2.21}$$

where $\boldsymbol{\Delta}$ is parallel to the vector $\mathbf{d}$, and $\mathbf{S} = \boldsymbol{\Delta} + \mathbf{K}$. The orthogonal contribution can be approximated by the estimation in Eq. (2.20), and the non-orthogonal contribution can be approximated by estimating $(\nabla\phi)_f$ using the weighted average as

$$(\nabla\phi)_f = b_f(\nabla\phi)_P + (1 - b_f)(\nabla\phi)_N, \tag{2.22}$$

where b_f is defined in Eq. (2.16) and $(\nabla\phi)_P$ can be approximated using the second-order approximation to the Gauss divergence theorem as

$$\int_{V_P} \nabla\phi \, dV = \int_{\partial V_P} \phi \cdot \mathbf{n} \, dS, \tag{2.23}$$

$$(\nabla\phi)_P V_P = \sum_f \left(\int_f \phi \cdot \mathbf{n}_f \, dS \right), \tag{2.24}$$

$$(\nabla\phi)_P = \frac{1}{V_P} \sum_f \mathbf{S}\phi_f. \tag{2.25}$$

There are several ways to find the vectors $\mathbf{\Delta}$ and $\mathbf{K}$. Two common approaches are given below.

1. **Minimum Correction Approach**

   $\mathbf{K}$ is chosen to be orthogonal to the vector $\mathbf{\Delta}$, as shown in Fig. 2.2, to keep the non-orthogonal contribution as small as possible. $\mathbf{\Delta}$ can be written as

   $$\mathbf{\Delta} = \frac{\mathbf{d} \cdot \mathbf{S}}{\mathbf{d} \cdot \mathbf{d}} \mathbf{d}. \tag{2.26}$$

   $\mathbf{\Delta}$ is the orthogonal projection of $\mathbf{S}$ onto $\mathbf{d}$, so that $\mathbf{K}$ has the minimal distance between $\mathbf{S}$ and $\mathbf{d}$.

2. **Over-relaxed Approach**

   $\mathbf{\Delta}$ is defined as

   $$\mathbf{\Delta} = \frac{\mathbf{S} \cdot \mathbf{S}}{\mathbf{d} \cdot \mathbf{S}} \mathbf{d}. \tag{2.27}$$

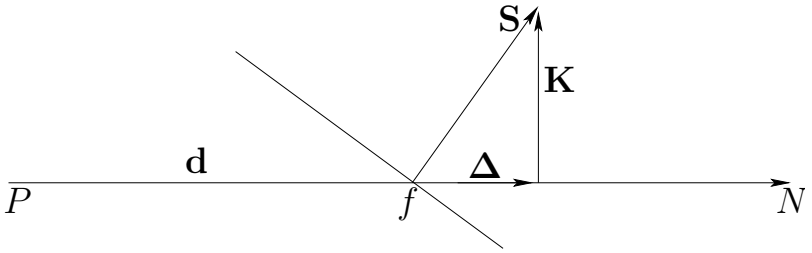


Figure 2.2: Vectors $\mathbf{\Delta}$ and $\mathbf{K}$ in the minimum correction approach.

Substituting Eq. (2.27) in Eq. (2.21) yields

$$\mathbf{S} \cdot (\nabla\phi)_f = |\mathbf{\Delta}| \frac{\phi_N - \phi_P}{|\mathbf{d}|} + \mathbf{K} \cdot (\nabla\phi)_f, \tag{2.28}$$

where $\frac{\phi_N - \phi_P}{|\mathbf{d}|}$ is the magnitude of the orthogonal projection of $(\nabla\phi)_f$ onto $\mathbf{\Delta}$. From Eq. (2.27), $|\mathbf{\Delta}|$ increases with the increase of non-orthogonality (decrease of the denominator), which indicates that the importance of the term in ϕ_P and ϕ_N is caused to increase with increase of non-orthogonality. The decomposition of $\mathbf{S}$ is shown in Fig. 2.3. More details are found in [42].

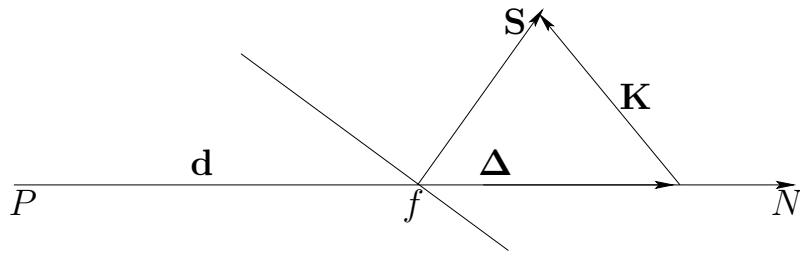


Figure 2.3: Vectors $\mathbf{\Delta}$ and $\mathbf{K}$ in the over-relaxed approach.

2.2.1.3 Discretization of Source Term

The source terms are the terms that cannot be written as temporal contribution, convection, or diffusion. These terms need to be linearized as

$$q_\phi(\phi) = q_u + q_P\phi, \tag{2.29}$$

where q_u and q_P can also depend on ϕ. The integral form of the source term can be approximated as

$$\begin{aligned}\int_V q_\phi(\phi)\, dV &= (q_u + q_P\phi)_P V_P \\ &= q_u V_P + q_P V_P \phi_P.\end{aligned} \tag{2.30}$$

2.2.1.4 Temporal Discretization

Applying the above spatial discretization methods, Eq. (2.6) can be written as

$$\begin{aligned}\int_t^{t+\Delta t} \left[\left(\frac{\partial(\rho\phi)}{\partial t} \right)_P V_P + \sum_f F\phi_f - \sum_f (\rho\Gamma_\phi)_f \mathbf{S} \cdot (\nabla\phi)_f \right] dt \\ = \int_t^{t+\Delta t} (q_u V_P + q_P V_P \phi_P)\, dt,\end{aligned} \tag{2.31}$$

where $\left(\frac{\partial(\rho\phi)}{\partial t}\right)_P V_P$ is the approximation to $\int_{V_P} \frac{\partial(\rho\phi)}{\partial t}\, dV$ by using the Gauss one-point centroidal rule. In order to fully discretize Eq. (2.31), the following approximations are used

$$\left(\frac{\partial(\rho\phi)}{\partial t}\right)_P = \frac{\rho_P^n \phi_P^n - \rho_P^o \phi_P^o}{\Delta t}, \tag{2.32}$$

$$\int_t^{t+\Delta t} \phi(t)\, dt = [w\phi^o + (1-w)\phi^n]\Delta t, \tag{2.33}$$

where $\phi^n = \phi(t+\Delta t)$, $\phi^o = \phi(t)$, and w is a constant. Using Eq. (2.32) and Eq. (2.33) in Eq. (2.31), and assuming that ρ and Γ_ϕ do not change with time, Eq. (2.31) becomes

$$\begin{aligned}
\frac{\rho_P\phi_P^n - \rho_P\phi_P^o}{\Delta t} V_P &+ \sum_f [(1-w)F\phi_f^n + wF\phi_f^o] \\
&- \sum_f [(1-w)(\rho\Gamma_\phi)_f \mathbf{S}\cdot(\nabla\phi)_f^n + w(\rho\Gamma_\phi)_f \mathbf{S}\cdot(\nabla\phi)_f^o] \\
&= q_u V_P + (1-w)q_P V_P \phi_P^n + w q_P V_P \phi_P^o.
\end{aligned} \tag{2.34}$$

The first-order explicit Euler method is obtained by letting $w = 1$, the first-order implicit Euler method is obtained by letting $w = 0$, and the second-order Crank-Nicholson method is obtained by letting $w = \frac{1}{2}$. The values of ϕ_f and $(\nabla\phi)_f$ depend on the values of ϕ in the neighbouring CV, therefore for any CV with centroid $\mathbf{x}_P$, Eq. (2.34) can be written as

$$a_P\phi_P^n + \sum_N a_N\phi_N^n = R_P, \tag{2.35}$$

where a_P includes the contribution from all terms corresponding to ϕ_P^n, i.e., the temporal derivative, convection and diffusion terms as well as the linear part of the source term. a_N include the corresponding terms in each of the neighbouring CVs. R_P includes the parts of the temporal derivative, convection and diffusion terms corresponding to the previous time-level as well as the constant part of the source term.

Assembling the fully discretized equations, Eq. (2.35), for all CVs yields a system of algebraic equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.36}$$

in each time step, where $\mathbf{A}$ is a sparse matrix containing a_P and a_N, $\mathbf{x}$ is the vector of unknown ϕ in all CVs, and $\mathbf{b}$ contains the source terms, R_P. This system is linear, i.e. $\mathbf{A}$ is constant, if the original continuous convection-diffusion equation is linear in ϕ, or if terms have been linearized.

The momentum equation is a convection-diffusion equation with the pressure gradient as a source term, and it can be discretized using the above discretization methods. However, there are some difficulties to solve the equation, e.g., (1) there are nonlinear terms such as the convection term $\nabla \cdot (\rho \mathbf{v}\mathbf{v})$ and the viscous stress tensor τ for a non-Newtonian fluid; (2) the continuity equation and the momentum equation are coupled and a treatment is required to handle the pressure-velocity coupling.

2.2.2 Pressure-Velocity Coupling

For incompressible fluids, the mass (continuity) and the momentum equation read

$$\nabla \cdot \mathbf{v} = 0, \tag{2.37}$$

$$\frac{\partial(\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) - \nabla \cdot \eta(\dot{\gamma})(\nabla \mathbf{v} + \nabla \mathbf{v}^T) = -\nabla p, \tag{2.38}$$

where η is the dynamic viscosity of fluids and $\dot{\gamma}$ is the shear rate. For Newtonian fluids, $\eta(\dot{\gamma})$ is a constant. For inelastic non-Newtonian fluids, there are different models of $\eta(\dot{\gamma})$. In this thesis, only Newtonian fluids are considered.

The difficulty of nonlinear terms such as $\nabla \cdot (\rho \mathbf{v}\mathbf{v})$ is solved by linearization in order to reduce the computational time. The nonlinear term $\nabla \cdot (\rho \mathbf{v}\mathbf{v})$ is linearized as

$$\begin{aligned}
\int_{V_P} \nabla \cdot (\rho \mathbf{v}\mathbf{v})\, dV &= \int_{\partial V_P} (\rho \mathbf{v}\mathbf{v} \cdot \mathbf{n})\, dS \\
&= \sum_f \mathbf{v}_f (\rho \mathbf{v})_f^o \cdot \mathbf{S} \\
&= \sum_f F^o \mathbf{v}_f \\
&= a_P \mathbf{v}_P + \sum_N a_N \mathbf{v}_N,
\end{aligned} \tag{2.39}$$

where $\mathbf{v}^o$ is the velocity from the previous time step and F^o is the flux from the

previous time step.

The difficulty of the coupling of pressure and velocity is solved by applying the Rhie and Chow procedure [72], in which a pressure equation is derived from the continuity and momentum equation. Specifically, the continuity equation, Eq. (2.37), can be discretized as

$$\int_{V_P} \nabla \cdot \mathbf{v} \, dV = \int_{\partial V_P} \mathbf{v} \cdot \mathbf{n} \, dS = \sum_f \mathbf{S} \cdot \mathbf{v}_f = 0. \tag{2.40}$$

The momentum equation, Eq. (2.38), can be semi-discretized as

$$a_P \mathbf{v}_P = \mathbf{H}(\mathbf{v}) - \nabla p, \tag{2.41}$$

where

$$\mathbf{H}(\mathbf{v}) = -\sum_N a_N \mathbf{v}_N + \frac{\mathbf{v}^o}{\Delta t}. \tag{2.42}$$

From Eq. (2.41),

$$\mathbf{v}_P = \frac{\mathbf{H}(\mathbf{v})}{a_P} - \frac{1}{a_P} \nabla p. \tag{2.43}$$

The velocity at faces of the CV can be interpolated as

$$\mathbf{v}_f = \left(\frac{\mathbf{H}(\mathbf{v})}{a_P} \right)_f - \left(\frac{1}{a_P} \nabla p \right). \tag{2.44}$$

Substituting Eq. (2.44) into Eq. (2.40) yields

$$\sum_f \mathbf{S} \cdot \left(\frac{1}{a_P} \nabla p \right)_f = \sum_f \mathbf{S} \cdot \left(\frac{\mathbf{H}(\mathbf{v})}{a_P} \right)_f . \tag{2.45}$$

The pressure gradient can be found by interpolating the pressure in each CV to the faces of the CV. Hence Eq. (2.41) can be written as

$$a_P \mathbf{v}_P = \mathbf{H}(\mathbf{v}) - \sum_f \mathbf{S} p_f. \tag{2.46}$$

The flux F can be found as

$$F = \mathbf{S} \cdot \left[\left(\rho \frac{\mathbf{H}(\mathbf{v})}{a_P} \right)_f - \left(\rho \frac{1}{a_P} \nabla p \right)_f \right]. \tag{2.47}$$

Eq. (2.45) and Eq. (2.46) are the discrete pressure and velocity equations, respectively. In order to solve them, the following three predictor-corrector methods are often to be used.

† **The Semi-Implicit Method for Pressure-Linked Equation (SIMPLE)**

The SIMPLE algorithm is used to solve steady-state flows. To increase the diagonal dominance of the matrix resulting from the discrete momentum equation, an under-relaxed form has been obtained by adding an artificial term to

both sides of Eq. (2.35) as

$$\frac{a_P}{\alpha_{\mathbf{v}}}\mathbf{v}_P^n + \sum_N a_N \mathbf{v}_N^n = R_P + \frac{1-\alpha_{\mathbf{v}}}{\alpha_{\mathbf{v}}} a_P \mathbf{v}_P^o, \tag{2.48}$$

where $\alpha_{\mathbf{v}}$ is the velocity under-relaxation factor ($0 < \alpha_{\mathbf{v}} \leq 1$).

The SIMPLE algorithm for Newtonian fluids can be summarized as following:

1. Start with an initial value of pressure p^*, which is either an initial guess or the value from the previous iteration.
2. Solve for the velocity $\mathbf{v}^*$ from the under-relaxed momentum equation, Eq. (2.48), using the guessed pressure p^* to find R_P. This step is called the momentum predictor.
3. Calculate the mass flux at the faces of CVs, $F^* = \mathbf{S} \cdot \left(\rho \frac{\mathbf{H}(\mathbf{v}^*)}{a_P}\right)_f$.
4. Solve Eq. (2.45) to find the new value of pressure p^{**}.
5. Correct the mass flux at the faces of CVs using p^{**} in Eq. (2.47), $F = F^* - \left(\rho \frac{1}{a_P} \nabla p^{**}\right)_f \cdot \mathbf{S}$.
6. Apply an under-relaxation factor $0 < \alpha_p \leq 1$ to find the new value of pressure $p^{new} = p^* + \alpha_p(p^{**} - p^*)$.
7. Calculate the corrected velocity $\mathbf{v}^{new}$ using Eq. (2.43) and the new value of pressure p^{new}.
8. Test for convergence, and repeat the steps by setting the new value of pressure p^{new} as the initial value if not converged.

It is desired to repeat step 4 for a number of iterations if there are non-orthogonal cells in a computational mesh. In OpenFOAM®, the parameter `nNonOrthogonalCorrectors` is specified in the file `<case>/system/fvSolution`. The recommended values for the under-relaxation factors according to [66] are $\alpha_p = 0.2$ and $\alpha_{\mathbf{v}} = 0.8$. Test for convergence is done by checking the residuals of the discrete pressure and velocity equations, Eq. (2.45) and Eq. (2.46), respectively. If the Euclidean norm of each residual is within a specified tolerance, then the SIMPLE procedure stops. In OpenFOAM®, the parameters are found in the file `<case>/system/fvSolution` under the name of `residualControl`.

† **Pressure Implicit with Splitting of Operators (PISO)**

The PISO algorithm was developed originally for computations of unsteady compressible flows [41]. Later it was further developed for steady calculations and for incompressible flows. The PISO algorithm uses more than one pressure corrector step. The PISO algorithm for Newtonian fluids can be summarized as following:

1. Calculate the velocity $\mathbf{v}^*$ using Eq. (2.46) and pressure p^* from the previous step.
2. Calculate the mass flux at the faces of CVs, $F^* = \mathbf{S} \cdot \left(\rho \frac{\mathbf{H}(\mathbf{v}^*)}{a_P}\right)_f$.
3. Solve Eq. (2.45) to find the new value of pressure p^{**}.

4. Correct the mass flux at the faces of CVs using p^{**} in Eq. (2.47), $F = F^* - \left(\rho \frac{1}{a_P} \nabla p^{**}\right)_f \cdot \mathbf{S}$.

5. Apply an under-relaxation factor $0 < \alpha_p \leq 1$ to find the new value of pressure $p^{new} = p^* + \alpha_p(p^{**} - p^*)$.

6. Calculate the corrected velocity $\mathbf{v}^{new}$ using Eq. (2.43) and the new value of pressure p^{new}.

7. Repeat steps 2 - 6 several (`nCorrectors` in OpenFOAM®) more times.

The parameters are found in the file `<case>/system/fvSolution`.

† **Merged PISO-SIMPLE (PIMPLE)**

The PIMPLE algorithm combines the SIMPLE and the PISO algorithms together, and is good to be used in transient calculations. The PIMPLE algorithm for Newtonian fluids can be summarized as following:

1. Calculate the velocity $\mathbf{v}^*$ using Eq. (2.48) and pressure p^* from the previous step.

2. Calculate the mass flux at the faces of CVs, $F^* = \mathbf{S} \cdot \left(\rho \frac{\mathbf{H}(\mathbf{v}^*)}{a_P}\right)_f$.

3. Solve Eq. (2.45) to find the new value of pressure p^{**}.

4. Correct the mass flux at the faces of CVs using p^{**} in Eq. (2.47), $F = F^* - \left(\rho \frac{1}{a_P} \nabla p^{**}\right)_f \cdot \mathbf{S}$.

5. Apply an under-relaxation factor $0 < \alpha_p \leq 1$ to find the new value of pressure $p^{new} = p^* + \alpha_p(p^{**} - p^*)$.

6. Calculate the corrected velocity $\mathbf{v}^{new}$ using Eq. (2.43) and the new value of pressure p^{new}.

7. Repeat steps 2 - 6 several (`nCorrectors` in OpenFOAM®) more times.

8. Test for convergence, and repeat the steps at most several (`nOuterCorrectors` in OpenFOAM®) more times if not converged.

Test for convergence is controlled by `residualControl` as in the SIMPLE algorithm. If `nOuterCorrectors` = 1, then the PIMPLE will operate in the PISO mode. The `nCorrectors`, `nOuterCorrectors` and `residualControl` parameters are all found in the file `<case>/system/fvSolution`.

2.2.3 Linear Solvers

In this subsection some of the numerical methods are described to solve the linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.49}$$

resulting from discretizing the equations.

There are two families of methods: direct methods and iterative methods. In this subsection, some of the basic iterative methods are discussed. The methods can be preconditioned using several techniques (see [62] for details).

1. **Gauss Seidel Method**

   For the linear system Eq. (2.49), if $\mathbf{A}$ is a symmetric positive definite matrix, the Gauss Seidel method uses the decomposition

$$\mathbf{A} = \mathbf{D} - \mathbf{U} - \mathbf{L}, \tag{2.50}$$

   where $\mathbf{D}$ is a diagonal matrix containing the diagonal elements of $\mathbf{A}$, $-\mathbf{U}$ is the upper triangular matrix of $\mathbf{A}$, $-\mathbf{L}$ is the lower triangular matrix of $\mathbf{A}$. The linear system Eq. (2.49) can be written as

$$(\mathbf{D} - \mathbf{L})\mathbf{x} = \mathbf{U}\mathbf{x} + \mathbf{b}. \tag{2.51}$$

   The Gauss Seidel method uses the value of $\mathbf{x}$ from the previous iteration on the right hand side of Eq. (2.51) to calculate the new value of $\mathbf{x}$:

$$\mathbf{x}_{k+1} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}\mathbf{x}_k + (\mathbf{D} - \mathbf{L})^{-1}\mathbf{b}, \tag{2.52}$$

   where $\mathbf{x}_{k+1}$ is the new value of $\mathbf{x}$, $\mathbf{x}_k$ is the value of $\mathbf{x}$ from the previous iteration.

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

   For the linear system Eq. (2.49), if $\mathbf{A}$ is a symmetric positive definite matrix, then solving the linear system Eq. (2.49) is equivalent to minimizing the quadratic function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$. The solution is updated iteratively

by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \tag{2.53}$$

where the step length α_k and the search direction $\mathbf{p}_k$ are defined below. The CG method chooses the set of search directions $\{\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_n\}$ such that the set is $\mathbf{A}$-conjugate, i.e., $\mathbf{p}_i^T \mathbf{A} \mathbf{p}_j = 0, \quad i \neq j$.

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}, \tag{2.54}$$

$$\mathbf{p}_k = \mathbf{r}_k + \beta_k \mathbf{p}_{k-1}, \tag{2.55}$$

where the residual $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}$, and $\beta_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_{k-1}^T \mathbf{r}_{k-1}}$. The CG method starts with an initial residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ and calculates the initial guess for the search direction $\mathbf{p}_0 = \mathbf{r}_0$. Then the following steps are repeated until the residual gets below a specified tolerance:

(i) Calculate the step length $\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$.

(ii) Calculate $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$.

(iii) Calculate the new residual $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$.

(iv) Calculate $\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$.

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

3. **Bi-Conjugate Gradient (BiCG) Method**

Unlike the CG method, the BiCG method is applicable for a non-symmetric

matrix $\mathbf{A}$. This method constructs two sets of search directions $\{\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_n\}$ for $\mathbf{A}$ and $\{\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_n\}$ for $\mathbf{A^T}$, and the two sets are mutually orthogonal, i.e., $\mathbf{q}_i^T \mathbf{A} \mathbf{p}_i = 0$. The BiCG method starts with an initial guess $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ and sets $\mathbf{p}_0 = \mathbf{q}_0 = \mathbf{s}_0 = \mathbf{r}_0$, and then repeats the following steps until convergence:

(i) Calculate the step length $\alpha_k = \frac{\mathbf{s}_k^T \mathbf{r}_k}{\mathbf{q}_k^T \mathbf{A} \mathbf{p}_k}$.

(ii) Calculate $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$.

(iii) Calculate the new residual of $\mathbf{A}$, $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$.

(iv) Calculate the new residual of $\mathbf{A}^T$, $\mathbf{s}_{k+1} = \mathbf{s}_k - \alpha_k \mathbf{A}^T \mathbf{q}_k$.

(v) Calculate $\beta_k = \frac{\mathbf{s}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{s}_k^T \mathbf{r}_k}$.

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

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

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

For the linear system Eq. (2.49), if the approximated solution is $\mathbf{x}_h$, then the error $\mathbf{e} = \mathbf{x} - \mathbf{x}_h$ assuming $\mathbf{x}$ is the exact solution to the linear system. The residual is defined as $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}_h$. The error $\mathbf{e}$ and the residual $\mathbf{r}$ satisfy

$$\begin{aligned} \mathbf{A}\mathbf{e} &= \mathbf{A}(\mathbf{x} - \mathbf{x}_h) \\ &= \mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{x}_h \\ &= \mathbf{r}. \end{aligned} \tag{2.56}$$

The GAMG method solves Eq. (2.56) on a coarse grid first and then interpolates the solution to the fine grid. This method is applied by using a restriction matrix $\mathbf{T}$ which transfers a vector from the find grid to the coarse grid, and an interpolation matrix $\mathbf{P}$ which returns the vector to the find grid.

The GAMG method repeats the following steps until convergence:

(i) Solve for $\mathbf{x}_h$ from $\mathbf{A}\mathbf{x} = \mathbf{b}$ through a few iterations.

(ii) Calculate the residual on the coarse grid, $\mathbf{r}_c = \mathbf{T}\mathbf{r}$.

(iii) Calculate the error $\mathbf{e}_c$ on the coarse grid by $\mathbf{r}_c$ from $\mathbf{A}_c\mathbf{e}_c = \mathbf{r}_c$.

(iv) Calculate $\mathbf{e}_h = \mathbf{P}\mathbf{e}_c$ by interpolating the error to the find grid.

(v) Add the error to the approximated solution, $\mathbf{x}_{new} = \mathbf{x}_h + \mathbf{e}_h$.

Chapter 3

Droplet Deformation and Breakup

3.1 Deformation and Breakup Computations of Drops in Axisymmetric Flows and Comparisons with the Taylor Analogy Breakup Model

Drop deformation and breakup are transient processes and it is important to study their long time behavior. However, in an Eulerian framework, one of the difficulties is that the computational domain needs to be large enough in order to simulate the entire process. This results in huge computational costs. In order to keep the drop within the fixed computational domain, the Shifted Eulerian Adaption (SEA) method

has been developed. In this approach, the location of the drop is adjusted every N time steps such that the center of mass of the liquid phase remains fixed in the domain. This shifting mechanism has been implemented into the `interFoam` solver of the open source software OpenFOAM® [63] which was also used for the CFD simulations.

In this section, we investigate the behavior of the Taylor drop oscillator using axisymmetric CFD simulations. The simulations are used to validate a modification which accounts for the change in the cross-sectional area of the deforming drop. It is found that this modification leads to improved drop deformation prediction. Further, the drop breakup is simulated for the bag breakup, the stamen breakup and the stripping breakup regimes. These simulations show that the breakup initiation times are in good agreement with experimental data, and that the breakup behavior in the respective breakup regimes compares well with observations reported in the literature.

3.1.1 TAB Model

The TAB model is a classic method for calculating droplet breakup, and this method is based on Taylor's analogy [87] between an oscillating and distorting droplet and a

spring mass system. The equation governing a damped, forced oscillator [64] is

$$m\ddot{x} = F - kx - d\dot{x}, \tag{3.1}$$

where m is the mass of the drop, k is the spring constant, d is the damping coefficient, and x is the displacement of the equator of the drop from its equilibrium position. In Eq. (3.1), F is the external force, $-kx$ is the restoring force, and $-d\dot{x}$ is the damping force. The physical dependencies of the coefficients in Eq. (3.1) are

$$\frac{F}{m} = C_F \frac{\rho_g v^2}{\rho_l r}, \tag{3.2}$$

$$\frac{k}{m} = C_k \frac{\sigma}{\rho_l r^3}, \tag{3.3}$$

$$\frac{d}{m} = C_d \frac{\mu_l}{\rho_l r^2}, \tag{3.4}$$

where ρ_g and ρ_l are the gas and liquid densities, v is the relative velocity between the gas and the droplet, r is the droplet radius, σ is the gas-liquid surface tension, and μ_l is the liquid dynamic viscosity. Drop breakup occurs if $x > C_b r$, where values for the dimensionless constants C_F, C_k, C_d and C_b are determined by comparing experimental and theoretical results [64], and are found to be $C_F = 1/3$, $C_k = 8$, $C_d = 5$, and $C_b = 1/2$. It has been reported by Grover et al. [29] that for gasoline sprays, $C_k = 0.6$ resulted in better agreement with measurements. Therefore, the need for model parameter calibration for different liquids is clearly needed. By nondimensionalizing

x by $C_b r$, letting

$$y = \frac{x}{C_b r} \tag{3.5}$$

and using Eqs. (3.2), (3.3) and (3.4) in Eq. (3.1) gives

$$\ddot{y} = \frac{C_F}{C_b}\frac{\rho_g}{\rho_l}\frac{v^2}{r^2} - \frac{C_k \sigma}{\rho_l r^3} y - \frac{C_d \mu_l}{\rho_l r^2}\dot{y}, \tag{3.6}$$

with breakup occurring if $y > 1$.

3.1.2 Modified TAB Model

The drag of a drop moving at relative velocity v in the gas is

$$F = \frac{1}{2} C_D \rho_g A v^2, \tag{3.7}$$

where C_D is the drag coefficient, ρ_g is the gas density, and A is the cross-sectional area of the drop. The TAB model is based on the assumption that the drop's cross-section is fixed. In reality, during deformation, the cross-sectional area changes, and here we assume a circular shape with radius $r + x$, where x is the increase in the cross-sectional radius. Under this assumption, the drag of a drop becomes

$$F = \frac{1}{2} C_D \rho_g \pi (r + x)^2 v^2. \tag{3.8}$$

Comparing the drag coefficient and dimensionless constants in the TAB model yields

$$C_D = \frac{8}{3} C_F, \tag{3.9}$$

where $C_F = 1/3$. Note that in general, the drag coefficient depends on the Reynolds number and the shape of the immersed object [52]. However, in the modified TAB model we assume that C_D is fixed as in the original TAB model. After nondimensionalization, the equation for the modified TAB model becomes

$$\ddot{y} = \frac{C_F}{C_b}(1 + C_b y)^2 \frac{\rho_g}{\rho_l} \frac{v^2}{r^2} - \frac{C_k \sigma}{\rho_l r^3} y - \frac{C_d \mu_l}{\rho_l r^2} \dot{y}. \tag{3.10}$$

By comparing Eqs. (3.6) and (3.10), the only difference is $(1 + C_b y)^2$ in the forcing term, which makes the modified TAB model a non-linear differential equation.

3.1.3 Problem Description

The problem of droplet deformation and breakup is studied by simulating a water droplet of 1 mm diameter in airflow. The transport properties at room temperature for water are the kinematic viscosity $\nu_l = 1.004 \times 10^{-6}\ m^2/s$ and the density $\rho_l = 998.2\ kg/m^3$ while the transport properties at room temperature for air are $\nu_g = 1.511 \times 10^{-5}\ m^2/s$ and $\rho_g = 1.205\ kg/m^3$. The surface tension between water and air is $\sigma = 0.07286\ kg/s^2$. The deformation and breakup of the water droplet is

simulated using the computational setup shown schematically in Fig. 3.1 for the three dimensional simulations and in Fig. 3.2 for the axisymmetric simulations.

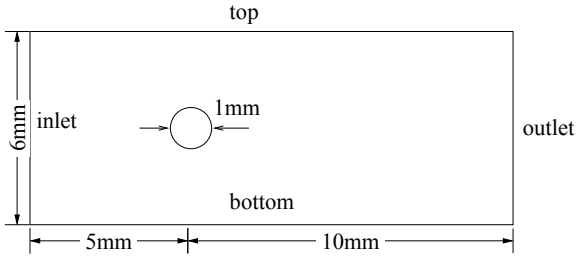


Figure 3.1: Three dimensional computational domain (front view).

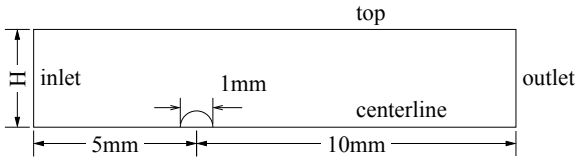


Figure 3.2: Axisymmetric computational domain.

Initially, the water droplet is at rest and the airflow is set at a constant velocity $\mathbf{v}_g$. On the inlet boundary the velocity is set to $\mathbf{v}_g$ and the pressure has zero normal gradient. On the outlet boundary, the velocity is set to zero normal gradient and the pressure is fixed to zero. The top boundary is given far-field conditions, that is, the velocity is set to $\mathbf{v}_g$ and the normal pressure gradient is zero.

3.1.4 Mathematical Model and Numerical Methods

The problem is formulated using the volume of fluid (VOF) method. The governing equations are the mass and momentum balance equations for the two-phase flow,

which hold over the entire computational domain, and are given by

$$\nabla \cdot \mathbf{v} = 0, \tag{3.11}$$

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \nabla \cdot (\mu \dot{\gamma}) + \sigma \kappa \delta_s \mathbf{n}, \tag{3.12}$$

where $\mathbf{v}$ is the velocity, p is the pressure, $\dot{\gamma} = \nabla \mathbf{v} + (\nabla \mathbf{v})^T$ is the rate-of-strain tensor, and μ and ρ are the dynamic viscosity and density, respectively. The last term on the right-hand side of Eq. (3.12) is the continuum surface force (CSF) and is nonzero only on the interface, as is indicated by the Dirac delta function $\delta_s = \delta(\mathbf{x} - \mathbf{x}_s)$, where $\mathbf{x}_s$ is a point on the interface.

In the VOF method, the interface between two phases is described by a scalar function α called the volume fraction function. This function takes the value $\alpha = 0$ in cells that contain only the continuous phase, i.e., air in this chapter, and the value $\alpha = 1$ in cells that contain only the disperse phase, i.e., water in this chapter. In cells where the interface is located, $0 < \alpha < 1$ and represents the volume fraction of dispersed phase in the cell. The volume fraction function α is governed by

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\mathbf{v}\alpha) = 0. \tag{3.13}$$

Equation (3.13) is used to reformulate the CSF term in Eq. (3.12) as $\sigma \kappa \nabla \alpha$ [26] and to represent the fluid transport properties as $\mu(\alpha) = \alpha \mu_l + (1 - \alpha)\mu_g$ and $\rho(\alpha) =$

$\alpha\rho_l + (1-\alpha)\rho_g$.

The above equations are solved using a modified version of the interFoam solver of OpenFOAM® [63], in which a second-order finite volume method (FVM) is used for spatial discretization, an implicit first-order method is used for temporal discretization, and interface compression is used for the volume fraction equation. See [18] for a complete description and evaluation of the standard interFoam solver.

In order to keep the drop within the fixed computational domain, the SEA method is used and the standard interFoam solver had to be modified such that the location of the drop is adjusted every N time steps. More precisely, the center of mass of the liquid phase is calculated every N time steps ($N = 100$ in this study) over the entire domain according to the formula

$$\bar{\mathbf{x}} = \frac{\int_V \alpha \mathbf{x}\, dV}{\int_V \alpha\, dV}, \tag{3.14}$$

where the denominator represents the total volume of the liquid phase. Note that Eq. (3.14) assumes that density is constant. This modified solver is called `interSEAFoam`. If the center of mass of the liquid phase has moved more than the distance of one cell, then all field values in the computational domain, including $\mathbf{v}$, p and α, are shifted (or mapped) back by one cell, while maintaining the boundary conditions. This shift introduces a small error at the inlet and outlet boundaries,

which, however, is corrected in the next time step through the SIMPLE iterations in that time step. In order to guarantee that there are no additional errors introduced around the drop, a uniform grid has been used.

3.1.5 Axisymmetry Assumption Justification

A three dimensional and an axisymmetric simulation, as described above, have been performed for the air inlet speed of $v_g = 20\ m/s$, corresponding to $We = 6.56$. The same mesh resolution was used for both domains. Figure 3.3 shows that the drop deformation contours agree well between the three dimensional and the axisymmetric simulations near the maximal deformation at $t = 1.8\ ms$. The discrepancy is at most within one cell, i.e., at the same level of mesh resolution. Figure 3.4 shows that the velocity and pressure fields of the three dimensional simulation are fully symmetric. Asymmetries in the flow fields, as for example the von Karman vortex street, are not observed. In fact, a two dimensional simulation of a flow over a non-deforming cylinder for the same flow conditions showed that it takes at least 6 ms until the symmetry starts to break and the von Karman vortices start to develop. Therefore, since the drop is deforming, asymmetries do not have time to evolve in the time frame under consideration, and the flow remains symmetric.

The discussion in the previous paragraph shows that the difference between the three

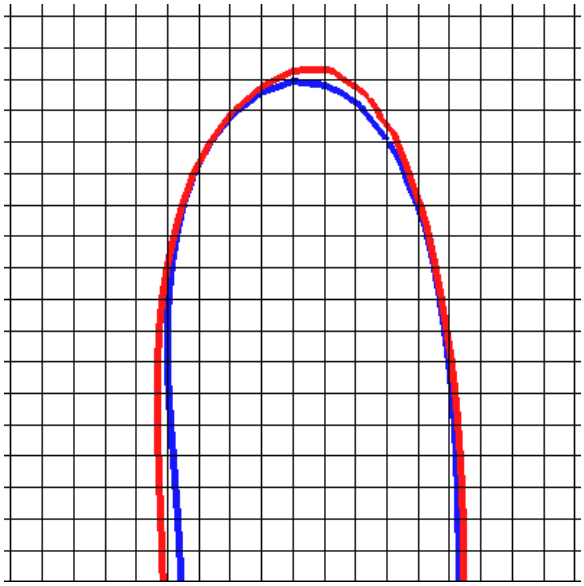
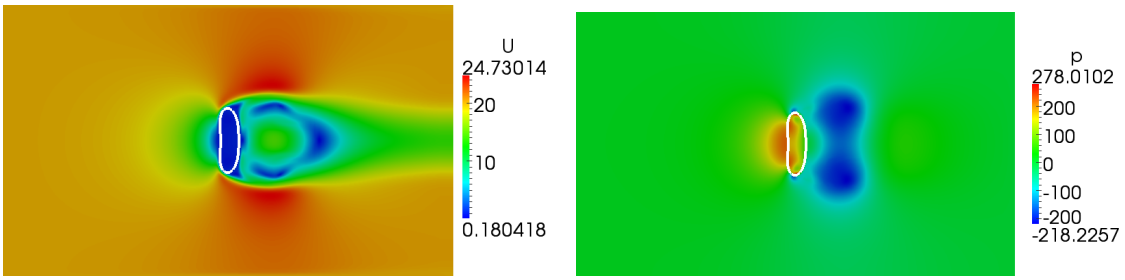


Figure 3.3: Drop deformation contours for the $We = 6.56$ case at $t = 1.8\,ms$; red: three dimensional, green: axisymmetric. (The cross-sectional radius is 0.71 mm.)



(a) (b)

Figure 3.4: (a) Velocity magnitude field on a cross-sectional plane through the center of the drop, (b) pressure field on a cross-sectional plane through the center of the drop for the three dimensional $We = 6.56$ case at $t = 1.8\ ms$.

dimensional and the axisymmetric simulations is negligible. Also, since the flow field is highly symmetric, it appears to be justified to continue the further investigations using axisymmetric simulations. This greatly decreases the computational cost.

3.1.6 Axisymmetric Simulations

3.1.6.1 Domain Independence Study

As discussed above, the computational domain, shown in Fig. 3.2, is an axisymmetric cylinder with far-field boundary conditions on the top boundary. In order to make sure that these boundary conditions do not influence the drop deformation behavior, a domain independence study has been conducted to determine the minimum acceptable height of the domain. Three domains with heights 2 mm, 3 mm and 4.5 mm have been used to simulate the case with 20 m/s inlet velocity corresponding to $We = 6.56$ and no breakup. The drop deformations of these computations are shown in Fig. 3.5. As is seen, the drop deformations for the domain heights corresponding to 3 mm and 4.5 mm are almost identical. Consequently, the domain height of 3 mm has been chosen for all subsequent computations.

3.1.6.2 Mesh Independence Study

The computational meshes consist of six blocks, three in the flow direction and two in the radial direction. A front view of the standard mesh is shown in Fig. 3.6. The block which contains the drop has cells of uniform length and height. The uniform

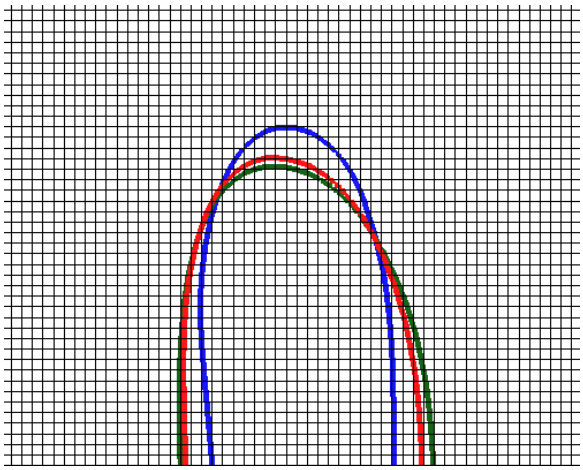


Figure 3.5: Drop deformation contours for the axisymmetric $We = 6.56$ case at $t = 1.8\ ms$; blue: $H = 2\ mm$, red: $H = 3\ mm$, green: $H = 4.5\ mm$.

cell size is important to guarantee mass conservation of the liquid when using the SEA method to compensate for the drop movement. The cells of the outer blocks are graded geometrically by a factor of eight such that the largest cells are located at the boundaries. This enables a finer grid near the drop where the velocity and pressure gradients are larger, and still yields sufficient mesh resolution at the boundaries.

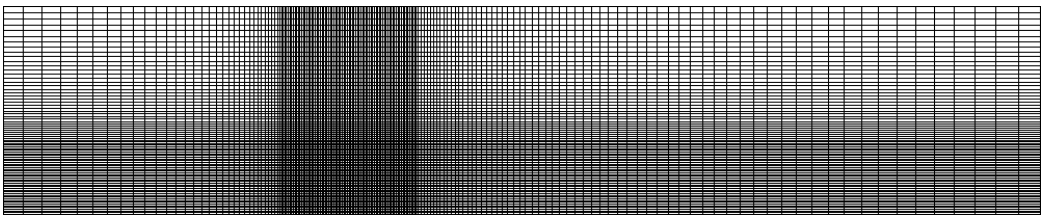


Figure 3.6: Axisymmetric computational mesh (length = 15 mm, height = 3 mm).

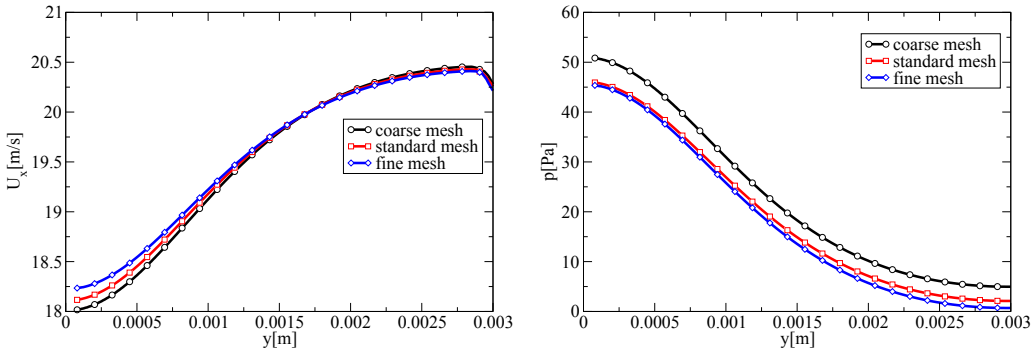
Three meshes have been considered for the mesh independence study. In each mesh the number of cells are changed by a factor of 1.5 in each direction, resulting in a change of number of cells by a factor of 2.25. This leads to meshes with a total

of $18,400$, $41,400$, and $92,880$ cells for the coarse, the standard and the fine mesh, respectively. The smallest cells are the uniform cells in the block that contains the liquid mass, and the smallest cell sizes for each mesh, together with the total number of cells, are listed in Table 3.1.

Table 3.1
Meshes used in the axisymmetric simulations

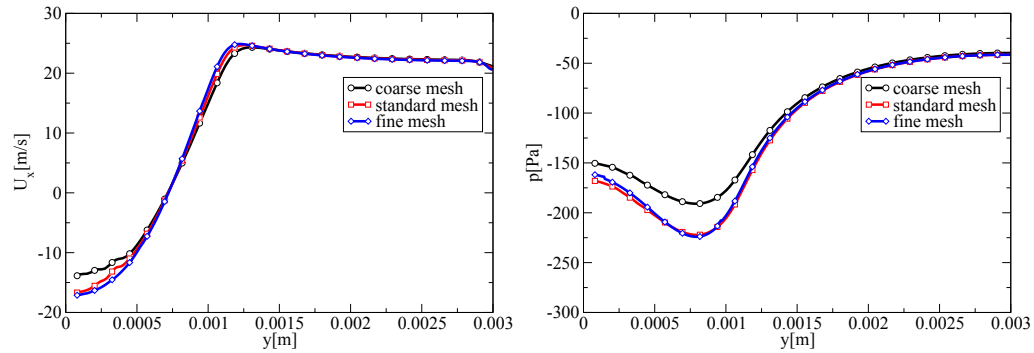
Mesh	Number of cells	Smallest cell size $[mm]$
Coarse	18400	0.0250×0.0250
Standard	41400	0.0167×0.0167
Fine	92880	0.0111×0.0111

The mesh independence investigation has been conducted for the case with 20 m/s inlet velocity corresponding to $We = 6.56$ and no breakup. The criteria for judging the mesh independence are the velocity x-component, U_x, and the pressure, p, along the y-direction at 1.5 mm in front of and behind the drop at the simulation time of $t = 1.8$ ms when the deformation is largest. As is seen in Figs. 3.7 and 3.8, the curves corresponding to the standard and the fine mesh are closer than the curves between the coarse and the standard mesh. From these figures we can conclude that sufficient mesh independence has been achieved with the standard mesh. Therefore, all subsequent simulations have been performed with the standard mesh.



(a) (b)

Figure 3.7: (a) Velocity x-component, (b) pressure along the line $x = -1.5\ mm$ for the $We = 6.56$ case at $t = 1.8\ ms$.



(a) (b)

Figure 3.8: (a) Velocity x-component, (b) pressure along the line $x = 1.5\ mm$ for the $We = 6.56$ case at $t = 1.8\ ms$.

3.1.6.3 Drop Deformation Compared with Taylor Oscillator

The performance of the TAB model and the modified TAB model has been evaluated using detailed CFD simulations. More specifically, axisymmetric CFD simulations have been performed for the case with 20 m/s inlet velocity (corresponding to $We = 6.56$ and no breakup) and compared with the behavior of the TAB model (see Eq. (3.6)) and the modified TAB model (see Eq. (3.10)). Note that the results of the two TAB models were obtained by means of Mathematica® [93]. The quantities that have been compared are the maximum amplitude of the deformation (measured from the centerline) and the period of the first oscillation starting with the undeformed drop. These values are listed in Table 3.2. Clearly, the modified TAB model is in better agreement with the axisymmetric CFD simulations than the original TAB model. This indicates that Taylor Oscillators with appropriate modifications are accurate one dimensional breakup models to predict drop deformation.

Table 3.2
Drop deformation, normalized drop deformation y, and first period for the $We = 6.56$ case

Model	Deformation $[mm]$	y [-]	Period $[ms]$
TAB	0.635	0.54	2.9
Modified TAB	0.700	0.80	3.5
CFD	0.710	0.84	3.4

3.1.6.4 Drop Breakup Compared with Taylor Oscillator

In this section, the breakup initiation times (defined below) and the associated drop deformations obtained from the axisymmetric CFD simulations are compared with the corresponding quantities of the TAB and the modified TAB models for the three Weber numbers $We = 10.33$, $We = 20.09$ and $We = 81.04$. Two dimensionless breakup times are typically measured in experiments: the initiation time T_{ini} and the total breakup time T_{tot}. According to Guildenbecher et al. [30] the initiation time is defined as the moment when the intact but deformed drop resembles an oblate spheroid, while the total breakup time is defined as the moment when the disintegrated drop and all its fragments have reached a stable state and no further breakup occurs. Time is nondimensionalized according to Ranger and Nicholls [70] by

$$T = t\frac{v}{\epsilon^{0.5} d_0}, \tag{3.15}$$

where T is the dimensionless time, t is the dimensional time, ϵ is the drop to ambient density ratio, v is the relative velocity between the drop and the ambient air, and d_0 is the diameter of the initially spherical drop. The dimensional and dimensionless initiation times are determined for the CFD simulations and are listed in Table 3.3. The numerical results are compared with the experimental observations of Guildenbecher et al. [30] where $T_{ini} \approx 1.5$. This value has been obtained under the assumption that

the dimensionless initiation time is independent of We and Oh when $Oh < 0.1$. It can be seen from Table 3.3 that the dimensionless initiation time is slightly decreasing with increasing Weber numbers. However, given the uncertainty of determining t_{ini} for the CFD simulations as well as in the experiments, the agreement with the experimental value of $T_{ini} = 1.5$ can be considered good. Note that the initiation and breakup times do not make sense for the Taylor oscillators because they cannot predict breakup by themselves. In fact, as is discussed in the TAB model description above, breakup is determined artificially by specifying a normalized breakup deformation criterion.

Table 3.3
Dimensional and dimensionless initiation times for the CFD simulations and normalized drop deformations y for the TAB, the modified TAB and the CFD simulations

	Times		Normalized deformations y		
We	t_{ini} $[ms]$	T_{ini} $[-]$	TAB	Modified TAB	CFD
10.33	1.6	1.39	0.83	1.54	1.69
20.09	1.1	1.34	1.44	2.42	2.83
81.04	0.5	1.22	1.78	2.64	3.00

At the initiation time $t = t_{ini}$, the normalized drop deformations, y, are calculated for the TAB model, the modified TAB model and the CFD simulations. The results are listed in Table 3.3. Clearly, the modified TAB model is in better agreement with the CFD simulation than the original TAB model, which indicates that the modified TAB model is better suited to predict drop deformations and the onset of drop breakup. One of the reasons for the discrepancy between the results from the TAB models

and CFD simulations is that the TAB models only represent one oscillation mode, and in reality there are many such modes [3]. It is discussed that the TAB models only keep track of the fundamental mode corresponding to the lowest order spherical zonal harmonic whose axis is aligned with the relative velocity vector between the drop and the ambient air. For larger Weber numbers, higher order modes become more significant for the breakup process, thus the discrepancy becomes larger, as is seen in Table 3.3.

3.1.6.5 Drop Breakup Simulations

The drop breakup simulations have been performed at different inlet air velocities of 25 m/s, 35 m/s and 70 m/s. This results in respective Weber numbers of $We = 10.33$, $We = 20.09$ and $We = 81.04$, which are associated with the bag breakup, the stamen breakup and the stripping breakup regimes, respectively. The results of these simulations are shown in Figs. 3.9, 3.10 and 3.11 at different times. In all three cases, the drop is first flattened before it starts its characteristic breakup. Measuring the radial extent of this deformation from the centerline, it turns out that the drop breakup criterion of the TAB model, $y > 1$, is satisfied. In fact, the maximum drop deformation before the onset of breakup is determined to be $y = 1.8$ for the $We = 10.33$ case, $y = 3.6$ for the $We = 20.09$ case, and $y = 2.4$ for the $We = 81.04$ case. Therefore, the breakup criterion depends on the Weber number and should be

adjusted accordingly in the TAB models.

Because these are axisymmetric simulations, the cross-sections of the liquid after breakup shown in Figs. 3.9, 3.10 and 3.11 have to be interpreted as toroidal-shaped rings. This is of course not realistic, but it is interesting to note that there is qualitative agreement of these ring-structures with experimentally observed product drops.

Figure 3.9 shows the bag breakup, where the bag starts forming at $t = 1.8\ ms$ and is held together with the belt. The bag then breaks up at $t = 2.1\ ms$ and only the belt is left. The stamen breakup is shown in Fig. 3.10. At $t = 1.4\ ms$ a bag is formed between the stem in the center and the outside belt. The last frame in this figure illustrates the breakup of the bag and the remaining center stem together with the outside belt.

Finally, the stripping breakup is illustrated in Fig. 3.11. At $t = 0.6\ ms$, the thin sheet starts to shed off droplets at the periphery. This droplet shedding is nicely illustrated at $t = 0.7\ ms$. Actually, because these are axisymmetric simulations, the "droplets" are cross-sections of toroidal-shaped rings. This is of course not realistic, but it is interesting to note that there is good agreement of these ring cross-sections with experimentally observed product drops, as reported in the literature, e.g., Pilch and Erdman [67], Faeth et al. [23] and Guildenbecher et al. [30].

(a) $t = 0\,ms$

(b) $t = 1.2\,ms$

(c) $t = 1.8\,ms$

(d) $t = 2.1\,ms$

Figure 3.9: Time sequence of bag breakup for $We = 10.33$.

3.1.7 Summary and Conclusions

Axisymmetric CFD simulations have been conducted to verify the deformation and breakup behavior of a drop in an air stream as predicted by the TAB model and as observed in experiments. In order to justify the axisymmetry assumption, a three dimensional simulation has been performed for a non-breaking drop at $We = 6.56$

(a) $t = 0\,ms$

(b) $t = 1.0\,ms$

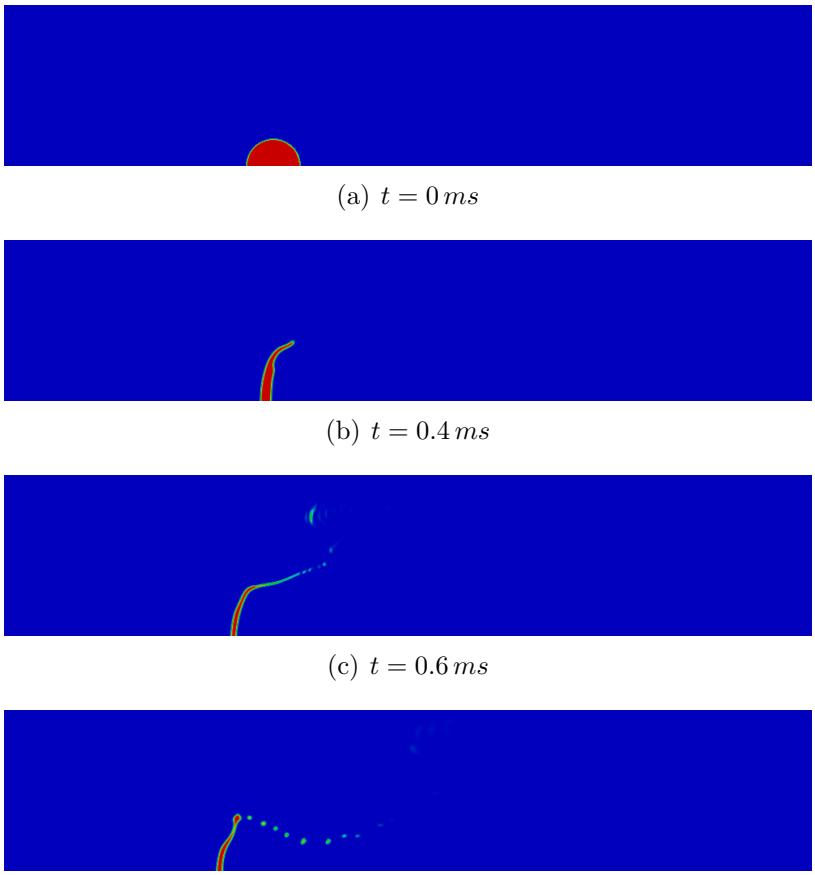
(c) $t = 1.4\,ms$

(d) $t = 1.5\,ms$

Figure 3.10: Time sequence of stamen breakup for $We = 20.09$.

and compared with a corresponding axisymmetric computation. Comparison between the drop deformation contours from the two simulations, and the fact that the three dimensional flow field was highly symmetric, indicates that the axisymmetry assumption is justified. This greatly decreased the computational cost of the remaining computations.

For the axisymmetric CFD simulations, a domain independence study has been conducted for the $We = 6.56$ case, where no breakup occurs, to determine the minimum



(a) $t = 0\,ms$

(b) $t = 0.4\,ms$

(c) $t = 0.6\,ms$

(d) $t = 0.7\,ms$

Figure 3.11: Time sequence of sheet-thinning breakup for $We = 81.04$.

acceptable height of the domain such that the far-field boundary conditions do not influence the drop deformation behavior. Then a mesh independence study has been performed for the same case by comparing the velocity x-component and the pressure along the y-direction at 1.5 mm in front of and behind the drop when the deformation is largest.

The original TAB model has been modified to account for the change in the aerodynamic drag due to the drop deformation. It is assumed that the cross-section is

a disk whose radius is changing according to the drop deformation, while the drag coefficient is kept constant. These TAB models have been compared with axisymmetric CFD simulations for a non-breaking drop at $We = 6.56$. It was found that the amplitude of the drop deformation and the period of the first drop oscillation are in good agreement with the modified TAB model. Further, these TAB models have been compared with axisymmetric CFD simulations for the bag breakup, the stamen breakup and the stripping breakup regimes at $We = 10.33$, $We = 20.09$ and $We = 81.04$, respectively. It was found that the initiation time for each breakup mode is comparable with experimental observations, and the drop deformation at the initiation time is in good agreement with the modified TAB model. By comparing results from these TAB models and those from axisymmetric CFD simulations, one can conclude that Taylor Oscillators are accurate one dimensional drop deformation models which can predict the onset of drop breakup.

Drop breakup has been studied by means of axisymmetric simulations at Weber numbers associated with the bag breakup, the stamen breakup and the stripping breakup regimes. These simulations show good qualitative agreement with the breakup behavior reported from experiments.

3.2 Deformation and Breakup Computations of Drops in Three Dimensional Symmetric Flows and Comparisons with Experimental Observations

This section is organized as follows. First, the problem description and solution approach is presented. This is followed by a fully three dimensional simulation at $We = 6.56$ for a deforming drop without breakup. The purpose of this simulation is to demonstrate that there is a high degree of symmetry in the flow and that a symmetry assumption can be made in order to reduce computational costs. The subsequent simulations are then performed for a one-quarter cross-section of the three dimensional domain, assuming an appropriate symmetric flow field. The symmetric simulations are performed for different Weber numbers which correspond to the bag breakup, the stamen breakup and the stripping breakup regimes, and the computational results are compared with experimental observations. Finally, the drop size distributions of each simulated breakup are fitted by different statistical distributions and the statistics of drops after breakup is analyzed and discussed. The analysis of product drop size distributions can be used to further develop breakup models together with product drop size distribution models.

3.2.1 Problem Description and Solution Approach

The problem considered here is the same as that described in Section 3.1.3, except for the computational domain. Specifically, the problem of droplet deformation and breakup is studied by simulating a water droplet of 1 mm diameter in airflow. The transport properties at room temperature for water are the kinematic viscosity $\nu_l = 1.004 \times 10^{-6}\ m^2/s$ and the density $\rho_l = 998.2\ kg/m^3$ while the transport properties at room temperature for air are $\nu_g = 1.511 \times 10^{-5}\ m^2/s$ and $\rho_g = 1.205\ kg/m^3$. The surface tension between water and air is $\sigma = 0.07286\ kg/s^2$. The computational domain is illustrated in Fig. 3.12. This domain resembles a one-quarter cross-section (when viewed in the main flow direction) of a flow over a sphere, with symmetry planes at the back and the bottom. The one-quarter droplet is located in the center between inlet and outlet.

Initially, the water droplet is at rest and the airflow is set at a constant velocity $\mathbf{v}_g$. On the inlet boundary the velocity is set to $\mathbf{v}_g$ and the pressure has zero normal gradient. On the outlet boundary, the velocity is set to zero normal gradient and the pressure is fixed to zero. On the far-field boundaries the velocities are set to $\mathbf{v}_g$ and the normal pressure gradient is zero. Symmetry boundary conditions are applied on the symmetry planes.

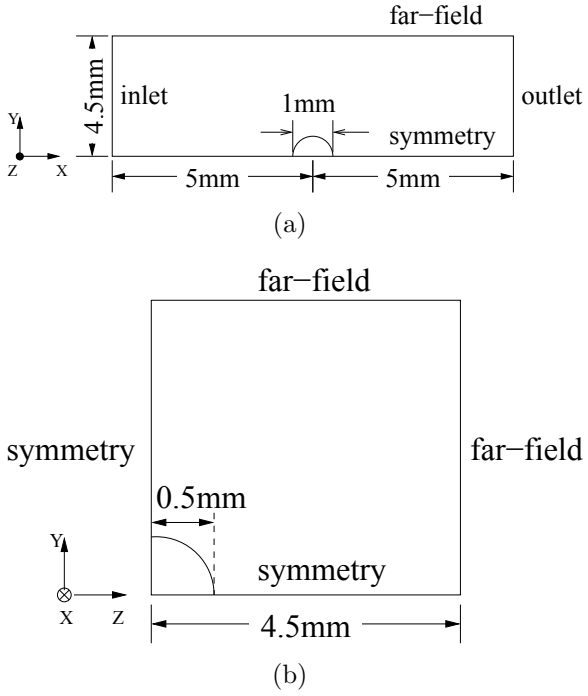


Figure 3.12: Three dimensional symmetric, one-quarter cross-section computational domain (top: front view, bottom: cross-sectional view).

The modified VOF solver, `interSEAFoam`, described in Section 3.1.4 is used to solve the current two-phase flow problem. This solver is a modified version of OpenFOAM's `interFoam` solver in which the Shifted Eulerian Adaption (SEA) method was incorporated (see Section 3.1.4 for details).

3.2.2 Symmetry Assumption Justification

In this section, we justify the reduction of the computational domain to one quarter of the full three dimensional domain. A simulation was performed on the full three

dimensional domain and on the quarter three dimensional domain, each for an airspeed $v_g = 20\ m/s$, corresponding to $We = 6.56$ and no breakup. The same mesh resolution was used for both simulations. Figure 3.13 shows that the drop deformation contours agree well between the three dimensional and the three dimensional symmetric simulations near the maximal deformation at $t = 1.8\ ms$. Figure 3.14 shows that the velocity and pressure fields of the three dimensional simulation are fully symmetric about both the plane $z = 0\ mm$ and the plane $y = 0\ mm$. As discussed in Section 3.1.5, asymmetries in the flow fields, for example the von Karman vortex street, do not have time to evolve in the time frame under consideration, since the drop is deforming. Therefore, the flow remains symmetric.

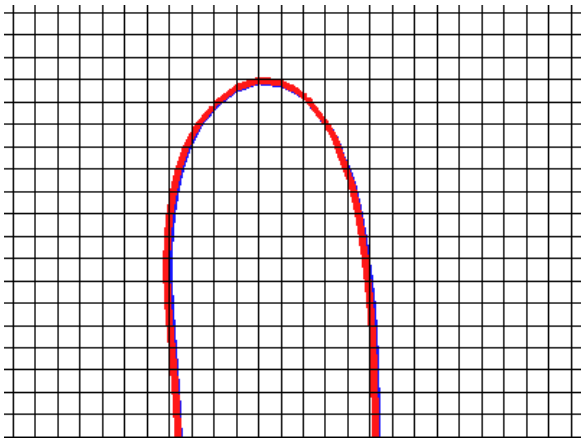


Figure 3.13: Drop deformation contours for $We = 6.56$ at $t = 1.8\ ms$; red: three dimensional symmetric, blue: three dimensional.

In summary, the difference between the fully three dimensional and the three dimensional symmetric simulations are negligible, and the flow field remains symmetric.

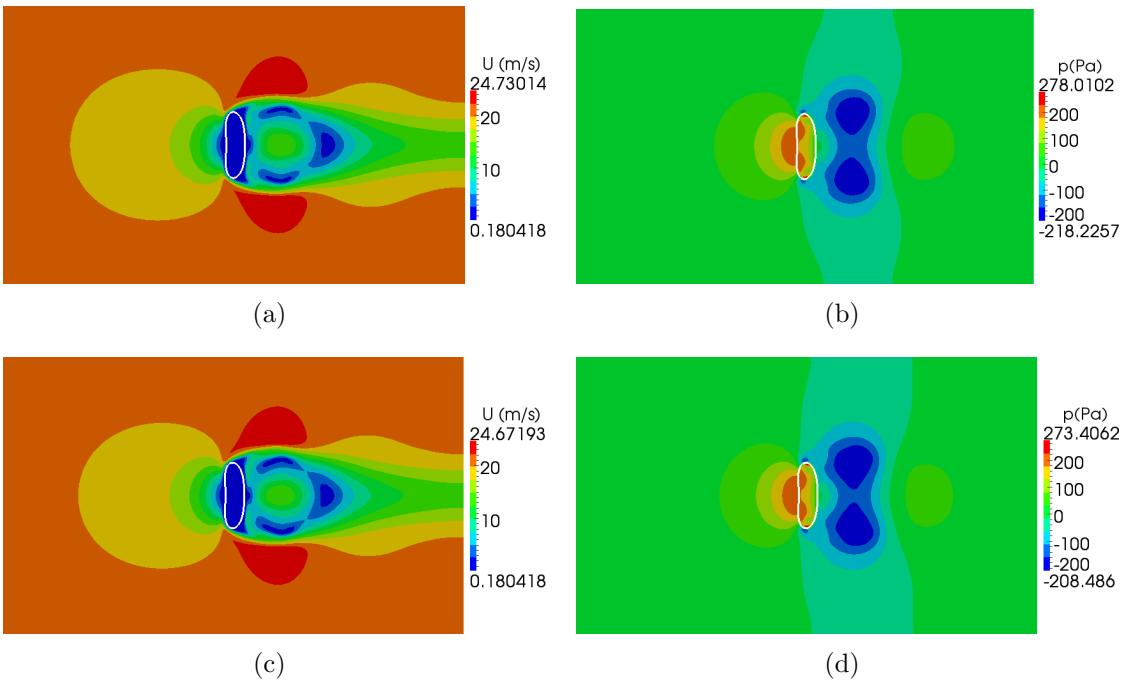


Figure 3.14: (a) Velocity magnitude on the plane $z = 0\ mm$, (b) pressure on the plane $z = 0\ mm$, (c) velocity magnitude on the plane $y = 0\ mm$, (d) pressure on the plane $y = 0\ mm$ for the three dimensional $We = 6.56$ case at $t = 1.8\ ms$.

Therefore, it is justified to continue the further investigations using three dimensional symmetric simulations. This greatly decreases the computational costs. The relative CPU times for the fully three dimensional and the three dimensional symmetric simulations are listed in Table 3.4.

Table 3.4
Relative CPU times for the fully three dimensional and the three dimensional symmetric simulations for $We = 6.56$ from $t = 0\ ms$ to $t = 1.8\ ms$

Simulation	Relative CPU time
fully three dimensional	9.78
three dimensional symmetric	1

3.2.3 Three Dimensional Symmetric Simulations

As discussed in Section 3.1.6.1, the minimum acceptable height of the computational domain is 3 mm. Because after breakup the product droplets tend to spread out over a large cross-section, the larger domain height of 4.5 mm has been chosen for all subsequent three dimensional symmetric computations.

3.2.3.1 Mesh Independence Study

The computational mesh consists of three blocks arranged consecutively in the main flow direction, as shown in Fig. 3.15. All three blocks have cells of uniform width (z-direction) and height (y-direction). The middle block has cells of uniform length (x-direction). The uniform cell size of the middle block is important to guarantee mass conservation of the liquid when using the SEA method to compensate for the drop movement. The cell length of the outer blocks are graded geometrically by a factor of eight such that the largest cells are located at the boundaries. This enables a finer grid near the drop(s) where the velocity and pressure gradients are larger, and still yields sufficient mesh resolution at the boundaries.

Three meshes have been considered for the mesh independence study. In each mesh the number of cells in each direction is changed by a factor of 1.5. This leads to meshes

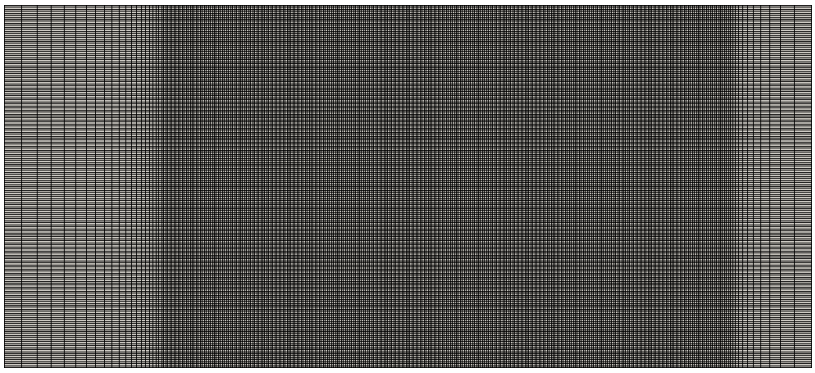
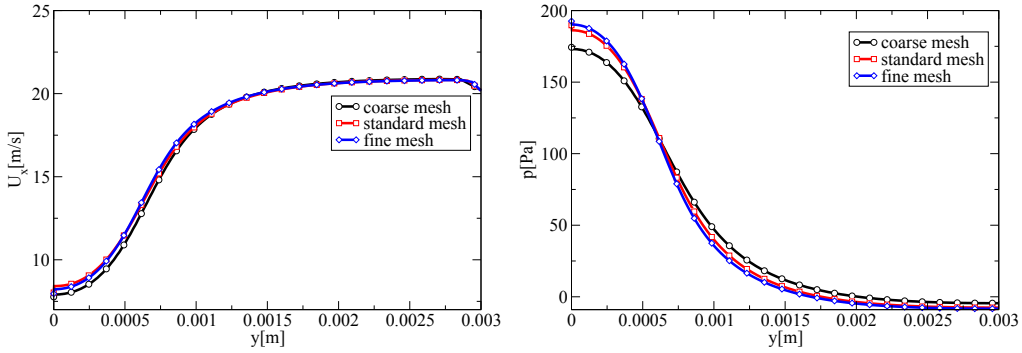


Figure 3.15: Three dimensional symmetric computational mesh (front view, length = 10 mm, height = 4.5 mm, the horizontal direction is the x-axis, the vertical direction is the y-axis, and the outward direction is the z-axis).

with a total of 3,004,800, 10,141,200 and 34,226,550 cells for the coarse, the standard and the fine mesh, respectively. The smallest cells are in the middle block and the smallest cell sizes for each mesh, together with the total number of cells, are listed in Table 3.5. The mesh independence investigation has been conducted for the case with 20 m/s inlet velocity corresponding to $We = 6.56$ and no breakup. The criteria for judging the mesh independence are the velocity x-component, U_x, and the pressure, p, along the y-direction at 0.5 mm in front of and behind the drop on the symmetry plane $z = 0$ mm at the simulation time $t = 1.8$ ms when the deformation is largest. As is seen in Figs. 3.16 and 3.17, the curves corresponding to the standard and the fine mesh are closer than the curves corresponding to the coarse and the standard mesh. From these figures we can conclude that sufficient mesh independence has been achieved with the standard mesh. Therefore, all subsequent simulations have been performed with the standard mesh.

Table 3.5
Meshes used in the three dimensional symmetric simulations

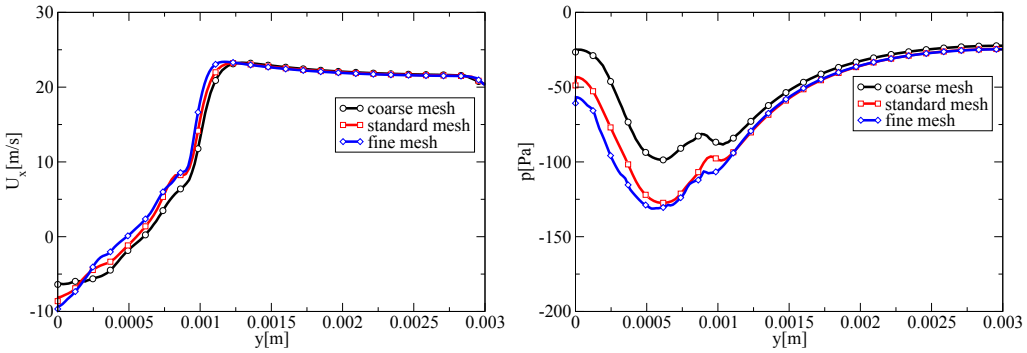
Mesh	Number of cells	Smallest cell size $[mm]$
Coarse	3,004,800	$0.0375 \times 0.0375 \times 0.0375$
Standard	10,141,200	$0.0250 \times 0.0250 \times 0.0250$
Fine	34,226,550	$0.0167 \times 0.0167 \times 0.0167$



(a) (b)

Figure 3.16: (a) Velocity x-component, (b) pressure along the line $x = -0.5\ mm$ on the plane $z = 0\ mm$ for $We = 6.56$ at $t = 1.8\ ms$.

Three dimensional symmetric simulations have been performed for different Weber numbers by adjusting the inlet airspeed to 30 m/s, 35 m/s and 70 m/s. This results in respective Weber numbers of 14.87, 20.09 and 81.04, which are associated with the bag breakup, the stamen breakup and the stripping breakup regimes, respectively. The numerical results of different regimes agree well with experimental observations, as is discussed in more detail below. All subsequent visualizations of the drop(s) are represented by reflecting the three dimensional symmetric simulation results around the symmetry planes $z = 0\ mm$ and $y = 0\ mm$ into fully three dimensional drop(s).



(a) (b)

Figure 3.17: (a) Velocity x-component, (b) pressure along the line $x = 0.5\ mm$ on the plane $z = 0\ mm$ for $We = 6.56$ at $t = 1.8\ ms$.

The CPU times for the three dimensional symmetric simulations for the bag, the stamen and the stripping breakup regimes are listed in Table 3.6. These CPU times have been obtained by multiplying the CPU times reported by the computations with the number of cores used. Therefore, these CPU times reflect non-parallel computations.

Table 3.6
CPU times for the three dimensional symmetric simulations for the bag, the stamen and the stripping breakup regimes

We	Simulation time $[ms]$	CPU time $[hr]$
14.87	3.9	2081
20.09	2.6	2120
81.04	1.2	2226

3.2.3.2 Bag Breakup ($We = 14.87$)

Chou and Faeth [13] show that the bag breakup regime of a drop consists of four

stages. In the first stage, the initially spherical drop deforms into an oblate spheroid with a disc shape. In the second stage, a hollow bag is formed and attached to a toroidal rim (also referred to as the belt). In the third stage, the bag breaks up into drops leaving behind the rim. In the fourth stage, the rim breaks up into drops caused by Rayleigh-Plateau instability. All four stages of the bag breakup regime are well captured in the simulation and are shown in Fig. 3.18. The flattening of the initially spherical drop is shown in Figs. 3.18(a) - 3.18(c), the formation of the bag is shown in Fig. 3.18(d), the breakup of the bag is shown in Figs. 3.18(e) and 3.18(f), and the breakup of the rim is shown in Figs. 3.18(g) and 3.18(h). The temporal evolution of the bag breakup is in good agreement with the experimental observations of Dai and Faeth [14] (see Fig. 3.19, where t/t^* is the dimensionless time calculated using Eq. (3.16)). Note that symmetry is not maintained in the experiments, especially after breakup. This is to be expected due to the random nature of the true phenomena and imperfect experimental conditions. Similar experimental observations have been reported by other authors including Gelfand [28], Chou and Faeth [13], Josephand et al. [43], Park et al. [65].

3.2.3.3 Stamen Breakup ($We = 20.09$)

The stamen breakup is simulated for a relative liquid-gas velocity of 35 m/s which corresponds to $We = 20.09$. As for the bag breakup regime, the initially spherical drop

flattens due to the high speed airstream (see Figs. 3.20(a) - 3.20(d)). Subsequently a bag with a stamen forms, as is shown in Fig. 3.20(e). Then rupture occurs at the bag membrane (see Fig. 3.20(f)). As the rupture widens, the bag membrane around the rupture splits up and the bag breaks up into drops, leaving behind a toroidal rim which is connected to the stamen by thread-like structures (see Fig. 3.20(g)). The stamen, the rim and the thread-like structures are elongated while bigger nodes form at the stamen and at the tips of the rim (see Fig. 3.20(h)). As the breakup proceeds (caused by Rayleigh-Plateau instabilities), the nodes from the stamen and the rim get pinched off (see Fig. 3.20(i)). Subsequently, both the stamen and the rim break up into drops (see Fig. 3.20(j)). The temporal evolution of the stamen breakup is in good agreement with the experimental observations of Dai and Faeth [14] (see Fig. 3.21, where t/t^* is the dimensionless time calculated using Eq. (3.16)). Similar experimental observations have been reported by other authors including Hirahara and Kawahashi [37], Gelfand [28], Josephand et al. [43].

3.2.3.4 Stripping Breakup ($We = 81.04$)

The stripping breakup is simulated for a relative liquid-gas velocity of 70 m/s which corresponds to $We = 81.04$. The high speed airflow induces large inertial forces to overcome the restoring effect of surface tension, which results in a backward-facing shell (see Figs. 3.22(a) - 3.22(c)). Rupture occurs at the periphery of the shell and

causes the breakup of the shell (see Fig. 3.22(d)). The high shear initiates the formation of ligaments due to Kelvin-Helmholtz instabilities at the periphery of the shell before the drop is flattened completely (see Fig. 3.22(e)). The ligaments are then stretched in the flow direction and smaller drops are pinched off from the free ends of the ligaments due to Rayleigh-Plateau instabilities (see Fig. 3.22(f)). After the outer layers of the drop have been stripped away, the stripping mechanism continues to shed off droplets from the remaining bulk drop, as is seen in Fig. 3.22(g). The temporal evolution of the stripping breakup is in good agreement with the experimental observations of Dai and Faeth [14] (see Fig. 3.23, where t/t^* is the dimensionless time calculated using Eq. (3.16)). Similar experimental observations have been reported by other authors including Gelfand [28], Josephand et al. [43].

3.2.3.5 Drop Breakup Time

Two dimensionless breakup times are typically measured: the initiation time T_{ini} and the total breakup time T_{tot}. According to Guildenbecher et al. [30] the initiation time is defined as the moment when the intact but deformed drop resembles an oblate spheroid, while the total breakup time is defined as the moment when the disintegrated drop and all its fragments have reached a stable state and no further breakup occurs. Time is nondimensionalized according to Ranger and Nicholls [70]

by

$$T = t\frac{v}{\epsilon^{0.5} d_0}, \tag{3.16}$$

where T is the dimensionless time, t is the dimensional time, ϵ is the drop to ambient density ratio, v is the relative velocity between the drop and the ambient air and d_0 is the diameter of the initially spherical drop. The dimensional and dimensionless initiation and total breakup times are determined for the CFD simulations and are listed in Table 3.7. The numerical results are compared with the experimental observations of Guildenbecher et al. [30], namely, that $T_{ini} \approx 1.5$ and $T_{tot} \approx 5.0$, assuming the dimensionless times are independent of We and Oh when $Oh < 0.1$. It can be seen from Table 3.7 that T_{ini} and T_{tot} are decreasing with increasing Weber numbers. However, given the uncertainty in determining t_{ini} and t_{tot} for the CFD simulations as well as in the experiments, the agreement of the dimensionless initiation times can be considered as good, and the agreement of the total breakup time as reasonable.

Table 3.7
Initiation and total breakup times for the bag, the stamen and the stripping breakup regimes

We	t_{ini} $[ms]$	T_{ini}	t_{tot} $[ms]$	T_{tot}
14.87	1.4	1.46	3.9	4.07
20.09	1.1	1.34	2.6	3.16
81.04	0.5	1.22	1.2	2.92

3.2.3.6 Drop Size Distributions

The drop sizes have been sampled in two stages using a modified version of the post-processing tool of Case et al. [12]. The modifications include a mechanism which allows the tracking of droplets in symmetric simulations.

One limitation of the VOF method is that not every drop can be resolved by the computational mesh, especially the tiny (unresolved) drops produced after breakup. Drop size distributions are determined in two stages because product droplets tend to leave the domain. In the first stage, the sampling time is chosen after the onset of breakup but before any of the resolved drops have left the domain. At this moment, the droplet statistics is performed for all the stable resolved drops, i.e., drops whose (local) Weber numbers are subcritical, and therefore, do not undergo further breakup. These drops are then marked and, if they are still in the domain, are ignored in the second sampling. The second sampling is performed at the end of the computation when all the drops' (local) Weber numbers are subcritical.

More formally, the droplet sampling can be expressed by the two equations

$$m_{tot} = m_{ur}^{(1)} + m_{sr}^{(1)} + m_{t}^{(1)}, \tag{3.17}$$

$$m_{ur}^{(1)} = m_{sr}^{(2)} + m_{t}^{(2)}. \tag{3.18}$$

In these equations, the superscripts (1) and (2) indicate the times of droplet sampling at the first and second stage, respectively. m_{tot} is the total mass of the initially spherical drop, $m_{ur}^{(1)}$ is the mass of the unstable resolved drops in the first stage, $m_{sr}^{(1)}$ is the mass of the stable resolved drops in the first stage and $m_t^{(1)}$ is the mass of the tiny (stable unresolved) drops in the first stage. $m_{sr}^{(2)}$ is the mass of the stable resolved drops in the second stage and $m_t^{(2)}$ is the mass of the tiny drops in the second stage.

The mass of the tiny drops that cannot be resolved by the computational mesh is calculated in each stage. At the first sampling time, the mass of the tiny drops in the first stage, $m_t^{(1)}$, is obtained using Eq. (3.17) by subtracting the mass of the resolved drops including stable and unstable ones in the first stage, $m_{ur}^{(1)} + m_{sr}^{(1)}$, from the total mass of the initially spherical drop, m_{tot}. At the second sampling time, the mass of the tiny drops in the second stage, $m_t^{(2)}$, is obtained using Eq. (3.18) by subtracting the mass of the stable resolved drops in the second stage, $m_{sr}^{(2)}$, from the mass of the unstable resolved drops in the first stage, $m_{ur}^{(1)}$. The number of tiny drops is estimated by the mass of all these tiny drops divided by the mass of a spherical drop of 0.00625 mm radius, i.e., a quarter of the size of the cell in the middle block of the computational mesh, which is assumed to be the mean radius of the tiny drops.

For the bag breakup case, the product drop sizes are first calculated at $t = 2.3$ ms when the breakup of the bag has finished and are then calculated at $t = 3.9$ ms when the breakup of the rim is complete. The final product drop size distribution of the

resolved drops for the bag breakup, shown in Fig. 3.24, is calculated by combining these two drop size data and is in good agreement with the experimental observations of Yao et al. [95]. As is seen in Table 3.8, there are 162 resolved drops, and the mean radius of these drops is 0.0528 mm. There are 66357 unresolved drops of 0.00625 mm radius. These numbers reflect the number of drops in the whole three dimensional domain, not just in the one-quarter domain used for the computations.

Table 3.8
Statistics of drops after breakup

	Resolved drops		Unresolved drops		
We	Number	Mean radius $[mm]$	Number	Mean radius $[mm]$	Percentage of mass
14.87	162	0.0528	66,357	0.00625	12%
20.09	257	0.0385	83,863	0.00625	16%
81.04	654	0.0341	117,295	0.00625	22%

For the stamen breakup case, the product drop sizes are first calculated at $t = 1.6\ ms$ when the breakup of the bag has finished and are then calculated at $t = 2.6\ ms$ when the breakup of the stamen and its surrounding rim is complete. The final product drop size distribution of the resolved drops for the stamen breakup, shown in Fig. 3.25, is calculated by combing these two drop size data. As is seen in Table 3.8, there are 257 resolved drops, and the mean radius of these drops is 0.0385 mm. There are 83863 unresolved drops of 0.00625 mm radius.

For the stripping breakup case, the product drop sizes are first calculated at $t = 0.8\ ms$ when the stripping of ligaments stretched from the rim around the middle bulk drop has finished and are then calculated at $t = 1.2\ ms$ when the stripping of the middle

bulk drop is complete. The final product drop size distribution of the resolved drops for the stripping breakup, shown in Fig. 3.26, is calculated by combining these two drop size data. As is seen in Table 3.8, there are 654 resolved drops, and the mean radius of these drops is 0.0341 mm. There are 117295 unresolved drops of 0.00625 mm radius.

Figures 3.24, 3.25 and 3.26 only show drops that are resolved by the computational mesh. Both lognormal and volume-weighted χ^2 distributions are used to fit the product drop size distribution data. The probability density function (PDF) of the lognormal distribution is

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}, \tag{3.19}$$

where $x = d/d_0$, normalized diameter of the product drop by the diameter of the initially spherical drop, d_0, μ is the mean of $\ln x$, and σ is the standard deviation of $\ln x$. The PDF of the volume-weighted χ^2 distribution is

$$f(r) = \frac{1}{6\bar{r}} \left(\frac{r}{\bar{r}}\right)^3 e^{-\frac{r}{\bar{r}}}, \tag{3.20}$$

where r is the radius of the product drop, $\bar{r}$ is the mean of r. The fits are in good agreement with the only available experimental data from Yao et al. [95] which is limited to the bag breakup regime. By calculating the sum of squared errors (SSE), i.e., the sum of the squares of difference between the observed value and the estimated value from a fit, Table 3.9 shows that the volume-weighted χ^2 fit is a little bit better

than the lognormal fit for the bag breakup, the lognormal fit is much better than the volume-weighted χ^2 fit for the stamen breakup, and both fits perform equally well for the stripping breakup.

Table 3.9
Sum of Squared Error (SSE) of fittings

Weber number	lognormal	vol.-weighted χ^2
14.87	714.98	634.03
20.09	1059.31	1526.87
81.04	3155.58	3146.79

3.2.4 Summary and Conclusions

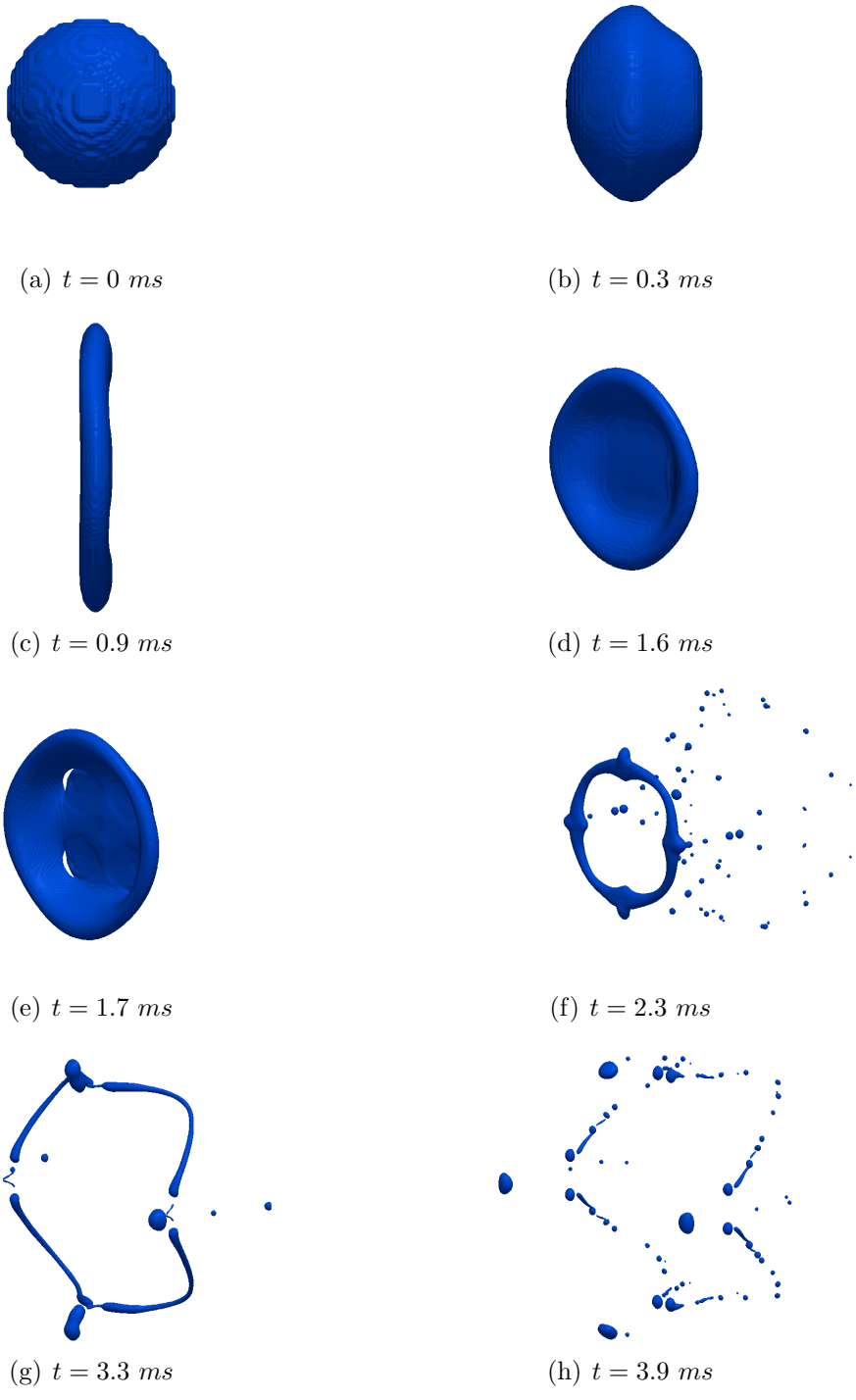
In this section, volume of fluid based multiphase flow simulations have been performed for water drops in air flows of different Weber numbers, corresponding to different breakup regimes. In order to keep the drop within the fixed computational domain, the location of the drop is adjusted using the SEA method developed in this research.

The computational domain was reduced to one-quarter of the full three dimensional domain by assuming symmetry. To justify the symmetry assumption, a fully three dimensional simulation has been carried out for a non-breaking drop for $We = 6.56$. From the fact that the three dimensional velocity and pressure fields are highly symmetric, the symmetry assumption is taken to be justified.

Drop breakup has been studied by means of three dimensional symmetric simulations

for Weber numbers associated with the bag breakup, the stamen breakup, and the stripping breakup regimes. These breakup modes are well captured in the present simulations, and the temporal evolution of the breakup processes as well as the initiation and total breakup times are in good agreement with experimental observations.

For each breakup mode in the simulations, the product drop sizes follow lognormal distribution and volume-weighted χ^2 distribution, and there are much more tiny drops than resolved drops. The statistics of drops for each breakup mode in the simulations indicates that atomizations for larger Weber numbers produce more drops of smaller size. All this is in good agreement with experimental observations.



(a) $t = 0\ ms$ (b) $t = 0.3\ ms$

(c) $t = 0.9\ ms$ (d) $t = 1.6\ ms$

(e) $t = 1.7\ ms$ (f) $t = 2.3\ ms$

(g) $t = 3.3\ ms$ (h) $t = 3.9\ ms$

Figure 3.18: Time sequence of bag breakup for $We = 14.87$.

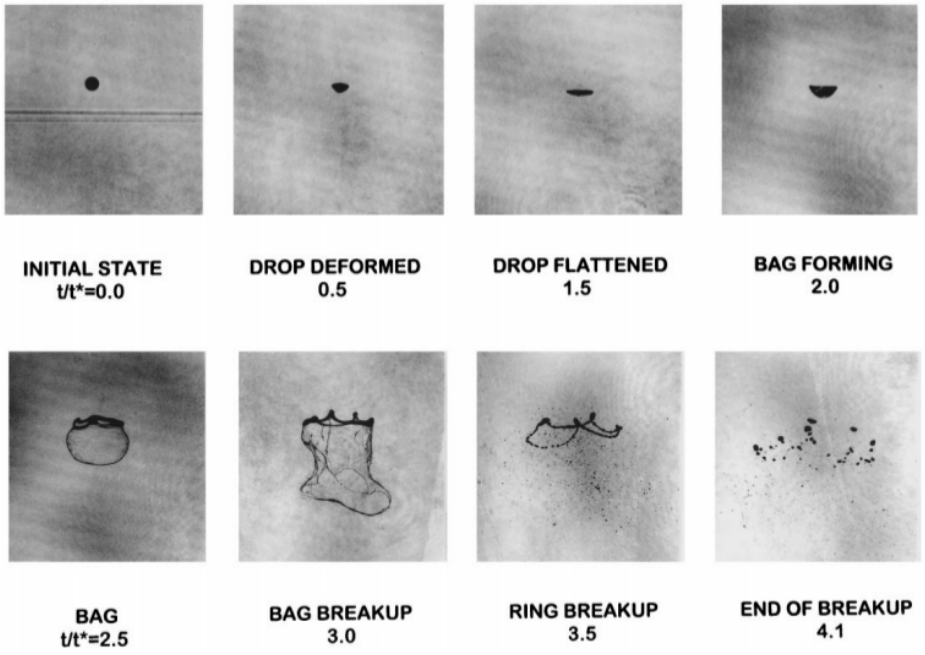
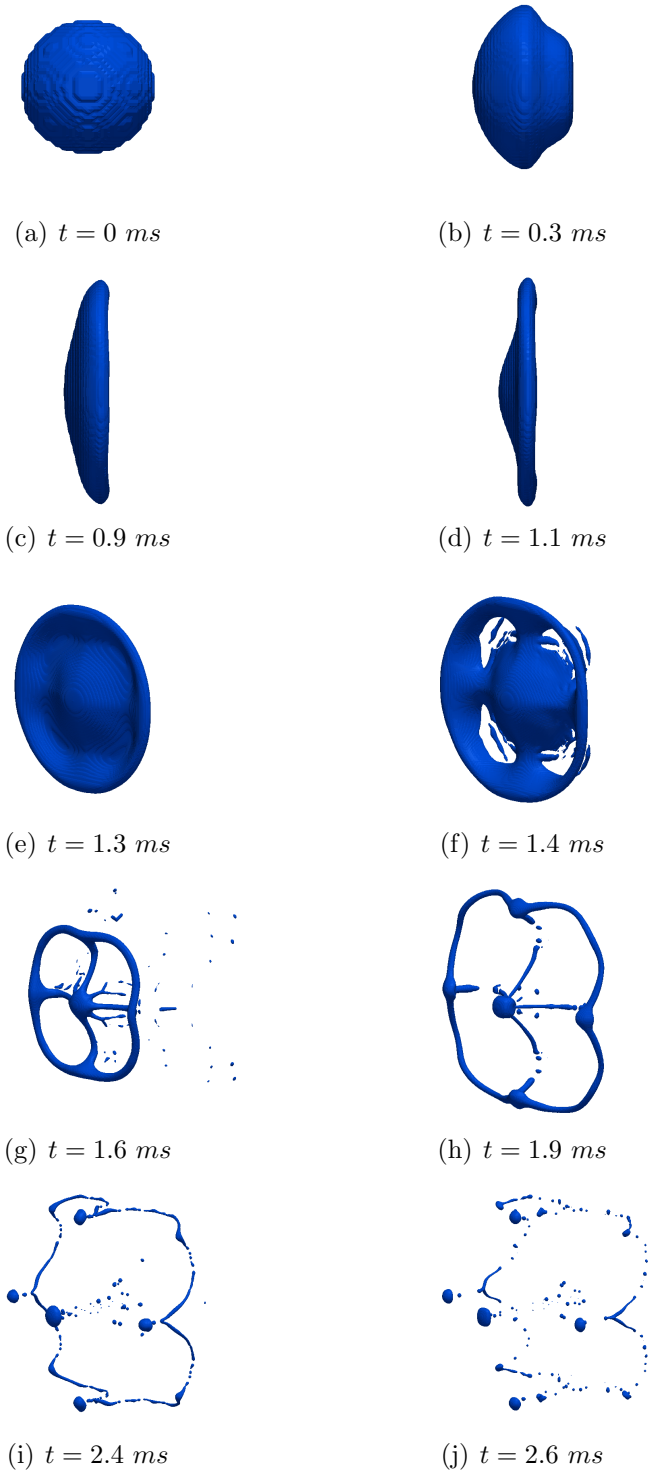


Figure 3.19: Pulse shadowgraphs of secondary breakup in the bag breakup regime (water, $We = 15$, $Oh = 0.0045$). Reprinted from International Journal of Multiphase Flow, 27(2), Dai and Faeth, Temporal properties of secondary drop breakup in the multimode breakup regime, p. 221, Copyright(2001), with permission from Elsevier. See documentation in Appendix B.



(a) $t = 0\ ms$ (b) $t = 0.3\ ms$

(c) $t = 0.9\ ms$ (d) $t = 1.1\ ms$

(e) $t = 1.3\ ms$ (f) $t = 1.4\ ms$

(g) $t = 1.6\ ms$ (h) $t = 1.9\ ms$

(i) $t = 2.4\ ms$ (j) $t = 2.6\ ms$

Figure 3.20: Time sequence of stamen breakup for $We = 20.09$.

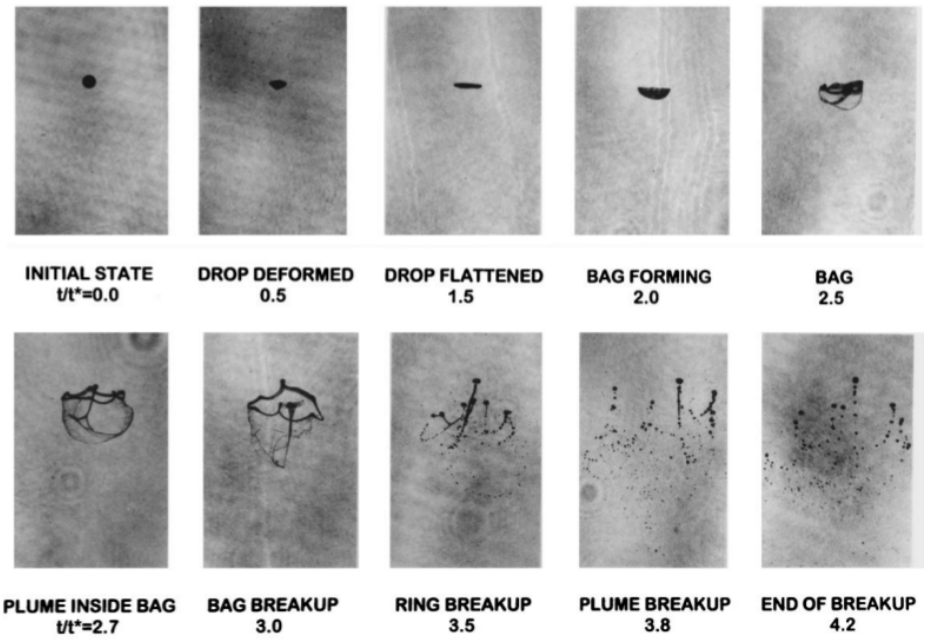
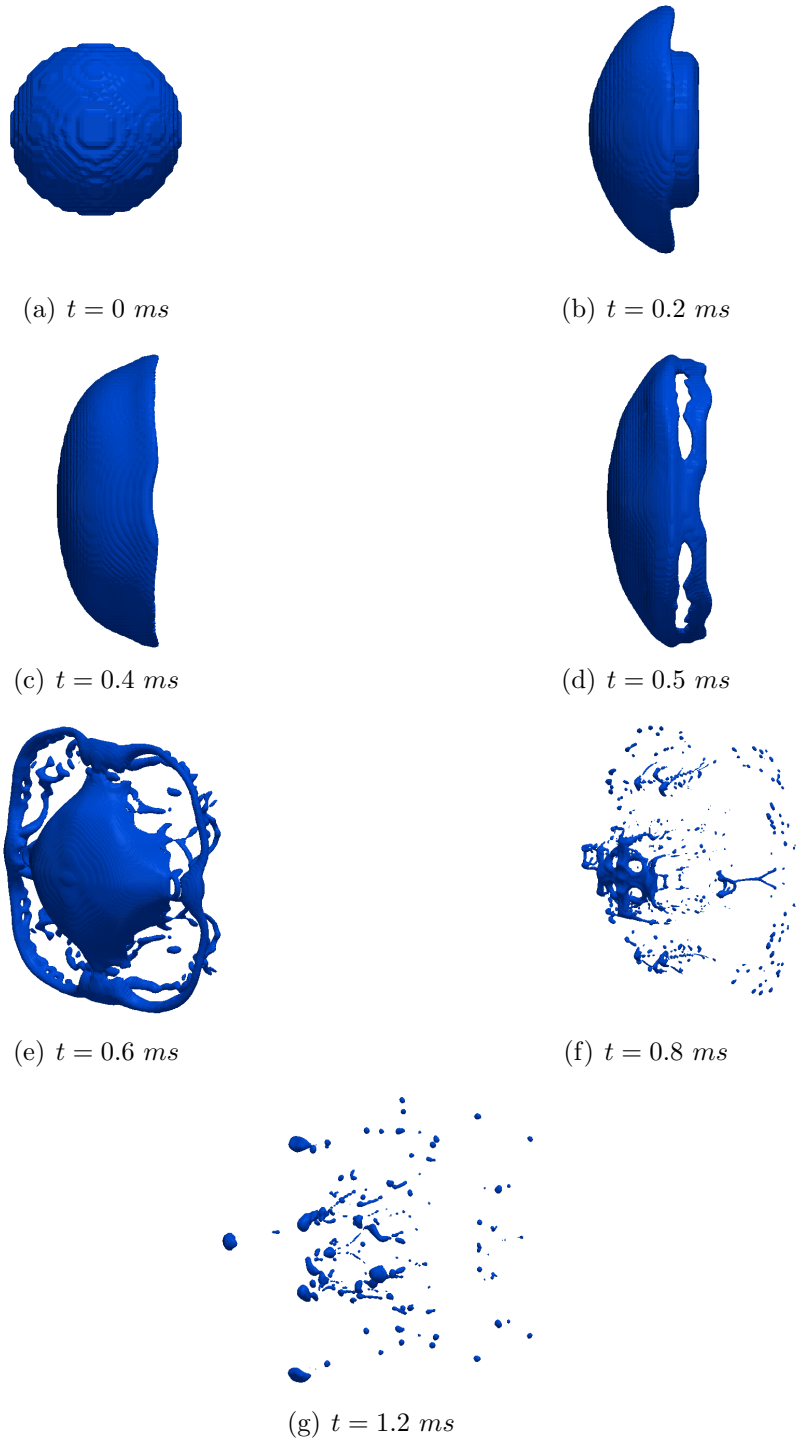


Figure 3.21: Pulse shadowgraphs of secondary breakup in the stamen breakup regime (water, $We = 20$, $Oh = 0.0045$). Reprinted from International Journal of Multiphase Flow, 27(2), Dai and Faeth, Temporal properties of secondary drop breakup in the multimode breakup regime, p. 222, Copyright(2001), with permission from Elsevier. See documentation in Appendix B.



(a) $t = 0\ ms$

(b) $t = 0.2\ ms$

(c) $t = 0.4\ ms$

(d) $t = 0.5\ ms$

(e) $t = 0.6\ ms$

(f) $t = 0.8\ ms$

(g) $t = 1.2\ ms$

Figure 3.22: Time sequence of stripping breakup for $We = 81.04$.

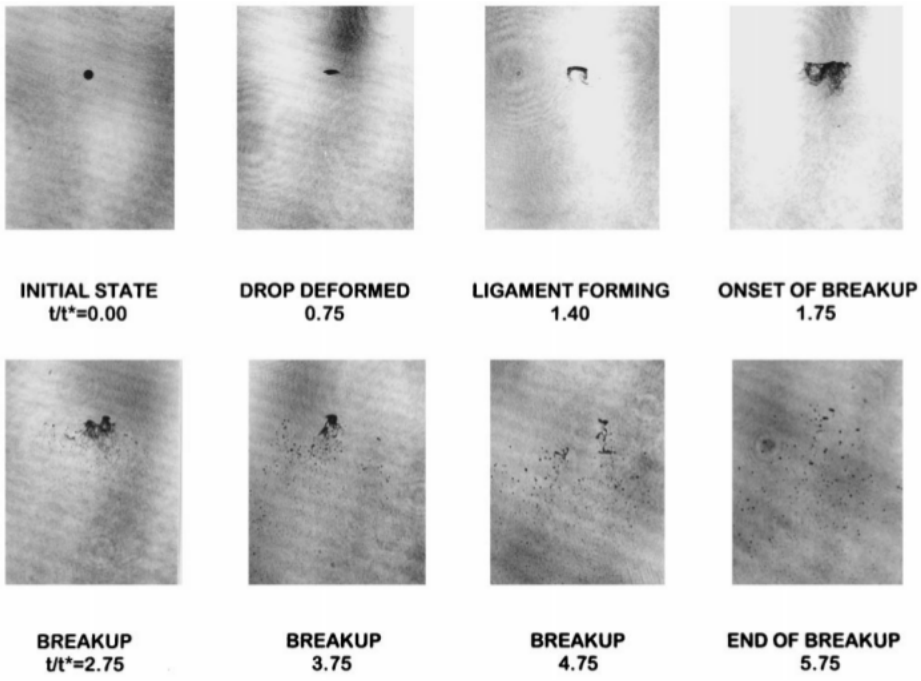


Figure 3.23: Pulse shadowgraphs of secondary breakup in the stripping breakup regime (ethyl alcohol, $We = 81$, $Oh = 0.0126$). Reprinted from International Journal of Multiphase Flow, 27(2), Dai and Faeth, Temporal properties of secondary drop breakup in the multimode breakup regime, p. 227, Copyright(2001), with permission from Elsevier. See documentation in Appendix B.

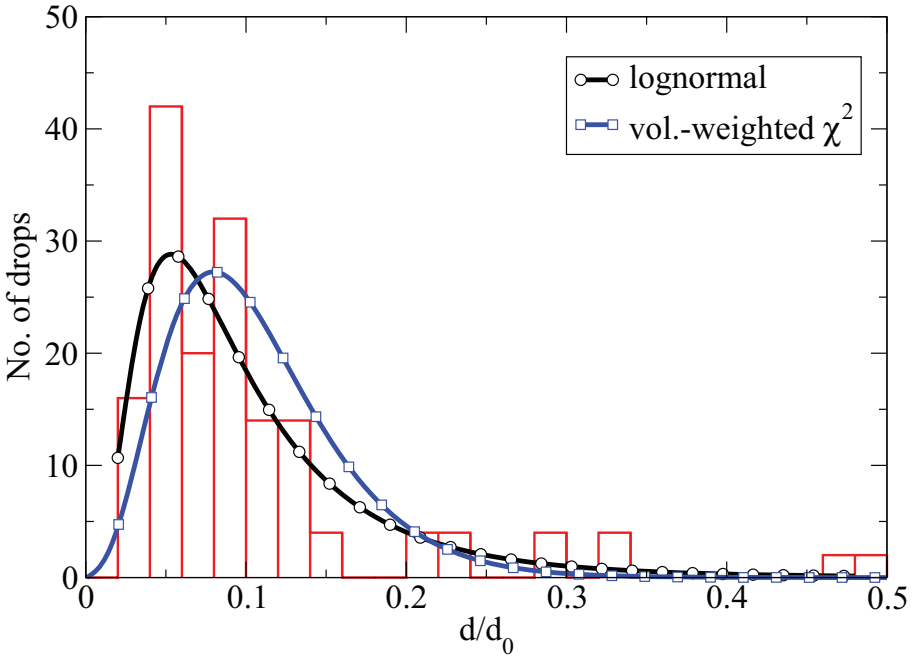


Figure 3.24: Drop size distribution for the bag breakup ($We = 14.87$). Corresponding curves drawn are the lognormal fit $1/(x\sigma\sqrt{2\pi})e^{-(\ln x-\mu)^2/(2\sigma^2)}$, where $x = d/d_0$, normalized diameter by the initial diameter $d_0 = 1\ mm$, with parameters $\mu = -2.49575$, $\sigma = 0.67368$ and the volume-weighted χ^2 fit $1/(6\bar{r})(r/\bar{r})^3 e^{-r/\bar{r}}$ with parameter mean of radius $\bar{r} = 0.0528\ mm$.

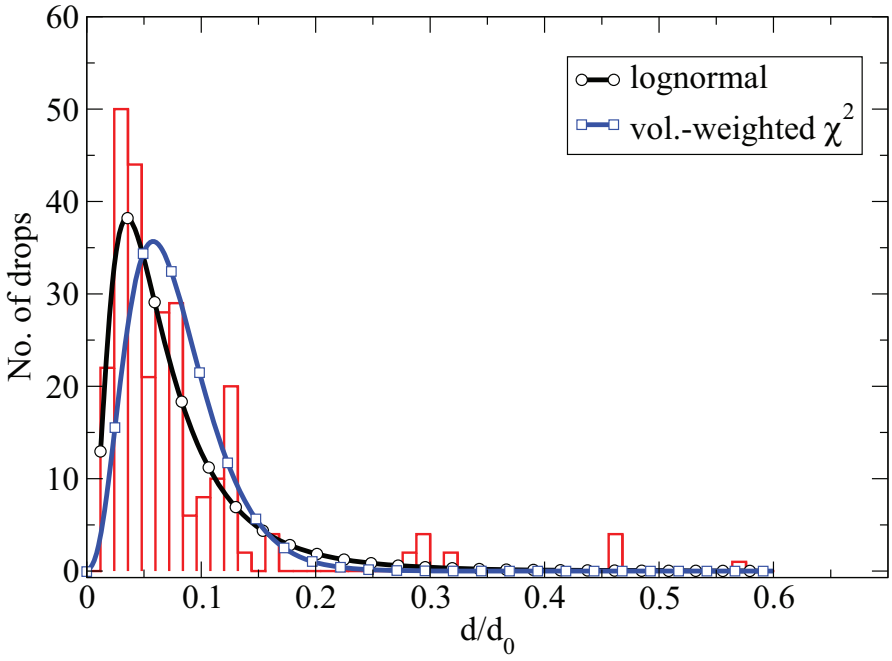


Figure 3.25: Drop size distribution for the stamen breakup ($We = 20.09$). Corresponding curves drawn are the lognormal fit $1/(x\sigma\sqrt{2\pi})e^{-(\ln x-\mu)^2/(2\sigma^2)}$, where $x = d/d_0$, normalized diameter by the initial diameter $d_0 = 1\ mm$, with parameters $\mu = -2.85274$, $\sigma = 0.71275$ and the volume-weighted χ^2 fit $1/(6\bar{r})(r/\bar{r})^3 e^{-r/\bar{r}}$ with parameter mean of radius $\bar{r} = 0.0385\ mm$.

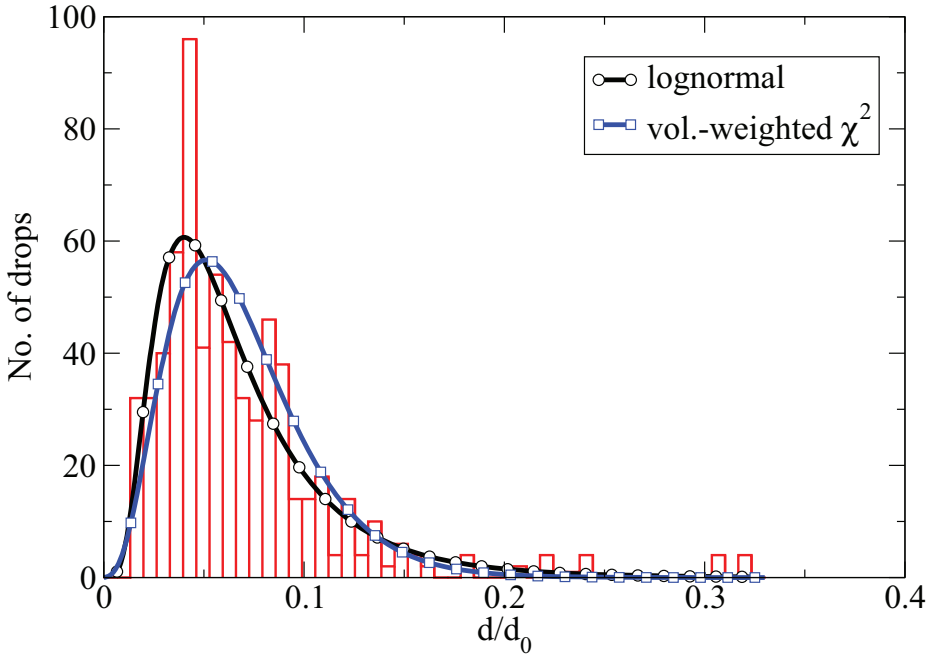


Figure 3.26: Drop size distribution for the stripping breakup ($We = 81.04$). Corresponding curves drawn are the lognormal fit $1/(x\sigma\sqrt{2\pi})e^{-(\ln x-\mu)^2/(2\sigma^2)}$, where $x = d/d_0$, normalized diameter by the initial diameter $d_0 = 1\ mm$, with parameters $\mu = -2.86721$, $\sigma = 0.59358$ and the volume-weighted χ^2 fit $1/(6\bar{r})(r/\bar{r})^3 e^{-r/\bar{r}}$ with parameter mean of radius $\bar{r} = 0.0341\ mm$.

Chapter 4

Convective Heat Transfer Coefficient Calculation

Convective heat transfer between two substances can be described by Newton's law of cooling

$$q_h = h_c(T_s - T_f), \tag{4.1}$$

where q_h is the heat flux, h_c is the convective heat transfer coefficient, T_s is the surface temperature, and T_f is the fluid reference temperature. If h_c is known, then Eq. (4.1) provides a computationally inexpensive way of determining heat transfer in complex multiphase flows such as sprays. In sprays, the heat transfer between the liquid and gas phase, including phase change such as solidification or evaporation, has to be

computed for millions of droplets. The only way such tasks can be handled with present-day computing technology is by relying on relatively simple models which utilize Eq. (4.1), and, therefore, use the correct values of h_c.

The heat transfer coefficient depends on the physical properties of the fluid, the type of flow, and the geometric configuration in which convection occurs. Therefore, in order to use Eq. (4.1), h_c has to be known for the specific problem under consideration. Heat transfer coefficients can be determined from theoretical considerations or by means of experiments. Another method is the use of CFD, as is employed in this thesis. In this approach, the heat transfer of the specific flow problem is simulated by resolving all the necessary time and length scales. Under these conditions, the convective heat transfer is in fact a macroscopic manifestation of the local heat conduction between the two media. In other words, when all the necessary scales are resolved, one can use Fourier's law of heat conduction to compute the heat transfer. Once the transferred heat, together with the temperatures T_s and T_f, is known, then h_c can be determined from Eq. (4.1).

4.1 Mathematical Model and Numerical Methods

For describing heat transfer in an incompressible Newtonian fluid in the laminar flow regime, the following governing equations are solved:

Conservation of mass

$$\nabla \cdot \mathbf{v} = 0, \tag{4.2}$$

Conservation of momentum

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = \nabla \cdot (\nu \nabla \mathbf{v}) - \frac{1}{\rho_0} \nabla p, \tag{4.3}$$

Conservation of energy

$$\frac{\partial T}{\partial t} + \nabla \cdot (\mathbf{v}\, T) - \nabla \cdot (\frac{k}{\rho_0 c_p} \nabla T) = 0, \tag{4.4}$$

where T is the temperature, ρ_0 is the density, $\mathbf{v}$ is the velocity, ν is the kinematic viscosity, p is the pressure, k is the thermal conductivity and c_p is the specific heat capacity.

Continuum (CFD) methods are used to compute the heat transfer between two entities by resolving all the necessary length and time scales. This requires the energy conservation equation, Eq. (4.4), which utilizes Fourier's law of heat conduction,

$$q_h = -k \nabla T. \tag{4.5}$$

From post-processing of the CFD solution, the surface temperature T_s and the reference temperature T_f are computed. The determination of T_f depends on the geometry configuration. There are different choices of the reference temperature used in different geometries. From Fourier's law of heat conduction, Eq. (4.5), the heat flux q_h is computed from post-processing of CFD data. The convective heat transfer coefficient, h_c, is then calculated using Eq. (4.1).

The convective heat transfer coefficient can also be expressed in terms of the Nusselt number,

$$Nu = \frac{h_c D_h}{k}, \tag{4.6}$$

where D_h is the hydraulic diameter,

$$D_h = \frac{4A}{P}, \tag{4.7}$$

where A is the cross-sectional area, and P is the wetted perimeter of the cross-section.

The feasibility and accuracy of calculating convective heat transfer coefficients using CFD has been investigated using OpenFOAM® for heat transfer in the laminar flow regime in the following two dimensional cases: (1) flow between parallel flat plates with constant plate wall temperature and with constant heat flux, respectively; (2) flow past a cylinder with constant cylinder wall temperature and with constant heat flux, respectively. The numerical results are then compared with experimental results.

4.2 Flow Between Parallel Flat Plates

4.2.1 Problem Description

The material properties used in the simulations are shown in Table 4.1.

Table 4.1
Material properties for air

Density	ρ	$1.225\ kg/m^3$
Dynamic viscosity	μ	$1.7894 \times 10^{-5}\ kg/m \cdot s$
Thermal conductivity	k	$0.0242 W/mK$
Specific heat capacity	c_p	$1006.43 J/kgK$

Two cases will be simulated using OpenFOAM®: (1) parallel flat plates with constant heat flux; (2) parallel flat plates with constant wall temperature. Both cases are illustrated below in Fig. 4.1.

Different reference temperatures are used to show their effects on the calculation of h_c: the constant reference temperature, T_{ref}, the centerline temperature, T_c, taken at the middle horizontal line in Figure 4.1, and the bulk temperature, T_b, which is defined as [51]

$$T_b = \frac{\int_y \rho c_p u T \, dy}{\dot{m} c_p}, \tag{4.8}$$

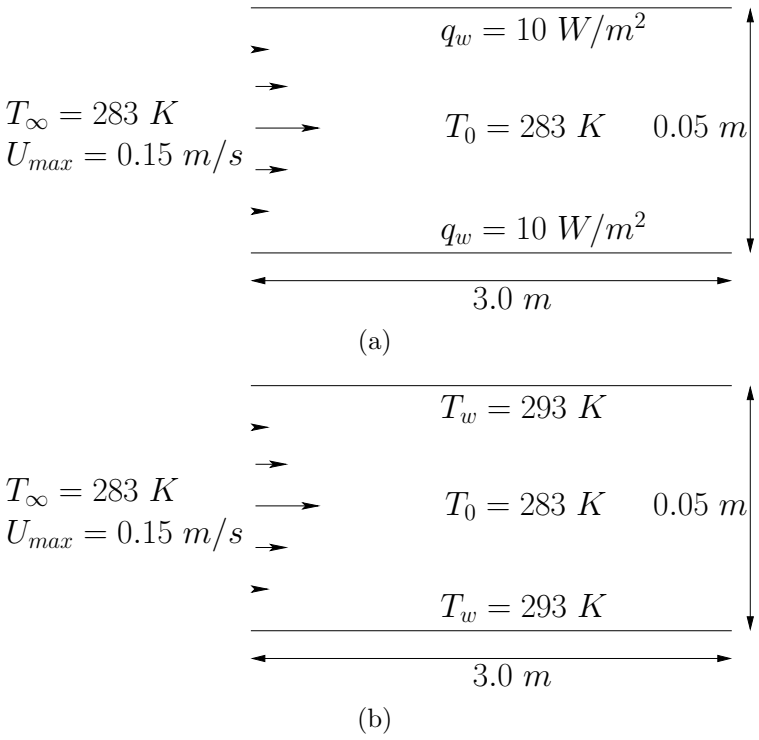


Figure 4.1: Schematic representation of the two case studies with (a) constant heat flux; (b) constant wall temperature.

where ρ is the fluid density, c_p is the specific heat capacity, u is the horizontal component of velocity, T is the temperature, and $\dot{m}$ is the mass flow rate. In this case, the material properties are considered constant, and Eq.(4.8) can be simplified to the following form:

$$T_b = \frac{\sum_{i=1}^{n}(u_i b_i T_i)}{U_{av} b}, \tag{4.9}$$

where u_i is the horizontal component of velocity in a control volume (CV), b_i is the height of the CV, T_i is the temperature in the CV, U_{av} is the horizontal component of velocity averaged over the distance between plates (the height of the domain), and b is the height of the domain.

According to Lienhard [51],

$$Nu = \begin{cases} 7.541 & \text{for fixed plate wall temperature} \\ \\ 8.235 & \text{for fixed plate wall heat flux} \end{cases} \tag{4.10}$$

for this geometry, D_h is twice the distance between parallel plates,

$$h_c = \frac{Nuk}{D_h} = \begin{cases} 1.825\ W/m^2K & \text{for fixed plate wall temperature} \\ \\ 1.993\ W/m^2K & \text{for fixed plate wall heat flux} \end{cases} . \tag{4.11}$$

A solver for transient, incompressible, Newtonian fluids with temperature transport using an adaptive time step, `icoTempFoamVarDt`, has been developed and used to perform the CFD simulations. From numerical experiments, it was determined that the simulation time of 70 s is sufficient for the thermal flow to reach steady state.

4.2.2 Mesh Independence Study

The geometry shown in Figure 4.1 is equipped with a computational mesh with uniform cell size along the flow direction and non-uniform cell size towards the wall surfaces. The initial mesh used for the constant heat flux (CHF) and constant wall temperature (CWT) cases has a total of 4800 cells (16 in the vertical direction and

300 in the horizontal direction). The height of the smallest cell is $4.3745 \times 10^{-4}\ m$ and it occurs at the wall boundary. A portion of the mesh is shown in Figure 4.2.

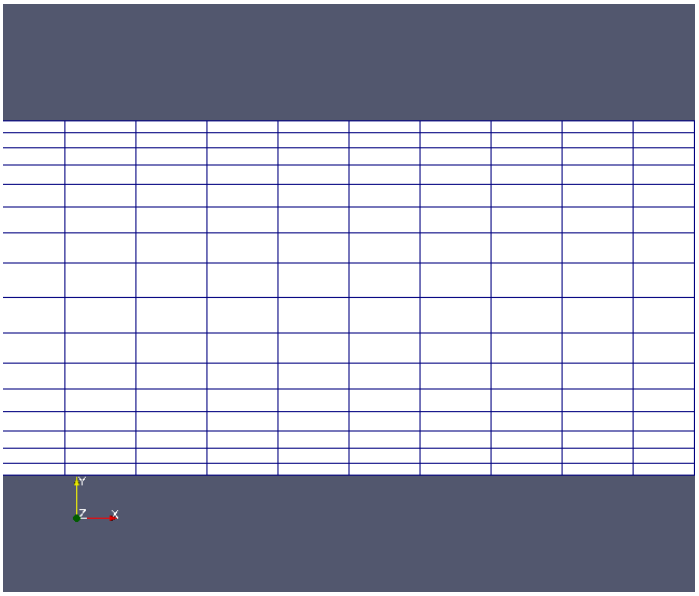


Figure 4.2: Initial mesh used for the CFD simulations.

The initial conditions for the velocity, U, the pressure, p, and the temperature, T, are given in Table 4.2.

Table 4.2
Initial conditions

U	$(0, 0, 0)\ m/s$
p	$0\ m^2/s^2$
T	$283\ K$

The boundary conditions for the CHF case are shown in Table 4.3.

Table 4.3
Boundary conditions for the CHF case

Inlet	U	parabolicVelocity
Inlet	p	zeroGradient
Inlet	T	fixedValue, 283 K
Outlet	U	zeroGradient
Outlet	p	fixedValue, 0 m^2/s^2
Outlet	T	zeroGradient
Wall	U	fixedValue, $(0, 0, 0)$ m/s
Wall	p	zeroGradient
Wall	T	fixedGradient, 413.2231404958678 K/m

The boundary conditions for the CWT case are shown in Table 4.4.

Table 4.4
Boundary conditions for the CWT case

Inlet	U	parabolicVelocity
Inlet	p	zeroGradient
Inlet	T	fixedValue, 283 K
Outlet	U	zeroGradient
Outlet	p	fixedValue, 0 m^2/s^2
Outlet	T	zeroGradient
Wall	U	fixedValue, $(0, 0, 0)$ m/s
Wall	p	zeroGradient
Wall	T	fixedValue, 293 K

The inlet velocity profile for both cases is a parabolic profile, which is used to speed up the computations in order to reach steady state more quickly, as is seen in Fig. 4.3 and 4.4.

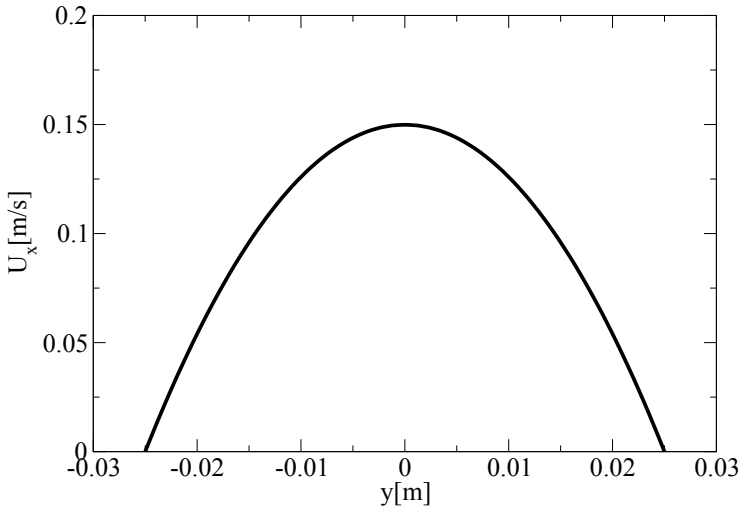


Figure 4.3: Parabolic velocity profile at the inlet.

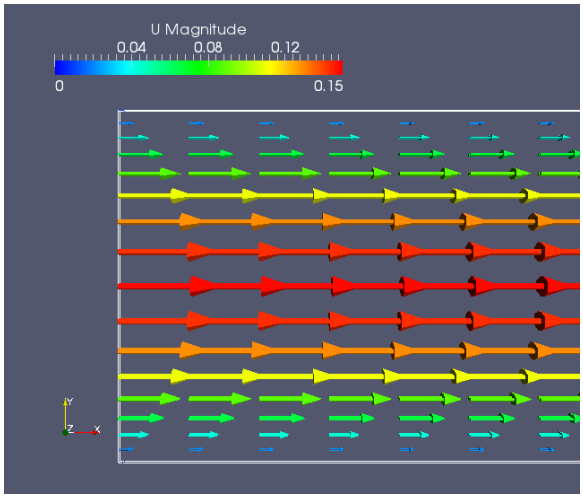


Figure 4.4: Parabolic velocity vector field at the inlet.

The bulk temperature is calculated through Eq.(4.9) using the temperature and velocity data in each cell. The convective heat transfer coefficient, h_{cx}, is calculated

through Eq.(4.1) using the three different fluid reference temperatures. The parameters used to solve Eq.(4.1) are outlined in Table 4.5, where T_{ref} is a constant value that needs to be specified, $T_c(x)$ is the horizontal temperature profile along the center line of the flow, $T_b(x)$ is the bulk temperature at different x, and $T_b(x)$ is calculated through Eq.(4.9) by post-processing of CFD data.

Table 4.5
Parameters used to calculate the convective heat transfer coefficient

	CHF	CWT
$q_w(x)$	$q_w = 10\ W/m^2$	To be calculated
$T_w(x)$	To be calculated	$T_w = 293\ K$
$T_f(x)$	$h_{cx} = \frac{q_w(x)}{T_w(x)-T_f(x)}$	$h_{cx} = \frac{q_w(x)}{T_w(x)-T_f(x)}$
$T_f(x) = T_{ref} = 283\ K$	$h_{cRef}(x) = \frac{10}{T_w(x)-283}$	$h_{cRef}(x) = \frac{q_w(x)}{293-283}$
$T_f(x) = T_c(x)$	$h_{cc}(x) = \frac{10}{T_w(x)-T_c(x)}$	$h_{cc}(x) = \frac{q_w(x)}{293-T_c(x)}$
$T_f(x) = T_b(x)$	$h_{cb}(x) = \frac{10}{T_w(x)-T_b(x)}$	$h_{cb}(x) = \frac{q_w(x)}{293-T_b(x)}$

The results for the convective heat transfer coefficient indicate that the reference temperature, T_f in Eq. (4.1), has a significant effect on the value of the convective heat transfer coefficient. According to Neale et al. [59], the bulk temperature $T_b(x)$ yields the best solution for the convective heat transfer coefficient calculations. The mesh independence study is presented in the following.

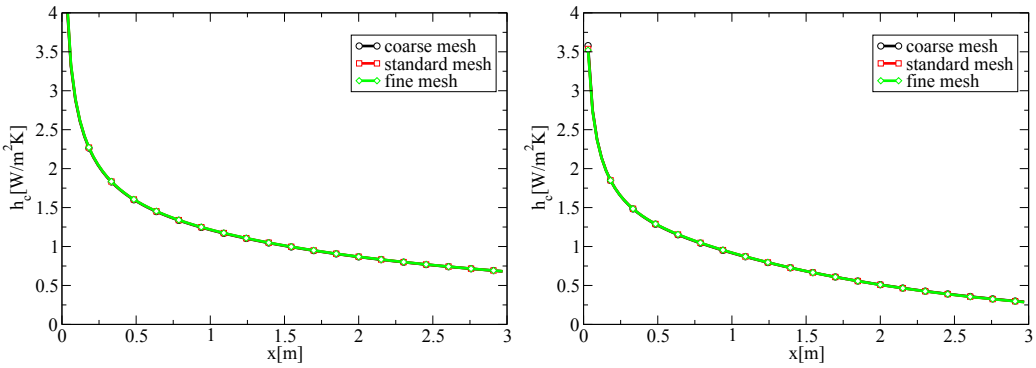
The convective heat transfer coefficients calculated are compared along the x direction for different meshes for both the CHF case and the CWT case. The coefficients calculated using the constant reference temperature, T_{ref}, the centerline temperature, $T_c(x)$, and the bulk temperature, $T_b(x)$, are part of this comparison.

The standard mesh has a total of 20400 cells. The coarse and the fine mesh are obtained from the standard mesh by reducing and increasing the number of cells along each direction by a factor of 2, respectively. The dimensions of the different meshes are listed in Table 4.6.

Table 4.6
Mesh dimensions

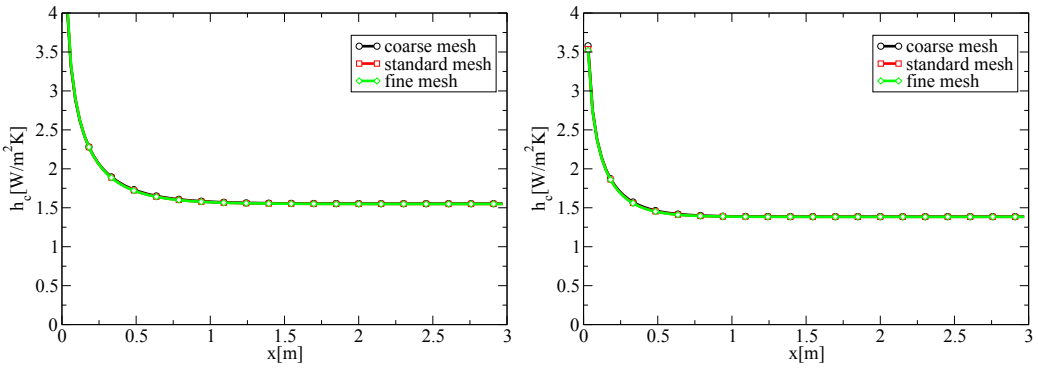
	Fine mesh	Standard mesh	Coarse mesh
Number of cells in y direction	64	32	16
Number of cells in x direction	1200	600	300
Total number of cells	76800	19200	4800

From Figs. 4.5 - 4.7, we can conclude that sufficient mesh independence has been achieved with the standard mesh.



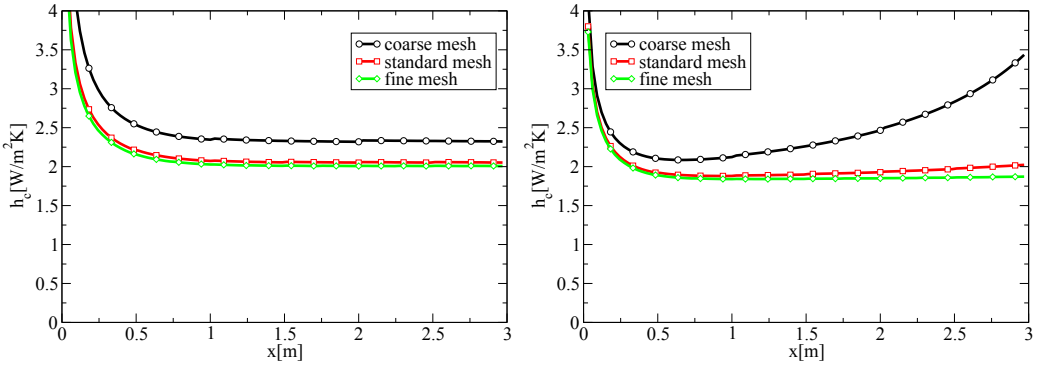
(a) (b)

Figure 4.5: h_c using the constant reference temperature (a) for the CHF case, (b) for the CWT case.



(a) (b)

Figure 4.6: h_c using the centerline reference temperature (a) for the CHF case, (b) for the CWT case.



(a) (b)

Figure 4.7: h_c using the bulk reference temperature (a) for the CHF case, (b) for the CWT case.

4.2.3 Validation of Numerical Results

4.2.3.0.1 Local Convective Heat Transfer Coefficient To validate the simulation results, the convective heat transfer coefficients calculated using the constant temperature, the centerline temperature and the bulk temperature as the reference temperature, are compared with those calculated by Neale et al. [59] using FLUENT® for both the CHF case (see Fig. 4.8) and the CWT case (see Fig. 4.9), respectively.

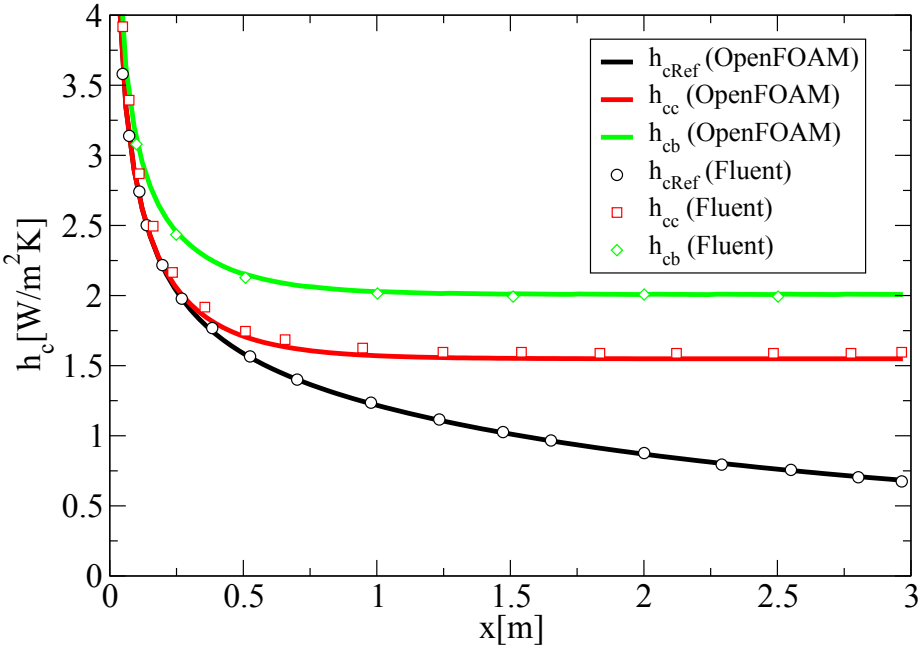


Figure 4.8: h_c for the CHF case: h_{cRef} calculated using the constant reference temperature; h_{cc} calculated using the centerline reference temperature; h_{cb} calculated using the bulk reference temperature.

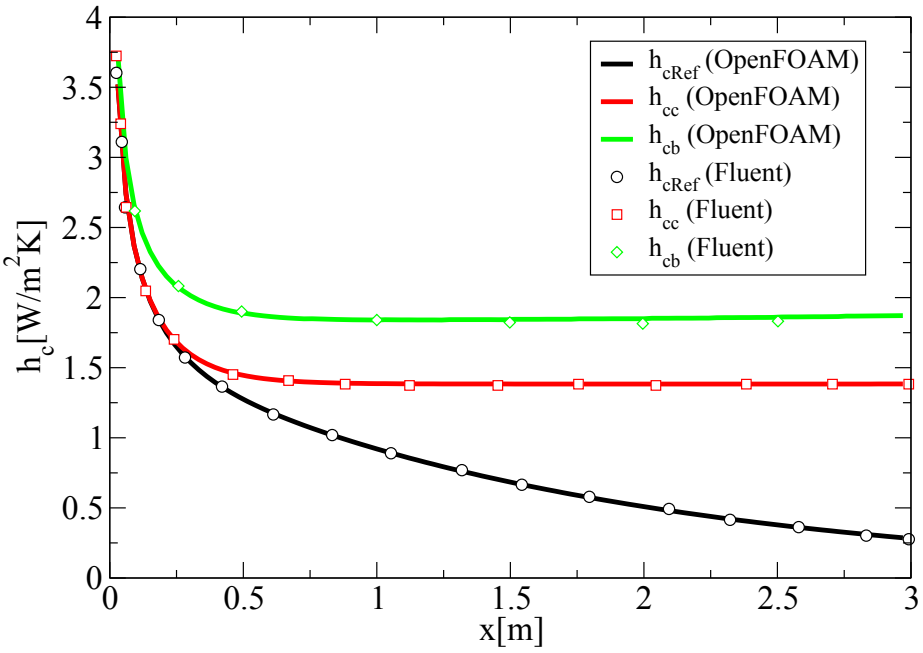


Figure 4.9: h_c for the CWT case: h_{cRef} calculated using the constant reference temperature; h_{cc} calculated using the centerline reference temperature; h_{cb} calculated using the bulk reference temperature

4.2.3.0.2 Average Convective Heat Transfer Coefficient The average convective heat transfer coefficient is obtained by averaging the local convective heat transfer coefficient over the plate surface from $x = 0.5\ m$ to $x = 2.5\ m$ to avoid inlet and outlet effects, i.e.,

$$\bar{h}_c = \frac{1}{2.5 - 0.5} \int_{0.5}^{2.5} h_c(x)\, dx. \tag{4.12}$$

The average convective heat transfer coefficient calculated in this study has good agreement with the analytical value in Eq. (4.11), as is seen in Table 4.7.

Table 4.7
Comparison of the average convective heat transfer coefficients with the analytical values

	CWT	CHF
Analytical value (W/m^2K)	1.825	1.849
Calculated value (W/m^2K)	1.935	2.026

4.3 Flow Past a Cylinder

4.3.1 Problem Description

A two dimensional airflow of uniform velocity of $U_\infty = 0.137402\ m/s$ ($Re = 10$), $U_\infty = 0.274804\ m/s$ ($Re = 20$), $U_\infty = 0.549608\ m/s$ ($Re = 40$), $U_\infty = 0.618309\ m/s$ ($Re = 45$), respectively, and temperature $T_\infty = 283\ K$ pasts over a circular cylinder of 1 mm diameter. The problem is simulated by considering the flow in a channel with a cylinder placed symmetrically between two parallel walls of distance $2L_y = 61\ mm$ with slip boundary conditions on the walls, as is seen in Fig. 4.10. The center of the cylinder is placed at a distance of $L_u = 30.5\ mm$ from the inlet and at a distance of $L_d = 30.5\ mm$ from the outlet, as is seen in Fig. 4.10. The surface of the cylinder is taken to be either at a constant wall temperature $T_w = 293\ K$, or at a uniform heat flux $q_w = 10\ W/m^2$. The physical properties of air are assumed to be temperature independent, as is seen in Table 4.8.

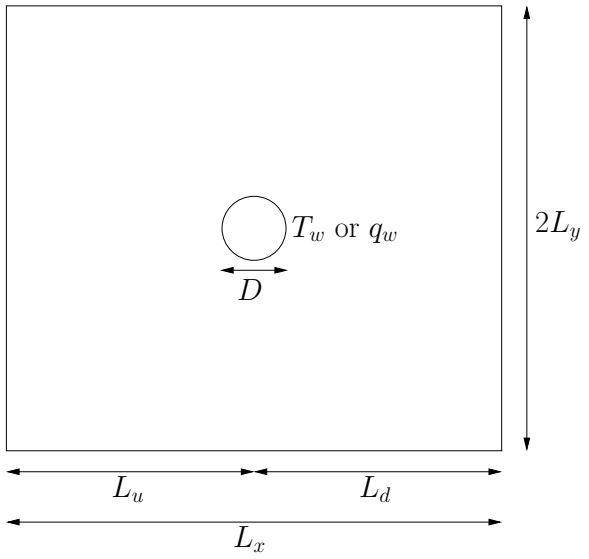


Figure 4.10: Schematic representation of a flow past a circular cylinder.

Table 4.8
Physical properties of air

Density	ρ	$1.225\ kg/m^3$
Kinematic viscosity	ν	$1.37402 \times 10^{-5}\ m^2/s$
Thermal conductivity	k	$0.0242 W/mK$
Specific heat capacity	c_p	$1006.43 J/kgK$

The local Nusselt number is averaged over the surface of the cylinder to obtain the average Nusselt number:

$$Nu = \frac{1}{\pi} \int_0^{\pi} Nu_{Local}\, d\theta, \tag{4.13}$$

where θ is the angular displacement from the front stagnation point. The average Nusselt number can be used in process engineering design calculations, e.g., to estimate the rate of heat transfer from the cylinder in the CWT case, or to estimate the average surface temperature of the cylinder for the UHF case.

4.3.2 Mesh Independence Study

As discussed in Section 4.2.1, `icoTempFoamVarDt` was developed to perform the CFD simulations. From numerical experiments, it was determined that the simulation time of 200 s is sufficient for the thermal flow to reach steady state.

The geometry in Fig. 4.10 is equipped with a non-uniform mesh with a finer grid near the cylinder wall, as is seen in Fig. 4.11 and 4.12.

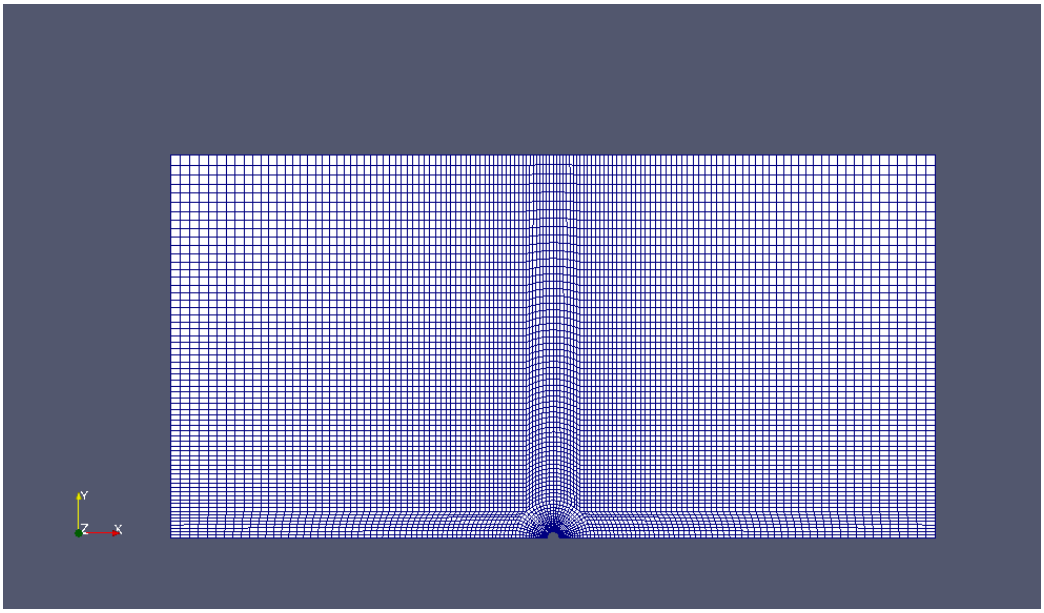


Figure 4.11: Computational Mesh.

The boundary conditions for the CWT case are shown in Table 4.9, and the boundary conditions for the UHF case are shown in Table 4.10.

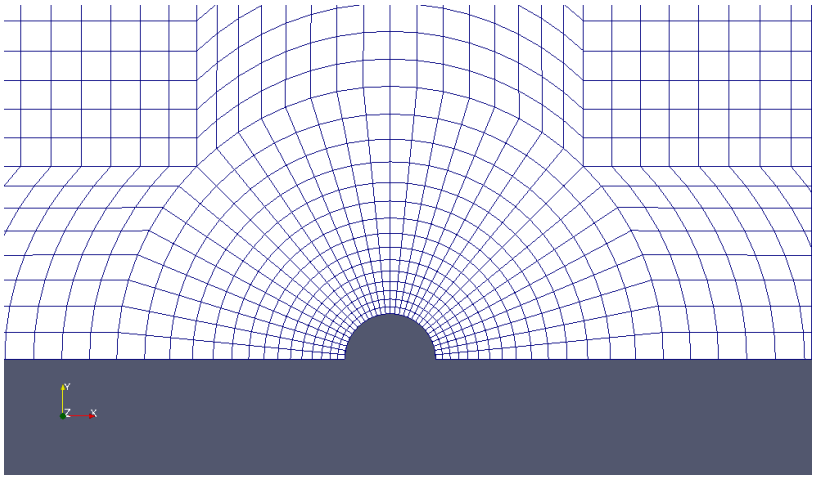


Figure 4.12: Enlargement of the computational mesh near the cylinder wall.

Table 4.9
Boundary conditions for the CWT case, $Re = 45$

Inlet	U	fixedValue, 0.618309 m/s
Inlet	p	zeroGradient
Inlet	T	fixedValue, 283 K
Outlet	U	zeroGradient
Outlet	p	fixedValue, 0 m^2/s^2
Outlet	T	zeroGradient
Cylinder wall surface	U	fixedValue, 0 m/s
Cylinder wall surface	p	zeroGradient
Cylinder wall surface	T	fixedValue, 293 K
Top and bottom boundary	U	slip
Top and bottom boundary	p	slip
Top and bottom boundary	T	slip

The local Nusselt numbers over the circular cylinder wall are compared for different meshes for both the CWT and the UHF cases, respectively. The standard mesh has a total of 19248 cells. The smallest cell is of size 8.562×10^{-14} m^3 and it occurs at the cylinder wall. The coarse and the fine mesh are obtained from the standard mesh by reducing and increasing the number of cells in each direction by a factor of 1.5,

Table 4.10
Boundary conditions for the UHF case, $Re = 45$

Inlet	U	fixedValue, 0.618309 m/s
Inlet	p	zeroGradient
Inlet	T	fixedValue, 283 K
Outlet	U	zeroGradient
Outlet	p	fixedValue, 0 m^2/s^2
Outlet	T	zeroGradient
Cylinder wall surface	U	fixedValue, 0 m/s
Cylinder wall surface	p	zeroGradient
Cylinder wall surface	T	fixedGradient, 413.223 K/m
Top and bottom boundary	U	slip
Top and bottom boundary	p	slip
Top and bottom boundary	T	slip

respectively. The total number of cells of the different meshes are listed in Table 4.11.

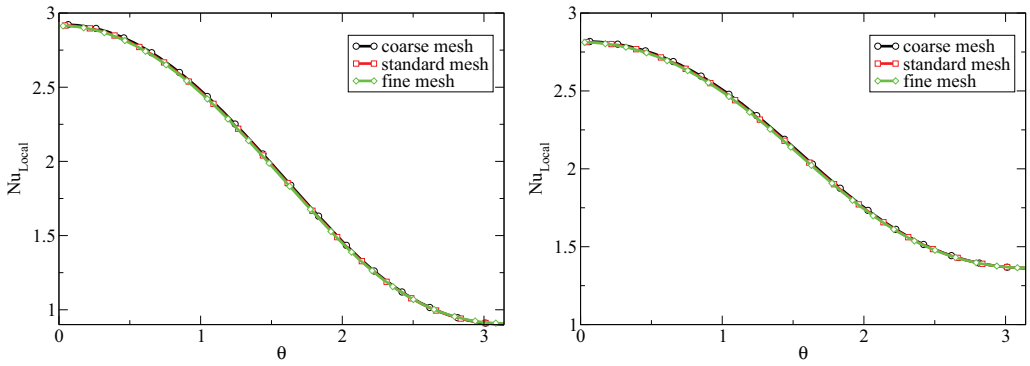
Table 4.11
Mesh dimensions

	Fine mesh	Standard mesh	Coarse mesh
Total number of cells	43344	19248	8544

From Figs. 4.13 - 4.15, we can conclude that sufficient mesh independence has been achieved with the standard mesh.

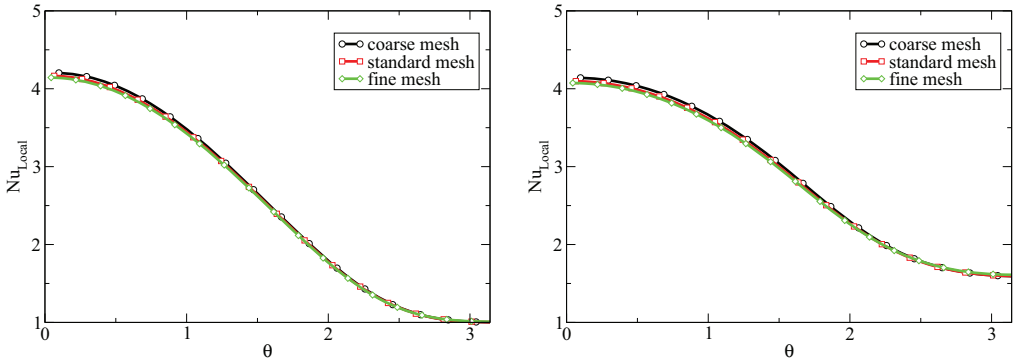
4.3.3 Validation of Numerical Results

To validate the numerical results, the local Nusselt numbers and the average Nusselt number are compared with those calculated by Bharti et al. [7] for the CWT case



(a) (b)

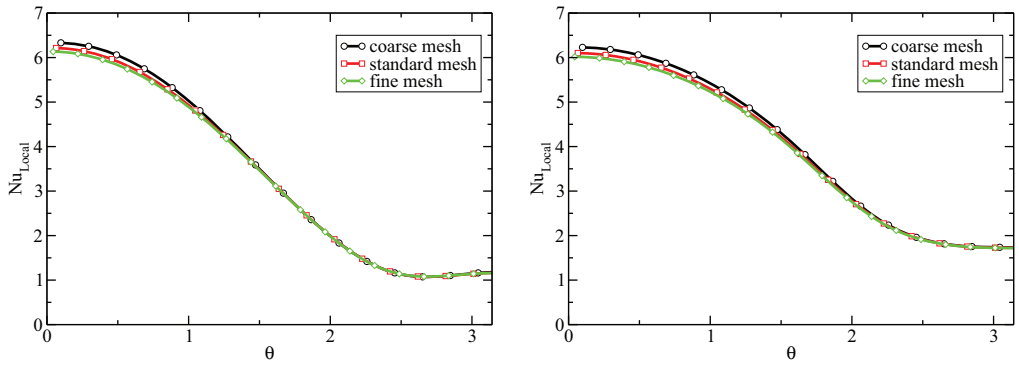
Figure 4.13: Local Nusselt number on the cylinder wall at $Re = 10$, $Pr = 0.7$ (a) for the CWT case, (b) for the UHF case.



(a) (b)

Figure 4.14: Local Nusselt number on the cylinder wall at $Re = 20$, $Pr = 0.7$ (a) for the CWT case, (b) for the UHF case.

and the UHF case, respectively.



(a) (b)

Figure 4.15: Local Nusselt number on the cylinder wall at $Re = 45$, $Pr = 0.7$ (a) for the CWT case, (b) for the UHF case.

4.3.3.0.3 Local Nusselt Number The local Nusselt numbers calculated in this study have good agreement with those calculated by Bharti et al. [7] at different Reynolds numbers for the CWT case (see Figs. 4.16 - 4.18) and for the UHF case (see Figs. 4.19 - 4.21). In these figures the angle, θ, $0 \leq \theta \leq \pi$, is the azimuthal angle of the disk starting at the stagnation point. The reference values denoted by circles are from Bharti et al. [7].

For both cases, the Nusselt number increases with an increase in the Reynolds number. These figures show that relatively large values of the Nusselt number are near the front stagnation point ($\theta = 0$), and the Nusselt number decreases along the cylinder wall to the minimum value near the point of separation due to the thickening of the thermal boundary layer. A gradual increase in values of the Nusselt number can be seen

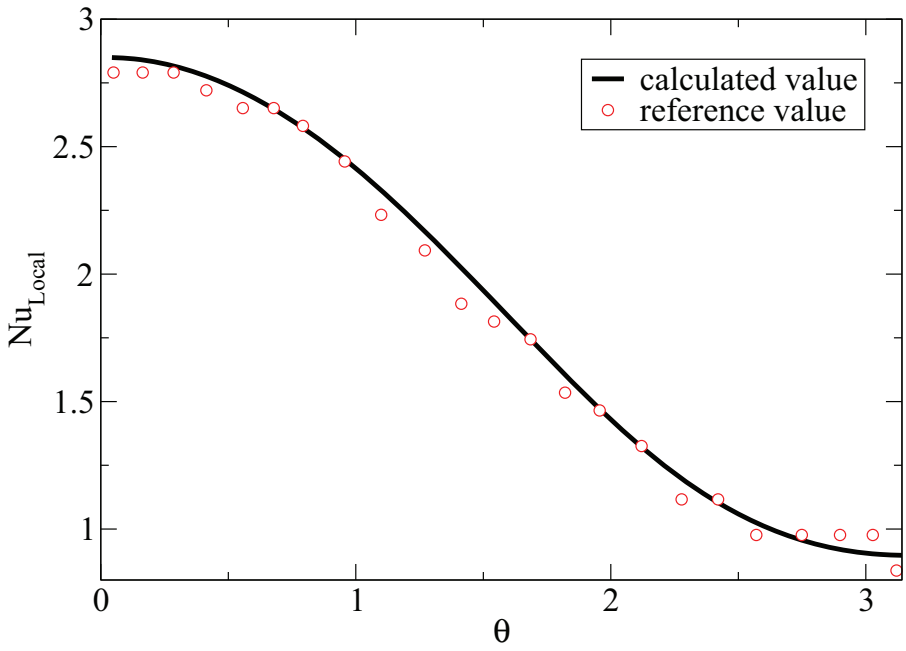


Figure 4.16: Local Nusselt numbers on the cylinder wall at $Re = 10$, $Pr = 0.7$ for the CWT case

with an increase in the Reynolds number from the point of separation to the rear stagnation point ($\theta = \pi$).

4.3.3.0.4 Average Nusselt Number The average Nusselt number is obtained by averaging the local Nusselt numbers over the cylinder wall through Eq. (4.13). The calculated average Nusselt number is in good agreement with the literature, as is seen in Table 4.12 for the CWT case, and Table 4.13 for the UHF case.

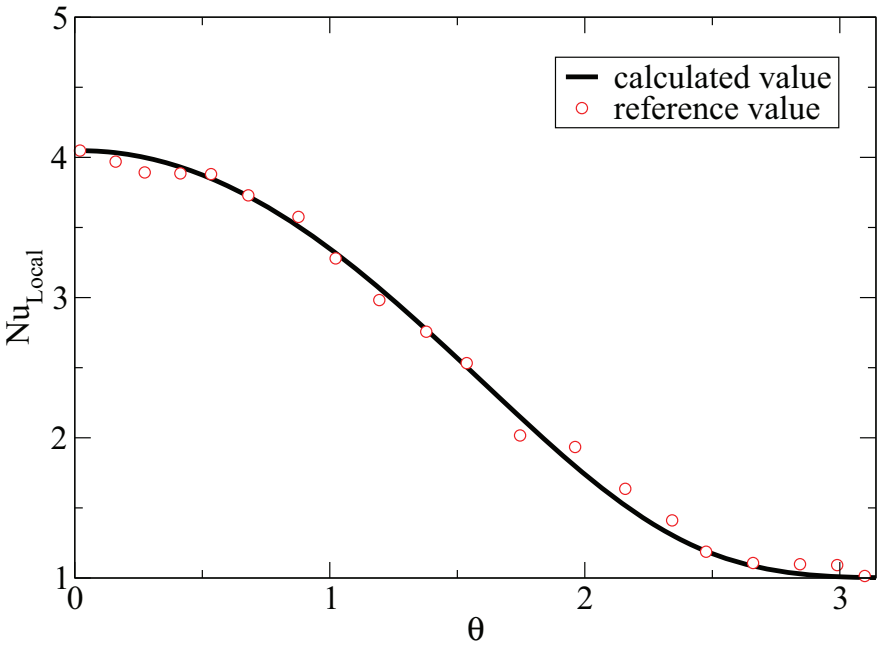


Figure 4.17: Local Nusselt numbers on the cylinder wall at $Re = 20$, $Pr = 0.7$ for the CWT case

4.4 Summary and Conclusions

In this chapter, CFD is used to calculate the convective heat transfer coefficients for the CWT case and the UHF case for two problems: flow between parallel flat plates and flow past a cylinder. The numerical results are in good agreement with the values reported in the literature.

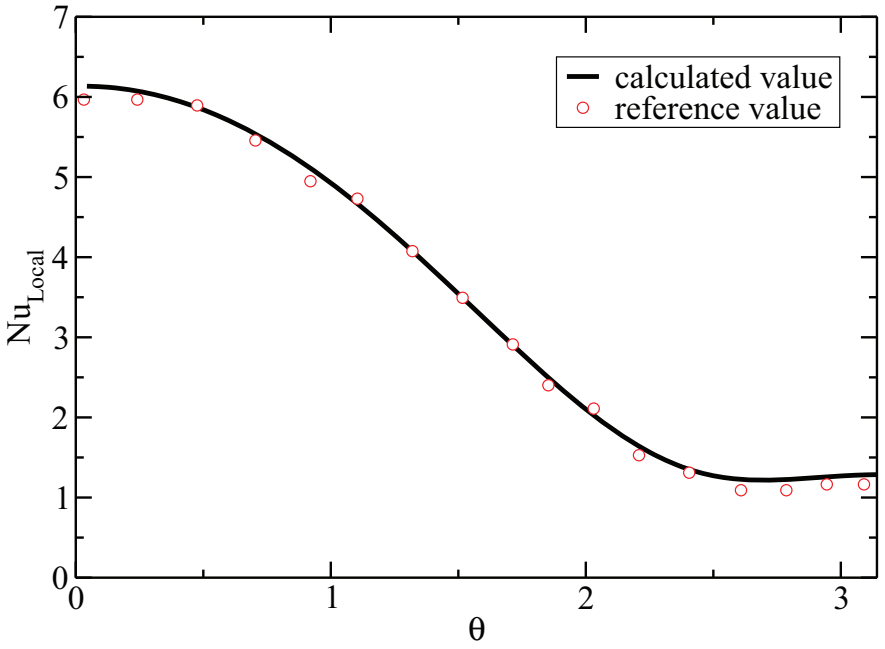


Figure 4.18: Local Nusselt numbers on the cylinder wall at $Re = 45$, $Pr = 0.7$ for the CWT case

Table 4.12
Comparison of the average Nusselt number ($Pr = 0.7$) with values from the literature for the CWT case

Source	$Re = 10$	$Re = 20$	$Re = 40$
Present results	1.8512	2.4481	3.2573
Bharti et al. [7]	1.8623	2.4653	3.2825
Badr [5]	-	2.5400	3.4800
Dennis et al. [17]	1.8673	2.5216	3.4317
Lange et al. [46][a]	1.8101	2.4087	3.2805
Soares et al. [77]	1.8600	2.4300	3.2000
Sparrow et al. [80][b]	1.6026	2.2051	3.0821
$Nu = 0.911 Re^{0.385} Pr^{1/3}$ [54]	1.9628	2.5632	3.3472

[a]Evaluated from their equation
[b]Experimental correlation

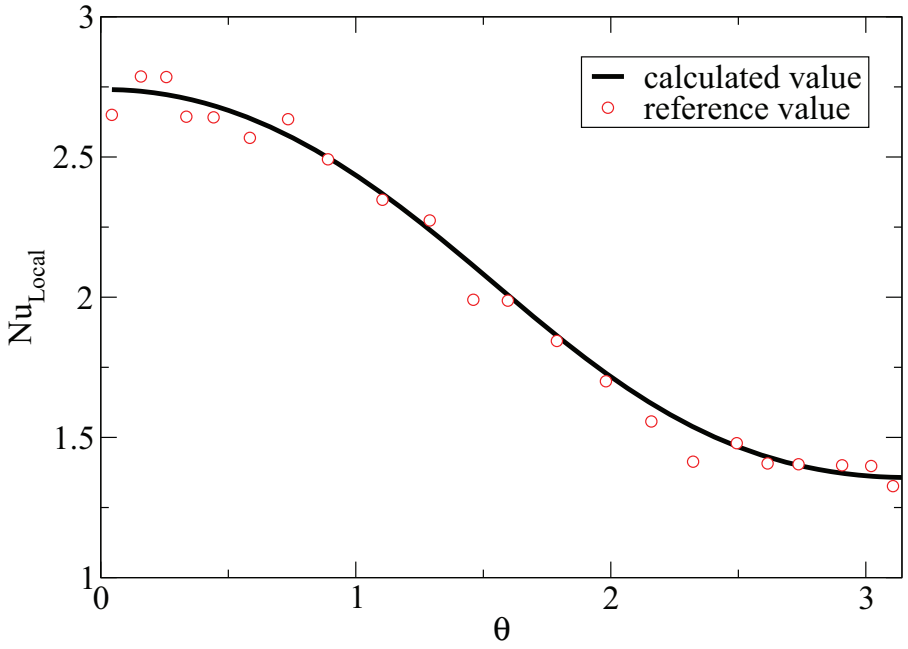


Figure 4.19: Local Nusselt numbers on the cylinder wall at $Re = 10$, $Pr = 0.7$ for the UHF case

Table 4.13
Comparison of the average Nusselt number ($Pr = 0.7$) with values from the literature for the UHF case

Source	$Re = 10$	$Re = 20$	$Re = 40$
Present results	2.0283	2.7695	3.7630
Bharti et al. [7]	2.0400	2.7788	3.7755
Ahmad et al. [1]	2.0410	2.6620	3.4720
Dhiman et al. [19]	2.1463	2.8630	3.7930

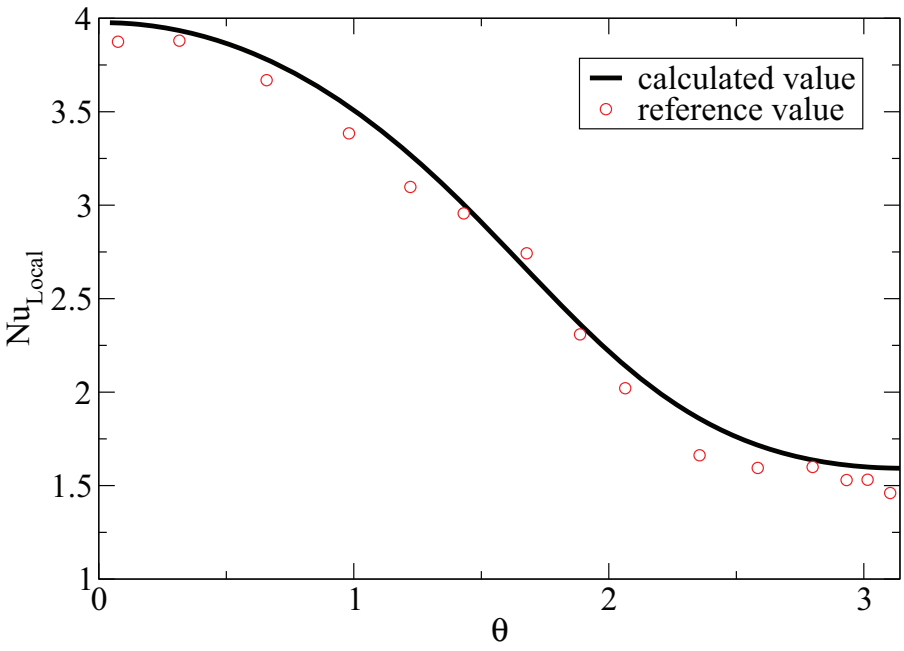


Figure 4.20: Local Nusselt numbers on the cylinder wall at $Re = 20$, $Pr = 0.7$ for the UHF case

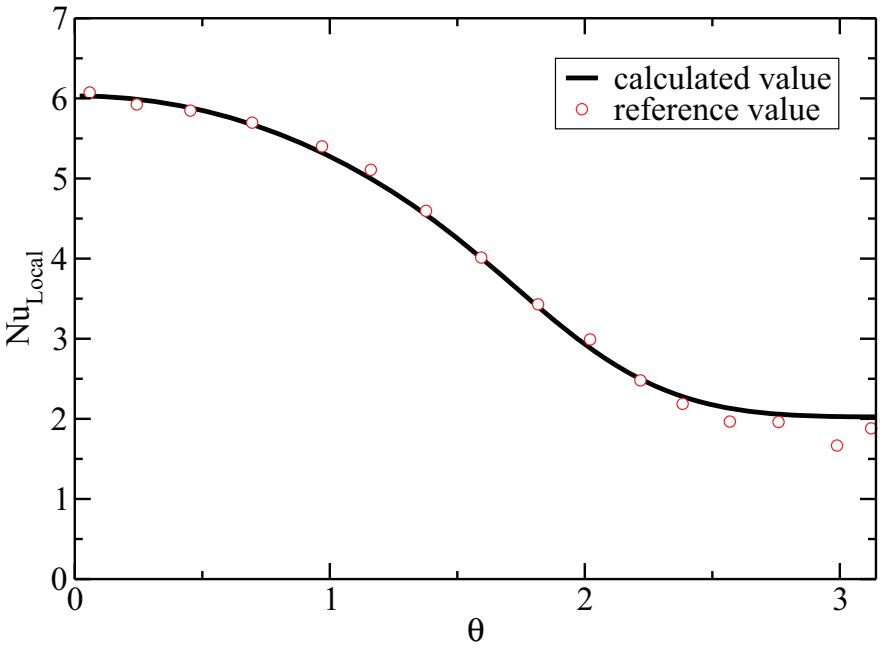


Figure 4.21: Local Nusselt numbers on the cylinder wall at $Re = 45$, $Pr = 0.7$ for the UHF case

Chapter 5

Droplet Solidification in Cold Airflow

Spray freezing processes involve solidification of millions of droplets in cold airflows. However, the present-day computational technology does not have the capacity to resolve these millions of droplets. One of the limitations of the heat transfer calculations in Chapter 4 is that the droplet is treated as a solid. In this chapter, the drop is treated as a liquid (water) but still undeformed with zero velocity on the boundary. The solidification inside the drop is simulated using an enhanced enthalpy-porosity model [6] (see Appendix A for the development and validation of the code), and the density change of water is accounted for by using the fourth-order temperature polynomial [45]. The modeling and simulations presented in this chapter give insight

and information that can be used for the development and improvement of simpler solidification models, as are, for example, used in sprays (see [84] for details).

5.1 Problem Description

The problem of droplet solidification in cold airflow is studied by simulating the solidification of a stationary spherical water droplet of 1 mm diameter in an external cold airflow. The physical properties of water, ice and air used in the simulations are shown in Table 5.1.

Table 5.1
Properties of water, ice and air used in the simulations

Material properties	value	Unit
ρ_l denity of water	999.8	kg/m^3
ρ_s denity of ice	916.8	kg/m^3
μ_l dynamic viscosity of water	0.001003	kg/ms
ν_0 kinematic viscosity of air	1.46e-5	m^2/s
k_l thermal conductivity of water	0.6	W/mK
k_s thermal conductivity of ice	2.26	W/mK
k_0 thermal conductivity of air	0.0242	W/mK
c_l specific heat capacity of water	4182.0	J/kgK
c_s specific heat capacity of ice	2116.0	J/kgK
α_0 thermal diffusivity of air	1.96e-5	m^2/s
L_f latent heat of fusion	335000	m^2/s^2
T_l liquid temperature	273.30	K
T_s solid temperature	273.00	K
g gravitational acceleration	9.81	m/s^2

The computational setup is shown schematically in Fig. 5.1 for the axisymmetric

simulation.

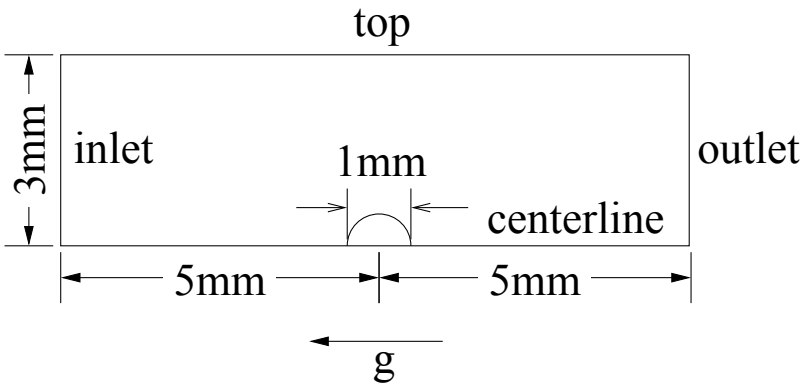


Figure 5.1: Axisymmetric computational domain.

Initially, the water droplet has a temperature of 274 K and the airflow has a temperature of 256 K with velocity 0.146 m/s corresponding to $Re = 10$. We apply the constant velocity 0.146 m/s, constant temperature 256 K and a zero normal gradient of pressure at the inlet, and we apply the constant pressure of 0 and a zero normal gradient of velocity and temperature at the outlet. The slip boundary condition is applied at the top.

5.2 Mathematical Model and Numerical Methods

The droplet solidification in the cold airflow is a multi-physics problem. Before the droplet starts to solidify, it is a liquid-gas two phase flow problem. Then the droplet starts to solidify from the outer surface of the drop. Once it is fully solidified on the outer surface of the drop, it becomes a conjugate heat transfer-type problem, in

which conduction in the solid and convection in the fluid must both be considered. The difference between this problem and a typical conjugate heat transfer problem is that instead of pure solid, it contains another liquid-solid phase change problem inside the drop. We assume that the drop is fixed in the computational domain with no drop deformation, and that the outer surface of the drop is solid, i.e., the velocity is zero on the outer surface of the drop.

For the air, since the Mach number is very small, incompressible Newtonian fluid model is used:

Conservation of mass

$$\nabla \cdot \mathbf{v} = 0, \tag{5.1}$$

Conservation of momentum

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = \nabla \cdot (\nu_0 \nabla \mathbf{v}) - \frac{1}{\rho_0} \nabla p + \mathbf{g}, \tag{5.2}$$

Conservation of energy

$$\frac{\partial T}{\partial t} + \nabla \cdot (\mathbf{v}\, T) - \nabla \cdot (\alpha_0 \nabla T) = 0, \tag{5.3}$$

where ρ_0 is the density of air, ν_0 is the kinematic viscosity of air, and α_0 is the thermal

diffusivity of air.

For the droplet, the enhanced enthalpy-porosity model (see Appendix A and [6] for details) is used to simulate the solidification of the water droplet:

Conservation of mass

$$\nabla \cdot \mathbf{v} = 0, \tag{5.4}$$

Conservation of momentum

$$\rho_l \frac{\partial \mathbf{v}}{\partial t} + \rho_l \mathbf{v} \cdot \nabla \mathbf{v} = \nabla \cdot (\mu \nabla \mathbf{v}) - \nabla p + \mathbf{S} + (\rho(T) - \rho_l)\mathbf{g}, \tag{5.5}$$

Conservation of energy

$$c_{mix} \frac{\partial T}{\partial t} + c_{mix}(\mathbf{v} \cdot \nabla T) + \rho_l L_f \frac{\partial \alpha}{\partial t} + \rho_l L_f (\mathbf{v} \cdot \nabla \alpha) - \nabla \cdot (k_{mix} \nabla T) = 0, \tag{5.6}$$

where

$$\rho(T) = \alpha \rho_l(T) + (1 - \alpha)\rho_s, \tag{5.7}$$

and $\rho_l(T)$ is the density of water, a fourth-order temperature polynomial given by

Kowalewski and Rebow [45]:

$$\begin{aligned}\rho_l(T) &= 999.840281167 + 0.0673268037314 \cdot T - 0.00894484552601 \cdot T^2 \\ &+ 8.78462866500 \cdot 10^{-5} \cdot T^3 + 6.62139792627 \cdot 10^{-7} \cdot T^4, \end{aligned} \tag{5.8}$$

where the temperature T is in degrees Celsius. Equation (5.8) is graphically represented in Fig. 5.2.

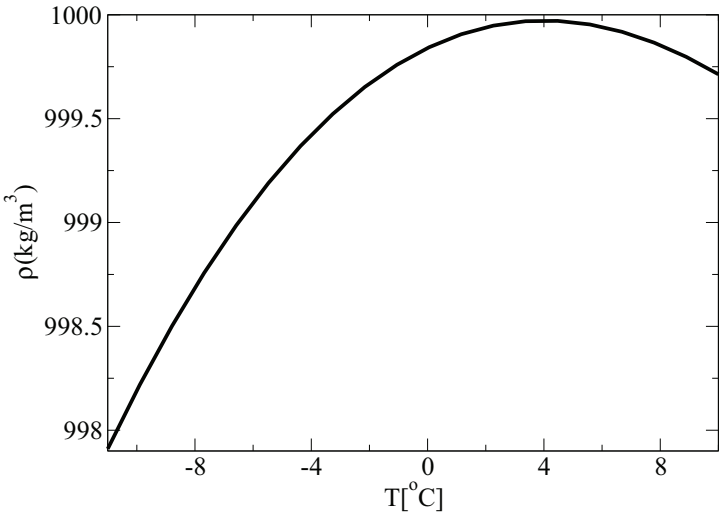


Figure 5.2: Plot of Eq. (5.8).

$$c_{mix} = \alpha\rho_l c_l + (1-\alpha)\rho_s c_s, \tag{5.9}$$

$$k_{mix} = \alpha k_l + (1 - \alpha) k_s. \tag{5.10}$$

S is the Darcy source term, and is defined as

$$\mathbf{S} = -C \frac{(1-\alpha)^2}{\alpha^3 + \epsilon} \mathbf{v}, \tag{5.11}$$

where $C = 10^8$, $\epsilon = 10^{-8}$ in our simulations, and α is the liquid fraction defined by

$$\alpha = \begin{cases} 0 & \text{if } T < T_s \\ \frac{T - T_s}{T_l - T_s} & \text{if } T_s < T < T_l \\ 1 & \text{if } T > T_l \end{cases}, \tag{5.12}$$

where T_s and T_l are solid and liquid temperatures of water, respectively. The other physical properties are constant and given in Table 5.1.

The coupling between the airflow and the droplet is ensured at the interface using a Flux Forward Temperature Backward (FFTB) method. The method is as follows: At the interface, the heat flux and the temperature must be conserved. This is achieved by introducing an inner loop which iterates the said quantities in the two domains on either side of the interface. The boundary heat flux at the interface in Domain 2 is prescribed equal to the calculated heat flux in Domain 1 (Flux Forward), i.e.,

$$\mathbf{q}_2^{n+1} = \mathbf{q}_1^n. \tag{5.13}$$

With this Neumann boundary condition, the temperature at the interface in Domain 2 can be calculated and is then used in Domain 1 as a Dirichlet boundary condition (Temperature Back), i.e.,

$$T_1^{n+1} = T_2^n. \tag{5.14}$$

The temperature and the heat flux are then calculated in Domain 1 and the loop is iterated till the temperature and the heat flux differences at the interface between the two domains are below a desired tolerance. When the convergence is reached, the physical time step is incremented.

5.3 Mesh Independence Study

We run the simulation to time $t = 1\ s$. The standard mesh is shown in Figs. 5.3 and 5.4. The cells in the air region are graded geometrically by a factor of four such that the smallest cells are located near the droplet. This enables a finer grid near the drop where the velocity and pressure gradients are the largest.

Three meshes have been considered for the mesh independence study. In each mesh the number of cells are changed by a factor of 1.5 in each direction, resulting in a change of total number of cells by a factor of 2.25. The smallest cell sizes for each mesh, together with the total number of cells, are listed in Table 5.2.

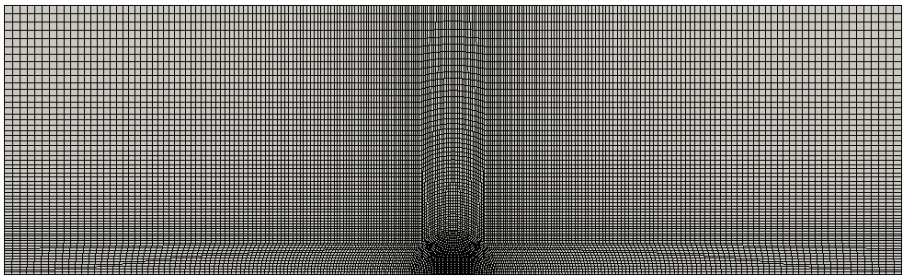


Figure 5.3: Axisymmetric computational mesh.

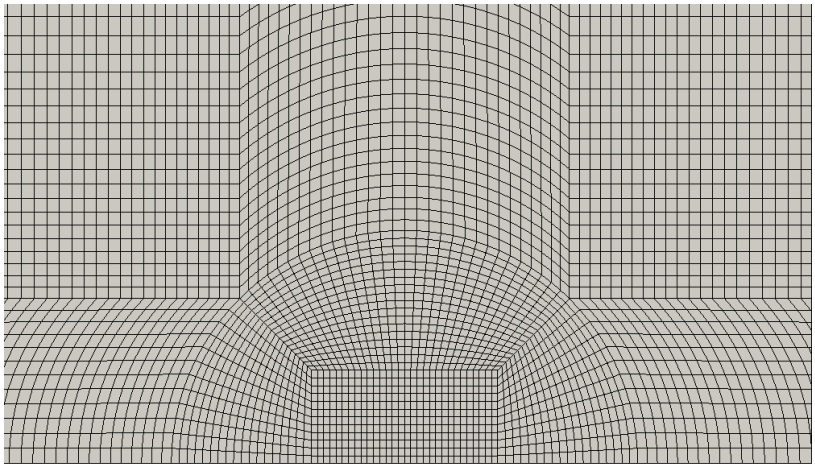


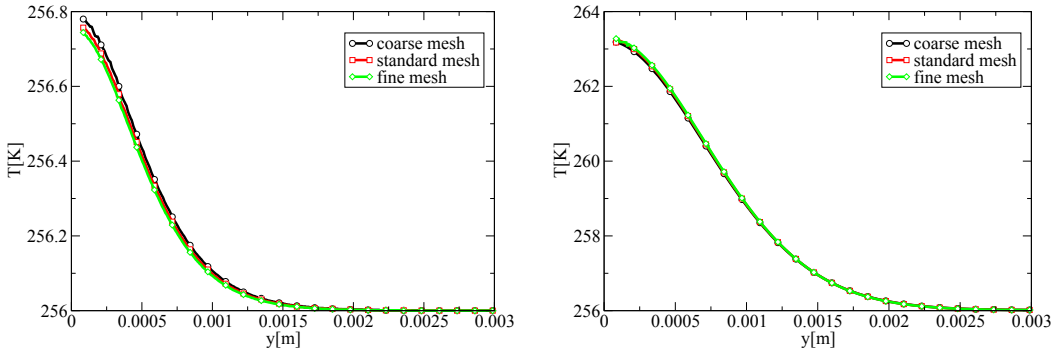
Figure 5.4: Enlargement of the axisymmetric computational mesh near the droplet.

The criteria for judging the mesh independence are the temperature, T, the velocity x-component, U_x, and the pressure, p, in the y-direction at 1 mm in front of and behind the drop, as well as the temperature, T, the velocity x-component, U_x, in the y-direction in the middle of the drop at the simulation time $t = 0.2$ s. As is seen in Figs. 5.5 - 5.9, the curves corresponding to the standard and the fine mesh are closer

Table 5.2
Meshes used in the axisymmetric simulations

Mesh	Number of cells	Smallest cell size $[mm]$
Coarse	$15,108$	0.0133×0.0133
Standard	$33,900$	0.0089×0.0089
Fine	$76,048$	0.0059×0.0059

than the curves between the coarse and the standard mesh. From these figures we can conclude that sufficient mesh independence has been achieved with the standard mesh.

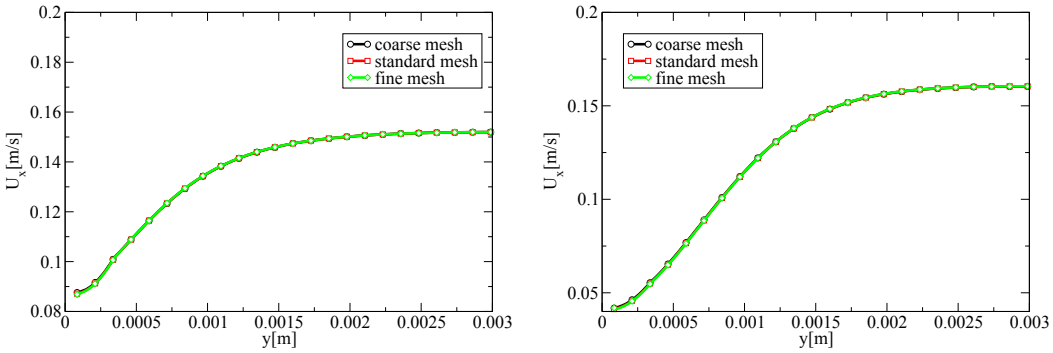


(a) (b)

Figure 5.5: Temperature (a) along the line $x = -1\ mm$ at $t = 0.2\ s$; (b) along the line $x = 1\ mm$ at $t = 0.2\ s$.

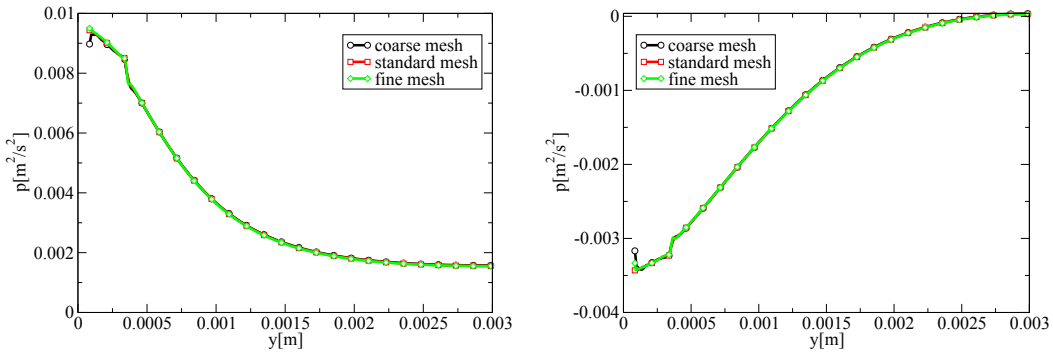
5.4 Results and Discussion

The water-ice interface is shown in Fig. 5.10 for the simulation times of 0.2 s, 0.4 s and 0.6 s. The solidification starts inward from the surface of the spherical water



(a) (b)

Figure 5.6: Velocity x-component (a) along the line $x = -1\ mm$ at $t = 0.2\ s$; (b) along the line $x = 1\ mm$ at $t = 0.2\ s$.



(a) (b)

Figure 5.7: Pressure (a) along the line $x = -1\ mm$ at $t = 0.2\ s$; (b) along the line $x = 1\ mm$ at $t = 0.2\ s$.

droplet because of the cold airflow around the drop. The water-ice interface is not symmetric along the line $x = 0\ mm$, i.e., the solidification rate is decreasing from the front stagnation point to the rear stagnation point along the surface of the spherical drop. This is because there is the largest heat transfer at the front stagnation point

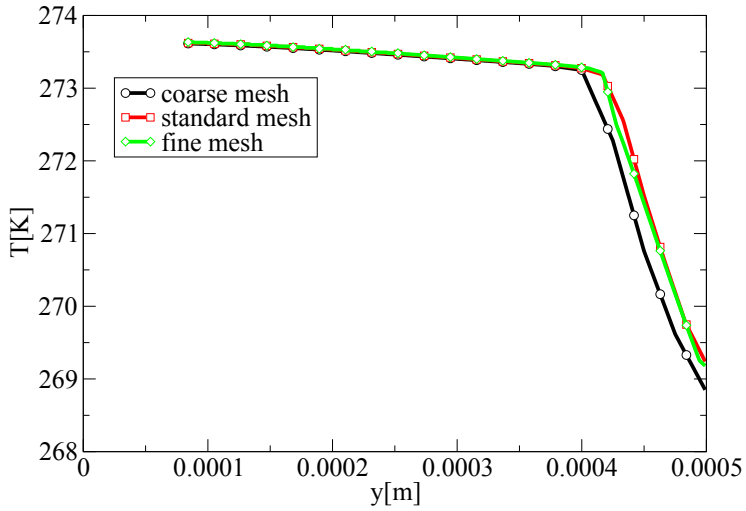


Figure 5.8: Temperature along the line $x = 0\ mm$ at $t = 0.2\ s$.

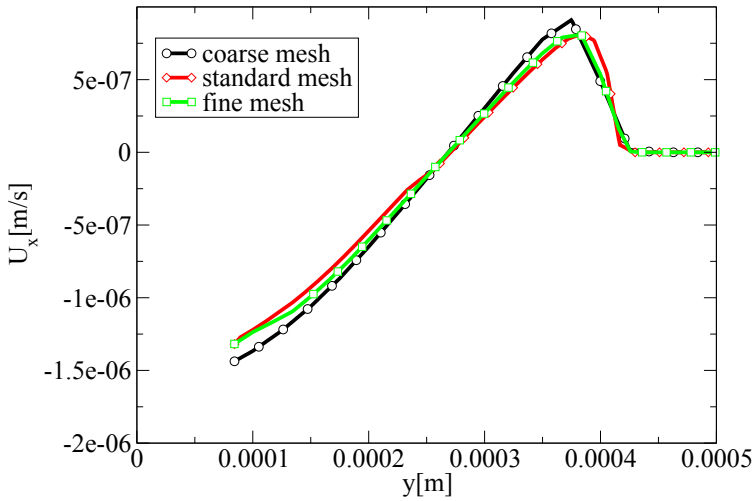


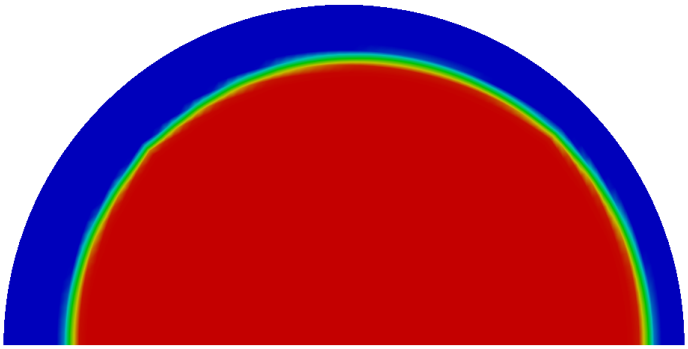
Figure 5.9: Velocity x-component along the line $x = 0\ mm$ at $t = 0.2\ s$.

and the rate of heat transfer decreases as the air is heated when it passes around the droplet.

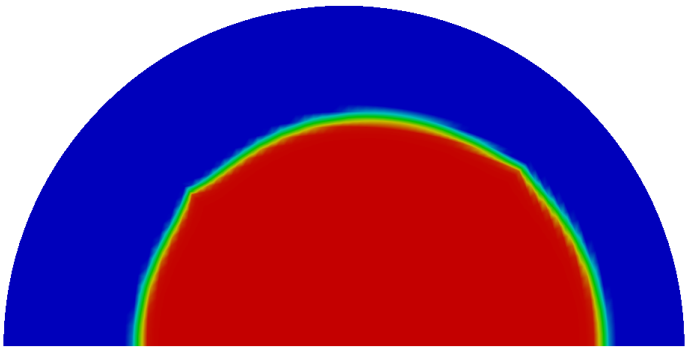
The velocity field inside the drop is shown in Fig. 5.11. Because the density of water is decreasing with decreasing temperature (for temperatures below 277.15 K), the water near the front stagnation point becomes less dense. It then starts to flow from the front stagnation point to the rear stagnation point along the phase-dividing interface of the drop, and then forms a circulation.

Some other information can also be extracted from the numerical results. For example, it needs around 0.9 s for the entire solidification. At $t = 0.1$ s, the thickness of the ice shell is between 0.023 mm and 0.047 mm, which indicates that after 0.1 s, the drop can be treated as a solid when calculating the convective heat transfer coefficient on the interface between the drop and airflow.

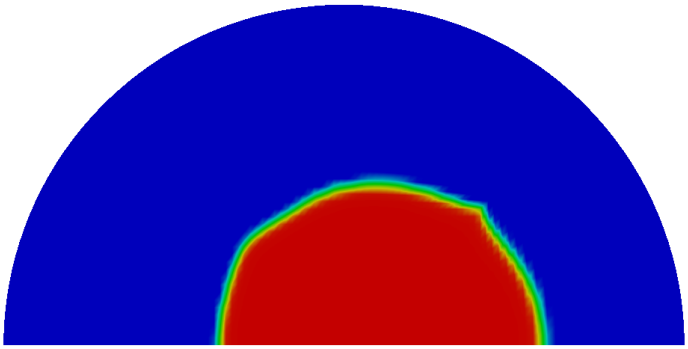
Qualitatively, the numerical results are physically reasonable. In order to obtain a more realistic behavior of the solidification process of a drop in a free stream, the current assumptions need to be adapted to a more general flow situation. More specifically, the original liquid-gas interface cannot be assumed to be fixed and the drop deformation, as investigated in Chapter 3, has to be taken into account. This, however, is the subject of a future study.



(a)



(b)



(c)

Figure 5.10: water (red) and ice (blue) inside the droplet at (a) $t = 0.2\ s$; (b) $t = 0.4\ s$; (c) $t = 0.6\ s$.

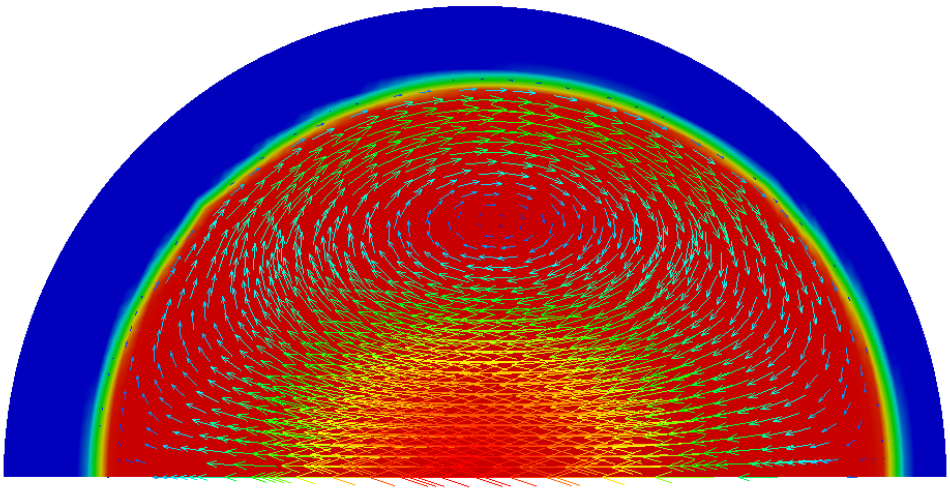


Figure 5.11: Velocity field inside the drop at $t = 0.2\ s$ (red: water, blue: ice).

Chapter 6

Summary and Future Work

This research focused on computational methods for the investigation of liquid drop phenomena in external gas flows. These phenomena include droplet deformation and breakup, convective heat transfer between liquid drop and gas flow, and droplet solidification in cold airflow. Computational solvers were developed or modified within the OpenFOAM® environment to study each of these phenomena.

First, to study droplet deformation and breakup, a two-phase flow solver, `interSEAFoam`, has been implemented into OpenFOAM®. This solver is based on the standard two-phase flow solver `interFoam`. It utilizes the Shifted Eulerian Adaptation (SEA) method which adjusts the location of the drop every N time steps, and

thus ensures that the drop remains within the fixed computational domain. Axisymmetric CFD simulations have been conducted to verify the deformation and breakup behavior of a drop in an air stream as predicted by the TAB model and as observed in experiments. The original TAB model has been modified to account for the change in the aerodynamic drag due to the drop deformation. It is found that the initiation time and the drop deformation at the initiation time for the bag breakup, the stamen breakup, and the stripping breakup regimes are in good agreement with the modified TAB model.

Three dimensional symmetric simulations have also been performed to study breakup behavior of a drop in an air stream for the bag, the stamen, and the stripping breakup regimes. The temporal evolution of each breakup mode is in good agreement with the experimental observations. The product drop size distribution of each breakup regime is quantified and is found to be consistent with the experimental observations.

Second, a solver for transient, incompressible, Newtonian fluids with temperature transport using an adaptive time step, `icoTempFoamVarDt`, was developed to test the accuracy and feasibility of using CFD to calculate convective heat transfer coefficients for constant wall temperature and constant heat flux boundary conditions, respectively, in two cases: flow between parallel flat plates and flow past a cylinder. The numerical results show good agreement with the literature and indicate good performance of using CFD to calculate convective heat transfer coefficients.

Finally, a solidification solver, `modPolyMeltFoam`, has been implemented into OpenFOAM®. This solver is based on an enhanced enthalpy-porosity model for phase change with natural convection [6]. In this model, the different thermophysical properties of the liquid and solid phases are taken into consideration. The code has been tested for pure natural convection of water in a cavity and solidification of water in a cavity. The results are in good agreement with the existing numerical results and experimental observations from the literature.

A fluid-fluid conjugate heat transfer solver with the assumption of zero velocity interface condition, `modFluidFluidChtMultiRegionFoam`, has been implemented into OpenFOAM®. This solver is based on `icoFoam`, `chtMultiRegionFoam` and `modPolyMeltFoam`. This solver has been used to simulate the solidification of a water droplet in a cold airflow. Qualitatively, the numerical results are physically reasonable.

Future Work

The computational solvers developed within the course of this research represent a significant step toward allowing the detailed computational investigation of liquid drop phenomena in external gas flows. To better analyze and understand these phenomena, further work is needed. Future work may include, but is not limited to:

† Further develop the modified TAB model with varying drag coefficient.

† Perform fully three dimensional simulations of drop deformation and breakup.

† Calculate convective heat transfer coefficients along the interface between the water drop and the surrounding air to build correlations.

† Further develop the fluid-fluid conjugate heat transfer solver to incorporate additional physics, e.g., drop deformation and interfacial velocities, and use it to study the solidification of a water drop in a cold airflow.

References

[1] Ahmad, R. A. and Qureshi, Z. H., Laminar Mixed Convection from a Uniform Heat Flux Horizontal Cylinder in a Crossflow, *Journal of Thermophysics and Heat Transfer*, vol. **6(2)**, pp. 277–287, 1992.

[2] Anandharamakrishnan, C., CFD Applications for the Food Industry, *Indian Food Industry*, vol. **22(6)**, pp. 62–68, 2003.

[3] Ashgriz, N., *Handbook of Atomization and Sprays: Theory and Applications*, Springer Science & Business Media, 2011.

[4] Babinsky, E. and Sojka, P. E., Modeling Drop Size Distributions, *Progress in Energy and Combustion Science*, vol. **28**, pp. 303–329, 2002.

[5] Badr, H. M., A Theoretical Study of Laminar Mixed Convection from a Horizontal Cylinder in a Cross Stream, *International Journal of Heat and Mass Transfer*, vol. **26(5)**, pp. 639–653, 1983.

[6] Belhamadia, Y., Kane, A. S., and Fortin, A., An Enhanced Mathematical Model for Phase Change Problems with Natural Convection, *International Journal of numerical analysis and modeling, series B*, vol. **3(2)**, pp. 192–206, 2012.

[7] Bharti, R. P., Chhabra, R. P., and Eswaran, V., A Numerical Study of the Steady Forced Convection Heat Transfer from an Unconfined Circular Cylinder, *Heat Mass Transfer*, vol. **43**, pp. 639–648, 2007.

[8] Blocken, B., Defraeye, T., Derome, D., and Carmeliet, J., High-resolution CFD Simulations for Forced Convective Heat Transfer Coefficients at the Facade of a Low-rise Building, *Building and Environment*, vol. **44(12)**, pp. 2396–2412, 2009.

[9] Bozzi, L. A., Feng, J. Q., Scott, T. C., and Pearlstein, A. J., Steady Axisymmetric Motion of Deformable Drops Falling or Rising Through a Homoviscous Fluid in a Tube at Intermediate Reynolds Number, *Journal of Fluid Mechanics*, vol. **336**, pp. 1–32, 1997.

[10] Brent, A. D., Voller, V. R., and Reid, K. J., Enthalpy-porosity Technique for Modeling Convection-diffusion Phase Change: Applications to the Melting of a Pure Metal, *Numerical Heat Transfer*, vol. **13**, pp. 297–318, 1988.

[11] Buren, P. D. V. and Viskanta, R., Interferometric Measurement of Heat Transfer During Melting from a Vertical Surface, *International Journal of Heat and Mass Transfer*, vol. **23**, pp. 568–571, 1980.

[12] Case, W. R., Dubey, B. N., Tanner, F. X., Feigl, K. A., and Windhab, E. J., Computational Analysis of the Spray Processing of Multiple Emulsion Structures, Poster presented at ISFRS 2012, Sixth International Symposium on Food Rheology and Structure, Zurich, Switzerland, April, 2012.

[13] Chou, W.-H. and Faeth, G. M., Temporal Properties of Secondary Drop Breakup in the Bag Breakup Regime, *International Journal of Multiphase Flow*, vol. **24**, pp. 889–912, 1998.

[14] Dai, Z. and Faeth, G. M., Temporal Properties of Secondary Drop Breakup in the Multimode Breakup Regime, *International Journal of Multiphase Flow*, vol. **27(2)**, pp. 217–236, 2001.

[15] Dandy, D. S. and Leal, L. G., Buoyancy-driven Motion of a Deformable Drop Through a Quiescent Liquid at Intermediate Reynolds Numbers, *Journal of Fluid Mechanics*, vol. **208**, pp. 161–192, 1989.

[16] Defraeye, T., Blocken, B., and Carmeliet, J., CFD Analysis of Convective Heat Transfer at the Surfaces of a Cube Immersed in a Turbulent Boundary Layer, *International Journal of Heat and Mass Transfer*, vol. **53(1)**, pp. 297–308, 2010.

[17] Dennis, S. C. R., Hudson, J. D., and Smith, N., Steady Laminar Forced Convection from a Circular Cylinder at Low Reynolds Numbers, *Physics of Fluids*, vol. **11(5)**, pp. 933–940, 1968.

[18] Deshpande, S. S., Anumolu, L., and Trujillo, M. F., Evaluating the Performance of the Two-phase Flow Solver interFoam, *Computational science & discovery*, vol. **5(1)**, p. 014016, 2012.

[19] Dhiman, A. K., Chhabra, R. P., Sharma, A., and Eswaran, V., Effects of Reynolds and Prandtl Numbers on Heat Transfer Across a Square Cylinder in the Steady Flow Regime, *Numerical Heat Transfer, Part A: Applications*, vol. **49(7)**, pp. 717–731, 2006.

[20] Dumouchel, C., The Maximum Entropy Formalism and the Prediction of Liquid Spray Drop-size Distribution, *Entropy*, vol. **11**, pp. 713–747, 2009.

[21] Evans, K. J. and Knoll, D. A., Temporal Accuracy Analysis of Phase Change Convection Simulations Using the JFNK-SIMPLE Algorithm, *International Journal for Numerical Methods in Fluids*, vol. **55(7)**, pp. 637–653, 2007.

[22] Evans, K. J., Knoll, D. A., and Pernice, M., Development of a 2-D Algorithm to Simulate Convection and Phase Transition Efficiently, *Journal of Computational Physics*, vol. **219**, pp. 404–417, 2006.

[23] Faeth, G. M., Hsiang, L. P., and Wu, P. K., Structure and Breakup Properties of Sprays, *International Journal of Multiphase Flow*, vol. **21**, pp. 99–127, 1995.

[24] Florez, W. F., Power, H., and Chejne, F., Numerical Solution of Thermal Convection Problems Using the Multi-domain Boundary Element Method, *Numerical Methods for Partial Differential Equations*, vol. **18(4)**, pp. 469–489, 2002.

[25] FLUENT®, Ansys, Inc., http://http://www.ansys.com/Products/Fluids/ANSYS-Fluent, , 2016.

[26] Francois, M., Cummins, S., Dendy, E., Kothe, D., Sicilian, J., and Williams, M., A Balanced-force Algorithm for Continuous and Sharp Interfacial Surface Tension Models within a Volume Tracking Framework, *Journal of Computational Physics*, vol. **213**, pp. 141–173, 2006.

[27] Gartling, D. K., 1980. Computer Methods in Fluids. Pentech, London, Ch. Finite Element Analysis of Convective Heat Transfer Problems with Change of Phase, pp. 257–284.

[28] Gelfand, B. E., Droplet Breakup Phenomena in Flows with Velocity Lag, *Progress in Energy and Combustion Science*, vol. **22(3)**, pp. 201–265, 1996.

[29] Grover, R. O., Assanis, D. N., Lippert, A. M., Tahry, S. H., Drake, M. C., Fansler, T. D., and Harrington, D. L., A Critical Analysis of Splash Criteria for GDI Spray Impingement, *ILASS Americas, 15th Annual Conference on Liquid Atomization and Spray Systems*, 2002.

[30] Guildenbecher, D. R., Lopez-Rivera, C., and Sojka, P. E., Secondary Atomization, *Experiments in Fluids*, vol. **46(3)**, pp. 371–402, 2009.

[31] Gupta, S. C., A Moving Grid Numerical Scheme for Multi-dimensional Solidification with Transition Temperature Range, *Computer Methods in Applied Mechanics and Engineering*, vol. **189(2)**, pp. 525–544, 2000.

[32] Han, J. and Tryggvason, G., Secondary Breakup of Axisymmetric Liquid Drops. I. Acceleration by a Constant Body Force, *Physics of Fluids*, vol. **11(12)**, pp. 3650–3667, 1999.

[33] Han, J. and Tryggvason, G., Secondary Breakup of Axisymmetric Liquid Drops. II. Impulsive Acceleration, *Physics of Fluids*, vol. **13(6)**, pp. 1554–1565, 2001.

[34] Hannoun, N., Alexiades, V., and Mai, T. Z., Resolving the Controversy Over Tin and Gallium Melting in a Rectangular Cavity Heated From the Side, *Numerical Heat Transfer*, vol. **44**, pp. 253–276, 2003.

[35] Hannoun, N., Alexiades, V., and Mai, T. Z., A Reference Solution for Phase Change with Convection, *International Journal for Numerical Methods in Fluids*, vol. **48(11)**, pp. 1283–1308, 2005.

[36] Harrison, C. and Weinberg, F., The Influence of Convection on Heat Transfer in Liquid Tin, *Metallurgical and Materials Transactions B*, vol. **16(2)**, pp. 355–357, 1985.

[37] Hirahara, H. and Kawahashi, M., Experimental Investigation of Viscous Effects upon a Breakup of Droplets in High-speed Air Flow, *Experiments in Fluids*, vol. **13(6)**, pp. 423–428, 1992.

[38] Hoffmanand, J. D. and Frankel, S., *Numerical Methods for Engineers and Scientists*, CRC press, 2001.

[39] Hwang, S. S., Liu, Z., and Reitz, R. D., Breakup Mechanisms and Drag Coefficients of High-Speed Vaporizing Liquid Drops, *Atomization and Sprays*, vol. **6(3)**, pp. 353–376, 1996.

[40] Ibrahim, E. A., Yang, H. Q., and Przekwas, A. J., Modeling of Spray Droplets Deformation and Breakup, *AIAA Journal of Propulsion and Power*, vol. **9**, pp. 651–654, 1993.

[41] Issa, R. I., Solution of the Implicitly Discretised Fluid Flow Equations by Operator-splitting, *Journal of Computational Physics*, vol. **62(1)**, pp. 40–65, 1986.

[42] Jasak, H., Error Analysis and Estimation in the Finite Volume Method with Applications to Fluid Flows, PhD thesis, Imperial College, University of London, 1996. PhD thesis.

[43] Josephand, D. D., Belanger, J., and Beavers, G. S., Breakup of a Liquid Drop Suddenly Exposed to a High-speed Airstream, *International Journal of Multiphase Flow*, vol. **25(6)**, pp. 1263–1303, 1999.

[44] Kowalewski, T. A. and Rebow, M., An Experimental Benchmark for Freezing Water in the Cubic Cavity, *ICHMT DIGITAL LIBRARY ONLINE*, Begel House Inc., 1997.

[45] Kowalewski, T. A. and Rebow, M., Freezing of Water in a Differentially

Heated Cubic Cavity, *International Journal of Computational Fluid Dynamics*, vol. **11(3-4)**, pp. 193–210, 1999.

[46] Lange, C. F., Durstand, F., and Breuer, M., Momentum and Heat Transfer from Cylinders in Laminar Crossflow at $10^{-4} \leq Re \leq 200$, *International Journal of Heat and Mass Transfer*, vol. **41(22)**, pp. 3409–3430, 1998.

[47] Lécot, C., Tembely, M., Soucemarianadin, A., and Tarhini, A., Numerical Simulation of the Drop Size Distribution in a Spray, *9th International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, Springer, 2010.

[48] LeVeque, R. J., *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press, 2002.

[49] Li, X. and Tankino, R. S., Droplet Size Distribution: A Derivation of a Nukiyama-Tanasawa Type Distribution Function, *Combustion Science and Technology*, vol. **56(1-3)**, pp. 65–76, 1987.

[50] Liang, P. Y., Eastes, T. W., and Gharakhari, A., Computer Simulations of Drop Deformation and Drop Breakup, *AIAA, ASME, SAE and ASEE, 24th Joint Propulsion Conference*, 1988.

[51] Lienhard, J. H., *A Heat Transfer Textbook*, Courier Corporation, 2013.

[52] Liu, L., Mather, D., and Reitz, R., Modeling the Effects of Drop Drag and Breakup on Fuel Sprays, *SAE Technical Paper 930072*, 1993.

[53] Liu, Z. and Reitz, R. D., An Analysis of the Distortion and Breakup Mechanisms of High Speed Liquid Drops, *International Journal of Multiphase Flow*, vol. **23(4)**, pp. 631–650, 1997.

[54] Lukes, J., External Flow Correlations, `http://www.seas.upenn.edu/~meam333/correlation/CorrelationsList.pdf`, , publication date unknown, accessed March 17, 2016.

[55] Meng, J. C. and Colonius, T., Droplet Breakup in High-Speed Gas Flows, *8th International Conference on Multiphase Flow Conference*, 2013.

[56] Meryman, H. T., Sublimation Freeze Drying without a Vacuum, *Science*, vol. **130**, pp. 628–629, 1959.

[57] Michalek, T. and Kowalewski, T. A., Simulations of the Water Freezing Process - Numerical Benchmarks, *Task Quarterly*, vol. **7(3)**, pp. 389–408, 2003.

[58] Morgan, K., A Numerical Analysis of Freezing and Melting with Convection, *Computer Methods in Applied Mechanics and Engineering*, vol. **28(3)**, pp. 275–284, 1981.

[59] Neale, A., Derome, D., Blocken, B., and Carmeliet, J., Determination of Surface

Convective Heat Transfer Coefficients by CFD, *11th NBEC Canadian Building Science and Technnology Conference*, 2007.

[60] Norton, T. and Sun, D. W., CFD - an Effective and Efficient Design and Analysis Tool for the Food Industry: A Review, *Trends in Food Science and Technology*, vol. **17**, pp. 600–620, 2006.

[61] Nukiyama, S. and Tanasawa, Y., Experiments on the Atomization of Liquids in an Air Stream. Report 3: On the Droplet-size Distribution in an Atomized Jet, *Transactions of the Japan Society of Mechanical Engineers*, vol. **5**, pp. 62–67, 1939.

[62] Open, C., OpenFOAM user guide, *OpenFOAM Foundation*, vol. **2(1)**, 2011.

[63] OpenFOAM®, The OpenFOAM® Foundation, http://www.openfoam.org, , 2011.

[64] O'Rourke, P. J. and Amsden, A. A., The TAB Method for Numerical Calculation of Spray Droplet Breakup, *SAE Technical Paper 872089*, 1987.

[65] Park, S. W., Kim, S., and Lee, C. S., Breakup and Atomization Characteristics of Mono-dispersed Diesel Droplets in a Cross-flow Air Stream, *International Journal of Multiphase Flow*, vol. **32(7)**, pp. 807–822, 2006.

[66] Peric, M., A Finite Volume Method for the Prediction of Three-dimensional

Fluid Flow in Complex Ducts, PhD thesis, Imperial College London (University of London), 1985. PhD thesis.

[67] Pilch, M. and Erdman, C. A., Use of Breakup Time Data and Velocity History Data to Predict the Maximum Size of Stable Fragments for Acceleration-induced Breakup of a Liquid Drop, *International Journal of Multiphase Flow*, vol. **13(6)**, pp. 741–757, 1987.

[68] Rady, M. A. and Mohanty, A. K., Natural Convection During Melting and Solidification of Pure Metals in a Cavity, *Numerical Heat Transfer, Part A: Applications*, vol. **29(1)**, pp. 49–63, 1996.

[69] Ramsey, J. W. and Sparrow, E. M., Melting and Natural Convection Due to a Vertical Embedded Heater, *Journal of Heat Transfer*, vol. **100**, pp. 368–370, 1978.

[70] Ranger, A. A. and Nicholls, J. A., Aerodynamic Shattering of Liquid Drops, *AIAA Journal*, vol. **7(2)**, pp. 285–290, 1969.

[71] Reitz, R. D., Modeling Atomization Processes in High-pressure Vaporizing Sprays, *Atomisation Spray Technology*, vol. **3**, pp. 309–337, 1987.

[72] Rhie, C. M. and Chow, W. L., Numerical Study of the Turbulent Flow Past an Airfoil with Trailing Edge Separation, *AIAA Journal*, vol. **21(11)**, pp. 1525–1532, 1983.

[73] Rosin, P. and Rammler, E., The Laws Governing the Fineness of Powdered Coal, *Journal of the Institute of Fuel*, vol. **7**, pp. 29–36, 1933.

[74] Samarskii, A. A., Vabishchevich, P. N., Iliev, O. P., and Churbanov, A. G., Numerical Simulation of Convection/Diffusion Phase Change Problems – A Review, *International Journal of Heat and Mass Transfer*, vol. **36(17)**, pp. 4095–4106, 1993.

[75] Sellens, R. W. and Brzustowski, T. A., A Prediction of the Drop Size Distribution in a Spray from First Principles, *Atomisation Spray Technology*, vol. **1**, pp. 89–102, 1985.

[76] Sivathanu, Y. R. and Gore, J. P., A Discrete Probability Function Method for the Equation of Radiative Transfer, *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. **49(3)**, pp. 269–280, 1993.

[77] Soares, A. A., Ferreira, J. M., and Chhabra, R. P., Flow and Forced Convection Heat Transfer in Crossflow of Non-Newtonian Fluids over a Circular Cylinder, *Industrial and Engineering Chemistry Research*, vol. **44(15)**, pp. 5815–5827, 2005.

[78] Sovani, S. D., Sojka, P. E., and Sivathanu, Y. R., Prediction of Drop Size Distribution from First Principles: the Influence of Fluctuations in Relative Velocity and Liquid Physical Properties, *Atomization and Sprays*, vol. **9**, pp. 133–152, 1999.

[79] Sovani, S. D., Sojka, P. E., and Sivathanu, Y. R., Prediction of Drop Size Distribution from First Principles: Joint PDF Effects, *Atomization and Sprays*, vol. **10**, pp. 587–602, 2000.

[80] Sparrow, E. M., Abraham, J. P., and Tong, J. C. K., Archival Correlations for Average Heat Transfer Coefficients for Non-circular and Circular Cylinders and for Spheres in Cross-flow, *International Journal of Heat and Mass Transfer*, vol. **47(24)**, pp. 5285–5296, 2004.

[81] Sparrow, E. M., Patankar, S. V., and Ramadhyani, S., A Analysis of Melting in the Presence of Natural Convection in the Melt Region, *Journal of Heat Transfer*, vol. **99**, pp. 520–526, 1977.

[82] Tanner, F. X., Liquid Jet Atomization and Droplet Breakup Modeling of Non-evaporating Diesel Fuel Sprays, *SAE Technical Paper 970050*, 1997.

[83] Tanner, F. X., Development and Validation of a Cascade Atomization and Drop Breakup Model for High-velocity Dense Sprays, *Atomization and Sprays*, vol. **14(3)**, pp. 211–242, 2004.

[84] Tanner, F. X., Feigl, K. A., and Windhab, E. J., A Three-stage Freezing Model for Liquid Droplets with Applicaitons to Food Sprays, *Atomization and Sprays*, vol. **20(11)**, pp. 1005–1016, 2010.

[85] Tanner, F. X. and Weisser, G., Simulation of Liquid Jet Atomization for Fuel

Sprays by Means of a Cascade Drop Breakup Model, *SAE Technical Paper 980808*, 1998.

[86] Tate, R. W. and Marshall, W. R., Atomization by Centrifugal Pressure Nozzles, *Chemical Engineering Progress*, vol. **49(4)**, pp. 169–174, 1953.

[87] Taylor, G. I., The Shape and Acceleration of a Drop in a High Speed Air Stream, Technical Report, *Scientific Papers of G. I. Taylor. ed., G. K. Batchelor*, 1963.

[88] Toro, E. F., *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*, Springer Science & Business Media, 2013.

[89] Tukovic, H. J. Z., Dynamic Mesh Handling in OpenFOAM Applied to Fluid-structure Interaction Simulations, *Proceedings of the V European Conference Computational Fluid Dynamics, Lisbon, Portugal, June*, pp. 14–17, 2010.

[90] Viswanath, R. and Jaluria, Y., A Comparison of Different Solution Methodologies for Melting and Solidification Problems in Enclosures, *Numerical Heat Transfer Part B: Fundamentals*, 1993.

[91] Voller, V. R. and Prakash, C., A Fixed Grid Numerical Modelling Methodology for Convection-diffusion Mushy Region Phase-change Problems, *International Journal of Heat and Mass Transfer*, vol. **30(8)**, pp. 1709–1719, 1987.

[92] Wesseling, P., *Principles of Computational Fluid Dynamics*, Springer Science & Business Media, 2009.

[93] Wolfram, S., Mathematica, Technical and Scientific Software, Wolfram Research Inc., http://www.wolfram.com, , 1988-2016.

[94] Xu, T.-H., Durst, F., and Tropea, C., The Three-parameter Log-hyperbolic Distribution and its Application to Particle Sizing, *Atomization and Sprays*, vol. **3(1)**, pp. 109–124, 1993.

[95] Yao, L., Wu, X., Wu, Y., Yang, J., Gao, X., Chen, L., Gréhan, G., and Cen, K., Characterization of Atomization and Breakup of Acoustically Levitated Drops with Digital Holography, *Applied Optics*, vol. **54(1)**, pp. A23–A31, 2015.

[96] Zaleski, S., Li, J., and Succi, S., Two-dimensional Navier-Stokes Simulation of Deformation and Breakup of Liquid Patches, *Physical Review Letters*, vol. **2**, p. 75, 1995.

Appendix A

Solidification under Natural Convection

Phase change problems are important in many engineering and industrial applications. Solidification of water is an example that has received a lot of experimental and numerical attention. Unlike metal or alloys, there is a big difference between thermophysical properties of water and ice, especially the specific heat capacities. The specific heat capacity of water is about 4202 J/kgK while the specific heat capacity of ice is around 2116 J/kgK. It is important and necessary to be able to simulate water freezing process with different thermophysical properties taken into account. The enthalpy-porosity model developed by Voller and Prakash [91] is a widely used model to describe the solidification processes. This model is, however,

limited to situations where the liquid and solid thermophysical properties are equal [21, 22, 34, 35]. Recently, Belhamadia et al. [6] presented an enhanced enthalpy-porosity model that allows to consider the case where liquid and solid thermophysical properties differ. This model has been implemented into OpenFOAM® to form a new solver `modPolyMeltFoam`. Two numerical benchmarks [57] and one experimental benchmark [45] are used to validate the model and its implementation.

A.1 Natural Convection of Water in a Cavity

A.1.1 Problem Description

We consider the natural convection of water in a two dimensional heated cavity with side $L = 38\ mm$ shown in Fig. A.1. Two vertical walls are isothermal and kept at temperatures $T_H = 283\ K$ and $T_C = 273\ K$, respectively. The top and bottom walls are assumed to be adiabatic. The initial temperature of water is $T_0 = 278\ K$.

A.1.2 Mathematical Model and Numerical Methods

For describing the heat transfer in an incompressible Newtonian fluid in the laminar flow regime, the following governing equations are used:

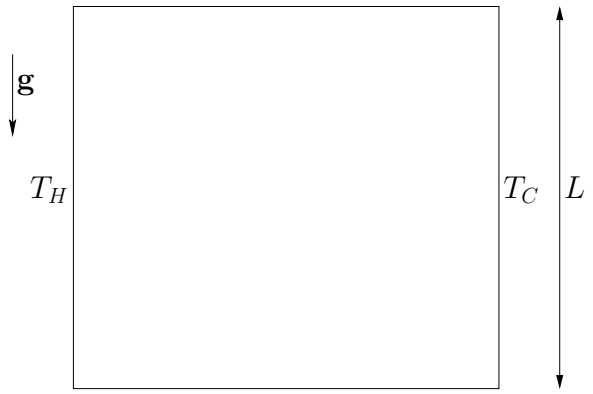


Figure A.1: Schematic of cavity.

Conservation of mass

$$\nabla \cdot \mathbf{v} = 0, \tag{A.1}$$

Conservation of momentum

$$\rho_0 \frac{\partial \mathbf{v}}{\partial t} + \rho_0 \mathbf{v} \cdot \nabla \mathbf{v} = \nabla \cdot (\mu \nabla \mathbf{v}) - \nabla p + (\rho(T) - \rho_0)\mathbf{g}, \tag{A.2}$$

Conservation of energy

$$c_p \frac{\partial T}{\partial t} + \nabla \cdot (\mathbf{v}\ c_p T) - \nabla \cdot (\frac{k}{\rho_0} \nabla T) = 0, \tag{A.3}$$

where T is the temperature, $\rho(T)$ is the density of water which depends on the temperature. ρ_0 is the water reference density, $\mathbf{v}$ is the velocity, μ is the dynamic viscosity, p is the pressure, k is the thermal conductivity and c_p is the specific heat capacity at constant pressure. Instead of using a linear variation of density, we use

a non-linear variation due to the strong non-linearity of water density around 277 K (see Fig. 5.2). The fourth-order temperature polynomial, $\rho(T)$, given by Kowalewski and Rebow [45] is used,

$$\begin{aligned} \rho(T) &= 999.840281167 + 0.0673268037314 \cdot T - 0.00894484552601 \cdot T^2 \\ &+ 8.78462866500 \cdot 10^{-5} \cdot T^3 + 6.62139792627 \cdot 10^{-7} \cdot T^4, \end{aligned} \tag{A.4}$$

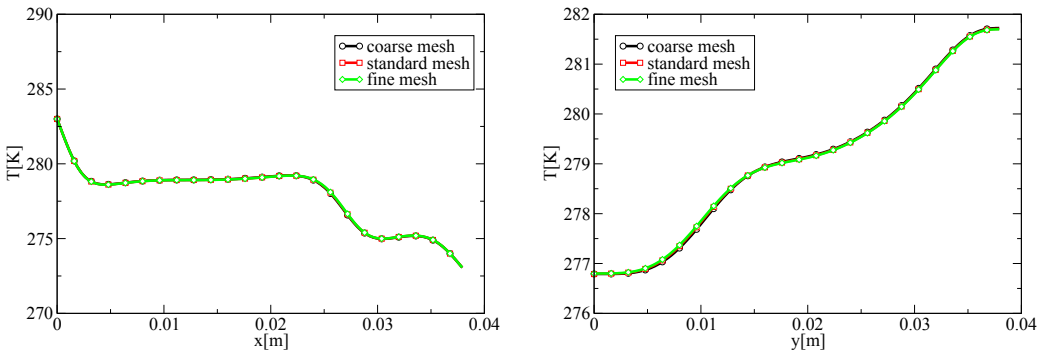
where the temperature T is given in degrees Celsius. The other physical parameters are kept constant and are given in Table A.1.

Table A.1
Properties of water used in the simulation

Material properties of water	Value	Unit
ρ_0	999.8	kg/m^3
μ	0.001003	kg/ms
$\nu = \mu/\rho_0$	$1.0032e^{-6}$	m^2/s
k	0.6	W/mK
c_p	4182.0	J/kgK
g	9.81	m/s^2

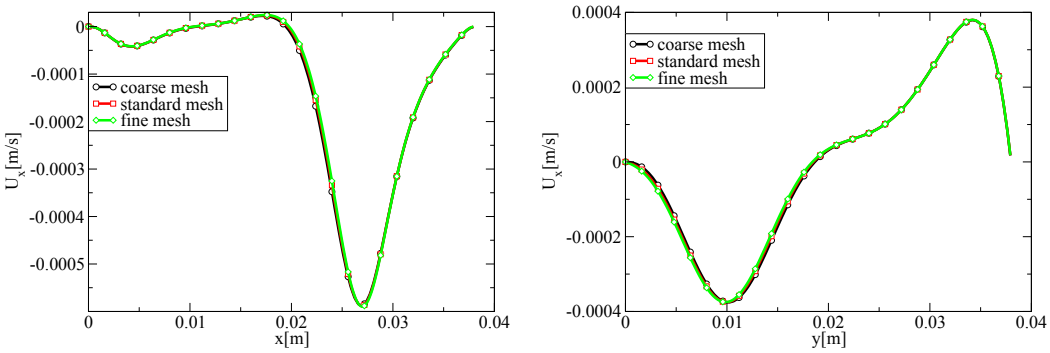
A.1.3 Mesh Independence Study

A mesh independence study has been conducted using the three meshes with 100×100 cells, 150×150 cells and 225×225 cells. The simulations are run for 800 seconds when steady state has been reached. Figures A.2 - A.4 show the temperature, T, the velocity x-component, U_x, and the velocity y-component, U_y, along the center horizontal line $y = 19\ mm$ and the center vertical line $x = 19\ mm$, respectively. From these figures, we can conclude that sufficient mesh independence has been achieved with the standard mesh. Therefore, all subsequent simulations have been performed with the standard mesh.



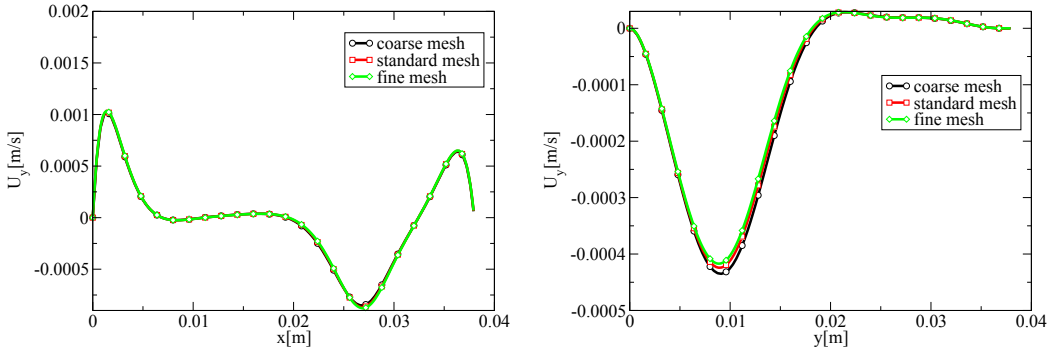
(a) (b)

Figure A.2: Temperature (a) along center horizontal line; (b) along center vertical line.



(a) (b)

Figure A.3: Velocity x-component (a) along center horizontal line; (b) along center vertical line.



(a) (b)

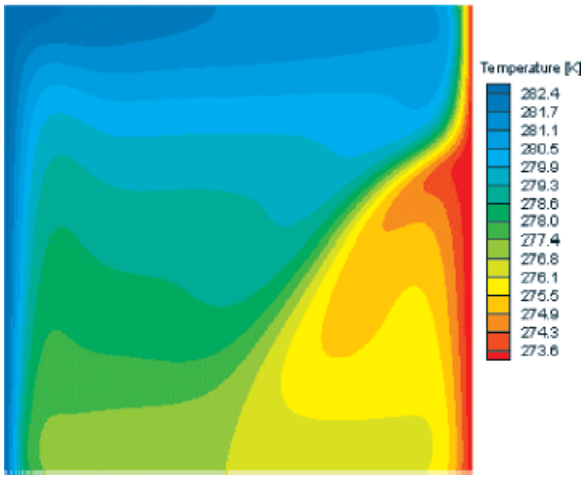
Figure A.4: Velocity y-component (a) along center horizontal line; (b) along center vertical line.

A.1.4 Validation of Numerical Results

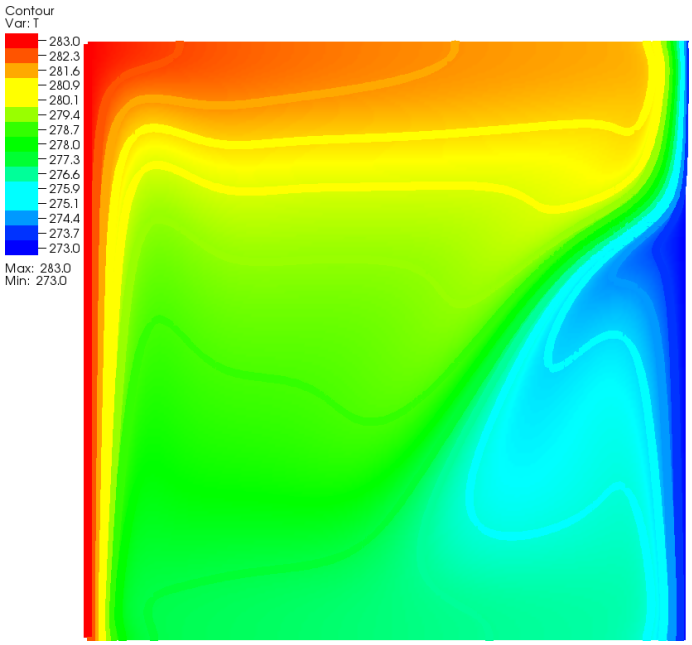
The simulation results are compared with computations performed by means of the commercial software packet FLUENT® [25] as reported in [57]. The temperature and velocity fields are shown in Figs. A.5 - A.7.

Qualitatively, there is good agreement between the OpenFOAM® and FLUENT® simulation results. These figures show that the flow pattern consists of two competing circulations. In the vicinity of the cold wall, two convection streams collide forming a clearly visible saddle point.

More detailed quantitative comparisons of the temperature and velocity are given in Figs. A.8 - A.10. As is seen in these figures, there is good quantitative agreement between the OpenFOAM® and the FLUENT® results. Since the same models are used here, the models and their implementations are validated.

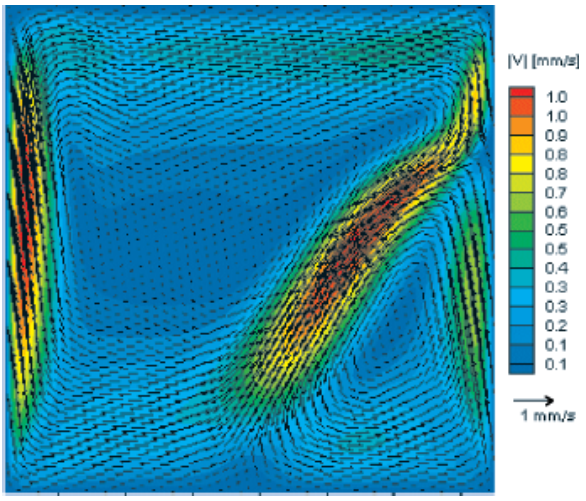


(a)

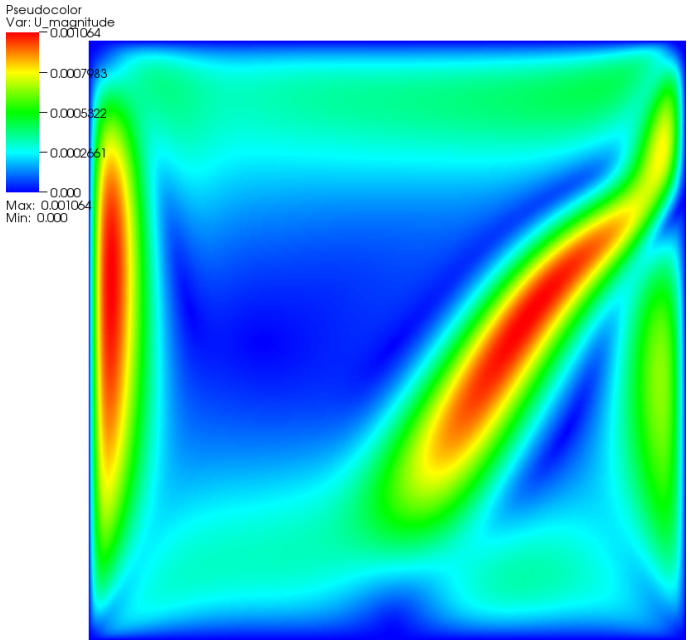


(b)

Figure A.5: Temperature field (a) FLUENT®, source: [57]; reprinted from TASK Quarterly, 7(3), Michalek and Kowalewski, Simulations of the water freezing process numerical benchmarks, p. 394, Copyright(2003), with permission from CI TASK. See documentation in Appendix C. (b) OpenFOAM®.

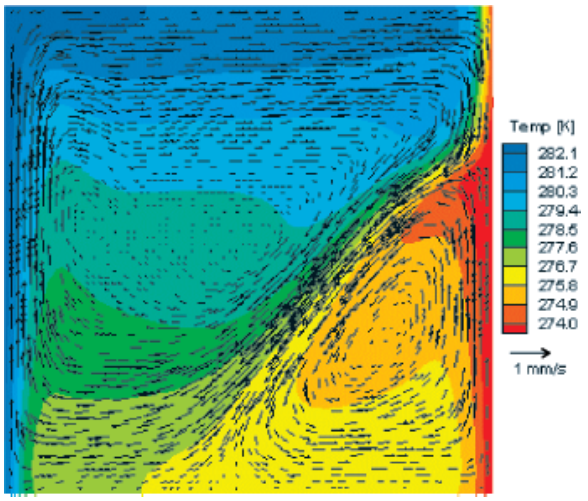


(a)

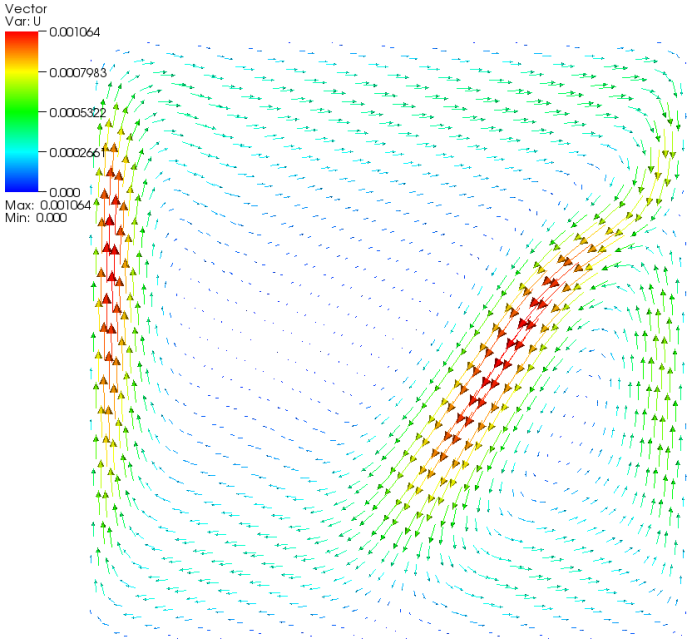


(b)

Figure A.6: Magnitude of velocity field (a) FLUENT®, source: [57]; reprinted from TASK Quarterly, 7(3), Michalek and Kowalewski, Simulations of the water freezing process numerical benchmarks, p. 394, Copyright(2003), with permission from CI TASK. See documentation in Appendix C. (b) OpenFOAM®.

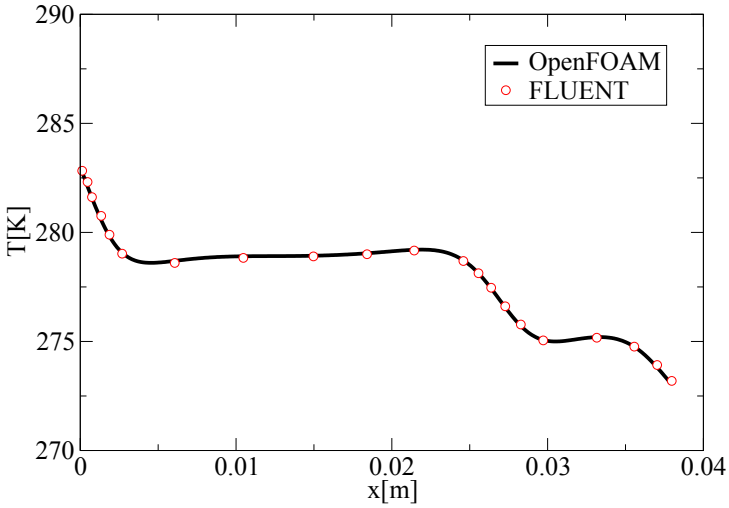


(a)

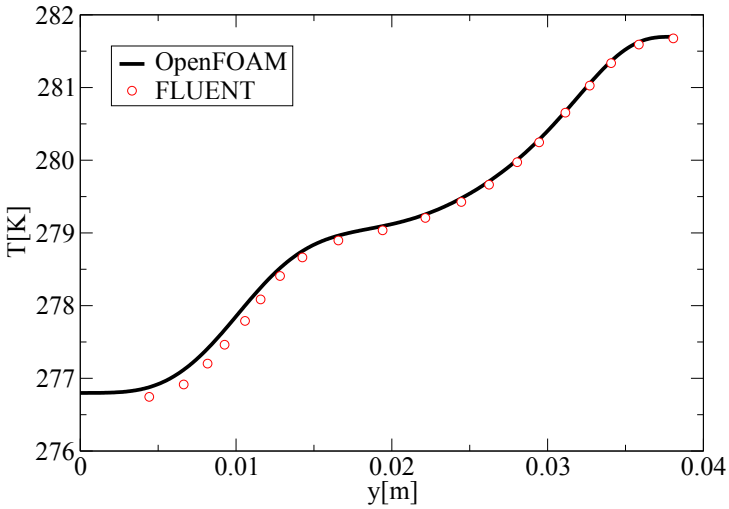


(b)

Figure A.7: Velocity vectors imposed on the temperature field (a) FLUENT®, source: [57]; reprinted from TASK Quarterly, 7(3), Michalek and Kowalewski, Simulations of the water freezing process numerical benchmarks, p. 394, Copyright(2003), with permission from CI TASK. See documentation in Appendix C. (b) OpenFOAM®.

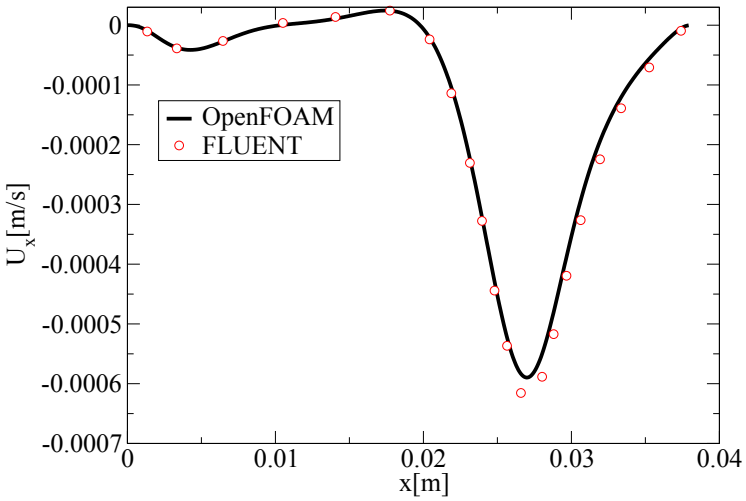


(a)

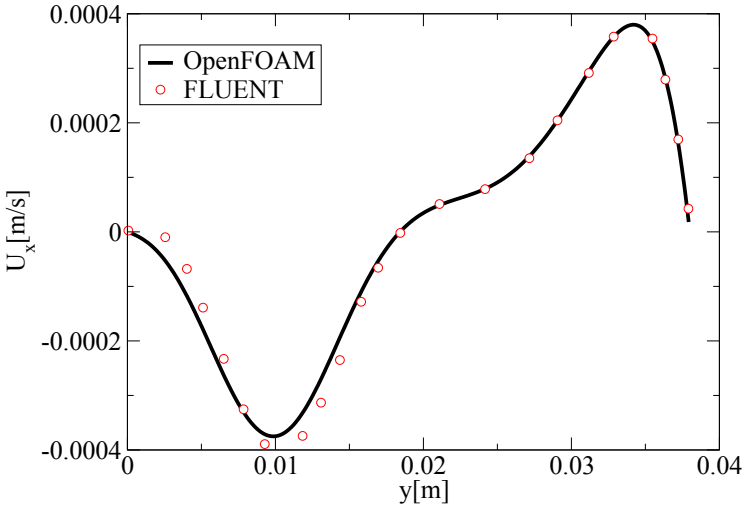


(b)

Figure A.8: Temperature (a) along center horizontal line; (b) along center vertical line. Source of FLUENT® results: Michalek and Kowalewski [57].



(a)



(b)

Figure A.9: Velocity x-component (a) along center horizontal line; (b) along center vertical line. Source of FLUENT® results: Michalek and Kowalewski [57].

A.2 Solidification of Water in a Cavity

A.2.1 Problem Description

The second numerical benchmark problem is water freezing in the same cavity as in Section A.1. We consider freezing of water after the thermal boundary condition is abruptly changed from $T_C = 273\ K$ to $T_C = 263\ K$ at the cold wall. The steady state solution obtained in Section A.1 becomes the initial condition in this section.

A.2.2 Mathematical Model and Numerical Methods

Brent et al. [10] noticed that a possible model that mimics the velocity behavior between liquid and solid phases would be that of a porous medium with the liquid flowing through a solid matrix. In a non-isothermal phase change process, this model has physical significance, whereas for an isothermal phase change process, the model is the result of numerical discretization. This model, together with the enthalpy method, is often called enthalpy-porosity model, developed by Voller and Prakash [91] to describe melting and solidification processes. This model has been widely used in many CFD software, e.g., FLUENT®.

Recently, Belhamadia et al. [6] developed an enhanced enthalpy-porosity model. This model allows to take different liquid and solid thermophysical properties into account. The governing equations are as follows:

Conservation of mass

$$\nabla \cdot \mathbf{v} = 0, \tag{A.5}$$

Conservation of momentum

$$\rho_l \frac{\partial \mathbf{v}}{\partial t} + \rho_l \mathbf{v} \cdot \nabla \mathbf{v} = \nabla \cdot (\mu \nabla \mathbf{v}) - \nabla p + \mathbf{S} + (\rho(T) - \rho_l)\mathbf{g}, \tag{A.6}$$

Conservation of energy

$$c_{mix} \frac{\partial T}{\partial t} + c_{mix}(\mathbf{v} \cdot \nabla T) + \rho_l L_f \frac{\partial \alpha}{\partial t} + \rho_l L_f (\mathbf{v} \cdot \nabla \alpha) - \nabla \cdot (k_{mix} \nabla T) = 0, \tag{A.7}$$

where

$$\rho(T) = \alpha \rho_l(T) + (1 - \alpha)\rho_s, \tag{A.8}$$

$\rho_l(T)$ is the density of water, a fourth-order temperature polynomial (see Eq. (A.4)),

$$c_{mix} = \alpha \rho_l c_l + (1 - \alpha)\rho_s c_s, \tag{A.9}$$

$$k_{mix} = \alpha k_l + (1 - \alpha)k_s. \tag{A.10}$$

S is the Darcy source term, and is defined as

$$\mathbf{S} = -C\frac{(1-\alpha)^2}{\alpha^3+\epsilon}\mathbf{v}, \tag{A.11}$$

where $C = 10^8$, $\epsilon = 10^{-8}$ in our simulation, and α is the liquid fraction defined by

$$\alpha = \begin{cases} 0 & \text{if } T < T_s \\ \frac{T-T_s}{T_l-T_s} & \text{if } T_s < T < T_l \\ 1 & \text{if } T > T_l \end{cases}, \tag{A.12}$$

where T_s and T_l are the solid and liquid temperatures of water, respectively. The other physical properties are constant and given in Table A.2. Note that the specific heat capacity of ice, c_s, is chosen to be the same as that of water, c_l, because of the same set up from the numerical benchmark problem [57].

A.2.3 Mesh Independence Study

A mesh independence study has been conducted using the three meshes with 100×100 cells, 150×150 cells and 225×225 cells, respectively. The simulations are run for 500 seconds when steady state has been reached. Figures A.11 - A.13 show the temperature, T, the velocity x-component, U_x, and the velocity y-component, U_y, along the center horizontal line $y = 19\ mm$ and the center vertical line $x = 19\ mm$,

Table A.2
Properties of water and ice used in the simulation

Material properties of water and ice	value	Unit
ρ_l denity of water	999.8	kg/m^3
ρ_s denity of ice	916.8	kg/m^3
μ dynamic viscosity	0.001003	kg/ms
k_l thermal conductivity of water	0.6	W/mK
k_s thermal conductivity of ice	2.26	W/mK
c_l specific heat capacity of water	4182.0	J/kgK
c_s specific heat capacity of ice	4182.0	J/kgK
L_f latent heat of fusion	335000	m^2/s^2
T_l liquid temperature of water	273.30	K
T_s solid temperature of water	273.00	K
g gravitational acceleration	9.81	m/s^2

respectively. From these figures, we can conclude that sufficient mesh independence has been achieved with the standard mesh. Therefore, all subsequent simulations have been performed with the standard mesh.

A.2.4 Validation of Numerical Results

Results from OpenFOAM® and FLUENT® [57] for the temperature and velocity fields are presented in Figs. A.14 and A.16.

During the first 100 s, the thickness of the ice layer is rather uniform, whereas after 100 s the main flow recirculation decreases the solidification rate in the upper part of the cavity and a characteristic belly-like shape of the ice front becomes evident.

More detailed quantitative comparisons of the temperature and velocity are given in Figs. A.17 - A.19. As is seen in these figures, there is good quantitative agreement between the OpenFOAM® and the FLUENT® results. Note that although the enhanced enthalpy-porosity model is used, it is used with the specific heat capacity of ice taken to be the same as the specific heat capacity of water, as used in [57]. The enhanced enthalpy-porosity model is validated under the same condition needed in the original enthalpy-porosity model.

An experimental benchmark of water freezing in a differentially heated cavity is considered. The cavity is a cube with side length of 38 mm. Two vertical black anodised walls are isothermal, kept at temperatures $T_H = 283\ K$ and $T_C = 263\ K$, respectively. The other four walls are made of 6 mm plexiglas, which have low thermal conductivity to ensure the entry of heat from the external laminar air stream at room temperature is neglected. The initial temperature of water and of all six walls is 273.5 K, i.e., just above the freezing point of water. The null initial velocity flow field is assumed.

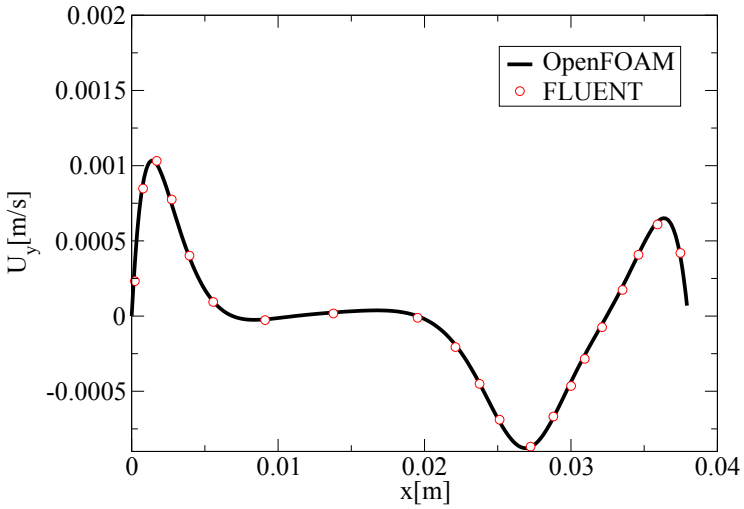
In our two dimensional simulation, the same governing equations are solved as in the previous case. For the thermal boundary conditions, without loss of much accuracy, idealized adiabatic boundary condition is assumed for the non-isothermal walls [45]. The same physical properties of water and ice are used as in the previous case except that the specific heat capacity of ice is the real one, i.e., $c_s = 2116.0\ J/kgK$.

We run the simulation to time $t = 2340\ s$. The ice front and velocity field from the

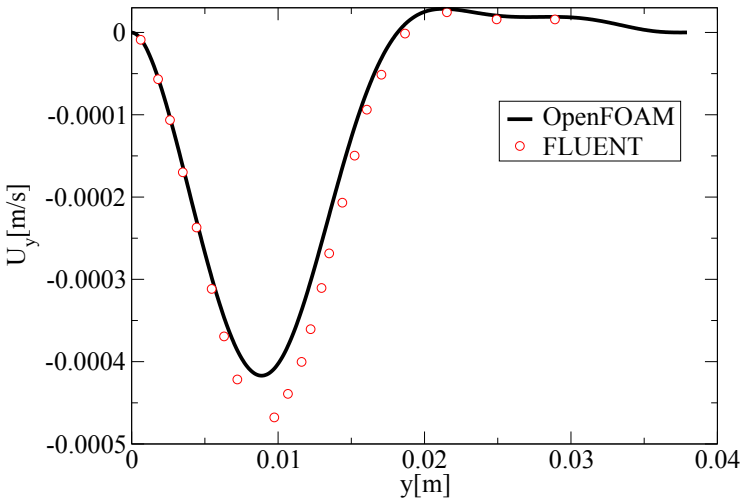
experiment and from OpenFOAM® are presented in Fig. A.20. It can be seen that the positions of the ice layers are almost identical, and both are almost perpendicular to the bottom of the cavity. Both of the flow fields have the same circulation flow pattern. Qualitatively, there is good agreement between the numerical results and the experiment.

A.3 Summary and Conclusions

In this appendix, an enhanced enthalpy-porosity model for phase change under natural convection is presented and implemented into OpenFOAM®. In this model, different thermophysical properties of liquid and solid phases are taken into consideration. This model has been tested for pure natural convection of water in a cavity and solidification of water in a cavity. The simulation results are in good agreement with both the numerical and the experimental results from the literature.

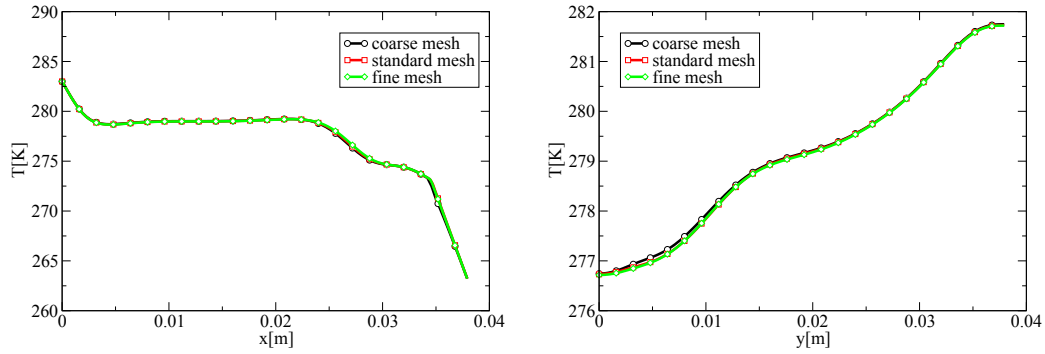


(a)



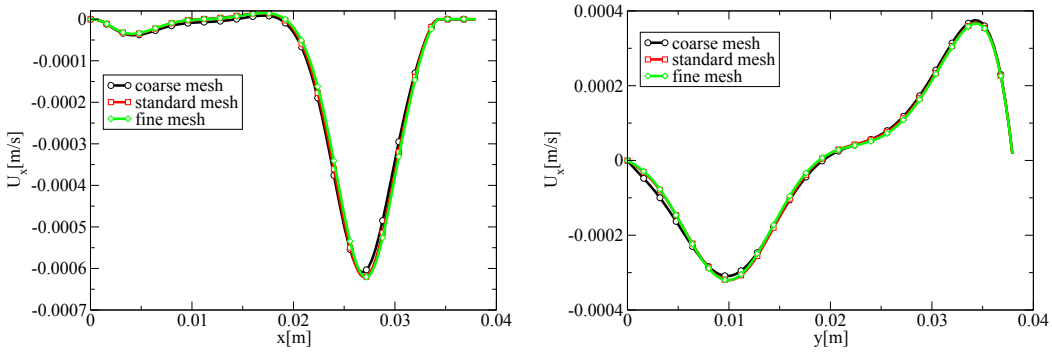
(b)

Figure A.10: y-component of velocity (a) along center horizontal line; (b) along center vertical line. Source of FLUENT® results: Michalek and Kowalewski [57].



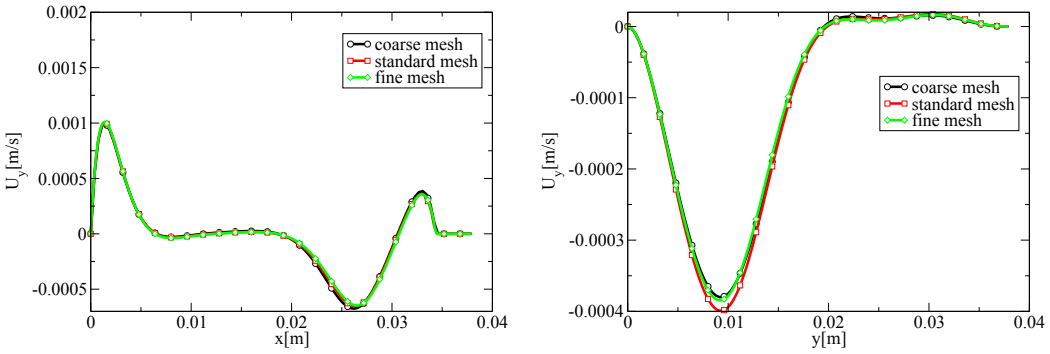
(a) (b)

Figure A.11: Temperature (a) along center horizontal line; (b) along center vertical line.



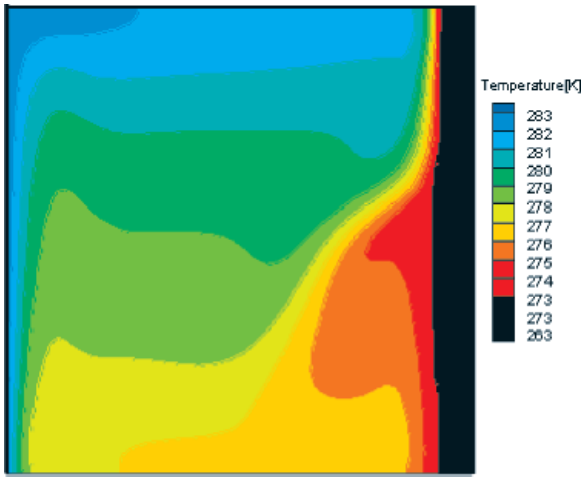
(a) (b)

Figure A.12: Velocity x-component (a) along center horizontal line; (b) along center vertical line.

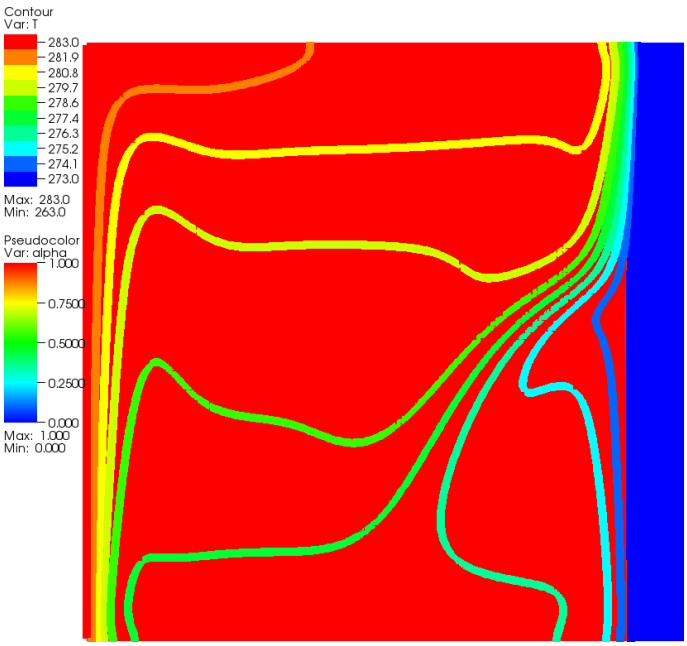


(a) (b)

Figure A.13: Velocity y-component (a) along center horizontal line; (b) along center vertical line.

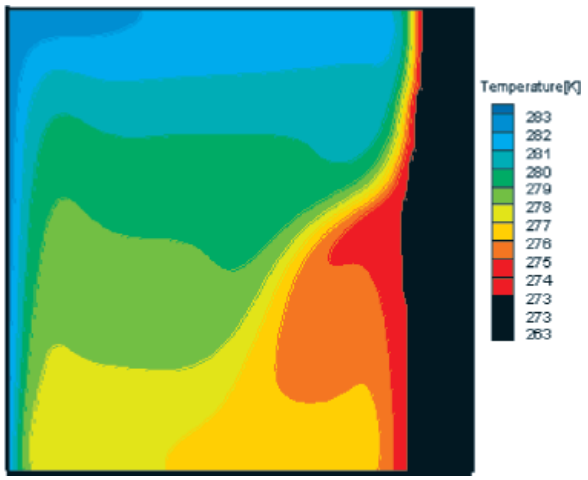


(a)

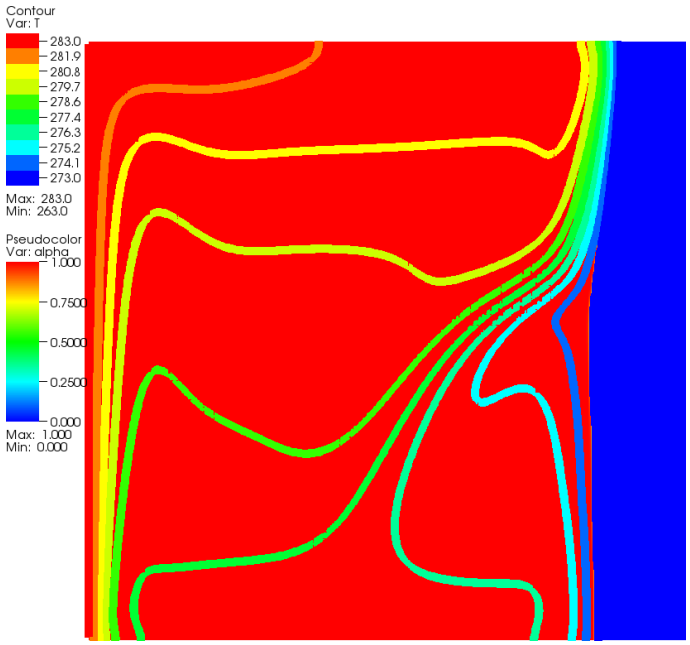


(b)

Figure A.14: Temperature contour at time $t = 100s$ (a) FLUENT®, source: [57]; reprinted from TASK Quarterly, 7(3), Michalek and Kowalewski, Simulations of the water freezing process numerical benchmarks, p. 400, Copyright(2003), with permission from CI TASK. See documentation in Appendix C. (b) OpenFOAM®.

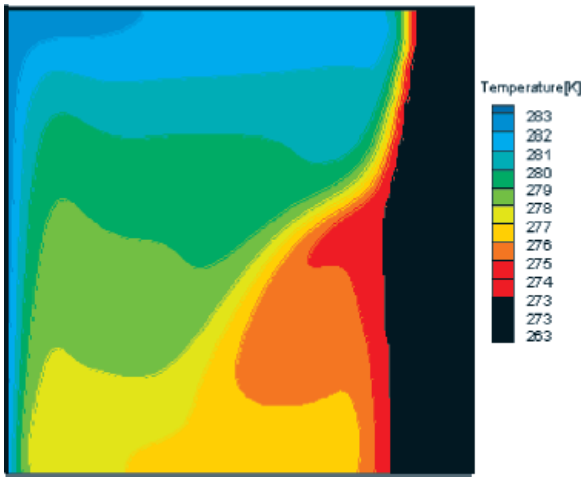


(a)

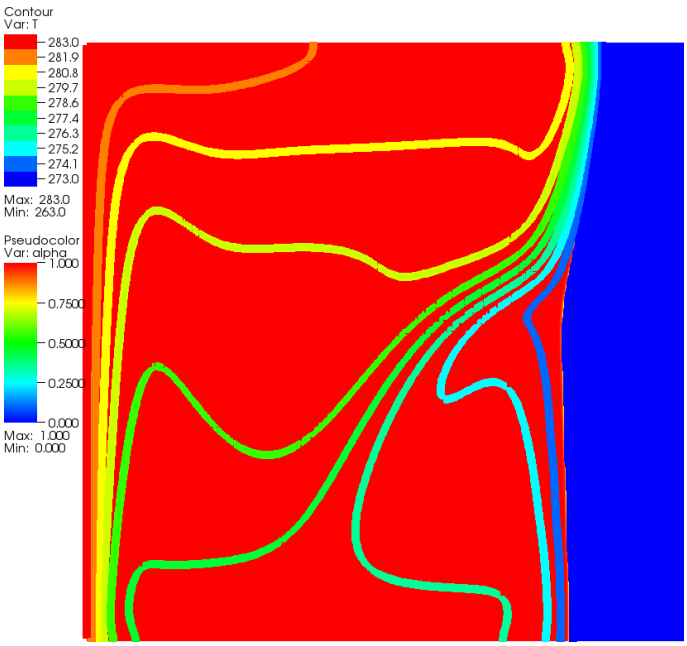


(b)

Figure A.15: Temperature contour at time $t = 300s$ (a) FLUENT®, source: [57]; reprinted from TASK Quarterly, 7(3), Michalek and Kowalewski, Simulations of the water freezing process numerical benchmarks, p. 400, Copyright(2003), with permission from CI TASK. See documentation in Appendix C. (b) OpenFOAM®.

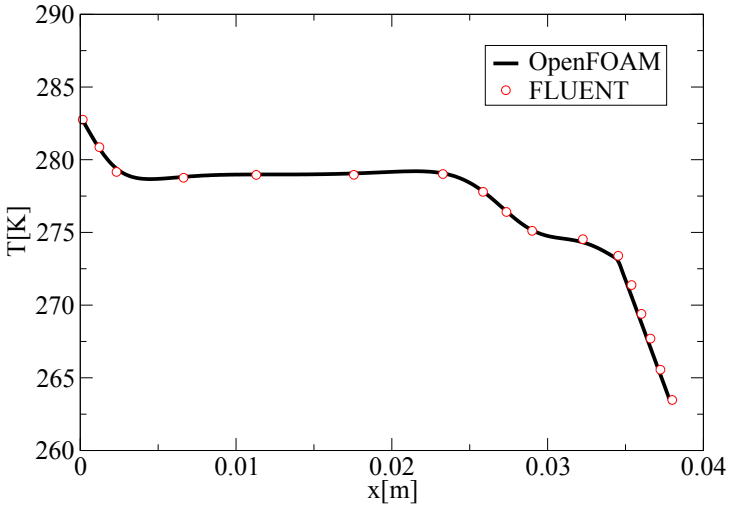


(a)

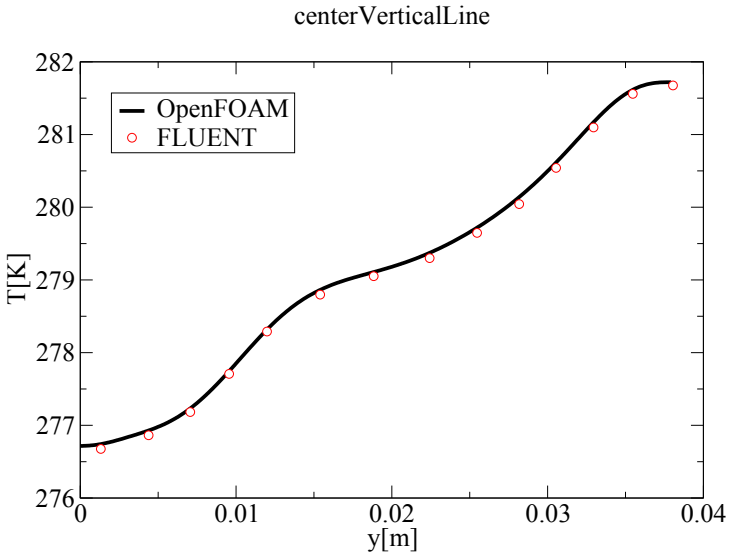


(b)

Figure A.16: Temperature contour at time $t = 500s$ (a) FLUENT®, source: [57]; reprinted from TASK Quarterly, 7(3), Michalek and Kowalewski, Simulations of the water freezing process numerical benchmarks, p. 400, Copyright(2003), with permission from CI TASK. See documentation in Appendix C. (b) OpenFOAM®.

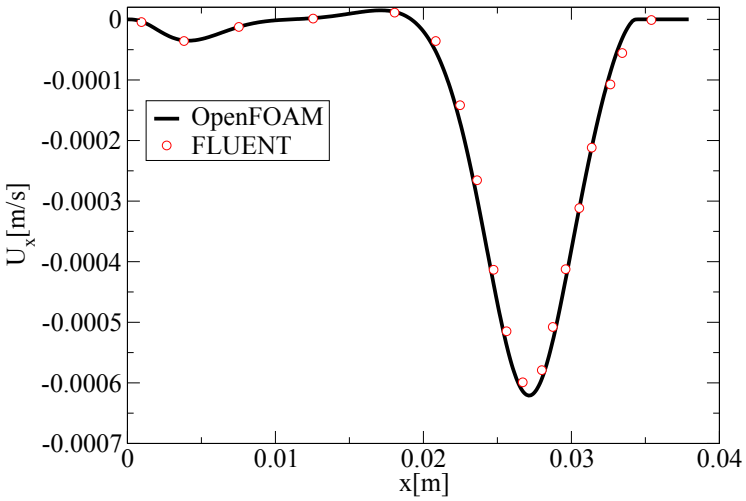


(a)

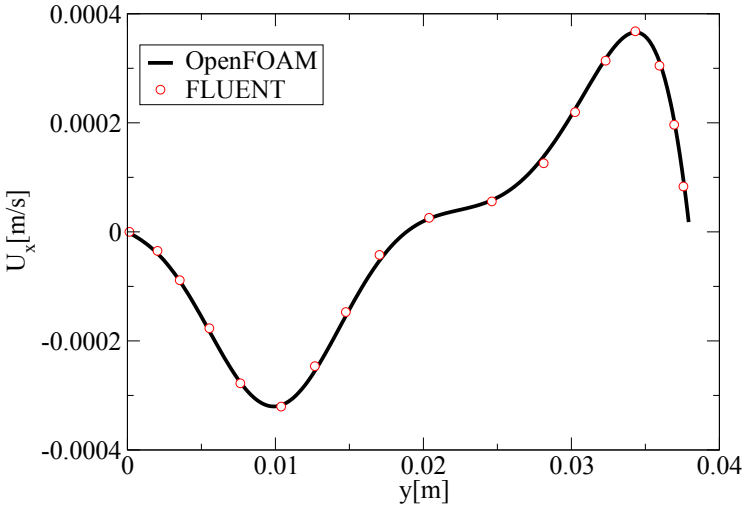


(b)

Figure A.17: Temperature (a) along center horizontal line; (b) along center vertical line. Source of FLUENT® results: Michalek and Kowalewski [57].

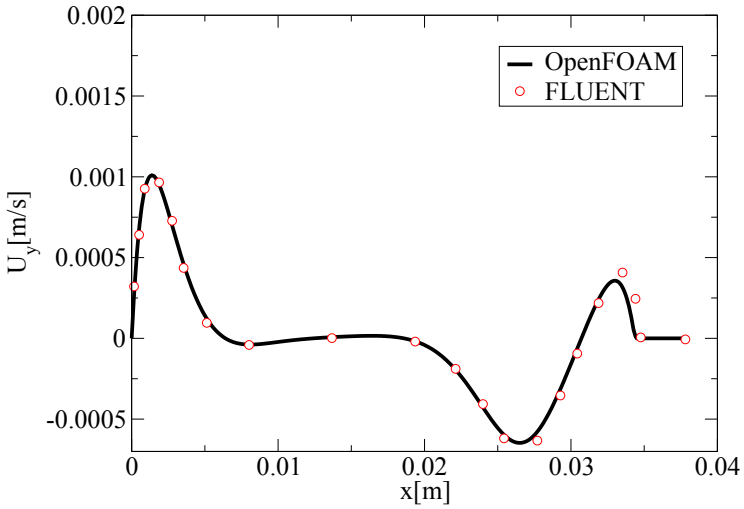


(a)

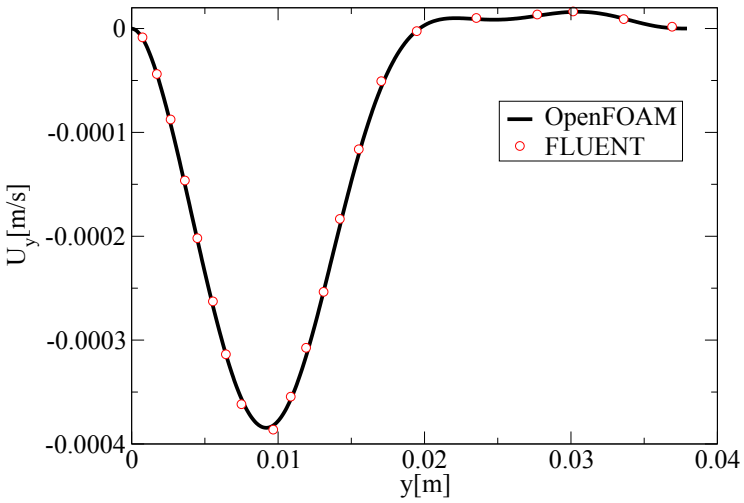


(b)

Figure A.18: Velocity x-component (a) along center horizontal line; (b) along center vertical line. Source of FLUENT® results: Michalek and Kowalewski [57].

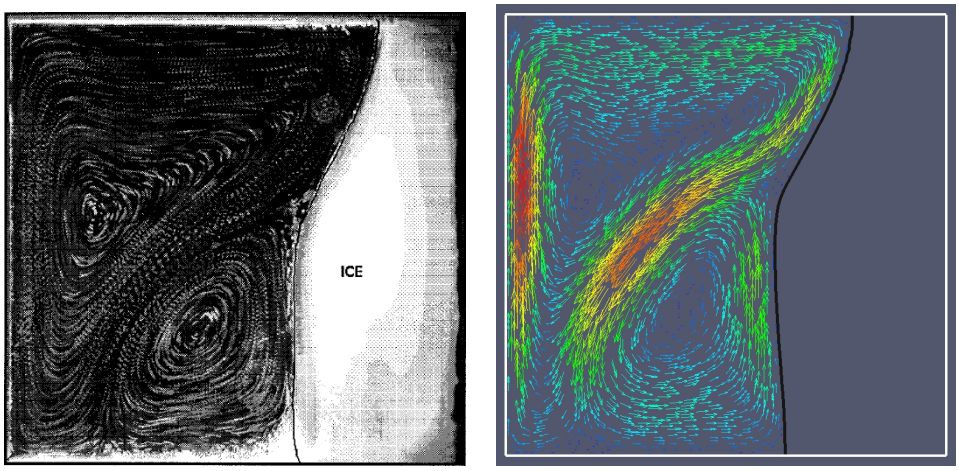


(a)



(b)

Figure A.19: y-component of velocity (a) along center horizontal line; (b) along center vertical line. Source of FLUENT® results: Michalek and Kowalewski [57].



(a) (b)

Figure A.20: Ice front and velocity field at $t = 2340\ s$. (a) experiment, source: Kowalewski and Rebow [44], republished with permission of Begell House Publishers, from An experimental benchmark for freezing water in the cubic cavity, T. A. Kowalewski and R. Marek, 1997; permission conveyed through Copyright Clearance Center, Inc. See documentation in Appendix D. (b) OpenFOAM®.

Appendix B

Letter for Figs. 3.19, 3.21, 3.23

ELSEVIER LICENSE
TERMS AND CONDITIONS

Apr 06, 2016

This is a License Agreement between Chao Liang ("You") and Elsevier ("Elsevier") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Elsevier, and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

Supplier	Elsevier Limited The Boulevard,Langford Lane Kidlington,Oxford,OX5 1GB,UK
Registered Company Number	1982084
Customer name	Chao Liang
Customer address	47831 US Highway 41 Apt 10 HOUGHTON, MI 49931
License number	3842671393197
License date	Apr 05, 2016
Licensed content publisher	Elsevier
Licensed content publication	International Journal of Multiphase Flow
Licensed content title	Temporal properties of secondary drop breakup in the multimode breakup regime
Licensed content author	Z. Dai,G.M. Faeth
Licensed content date	February 2001
Licensed content volume number	27
Licensed content issue number	2
Number of pages	20
Start Page	217
End Page	236
Type of Use	reuse in a thesis/dissertation
Portion	figures/tables/illustrations
Number of figures/tables /illustrations	3
Format	electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No
Original figure numbers	figures 1, 2, 7
Title of your thesis/dissertation	Development of Computational Methods for the Investigation of Liquid Drop Phenomena in External Flows

Expected completion date	May 2016
Estimated size (number of pages)	223
Elsevier VAT number	GB 494 6272 12
Permissions price	0.00 USD
VAT/Local Sales Tax	0.00 USD / 0.00 GBP
Total	0.00 USD
Terms and Conditions	

INTRODUCTION

1. The publisher for this copyrighted material is Elsevier. By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction (along with the Billing and Payment terms and conditions established by Copyright Clearance Center, Inc. ("CCC"), at the time that you opened your Rightslink account and that are available at any time at http://myaccount.copyright.com).

GENERAL TERMS

2. Elsevier hereby grants you permission to reproduce the aforementioned material subject to the terms and conditions indicated.
3. Acknowledgement: If any part of the material to be used (for example, figures) has appeared in our publication with credit or acknowledgement to another source, permission must also be sought from that source. If such permission is not obtained then that material may not be included in your publication/copies. Suitable acknowledgement to the source must be made, either as a footnote or in a reference list at the end of your publication, as follows:
"Reprinted from Publication title, Vol /edition number, Author(s), Title of article / title of chapter, Pages No., Copyright (Year), with permission from Elsevier [OR APPLICABLE SOCIETY COPYRIGHT OWNER]." Also Lancet special credit - "Reprinted from The Lancet, Vol. number, Author(s), Title of article, Pages No., Copyright (Year), with permission from Elsevier."
4. Reproduction of this material is confined to the purpose and/or media for which permission is hereby given.
5. Altering/Modifying Material: Not Permitted. However figures and illustrations may be altered/adapted minimally to serve your work. Any other abbreviations, additions, deletions and/or any other alterations shall be made only with prior written authorization of Elsevier Ltd. (Please contact Elsevier at permissions@elsevier.com)
6. If the permission fee for the requested use of our material is waived in this instance, please be advised that your future requests for Elsevier materials may attract a fee.
7. Reservation of Rights: Publisher reserves all rights not specifically granted in the combination of (i) the license details provided by you and accepted in the course of this licensing transaction, (ii) these terms and conditions and (iii) CCC's Billing and Payment terms and conditions.

8. License Contingent Upon Payment: While you may exercise the rights licensed immediately upon issuance of the license at the end of the licensing process for the transaction, provided that you have disclosed complete and accurate details of your proposed use, no license is finally effective unless and until full payment is received from you (either by publisher or by CCC) as provided in CCC's Billing and Payment terms and conditions. If full payment is not received on a timely basis, then any license preliminarily granted shall be deemed automatically revoked and shall be void as if never granted. Further, in the event that you breach any of these terms and conditions or any of CCC's Billing and Payment terms and conditions, the license is automatically revoked and shall be void as if never granted. Use of materials as described in a revoked license, as well as any use of the materials beyond the scope of an unrevoked license, may constitute copyright infringement and publisher reserves the right to take any and all action to protect its copyright in the materials.
9. Warranties: Publisher makes no representations or warranties with respect to the licensed material.
10. Indemnity: You hereby indemnify and agree to hold harmless publisher and CCC, and their respective officers, directors, employees and agents, from and against any and all claims arising out of your use of the licensed material other than as specifically authorized pursuant to this license.
11. No Transfer of License: This license is personal to you and may not be sublicensed, assigned, or transferred by you to any other person without publisher's written permission.
12. No Amendment Except in Writing: This license may not be amended except in a writing signed by both parties (or, in the case of publisher, by CCC on publisher's behalf).
13. Objection to Contrary Terms: Publisher hereby objects to any terms contained in any purchase order, acknowledgment, check endorsement or other writing prepared by you, which terms are inconsistent with these terms and conditions or CCC's Billing and Payment terms and conditions. These terms and conditions, together with CCC's Billing and Payment terms and conditions (which are incorporated herein), comprise the entire agreement between you and publisher (and CCC) concerning this licensing transaction. In the event of any conflict between your obligations established by these terms and conditions and those established by CCC's Billing and Payment terms and conditions, these terms and conditions shall control.
14. Revocation: Elsevier or Copyright Clearance Center may deny the permissions described in this License at their sole discretion, for any reason or no reason, with a full refund payable to you. Notice of such denial will be made using the contact information provided by you. Failure to receive such notice will not alter or invalidate the denial. In no event will Elsevier or Copyright Clearance Center be responsible or liable for any costs, expenses or damage incurred by you as a result of a

denial of your permission request, other than a refund of the amount(s) paid by you to Elsevier and/or Copyright Clearance Center for denied permissions.

LIMITED LICENSE

The following terms and conditions apply only to specific license types:
15. **Translation**: This permission is granted for non-exclusive world **English** rights only unless your license was granted for translation rights. If you licensed translation rights you may only translate this content into the languages you requested. A professional translator must perform all translations and reproduce the content word for word preserving the integrity of the article.
16. **Posting licensed content on any Website**: The following terms and conditions apply as follows: Licensing material from an Elsevier journal: All content posted to the web site must maintain the copyright information line on the bottom of each image; A hyper-text must be included to the Homepage of the journal from which you are licensing at http://www.sciencedirect.com/science/journal/xxxxx or the Elsevier homepage for books at http://www.elsevier.com; Central Storage: This license does not include permission for a scanned version of the material to be stored in a central repository such as that provided by Heron/XanEdu.
Licensing material from an Elsevier book: A hyper-text link must be included to the Elsevier homepage at http://www.elsevier.com . All content posted to the web site must maintain the copyright information line on the bottom of each image.

Posting licensed content on Electronic reserve: In addition to the above the following clauses are applicable: The web site must be password-protected and made available only to bona fide students registered on a relevant course. This permission is granted for 1 year only. You may obtain a new license for future website posting.
17. **For journal authors:** the following clauses are applicable in addition to the above:
Preprints:
A preprint is an author's own write-up of research results and analysis, it has not been peer-reviewed, nor has it had any other value added to it by a publisher (such as formatting, copyright, technical enhancement etc.).
Authors can share their preprints anywhere at any time. Preprints should not be added to or enhanced in any way in order to appear more like, or to substitute for, the final versions of articles however authors can update their preprints on arXiv or RePEc with their Accepted Author Manuscript (see below).
If accepted for publication, we encourage authors to link from the preprint to their formal publication via its DOI. Millions of researchers have access to the formal publications on ScienceDirect, and so links will help users to find, access, cite and use the best available version.

Please note that Cell Press, The Lancet and some society-owned have different preprint policies. Information on these policies is available on the journal homepage.
Accepted Author Manuscripts: An accepted author manuscript is the manuscript of an article that has been accepted for publication and which typically includes author-incorporated changes suggested during submission, peer review and editor-author communications.
Authors can share their accepted author manuscript:

- immediately
  - via their non-commercial person homepage or blog
  - by updating a preprint in arXiv or RePEc with the accepted manuscript
  - via their research institute or institutional repository for internal institutional uses or as part of an invitation-only research collaboration work-group
  - directly by providing copies to their students or to research collaborators for their personal use
  - for private scholarly sharing as part of an invitation-only work group on commercial sites with which Elsevier has an agreement
- after the embargo period
  - via non-commercial hosting platforms such as their institutional repository
  - via commercial sites with which Elsevier has an agreement

In all cases accepted manuscripts should:

- link to the formal publication via its DOI
- bear a CC-BY-NC-ND license - this is easy to do
- if aggregated with other manuscripts, for example in a repository or other site, be shared in alignment with our hosting policy not be added to or enhanced in any way to appear more like, or to substitute for, the published journal article.

Published journal article (JPA): A published journal article (PJA) is the definitive final record of published research that appears or will appear in the journal and embodies all value-adding publishing activities including peer review co-ordination, copy-editing, formatting, (if relevant) pagination and online enrichment.
Policies for sharing publishing journal articles differ for subscription and gold open access articles:
Subscription Articles: If you are an author, please share a link to your article rather than the full-text. Millions of researchers have access to the formal publications on ScienceDirect, and so links will help your users to find, access, cite, and use the best available version.
Theses and dissertations which contain embedded PJAs as part of the

formal submission can be posted publicly by the awarding institution with DOI links back to the formal publications on ScienceDirect.
If you are affiliated with a library that subscribes to ScienceDirect you have additional private sharing rights for others' research accessed under that agreement. This includes use for classroom teaching and internal training at the institution (including use in course packs and courseware programs), and inclusion of the article for grant funding purposes.
Gold Open Access Articles: May be shared according to the author-selected end-user license and should contain a CrossMark logo, the end user license, and a DOI link to the formal publication on ScienceDirect.
Please refer to Elsevier's posting policy for further information.
18. **For book authors** the following clauses are applicable in addition to the above: Authors are permitted to place a brief summary of their work online only. You are not allowed to download and post the published electronic version of your chapter, nor may you scan the printed edition to create an electronic version. **Posting to a repository:** Authors are permitted to post a summary of their chapter only in their institution's repository.
19. **Thesis/Dissertation**: If your license is for use in a thesis/dissertation your thesis may be submitted to your institution in either print or electronic form. Should your thesis be published commercially, please reapply for permission. These requirements include permission for the Library and Archives of Canada to supply single copies, on demand, of the complete thesis and include permission for Proquest/UMI to supply single copies, on demand, of the complete thesis. Should your thesis be published commercially, please reapply for permission. Theses and dissertations which contain embedded PJAs as part of the formal submission can be posted publicly by the awarding institution with DOI links back to the formal publications on ScienceDirect.

Elsevier Open Access Terms and Conditions
You can publish open access with Elsevier in hundreds of open access journals or in nearly 2000 established subscription journals that support open access publishing. Permitted third party re-use of these open access articles is defined by the author's choice of Creative Commons user license. See our open access license policy for more information.
Terms & Conditions applicable to all Open Access articles published with Elsevier:
Any reuse of the article must not represent the author as endorsing the adaptation of the article nor should the article be modified in such a way as to damage the author's honour or reputation. If any changes have been made, such changes must be clearly indicated.
The author(s) must be appropriately credited and we ask that you include the end user license and a DOI link to the formal publication on ScienceDirect.

If any part of the material to be used (for example, figures) has appeared in our publication with credit or acknowledgement to another source it is the responsibility of the user to ensure their reuse complies with the terms and conditions determined by the rights holder.
Additional Terms & Conditions applicable to each Creative Commons user license:
CC BY: The CC-BY license allows users to copy, to create extracts, abstracts and new works from the Article, to alter and revise the Article and to make commercial use of the Article (including reuse and/or resale of the Article by commercial entities), provided the user gives appropriate credit (with a link to the formal publication through the relevant DOI), provides a link to the license, indicates if changes were made and the licensor is not represented as endorsing the use made of the work. The full details of the license are available at http://creativecommons.org/licenses/by/4.0.
CC BY NC SA: The CC BY-NC-SA license allows users to copy, to create extracts, abstracts and new works from the Article, to alter and revise the Article, provided this is not done for commercial purposes, and that the user gives appropriate credit (with a link to the formal publication through the relevant DOI), provides a link to the license, indicates if changes were made and the licensor is not represented as endorsing the use made of the work. Further, any new works must be made available on the same conditions. The full details of the license are available at http://creativecommons.org/licenses/by-nc-sa/4.0.
CC BY NC ND: The CC BY-NC-ND license allows users to copy and distribute the Article, provided this is not done for commercial purposes and further does not permit distribution of the Article if it is changed or edited in any way, and provided the user gives appropriate credit (with a link to the formal publication through the relevant DOI), provides a link to the license, and that the licensor is not represented as endorsing the use made of the work. The full details of the license are available at http://creativecommons.org/licenses/by-nc-nd/4.0. Any commercial reuse of Open Access articles published with a CC BY NC SA or CC BY NC ND license requires permission from Elsevier and will be subject to a fee. Commercial reuse includes:

- Associating advertising with the full text of the Article
- Charging fees for document delivery or access
- Article aggregation
- Systematic distribution via e-mail lists or share buttons

Posting or linking by commercial companies for use by customers of those companies.

20. **Other Conditions**:

v1.8

Appendix C

Letter for Figs. A.5(a), A.6(a), A.7(a), A.14(a), A.15(a), A.16(a)

Michigan Tech

Chao Liang <chaolian@mtu.edu>

Request Permission to Reuse Figures

"Jarosław Rybicki" <ryba@pg.gda.pl> Thu, Apr 7, 2016 at 4:22 AM
Reply-To: ryba@pg.gda.pl
To: Chao Liang <chaolian@mtu.edu>

Dear Chao,

In reference to email of April 7, 2016 we hereby grant you permission to use, at no charge, Figure 2(a), 2(b), 2(d) and Figure 6(a), 6(c), 6(e) from the article, Michalek, T. and Kowalewski, T. A., Simulations of the water freezing process – numerical benchmarks, Task Quarterly, vol. 7(3), pp. 389–408, 2003 in your PhD Thesis entitled Computational Methods for the Investigation of Liquid Drop Phenomena in External Gas Flows.

Best regards,

Prof. J. Rybicki

TASK Quarterly
[Quoted text hidden]

Appendix D

Letter for Fig. A.20(a)

Begell House LICENSE
TERMS AND CONDITIONS

Apr 06, 2016

This is a License Agreement between Chao Liang ("You") and Begell House ("Begell House") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Begell House, and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number	3842860812560
License date	Apr 06, 2016
Licensed content publisher	Begell House
Licensed content title	ICHMT DIGITAL LIBRARY
Licensed content date	Dec 31, 1969
Type of Use	Thesis/Dissertation
Requestor type	Author of requested content
Format	Electronic
Portion	image/photo
Number of images/photos requested	1
Title or numeric reference of the portion(s)	Figure 1 left portion
Title of the article or chapter the portion is from	An Experimental Benchmark for Freezing Water in the Cubic Cavity
Editor of portion(s)	N/A
Author of portion(s)	Tomasz A. Kowalewski and Marek Rebow
Volume of serial or monograph.	N/A
Issue, if republishing an article from a serial	N/A
Page range of the portion	N/A
Publication date of portion	1997
Rights for	Main product
Duration of use	Life of current edition
Creation of copies for the disabled	no
With minor editing privileges	no
For distribution to	Worldwide
In the following language(s)	Original language of publication
With incidental promotional use	no

The lifetime unit quantity of new product	Up to 499
Made available in the following markets	professional
The requesting person/organization is:	Chao Liang
Order reference number	None
Author/Editor	Chao Liang
The standard identifier	Thesis
Title	Development of Computational Methods for the Investigation of Liquid Drop Phenomena in External Flows
Publisher	Michigan Technological University
Expected publication date	May 2016
Estimated size (pages)	223
Total (may include CCC user fee)	0.00 USD
Terms and Conditions	

TERMS AND CONDITIONS

The following terms are individual to this publisher:

User is responsible for identifying and seeking separate licenses (under this service or otherwise) for, any of such third party materials which are identified anywhere in the works by permission without a separate license, such third party materials may not be used.

Other Terms and Conditions:

STANDARD TERMS AND CONDITIONS

1. Description of Service; Defined Terms. This Republication License enables the User to obtain licenses for republication of one or more copyrighted works as described in detail on the relevant Order Confirmation (the “Work(s)”). Copyright Clearance Center, Inc. (“CCC”) grants licenses through the Service on behalf of the rightsholder identified on the Order Confirmation (the “Rightsholder”). “Republication”, as used herein, generally means the inclusion of a Work, in whole or in part, in a new work or works, also as described on the Order Confirmation. “User”, as used herein, means the person or entity making such republication.

2. The terms set forth in the relevant Order Confirmation, and any terms set by the Rightsholder with respect to a particular Work, govern the terms of use of Works in connection with the Service. By using the Service, the person transacting for a republication license on behalf of the User represents and warrants that he/she/it (a) has been duly authorized by the User to accept, and hereby does accept, all such terms and conditions on behalf of User, and (b) shall inform User of all such terms and conditions. In the event such person is a “freelancer” or other third party independent of User and CCC, such party shall be deemed jointly a “User” for purposes of these terms and conditions. In any event, User shall be deemed to have accepted and agreed to all such terms and

conditions if User republishes the Work in any fashion.

3. Scope of License; Limitations and Obligations.

3.1 All Works and all rights therein, including copyright rights, remain the sole and exclusive property of the Rightsholder. The license created by the exchange of an Order Confirmation (and/or any invoice) and payment by User of the full amount set forth on that document includes only those rights expressly set forth in the Order Confirmation and in these terms and conditions, and conveys no other rights in the Work(s) to User. All rights not expressly granted are hereby reserved.

3.2 General Payment Terms: You may pay by credit card or through an account with us payable at the end of the month. If you and we agree that you may establish a standing account with CCC, then the following terms apply: Remit Payment to: Copyright Clearance Center, Dept 001, P.O. Box 843006, Boston, MA 02284-3006. Payments Due: Invoices are payable upon their delivery to you (or upon our notice to you that they are available to you for downloading). After 30 days, outstanding amounts will be subject to a service charge of 1-1/2% per month or, if less, the maximum rate allowed by applicable law. Unless otherwise specifically set forth in the Order Confirmation or in a separate written agreement signed by CCC, invoices are due and payable on “net 30” terms. While User may exercise the rights licensed immediately upon issuance of the Order Confirmation, the license is automatically revoked and is null and void, as if it had never been issued, if complete payment for the license is not received on a timely basis either from User directly or through a payment agent, such as a credit card company.

3.3 Unless otherwise provided in the Order Confirmation, any grant of rights to User (i) is “one-time” (including the editions and product family specified in the license), (ii) is non-exclusive and non-transferable and (iii) is subject to any and all limitations and restrictions (such as, but not limited to, limitations on duration of use or circulation) included in the Order Confirmation or invoice and/or in these terms and conditions. Upon completion of the licensed use, User shall either secure a new permission for further use of the Work(s) or immediately cease any new use of the Work(s) and shall render inaccessible (such as by deleting or by removing or severing links or other locators) any further copies of the Work (except for copies printed on paper in accordance with this license and still in User's stock at the end of such period).

3.4 In the event that the material for which a republication license is sought includes third party materials (such as photographs, illustrations, graphs, inserts and similar materials) which are identified in such material as having been used by permission, User is responsible for identifying, and seeking separate licenses (under this Service or otherwise) for, any of such third party materials; without a separate license, such third party materials may not be used.

3.5 Use of proper copyright notice for a Work is required as a condition of any license granted under the Service. Unless otherwise provided in the Order Confirmation, a proper copyright notice will read substantially

as follows: “Republished with permission of [Rightsholder’s name], from [Work's title, author, volume, edition number and year of copyright]; permission conveyed through Copyright Clearance Center, Inc. ” Such notice must be provided in a reasonably legible font size and must be placed either immediately adjacent to the Work as used (for example, as part of a by-line or footnote but not as a separate electronic link) or in the place where substantially all other credits or notices for the new work containing the republished Work are located. Failure to include the required notice results in loss to the Rightsholder and CCC, and the User shall be liable to pay liquidated damages for each such failure equal to twice the use fee specified in the Order Confirmation, in addition to the use fee itself and any other fees and charges specified.
3.6 User may only make alterations to the Work if and as expressly set forth in the Order Confirmation. No Work may be used in any way that is defamatory, violates the rights of third parties (including such third parties' rights of copyright, privacy, publicity, or other tangible or intangible property), or is otherwise illegal, sexually explicit or obscene. In addition, User may not conjoin a Work with any other material that may result in damage to the reputation of the Rightsholder. User agrees to inform CCC if it becomes aware of any infringement of any rights in a Work and to cooperate with any reasonable request of CCC or the Rightsholder in connection therewith.
4. Indemnity. User hereby indemnifies and agrees to defend the Rightsholder and CCC, and their respective employees and directors, against all claims, liability, damages, costs and expenses, including legal fees and expenses, arising out of any use of a Work beyond the scope of the rights granted herein, or any use of a Work which has been altered in any unauthorized way by User, including claims of defamation or infringement of rights of copyright, publicity, privacy or other tangible or intangible property.
5. Limitation of Liability. UNDER NO CIRCUMSTANCES WILL CCC OR THE RIGHTSHOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES (INCLUDING WITHOUT LIMITATION DAMAGES FOR LOSS OF BUSINESS PROFITS OR INFORMATION, OR FOR BUSINESS INTERRUPTION) ARISING OUT OF THE USE OR INABILITY TO USE A WORK, EVEN IF ONE OF THEM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In any event, the total liability of the Rightsholder and CCC (including their respective employees and directors) shall not exceed the total amount actually paid by User for this license. User assumes full liability for the actions and omissions of its principals, employees, agents, affiliates, successors and assigns.
6. Limited Warranties. THE WORK(S) AND RIGHT(S) ARE PROVIDED “AS IS”. CCC HAS THE RIGHT TO GRANT TO USER THE RIGHTS GRANTED IN THE ORDER CONFIRMATION DOCUMENT. CCC AND THE RIGHTSHOLDER DISCLAIM ALL OTHER WARRANTIES RELATING TO THE WORK(S) AND RIGHT(S), EITHER EXPRESS OR IMPLIED,

INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. ADDITIONAL RIGHTS MAY BE REQUIRED TO USE ILLUSTRATIONS, GRAPHS, PHOTOGRAPHS, ABSTRACTS, INSERTS OR OTHER PORTIONS OF THE WORK (AS OPPOSED TO THE ENTIRE WORK) IN A MANNER CONTEMPLATED BY USER; USER UNDERSTANDS AND AGREES THAT NEITHER CCC NOR THE RIGHTSHOLDER MAY HAVE SUCH ADDITIONAL RIGHTS TO GRANT.

7. Effect of Breach. Any failure by User to pay any amount when due, or any use by User of a Work beyond the scope of the license set forth in the Order Confirmation and/or these terms and conditions, shall be a material breach of the license created by the Order Confirmation and these terms and conditions. Any breach not cured within 30 days of written notice thereof shall result in immediate termination of such license without further notice. Any unauthorized (but licensable) use of a Work that is terminated immediately upon notice thereof may be liquidated by payment of the Rightsholder's ordinary license price therefor; any unauthorized (and unlicensable) use that is not terminated immediately for any reason (including, for example, because materials containing the Work cannot reasonably be recalled) will be subject to all remedies available at law or in equity, but in no event to a payment of less than three times the Rightsholder's ordinary license price for the most closely analogous licensable use plus Rightsholder's and/or CCC's costs and expenses incurred in collecting such payment.

8. **Miscellaneous.**

8.1 User acknowledges that CCC may, from time to time, make changes or additions to the Service or to these terms and conditions, and CCC reserves the right to send notice to the User by electronic mail or otherwise for the purposes of notifying User of such changes or additions; provided that any such changes or additions shall not apply to permissions already secured and paid for.

8.2 Use of User-related information collected through the Service is governed by CCC’s privacy policy, available online here: http://www.copyright.com/content/cc3/en/tools/footer/privacypolicy.html.

8.3 The licensing transaction described in the Order Confirmation is personal to User. Therefore, User may not assign or transfer to any other person (whether a natural person or an organization of any kind) the license created by the Order Confirmation and these terms and conditions or any rights granted hereunder; provided, however, that User may assign such license in its entirety on written notice to CCC in the event of a transfer of all or substantially all of User’s rights in the new material which includes the Work(s) licensed under this Service.

8.4 No amendment or waiver of any terms is binding unless set forth in writing and signed by the parties. The Rightsholder and CCC hereby object to any terms contained in any writing prepared by the User or its principals, employees, agents or affiliates and purporting to govern or otherwise relate to the licensing transaction described in the Order

Confirmation, which terms are in any way inconsistent with any terms set forth in the Order Confirmation and/or in these terms and conditions or CCC's standard operating procedures, whether such writing is prepared prior to, simultaneously with or subsequent to the Order Confirmation, and whether such writing appears on a copy of the Order Confirmation or in a separate instrument.
8.5 The licensing transaction described in the Order Confirmation document shall be governed by and construed under the law of the State of New York, USA, without regard to the principles thereof of conflicts of law. Any case, controversy, suit, action, or proceeding arising out of, in connection with, or related to such licensing transaction shall be brought, at CCC's sole discretion, in any federal or state court located in the County of New York, State of New York, USA, or in any federal or state court whose geographical jurisdiction covers the location of the Rightsholder set forth in the Order Confirmation. The parties expressly submit to the personal jurisdiction and venue of each such federal or state court.If you have any comments or questions about the Service or Copyright Clearance Center, please contact us at 978-750-8400 or send an e-mail to info@copyright.com.
v 1.1

Appendix E

interSEAFoam Code

interSEAFoam.C

```
#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 "dropFromAlpha.H" //chao

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;


int time_count = 0; //chao

Vector<double> centroid_old, centroid, ↩
   totalCentroid_; //chao
double nCells_x_real, nCells_y_real, nCells_z_real; ↩
   //chao
int nCells_x, nCells_y, nCells_z; //chao

#include "calcCOM.H" //chao
centroid_old = totalCentroid_; //chao

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

    runTime++;
    time_count ++; //chao

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

    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();
        }
    }

    if (time_count == every_time_steps.value()) //↩
       chao
    {
        #include "dragBackIfNeeded.H" //chao
        time_count = 0;
    }
```

```
        if (runTime.outputTime()) //chao
        {
            #include "dragBackIfNeeded.H" //chao
      time_count = 0;
        }

        runTime.write();

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

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

    return 0;
}
```

calcCOM.H

```
{
    //calculate the center of mass of liquid phase in ↩
       the entire computational domain
    scalar totalVolume_ = 0.0;
    totalCentroid_ = Vector<scalar>::zero;
    scalar cellVolume = 0.0;

    forAll(mesh.C(), cellID)
    {
        cellVolume = mesh.cellVolumes()[cellID] * alpha1↩
           [cellID];
        totalVolume_     += cellVolume;
        totalCentroid_   += mesh.cellCentres()[cellID] *↩
            cellVolume;
    }
    //parallel version start
    if (Pstream::parRun())
    {
        reduce(totalVolume_, sumOp<scalar>());
        reduce(totalCentroid_, sumOp<vector>());
```

```
    }
    //parallel version end
    totalCentroid_ /= totalVolume_;
}
```

dragBackIfNeeded.H

```
#include "calcCOM.H"
centroid = totalCentroid_;

Info << "Now alpha centroid: " << centroid.x() << " " <<↩
    centroid.y() << " " << centroid.z() << endl;
Info << "centroid_old: " << centroid_old.x() << " " << ↩
   centroid_old.y() << " " << centroid_old.z() << endl;

nCells_x_real = (centroid.x() - centroid_old.x())/dx.↩
   value();
nCells_y_real = (centroid.y() - centroid_old.y())/dy.↩
   value();
nCells_z_real = (centroid.z() - centroid_old.z())/dz.↩
   value();
Info << "move along x dir by " << nCells_x_real << "↩
   cells" << endl;
Info << "move along y dir by " << nCells_y_real << "↩
   cells" << endl;
Info << "move along z dir by " << nCells_z_real << "↩
   cells" << endl;

nCells_x = (int)nCells_x_real;
nCells_y = (int)nCells_y_real;
nCells_z = (int)nCells_z_real;

bool drag = false;

if (xMove.value() == 1)
{
if (nCells_x > 0)
```

```
{
  drag = true;

  Info << "Drag back left " << nCells_x << " cells." << ↩
     endl;

for (int i = 0; i < nCells_x; i++)
{

    if (Pstream::parRun() && nx > 1)
    {
       #include "dragLeftComm.H"
    }
    else
   {

    for (int cellI = 0; cellI < mesh.cells().size(); ↩
       cellI++)
    {
        labelList neighbors = mesh.cellCells()[cellI];
  for (int itr=0; itr<neighbors.size(); itr++)
        {
      if (mesh.C()[neighbors[itr]].x() - mesh.C()[cellI↩
         ].x() > 1e-10)
            {
    alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
       1 cell
    p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
       cell
    U[cellI] = U[neighbors[itr]]; //drag back 1 cell
            }
        }
    }

   }
}
}

else if (nCells_x < 0)
{
   drag = true;
```

```
    Info << "Drag back right " << nCells_x << " cells." ↩
       << endl;

     for (int i = 0; i < (-1)*nCells_x; i++)
    {

     if (Pstream::parRun() && nx > 1)
     {
        #include "dragRightComm.H"
     }
     else
    {

     for (int cellI = mesh.cells().size()-1; cellI >=0; ↩
        cellI--)
     {
         labelList neighbors = mesh.cellCells()[cellI];
  for (int itr=0; itr<neighbors.size(); itr++)
         {
        if (mesh.C()[cellI].x() - mesh.C()[neighbors[itr↩
           ]].x() > 1e-10)
               {
     alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
        1 cell
     p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
        cell
     U[cellI] = U[neighbors[itr]]; //drag back 1 cell
               }
         }
     }
    }
    }
}
}

if (yMove.value() == 1)
{
if (nCells_y > 0)
{
     drag = true;
     Info << "Drag back down " << nCells_y << " cells." ↩
        << endl;
```

```
    for (int i = 0; i < nCells_y; i++)
    {

     if (Pstream::parRun() && ny > 1)
     {
        #include "dragDownComm.H"
     }
     else
    {

     for (int cellI = 0; cellI < mesh.cells().size(); ↩
        cellI++)
     {
         labelList neighbors = mesh.cellCells()[cellI];
   for (int itr=0; itr<neighbors.size(); itr++)
         {
       if (mesh.C()[neighbors[itr]].y() - mesh.C()[cellI↩
          ].y() > 1e-10)
              {
     alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
        1 cell
     p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
        cell
     U[cellI] = U[neighbors[itr]]; //drag back 1 cell
              }
         }
     }
    }
    }
}

else if (nCells_y < 0)
{
     drag = true;
     Info << "Drag back up " << nCells_y << " cells." << ↩
        endl;
    for (int i = 0; i < (-1)*nCells_y; i++)
    {

     if (Pstream::parRun() && ny > 1)
     {
        #include "dragUpComm.H"
```

```
    }
    else
   {
    for (int cellI = mesh.cells().size()-1; cellI >=0; ↩
       cellI--)
    {
        labelList neighbors = mesh.cellCells()[cellI];
  for (int itr=0; itr<neighbors.size(); itr++)
        {
      if (mesh.C()[cellI].y() - mesh.C()[neighbors[itr↩
         ]].y() > 1e-10)
            {
    alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
       1 cell
    p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
       cell
    U[cellI] = U[neighbors[itr]]; //drag back 1 cell
            }
        }
    }
   }
   }
}
}

if (zMove.value() == 1)
{
if (nCells_z > 0)
{
    drag = true;
    Info << "Drag back back " << nCells_z << " cells." ↩
       << endl;
   for (int i = 0; i < nCells_z; i++)
   {

    if (Pstream::parRun() && nz > 1)
    {
       #include "dragBackComm.H"
    }
    else
   {
```

```
    for (int cellI = 0; cellI < mesh.cells().size(); ↩
       cellI++)
    {
        labelList neighbors = mesh.cellCells()[cellI];
  for (int itr=0; itr<neighbors.size(); itr++)
        {
      if (mesh.C()[neighbors[itr]].z() - mesh.C()[cellI↩
         ].z() > 1e-10)
            {
    alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
       1 cell
    p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
       cell
    U[cellI] = U[neighbors[itr]]; //drag back 1 cell
            }
        }
    }
   }
   }
}

else if (nCells_z < 0)
{
    drag = true;
    Info << "Drag back front " << nCells_z << " cells." ↩
       << endl;
   for (int i = 0; i < (-1)*nCells_z; i++)
   {

    if (Pstream::parRun() && nz > 1)
    {
       #include "dragFrontComm.H"
    }
    else
   {
    for (int cellI = mesh.cells().size()-1; cellI >=0; ↩
       cellI--)
    {
        labelList neighbors = mesh.cellCells()[cellI];
  for (int itr=0; itr<neighbors.size(); itr++)
        {
```

```
      if (mesh.C()[cellI].z() - mesh.C()[neighbors[itr↩
         ]].z() > 1e-10)
            {
    alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
       1 cell
    p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
       cell
    U[cellI] = U[neighbors[itr]]; //drag back 1 cell
            }
        }
    }
   }
   }
}
}

if (drag)
{
    phi = linearInterpolate(U) & mesh.Sf();
    #include "correctPhi.H"
    rhoPhi = rho1*phi;
    interface.correct();
    rho = alpha1*rho1 + (scalar(1.0)-alpha1)*rho2;
}
```

dragBackComm.H

```
{
 //Loop over processor patches
 Info << "Sending stuff" << endl;
 forAll (mesh.boundaryMesh(),patchInd)
 {
  const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
     ];
  if (typeid(patch) == typeid(processorPolyPatch))
  {
```

```
 const processorPolyPatch& procpatch = dynamic_cast<↩
    const processorPolyPatch&>(patch);

 //only for master chunk
 for (int i = 0; i < senderBack.size(); i++)
 {
 if (procpatch.myProcNo() == senderBack[i] && ↩
    procpatch.neighbProcNo() == receiverBack[i])
 {
  //Make buffers
  Field<scalar> mybuffer_alpha1(patch.size());
  Field<scalar> mybuffer_p_rgh(patch.size());
  Field<vector> mybuffer_U(patch.size());
  const labelList& internalcells = patch.faceCells();
  forAll(internalcells, ind)
  {
   label curcell = internalcells[ind];
   mybuffer_alpha1[ind] = alpha1[curcell];
   mybuffer_p_rgh[ind] = p_rgh[curcell];
   mybuffer_U[ind] = U[curcell];
  }

  //Send buffer to neighbor
  OPstream tNP(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,0);
  tNP << mybuffer_alpha1 << endl;
  OPstream tNP2(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,1);
  tNP2 << mybuffer_p_rgh << endl;
  OPstream tNP3(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8*3,2);
  tNP3 << mybuffer_U << endl;

  //drag back in subdomain start
  for (int cellI = 0; cellI < mesh.cells().size(); ↩
     cellI++)
  {
      labelList neighbors = mesh.cellCells()[cellI];
for (int itr=0; itr<neighbors.size(); itr++)
      {
     if (mesh.C()[neighbors[itr]].z() - mesh.C()[cellI↩
        ].z() > 1e-10)
```

```
            {
    alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
       1 cell
    p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
       cell
    U[cellI] = U[neighbors[itr]]; //drag back 1 cell
            }
        }
    }
    //drag back in subdomain end
    break;
  }
 }
 }
}

forAll (mesh.boundaryMesh(),patchInd)
{
 const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
    ];
 if (typeid(patch) == typeid(processorPolyPatch))
 {
  const processorPolyPatch& procpatch = dynamic_cast<↩
     const processorPolyPatch&>(patch);

  for (int i = 0; i < receiverBack.size(); i++)
  {
  if (procpatch.myProcNo() == receiverBack[i] && ↩
     procpatch.neighbProcNo() == senderBack[i])
  {
   for (int j = 0; j < receiverOnlyBack.size(); j++)
   {
   if (procpatch.myProcNo() == receiverOnlyBack[j])
   {

   //drag back in subdomain start
   for (int cellI = 0; cellI < mesh.cells().size(); ↩
      cellI++)
   {
       labelList neighbors = mesh.cellCells()[cellI];
 for (int itr=0; itr<neighbors.size(); itr++)
       {
```

```
    if (mesh.C()[neighbors[itr]].z() - mesh.C()[cellI↩
       ].z() > 1e-10)
           {
  alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
     1 cell
  p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
     cell
  U[cellI] = U[neighbors[itr]]; //drag back 1 cell
           }
      }
  }
  //drag back in subdomain end
  break;
  }
 }

  //only for slave chunk
  //Make buffer
  Field<scalar> yourbuffer_alpha1(patch.size());
  IPstream fNP(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,0);
  fNP >> yourbuffer_alpha1;
  Field<scalar> yourbuffer_p_rgh(patch.size());
  IPstream fNP2(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,1);
  fNP2 >> yourbuffer_p_rgh;
  Field<vector> yourbuffer_U(patch.size());
  IPstream fNP3(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8*3,2);
  fNP3 >> yourbuffer_U;

  {
   const labelList& internalcells = patch.faceCells();
   forAll(internalcells, ind)
   {
    label curcell = internalcells[ind];
    alpha1[curcell] = yourbuffer_alpha1[ind];
    p_rgh[curcell] = yourbuffer_p_rgh[ind];
    U[curcell] = yourbuffer_U[ind];
   }
  }
  break;
```

```
    }
    }
   }
  }
 }
```

dragFrontComm.H

```
{
 //Loop over processor patches
 Info << "Sending stuff" << endl;
 forAll (mesh.boundaryMesh(),patchInd)
 {
  const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
     ];
  if (typeid(patch) == typeid(processorPolyPatch))
  {
   const processorPolyPatch& procpatch = dynamic_cast<↩
      const processorPolyPatch&>(patch);
   //only for master chunk -> cut work in half
   for (int i = 0; i < senderFront.size(); i++)
   {
   if (procpatch.myProcNo() == senderFront[i] && ↩
      procpatch.neighbProcNo() == receiverFront[i])
   {
    //Make buffers
    Field<scalar> mybuffer_alpha1(patch.size());
    Field<scalar> mybuffer_p_rgh(patch.size());
    Field<vector> mybuffer_U(patch.size());
    const labelList& internalcells = patch.faceCells();
    forAll(internalcells, ind)
    {
     label curcell = internalcells[ind];
     mybuffer_alpha1[ind] = alpha1[curcell];
     mybuffer_p_rgh[ind] = p_rgh[curcell];
     mybuffer_U[ind] = U[curcell];
    }
```

```
    //Send buffer to neighbor
    OPstream tNP(Pstream::blocking,procpatch.↩
       neighbProcNo(),patch.size()*8,0);
    tNP << mybuffer_alpha1 << endl;
    OPstream tNP2(Pstream::blocking,procpatch.↩
       neighbProcNo(),patch.size()*8,1);
    tNP2 << mybuffer_p_rgh << endl;
    OPstream tNP3(Pstream::blocking,procpatch.↩
       neighbProcNo(),patch.size()*8*3,2);
    tNP3 << mybuffer_U << endl;

    //drag back in subdomain start
    for (int cellI = mesh.cells().size()-1; cellI >=0; ↩
       cellI--)
    {
        labelList neighbors = mesh.cellCells()[cellI];
  for (int itr=0; itr<neighbors.size(); itr++)
        {
      if (mesh.C()[cellI].z() - mesh.C()[neighbors[itr↩
         ]].z() > 1e-10)
            {
    alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
       1 cell
    p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
       cell
    U[cellI] = U[neighbors[itr]]; //drag back 1 cell
            }
        }
    }
    //drag back in subdomain end
    break;
   }
  }
  }
}

forAll (mesh.boundaryMesh(),patchInd)
{
  const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
     ];
  if (typeid(patch) == typeid(processorPolyPatch))
```

```
{
 const processorPolyPatch& procpatch = dynamic_cast<↩
    const processorPolyPatch&>(patch);

 for (int i = 0; i < receiverFront.size(); i++)
 {
 if (procpatch.myProcNo() == receiverFront[i] && ↩
    procpatch.neighbProcNo() == senderFront[i])
 {
  for (int j = 0; j < receiverOnlyFront.size(); j++)
  {
  if (procpatch.myProcNo() == receiverOnlyFront[j])
  {

  //drag back in subdomain start
  for (int cellI = mesh.cells().size()-1; cellI >=0; ↩
     cellI--)
  {
      labelList neighbors = mesh.cellCells()[cellI];
for (int itr=0; itr<neighbors.size(); itr++)
      {
     if (mesh.C()[cellI].z() - mesh.C()[neighbors[itr↩
        ]].z() > 1e-10)
           {
  alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
     1 cell
  p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
     cell
  U[cellI] = U[neighbors[itr]]; //drag back 1 cell
           }
      }
  }
  //drag back in subdomain end
  break;
  }
 }

  //only for slave chunk
  //Make buffer
  Field<scalar> yourbuffer_alpha1(patch.size());
  IPstream fNP(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,0);
```

```
    fNP >> yourbuffer_alpha1;
    Field<scalar> yourbuffer_p_rgh(patch.size());
    IPstream fNP2(Pstream::blocking,procpatch.↩
       neighbProcNo(),patch.size()*8,1);
    fNP2 >> yourbuffer_p_rgh;
    Field<vector> yourbuffer_U(patch.size());
    IPstream fNP3(Pstream::blocking,procpatch.↩
       neighbProcNo(),patch.size()*8*3,2);
    fNP3 >> yourbuffer_U;

    {
     const labelList& internalcells = patch.faceCells();
     forAll(internalcells, ind)
     {
      label curcell = internalcells[ind];
      alpha1[curcell] = yourbuffer_alpha1[ind];
      p_rgh[curcell] = yourbuffer_p_rgh[ind];
      U[curcell] = yourbuffer_U[ind];
     }
    }
    break;
   }
   }
  }
 }
}
```

dragDownComm.H

```
{
 //Loop over processor patches
 Info << "Sending stuff" << endl;
 forAll (mesh.boundaryMesh(),patchInd)
 {
  const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
     ];
  if (typeid(patch) == typeid(processorPolyPatch))
```

```
{
 const processorPolyPatch& procpatch = dynamic_cast<↩
    const processorPolyPatch&>(patch);
 //only for master chunk -> cut work in half
 for (int i = 0; i < senderDown.size(); i++)
 {
 if (procpatch.myProcNo() == senderDown[i] && ↩
    procpatch.neighbProcNo() == receiverDown[i])
 {
  //Make buffers
  Field<scalar> mybuffer_alpha1(patch.size());
  Field<scalar> mybuffer_p_rgh(patch.size());
  Field<vector> mybuffer_U(patch.size());
  const labelList& internalcells = patch.faceCells();
  forAll(internalcells, ind)
  {
   label curcell = internalcells[ind];
   mybuffer_alpha1[ind] = alpha1[curcell];
   mybuffer_p_rgh[ind] = p_rgh[curcell];
   mybuffer_U[ind] = U[curcell];
  }

  //Send buffer to neighbor
  OPstream tNP(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,0);
  tNP << mybuffer_alpha1 << endl;
  OPstream tNP2(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,1);
  tNP2 << mybuffer_p_rgh << endl;
  OPstream tNP3(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8*3,2);
  tNP3 << mybuffer_U << endl;

  //drag back in subdomain start
  for (int cellI = 0; cellI < mesh.cells().size(); ↩
     cellI++)
  {
      labelList neighbors = mesh.cellCells()[cellI];
for (int itr=0; itr<neighbors.size(); itr++)
      {
    if (mesh.C()[neighbors[itr]].y() - mesh.C()[cellI↩
       ].y() > 1e-10)
```

```
            {
    alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
       1 cell
    p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
       cell
    U[cellI] = U[neighbors[itr]]; //drag back 1 cell
            }
        }
    }
    //drag back in subdomain end
    break;
  }
  }
 }
 }

forAll (mesh.boundaryMesh(),patchInd)
{
 const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
    ];
 if (typeid(patch) == typeid(processorPolyPatch))
 {
  const processorPolyPatch& procpatch = dynamic_cast<↩
     const processorPolyPatch&>(patch);

  for (int i = 0; i < receiverDown.size(); i++)
  {
  if (procpatch.myProcNo() == receiverDown[i] && ↩
     procpatch.neighbProcNo() == senderDown[i])
  {
   for (int j = 0; j < receiverOnlyDown.size(); j++)
   {
   if (procpatch.myProcNo() == receiverOnlyDown[j])
   {

   //drag back in subdomain start
   for (int cellI = 0; cellI < mesh.cells().size(); ↩
      cellI++)
   {
       labelList neighbors = mesh.cellCells()[cellI];
 for (int itr=0; itr<neighbors.size(); itr++)
       {
```

```
    if (mesh.C()[neighbors[itr]].y() - mesh.C()[cellI↩
       ].y() > 1e-10)
           {
 alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
    1 cell
 p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
    cell
 U[cellI] = U[neighbors[itr]]; //drag back 1 cell
           }
      }
 }
 //drag back in subdomain end
 break;
 }
}
 //only for slave chunk
 //Make buffer
 Field<scalar> yourbuffer_alpha1(patch.size());
 IPstream fNP(Pstream::blocking,procpatch.↩
    neighbProcNo(),patch.size()*8,0);
 fNP >> yourbuffer_alpha1;
 Field<scalar> yourbuffer_p_rgh(patch.size());
 IPstream fNP2(Pstream::blocking,procpatch.↩
    neighbProcNo(),patch.size()*8,1);
 fNP2 >> yourbuffer_p_rgh;
 Field<vector> yourbuffer_U(patch.size());
 IPstream fNP3(Pstream::blocking,procpatch.↩
    neighbProcNo(),patch.size()*8*3,2);
 fNP3 >> yourbuffer_U;

 {
  const labelList& internalcells = patch.faceCells();
  forAll(internalcells, ind)
  {
   label curcell = internalcells[ind];
   alpha1[curcell] = yourbuffer_alpha1[ind];
   p_rgh[curcell] = yourbuffer_p_rgh[ind];
   U[curcell] = yourbuffer_U[ind];
  }
 }
 break;
}
```

```
    }
   }
  }
 }
}
```

dragUpComm.H

```
{
 //Loop over processor patches
 Info << "Sending stuff" << endl;
 forAll (mesh.boundaryMesh(),patchInd)
 {
  const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
     ];
  if (typeid(patch) == typeid(processorPolyPatch))
  {
   const processorPolyPatch& procpatch = dynamic_cast<↩
      const processorPolyPatch&>(patch);
   //only for master chunk -> cut work in half
   for (int i = 0; i < senderUp.size(); i++)
   {
   if (procpatch.myProcNo() == senderUp[i] && procpatch.↩
      neighbProcNo() == receiverUp[i])
   {
    //Make buffers
    Field<scalar> mybuffer_alpha1(patch.size());
    Field<scalar> mybuffer_p_rgh(patch.size());
    Field<vector> mybuffer_U(patch.size());
    const labelList& internalcells = patch.faceCells();
    forAll(internalcells, ind)
    {
     label curcell = internalcells[ind];
     mybuffer_alpha1[ind] = alpha1[curcell];
     mybuffer_p_rgh[ind] = p_rgh[curcell];
     mybuffer_U[ind] = U[curcell];
    }
```

```
    //Send buffer to neighbor
    OPstream tNP(Pstream::blocking,procpatch.↩
       neighbProcNo(),patch.size()*8,0);
    tNP << mybuffer_alpha1 << endl;
    OPstream tNP2(Pstream::blocking,procpatch.↩
       neighbProcNo(),patch.size()*8,1);
    tNP2 << mybuffer_p_rgh << endl;
    OPstream tNP3(Pstream::blocking,procpatch.↩
       neighbProcNo(),patch.size()*8*3,2);
    tNP3 << mybuffer_U << endl;

    //drag back in subdomain start
    for (int cellI = mesh.cells().size()-1; cellI >=0; ↩
       cellI--)
    {
        labelList neighbors = mesh.cellCells()[cellI];
  for (int itr=0; itr<neighbors.size(); itr++)
        {
      if (mesh.C()[cellI].y() - mesh.C()[neighbors[itr↩
         ]].y() > 1e-10)
            {
    alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
       1 cell
    p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
       cell
    U[cellI] = U[neighbors[itr]]; //drag back 1 cell
            }
        }
    }
    //drag back in subdomain end
    break;
   }
   }
  }
}

forAll (mesh.boundaryMesh(),patchInd)
{
  const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
     ];
  if (typeid(patch) == typeid(processorPolyPatch))
  {
```

```
 const processorPolyPatch& procpatch = dynamic_cast<↩
    const processorPolyPatch&>(patch);

 for (int i = 0; i < receiverUp.size(); i++)
 {
 if (procpatch.myProcNo() == receiverUp[i] && ↩
    procpatch.neighbProcNo() == senderUp[i])
 {
  for (int j = 0; j < receiverOnlyUp.size(); j++)
  {
  if (procpatch.myProcNo() == receiverOnlyUp[j])
  {

  //drag back in subdomain start
  for (int cellI = mesh.cells().size()-1; cellI >=0; ↩
     cellI--)
  {
      labelList neighbors = mesh.cellCells()[cellI];
for (int itr=0; itr<neighbors.size(); itr++)
      {
     if (mesh.C()[cellI].y() - mesh.C()[neighbors[itr↩
        ]].y() > 1e-10)
           {
  alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
     1 cell
  p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
     cell
  U[cellI] = U[neighbors[itr]]; //drag back 1 cell
           }
      }
  }
  //drag back in subdomain end
  break;
  }
 }

  //only for slave chunk
  //Make buffer
  Field<scalar> yourbuffer_alpha1(patch.size());
  IPstream fNP(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,0);
  fNP >> yourbuffer_alpha1;
```

```
    Field<scalar> yourbuffer_p_rgh(patch.size());
    IPstream fNP2(Pstream::blocking,procpatch.↩
       neighbProcNo(),patch.size()*8,1);
    fNP2 >> yourbuffer_p_rgh;
    Field<vector> yourbuffer_U(patch.size());
    IPstream fNP3(Pstream::blocking,procpatch.↩
       neighbProcNo(),patch.size()*8*3,2);
    fNP3 >> yourbuffer_U;

    {
     const labelList& internalcells = patch.faceCells();
     forAll(internalcells, ind)
     {
      label curcell = internalcells[ind];
      alpha1[curcell] = yourbuffer_alpha1[ind];
      p_rgh[curcell] = yourbuffer_p_rgh[ind];
      U[curcell] = yourbuffer_U[ind];
     }
    }
    break;
   }
   }
  }
 }
}
```

dragLeftComm.H

```
{
 //Loop over processor patches
 forAll (mesh.boundaryMesh(),patchInd)
 {
  const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
     ];
  if (typeid(patch) == typeid(processorPolyPatch))
  {
```

```
 const processorPolyPatch& procpatch = dynamic_cast<↩
    const processorPolyPatch&>(patch);

 for (int i = 0; i < senderLeft.size(); i++)
 {

 if (procpatch.myProcNo() == senderLeft[i] && ↩
    procpatch.neighbProcNo() == receiverLeft[i])
 {
  //Make buffers
  Field<scalar> mybuffer_alpha1(patch.size());
  Field<scalar> mybuffer_p_rgh(patch.size());
  Field<vector> mybuffer_U(patch.size());
  const labelList& internalcells = patch.faceCells();

  forAll(internalcells, ind)
  {
   label curcell = internalcells[ind];
   mybuffer_alpha1[ind] = alpha1[curcell];
   mybuffer_p_rgh[ind] = p_rgh[curcell];
   mybuffer_U[ind] = U[curcell];
  }

  //Send buffer to neighbor
  OPstream tNP(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,0);
  tNP << mybuffer_alpha1 << endl;
  OPstream tNP2(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,1);
  tNP2 << mybuffer_p_rgh << endl;
  OPstream tNP3(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8*3,2);
  tNP3 << mybuffer_U << endl;

  //drag back in subdomain start
  for (int cellI = 0; cellI < mesh.cells().size(); ↩
     cellI++)
  {
      labelList neighbors = mesh.cellCells()[cellI];
for (int itr=0; itr<neighbors.size(); itr++)
      {
```

```
      if (mesh.C()[neighbors[itr]].x() - mesh.C()[cellI↩
         ].x() > 1e-10)
            {
    alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
       1 cell
    p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
       cell
    U[cellI] = U[neighbors[itr]]; //drag back 1 cell
            }
        }
    }
    //drag back in subdomain end
    break;
   }
   }
   }
}

forAll (mesh.boundaryMesh(),patchInd)
{
 const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
    ];
 if (typeid(patch) == typeid(processorPolyPatch))
 {
  const processorPolyPatch& procpatch = dynamic_cast<↩
     const processorPolyPatch&>(patch);

  for (int i = 0; i < receiverLeft.size(); i++)
  {
  if (procpatch.myProcNo() == receiverLeft[i] && ↩
     procpatch.neighbProcNo() == senderLeft[i])
  {
   for (int j = 0; j < receiverOnlyLeft.size(); j++)
   {
   if (procpatch.myProcNo() == receiverOnlyLeft[j])
   {
    //drag back in subdomain start
    for (int cellI = 0; cellI < mesh.cells().size(); ↩
       cellI++)
    {
       labelList neighbors = mesh.cellCells()[cellI];
 for (int itr=0; itr<neighbors.size(); itr++)
```

```
    {
  if (mesh.C()[neighbors[itr]].x() - mesh.C()[cellI↩
     ].x() > 1e-10)
        {
alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
   1 cell
p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
   cell
U[cellI] = U[neighbors[itr]]; //drag back 1 cell
        }
    }
 }
 break;
}
}
//drag back in subdomain end
//only for slave chunk
//Make buffer

Field<scalar> yourbuffer_alpha1(patch.size());
IPstream fNP(Pstream::blocking,procpatch.↩
   neighbProcNo(),patch.size()*8,0);
fNP >> yourbuffer_alpha1;

Field<scalar> yourbuffer_p_rgh(patch.size());
IPstream fNP2(Pstream::blocking,procpatch.↩
   neighbProcNo(),patch.size()*8,1);
fNP2 >> yourbuffer_p_rgh;

Field<vector> yourbuffer_U(patch.size());
IPstream fNP3(Pstream::blocking,procpatch.↩
   neighbProcNo(),patch.size()*8*3,2);
fNP3 >> yourbuffer_U;


{
 const labelList& internalcells = patch.faceCells();

 forAll(internalcells, ind)
 {
  label curcell = internalcells[ind];
  alpha1[curcell] = yourbuffer_alpha1[ind];
```

```
      p_rgh[curcell] = yourbuffer_p_rgh[ind];
      U[curcell] = yourbuffer_U[ind];
     }

    }
    break;
   }
   }
  }
 }
}
```

dragRightComm.H

```
{

 //Loop over processor patches

 forAll (mesh.boundaryMesh(),patchInd)
 {
  const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
     ];
  if (typeid(patch) == typeid(processorPolyPatch))
  {
   const processorPolyPatch& procpatch = dynamic_cast<↩
      const processorPolyPatch&>(patch);

   for (int i = 0; i < senderRight.size(); i++)
   {
   if (procpatch.myProcNo() == senderRight[i] && ↩
      procpatch.neighbProcNo() == receiverRight[i])
   {

    //Make buffers
    Field<scalar> mybuffer_alpha1(patch.size());
    Field<scalar> mybuffer_p_rgh(patch.size());
    Field<vector> mybuffer_U(patch.size());
```

```
  const labelList& internalcells = patch.faceCells();
  forAll(internalcells, ind)
  {
   label curcell = internalcells[ind];
   mybuffer_alpha1[ind] = alpha1[curcell];
   mybuffer_p_rgh[ind] = p_rgh[curcell];
   mybuffer_U[ind] = U[curcell];
  }

  //Send buffer to neighbor

  OPstream tNP(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,0);
  tNP << mybuffer_alpha1 << endl;
  OPstream tNP2(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8,1);
  tNP2 << mybuffer_p_rgh << endl;
  OPstream tNP3(Pstream::blocking,procpatch.↩
     neighbProcNo(),patch.size()*8*3,2);
  tNP3 << mybuffer_U << endl;

  //drag back in subdomain start
  for (int cellI = mesh.cells().size()-1; cellI >=0; ↩
     cellI--)
  {
      labelList neighbors = mesh.cellCells()[cellI];
for (int itr=0; itr<neighbors.size(); itr++)
      {
    if (mesh.C()[cellI].x() - mesh.C()[neighbors[itr↩
       ]].x() > 1e-10)
          {
  alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
     1 cell
  p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
     cell
  U[cellI] = U[neighbors[itr]]; //drag back 1 cell
          }
      }
  }
  //drag back in subdomain end
  break;
 }
```

```
  }
 }
}

forAll (mesh.boundaryMesh(),patchInd)
{
 const polyPatch& patch = mesh.boundaryMesh()[patchInd↩
    ];
 if (typeid(patch) == typeid(processorPolyPatch))
 {
  const processorPolyPatch& procpatch = dynamic_cast<↩
     const processorPolyPatch&>(patch);

  for (int i = 0; i < receiverRight.size(); i++)
  {
  if (procpatch.myProcNo() == receiverRight[i] && ↩
     procpatch.neighbProcNo() == senderRight[i])
  {
   for (int j = 0; j < receiverOnlyRight.size(); j++)
   {
   if (procpatch.myProcNo() == receiverOnlyRight[j])
   {
    //drag back in subdomain start
    for (int cellI = mesh.cells().size()-1; cellI >=0; ↩
       cellI--)
    {
       labelList neighbors = mesh.cellCells()[cellI];
 for (int itr=0; itr<neighbors.size(); itr++)
       {
     if (mesh.C()[cellI].x() - mesh.C()[neighbors[itr↩
        ]].x() > 1e-10)
           {
   alpha1[cellI] = alpha1[neighbors[itr]]; //drag back ↩
      1 cell
   p_rgh[cellI] = p_rgh[neighbors[itr]]; //drag back 1 ↩
      cell
   U[cellI] = U[neighbors[itr]]; //drag back 1 cell
           }
       }
    }
    //drag back in subdomain end
    break;
```

```
      }
     }

     //only for slave chunk
     //Make buffer

     Field<scalar> yourbuffer_alpha1(patch.size());
     IPstream fNP(Pstream::blocking,procpatch.↩
        neighbProcNo(),patch.size()*8,0);
     fNP >> yourbuffer_alpha1;
     Field<scalar> yourbuffer_p_rgh(patch.size());
     IPstream fNP2(Pstream::blocking,procpatch.↩
        neighbProcNo(),patch.size()*8,1);
     fNP2 >> yourbuffer_p_rgh;
     Field<vector> yourbuffer_U(patch.size());
     IPstream fNP3(Pstream::blocking,procpatch.↩
        neighbProcNo(),patch.size()*8*3,2);
     fNP3 >> yourbuffer_U;

     {
      const labelList& internalcells = patch.faceCells();
      forAll(internalcells, ind)
      {
       label curcell = internalcells[ind];
       alpha1[curcell] = yourbuffer_alpha1[ind];
       p_rgh[curcell] = yourbuffer_p_rgh[ind];
       U[curcell] = yourbuffer_U[ind];
      }

     }
     break;
    }
   }
  }
 }
}
```

Appendix F

modPolyMeltFoam Code

modPolyMeltFoam.C

```
#include "fvCFD.H"
#include "mathematicalConstants.H"
#include "pimpleControl.H"

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

    pimpleControl pimple(mesh);

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

    while (runTime.loop())
    {
```

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

        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setDeltaT.H"

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

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

        runTime.write();

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

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

    return 0;
}
```

createFields.H

```
    // Reading fields
    Info<< "Reading field T\n" << endl;
    volScalarField T
    (
        IOobject
        (
            "T",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
```

```
);

Info<< "Reading field alpha\n" << endl;
volScalarField alpha
(
    IOobject
    (
        "alpha",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field p_rgh\n" << endl;
volScalarField p_rgh
(
    IOobject
    (
        "p_rgh",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

#include "createPhi.H"

// Reading transport properties
Info<< "Reading thermophysical properties\n" << endl↩
    ;
#include "readTransportProperties.H"

// Calculating fit-parameters for phase change ↩
    function
Info<< "Calculating phase change properties\n" << ↩
    endl;

dimensionedScalar Tmelt
```

```
(
    "Tmelt",
(Tl+Ts)/2.0
);
// Kinematic density for buoyancy force
//set T, rhoS, rho to be dimensionless starts
T.dimensions().reset(dimless);
rhoS.dimensions().reset(dimless);
rho.dimensions().reset(dimless);
//set T, rhoS, rho to be dimensionless ends

volScalarField rhok
(
    IOobject
    (
        "rhok",
        runTime.timeName(),
        mesh
    ),
    (alpha*(2205.355538389409+T*(-29.29618917818346
    +T*(0.21520074533582612+T↩
       *(-0.00063521036689868 39
    +T*6.62139792627e-7))))
    + (1.0-alpha)*rhoS
    )/rho


);
//reset back T, rhoS, rho to be dimensional starts
T.dimensions().reset(dimensionSet(0,0,0,1,0,0,0));
rhoS.dimensions().reset(dimensionSet(1,-3,0,0,0,0,0)↩
   );
rho.dimensions().reset(dimensionSet(1,-3,0,0,0,0,0))↩
   ;
//reset back T, rhoS, rho to be dimensional ends
// D'arcy-type source term field
volScalarField DC
(
    IOobject
    (
        "DC",
        runTime.timeName(),
        mesh
    ),
    DCl*Foam::pow(1.0-alpha,2)/(Foam::pow(alpha,3)+↩
       DCs)
);
// Thermal conductivity field
volScalarField lambda
```

```
(
    IOobject
    (
        "lambda",
        runTime.timeName(),
        mesh
    ),
    alpha*lambdaL+(1.0-alpha)*lambdaS
);

// Heat capacity field
volScalarField cp
(
    IOobject
    (
        "cp",
        runTime.timeName(),
        mesh
    ),
    alpha*cpL+(1.0-alpha)*cpS
);

// Kinematic viscosity field
volScalarField nu
(
    IOobject
    (
        "nu",
        runTime.timeName(),
        mesh
    ),
    alpha*nuL+(1.0-alpha)*nuS
);

Info<< "Calculating field g.h\n" << endl;
volScalarField gh("gh", g & mesh.C());
surfaceScalarField ghf("ghf", g & mesh.Cf());

volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    p_rgh + rhok*gh
);

label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell
(
    p,
```

```
        p_rgh,
        mesh.solutionDict().subDict("PIMPLE"),
        pRefCell,
        pRefValue
    );

    if (p_rgh.needReference())
    {
        p += dimensionedScalar
        (
            "p",
            p.dimensions(),
            pRefValue - getRefCellValue(p, pRefCell)
        );
    }
```

readTransportProperties.H

```
    IOdictionary transportProperties
    (
        IOobject
        (
            "transportProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );

  dimensionedScalar pi = constant::mathematical::pi;

    // solid -> phase 1
    // liquid -> phase 2

    // Reading density rho
    dimensionedScalar rho(transportProperties.lookup("↩
       rho"));

    // Reading density rhoS
    dimensionedScalar rhoS(transportProperties.lookup("↩
       rhoS"));

    // Reading thermal conductivity lambda
    dimensionedScalar lambdaS(transportProperties.lookup↩
       ("lambdaS"));
    dimensionedScalar lambdaL(transportProperties.lookup↩
       ("lambdaL"));
```

```
// Reading heat capacity cp
dimensionedScalar cpS(transportProperties.lookup("↩
    cpS"));
dimensionedScalar cpL(transportProperties.lookup("↩
    cpL"));

// Reading kinematic viscosity
dimensionedScalar nuS(transportProperties.lookup("↩
    nuS"));
dimensionedScalar nuL(transportProperties.lookup("↩
    nuL"));

// Reading latent heat of fusion hs
dimensionedScalar hs(transportProperties.lookup("hs"↩
    ));

// Reading solid bound of melting temperature Ts
dimensionedScalar Ts(transportProperties.lookup("Ts"↩
    ));

// Reading liquid bound of melting temperature Tl
dimensionedScalar Tl(transportProperties.lookup("Tl"↩
    ));

// Reading volume expansion factor beta
dimensionedScalar beta(transportProperties.lookup("↩
    beta"));

// Reading large D'arcy-type source term constant ↩
    DCl
dimensionedScalar DCl(transportProperties.lookup("↩
    DCl"));

// Reading small D'arcy-type source term constant ↩
    DCs
dimensionedScalar DCs(transportProperties.lookup("↩
    DCs"));
```

UEqn.H

```
// Solve the momentum equation

fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvm::laplacian(nu, U)
```

```
      + fvm::SuSp(DC, U)
    );

    UEqn.relax();

    if (pimple.momentumPredictor())
    {
        solve
        (
            UEqn
         ==
            fvc::reconstruct
            (
                (
                  - ghf*fvc::snGrad(rhok)
                  - fvc::snGrad(p_rgh)
                )*mesh.magSf()
            )
        );
    }
```

pEqn.H

```
{
    volScalarField rAU("rAU", 1.0/UEqn.A());
    surfaceScalarField rAUf("(1|A(U))", fvc::interpolate↩
       (rAU));

    U = rAU*UEqn.H();

    phi = (fvc::interpolate(U) & mesh.Sf())
        + fvc::ddtPhiCorr(rAU, U, phi);

    surfaceScalarField buoyancyPhi(rAUf*ghf*fvc::snGrad(↩
       rhok)*mesh.magSf());
    phi -= buoyancyPhi;

    while (pimple.correctNonOrthogonal())
    {
        fvScalarMatrix p_rghEqn
        (
            fvm::laplacian(rAUf, p_rgh) == fvc::div(phi)
        );

        p_rghEqn.setReference(pRefCell, getRefCellValue(↩
           p_rgh, pRefCell));
```

```
        p_rghEqn.solve(mesh.solver(p_rgh.select(pimple.↩
           finalInnerIter())));

        if (pimple.finalNonOrthogonalIter())
        {
            // Calculate the conservative fluxes
            phi -= p_rghEqn.flux();

            // Explicitly relax pressure for momentum ↩
               corrector
            p_rgh.relax();

            // Correct the momentum source with the ↩
               pressure gradient flux
            // calculated from the relaxed pressure
            U -= rAU*fvc::reconstruct((buoyancyPhi + ↩
               p_rghEqn.flux())/rAUf);
            U.correctBoundaryConditions();
        }
    }

    #include "continuityErrs.H"

    p = p_rgh + rhok*gh;

    if (p_rgh.needReference())
    {
        p += dimensionedScalar
        (
            "p",
            p.dimensions(),
            pRefValue - getRefCellValue(p, pRefCell)
        );
        p_rgh = p - rhok*gh;
    }
}
```

TEqn.H

```
// Solving the energy equation
{
    volScalarField coeff
    (
        IOobject
        (
            "coeff",
            runTime.timeName(),
            mesh
```

```
        ),
        4.0*exp(-pow(4.0*(T-Tmelt)/(Tl-Ts),2))/Foam::↩
           sqrt(pi)/(Tl-Ts)

    );

    fvScalarMatrix TEqn
    (
        cp*fvm::ddt(T)
      + hs*coeff*fvm::ddt(T)
      + (U & (cp*fvc::grad(T)+hs*coeff*fvc::grad(T)))
      - fvm::laplacian(lambda/rho, T)
    );

    TEqn.relax();
    TEqn.solve();

    alpha = 0.5*Foam::erf(4.0*(T-Tmelt)/(Tl-Ts))+scalar↩
       (0.5);

    T.dimensions().reset(dimless);
    rhoS.dimensions().reset(dimless);
    rho.dimensions().reset(dimless);

    rhok = (alpha*(2205.355538389409+T↩
       *(-29.29618917818346
          +T*(0.21520074533582612+T↩
             *(-0.0006352103668986839
          +T*6.62139792627e-7))))
            + (1.0-alpha)*rhoS
           )/rho;

    T.dimensions().reset(dimensionSet(0,0,0,1,0,0,0));
    rhoS.dimensions().reset(dimensionSet(1,-3,0,0,0,0,0)↩
       );
    rho.dimensions().reset(dimensionSet(1,-3,0,0,0,0,0))↩
       ;

    cp = alpha*cpL+(1.0-alpha)*cpS;
    lambda = alpha*lambdaL+(1.0-alpha)*lambdaS;
    nu = alpha*nuL+(1.0-alpha)*nuS;
    DC = DCl*Foam::pow(1.0-alpha,2)/(Foam::pow(alpha,3)+↩
       DCs);
}
```

Appendix G

modFluidFluidChtMultiRegionFoam Code

modFluidFluidChtMultiRegionFoam.C

```
#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulenceModel.H"
#include "fixedGradientFvPatchFields.H"
#include "regionProperties.H"
#include "icoCourantNo.H"
#include "solidCourantNo.H"

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

    regionProperties rp(runTime);

    #include "createFluidMeshes.H"
```

```
#include "createSolidMeshes.H"

#include "createFluidFields.H"
#include "createSolidFields.H"

#include "initContinuityErrs.H"
#include "readTimeControls.H"
#include "icoMultiRegionCourantNo.H"
#include "solidMultiRegionCourantNo.H"
#include "setInitialMultiRegionDeltaT.H"

while (runTime.run())
{
    #include "readTimeControls.H"
    #include "readPIMPLEControls.H"
    #include "icoMultiRegionCourantNo.H"
    #include "solidMultiRegionCourantNo.H"
    #include "setMultiRegionDeltaT.H"

    runTime++;

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

    if (nOuterCorr != 1)
    {
        forAll(fluidRegions, i)
        {
            #include "setRegionFluidFields.H"
            #include "storeOldFluidFields.H"
        }

  forAll(solidRegions, i)
  {
            #include "setRegionSolidFields.H"
      #include "storeOldSolidFields.H"
  }
    }

    // --- PIMPLE loop
    for (int oCorr=0; oCorr<nOuterCorr; oCorr++)
    {
        bool finalIter = oCorr == nOuterCorr-1;

        forAll(fluidRegions, i)
        {
            Info<< "\nSolving for fluid region "
                << fluidRegions[i].name() << endl;
            #include "setRegionFluidFields.H"
            #include "↩
               readFluidMultiRegionPIMPLEControls.H"
            #include "solveFluid.H"
        }

        forAll(solidRegions, i)
        {
            Info<< "\nSolving for solid region "
```

```
                    << solidRegions[i].name() << endl;
                #include "setRegionSolidFields.H"
                #include "↩
                   readSolidMultiRegionPIMPLEControls.H"
                #include "solveSolid.H"
            }
        }

        runTime.write();

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

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

    return 0;
}
```

fluid/solveFluid.H

```
// Solve the Momentum equation
#include "UEqn.H"
// Solve temperature field
#include "TEqn.H"
//PISO Loop
for (int corr=0; corr<nCorr; corr++)
{
  #include "pEqn.H"
}
    turb.correct();

    //Calculate continuity errors for multiregion ↩
       incompressible flow
    {
    volScalarField contErr = fvc::div(phi);
    scalar sumLocalContErr = runTime.deltaT().value()*↩
       mag(contErr)().weightedAverage(mesh.V()).value();
    scalar globalContErr = runTime.deltaT().value()*↩
       contErr.weightedAverage(mesh.V()).value();
    cumulativeContErr[i] += globalContErr;

     Info<< "time step continuity errors : sum local = "↩
         << sumLocalContErr
         << ", global = " << globalContErr
```

```
            << ", cumulative = " << cumulativeContErr[i]
            << endl;
    }
    // Explicitly relax pressure for momentum corrector ↩
       except for last corrector
    if (oCorr != nOuterCorr-1)
    {
      p.relax();
    }

    U -= rUA*fvc::grad(p);
    U.correctBoundaryConditions();
}
```

fluid/UEqn.H

```
// Solve the Momentum equation

tmp<fvVectorMatrix> UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvm::laplacian(nu, U)
  + turb.divDevReff(U)
);

if (oCorr == nOuterCorr-1)
{
    UEqn().relax(1);
}
else
{
    UEqn().relax();
}

volScalarField rUA = 1.0/UEqn().A();

if (momentumPredictor)
{
    if (oCorr == nOuterCorr-1)
    {
        solve(UEqn() == -fvc::grad(p), mesh.solver("↩
           UFinal"));
    }
    else
    {
        solve(UEqn() == -fvc::grad(p));
```

```
    }
}
else
{
    U = rUA*(UEqn().H() - fvc::grad(p));
    U.correctBoundaryConditions();
}
```

fluid/pEqn.H

```
{
    U = rUA*UEqn().H();

    if (nCorr <= 1)
    {
        UEqn.clear();
    }

    phi = (fvc::interpolate(U) & mesh.Sf())
        + fvc::ddtPhiCorr(rUA, U, phi);

    adjustPhi(phi, U, p);

    // Non-orthogonal pressure corrector loop
    for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth↩
       ++)
    {
        // Pressure corrector
        fvScalarMatrix pEqn
        (
            fvm::laplacian(rUA, p) == fvc::div(phi)
        );

        pEqn.setReference(pRefCell, getRefCellValue(p, ↩
           pRefCell));

        if
        (
            oCorr == nOuterCorr-1
         && corr == nCorr-1
         && nonOrth == nNonOrthCorr
        )
        {
            pEqn.solve(mesh.solver("pFinal"));
        }
        else
        {
            pEqn.solve();
```

```
        }

        if (nonOrth == nNonOrthCorr)
        {
            phi -= pEqn.flux();
        }
}
```

fluid/TEqn.H

```
{
    fvScalarMatrix TEqn
    (
        fvm::ddt(T)
      + fvm::div(phi, T)
      - fvm::laplacian(DT, T)
    );

    if (oCorr == nOuterCorr-1)
    {
        TEqn.relax();
        TEqn.solve(mesh.solver("TFinal"));
    }
    else
    {
        TEqn.relax();
        TEqn.solve();
    }
    Info << "fluid region: " << max(T) << endl;
    Info << min(T) << endl;
}
```

solid/solveSolid.H

```
if (finalIter)
{
    mesh.data::add("finalIteration", true);
}
```

```
#include "UEqn.H"

#include "TEqn.H"

// --- PISO loop
for (int corr=0; corr<nCorr; corr++)
{
    #include "pEqn.H"
}

if (finalIter)
{
    mesh.data::remove("finalIteration");
}
```

solid/UEqn.H

```
    // Solve the momentum equation

    fvVectorMatrix UEqn
    (
        fvm::ddt(U)
      + fvm::div(phi, U)
      - fvm::laplacian(nu, U)
      + fvm::SuSp(DC, U)
    );

    UEqn.relax();

    if (momentumPredictor)
    {
        solve
        (
            UEqn
         ==
            fvc::reconstruct
            (
                (
                  - ghf*fvc::snGrad(rhok)
                  - fvc::snGrad(p_rgh)
                )*mesh.magSf()
            ),
            mesh.solver(U.select(finalIter))
        );
    }
```

solid/pEqn.H

```
{
    volScalarField rAU("rAU", 1.0/UEqn.A());
    surfaceScalarField rAUf("(1|A(U))", fvc::interpolate↩
       (rAU));

    U = rAU*UEqn.H();

    phi = (fvc::interpolate(U) & mesh.Sf())
        + fvc::ddtPhiCorr(rAU, U, phi);

    surfaceScalarField buoyancyPhi(rAUf*ghf*fvc::snGrad(↩
       rhok)*mesh.magSf());
    phi -= buoyancyPhi;

    for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth↩
       ++)
    {
        fvScalarMatrix p_rghEqn
        (
            fvm::laplacian(rAUf, p_rgh) == fvc::div(phi)
        );

        p_rghEqn.setReference(pRefCell, getRefCellValue(↩
           p_rgh, pRefCell));

        p_rghEqn.solve(mesh.solver(p_rgh.select( oCorr ↩
           == nOuterCorr-1
                    && corr == nCorr-1 && nonOrth == ↩
                       nNonOrthCorr)));

        if (nonOrth == nNonOrthCorr)
        {
            // Calculate the conservative fluxes
            phi -= p_rghEqn.flux();

            // Explicitly relax pressure for momentum ↩
               corrector
            p_rgh.relax();

            // Correct the momentum source with the ↩
               pressure gradient flux
            // calculated from the relaxed pressure
            U -= rAU*fvc::reconstruct((buoyancyPhi + ↩
               p_rghEqn.flux())/rAUf);
            U.correctBoundaryConditions();
        }
    }
```

```
    #include "solidContinuityErrs.H"

    p = p_rgh + rhok*gh;

    if (p_rgh.needReference())
    {
        p += dimensionedScalar
        (
            "p",
            p.dimensions(),
            pRefValue - getRefCellValue(p, pRefCell)
        );
        p_rgh = p - rhok*gh;
    }
}
```

solid/TEqn.H

```
// Solving the energy equation
{
    volScalarField coeff
    (
        IOobject
        (
            "coeff",
            runTime.timeName(),
            mesh
        ),
        4.0*exp(-pow(4.0*(T-Tmelt)/(Tl-Ts),2))/Foam::↩
           sqrt(pi)/(Tl-Ts)

    );

    fvScalarMatrix TEqn
    (
        cp*fvm::ddt(T)
      + hs*coeff*fvm::ddt(T)
      + (U & (cp*fvc::grad(T)+hs*coeff*fvc::grad(T)))
      - fvm::laplacian(lambda/rho, T)
    );

    TEqn.relax();
    TEqn.solve();

    Info << "solid region: " << max(T) << endl;
    Info << min(T) << endl;
```

```
    alpha = 0.5*Foam::erf(4.0*(T-Tmelt)/(Tl-Ts))+scalar(0.5);

    T.dimensions().reset(dimless);
    rhoS.dimensions().reset(dimless);
    rho.dimensions().reset(dimless);

    rhok = (alpha*(2205.355538389409+T
       *(-29.29618917818346
            +T*(0.21520074533582612+T
               *(-0.0006352103668986839
            +T*6.62139792627e-7))))
            + (1.0-alpha)*rhoS
           )/rho;

    T.dimensions().reset(dimensionSet(0,0,0,1,0,0,0));
    rhoS.dimensions().reset(dimensionSet(1,-3,0,0,0,0,0));
    rho.dimensions().reset(dimensionSet(1,-3,0,0,0,0,0));

    cp = alpha*cpL+(1.0-alpha)*cpS;
    lambda = alpha*lambdaL+(1.0-alpha)*lambdaS;
    nu = alpha*nuL+(1.0-alpha)*nuS;
    DC = DCl*Foam::pow(1.0-alpha,2)/(Foam::pow(alpha,3)+DCs);
}
```