Numerical Simulations of Cavitation in Blood Vessels Using Parallel Front Tracking Method

Ahmed Basil Kottilingal

UCOM

SORBONNE UNIVERSITÉ

THÈSE DE DOCTORAT
SORBONNE UNIVERSITE

École doctorale n° 391: Sciences mécaniques, acoustique, électronique et robotique de Paris

réalisée à

Institut Jean le Rond ∂'Alembert

pour obtenir le grade de

Docteur de Sorbonne Université

présentée par

Ahmed Basil KOTTILINGAL

Sujet de thèse

Numerical Simulations of Cavitation in Blood Vessels Using Parallel Front Tracking Method

Soutenue le 12 Décembre 2023 devant un jury composé de

Dr. Stéphane VINCENT	Rapporteur	Université Gustave Eiffel, Champs-Sur-Marne
Dr. Manolis GAVAISES	Rapporteur	City University, London
Dr. Stéphane POPINET	Président du jury	Sorbonne Université, Paris
Dr. Daniel FUSTER	Invité	Sorbonne Université, Paris
Dr. Ruben SCARDOVELLI	Examinateur	University of Bologna, Bologna
Dr. Taraneh SAYADI	Examinatrice	Sorbonne Université, Paris
Dr. Stéphane ZALESKI	Directeur de thèse	Sorbonne Université, Paris

Abstract

Targeted Drug Delivery, using acoustic cavitation of coated microbubbles (MB), is currently a significant area of research in medicinal biology. Due to the multifaceted nature of the phenomena involved, numerical modeling of targeted drug delivery is a complex task. This thesis primarily focuses on the implementation of immersed boundary methods (along with front tracking methods) to capture compressible multiphase flows and fluid-structure interactions between fluids and thin elastic capillary walls. These models are essential for simulating drug delivery. The immersed boundary method is applied within a cell-based adaptive mesh refinement framework.

Additionally, this thesis delves into addressing the challenges associated with scalable models for adaptive mesh refinement (AMR)-based immersed boundary methods. Partition of the Eulerian Mesh aiming for optimized communication among parallel processors in solvers that use both Lagrangian Surface Mesh and AMR-based Eulerian mesh is challenging because of the different types of communication involved (Lagrangian to Eulerian, Eulerian to Lagrangian, Lagrangian to Lagrangian) and also due to the adaptive nature of the Eulerian Mesh. The thesis proposes a partition of Lagrangian mesh such a way that each processor owns the local vertices of the mesh and maintains a ghost layer of elements and vertices, which ensures all the local vertices have their valence vertices, edges, and elements either in the local partition or the ghost layer. This kind of partitioning is optimal for communication but requires synchronized algorithms for operations involving front regridding and front topology operations.

The compressible multiphase flow solver, implemented using front tracking, extends the Volume-of-Fluid (VoF)-based All-Mach solver developed by Fuster and Popinet 2018, which handles compressibility of both fluids and surface tension effects.

Résumé

Administration ciblée de médicaments, utilisant la cavitation acoustique de microbulles enrobées , constitue actuellement un domaine de recherche important en biologie médicale. En raison de caractère multiforme des phénomènes mis en jeu, modélisation numérique des phénomènes ciblés l'administration de médicaments est une tâche complexe. Cette thèse se concentre principalement sur mise en œuvre de méthodes de frontières immergées (ainsi que de méthodes de suivi de interface) pour capturer les écoulements multiphasiques compressibles et les interactions fluide-structure entre les fluides et les fines parois capillaires élastiques. Ces modèles sont indispensables pour simuler l'administration de médicaments. La méthode des limites immergées est appliquée dans un cadre de raffinement de maillage adaptatif basé sur les cellules.

De plus, cette thèse se penche sur les défis associés avec des modèles évolutifs pour le raffinement adaptatif du maillage (AMR) immergé méthodes de frontière. Partition du maillage eulérien visant une communication optimisée parmi les processeurs parallèles dans les solveurs qui utilisent à la fois le maillage de surface lagrangien et Le maillage eulérien basé sur l'AMR est un défi en raison des différents types de communication impliquée (Lagrangien à Eulérien, Eulérien à Lagrangien, Lagrangien à Lagrangien) et également en raison de la nature adaptative du maillage eulérien. La thèse propose une partition du maillage lagrangien de telle sorte que chaque processeur possède les sommets locaux du maillage et maintient une couche fantôme d'éléments et les sommets, ce qui garantit que tous les sommets locaux ont leurs sommets de valence, bords et éléments soit dans la partition locale, soit dans la couche fantôme. Cette sorte du partitionnement est optimal pour la communication mais nécessite une synchronisation algorithmes pour les opérations impliquant des opérations de remaillage frontal et de topologie frontale.

Le solveur d'écoulement multiphasique compressible, implémenté à l'aide du suivi frontal, étend le Solveur All-Mach basé sur le volume de fluide (VoF) développé par Fuster et Popinet, qui gère la compressibilité des fluides et les effets de tension superficielle.

Acknowledgement

I want to start by acknowledging my thesis advisor, Prof. Stephane Zaleski, to whom I am highly indebted for his guidance, discussions, and ideas. Moreover, I thank him for his academic and emotional support during difficult times, without which I believe I could not have made this.

I want to acknowledge Dr. Stephane Popinet for providing the open platform basilisk.fr/, which has both fascinated and motivated me and given me an opportunity as a platform for my PhD work. I acknowledge Dr. Daniel Fuster, Dr. Michel Versluis, Dr. Gretar Tryggvason, and Dr. Pascal Frey for their input in this thesis.

I would like to sincerely express my acknowledgment to the European Union (EU) for funding the MSCA-ITN (grant agreement number 813766) project Ultrasound Cavitation in sOft Matter (UCOM) and thus allowing an excellent opportunity for my Ph.D.

I thank my colleague and friend, Dr. Mandeep Saini, for all the discussions, suggestions, and support. I also thank Dr. Lijun T Raju and Ali Rezai for hosting me at the University of Twente, Enschede. I am grateful to Yash and Xiangbin for all the discussions and help. I thank all my friends for contributing cherishing moments to my life and providing emotional support.

I express my gratitude to Amalia Petrova, Simona Otarasanu, Evelyne Mignon, Olivier Labbey, Pascal Ray and Patrick Cao for administrative and technical support.

In closing, my heartfelt gratitude extends to my family, whose unwavering support has guided me through my good and bad days.

Contents

List of Symbols and Abbreviations

0.1 Abbreviations

Abbreviation	**Description**
FT	Front Tracking
IB	Immersed Boundary
E-L	Eulerian-Langrangian
N-S	Navier-Stokes
CFL	Courant–Friedrichs–Lewy
SFC	Space Filling Curve

0.2 Symbols

Notation	**Description**
Re	Reynolds' number.
Ma	Mach Number
Ca	Capillary Number
La	Laplace Number
We	Weber Number
P	Pressure ratio used in Sec:5.4.1
E_1 and E_2	Non-dimensional number defined using linear and bending elastic coefficients defined in Section:6.4
$\mathbb{W}$	Set of whole numbers
$\mathbb{N}_m := \{0, 1, .., m-1\}$	Set of whole numbers less than m
$\mathbb{R}$	Real Numbers
$\mathbb{R}_{\geq 0}$	Set of non-negative real numbers
$D \in \{2, 3\}$	Dimension of Euclidian Space
$\mathbb{R}^D$	Euclidian Space
$\{^{\cup}\hat{\mathbf{e}}_d\} := \{\hat{\mathbf{e}}_0, .., \hat{\mathbf{e}}_{D-1}\}$	Orthonormal basis vectors Euclidian Space
$d,\ \hat{\mathbf{e}}_d$	Direction (a basis vector) in $\mathbb{R}^D$. $d \in \mathbb{N}_D$
Ω	Computational Domain. For explanation, we simplified $\Omega = [0, 1]^D \subset \mathbb{R}^D$
$\partial\Omega$	Computational Surface

Ω_1	Space occupied by reference fluid. $\Omega_1 \subseteq \Omega$
Ω_0	Space occupied by non-reference fluid. $\Omega_0 \subseteq \Omega$
$\Gamma(t)$	Interface or membrane $\mathbb{R}^{D-1}$ manifold immersed in Ω.
	For Simplification $\Gamma \subset \Omega \setminus \partial\Omega$
$\boldsymbol{\mathcal{T}}$	AMR Tree (Octree/Quadtree/Bitree)
$\boldsymbol{\mathcal{T}}^+$	Parent Tree
$\overline{\boldsymbol{\mathcal{T}}}$	Saturated Tree
$\boldsymbol{\mathcal{T}}_p^+$	Local (in a processor with rank p) parent tree in MPI computation
$\boldsymbol{\mathcal{M}}$	Front or Surface Mesh
$\boldsymbol{\mathcal{L}}$	Set of leaves of $\boldsymbol{\mathcal{T}}$
$\boldsymbol{\mathcal{P}}$	Set of internal cells of $\boldsymbol{\mathcal{T}}$
$\boldsymbol{\mathcal{C}}$	Set of all cells of $\boldsymbol{\mathcal{T}}$. $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{L}} \cup \boldsymbol{\mathcal{P}}$
$\boldsymbol{\mathcal{H}}$	Set of halo cells of $\boldsymbol{\mathcal{T}}^+$
$\boldsymbol{\mathcal{C}}^+$	Set of all cells of $\boldsymbol{\mathcal{T}}^+$. $\boldsymbol{\mathcal{C}}^+ = \boldsymbol{\mathcal{L}} \cup \boldsymbol{\mathcal{P}} \cup \boldsymbol{\mathcal{H}}$
$\overline{\boldsymbol{\mathcal{C}}}$	Set of all cells of saturated tree. $\overline{\boldsymbol{\mathcal{C}}} \supseteq \boldsymbol{\mathcal{C}}^+ \supseteq \boldsymbol{\mathcal{C}}$
c_0	The root cell of the tree. Can take as $c_0 := (0,0,0,0)$ (in 3D)
$c := (i,j,k,l)$	Represent a cell of the tree
h	Dimension of the cell c. $h = 2^{-l}$
Ω_c	Control volume of the cell c
$\partial\Omega_c$	Control surface of the cell c
$\overline{\Omega}_c$	Ω_c excluding points on right and top (and front) faces.
$\boldsymbol{x}_c$	Cell center. Centroid of the control volume Ω_c
$\mathcal{F}$	Represents a face of the cell c
$\{^{\cup}\mathcal{F}\}_c$	Set of all the faces of the cell c
$\partial_{\mathcal{F}}\Omega_c$	Control surface of the face $\mathcal{F}$. $\partial_{\mathcal{F}}\Omega_c \subset \partial\Omega_c$
$\boldsymbol{x}_{\mathcal{F}}$	Face center. Centroid of the contral face $\partial_{\mathcal{F}}\Omega_c$
$\{^{\cup}n\}_c^+$	All cells in the 5x5(x5) neighborhood of c
$\{^{\cup}n\}_c$	Neighbors of c. $\{^{\cup}n\}_c = \{^{\cup}n\}_c^+ \setminus \{c\}$
$\{^{\cup}\bar{n}\}_c$	Contact leaves of c that shares $\partial\Omega_c$
$\phi(\boldsymbol{x})$	A primary variable in the formulation for an Eulerian point $\boldsymbol{x} \in \Omega$
$\phi^s(\boldsymbol{x}^s)$	A variable in the formulation for an Lagrangian point $\boldsymbol{x} \in \Gamma(t)$
$\{^{\cup}\phi\}$	Set of all the variables in the formulation
$\phi_h(\boldsymbol{x}_c)$ or ϕ_h	Solution (from Discretised equations) of ϕ at cell center
$\{^{\cup}\phi_h\}_c$	Solution set correponding to a leaf cell
$\left\{^{\cup}\{^{\cup}\phi_h\}_c\right\}_{\boldsymbol{\mathcal{T}}}$	Solution set for all the cells
$\boldsymbol{\mathcal{S}}_{\boldsymbol{\mathcal{T}}}^{\phi}$	Scalar ϕ.
	Maps each leaf, $c \in \boldsymbol{\mathcal{L}}$, to the solution $\boldsymbol{\mathcal{S}}_{\boldsymbol{\mathcal{T}}}^{\phi}[c] := \phi_h(\boldsymbol{x}_c)$
z	z-index ($z \in \mathbb{W}$)
$\mathcal{Z}(z) : \mathbb{N}_{\lvert\boldsymbol{\mathcal{L}}\rvert} \mapsto \boldsymbol{\mathcal{L}}$	SFC Curve that maps z-index, $z \in \mathbb{N}_{\lvert\boldsymbol{\mathcal{L}}\rvert}$, to a leaf, $c \in \boldsymbol{\mathcal{L}}$
$\mathcal{Z}^+(z) : \mathbb{N}_{\lvert\boldsymbol{\mathcal{C}}^+\rvert} \mapsto \boldsymbol{\mathcal{C}}^+$	SFC Curve that maps whole number to a cell in the parent tree
$\boldsymbol{\mathcal{S}}_{\boldsymbol{\mathcal{T}}}^{\mathcal{Z}}(c) : \boldsymbol{\mathcal{L}} \mapsto \mathbb{N}_{\lvert\boldsymbol{\mathcal{L}}\rvert}$	Inverse of $\mathcal{Z}$ and maps a leaf cell,$c \in \boldsymbol{\mathcal{L}}$, to its z-index, $z \in \mathbb{N}_{\lvert\boldsymbol{\mathcal{L}}\rvert}$
$\boldsymbol{\mathcal{V}}$	Set of Vertices of $\boldsymbol{\mathcal{M}}$
$\boldsymbol{\mathcal{E}}$	Set of Elements of $\boldsymbol{\mathcal{M}}$
$\boldsymbol{\mathcal{N}}$	Set of neighbor tuples (of elements in $\boldsymbol{\mathcal{E}}$) of $\boldsymbol{\mathcal{M}}$
$\{^{\cup}l\}$	Set of (undirected) edges of $\boldsymbol{\mathcal{M}}$

e		A frontelement (Triangle in 3D or an edge in 2D). $e \in \boldsymbol{\mathcal{E}}$
l		An edge (in 3D)
$\{^{\cup} l\}_e$		Set of (directed) edges of the elements e
v or $\boldsymbol{x}^s$		Frontpoint or Vertex $v, \boldsymbol{x}^s \in \boldsymbol{\mathcal{V}}$.
		In continuous Lagrangian representation $\boldsymbol{x}^s \in \Gamma(t)$.
		In IBM $\boldsymbol{x}^s(r,s,t) : V \subset \mathbb{R}^2 \times \mathbb{R}_{\geq 0} \mapsto \Gamma(t)$ represents the mapping
$o(\boldsymbol{x}^s)$		owner leaf cell of vertex $\boldsymbol{x}^s$. $o(\boldsymbol{x}^s) = c \iff \boldsymbol{x}^s \in \Omega_c$
k		Number of processors
$\{^{\cup} p\} := \mathbb{N}_k$		Set of (ranks of) processors
p		rank or processor id. $p \in \{^{\cup} p\} := \mathbb{N}_k$
$\Pi_{\boldsymbol{\mathcal{L}}}$ $\{\boldsymbol{\mathcal{L}}_0, \boldsymbol{\mathcal{L}}_1, ...\boldsymbol{\mathcal{L}}_{k-1}\}$	:=	Balanced k-way partion of $\boldsymbol{\mathcal{L}}$ with $\lvert\boldsymbol{\mathcal{L}}_p\rvert \leq \lceil \lvert\boldsymbol{\mathcal{L}}\rvert/k \rceil \; \forall \, p \in \mathbb{N}_k$
$\Pi_{\boldsymbol{\mathcal{V}}}$ $\{\boldsymbol{\mathcal{V}}_0, \boldsymbol{\mathcal{V}}_1, ...\boldsymbol{\mathcal{V}}_{k-1}\}$	:=	Partition of $\boldsymbol{\mathcal{V}}$
$\Pi_{\boldsymbol{\mathcal{E}}}$ $\{\boldsymbol{\mathcal{E}}_0, \boldsymbol{\mathcal{E}}_1, ...\boldsymbol{\mathcal{E}}_{k-1}\}$	:=	Partition of $\boldsymbol{\mathcal{E}}$
$\boldsymbol{\mathcal{L}}_p$		Set of leaf cells owned by a processor with rank p.
		$\boldsymbol{\mathcal{L}}_p \subset \boldsymbol{\mathcal{L}}$ and $\lvert\boldsymbol{\mathcal{L}}_p\rvert > 0$
Ω_p		Domain owned by a processor with rank p. $\Omega_p \cap_{c \in \boldsymbol{\mathcal{L}}} \Omega_c$
$\overline{\Omega}_p$		$\overline{\Omega}_p \cap_{c \in \boldsymbol{\mathcal{L}}} \overline{\Omega}_c$
$\boldsymbol{\mathcal{V}}_p$		Set of vertices owned by a processor with rank p. $\boldsymbol{\mathcal{V}}_p \subseteq \boldsymbol{\mathcal{V}}$
$\boldsymbol{\mathcal{E}}_p$		Set of elements owned by a processor with rank p. $\boldsymbol{\mathcal{E}}_p \subseteq \boldsymbol{\mathcal{E}}$
$\boldsymbol{\mathcal{N}}_p$		Set of neighbors (of elements in $\boldsymbol{\mathcal{E}}_p$) owned
		by a processor with rank p
$\boldsymbol{\mathcal{C}}_p$		Set of parent tree cells owned by a processor with rank p.
		$\boldsymbol{\mathcal{L}}_p \subset \boldsymbol{\mathcal{C}}_p$
$\boldsymbol{\mathcal{E}}_p^+$		All local elements in ($e \in \boldsymbol{\mathcal{E}}_p$) and ghost elements.
		Ghost elements. ($e \in \boldsymbol{\mathcal{E}}_p^+ \setminus \boldsymbol{\mathcal{E}}_p$) are non-local
		valence elements of local cells ($e \in \boldsymbol{\mathcal{E}}_p$).
$\boldsymbol{\mathcal{V}}_p^+$		All vertices in $\boldsymbol{\mathcal{V}}_p$ and the non-local vertices of $\boldsymbol{\mathcal{E}}_p^+$
$\boldsymbol{\mathcal{N}}_p^+$		All neighbor tuples of elements in $\boldsymbol{\mathcal{E}}_p^+$
$\boldsymbol{\mathcal{C}}_p^+$		Set of parent tree cells owned by a processor with rank p
		with ghost cells and non-local parents that constitutes local
		parent tree $\boldsymbol{\mathcal{T}}_p^+$
$\{^{\cup} n\}_p =$		Communication neighbors of p.
		$\{^{\cup} n\}_p := \{\mathcal{P}_c(c) \in \mathbb{N}_k \setminus \{p\} \mid \exists c \in \boldsymbol{\mathcal{C}}_p^+\}$
$\mathcal{P}_c(c) : \boldsymbol{\mathcal{C}}^+ \mapsto \mathbb{N}_k$		Maps a cell $c \in \boldsymbol{\mathcal{C}}^+$ in the set $\mathbb{N}_{\lvert\boldsymbol{\mathcal{C}}^+\rvert}$ to the (rank of its) owner processor
$\mathcal{P}_v(v) : \boldsymbol{\mathcal{V}} \mapsto \mathbb{N}_k$		Maps a vertex $v \in \boldsymbol{\mathcal{V}}$ to the (rank of its) owner processor.
		$\mathcal{P}_v(\boldsymbol{x}^s) = p \iff \boldsymbol{x}^s \in \overline{\Omega}_p$
$\mathcal{P}_e(e) : \boldsymbol{\mathcal{E}} \mapsto \mathbb{N}_k$		Maps an element e to the (rank of its) owner processor.
		$\mathcal{P}_e((v_{i_0}, v_{i_1}, v_{i_2})) = p \iff min\{\mathcal{P}_v(v_{i_0}), \mathcal{P}_v(v_{i_2}), \mathcal{P}_v(v_{i_2})\} = p$

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The acoustic response of microbubbles is one of the most researched scientific topics, which has found many applications in science and medicinal biology, including ultrasound diagnosis, lithotripsy (breaking down of the kidney stones), etc. One such noteworthy application is the *Targeted drug delivery* [1] (Fig:1.1) using acoustic cavitation, where the response of coated bubbles can be used to enhance the intake of medicine near the targeted tissues of the body. One such application, as hypothesized, is in the cancer treatment where the medicine is the chemotherapeutic medicine and the target is the cancerous tissues. In this treatment, we use monodisperse microbubbles, which are stabilized by lipid coating, are injected and transported to the target site and, with the help of guided ultrasound microbubbles are notified, and their response is used to increase the intake of medicine by the mechanism called as *sonoporation* (or creating pores using ultrasound) where the porosity of the epithelial walls (of the blood capillaries which feed the cancerous tissues) are increased.

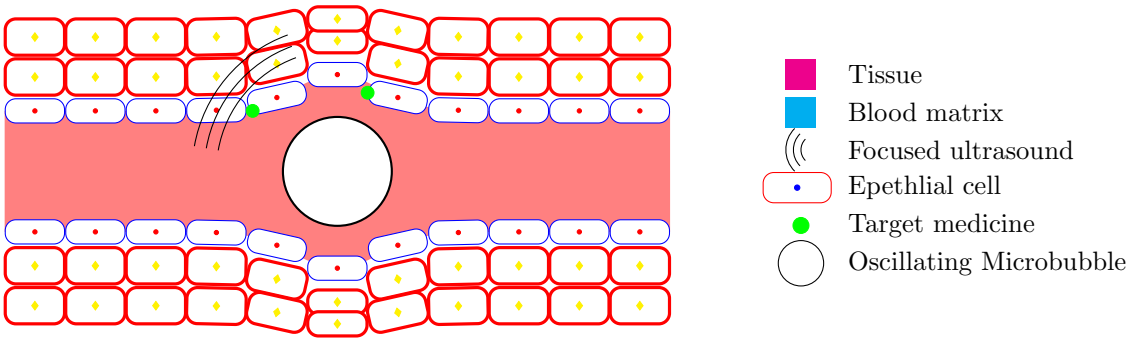


Figure 1.1: The capillary blood vessels that feed the target tissues are fed with target medicine and microbubbles. The microbubbles are insonified with *high-intensity focused ultrasound* (HIFU). The ultrasound cavitating bubble changes the porosity of the one-cell thick epithelial coating, which increases the intake of target medicine.

1.1.1 Numerical Modelling

Numerical modeling of the above-discussed problem is challenging because of the multiphysics nature of the problem (Fig:1.2). The model requires the implementation of compressible multiphase flow and fluid-structure interaction in the problem. The motivation of this thesis is to develop a general platform for the simulation of cavitation of microbubbles inside blood vessels.

The model employs an Eulerian-Lagrangian framework to capture compressible multiphase flow using the front tracking method [2] and interaction of the fluid with the thin microvessel using immersed boundary method [3].

1.2 Literature

Acoustic Cavitation: *Cavitation* is the physical phenomena of formation (and collapse) of a 'cavity' in a varying pressure flow, which in its broader sense includes boiling and pressure cavitation. *Acoustic cavitation* is a general term involving the response of a bubble or cloud of bubbles to an acoustic field. Rayleigh [4] gave the first mathematical analysis for cavitation, where he considered the collapse of a spherical cavity in an infinite liquid.

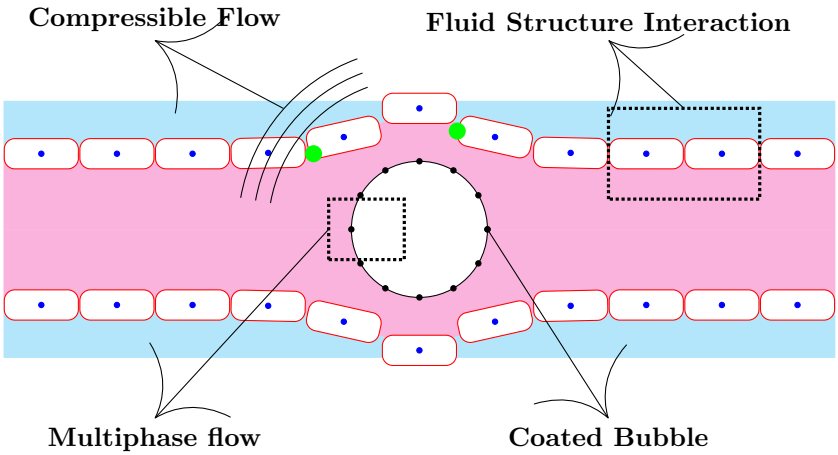


Figure 1.2: The objective of this thesis is to capture multiple physical phenomena involved in acoustic cavitation of microbubbles inside blood vessels.

In *stable acoustic cavitation*, microbubbles oscillate around an equilibrium size when a time-varying (commonly sinusoidal) pressure perturbation is applied [5].High echo amplitude of microbubble *ultrasound contrast agents* make them a useful agent for detecting blood vessels using ultrasound imaging. These microbubbles which have diameter $2\mu m$ to $5\mu m$, unlike RBC and other blood particles have a large contrast of *echogenicity* (ability to reflect ultrasound) compared to the blood plasma, and makes them a prime choice as a contrast agent [6]. Microbubble UCAs are comprised of gaseous bubbles with an outer shell for the stability of the bubble. Their size makes them ideal for transporting through micro-sized blood capillaries[6]. Besides their use as a contrast agent, microbubbles has been investigated for its application in drug and gene delivery [7].

The oscillation of microbubbles increases the transmural pressure through the blood vessel and vessel wall permeability [8].Increasing the amplitude of acoustic pressure increases the bubble's nonlinear response and the blood vessel's circumferential stress [8].

The study of the fragmentation of delivery vehicles (encapsulated bubbles) is important in achieving the localization of delivery wherever intended [8].[]One mechanism is using the violent bubble collapse that ruptures the microvessel and (obvious) permeability to the tissue interstitial space [9]. has shown how the dispersion of RBCs and polymer microspheres ($\approx 500nm$) into the target tissues can be achieved by transient acoustic cavitation [9] [10]. In inertial cavitation of microbubbles, immediate neighbors to the vessel wall are destroyed, and there is diffusion of medicine (some test uses fluorescent molecules -calcein) at mid-distance

from the cavitation collapse center [11] [12]. Identified these two length scales as *killing radius* (approximate radius from the collapse center in which the cells are destroyed) and *blast radius* (the average radius from the collapse center upto which there is reversible permeation of calcine, the target medicine used in the study and beyond which the cells are unaffected [13]. shown that, $\approx 500kHz$ ultrasound applied to a $4.5\mu m$ microbubble at a distance of $\approx 10 - 20\mu m$ away from a mono-layered planar membrane that separates a half-space filled with tissue and undergoes inertial cavitation can produce *sonopore* of daimeter $\approx 20\mu m$.

[14]. have shown that thrombolysis using recombinant tissue plasminogen activator (rt-PA) is more effective for frequencies that correspond to stable cavitation compared to larger frequencies that result in inertial cavitation. [15] Showed a strong correlation between subharmonic emission during stable cavitation in microbubble-enhanced rt-PA thrombolysis

Immersed Boundary Method: The *immersed boundary method* (IBM) is a monolithic approach for the formulation of fluid-structure interaction of thin elastic membranes with fluid. Peskin [16] originally formulated IBM to simulate the flow of incompressible fluid through a channel which is attached to elastic membranes representing the flow of blood through heart valves. IBM uses governing equations in both Eulerian and Lagrangian frameworks which are discretized, respectively on Eulerian and Lagrangian meshes, and the formulation involves interpolation between the meshes using discrete delta function [3].

Multiphase flows: DNS simulation of multiphase flows is primarily classified into interface capturing and interface tracking methods. In the interface capturing method, a discrete Eulerian variable implicitly captures an interface. Two of the most common interface capturing methods are *volume of fluid* (VoF) method and *Level Set* (LS) method.

The VoF method, introduced by Hirt and Nichols [17], uses void fraction (fraction of volume occupied by one of the fluids called as *reference fluid*), and the integration of conservative transport the equation can be formulated to conserve the volume fraction as in the case of geometric VoF ([18]). In geometric VoF, the interface is represented by a locally reconstructed interface, and a better representation of it can be using piecewise linear interface calculation (PLIC) [19], [20], [21]. The advection of void fraction inherently takes care of the topology modifications. It is advantageous as it reduces the computational complexity but appears to be the source of numerous numerical (non-physical) droplets during simulations involving breakups and atomizations. In the level set method [22] [23], the signed distance function is used to capture the interface. LS requires reinitialization to correct the distance function after advection. LS doesn't guarantee mass conservation like VoF, which gave rise to methods like couple LS-VoF (CLSVOF) [24] designed to take advantage of both methods.

The front-tracking method, which falls under the class of interface-tracking multiphase flow solvers explicitly track marker points on the interface. The method, first developed by Unverdi and Tryggvasson [2], is an extension of Peskin's immersed boundary method [3]. The interface is comprised of discrete oriented patches called front elements whose vertices are the marker points. The front-tracking method gives the advantage of a sharp interface representation and an accurate evaluation of surface derivatives, and it avoids automatic topology changes. However, additional programming design is required for the maintenance of surface mesh and routines for front regridding (also called surface remeshing), front topology changes, and intergrid (between Eulerian and Lagrangian grids) communication.

Parallel Front Tracking: Parallelization of Eulerian-Lagrangian methods, like the FT method, is more challenging than an Eulerian method like VOF because of the two meshes involved. So, the algorithm is required to parallelize the routines on the meshes simultaneously

and reduce the communication required between the meshes. Many of the earlier parallel solvers ([25] [26]) were not developed for the scalability of Lagrangian meshes. [27] that employs a heuristic partition algorithm for the partition of both the unstructured AMR grid and the surface mesh achieves scalability for only a few processors. [28] uses two different schemes to parallelize the simulation of bubbly flows in a uniform mesh.

1.3 UCOM-ITN

This thesis is fully funded by the European Union under the MSCA-ITN grant (grant agreement number 813766). This project, named Ultrasound Cavitation in sOft Matter (UCOM), has facilitated 15 Ph.D. theses, including this.

1.4 Thesis Outline

Apart from this introduction chapter, the thesis has chapters on

(i) general implementation of the Eulerian-Lagrangian method,
(ii) parallel strategies on implementation aimed at good scalability,
(iii) balanced implementation of front tracking method,
(iv) compressible multiphase flow solver using front tracking method,
(v) axisymmetric simulation of bubble oscillation inside the blood vessel,
(vi) conclusion of chapters, and
(vii) appendix.

The solver discussed in this thesis is implemented in the open code platform basilisk [29]. Basilisk [29], a PDE solver that uses a cell-based adaptive mesh refinement (AMR) Eulerian grid is highly scalable in parallel computing (Fig:4.10). The code is available in `https://github.com/basilkottilingal/FT2D` and `http://basilisk.fr/sandbox/AhmedBasil/`

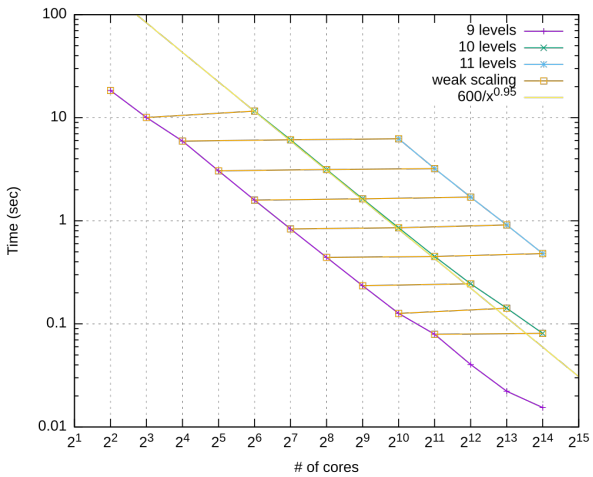


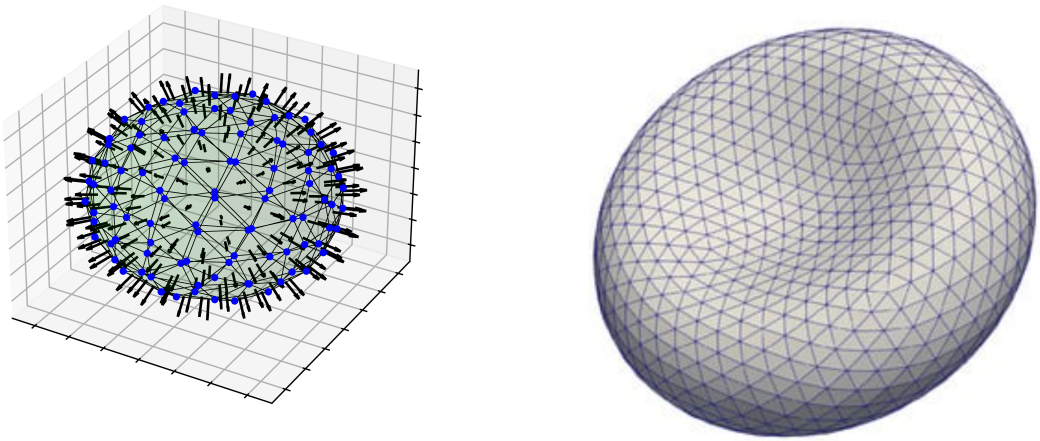
Figure 1.3: Weak Scalability of Eulerian grid (AMR) in basilisk [29].

Chapter 2

Simulation of Fluid Flows with Moving Interfaces and Membranes

2.1 Introduction

Immersed boundary (IB) methods are a general class of continuum solvers [3], like, immersed boundary method (IBM) developed by Peskin to study *fluid structure interaction* (FSI) problems involving flow interaction with thin membranes and *front tracking method* developed by [2] to study multiphase flows, that uses a Lagrangian Mesh representing a thin elastic structure [30], fluid-fluid interface [2], a shock interface [31], etc, which is *immersed* in an Eulerian grid. These kinds of solvers aim to solve a *unified* set of partial differential equations for the entire Eulerian grid with some of these equations involving *source terms* at the interface, for example, surface tension in the momentum equation in the case of front tracking. They come under the general class of Eulerian-Lagrangian methods and might be mentioned in future references.



(a) *(b)*

Figure 2.1: Two types of immersed interfaces: *(a) fluid-fluid interface*: An closed oriented surface composed of small oriented patches that represent a fluid-fluid interface. *(b) thin solid membrane*: The surface of red blood cell is considered as a membrane of zero thickness which is represented here. In both cases, the surface mesh is immersed in an Eulerian grid.

This chapter discusses the implementation of the following computational methods

1. Multiphase flow method using front tracking method

2. Fluid Structure of thin membrane using immersed fibers.

Most of the definitions and implementations explained in this chapter are general for both the solvers (unless specifically mentioned) and aims at the scalability of these methods in parallel computing.

2.2 Representation and Discretisation of Eulerian and Lagrangian Domains

The governing equations of the fluid flow problem are defined in the Eulerian framework for all the points in the computational space Ω and in the Lagrangian framework for all the points on the interface Γ. Ω a finite subset of the Euclidian space $\mathbb{R}^D$ (where dimension $D = 2$ or $D = 3$) and let's represents the orthonormal vectors of $\mathbb{R}^D$ as $\{^{\cup}\hat{\mathbf{e}}_d\}$ where $d \in \mathbb{N}_D$. In this thesis, the computational space is discretized using a cell-based *adpative mesh refinement* (AMR) grid and the solution are sought a discrete set of Eulerian points which are the centroids of grid cells (control volumes in the AMR grid) using the finite volume method. Refer section:2.4 for further details.

A $(D-1)$-dimensional manifold that represents an interface or membrane in $\mathbb{R}^D$ are defined here.

2.2.1 Fluid Structure Interfaction using Immersed Fibers

A computational domain $\Omega \subset \mathbb{R}^D$ is composed of a fluid which is interacting with a thin solid membrane of 0 thickness. The thin structure is represented by a surface, $\Gamma(t) \subset \Omega$, which may not be necessarily closed. The infinitesimally thick membrane $\Gamma(t)$ is a regular surface (at any time $t \geq 0$) in the Euclidian space $\mathbb{R}^3$ with every point of $\Gamma(t)$ has an open neighborhood $U \subset \Gamma(t)$ for which there is an open subset V of $\mathbb{R}^2$ and a time-dependent homomorphism $M_t(r,s) : V \to U$ such that the map M_t is C^∞ smooth and for each point $(r,s) \in V$ and any given time $t \in [0,T]$, the two partial derivatives $\frac{\partial M_t}{\partial r}$ and $\frac{\partial M_t}{\partial s}$ are linearly independent. If we take $V \subset \mathbb{R}^2$ as the set of global parameters (r,s), then for any time $t \in [0,T] \subset \mathbb{R}_{\geq 0}$, the coordinates of each Lagrangian material point $\boldsymbol{x}^s$ of the membrane can be considered as the mapping

$$\boldsymbol{x}^s(r,s,t) : V \times [0,T] \mapsto \Gamma(t) \ , \tag{2.1}$$

which is represented in Fig:2.2.

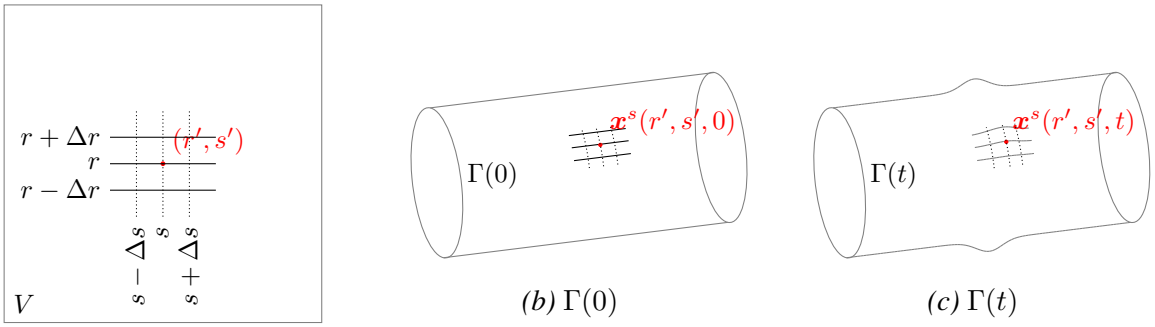
Membrane Energy and Force Density

The elastic energy of the membrane at any time is the functional $E[\boldsymbol{x}^s(\cdot,\cdot,t)]$ which depends on the whole set of points on the membrane and their geometric information. The membrane elastic energy functional E_Γ can be written in the form $E_\Gamma(t) = g(\boldsymbol{x}^s_A, \frac{\partial \boldsymbol{x}^s_A}{\partial r}, \frac{\partial \boldsymbol{x}^s_A}{\partial s}, ..., \boldsymbol{x}^s_B, \frac{\partial \boldsymbol{x}^s_B}{\partial r}, \frac{\partial \boldsymbol{x}^s_B}{\partial s}, ..., \boldsymbol{x}^s_C, ..)$ where $\boldsymbol{x}^s_A := \boldsymbol{x}^s(r_A, s_A, t)$, $\frac{\partial \boldsymbol{x}^s_A}{\partial r} := \frac{\partial \boldsymbol{x}^s}{\partial r}(r_A, s_A, t)$ (and similarly for other derivatives) and $\{\boldsymbol{x}^s_A, \boldsymbol{x}^s_B, ..\} = \Gamma(t)$ is the collection of points on the interface. E_Γ depends on the configuration of the membrane and can be written as the integral of elastic energy density $\mathcal{E}(\boldsymbol{x}^s)$ as

$$E[\boldsymbol{x}^s(\cdot,\cdot,t)] = \int_{\Gamma(t)} \mathcal{E}(\boldsymbol{x}^s) dA = \int_V \overline{\mathcal{E}}(r,s,t) dr ds \tag{2.2}$$

Even though the local energy density depends on the configuration $\Gamma(t)$ usually, it is a function that depends on the local deformation (w.r.t it's initial configuration) and independent of

translation which gives $\mathcal{E}(\boldsymbol{x}^s) = \mathcal{E}\big(\frac{\partial}{\partial r}\boldsymbol{x}^s(r,s,t), \frac{\partial}{\partial s}\boldsymbol{x}^s(r,s,t), \frac{\partial^2}{\partial s \partial r}\boldsymbol{x}^s(r,s,t), .., ..\big)$



(a) V *(b)* $\Gamma(0)$ *(c)* $\Gamma(t)$

Figure 2.2: *(a)* Any Lagrangian membrane point can be uniquely represented by $(r', s') \in V$ and is mapped to it's coordinate *(b)* in the initial configuration, $\Gamma(t=0)$ $\boldsymbol{x}^s(r', s', 0)$ and it's *(c)* configuration $\Gamma(t)$ at an arbitray time $t > 0$. Here, the membrane is a representation of a cylinder, and every point on it is represented by a unique parametric tuple.

If the configuration $\boldsymbol{x}^s(\cdot,\cdot,t)$ is perturbed by $\wp\boldsymbol{x}^s(\cdot,\cdot,t)$ it produces a perturbation in the elastic energy of the membrane by $\wp E_\Gamma$ given by

$$\wp E = \int_V -\overline{\boldsymbol{F}}(r,s,t) \cdot \wp\boldsymbol{x}^s(r,s,t) dr ds = \int_\Gamma -\boldsymbol{F}(\boldsymbol{x}^s) \cdot \wp\boldsymbol{x}^s dA, \tag{2.3}$$

where $\overline{\boldsymbol{F}}(r,s,t)$ is the force density at $(r,s) \in V$ from the virtual perturbation of the configuration and $\boldsymbol{F}(\boldsymbol{x}^s) = \overline{\boldsymbol{F}}(r,s,t)/\|\frac{\partial \boldsymbol{x}^s}{\partial r} \times \frac{\partial \boldsymbol{x}^s}{\partial s}\|$.

$$\overline{\boldsymbol{F}}(r,s,t) dr ds = -\frac{\wp E[\boldsymbol{x}^s(.,.,t)]}{\wp \boldsymbol{x}^s(r,s,t)}. \tag{2.4}$$

Representation of membrane by continuous collection of fibers and energy functional are discussed in chapter:6.

2.2.2 Multiphase flow using Front Tracking

A computational domain $\Omega \subset \mathbb{R}^D$ where $D \in \{2,3\}$ is the dimension of Eulerian space. The computational domain is composed of two immiscible fluids that do not undergo any phase change. The fluids, represented as $fluid-0$ and $fluid-1$ occupies respectively $\Omega_1(t)$ and $\Omega_1(t)$ which are separated by a deforming fluid-fluid interface $\Gamma(t)$. $\Gamma(t)$ is an closed oriented surface in $\mathbb{R}^{D-1}$. $\Gamma(t)$, $\Omega_0(t)$ and $\Omega_1(t)$ satisfies

$$\Omega_0(t) \cup \Omega_1(t) = \Omega \tag{2.5}$$

$$\Omega_1(t) \cap \Omega_1(t) = \Gamma(t) \tag{2.6}$$

Immiscibility and no phase change condition gives the condition $[[\boldsymbol{u}]]_\Gamma = \boldsymbol{0}$ where $[[\boldsymbol{u}]]_\Gamma$ represents the velocity jump difference at the interface.

Similar to the definition of membrane in section Sec:2.2.1, a fluid-fluid interface $\Gamma(t)$ is as a smooth regular surface, which is also closed and oriented. Unlike the immersed membrane method, we do not maintain any global mapping $\boldsymbol{x}^s(r,s,t) : V \times [0,T] \mapsto \Gamma$, as any arbitrary local parametrization $M(r,s) : V \mapsto U \subset \Gamma$ (with $\frac{\partial M}{\partial r}$ and $\frac{\partial M}{\partial s}$ are linearly independent) can be used to for calculating force density $\boldsymbol{F}(\boldsymbol{x}^s)$.

Interface Representation

In front-tracking method, the interface is represented by *marker* points that lies on the interface and the subdomains corresponding to the fluid components can be derived from the interface. Given an oriented surface Γ, you can always obtain Ω_0 and Ω_1 that satisfies Eq:2.5 and Eq:2.6

$$\Omega_0(t^n), \Omega_1(t^n), \{^{\cup} f(\boldsymbol{x}_c)\}^n \xleftarrow{\text{FT/IBM}} \Gamma(t^n)$$

where $\{^{\cup} f(\boldsymbol{x}_c)\}^n$ is the set of values of *void fraction* (the fractional volume occupied by *reference fluid* in a control volume) in each cell of the tree $\boldsymbol{\mathcal{T}}(t^n)$. There are some sections in the thesis that also uses other multiphase flow schemes, like *volume of fluid* (VOF) in 5.2.4, which unlike IB methods, uses *void fraction*, The interface and the fluid domains can be derived from the void fraction

$$\Omega_0(t^n), \Omega_1(t^n), \Gamma(t^n) \xleftarrow{\text{VOF}} \{^{\cup} f(\boldsymbol{x}_c)\}^n$$

Surface Tension

The surface energy density for a fluid-fluid interface with surface tension $\sigma(\boldsymbol{x}^s)$ is simply $\mathcal{E}(\boldsymbol{x}^s) = \sigma(\boldsymbol{x}^s)$ (and $\overline{\mathcal{E}}(r,s) = \sigma(r,s)\|\frac{\partial \boldsymbol{x}^s}{\partial r} \times \frac{\partial \boldsymbol{x}^s}{\partial s}\|$ where (r,s) is an arbitrary parametrisation local to $\boldsymbol{x}^s(r,s)$). We can simplify the force density as

$$\boldsymbol{F}(\boldsymbol{x}^s) := \sigma\kappa(\boldsymbol{x}^s)\hat{\boldsymbol{n}}(\boldsymbol{x}^s) + \kappa(\boldsymbol{x}^s)\boldsymbol{\nabla}_s\sigma(\boldsymbol{x}^s), \tag{2.7}$$

where the last term is zero in the case of constant surface tension. Using Eq:2.7 in momentum conservation equation for an arbitrary control volume containing interface yields the stress jump condition at the interface, given as,

$$[[-p\mathbf{I} + \boldsymbol{\tau}]]_\Gamma \cdot \hat{\boldsymbol{n}} = \sigma\kappa\hat{\boldsymbol{n}} + \kappa\boldsymbol{\nabla}_s\sigma, \tag{2.8}$$

where $\boldsymbol{\tau}$ is the viscous stress. Refer [32] for the details of the derivation of Eq:2.8.

Discretization of the fluid interface is done using a oriented triangular surface mesh and is discussed in detail in Section:2.5. Detailed implementation of the front tracking method is discussed in chapter:4 and chapter:5.

2.3 Governing Equations

Given an interface/membrane of zero thickness immersed in a fluid [33] [3], the governing equation that corresponds to the fluid in the Eulerian frame ($\forall\ (\boldsymbol{x},t) \in \Omega \times [0,T]$) and membrane/interface in the Lagrangian frame ($\forall\ \boldsymbol{x}^s \in \Gamma(t)$) can be collectively written as

$$\boldsymbol{\nabla}\cdot\boldsymbol{u} = 0 \qquad \forall \boldsymbol{x} \in \Omega,\ t \geq 0 \tag{2.9}$$

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{u}\right) = -\boldsymbol{\nabla}p + \mu\boldsymbol{\Delta}\boldsymbol{u} + \boldsymbol{f} \qquad \forall \boldsymbol{x} \in \Omega,\ t \geq 0 \tag{2.10}$$

$$\boldsymbol{f}(\boldsymbol{x},t) = \int_\Gamma \delta(\boldsymbol{x} - \boldsymbol{x}^s)\boldsymbol{F}(\boldsymbol{x}^s)dA \qquad \forall \boldsymbol{x} \in \Omega,\ t \geq 0 \tag{2.11}$$

where the membrane force is given by,

$$\boldsymbol{F}dA = -\frac{\wp E}{\wp \boldsymbol{x}^s} \qquad \forall \boldsymbol{x}^s \in \Gamma(t),\ t \geq 0 \tag{2.12}$$

and the membrane advects,

$$\frac{\partial}{\partial t}\boldsymbol{x}^s = \int_\Omega \boldsymbol{u}(\boldsymbol{x})\delta(\boldsymbol{x}-\boldsymbol{x}^s)dV \qquad \forall \boldsymbol{x}^s \in \Gamma(t),\ t \geq 0 \tag{2.13}$$

where ρ and μ are, respectively, the fluid density and viscosity, $\boldsymbol{F}$ is the membrane/interface force (per unit surface area), $E[\boldsymbol{x}^s(\cdot,\cdot,t)]$ is the elastic potential energy functional or surface energy of the topology set $\Gamma(t)$ which depends on the configuration $\Gamma(t)$ and $\frac{\wp E}{\wp \boldsymbol{x}^s}$ is the Frechet derivative of the functional E, $\boldsymbol{f}(\boldsymbol{x},t)$ is the interface force (per unit volume) imparted by the interface/membrane on the fluid. In the case of the front-tracking method for multiphase flows or in the case of fluid-structure interaction of a closed membrane that separates two distinct fluids (ex, capsule), the fluid properties are averaged properties that are found by a heaviside or indicator functions. In the case of IBM, the force density equation in Eulerian coordinate (Eq:2.11) can also be written as an integral over V (Eq:2.1)

$$\boldsymbol{f}(\boldsymbol{x},t) = \int_V \delta(\boldsymbol{x}(r,s,t)-\boldsymbol{x}^s)\overline{\boldsymbol{F}}(r,s,t)drds \qquad \forall \boldsymbol{x} \in \Omega,\ t \geq 0 \tag{2.14}$$

2.4 Cell-Based AMR Grid

The discretization of Eulerian computational space using a *tree*, is called as tree-based *adaptive mesh refinement* (AMR) in the literature [34] [35]. In computational fluid dynamics, or rather computational continuum mechanics, in general, a hierarchical tree is used to discretize the continuous computational space where we know apriori that solution of at least one of the interested primary variables, say ϕ, has spatial frequencies of different order of magnitudes. A tree-based grid allows to have control volume (associated with *cells*) of different dimensions that allow to capture the above mentioned varying spatial frequencies of the solutions without discretizing the entire computational space to the maximum refinement level. This allows us to optimize the computational load (CPU or GPU load), memory (RAM) requirements and thus, the computational time involved in solving discretized equations. For example, in *direct numerical simulations* (DNS) and *large eddy simulations* (LES) of turbulence flows, we can use AMR grids to capture eddies of length scales starting from the Kolmogorov scale to the largest eddy involved in the problem.

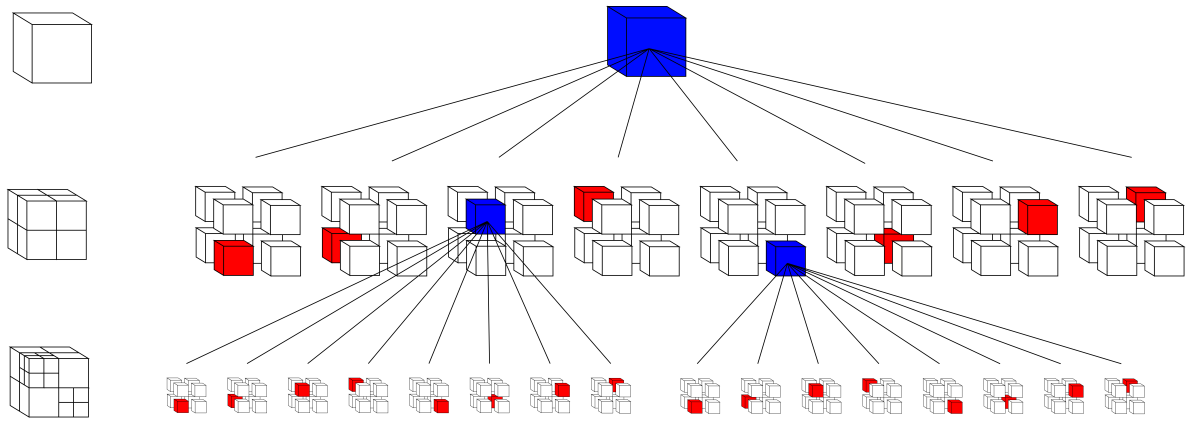


Figure 2.3: *Discretisation of computational domain in* $\mathbb{R}^3$*using AMR grid:* The hierarchical structure of the grid discretization of a 3D computational domain using an octree, $\mathcal{T}$. The leaf-cells and internal -cells are marked in red and blue color respectively.

2.4.1 Tree: Quadtrees and Octrees

A tree , *quadtrees* (for $D = 2$) or *octrees* (for $D = 3$), is a hierarchical graph used to discretize computational domains as in Fig:2.5 and Fig:2.4 which will be mentioned as tree in general. The nodes of the graph are called cells . Every internal node has 2^D children nodes called as children . So every internal node is the (only) parent to 2^{2D} children. A node or a cell of the tree is represented by the index c. Every cell c has an associated control volume $\Omega_c \subseteq \Omega$. Terminal nodes or the cells with no children are called leaves (leaf in singular). The root node has the associated control volume Ω itself, i.e if c is the root node then $\Omega_c = \Omega$ and for every other nodes $\Omega_c \subset \Omega$. The index-set $\{^{\cup}c\} = \mathcal{C} \subseteq \overline{\mathcal{C}}$.

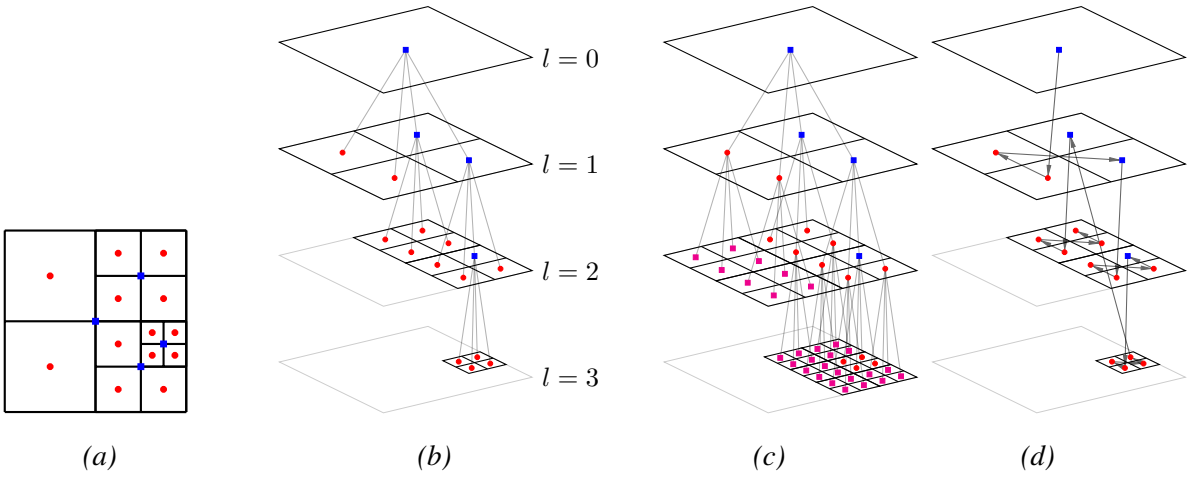


Figure 2.4: *(a)* A tree $\mathcal{T}$ in 2D (quadtree), *(b)* Hierarchical Structure of the $\mathcal{T}$ shows how an internal-cell (in blue) is connected to 4 of its children . *(c)* The parent-tree , $\mathcal{T}^+$ is comprised of leaves $\mathcal{L}$ (●) , internal-cells $\mathcal{P}$ (■) and halo-cells $\mathcal{H}$ (■); *(d)* Traversing through the $\mathcal{T}$ using the Morton curve.

Tree as a Directed Hypergraph

A tree, $\mathcal{T}$, can be defined as a *directed graph* [1] $\mathcal{T} := (\mathcal{C}, \mathcal{E}_{\mathcal{T}})$ where the index-set (or the set of cells), $\mathcal{C}$, is the set of *vertices* of the graph and the set $\mathcal{E}_{\mathcal{T}} := \{^{\cup}(p_c, c)\}$ is the set of *directed edges* of the graph with each edge is an ordered pair of a parent and a child cell .

The index-set comprises of the cells (or indices or nodes) of a tree which can be represented as a tuple, (i, j, l) in 2D or (i, j, k, l) in 3D, where i, j and k are spatial index of the cell corresponding to each dimension of $\mathbb{R}^D$ and l represent the refinement level of cell. Refinement level of a cell written as $\texttt{level}(c)$ is the natural number l, $l \in \mathbb{N}_{\texttt{depth}(\mathcal{T})+1}$ is an indicator of how small the control volume of c is with respect to that of the computational domain. So, the tuplet is used interchangeably with c to identify a cell i.e. $c := (i, j, l)$ or $c := (i, j, k, l)$. The index-set of the tree ($\mathcal{C}$) is a subset of the full-index-set ($\overline{\mathcal{C}}$) which corresponds to a *saturated tree*. [2] If there is $\texttt{maxlevel}$ that satisfies $\texttt{depth}(\mathcal{T}) \leq \texttt{maxlevel}$ (given any moment), then the full-index-set can be written as $\overline{\mathcal{C}} := \{(i, j, k, l) \mid i, j, k \in \mathbb{N}_{2^l} \; \forall \; 0 \leq l \leq \texttt{maxlevel}\}$

Internal Cells and Leaf Cells

An internal-cell is a cell which has 2^D children and it's set, internal-cells , is given by $\mathcal{P} := \{p_c \mid (p_c, c) \in \mathcal{E}_{\mathcal{T}}\}$. A leaf-cell is a cell which does not have any children , and it is set

[1] A hypergraph , $H := (V, E)$, is constructed out of a set of graph vertices/nodes ($v_i \in V$) and a set of connections among the vertices called as hyperedges/nets $e_i \subseteq E \; \forall \; e_i \in E$. The hypergraph is a graph if $|e_i| = 2 \; \forall \; e_i \in E$ and a graph is a directed graph if all the edges are directed (ordered set of two vertices) ($E = \{e_i := (v_{i_0}, v_{i_1})\}$).

[2] A *saturated tree*, $\mathcal{T}_{max}$, is a tree with all its leaf-cells are at the maxlevel set by the user. So a tree $\mathcal{T}$ is saturated if all the leaf-cells , $c = (i, j, k, l) \in \mathcal{L}$ are at the maxlevel , $l = \texttt{maxlevel}(\mathcal{T})$.

can be defined as leaves $\boldsymbol{\mathcal{L}} := \{c \mid (p_c, c) \in \boldsymbol{\mathcal{E}_\mathcal{T}}\} \setminus \boldsymbol{\mathcal{P}}$. The sets of leaves , $\boldsymbol{\mathcal{L}}$, and internal-cells , $\boldsymbol{\mathcal{P}}$, are mutually exclusive and collectively exhaustive partition of $\boldsymbol{\mathcal{C}}$.

Root cell

The root cell, $c_0(\boldsymbol{\mathcal{T}})$ is the only cell with no parent, i.e $\{c_0\} = \boldsymbol{\mathcal{P}} \setminus \{c \mid (p_c, c) \in \boldsymbol{\mathcal{E}_\mathcal{T}}\}$.

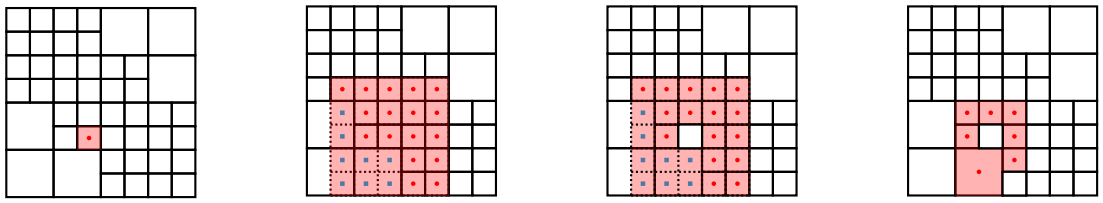
Halo Cells and Parent Tree

In order to do interpolations of variables in a discretized equation, we need cells in the neighborhood which may be neither a leaf-cell nor an internal-cell. For a second-order interpolation in 3D, this neighborhood is a $5 \times 5 \times 5$ grid of cells centered around a leaf-cell. Cells in this neighborhood of a leaf-cell which is neither a internal-cell nor a leaf-cell is called a halo-cell and its set is called as halo-cells ,$\boldsymbol{\mathcal{H}}$. So at any the time step of the simulation, the algorithm maintains a superset of $\boldsymbol{\mathcal{C}}$ called as all-cells which is defined as $\boldsymbol{\mathcal{C}}^+ := \boldsymbol{\mathcal{H}} \cup \boldsymbol{\mathcal{P}} \cup \boldsymbol{\mathcal{L}}$. The corresponding tree is called as the parent-tree, which is the another directed graph $\boldsymbol{\mathcal{T}}^+ := (\boldsymbol{\mathcal{C}}^+, \boldsymbol{\mathcal{E}}^+_{\boldsymbol{\mathcal{T}}})$.

Subsets of 'all-cells' ($\boldsymbol{\mathcal{C}}^+$)

$\boldsymbol{\mathcal{L}}$, the set of leaf-cells , $\boldsymbol{\mathcal{P}}$, the set of internal-cells and $\boldsymbol{\mathcal{H}}$, the set of halo-cells are mutually exclusive and collectively exhaustive subsets of $\boldsymbol{\mathcal{C}}^+$.

The parent of a cell, $c \in \boldsymbol{\mathcal{C}}^+ \setminus \{c_0\}$, is denoted by $p_c(c)$ or $\texttt{parent}(c)$. The set of children or daughters of an internal-cell is denoted by $\{^{\cup} d\}_c$.



(a) $c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})$ *(b)* $\{^{\cup} n\}^+_c$ *(c)* $\{^{\cup} n\}_c$ *(d)* $\{^{\cup} \overline{n}\}_c$

Figure 2.5: *(a)* A cell c in the leaves $\boldsymbol{\mathcal{L}}$ of a quadtree $\boldsymbol{\mathcal{T}}$. *(b)* cell-neighborhood $\{^{\cup} n\}^+_c$; *(c)* neighbors $\{^{\cup} n\}_c$; and *(d)* contact neighbors $\{^{\cup} \overline{n}\}_c$ of the `cell`.

For every leaf-cell ,$c \in \boldsymbol{\mathcal{L}}$, we define $\{^{\cup} n\}^+_c$ as the set of cells in the cell-neighborhood , we define $\{^{\cup} n\}_c$ as the set of neighbors , and $\{^{\cup} \overline{n}\}_c$ as the set of contact-neighbors (leaf-cells that share either a vertex or an edge or a face with c). Refer Fig:2.5 for illustration of these sets.

2.4.2 Control Volumes in an AMR grid

For simplifications, let's take the computational domain as a unit cube in 3D such that

$$\Omega := \left\{ \boldsymbol{x} \in \mathbb{R}^3 \;\middle|\; x_i \in [0,1] \;\forall\; i \in \{0,1,2\} \right\} \tag{2.15}$$

Then the root cell $c = (0,0,0,0)$ has associated volume Ω and for any cell (i,j,k,l) of the tree we have the control volume Ω_c with the its control surface $\partial\Omega_c$ and its interior $int(\Omega_c) = \Omega_c \setminus \partial\Omega_c$. For every cell $c = (i,j,k,l)$ in $\{^{\cup} c\}^+$,

$$\underset{\subset \mathbb{R}^D}{\Omega_c} := \left\{ (x_0, x_1, x_2) \in \mathbb{R}^3 \;\middle|\; x_0 \in \left[\frac{i}{2^l}, \frac{(i+1)}{2^l}\right], x_1 \in \left[\frac{j}{2^l}, \frac{(j+1)}{2^l}\right], x_3 \in \left[\frac{k}{2^l}, \frac{(k+1)}{2^l}\right] \right\} \tag{2.16}$$

$$\underset{\in \mathbb{R}^D}{\boldsymbol{x}_c} := \left(\frac{i+1/2}{2^l}, \frac{j+1/2}{2^l}, \frac{k+1/2}{2^l}\right) \tag{2.17}$$

where $\boldsymbol{x}_c$ is the barycenter of the control volume Ω_c. The averaged are We can see the control volume $\Omega_c \subseteq \Omega$ is another cube of edge dimension $1/2^l$ smaller compared to the computational domain. The maximum refinement level represented by l_{max} corresponding to the smallest control cells is called as the *depth* of the tree . The smallest dimension $1/2^{l_{max}}$ is of the order of smallest length scale that can be captured by a tree of depth l_{max}.

2.4.3 Control Surfaces in an AMR grid

A face ,$\mathcal{F}$, of a cell is the subset of the control surface $\partial\Omega_c$ (which is a face/edge of the cubic/square control volume) and at whose barycenter (face center), $\boldsymbol{x}_{\mathcal{F}}$, we evaluate the average integral of fluxes of conservative quantities and face normal component of velocity. For every leaf-cell the set of control faces is given by $\{^{\cup}\mathcal{F}\}_c$ (with $|\{^{\cup}\mathcal{F}\}_c| = 6$ in 3D or $|\{^{\cup}\mathcal{F}\}_c| = 4$ in 2D) and the set of control faces of the tree is given by $\{^{\cup}\mathcal{F}\}_{\boldsymbol{\mathcal{T}}} := \{\mathcal{F} \in \{^{\cup}\mathcal{F}\}_c \mid c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})\}$.

2.4.4 A Valid Tree

The graph $\boldsymbol{\mathcal{T}}$ of an AMR tree that represents a discretized Eulerian grid if the set of leaf cells of the tree satisfies

1. collectively exhaustive control volume, i.e., $\Omega = \bigcup_{c \in \boldsymbol{\mathcal{L}}} \Omega_c$
2. mutually exclusive interior of the control volume, i.e., for all $c,\ c' \in \boldsymbol{\mathcal{L}}$ $\left(\Omega_c \setminus \partial\Omega_c\right) \bigcap \left(\Omega_{c'} \setminus \partial\Omega_{c'}\right) = \emptyset \iff c \neq c'$. The set $\Omega_c \setminus \partial\Omega_c$ is the set of interior points of the control volume Ω_c. Two distinct leaves may share a face, an edge, or a vertex, or nothing.
3. Apart from the above two general validity conditions of a tree, we also impose *1:2 refinement criteria* by which all the contact-neighbors of a leaf-cell cannot be *too fine* (at a depth of 2 or more than that of the leaf-cell).

$$\forall\ c,\ c' \in \boldsymbol{\mathcal{L}} \text{ with } c \neq c', \qquad \Omega_c \cap \Omega_{c'} \neq \emptyset \implies l - l' \in \{-1, 0, 1\}$$

4. Moreover, from the point of view of computational limits, a user-defined `maxlevel` is also imposed on the tree so that $\texttt{depth}(\boldsymbol{\mathcal{T}}) \leq \texttt{maxlevel}(\boldsymbol{\mathcal{T}})$ at all timestep.

2.4.5 Traverse through Tree

The traversal through the tree, in general, can be achieved using any kind of *space filling curve* (SFC). A SFC, in its analytical sense, is a continuous surjective map f from a compact set $\mathcal{I} \subset \mathbb{R}$ (for simplicity let's take $\mathcal{I} = [0, 1] \subset \mathbb{R}$) to Ω and as you traverse from 0 to 1, the mapping f cover the entirety of the computational space Ω i.e the image of mapping $f(\mathcal{I}) = \Omega$ [36] [37].For a smooth manifold Ω, the mapping cannot be both continuous and bijective (Eugen Netto 1879). In the discrete sense, we can define the SFC as a bijective map from $\mathbb{N}_{|\boldsymbol{\mathcal{L}}|}$ to the leaves of the tree , $\boldsymbol{\mathcal{L}}$.

$$\mathcal{Z} : \mathbb{N}_{|\boldsymbol{\mathcal{L}}|} \mapsto \boldsymbol{\mathcal{L}} \tag{2.18}$$

The inverse of the SFC is a scalar $\boldsymbol{\mathcal{S}}_z$ which maps from a leaf-cell to the *z-curve index* of the leaf-cell . So as you traverse from $z = 0$ to $z = \mathbb{N}_{|\boldsymbol{\mathcal{L}}|}$, the SFC mapping $\mathcal{Z}(z)$ traverses through all the leaf-cells in the tree (and correspondingly all points in the the control volume

is covered). The map $\mathcal{Z}$ can be derived from the SFC for the corresponding saturated tree, $\overline{\mathcal{Z}}$ which is a bijective map from $\mathbb{N}_{|\overline{\mathcal{C}}|}$ to the full-index-set $\overline{\mathcal{C}}$ [3] Hilbert curve and Peano Curve [37] are non-overlapping SFCs while Morton curve, even though doesn't satisfy *locality preserving* condition, is the easiest to implement. Morton curve, which is implemented in Basilisk [29], is discussed in the Appendix:A.2.

(a) *(b)*

Figure 2.6: Z shaped traversing among leaf siblings tree using Morton curve in *(a)* quadtree; and in *(b)* octree.

2.4.6 Cache of Leaves

Since there are many instances of the set of leaf-cells are used in a solver, the Indices of leaf-cells (or their memory pointers) are stored in a linear array which is the cache $\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{L}}}$ (with $\boldsymbol{\mathcal{C}}_{\boldsymbol{\mathcal{L}}}[z] = \mathcal{Z}(z)$ for all $0 \leq z < |\boldsymbol{\mathcal{L}}|$).

2.4.7 Scalars

The discrete value of scalars or vectors corresponding to each leaf-cells of a tree , $\boldsymbol{\mathcal{T}}$ are stored as tree-scalars which is an indexing mapping which maps the index of a leaf-cell to its corresponding value denoted as $\boldsymbol{\mathcal{S}}^{\phi}_{\boldsymbol{\mathcal{T}}}$ which by definition satisfies $\boldsymbol{\mathcal{S}}^{\phi}_{\boldsymbol{\mathcal{T}}}[c] := \phi_h(\boldsymbol{x}_c) \, \forall \, c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})$. This definition can be extended to prolongation cells and restriction cells.

Inter-Level Scalar Interpolation

For the objective of *(a)* calculating scalar values in cell-neighbors which are not leaf-cells (interior/halo cells) ;*(b)* estimating adaptive-wavelet error (Refer section:) ; and *(c)* coarsening/refinement we need to interpolate scalar values from either a parent to its children or vice-verse which is called *inter-level scalar interpolation*. Interpolating from a parent to it is children is called *interpolation* or *prolongation* and interpolating from the set of siblings to their parent is called *restriction*.

[3] The maps $\mathcal{Z}$ and $\overline{\mathcal{Z}}$ satisfies

1. $\overline{\mathcal{Z}} : \mathbb{N}_{|\overline{\mathcal{C}}|} \mapsto \overline{\mathcal{C}}$ is a bijective mapping and there exists an inverse, which is an *saturated* tree-scalar $\overline{\boldsymbol{\mathcal{S}}}^{\mathcal{Z}}_{\boldsymbol{\mathcal{T}}}$.
2. Every internal-cell have z-index lower than that of each of its children. For all cell $c \in \overline{\mathcal{C}} \setminus \{c_0\}$ (except the root) and its parent p_c, we have $\overline{\boldsymbol{\mathcal{S}}}^{z}_{\boldsymbol{\mathcal{T}}}[c] > \overline{\boldsymbol{\mathcal{S}}}^{\mathcal{Z}}_{\boldsymbol{\mathcal{T}}}[p_c]$. The inverse of the map $\mathcal{Z}$ satisfies $\overline{\boldsymbol{\mathcal{S}}}^{z}_{\boldsymbol{\mathcal{T}}}[c] < \overline{\boldsymbol{\mathcal{S}}}^{z}_{\boldsymbol{\mathcal{T}}}[c'] \implies 0 \leq \boldsymbol{\mathcal{S}}^{z}_{\boldsymbol{\mathcal{T}}}[c] < \boldsymbol{\mathcal{S}}^{z}_{\boldsymbol{\mathcal{T}}}[c'] < |\boldsymbol{\mathcal{L}}|$ for distinct leaf-cells c and c'.
3. *locality preserving*: The leaf-cells $\mathcal{Z}(z)$ and $\mathcal{Z}(z+1)$ are contact-neighbors .

2.4.8 Temporal Discretisation

The continuous time-space $[0, T] \subset \mathbb{R}_{0+}$ for which the solution of primary variables is sought, is discretized into a sequence , $(t^n)_{0 \leq n \leq N} = \left(t^0, .., t^N\right)$ with $t^0 = 0$, $t^N = T$ and $0 < \Delta t^n := (t^{n+1} - t^n) \leq \Delta t^n_{max}$. The maximum time step size Δt^n_{max}, satisfies the CFL condition,

$$\Delta t^n_{max} = \frac{1}{2} \underset{c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})}{\operatorname{argmin}} \left(\frac{2^{-l}}{||\boldsymbol{u}||} \right)$$

where $\boldsymbol{u}$ is the cell centered velocity of cell with size l. However, this value might be further reduced by other stability criteria imposed by the schemes used for spatio-temporal discretization of governing equations.

2.4.9 Governing Equation and Solution set

The system is governed by m number of governing *partial differential equations* and the solution of the m *governing variables* or *primary variables* that satisfies those equations is the real-valued set $\{^{\cup}\phi\}\ \forall\ \boldsymbol{x} \in \Omega$ (with $m = |\{^{\cup}\phi\}|$) that represent the unique state of the system at any given time. The objective of a computational solver that uses an AMR grid to find the the solution set for all the leaf cells and for all the discretized time steps. The solution set for these variables for each leaf-cells evaluated at their respective barycenters $\boldsymbol{x}_c$ are represented as $\{^{\cup}\phi_h\}_c$

$$\{^{\cup}\phi_h\}_c := \{\phi_h(\boldsymbol{x}_c) \mid \phi \in \{^{\cup}\phi\}\}$$

The solution set for the all the leaf-cells of the tree $\boldsymbol{\mathcal{T}}$ is written as $\left\{^{\cup}\{^{\cup}\phi_h\}_c\right\}_{\boldsymbol{\mathcal{L}}}$ (whose size is same the number of leaf-cells , $|\boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}}^n)|$, of the tree at the given time step t^n).

Let us say the fluid flow is defined by two set of $|\{^{\cup}\phi\}|$ number of equations where both the set corresponds to fluids occupying $\Omega_0(t)$ and $\Omega_1(t)$ repectively. In a monolithic approach, the discretised solution set $\{^{\cup}\phi_h\}$ is the *averaged* or *unified* solution. The *average* property is usually calculated using a discrete heaviside or color function in the cell. We assume the following information is provided

1. the jump conditions at the interface $[[\ ]]_{\Gamma(t)}$ corresponding to each variables,
2. initial conditions
   (a) the initial subdomain $\Omega_0(0)$ and $\Omega_1(0)$ corresponding to fluid components and the interface $\Gamma(t = 0)$
   (b) initial values of variables $\phi_h(t = 0)\ \forall\ \phi \in \{^{\cup}\phi\}$
3. sufficient boundary conditions on $\partial\Omega$

2.5 Fluid-Fluid Interface: An Oriented Surface

Implementation of a surface mesh that represents a discretized fluid-fluid interface is described in this section where a continuous smooth interface of C^{∞} is approximated by collection of connected triangular patches called a *Front*.

2.5.1 Front: Discretised Interface

In this thesis, the oriented surface is represented by connected triangular patches called *Front* (might also be mentioned as *surface mesh*) which is an extension of hypergraph [4] . The triangles that constitute the surface mesh will be referred to, in the thesis, as *frontelements* or *elements* or simply *triangles* (in 3D) and the marker points, which are also the vertices of the front elements, are called *frontpoints* or *markerpoints* or *vertices*. The triangles are closely knit, maintaining no gap on the surface mesh. The triangles are oriented, or in other words, the vertices are in clockwise order concerning their outward normal for all front elements.

2.5.2 Set of *marker points* or *frontpoints*

Let us define the discrete set of front points that lie on the interface, called as vertices $\boldsymbol{\mathcal{V}}$

$$\boldsymbol{\mathcal{V}} := \left\{ \boldsymbol{x}_i^s \in \mathbb{R}^D \;\middle|\; i \in \mathbb{N}_{|\boldsymbol{\mathcal{V}}|} \right\} \tag{2.19}$$

2.5.3 Set of *frontelements*

The marker points themselves cannot represent an oriented surface; rather need oriented patches (*frontelements* or *elements*) which are an ordered set of marker points. The marker points are also mentioned also as vertices of the element). The orientation (the discrete normal of the element) of the patch is embedded in the order in which the vertices are tuples. In this thesis, the surface mesh is comprised of front elements, which are *simplices* in $\mathbb{R}^D$ (a line segment in 2D and a triangle in 3D). So, an element can be represented by a tuple of vertices , $e_i := (v_{i_0}, v_{i_1})$ in 2D and $e_i := (v_{i_0}, v_{i_1}, v_{i_2})$ in 3D. Let us define the discrete set of triangles, elements , as a set of tuplets of vertices , $\boldsymbol{\mathcal{E}}$,

$$\boldsymbol{\mathcal{E}} := \left\{ e_i = (v_{i_0}, v_{i_1}, v_{i_2}) \in \boldsymbol{\mathcal{V}}^3 \;\middle|\; i \in \mathbb{N}_{|\boldsymbol{\mathcal{E}}|} \right\} \tag{2.20}$$

The set of directed edges of the triangle, element-edges $e_i = (v_{i_0}, v_{i_1}, v_{i_2})$, is the set of ordered tuples

$$\left\{ {}^{\cup}l \right\}_{e_i} := \left\{ (v_{i_0}, v_{i_1}), (v_{i_1}, v_{i_2}), (v_{i_2}, v_{i_0}) \right\} \tag{2.21}$$

The set of edges of the surface mesh is the set edges given by

$$\left\{ {}^{\cup}l \right\} := \left\{ \{v_{i_j}, v_{i_k}\} \;\middle|\; j \neq k \;\middle|\; \forall\ (v_{i_0}, v_{i_1}, v_{i_2}) \in \boldsymbol{\mathcal{E}} \right\} \tag{2.22}$$

Except for the triangles whose edge(s) lie(s) on the domain boundary, every triangle $e_i \in \boldsymbol{\mathcal{E}}$ share its edges with three other distinct triangles $n_{i_0}, n_{i_1}, n_{i_2} \in \boldsymbol{\mathcal{E}}$ which are called the *neighbors* of e_i. Thus, we have a set of tuples of neighbors and their corresponding graph is defined as

$$\boldsymbol{\mathcal{N}} := \left\{ n_i = (e_{i_0}, e_{i_1}, e_{i_2}) \in \boldsymbol{\mathcal{E}}^3 \;\middle|\; i \in \mathbb{N}_{|\boldsymbol{\mathcal{E}}|} \right\} \tag{2.23}$$

[4] A surface mesh can be represented in different ways, like a *hypergraph* $H = (V, E)$ where the set V is the *vertices* of the hypergraph, which in the context of a surface mesh is the set of Lagrangian points on the interface and each *hyperedge/net* of the hypergraph $e_i \in E$ constitutes an elemental surface patch. In the case of the triangular surface mesh each net satisfies $|e_i| = 3$. However, this definition lacks the information on the orientation of each element. If there is some sequence which embeds the orientation, say $(v_{i_0}, v_{i_1}, v_{i_2})_{0 \leq j < 3}\ \forall\, e_i = \{v_{i_0}, v_{i_1}, v_{i_2}\} \in E$, then the hypergraph can sufficiently represents a discretized oriented surface.

2.5.4 Surface Mesh or Front $\mathcal{M}$

Once the sets and graphs corresponding to front points and front elements and are defined we can define the graph $\boldsymbol{\mathcal{M}}$ of the front which embeds the complete information of the surface mesh as a tuple of linked-lists $\boldsymbol{\mathcal{E}}$, $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{N}}$

$$\boldsymbol{\mathcal{M}} := (\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{N}}) \tag{2.24}$$

Even though $\boldsymbol{\mathcal{N}}$ is redundant from the definition of $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{N}}$ can always be derived from $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{E}}$ by looking for shared edges) but we always maintain the $\boldsymbol{\mathcal{N}}$ for computational efficiency [5].

2.5.5 Patch, Mapping

As mentioned earlier, an element e is a simplex with a set of points in the Eulerian space, $P_e \subset \mathbb{R}^D$, and a normal $\boldsymbol{n}_e$. The patch P_e oriented patch has an associated map of a continuous set of points in $\mathbb{R}^D$, which constitutes the interface Γ. Let's define the barycenter $\boldsymbol{x}_e^s$ of the element e ($e := (v_0, v_i)$ or $e := (v_0, v_1, v_2)$) $\boldsymbol{x}_e^s = \frac{1}{D}\sum_{j=0}^{D} \boldsymbol{x}_{i_j}^s$ then the continuous set of points on the element with vertex coordinates $(\boldsymbol{x}_{v_0}^s, \boldsymbol{x}_{v_2}^s, \boldsymbol{x}_{v_2}^s)$, can be written as a function of local parameters $(r, s) \in U := \{(r, s) \mid r, s \in [0, 1] \subset \mathbb{R}_{\geq 0} \text{ with } r + s \leq 1\}$,

$$\underset{\subset \mathbb{R}^3}{P_e} := \left\{ r\left(\boldsymbol{x}_{v_0}^s - \boldsymbol{x}_e^s\right) + s\left(\boldsymbol{x}_{v_1}^s - \boldsymbol{x}_e^s\right) + (1 - r - s)\left(\boldsymbol{x}_{v_2}^s - \boldsymbol{x}_e^s\right) \;\middle|\; (r, s) \in U \right\} \tag{2.25}$$

For every triangle e, we construct a local parametric mapping M_e from $U \in \mathbb{R}^2$ to $V_e \in \mathbb{R}^3$ which constitutes the interface Γ of class $C^k (k \geq 1)$.

$$\Gamma = \bigcup_{e \in \{^{\cup} e\}} \left(\bigcup_{(r,s) \in U} M_e(r, s) \right) \tag{2.26}$$

The corners of a front element are ordered in counter-clockwise with respect to patch normal. Let us define the discrete surface normal at the barycenter of $\boldsymbol{e}$ of the element $e = (v_0, v_1, v_2)$

$$\boldsymbol{n}_e = \frac{\left(\boldsymbol{x}_{v_1}^s - \boldsymbol{x}_{v_0}^s\right) \times \left(\boldsymbol{x}_{v_2}^s - \boldsymbol{x}_{v_0}^s\right)}{\left|\left|\left(\boldsymbol{x}_{v_1}^s - \boldsymbol{x}_{v_0}^s\right) \times \left(\boldsymbol{x}_{v_2}^s - \boldsymbol{x}_{v_0}^s\right)\right|\right|_2} \tag{2.27}$$

2.5.6 A Valid Front

A valid front $\boldsymbol{\mathcal{M}}$ that represent a closed oriented surface satisfies the following

1. Non-empty interior of the patch [6] : For every triangle e, $P_e \setminus \partial P_e \neq \emptyset$

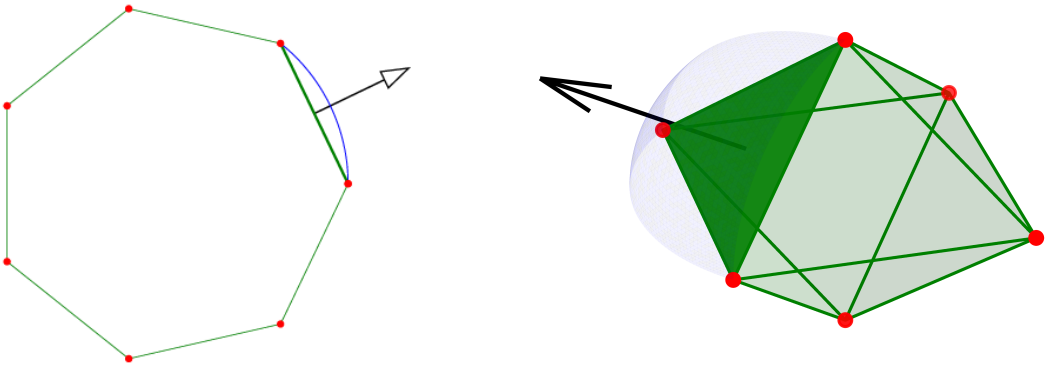
[5] For every i-th triangle $e_i := (v_{i_0}, v_{i_1}, v_{i_2}) \in \boldsymbol{\mathcal{E}}$ and with $(e,) \in \boldsymbol{\mathcal{E}}$ and $(e, (n_0, n_1, n_2)) \in \boldsymbol{\mathcal{N}}$ and for every edge $l_j = (v_j, v_k) \in \{^{\cup} l_e\}_e$ there exits an edge $l_p = (v_p, v_q) \in \{^{\cup} l_e\}_{n_j}$ such that $v_j = v_q$ and $v_k = v_p$ i.e every neighbor n_j shares an edge with e.

[6] Using $\boldsymbol{\mathcal{M}} = (\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{N}})$, we can check for the above validity condition if the following conditions are satisfied by all the triangles on the mesh. So $\forall\ (v_0, v_1, v_2) \in \{^{\cup} e\}(\boldsymbol{\mathcal{M}})$.

(a) Two vertex indices of a triangle cannot be the same: i.e $v_i = v_j \iff i = j$

(b) Non-empty edge interior: i.e $\left|\left|\boldsymbol{x}_{v_i}^s - \boldsymbol{x}_{v_j}^s\right|\right|_2 > 0 \ \forall\ i \neq j$

(c) Non-collinear vertices: i.e $\left|\left|\left(\boldsymbol{x}_{v_1}^s - \boldsymbol{x}_{v_0}^s\right) \times \left(\boldsymbol{x}_{v_2}^s - \boldsymbol{x}_{v_0}^s\right)\right|\right| > 0$ The conditions **(b)** and **(c)** are automatically



(a) *(b)*

Figure 2.7: *Front* or *Surface Mesh*: An oriented surface Γ, is represented by oriented surface patces (green colored). The discrete normal of the patch (black colored quiver) and the mapping from the points on the patch to the surface (colored blue) is mentioned in the figure. *(a)* The front element is a line segment *(b)* The frontelement

2. The interior of no two patches intersects: For any two triangles $e, e' \in \{^{\cup}e\}(\boldsymbol{\mathcal{M}})$,

$$\left(P_e \setminus \partial P_e\right) \bigcap \left(P_{e'} \setminus \partial P_{e'}\right) \neq \emptyset \iff e = e'$$

3. All the edges of the triangle either lie on the computational boundary $\partial\Omega$ or is shared by another triangle. So for every triangle $\forall\, e \in \boldsymbol{\mathcal{E}}$, we have $\partial P_e \setminus \partial\Omega = \bigcup_{e' \in \boldsymbol{\mathcal{E}} \setminus \{e\}} (P_e \cap P_{e'})$. We limit our discussion to problems where the interface Γ doesn't intersect the computational surface $\partial\Omega$, which means we do not consider inlet, outlet BC, contact line problems, etc. ($\Gamma \cap \partial\Omega = \emptyset$
4. Smooth continuous tangent plane [38] (Refer Fig:2.8).

Isotropic Triangles

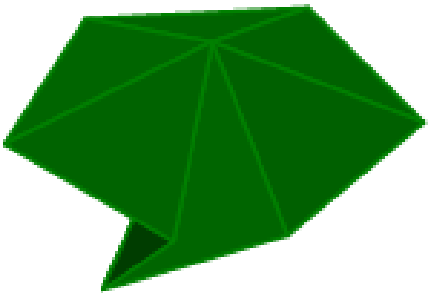


Figure 2.8: An example of non conforming point to tangent plane

Apart from the general validity conditions of a surface mesh, there are certain conditions to be $\boldsymbol{\mathcal{M}}$, we impose the following conditions.

met if the isotropic mesh conditions (section:) are imposed.

1. *isotropic conditions*: Every edge $l = (v_0, v_1) \in \{^{\cup} l\}(\boldsymbol{\mathcal{M}})$ satisfies the condition

$$a_{max}/2 \leq \left|\left|\boldsymbol{x}^s_{v_0} - \boldsymbol{x}^s_{v_1}\right|\right|_2 \leq a_{max}$$

$a_{max}/\sqrt{2}$ is the ideal edge length of the triangles in the mesh, which is usually chosen as a number close to $h_{min}/2$ where h_{min} is the edge length of the maximum refined control volume in the tree .

2. *quality condition*: The shape quality of an element $e \in \boldsymbol{\mathcal{E}}$ satisfies

$$Q_e := \frac{|\partial P_e|^2}{12\sqrt{3}|P_e|} \leq Q_{max}$$

where $|\partial P_e|$ and $|P_e|$ are respectively the perimeter and area of the triangle.

2.5.7 Databases for $\mathcal{M}$ and $\mathcal{T}$

As we have established in the previous sections, both the Grids involved are susceptible to changes, thus the implementation of the graphs $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{\mathcal{M}}$ requires the dynamic insertion and deletion capability.

Refer to Basilisk source code [29] to see the implementation of a cell-based AMR grid. Refer to Appendix (Sec:A.1.1) for the implementation of data structures which is capable of insertion and deletion. This implementation uses integer indices for straightforward implementation in MPI.

2.6 Grid Modification

2.6.1 Tree Modification

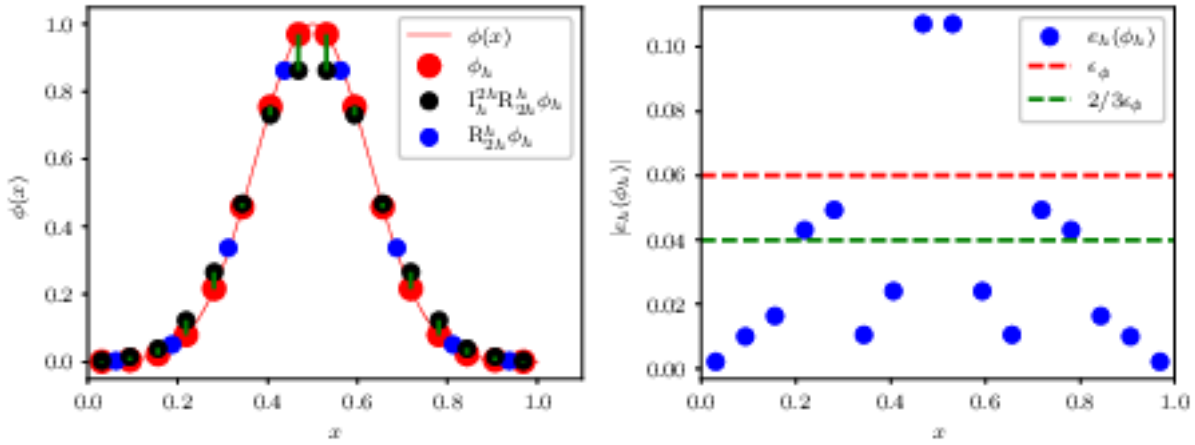
As the fundamental characteristic of an AMR grid, the tree can modify in time by either a cell *refinement* to accommodate high spatial gradients of variables or a cell *coarsening* to optimize (reduce the number of cells) in case of a low spatial gradient. During a refinement a leaf cell of *level*, l, divides to create 2^D children leaf cells of level $l+1$, while during a coarsening operation 2^D children leaves collapses to produce a new leaf cell of level l.

Wavelet-based Adaptive Mesh Refinement

The criteria to determine whether a cell should be refined or coarsened or left untouched is determined by error estimated in the discrete value of a variable by *wavelet functions* and the tolerance allowed for the particular variable. The error calculated by the wavelet algorithm is given

$$e_h(\phi_h) = \phi_h - \mathrm{I}^h_{2h}\left(\mathrm{R}^{2h}_h(\phi_h)\right) \tag{2.28}$$

where R^h_{2h} is the *restriction* operator that *averages* or *downsamples* a discrete value of variable ϕ, ϕ_h, corresponding to a grid level l (with cell size h) to its coarser level $l+1$ (with cell size $2h$). The downsampled value thus produced by averaging or restriction operation,$\phi_{2h} = \mathrm{R}^h_{2h}(\phi_h)$, is *interpolated* or *upsampled* back to level l using an *interpolation* or *prolongation* operator, I^{2h}_h. The absolute value of the difference between the original value and the interpolated value is an estimate of the error. In a finite-volume solver, generally, the restriction operation is an



(a) *(b)*

Figure 2.9: *wavelet based AMR algorithm*: The 1-D computaional domain ,$\Omega = [0, 1] \subset \mathbb{R}$, is discretized by a *bitree* (1D extension of quad/octree). At an instance t^n, the bitree, that represents the computational domain has all its leaf cells at level 4 and a scalar ϕ stored at the center of each of these cells have the discrete value $\phi_h(x_h) = e^{-32(x-0.5)^2}$. The error corresponding to each of the leaf cell $e_h(\phi_h(x_h))$ is is determined by prolongation operation, which preceded by a restriction operation. Figure *(a)* shows the discrete value at the leaf cell center ϕ_h in red, the downsampled values to the higher level $\phi_{2h} = \mathrm{R}_h^{2h}(\phi_h)$ in blue, and the re-interpolated value at the cell center of the leaves,$\mathrm{I}_{2h}^h \mathrm{R}_h^{2h}(\phi_h)$, in black. The error between the initial value and the re-interpolated value is represented in green. [Van Hooft Popinet] [Prouvost] [Basilisk]. Figure *(b)* shows the the absolute value of the error calculated by the wavelet scheme at the center of each cell in blue. When this value is above the user defined tolerance $\epsilon_\phi = 0.6$ (red line) the cells are refined, and when this value is less than $2/3\epsilon_\phi$ the cells are coarsened unless limited by the $1:2$ refinement criteria.

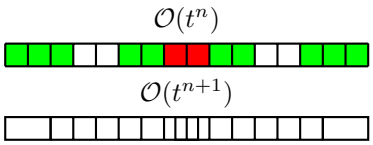


Figure 2.10: *adapt wavelet algorithm*: The grid is used to solve the governing equation in two consecutive time steps are shown here $\boldsymbol{\mathcal{T}}(t^n)$ and $\boldsymbol{\mathcal{T}}(t^{n+1})$. The cells of the grid $\boldsymbol{\mathcal{T}}(t)n$, are looked at whether they are to be refined ($|e_h| > \epsilon_\phi$) colored in red or to be coarsened ($|e_h| < 2/3\epsilon_\phi$). All the red cells will be refined (and their neighbors, if necessary) and all the green cells will be coarsened unless their $1:2$ refinement criteria will be violated.

arithmetic volume averaging of values in children cells to their parent cell, and the interpolation operation is a bilinear (or trilinear) interpolation from a cell to its children. This cannot be a generalized example. For instance, sometimes these operators might be written to capture the specific jump condition associated with the particular variables)

AMR Algorithm

The error was estimated using the wavelet-based algorithm is compared with the user-defined tolerance ϵ_ϕ. So if there is a set of variables $\{^\cup\phi\}_\epsilon \subseteq \{^\cup\phi\}$ for which the tolerance is defined, then refine all the cells with

$$\underset{\phi\in\{^\cup\phi\}_\epsilon}{argmax} \frac{|e_h(\phi_h)|}{\epsilon_\phi} \geq 1 \quad \text{and} \quad l < l_{max}$$

Are refined. Similarly, all the cells with $e_h(\phi_h) < \frac{2}{3}\epsilon_\phi$ are coarsened, provided the neighbors are not *too fine* and refinement of these cells will violate the $1:2$ refinement criteria. Similarly, all the cells with $\epsilon_\phi < e_h(\phi_h) < \frac{2}{3}\epsilon_\phi$ are left unattended unless the refinement of contact neighbors demand its refinement. An example is illustrated in Fig:2.10 and Fig:2.9 where a 1D computational space discretized using an AMR grid is modified based on the error calculated by the adapt wavelet algorithm.

2.6.2 Front Modification

Since the surface grid represents an ever-evolving interface, the marker points that represent *advects* with the local velocity. To satisfy all the *validity conditions* listed above, we need to modify the surface mesh. There are two kinds of surface mesh modifications,

(a) **Regridding** : Because of the flow, the interface moves, which can stretch or compress it locally, resulting in highly skewed surface mesh whose edge sizes are too small or too big compared to Eulerian grid size or triangles with bad aspect ratios. So, to maintain the edge sizes of the grid within a specified interval and a quality surface mesh, we *regrid* the front [39] [32] .

(b) **Topology changes** : When there is coalescence or break up of drops or bubbles, the topology changes, giving rise to more or lesser drops/bubbles [32] [40] [41].

Parallel implementation of the regridding algorithm is described in Section:3.8. This thesis doesn't discuss implementation of topology changes during advection which is left for future work.

2.7 Inter Grid Communication

The grids ($\boldsymbol{\mathcal{T}}$ and $\boldsymbol{\mathcal{M}}$) are required to communicate with each other during

1. Calculating volume average of interfacial force in a control volume

$$\mathbf{f}(\mathbf{x}_c) = \frac{1}{|\Omega_c|}\int_{\Omega_c} dV \int_\Gamma \delta(\boldsymbol{x}^s - \mathbf{x})\mathbf{F}(\boldsymbol{x}^s)dA \approx \sum_{e\in\boldsymbol{\mathcal{E}}(\boldsymbol{\mathcal{M}})} \delta_h(\boldsymbol{x}_e^s - \mathbf{x}_c)\ \mathbf{F}(\boldsymbol{x}_e^s)|P_e| \quad (2.29)$$

   where $\mathbf{f}(\mathbf{x}_c)$ is the volume averaged force calculated at the centroid $\mathbf{x}_c$ of the cell c with control volume Ω_c, $\mathbf{F}(\boldsymbol{x}^s)dA$ is the interfacial force on an infinitesimal patch dA, $\boldsymbol{\mathcal{E}}(\boldsymbol{\mathcal{M}})$ is the linked-list of elements on the surface mesh, $\boldsymbol{x}_e^s$ is the centroid of the element patch P_e, $\mathbf{F}(\boldsymbol{x}_e^s)\ |P_e|$ is the force calculated on the patch.

2. Calculating velocity at the marker points to advect, calculating surface velocity gradient, etc

$$\mathbf{u}^s(\boldsymbol{x}^s) = \int \mathbf{u}(\mathbf{x})\delta(\boldsymbol{x}^s - \mathbf{x})dV \approx \sum_{c\in\boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})} \delta_h(\boldsymbol{x}_e^s - \mathbf{x}_c)\ \mathbf{u}(\mathbf{x}_c)\ |\Omega_c| \quad (2.30)$$

   where $\mathbf{u}(\boldsymbol{x}^s)$ is the velocity of the point $\boldsymbol{x}^s$ on the interface, $\boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})$ is the set of leaf-cells of the Eulerian Mesh, $\mathbf{u}(\mathbf{x})$ the cell-centered velocity.

In both the above equations $|\Omega_c| = h^D$ is the volume of the control volume, $|P_e|$ is the area of the triangular patch, and the discrete δ_h function satisfies

$$\sum_{c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})} h^D \, \delta_h(\mathbf{x} - \mathbf{x}_c) = 1 \tag{2.31}$$

$$\sum_{c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})} h^D \, (\mathbf{x} - \mathbf{x}_c) \, \delta_h(\mathbf{x} - \mathbf{x}_c) = 0 \tag{2.32}$$

for all $\mathbf{x} \in \mathbb{R}^D$. In a uniform grid, with cell size h, a function that satisfy the above conditions can be given by [Peskin 2003]

$$\delta_h(\mathbf{x}) = \frac{1}{h^D} \prod_{d \in \mathbb{N}_D} \phi_{\boldsymbol{\mathcal{TM}}}(\frac{x_d}{h}) \tag{2.33}$$

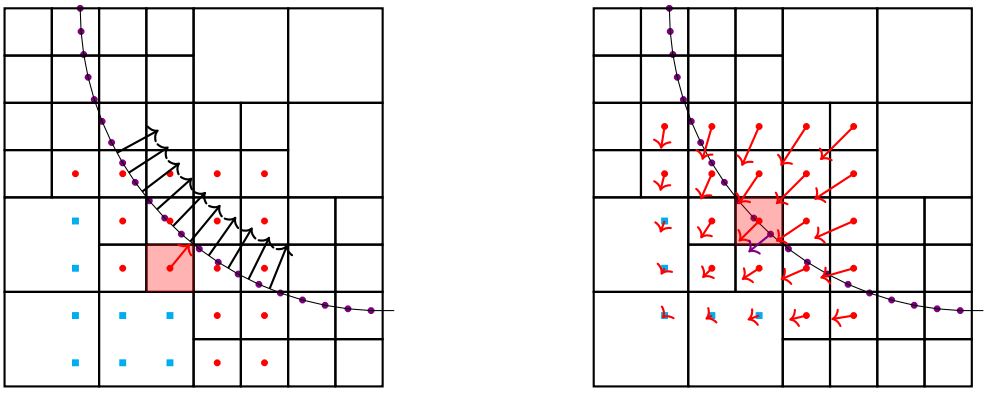
where

$$\phi_{\boldsymbol{\mathcal{TM}}}(r) = \begin{cases} 0 & |r| \geq 2 \\ \frac{1}{8}\left(5 - 2|r| - \sqrt{-7 - 12|r| - 4r^2}\right) & 1 \leq |r| < 2 \\ \frac{1}{8}\left(3 - 2|r| - \sqrt{1 + 4|r| - 4r^2}\right) & |r| < 1 \end{cases} \tag{2.34}$$

and the following equation is a good approximation for 2.35

$$\phi_{\boldsymbol{\mathcal{TM}}}(r) = \begin{cases} 0 & |r| \geq 2 \\ \frac{1}{4}\left(1 + cos\frac{\pi r}{2}\right) & |r| < 2 \end{cases} \tag{2.35}$$

In both the interpolation (Eq:2.29 and Eq:2.30) the δ_h is zero outside the $5 \times 5 (\times 5)$ neighborhood of $\mathbf{x}_c$ and $\boldsymbol{x}^s$ respectively where $|r| > 2$ in the Eq:2.35.



(a) *(b)*

Figure 2.11: *(a)*: Calculation of volume average interface forces can be calculated as a summation of the elements in the neighborhood weighted by δ_h. *(b)*: Velocity at the marker point location can be interpolated from the grid points in the neighborhood

2.7.1 Eulerian mesh to Lagrangian mesh interpolation

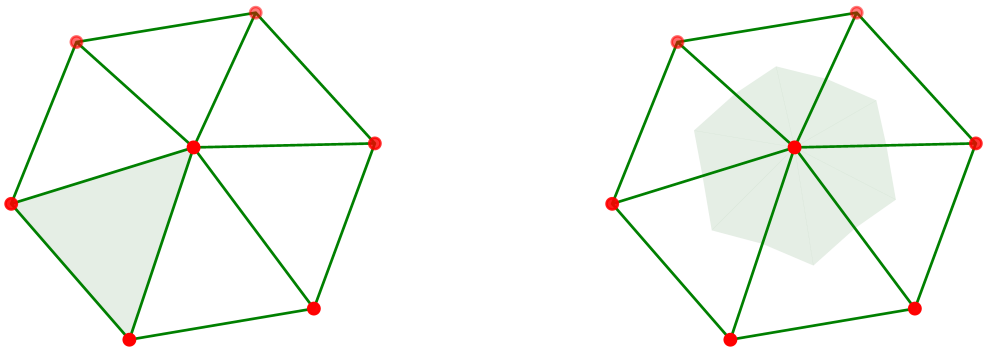
In order to interpolate a variable, say ϕ at a Lagrangian point, $v := \boldsymbol{x}^s$, on the surface mesh, $\boldsymbol{\mathcal{M}}$ from the Eulerian mesh using,

$$\phi^s(v = \boldsymbol{x}^s) = \sum_{c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})} \phi(\mathbf{x}_c)\delta_h(\mathbf{x}_c - \boldsymbol{x}^s)\,|\Omega_c| \tag{2.36}$$

$$= \sum_{c \in \{^{\cup}n\}_v^+} h^D \phi(\mathbf{x}_c)\delta_h(\mathbf{x}_c - \boldsymbol{x}^s) \tag{2.37}$$

Instead of summing over all leaves , we can limit to the cells in the vertex-neighborhood ($\{^{\cup}n\}_v^+$).

2.7.2 Lagrangian Mesh to Eulerian Mesh Interpolation



(a) *(b)*

Figure 2.12: Types of patches on the surface mesh. *(a)*: *elemental* patches. *(b)*: *vertex centered* patches.

We calculate volume averaged interfacial force term by interpolating the interfacial stress calculated on the Lagrangian mesh $\boldsymbol{\mathcal{M}}$ to the leaf cells of the Eulerian mesh $\boldsymbol{\mathcal{O}}$. The interface force per unit volume can be calculated by either summing over *elemental patches* or *vertex centered patches*

$$\mathbf{f}(\mathbf{x}_c) = \begin{cases} \sum\limits_{e \in \boldsymbol{\mathcal{E}}} \delta_h(\boldsymbol{x}_e^s - \mathbf{x}_c)\,\mathbf{F}(\boldsymbol{x}_e^s)\,|P_e| \\ \sum\limits_{v \in \boldsymbol{\mathcal{V}}} \delta_h(\boldsymbol{x}_v^s - \mathbf{x}_c)\,\mathbf{F}(\boldsymbol{x}_v^s)\,|P_v| \end{cases} \tag{2.38}$$

where $\mathbf{F}(\boldsymbol{x}_e^s)\,|P_e|$ and $\mathbf{F}(\boldsymbol{x}_v^s)\,|P_v|$ are the interfacial forces calculated on *elemental patch* as in Fig:2.12*(a)* and *vertex centered patch* as in Fig:2.12*(b)* respectively and the summations are respectively carried out on all the front elements ($\boldsymbol{\mathcal{E}}$) and on all the frontpoints ($\boldsymbol{\mathcal{V}}$) of the surface mesh,$\boldsymbol{\mathcal{M}}$.

As a direct inference (shown in Fig:2.11), the summation in Eq:2.29 can be reduced to the elements in the neighborhood of the leaf cell c. Similarly, the summation in Eq:2.30 can be reduced to the cells in the neighborhood of the vertex $\boldsymbol{x}^s$. The summation can be restricted just

to the neighborhood as

$$\mathbf{f}(\mathbf{x}_c) = \begin{cases} \sum\limits_{c' \in \{^{\cup}n\}_c^+} \sum\limits_{e \in \{^{\cup}e\}_{c'}} \delta_h(\boldsymbol{x}_e^s - \mathbf{x}_c)\, \mathbf{F}(\boldsymbol{x}_e^s)\, |P_e| \\ \sum\limits_{c' \in \{^{\cup}n\}_c^+} \sum\limits_{v \in \{^{\cup}v\}_{c'}} \delta_h(\boldsymbol{x}_v^s - \mathbf{x}_c)\, \mathbf{F}(\boldsymbol{x}_v^s)\, |P_v| \end{cases} \tag{2.39}$$

where $\{^{\cup}n\}_c^+$ is the set of cells in the neighborhood of the cell c, $\{^{\cup}e\}_{c'}$ and $\{^{\cup}v\}_{c'}$ are respectively the set of elements and vertices in the cell c'.

Chapter 3

Parallel Strategies

3.1 Parallel Strategies

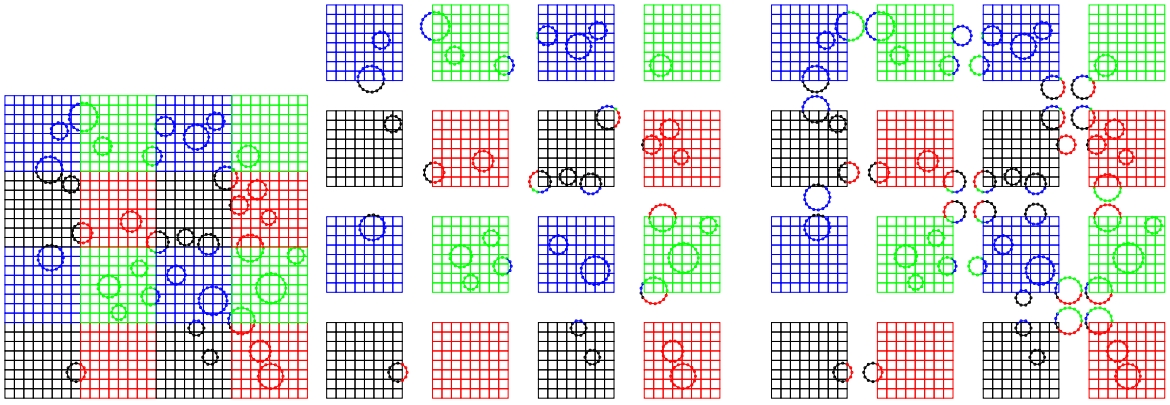
This section discusses the implementation of AMR-based immersed boundary methods in an *MPI* parallel architecture. MPI parallelization is based on *"single program, multiple data"* (SPMD) paradigm where we distribute the computational load among different processors, such a way that all the parallel processors execute the same set of routines/functions but on mutually exclusive and exhaustive partitions/blocks of the data sets which are stored in individually allocated memory partitions (*distributed memory*) of the RAM. In general, a processor operating on a local block requires the updated values in the non-local block, which are communicated between the processors using *blocking/non-blocking* MPI communications.

This chapter discusses implementing a scalable development of an architecture for a multiphase flow solver using the FT method. Most parallel front-tracking solvers available are optimized for specific multiphase flow problems like bubbly flows or do not put effort into scalability for a more significant number of nodes.

3.2 Literature Review

Scalable Eulerian-based multiphase flow simulations, like VOF and level-set methods, are already a well-established and deeply pondered subject of research. The parallelization of such methods is dependent on the type of mesh used for spatial discretization. In the context of AMR grid the parallelizability depends on computational and memory cost associated with the AMR mesh partition/repartition (ex: SFC-based methods like the Morton curve, heuristic graph partitioning schemes) and the communication cost. Eulerian-Lagrangian methods, like the FT method, has further complexity associated with the surface mesh is involved, and inter-grid communication is required. The scalability of the FT method can be simplified if we eliminate the connectivity of the surface elements and associate a marker point with an edge of the AMR mesh ([42], [43], [44]) These methods decrease the complexity associated with topology modifications but imposes the condition that all marker points should be on an edge/face of the grid, which removes the freedom associated with a marker point to be anywhere in Ω and requires additional algorithms (or databases) to retrieve the orientation and connectivity of surface elements.

Early schemes of parallelized computation of Front Tracking were not developed with the intention of scalability. One such scheme used a *master* processor exclusively assigned



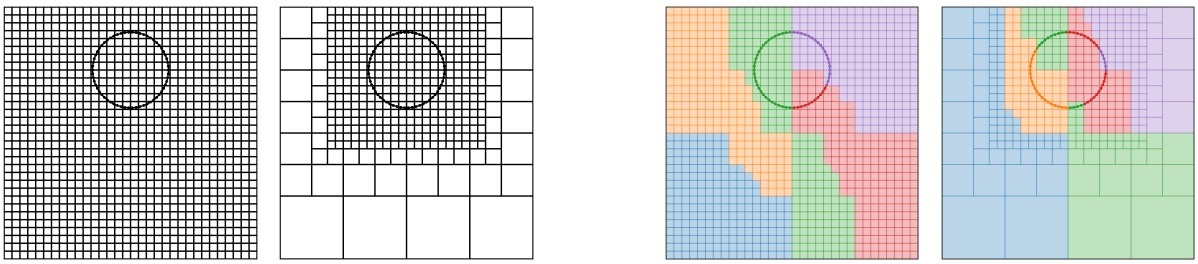
(a) (b) (c)

Figure 3.1: *(a)* parallel simulation of bubbly flows in a 2D structured mesh. *(b)* The processor whose domain contains the centroid of the bubble, takes the ownership of (Lagrangian routines) the bubble. *(c)* All the processors whose domain intersect with the bubble, take co-ownership of the bubble, and all the routines are redundantly done in all the processors.

with all the surface-related routines and so that it had to communicate with (from and to) all the other processors ([25] [26]) . These schemes only focused on the scalability of the Eulerian grid computation and lacks objectivity and discussions of parallelized schemes that are generally associated with an Eulerian-Lagrangian solvers. [45] [46] uses *mirror domain technique* where there is a copy of the whole of Lagrangian mesh in all the processor and each processor operates only on a subset of the dataset which is followed by communication between the fellow processors so that each non-local vertices and elements are up-to-date. This scheme does not have an actual partition of the Lagrangian mesh, and there is a requirement of *broadcast* communication (communicate to all the processors) from each processor which makes it unsuitable for the large surface mesh. A partition of Lagrangian mesh can be seen in [27], [47], [28]. [28] uses a triangulated mesh immersed in an AMR grid where both the AMR grid and Front grid are partitioned using a heuristic graph partitioning algorithm, *parMETIS*. The work is done for a small number of drops/bubbles in up to 196 processes Moreover, it could only achieve modest scaling. [28] uses partition Lagrangian mesh for simulating a large number of bubbles in a uniform mesh. In [28], each bubble is distributed among processors in such a way that (i) Either the processor owns (the routines associated with) a bubble whose centroid falls in its domain and all the processors whose domain intersect with the interface of this bubble are updated, or (ii) all the processors whose domain intersects with the interface of a particular bubble co-own the bubble and redundantly do all the Lagrangian routines are associated with that particular bubble/drop, so that the communications can be avoided (Refer to Fig:3.1). In both cases, as each bubble advects, the owner (or co-owners) and the list of processors whose domains intersect the bubble interface has to be updated. This method is suitable for tiny circular bubbles/drops in a uniform grid so that the number of processors whose domain intersects the interface is limited and can be easily predicted [1].

Creating local groups of processors that co-own each bubble as in []is not suitable for AMR grids as the domain associated with each processor changes dynamically. Morton curve-based graph partition of AMR grid can result in a discontinuous processor domain, and sharing a bubble among all the processors whose domain intersects with the bubble interface is non-

[1]locating the processor whose domain contains a point is straightforward in a uniform grid because processor domains are fixed, and the cells are of the same dimension.

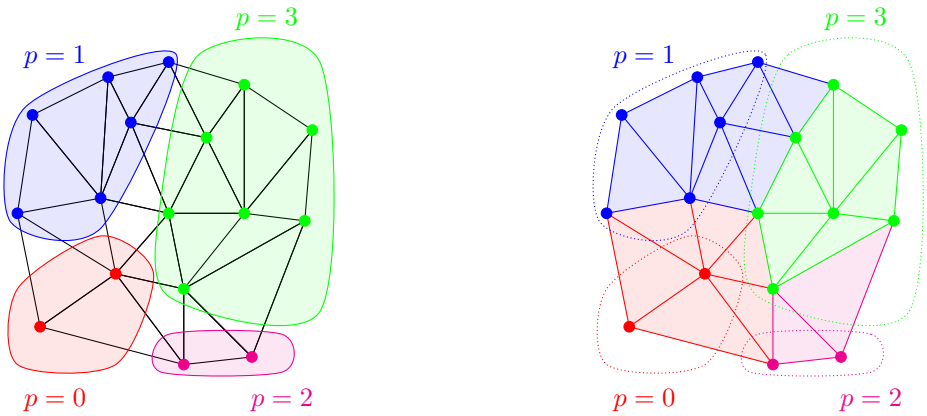


(a) (b)

Figure 3.2: **The challenge in parallelizing Front Tracking in an adaptive mesh refinement environment**: Due to the load balancing of an AMR grid after mesh adaptation, the processor owner of cells in the local neighborhood of front points and front elements may change dramatically.

practical as this list can change dramatically after every advection time step or every adapt wavelet. This kind of partition is not suitable for large interfaces per topology.

In this thesis, we partition the surface mesh in such a way that each processor owns the marker points that lie inside the processor domain so that the marker points and leaf cells are local to each other are in the same processor (Refer fig:) .



(a) (b)

Figure 3.3: Illustration of distributed front where the the domain is distributed among four processors (colored separately): *(a)* The vertices are owned ($\mathcal{P}_v(v)$) by the respective processors whose domain contain the vertex coordinate. *(b)* Ownership of edges and triangles are in such a way that the processor of the least rank that owns the vertices of edges and triangles. (The owner of edges, $\mathcal{P}_l(l)$, and triangles, $\mathcal{P}_e(e)$, are colored in accordance with rank of processors)

3.3 Definitions used in this chapter

3.3.1 Partition of Weighted Graphs for Parallel Computing

Definition of terms related to a weighted hypergraph in the context of parallel computing.

Graph: A *graph* $G := (V, E)$ is composed wth $|V|$ **vertices** ($v_i \in V$) and $|E|$ *edges* such that each edge $e_i \in E$ (say $e_i = \{v_{i_0}, v_{i_1}\}$) is a set of two vertices $v_{i_0}, v_{i_1} \in V$.

Hypergraph: A *hypergraph* $H := (V, E)$ is a more general graph (G) with every edge

$e_i \in E$ (called as *hyperedge/net*) satisfies $e_i \subseteq V$ such that $|e_i|$ is not restricted to 2 as in the case of graph.

Directed hypergraph: A *directed hypergraph* $H := (V, E)$ is defined in such a way that every edge $e \in E$ (called as *directed hyperedge/net*) is an ordered pair of two sets of vertices $e_i := (V_{i_0}, V_{i_1})$ that satisfies $V_{i_0}, V_{i_1} \subseteq V$.

Weighted Hypergraph: A *weighted hypergraph* $H := (V, E, c, w)$ is a hypergraph with a cost defined for each vertex $c : V \mapsto \mathbb{R}_{>0}$ and edge $w : E \mapsto \mathbb{R}_{>0}$.

k-way Partition of hypergraph: A *k-way partition* of the hypergraph $H := (V, E, c, w)$ into k subsets of vertices is $\Pi_V := \{V_0, V_1, ...V_{k-1}\}$ (with $\cup_{p \in \mathbb{N}_k} V_p = V$, $V_{p'} \cap V_p = \emptyset \; \forall \; 0 \leq p' < p < k$ and $|V_p| > 0 \; \forall \; 0 \leq p < k$).

Balanced Partition: The partition $\Pi_V := \{V_0, V_1, ...V_{k-1}\}$ is *ϵ-balanced* (for a given $\epsilon \geq 0$) if the cost of all blocks satisfies $c(V_p) \leq L_{max} := (1 + \epsilon)\lceil c(V)/k \rceil$ where the total cost of computation and cost of computation of p-th partition are given by $c(V) = \sum_{v \in V} c(v)$ and $c(V_p) = \sum_{v \in V_p} c(v)$ respectively.

k-way Partition of edges hypergraph: A *k-way partition* of the the edges of hypergraph $H := (V, E, c, w)$ into k subsets is the partition $\Pi_E := \{E_0, E_1, ...E_{k-1}\}$ (with $\cup_{p \in \mathbb{N}_k} E_p = E$ and $E_{p'} \cap E_p = \emptyset \; \forall \; 0 \leq p' < p < k$ and $|E_p| > 0 \; \forall \; 0 \leq p < k$).

Overload: When the tree goes adaptive mesh refinement, then the corresponding hypergraph $H = (V, E, c, w)$ may change to $H' = (V', E', c', w')$ then the processors may go *overload*, i.e, $c(V'_p) > L'_{max} := (1 + \epsilon)\lceil c(V')/k \rceil$ for some $p \in \mathbb{N}_k$.

Repartition: *repartition* is simply partitioning whenever the hypergraph modifies (also called as *rebalancing* or *dynamic load balancing*).

Edge-cut: Given partitions Π_V, a *cut-edge* is any pair of vertex $\{v, v'\}$ with both of them belonging to different blocks ($v \in V_p$ and $v' \in V_{p'}$ with $p' \neq p$) given $\{v, v'\} \subseteq e$ for some $e \in E$. The set of all cut-edges is the set *edge-cut* which can be defined as $\texttt{cut}(\Pi_V) := \big\{\{v, v'\} \;\big|\; \exists e \in E$ such that $v, v' \in e$ and $v \in V_p, v' \in V_{p'}$ with $p \neq p'\big\}$.

Communication neighbors: The k-way partition intended with MPI parallel processing requires MPI communication between all the pair of processors $\{p, p'\}$ (given $p \neq p'$) such that there is a cut edge, which is shared among p and p'. So, for a local processor (say with rank p), the set of all processors with which it has to communicate (send or receive) is called *communication neighbors* which can be given by $\{^{\cup}n\}_p := \{\mathcal{P}_V(v') \mid \exists \{v, v'\} \in \texttt{cut}(\Pi_V)$ with $\mathcal{P}_V(v) = p\}$ where $\mathcal{P}_V(v)$ is the processor rank of vertex v.

Boundary-vertices: *Boundary-vertices*, $\texttt{boundary}(\Pi_V) := \big\{v, v' \mid \{v, v'\} \in \texttt{cut}(\Pi_V)\big\}$, are the endpoints of cut-edges.

MPI Computation: In MPI computation every processor with rank p possess the ownership (on the computation) of $V_p \subseteq V$ and $E_p \subseteq E$ and maintains an extended partition of vertices (V_p^+) and nets (E_p^+). The portions $V_p^+ \setminus V_p$ and $E_p^+ \setminus E_p$ is called *ghost layer* which helps to maintain the connectivity of the graph H. The sets $\texttt{boundary}(\Pi_V) \bigcap (V_p^+ \setminus V_p)$ and $\texttt{boundary}(\Pi_V) \bigcap (V_p)$ corresponds to the vertices received and sent during and MPI communication.

MPI communication optimisation: The objective of MPI programmes that works on partitions Π_V and Π_E can be minimisation of the total cost associated with edge-cut or boundary vertices. For simplifaction of discussion, if we take the cost associated with every vertex in boundary-vertices as 1, $c(v) = 1 \; \forall \; v \in V'$, and every edge in edge-cut as 1, $w(e) = 1 \; \forall \; e \in E'$, we can assume the objective of the MPI computation is to simultaneously minimize the size of the (local) edge-cut, $\big|\big\{\{v_p, v_{p'}\} \in \texttt{cut}(\Pi_V) \mid v_p \in V_p\big\}\big|$ or (local) boundary-vertices,

$\left|\texttt{boundary}(\Pi_V) \bigcap V_p\right|$, for all $p \in \{^{\cup}p\}$.

3.3.2 Front/Octree as a Hypergraph

A front $\boldsymbol{\mathcal{M}} := (\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{N}})$ can also be written in the form of a hypergraph. $H = (V, E)$ is an equivalent to $\boldsymbol{\mathcal{M}}$ if V represents the set of vertices $\boldsymbol{\mathcal{V}}$ and E represents the set of (unordered) vertices ($\{v_{i_0}, v_{i_1}, v_{i_2}\}$) of all edges e in $\boldsymbol{\mathcal{E}}$. Even though this representation lacks the orientation, we can establish the analysis of graph partition on a front.

Similarly a tree $\boldsymbol{\mathcal{T}}$ can be considered as a directed hypergraph $H := (V, E)$ with V taken as the set of cells of parent tree, $\boldsymbol{\mathcal{C}}^+$. The set E can be considered in different ways, for example, it can be considered as a set of directed hyperedges, $e := (\{c\}, \{^{\cup}n\}_c^+)$, where a leaf cell($c \in \boldsymbol{\mathcal{L}}$) is mapped to the cells in it's neighborhood ($\{^{\cup}n\}_c^+$).

3.4 Problem Statement

In the abstract sense, given a k number of processors with partitioned memory, the aim of a parallel Eulerian-Lagrangian algorithm in the MPI paradigm is to compute the discretized equation in the minimum time with a suitable memory partition so that none of the processor

---

Algorithm 1: `FT-SOLVER`

Data:

1. Computational domain, $\Omega = [0, 1]^D$,
2. Initial conditions in Ω (ex: $\phi_0(\boldsymbol{x}, t = 0)$ is given by some functions like $f_\phi(\boldsymbol{x})$),
3. boundary conditions on $\partial\Omega$ (ex: $\nabla\phi(\boldsymbol{x} \in \partial\Omega)$ is given by some functions like $g_\phi(\boldsymbol{x})$),
4. Interface $\Gamma^0 := \Gamma(t = 0) \subset (\Omega \setminus \partial\Omega)$, (ex: implicit function like $I(\boldsymbol{x}) = 0$ or parametric function $\boldsymbol{x}_0^s(t_1, t_2)$),
5. Jump conditions on Γ (ex: $[[\nabla\phi(\boldsymbol{x}^s)]]$ is given by some function like $h_\phi(\boldsymbol{x}^s)$),
6. End time $t^N := T > 0$

Result:

1. Discretised domain: AMR grid $\boldsymbol{\mathcal{T}}^N := \boldsymbol{\mathcal{T}}(T)$,
2. Discretised Solution Set, $\{^{\cup}\phi_h\}(\boldsymbol{x}_c)\ \forall\ c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}}^N)$,
3. Discretised Interface: Front $\boldsymbol{\mathcal{M}}^N := \boldsymbol{\mathcal{M}}(T)$

---

1. Discretize:
   - 1.1 Discretise Ω by an initial AMR grid $\boldsymbol{\mathcal{T}}^0$
   - 1.2 Discretise Γ_0 by an initial Front $\boldsymbol{\mathcal{M}}^0$
2. Partition:
   - 1.1. Distribute the cells $\boldsymbol{\mathcal{C}}(\boldsymbol{\mathcal{T}}^0)$ into k partitions as $\Pi_{\boldsymbol{\mathcal{C}}} := \{\boldsymbol{\mathcal{C}}_0, \boldsymbol{\mathcal{C}}_1, .., \boldsymbol{\mathcal{C}}_{k-1}\}$ with *balanced* leaves.
   - 1.2. Distribute the front vertices $\boldsymbol{\mathcal{V}}(\boldsymbol{\mathcal{M}}^0)$ as $\Pi_{\boldsymbol{\mathcal{V}}} := \{\boldsymbol{\mathcal{V}}_0, \boldsymbol{\mathcal{V}}_1, ..\boldsymbol{\mathcal{IV}}_{k-1}\}$
   - 1.3. Distribute the front elements $\boldsymbol{\mathcal{E}}(\boldsymbol{\mathcal{M}}^0)$ as $\Pi_{\boldsymbol{\mathcal{E}}} := \{\boldsymbol{\mathcal{E}}_0, \boldsymbol{\mathcal{E}}_1, ..\boldsymbol{\mathcal{E}}_{k-1}\}$
3. Communication Neighbors and Ghost Subsets:
   - 3.1. Maintain the local parent tree $\boldsymbol{\mathcal{T}}_p^+$ from $\boldsymbol{\mathcal{C}}_p$ and ghost cells (non-local neighbor + parents)
   - 3.2. Maintain the expanded elements list $\boldsymbol{\mathcal{E}}_p^+$ from $\boldsymbol{\mathcal{E}}_p$ and ghost elements (non-local valence elements)
   - 3.3. Maintain the expanded elements list $\boldsymbol{\mathcal{V}}_p^+$ from $\boldsymbol{\mathcal{V}}_p$ and non-local vertices of $\boldsymbol{\mathcal{E}}_p^+$;
   - 3.4. Maintain the set of communication neighbors $\{^{\cup}n\}_p$ of p

---

4. while *(MPI running)* do // rank of this processor is $p \in \mathbb{N}_k$
   4.1. Initialize:
      4.1.1. Initialise time step $t^n = t^0 = 0$
      4.1.2. Initial condition $\phi_0(\boldsymbol{x}_c)\ \forall\ c \in \boldsymbol{\mathcal{L}}_p$ /* for all the variables using $\phi_0(\boldsymbol{x}_c, t = 0) \leftarrow f_\phi(\boldsymbol{x}_c)$ */
   4.2. while $(t^n < T)$ do // time step integration
      4.2.1. Calculate Δt^n from stability criteria
      4.2.2. Communication+Inter level interpolation: Update $\phi(\boldsymbol{x}_c)$ in all the required cells of $\boldsymbol{\mathcal{T}}_p^+$
      4.2.3. Advection of Γ^n and Repartition of $\boldsymbol{\mathcal{M}}^n$:
         4.2.3.1. Calculate Lagrangian velocity $\boldsymbol{u}^s(\boldsymbol{x}^s)\ \forall\ \boldsymbol{x}^s \in \boldsymbol{\mathcal{V}}_p$ using $\boldsymbol{\mathcal{T}}$ to $\boldsymbol{\mathcal{M}}$ interpolation of $\boldsymbol{u}(\boldsymbol{x}_c)$
         4.2.3.2. New interface position Γ^{n+1} by advecting local vertices $\boldsymbol{x}^s \in \boldsymbol{\mathcal{V}}_p$ using $\boldsymbol{u}^s(\boldsymbol{x}^s)$
         4.2.3.3. Parallel Topology change
         4.2.3.4. Parallel Regrid Algorithm
         4.2.3.5. Repartition $\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}$ and Update $\boldsymbol{\mathcal{V}}_p^+, \boldsymbol{\mathcal{E}}_p^+$
      4.2.4. One fluid formulation
         4.2.4.1. Calculate heaviside or indicator function $H^{n+1}(\boldsymbol{x}_c)\ \forall\ c \in \boldsymbol{\mathcal{L}}_p$ using $\boldsymbol{\mathcal{T}}$ to $\boldsymbol{\mathcal{M}}$ interpolation of $\boldsymbol{u}(\boldsymbol{x}_c)$
         4.2.4.2. Calculate average fluid properties $\rho^{n+1}(\boldsymbol{x}_c), \mu^{n+1}(\boldsymbol{x}_c)$ using H^{n+1}
      4.2.5. Interface Forces
         4.2.5.1. Calculate $\boldsymbol{F}^{n+1}(\boldsymbol{x}^s)\ \forall\ \boldsymbol{x}^s \in \boldsymbol{\mathcal{V}}_p$ on Γ^{n+1} using $\boldsymbol{\mathcal{M}}$ to $\boldsymbol{\mathcal{M}}$ surface operators
         4.2.5.2. Calculate $\boldsymbol{f}^{n+1}(\boldsymbol{x}_c)\ \forall\ c \in \boldsymbol{\mathcal{L}}_p$ using $\boldsymbol{\mathcal{M}}$ to $\boldsymbol{\mathcal{T}}$ interpolation of $\boldsymbol{F}^{n+1}$
      4.2.6. Solve $\boldsymbol{u}^{n+1}, p^{n+1}$ from $\boldsymbol{u}^n, \rho^{n+1}, \mu^{n+1}, \boldsymbol{f}^{n+1}$ using projection algorithm
      /* The projection algorithm flux calculation, advection algorithm, Poisson solver, etc. In parallel solver, all these routines require an interprocessor communication routines among communication neighbors, $\{^{\cup} n\}_p$, inter-level interpolations */
      4.2.7. AMR + Dynamic Balancing
         4.2.5.1. *1:2 Refinement* of $\boldsymbol{\mathcal{T}}^n$ gives $\boldsymbol{\mathcal{T}}^n$ adapt wavelet error
         4.2.5.2. Restriction/prolongation of each variables, $\{^{\cup}\phi\}$, to the new leaves, $\boldsymbol{\mathcal{L}}_p^{n+1} \setminus \boldsymbol{\mathcal{L}}_p^n$
         4.2.5.3. Repartition by dynamic load balancing: Update $\boldsymbol{\mathcal{I}}_{\boldsymbol{\mathcal{T}}_p}$ and $\boldsymbol{\mathcal{T}}_p^+$
         4.2.5.3. Move the scalar data $\boldsymbol{\mathcal{S}}_{\boldsymbol{\mathcal{T}}}^{\phi}$ to their new owner processor
         4.2.5.3. Repartition $\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}$ and Update $\boldsymbol{\mathcal{V}}_p^+, \boldsymbol{\mathcal{E}}_p^+$
      4.2.8 Update time: $n \leftarrow n + 1,\ t^n \leftarrow t^n + \Delta t^n$
   end
end

runs out of memory. The objective of a scalability is that the algorithm is optimal for achieving the above-mentioned objective for any given number of processors. Using the detailed algorithm of a parallel Eulerian-Lagrangian flow solver algorithm in MPI in described in Algo:1, and the classification of mesh operators listed in the Table:3.1, we discuss the cost and metrics involved in the scalable Eulerian-Lagrangian solvers.

3.4.1 Discussion on Metrics

The primary step in optimizing the efficiency of parallel implementation of Eulerian-Lagrangian methods can be identifying the types of routines. The routines can be classified as Eulerian,

Eulerian-Lagrangian and Lagrangian routines. Some of these operations modify the associated (hyper)graphs, and the cost and frequency of the eventual repartition is a crucial element in choosing the graph partitioning method. All these routines, which are operating on partitioned subsets, require MPI communications. Uniformity of both the size and computational load among each partitioned data set and minimization of communication requirements are the general objectives of any MPI parallel computing algorithms. In the context of Eulerian-Lagrangian meshes where both the meshes are capable of modifying we Also, take into consideration the memory and computational cost associated with repartitioning (while AMR or Front regridding) and the the frequency at which this repartitioning occurs.

An AMR grid $\boldsymbol{\mathcal{T}}$ of $\mathbb{R}^D$ with a maximum depth L has number of leaf cells $|\boldsymbol{\mathcal{L}}| < 2^{DL}$ is partitioned among $k := |\{\overset{\cup}{p}\}|$ processors is immersed with a Front $\boldsymbol{\mathcal{M}}$ with number of vertices and elements $|\boldsymbol{\mathcal{V}}|$ and $|\boldsymbol{\mathcal{E}}|$ respectively. The communication cost among parts of the Eulerian grid can be directly correlated with either of the sets: *edge-cut* or *boundary-vertices* (refer Section:3.3.1). Edge-cut of each processor, $\texttt{cut}(\Pi_{\mathcal{C}^+})$ of an AMR grid partitioned among k processors satisfies $|edge - cut| = cN^{(D-1)/D}$ [48], where $N = |\boldsymbol{\mathcal{L}}|/k$ is the number of leaves per processor. The constant c is optimal (least) for heuristic-based graph partition compared to that of SFC-based graph partition algorithms, but those methods are not ideal for the reasons *(i)* high repartition cost which is not suitable for solvers and requires high-frequency of load balancing (AMR) and *(ii)* large memory demanded by partition/repartition algorithm. Among SFC-based partition Hilbert curve has a lower edge-cut compared to Morton curve, as it is quite evident from the locality-preserving nature of Hilbert curves. From a practical point of view (availability of the free platform Basilisk [29]), the Lagrangian mesh is partitioned based on Morton. In general, we can write the cost of communication of AMR grid is $\mathcal{O}(N^{(D-1)/D})$.

The size of the edge cut of the partition of the front (that represents an $(D-1)$-dimensional manifold in $\mathbb{R}^D$) can be taken as $\mathcal{O}(N^{(D-2)/D})$ where $N = |\boldsymbol{\mathcal{L}}|/k$. This gives the cost of communication during Lagrangian and Eulerian-Lagrangian routines is of the order of $\mathcal{O}(N^{(D-2)/D})$.

During AMR modification, the communication cost for dynamic load balancing based on the SFC curve is $\mathcal{O}(N)$. In this thesis, the partition of the front is based on the locality of the owner cell of vertices (Section:3.5.1) which guarantees that the communication cost of repartition of the front is also $\mathcal{O}(N)$ where N can be taken as $|\boldsymbol{\mathcal{E}}|/k$.

3.5 Implementation

In this section, we discuss in detail the partition of grids, optimized communication strategies, and mesh adaptation are discussed in this section.

3.5.1 Partition of Tree and Front

As mentioned in section: and section: a tree and a front can be represented as hypergraphs (or more general combinatorial maps) or rather *weighted hypergraph* (refer Section:3.3.1) in the context of parallel computing. *Partition* of hypergraphs for parallel computation satisfies *balanced constraint* (refer section:3.3.1) with other minimization objectives like the cost of inter-processor communication. When the tree modifies, processors may go *overload* (refer section:3.3.1) and you need to do *repartition* (or *rebalance*) such that the balance constraint is satisfied after any mesh modification.

	Eulerian	Eulerian-Lagrangian	Lagrangian
Requires repartition	(i) AMR	-	(i) Regridding (ii) Topology change
No repartition	(i) Inter-level communications (eq: prolongation/restriction) (ii) Intra-level communication (eq: scalar interpolation)	(i) Eulerian mesh to Lagrangian mesh interpolation (ex: Velocity at marker points) (ii) Lagrangian mesh to Eulerian mesh interpolation (ex: Interfacial force at leaf cells)	(i) Surface Interpolation (ex: Curvature calculation, surface derivatives, etc.)

Table 3.1: General routines in an Eulerian-Lagrangian solvers

When we come to the specific context of the solver that uses both Eulerian and Lagrangian meshes simultaneously; we need to partition both tree and front . Since most of the routines function on the set of leaf-cells , it is meaningful to distribute tree such that all the partitions have more or less equal number of leaf-cells . Interprocessor communication can be minimized depending on the SFC. The use of a locality-preserving SFC can reduce the inter-processor communication load.

3.5.2 Partition of Tree and Parent Tree

This subsection explains the partition of tree , which was already implemented in [29] for the sake of definitions required by subsequent sections.

Since most of the routines related to the AMR solvers are done on the set of leaves ($\boldsymbol{\mathcal{L}}$) partition of the parent tree ($\boldsymbol{\mathcal{T}}^+$) which is comprised of the cells $\boldsymbol{\mathcal{C}}^+$ is partitioned in such a way that the number of leaf-cells per processors is *balanced* [2] There are many ways to balance the partition of leaves but we can make it unique if we make use of an SFC (which is Morton curve in [29]). Set of leaves $\boldsymbol{\mathcal{L}}$ of the tree $\boldsymbol{\mathcal{T}}$ can be distributed among k number of procs with their ranks in the set $\{^{\cup}p\} = \mathbb{N}_k$ using the Morton curve $\mathcal{Z}$ as $\Pi_{\boldsymbol{\mathcal{L}}} := \{\boldsymbol{\mathcal{L}}_0, \boldsymbol{\mathcal{L}}_1, .., \boldsymbol{\mathcal{L}}_{k-1}\}$ such that the number of leaf-cells in each partition is given by $|\boldsymbol{\mathcal{L}}_p| := \lceil |\boldsymbol{\mathcal{L}}|/k \rceil$ for all procs with rank $p < (|\boldsymbol{\mathcal{L}}| \mod k)$ and $|\boldsymbol{\mathcal{L}}_p| := \lfloor |\boldsymbol{\mathcal{L}}|/k \rfloor$ for all other procs.

$$\boldsymbol{\mathcal{L}}_p := \left\{ c = \mathcal{Z}(z) \in \boldsymbol{\mathcal{L}} \;\middle|\; \sum_{0 \le p' < p} |\boldsymbol{\mathcal{L}}_{p'}| \le z < \sum_{0 \le p' \le p} |\boldsymbol{\mathcal{L}}_{p'}| \right\} \quad \forall\, p \in \mathbb{N}_k, \tag{3.1}$$

There is a partition of all the cells ($\boldsymbol{\mathcal{C}}$) of the tree $\boldsymbol{\mathcal{T}}^+$) with every p-th partition ($\boldsymbol{\mathcal{I}}^+_{\boldsymbol{\mathcal{T}}_p}$) is a superset of $\boldsymbol{\mathcal{L}}_p$ and also contains the ghost cells for MPI communication. Refer to [29] for the conditions for partitioning the parent tree and the MPI ghost cells.

[2] Partition of tree is ϵ-balanced with $\epsilon = 0$. If the cost of computation is 1 for leaf-cells and 0 for other cells of the parent-tree then 0-balanced partition gives $\lfloor |\boldsymbol{\mathcal{L}}|/k \rfloor$ or $\lceil |\boldsymbol{\mathcal{L}}|/k \rceil$ number of leaf-cells per processor and the distribution of other cells are based on some other objective functions which are out of scope for this chapter.

Local Tree

There is a partition of all the cells ($\boldsymbol{\mathcal{C}}^+$) of the parent tree ($\boldsymbol{\mathcal{T}}^+$) with every p-th partition ($\boldsymbol{\mathcal{C}}_p^+$) is a superset of $\boldsymbol{\mathcal{L}}_p$ and also contains the ghost cells for MPI communication. Refer [29] for the conditions for partitioning the parent tree and the MPI ghost cells. The cell rank $\mathcal{P}_c : \boldsymbol{\mathcal{C}}^+ \mapsto \mathbb{N}_k$ maps every cell of the parent tree to a unique processor. For the leaf cells, the cell rank can be defined using Eq:3.1 as

$$\mathcal{P}_c(c) = p \iff c \in \boldsymbol{\mathcal{L}}_p \tag{3.2}$$

Ghost Cells and Communication Neighbors

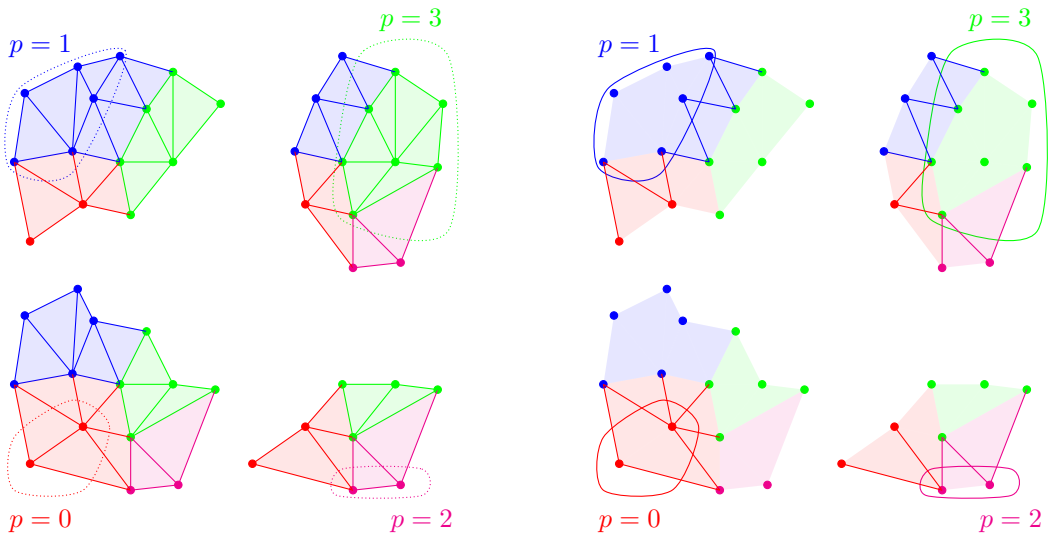
As mentioned above, for a processor with rank p, the extended list of cell partition, ($\boldsymbol{\mathcal{C}}_p^+$) contains ghost cells (i.e the cells $c \in \boldsymbol{\mathcal{C}}_p^+$ with $\mathcal{P}_c(c) \neq p$) and the list of the ranks of them is the list of communication neighbors of p, given by

$$\{^{\cup} n\}_p := \{p' \in \mathbb{N}_k \setminus \{p\} \mid \exists c \in \boldsymbol{\mathcal{C}}_p^+ \text{ s.t } \mathcal{P}_c(c) = p'\} \tag{3.3}$$

Local subdomain

The subset of control volume associated with a processor, $\Omega_p = \cup_{c \in \boldsymbol{\mathcal{L}}_p} \Omega_c$. In order to associate a unique owner with any point $\boldsymbol{x} \in \Omega$, we define $\overline{\Omega}_p := \cup_{c \in \boldsymbol{\mathcal{L}}_p} \overline{\Omega}_c$ and any discrete Lagrangian point $\boldsymbol{x}^s \in \{^{\cup} v\}$ is considered with a unique processor owner,p, if $\boldsymbol{x}^s \in \overline{\Omega}_p$.

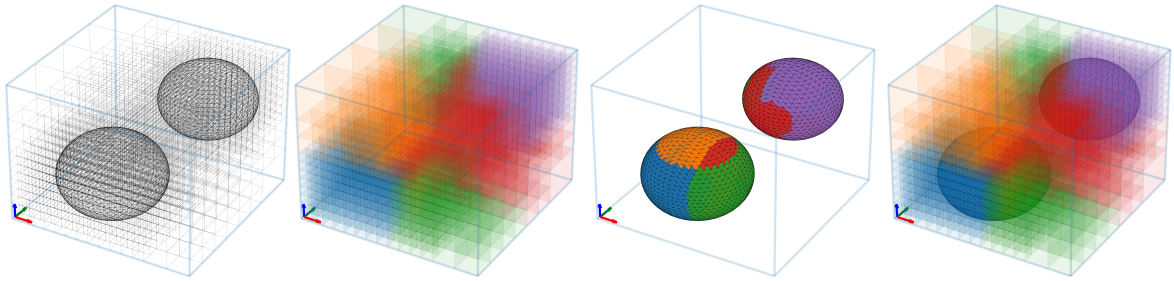
3.5.3 Partition of Front



(a) (b)

Figure 3.4: *(a)* Every processor owns the local partition of vertices ($\boldsymbol{\mathcal{V}}_p$) and triangles ($\boldsymbol{\mathcal{E}}_p$). The ghost layer contains all the non-local triangles ($\boldsymbol{\mathcal{E}}_p^+ \setminus \boldsymbol{\mathcal{E}}_p$) and their non-local vertices ($\boldsymbol{\mathcal{V}}_p^+ \setminus \boldsymbol{\mathcal{V}}_p$). *(b)* Edge cut of partition.

Partition of the front can be either aimed at balanced distribution of computational load associated with Lagrangian routines or can be aimed at maintaining a concurrency with the locality of nearby leaf cells. In this thesis, the latter is preferred since balanced partition of



(a) (b) (c) (d)

Figure 3.5: Distribution of Eulerian and Lagrangian Meshes in parallel computing. *(a)* Octree leaf cells and front elements, *(b)* Partition of Octree ($\mathcal{T}$) leaves among five procs. (Color represents rank), *(c)* Partition of front ($\mathcal{M}$) *(d)* Partition of $\mathcal{M}$ such that the locality of vertices and elements are in the neighborhood of leaf cells.

Lagrangian mesh doesn't guarantee the local subsets of triangles and vertices are local to the local Eulerian subdomain, Ω_p.

If the leaf cells are partitioned by the SFC, as in Eq:3.1, then vertices can be partitioned in such a way that each processor owns the local vertices. If we define the rank of a vertex as the rank of the processor, which owns it, and similarly, we define rank for an (undirected) edge and element of the front as the,

$$\mathcal{P}_v(v) : \boldsymbol{\mathcal{V}} \mapsto \mathbb{N}_k \quad : \quad \text{Maps a vertex } v \in \boldsymbol{\mathcal{V}} \text{ to the (rank of its) owner processor}$$
$$\mathcal{P}_v(\boldsymbol{x}^s) = p \iff \boldsymbol{x}^s \in \overline{\Omega}_p \tag{3.4}$$
$$\mathcal{P}_l(l) : \{^{\cup} l\} \mapsto \mathbb{N}_k \quad : \quad \text{Maps an edge } l \in \{^{\cup} l\} \text{ to the (rank of its) owner processor}$$
$$\mathcal{P}_l(\{v_{i_0}, v_{i_1}\}) = p \iff \underset{j}{argmin}\{\mathcal{P}_v(v_{i_j})\} = p \tag{3.5}$$
$$\mathcal{P}_e(e) : \boldsymbol{\mathcal{E}} \mapsto \mathbb{N}_k \quad : \quad \text{Maps an element } e \text{ to the (rank of its) owner processor.}$$
$$\mathcal{P}_e((v_{i_0}, v_{i_1}, v_{i_2})) = p \iff \underset{j}{argmin}\{\mathcal{P}_v(v_{i_j})\} = p \tag{3.6}$$

So the partition is such that each processor p owns vertices, edges, and elements which are local, i.e., whose ranks are p. These subsets are represented by $\boldsymbol{\mathcal{V}}_p$, $\{^{\cup} l\}_p$ and $\boldsymbol{\mathcal{E}}_p$ and we maintain an extended set of elements $\boldsymbol{\mathcal{E}}_p^+$ which contains all the ghost elements, $\boldsymbol{\mathcal{E}}_p^+ \setminus \boldsymbol{\mathcal{E}}_p^+$ (non-local valence elements of elements in $\boldsymbol{\mathcal{E}}_p$). Partition of front is represented in Fig:3.3 Fig:3.9 and Fig:3.5. From Eq:3.4-Eq:3.6, we need to know the owner leaf cell of each vertex to determine the rank of vertices, edges, and elements which are found using Algo:3 (`LOCATE` routine). The algorithm of partition is shown in Algo:2.

`LOCATE`($\boldsymbol{x}^s$, $\mathcal{T}$): locating the owner cell of $\boldsymbol{x}^s$

The algorithm `LOCATE`($\boldsymbol{x}^s$, $\mathcal{T}$) (which is equivalent of $o(\boldsymbol{x}^s)$) returns the vertex-owner of $\boldsymbol{x}^s$. The vertex-owner is a leaf-cell , which can be of any refinement level. So, we have to use iterative search to find it. Rank of vertex, $\mathcal{P}_v(\boldsymbol{x}^s)$ is equivalent it $\mathcal{P}_c\big(o(\boldsymbol{x}^s)\big)$ is implemented using `LOCATE-RANK`.

Algorithm 2: `FRONT-PARTITION`

Data: Every processor owns
1. The local parent tree $\mathcal{T}_p^+$ with cells $\mathcal{C}_p^+$ with cell rank $\mathcal{P}_c(c)$ for all cells $c \in \mathcal{C}_p^+$
2. A copy of (global) front with ghost layers $\mathcal{M}_p := (\mathcal{V}_p, \mathcal{E}_p, \mathcal{N}_p)$

Result:
1. Local Mesh: Each processor reduces $\mathcal{M}$ to $\mathcal{M}_p^+ := (\mathcal{V}_p^+, \mathcal{E}_p^+, \mathcal{N}_p^+)$.
2. Connectivity: Connectivity is maintained inherently by maintaining the rank of ghost elements and vertices. ($\mathcal{P}_v(v)\ \forall\ v \in \mathcal{V}_p^+$ and $\mathcal{P}_e(e)\ \forall\ e \in \mathcal{E}_p^+$)

```
while MPI running do                                   /* Rank of 'this' processor is p */
    1. for each vertex v do                            /* v ∈ V */
        1.1. P_v(v) ← −1                               /* Unknown rank */
    end
    2. for each vertex e do                            /* e ∈ E */
        2.1. P_e(e) ← −1                               /* Unknown rank */
    end
    3. for each vertex v do                            /* v ∈ V */
        if LOCATE-RANK(v) == p then                    /* v ∈ Ω̄_p */
            3.1. P_v(v) ← p                            /* Local vertex */
            3.2. for each valence triangle e of v do
                3.2.1. P_e(e) ← p                      /* Assume Local element */
                3.2.2. for each vertex v' of e do      /* v ∈ V */
                    3.2.3. p' ← LOCATE-RANK(v')
                    3.2.3. P_v(v') ← p'
                    3.2.3. P_e(e) ← min{p', P_e(e)}
                end
            end
        end
    end
        /* Now local sets (V_p and E_p) are found. The algorithm used for the rank of a
        vertex, LOCATE-RANK(v), in 3.1. and 3.2.3. is equivalent to P_c(LOCATE(v)) and
        only works if the owner leaf cell of v is either a local cell or a ghost cell in the
        local parent tree c ∈ C_p^+ ⋂ L. That is the reason we iterate through valence
        vertices and find their rank rather than finding the rank of all vertices. The next
        step will be to find all the missing ghost elements and their vertices. */
    3.2. for each local triangle e do                  /* P_e(e) == p */
        3.2. for each valence triangle e' of e do      /* Can skip if P_e(e') ≠ −1 */
            3.2.1. P_e(e') ← p
            3.2.2. for each vertex v' of e' do
                3.2.3. p' ← LOCATE-RANK(v')
                3.2.3. P_v(v') ← p'
                3.2.3. P_e(e) ← min{p', P_e(e)}
            end
        end
    end
    Delete all vertices with P_v(v) == −1
    Delete all elements with P_e(e) == −1
end
```

Algorithm 3: LOCATE$(\boldsymbol{x}^s, \boldsymbol{\mathcal{T}}_p^+)$

Data: Lagrangian Point $\boldsymbol{x}^s$, Local parent tree $\boldsymbol{\mathcal{T}}_p^+$ with cells $\boldsymbol{\mathcal{C}}_p^+$ and each of their rank $(\mathcal{P}_c(c)\ \forall\ c \in \boldsymbol{\mathcal{C}}_p^+)$

Result: Owner (local) leaf-cell , $c \in \boldsymbol{\mathcal{L}}_p$ if $\boldsymbol{x}^s \in \overline{\Omega}_p$

```
l ← depth(T_p^+)
while l ≥ 0 do
    h ← 2^{-l}
    c ← (⌊x_0^s/h⌋, ⌊x_1^s/h⌋, ⌊x_2^s/h⌋, l)
    if c ∈ L_p then
        return(c)
    end
    l ← l − 1
end
c ← (−1, −1, −1, −1)
return(c)
/* Not found. Returns an invalid cell. */
```

Algorithm 4: LOCATE-RANK$(\boldsymbol{x}^s, \boldsymbol{\mathcal{T}}_p^+)$

Data: Lagrangian Point $\boldsymbol{x}^s$, Local parent tree $\boldsymbol{\mathcal{T}}_p^+$ with cells $\boldsymbol{\mathcal{C}}_p^+$ and each of their rank $(\mathcal{P}_c(c)\ \forall\ c \in \boldsymbol{\mathcal{C}}_p^+)$

Result: Rank of owner leaf-cell , $c \in \boldsymbol{\mathcal{L}} \bigcap \boldsymbol{\mathcal{C}}_p^+$ if c is a leaf cell in the local parent tree

```
l ← depth(T_p^+)
while l ≥ 0 do
    h ← 2^{-l}
    c ← (⌊x_0^s/h⌋, ⌊x_1^s/h⌋, ⌊x_2^s/h⌋, l)
    if c ∈ C_p^+ ⋂ L then /* A Leaf cell of parent tree */
        return(Rank(c))
    end
    l ← l − 1
end
return(−1)
/* Not found. Returns an invalid rank. */
```

3.6 Repartition

As mentioned in the earlier chapter (2.6), both $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{\mathcal{M}}$ modify in time, and the data structures that store a set of leaves (and all other cells), vertices, elements, and neighbors are repartitioned (such that equations Eq:3.1, Eq:3.4-3.6, are always satisfied). The repartition is also as called *dynamic load balancing*. The two scenarios that arise from repartition are adaptive mesh refinement and the front advection.

3.6.1 Adaptive Mesh Refinement

When the tree is modified by adaptive mesh refinement using the adapt wavelet algorithm (section:2.28), dynamic load balancing of the AMR grid is required. In the context of the Eulerian-Lagrangian method, we also need to redistribute the front so that the equations (Eq:3.4-Eq:3.6) are satisfied. As mentioned, the dynamic balancing of the Eulerian grid is based on the z-order curve of the new tree. In order to carry out the repartition of vertices, we can define a z-order curve for vertices so that we can identify vertices in each leaf cell.

Z-order of Vertices

As in the point quadtree representation [49] [50], we can define SFC curve of vertices, $\mathcal{Z}_{\boldsymbol{\mathcal{V}}} : \mathbb{N}_{|\boldsymbol{\mathcal{V}}|} \mapsto \boldsymbol{\mathcal{V}}$ as shown in Fig:3.6. If we store the vertices $\boldsymbol{\mathcal{V}}$ in a cache called $\boldsymbol{\mathcal{V}}'$ along with two scalars $\boldsymbol{\mathcal{S}}_{\boldsymbol{\mathcal{T}}}^{nv}$ and $\boldsymbol{\mathcal{S}}_{\boldsymbol{\mathcal{T}}}^{zv}$ which store respectively the number of vertices in a leaf cell and starting index (in the cache) of a vertex in the leaf cell, then all the vertices in a leaf cell can be

found using

$$v_i := \boldsymbol{\mathcal{V}}'[i] \in \overline{\Omega}_c \text{ if } \boldsymbol{\mathcal{S}}^{zv}_{\boldsymbol{\mathcal{T}}}[c] \leq i < \boldsymbol{\mathcal{S}}^{zv}_{\boldsymbol{\mathcal{T}}}[c] + \boldsymbol{\mathcal{S}}^{nv}_{\boldsymbol{\mathcal{T}}}[c] \tag{3.7}$$

The algorithm is shown in Algo:5 and also refers to Fig:3.6

Algorithm 5: `VERTEX-CACHE`

Data:
1. The tree $\boldsymbol{\mathcal{T}}$ with leaves $\boldsymbol{\mathcal{L}}$
2. The front $\boldsymbol{\mathcal{M}} := (\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{N}})$

Result:
1. Cache $\boldsymbol{\mathcal{V}}'$ of $\boldsymbol{\mathcal{V}}$
2. Scalar $\boldsymbol{\mathcal{S}}^{nv}_{\boldsymbol{\mathcal{T}}}$ which stores the number of vertices in a cell.
3. Scalar $\boldsymbol{\mathcal{S}}^{zv}_{\boldsymbol{\mathcal{T}}}$, which stores the starting index of vertices in a cell.

1. Iterate through each leaf $c \in \boldsymbol{\mathcal{L}}$ by $\mathcal{Z}$
   1.1. $\boldsymbol{\mathcal{S}}^{nv}_{\boldsymbol{\mathcal{T}}}[c] \leftarrow 0$
2. Iterate through each vertex $v_i \in \boldsymbol{\mathcal{V}}$
   2.1. $c \leftarrow \texttt{LOCATE}(v_i)$
   2.2. $\boldsymbol{\mathcal{S}}^{nv}_{\boldsymbol{\mathcal{T}}}[c] \leftarrow \boldsymbol{\mathcal{S}}^{nv}_{\boldsymbol{\mathcal{T}}}[c] + 1$
3. $zv \leftarrow 0$
4. Iterate through each leaf $c \in \boldsymbol{\mathcal{L}}$ by $\mathcal{Z}$
   4.1. $zv \leftarrow zv + \boldsymbol{\mathcal{S}}^{nv}_{\boldsymbol{\mathcal{T}}}[c]$
   4.1. $\boldsymbol{\mathcal{S}}^{zv}_{\boldsymbol{\mathcal{T}}}[c] \leftarrow zv$
5. Iterate through each vertex $v_i \in \boldsymbol{\mathcal{V}}$
   5.1. $c \leftarrow \texttt{LOCATE}(v_i)$
   5.2. $\boldsymbol{\mathcal{S}}^{zv}_{\boldsymbol{\mathcal{T}}}[c] = \boldsymbol{\mathcal{S}}^{zv}_{\boldsymbol{\mathcal{T}}}[c] - 1$
   5.3. $j \leftarrow \boldsymbol{\mathcal{S}}^{zv}_{\boldsymbol{\mathcal{T}}}[c]$
   5.4. $\boldsymbol{\mathcal{V}}'[j] \leftarrow v_i$

/* Now every vertices $\boldsymbol{\mathcal{V}}'[i]$ in the cache with index $\boldsymbol{\mathcal{S}}^{zv}_{\boldsymbol{\mathcal{T}}}[c] \leq i < \boldsymbol{\mathcal{S}}^{zv}_{\boldsymbol{\mathcal{T}}} + \boldsymbol{\mathcal{S}}^{nv}_{\boldsymbol{\mathcal{T}}}$ belongs to the leaf c. */

3.6.2 Advection of Front

The marker points on the surface mesh advect with local velocity. In distributed computing, each processor updates the coordinates of all local front points, $\forall \boldsymbol{x}^s \in \boldsymbol{\mathcal{V}}_p$:

$$\boldsymbol{x}^s \leftarrow \boldsymbol{x}^s + \Delta t^n \sum_{c' \in \{\cup n\}^+_c} \boldsymbol{u}(\boldsymbol{x}_{c'}) \delta_h(\boldsymbol{x}_{c'} - \boldsymbol{x}^s) \text{ where } c = o(\boldsymbol{x}^s) \tag{3.8}$$

where the summation is carried out on all the cells in the neighbourhood of the owner cell ($o(\boldsymbol{x}^s)$) of the vertex $\boldsymbol{x}^s$. Since the CFL condition $\frac{|\boldsymbol{u}|\Delta t^n}{h_{min}} < 0.5$ is imposed, it is guaranteed that the the new coordinates of the vertex will be in the 3x3x3 neighborhood of $o(\boldsymbol{x}^s)$ and we can find the new rank of the vertex using `LOCATE-RANK` (Algo:4). Subsequently, the ghost layer is updated by looking for rank of valance triangles and their vertices (as in Algo:2).

After every (or every few) number of time steps, the front mesh has to undergo remeshing or topology modification routines (Section:3.8). Regridding in partitioned surface mesh is discussed in Section:3.8. Parallel front topology modification operation is not discussed in this thesis and is left for future development.

3.7 Inter Grid Communication in Parallel

3.7.1 Cells, Elements, and Vertices in the neighborhood

As you have seen in equations Eq:3.4-Eq:3.6, since δ_h is 0 for $|r| > 2$, we can optimise the summation if we have the vertex-neighborhood ($\{c' \in \{^{\cup}n\}_c^+ \mid c = o(\boldsymbol{x}^s)\}$) of a front point ($\boldsymbol{x}^s$) and all the front points ($\{v \in \{^{\cup}v\}_{c'} \mid c' \in \{^{\cup}n\}_c^+\}$) or front elements ($\{e \in \{^{\cup}e\}_{c'} \mid c' \in \{^{\cup}n\}_c^+\}$) in the `cell-neighborhood` of a leaf cell (c).

1. The cell-neighborhood $\{c' \in \{^{\cup}n\}_c^+\}$ of the leaf-cell $c := (i, j, k, l)$ can be easily found by changing the indices like $i \leftarrow i \pm 1, 2$

2. The cells in the neighborhood of the marker point $\boldsymbol{x}^s$, is the cell-neighborhood of the vertex-owner cell, ($o(\boldsymbol{x}^s)$) of the marker point ($\boldsymbol{x}^s$). In a uniform grid of `depth` l, the owner cell is simply $\left(\left\lfloor \frac{x_0^s}{2^l} \right\rfloor, \left\lfloor \frac{x_1^s}{2^l} \right\rfloor, \left\lfloor \frac{x_2^s}{2^l} \right\rfloor, l\right)$ and in an AMR grid, $\boldsymbol{\mathcal{T}}$, you can find it using Algo:3 `LOCATE`..

3. The front points and front elements in the neighborhood of a leaf cell $c \in \boldsymbol{\mathcal{O}}$ can be easily obtained if you have the set of front points ($\{^{\cup}v\}_{c'}$) or front elements ($\{^{\cup}e\}_{c'}$) in every cells c' of the parent tree $\boldsymbol{\mathcal{T}}^+$. The number of front points/front elements in a cell is different for different cells. So, instead of creating front element arrays for each cell, we can create a single front element cache that is in correspondence with the z-order of leaves. (Refer Algo:5)

3.8 Surface Regridding or Remeshing in Parallel

As mentioned in Section:, front regridding is essential for Eulerian-Lagrangian methods involving a moving front. There are two aspects involved with front regridding (or *surface remeshing* as in most of the literature): *(i)* metrics associated with surface quality which is demanded by the solver; and *(ii)* time complexity, especially in the context of partitioned sets. With the perspective of the objectives of this thesis, which is establishing a parallel front tracking code with distributed Eulerian and Lagrangian meshes, this section discusses how to efficiently regrid a partitioned front or Lagrangian mesh with optimised communication.

The objectives of surface mesh regridding are maintaining the edge lengths within an interval $[a_{min}, a_{max}]$ where a_{max} is of the order of $h/2$, maintaining quality triangles (closer to an equilateral triangle) and smoother tangent planes for calculating surface derivatives. The assigned edge length interval avoids leaks in δ_h interpolation between Eulerian and Lagrangian meshes and avoid unnecessary small triangles, which do not give any advantage. The most common metrics that are used to measure the quality of a surface mesh is *(i)* minimum or maximum angle (optimal angle is 60 degrees), *(ii) quality* or aspect ratio (optimal value of quality and aspect ratio are both 1) and *(iii)* the valence of the vertices (optimal valence of an internal vertex is 6) [51]. The quality of the triangle can be expressed as $Q(t) := \frac{6A_t}{\sqrt{3}S_tE_t}$ where A_t, S_t and E_t are respectively, triangle's area, half perimeter, and length of the longest edge. There are different classifications for remeshing algorithms [51] among which we use surface remeshing by local mesh modification methods [52] which is suitable for front tracking methods [53] [32]. In local mesh modification methods, edge manipulations are done in series over all edges.

Data Structure Requirement

During mesh modification, we might add new front points and front elements or delete the existing front points and front elements. Additionally, all these operations involve changes in the way triangles are connected, and thus, the sets of vertex coordinates $\boldsymbol{\mathcal{V}}$, triangles $\boldsymbol{\mathcal{E}}$, neighbors $\boldsymbol{\mathcal{N}}$ undergo insertions, deletions, and modification. So, a triangle based surface mesh data structures similar to [53] is used in this thesis where the sets of vertices, triangles and neighbors are stored in a linked list which are capable of insertions and deletions. Refer to Sec:2.5 and Sec:A.1.

3.8.1 Regridding in Parallel

In parallel remeshing, local mesh operations are divided among processors such that each processor, with rank p, operates on local edges ($\{^{\cup}l\}_p$) and local vertices ($\boldsymbol{\mathcal{V}}$). However, during edge operations, special care has to be given for local edge-cut ($\texttt{cut}(\Pi_{\boldsymbol{\mathcal{V}}}) \bigcup \{^{\cup}l\}_p$) to avoid race condition.

Avoiding Race Condition in MPI

Every edge operation creates or deletes triangles and vertices, which results in the modification of the triangle connectivity in the neighborhood of the edge (Fig:3.8*(b)*). Therefore, the triangle-based database used in the thesis cannot be directly parallelized or multithreaded, because multiple threads performing edge collapse or edge split operations on two neighboring edges might require changes in the connectivity of the same triangles, leading to a *race condition* where different thread or processor tries to modify the same data simultaneously. Apart from adding or deleting vertices and elements, an edge operation also changes the connectivity of the triangles in the valence region of the edge. The valence region of an edge is

shown in Fig:3.8*(b)*. Due to the reason, the connectivity changes during an edge operation the corresponding edge and makes the edge operation challenging for the edges in the edge-cut ($l \in \texttt{cut}(\Pi_{\boldsymbol{\mathcal{V}}})$) because of the *race condition* where different processors concurrently tries to modify same data structure ($\boldsymbol{\mathcal{V}}$, $\boldsymbol{\mathcal{E}}$ and $\boldsymbol{\mathcal{N}}$) at the same memory location. So, an edge operation (collapsing or splitting) is carried out in two steps where in the first step of the operation we carried out only the local edges, which are not edge-cut and which lie inside the local domain completely (*inside edge*). *(b)* Subsequently, the edge operation is carried out on local edges, which is also a cut-edge ($\{\overset{\cup}{}l\}_p \bigcap \texttt{cut}(\Pi_{\boldsymbol{\mathcal{V}}})$). The algorithm is explained in 6 and Fig:3.9. The algorithm can be further improved if the mesh is implemented using a half-edge mesh [54] which gives more independence in edge operation [55]. However, this is a future work.

Algorithm

3.8.2 Future Plan

The algorithm still requires further optimization for better scalability. To avoid the race condition in edge operation, they are performed in steps as described in the section above. This restriction can be addressed by employing a half-edge mesh which is left for future improvements. A half-edge mesh [54] stores a triangle as a set of three consecutive directed half-edges, where each half-edge is a directed edge connecting two vertices of a triangle. In the database, every half-edge is connected to a *previous* and *next* half-edge of the triangle. Similarly, every half-edge has a *twin* or *flip* half-edge, which is the oppositely directed half-edge of the neighboring triangle. A half-edge mesh offers the advantage of edge collapse and edge split on every edges independently. However, this approach might result in a temporarily invalid mesh where a polygon has more than three edges (no longer a triangle) or a triangle with zero area, which can be corrected afterward. This independence in edge operations can be utilized for hyperthreading and MPI parallelization techniques.

The regrid algorithm discussed in this chapter doesn't consider volume loss which can be improved by employing algorithms like those discussed in [56] [57] [58].

3.9 Results

We discuss in detail the implementation and benchmark comparison of front tracking with existing Literature in Chapter:4. Fig:3.10 and Fig:3.11 illustrate AMR and parallel capabilities of the multiphase flow solver in 2D.

3.9.1 Scalability

In this section, we will evaluate the scalability of the parallel front tracking solver discussed in the previous sections. Scalability of AMR Eulerian grid using Morton-curve-based partition implemented in-house solver basilisk [29] is shown in Fig:4.10.

In order to test the scalability of repartition after advection (and subsequent regridding), it is looked into with a simple test case. Here, four circular droplets are advected in a solenoidal velocity field (given by stream function $\psi(x,y) = \frac{1}{\pi}\sin^2(\pi x)\sin^2(\pi x)\cos(\pi\frac{t}{T})$). After each time step,*(i)* the front undergoes regridding to even out the edge lengths, and *(ii)* the grid undergoes AMR remeshing to capture the interface at maximum refinement. The average computational time of different numbers of processors and levels are shown in Fig:3.12. The

Algorithm 6: PARALLEL-REGRID

Data:
1. Local parent tree $\boldsymbol{\mathcal{T}}_p^+$ with cells $\boldsymbol{\mathcal{C}}_p^+$ with cell rank $\mathcal{P}_c(c)$ for all cells $c \in \boldsymbol{\mathcal{C}}_p^+$
2. Local Front with ghost layers $\boldsymbol{\mathcal{M}}_p^+ := (\boldsymbol{\mathcal{V}}_p^+, \boldsymbol{\mathcal{E}}_p^+, \boldsymbol{\mathcal{N}}_p^+)$ with processor rank for all vertices ($\mathcal{P}_v(v)\ \forall\ v \in \boldsymbol{\mathcal{V}}_p^+$) and triangles ($\mathcal{P}_e(e)\ \forall\ e \in \boldsymbol{\mathcal{E}}_p^+$) given.

Result:
1. Improved Local Front $\boldsymbol{\mathcal{M}}_p := (\boldsymbol{\mathcal{V}}_p, \boldsymbol{\mathcal{E}}_p, \boldsymbol{\mathcal{N}}_p)$ with every vertices ($v \in \boldsymbol{\mathcal{V}}_p$) and triangles ($e \in \boldsymbol{\mathcal{E}}_p$) satisfying the isotropic condition listed in Sec:2.5.6.

---

while *MPI running* do
- 1. for *each local edge* $l := \{v_i, v_j\}$ *except edge-cut* do /* $l \in \{^{\cup} l\}_p \setminus \texttt{cut}(\Pi_{\boldsymbol{\mathcal{V}}})$ */
  - 1.1 if *edge size is larger than* a_{max} then
    - 1.1.1 Split l
  - end
- end
- 2. Wait for nearby processors to finish step 1. /* MPI_Barrier could work */
- 3. for *each local edge* l *which is a cut-edge* do /* $l \in \{^{\cup} l\}_p \bigcap \texttt{cut}(\Pi_{\boldsymbol{\mathcal{V}}})$ */
  - 3.1. if *edge size is larger than* a_{max} then /* */
    - 3.1.1. Split l
  - end
- end
- 4. MPI communicate (non-block send/receive) to update vertices, elements, and connectivity in ghost region
- 5. for *each local edge* $l := \{v_i, v_j\}$ *except edge-cut* do /* $l \in \{^{\cup} l\}_p \setminus \texttt{cut}(\Pi_{\boldsymbol{\mathcal{V}}})$ */
  - 5.1. if *edge size is smaller than* a_{min} then /* */
    - 5.1.1. Collapse l
      which is left for future improvements
  - end
- end
- 6. Wait for nearby processors to finish step 4. /* MPI_Barrier could work */
- 7. for *each local edge* l *which is a cut-edge* do /* $l \in \{^{\cup} l\}_p \bigcap \texttt{cut}(\Pi_{\boldsymbol{\mathcal{V}}})$ */
  - 7.1 if *edge size is larger than* a_{max} then /* */
    - 7.1.1. Collapse l
  - end
- end
- 8. MPI communicate (non-block send/receive) to update ghost region
- 9. for *each local vertex* v do /* $v \in \boldsymbol{\mathcal{V}}_p$ */
  - 9.1. Vertex smooth v
- end
- 10. MPI communicate (non-block send/rev) to update vertices in ghost region
- 11. Repeat Steps 1.-10. for few cycles (Ex: 6 smoothing cycles)

end

---

number of vertices and elements increases as the refinement level increases. Referring to Fig:3.12*(b)*, as the number of vertices and elements increase, the solver becomes more efficient.

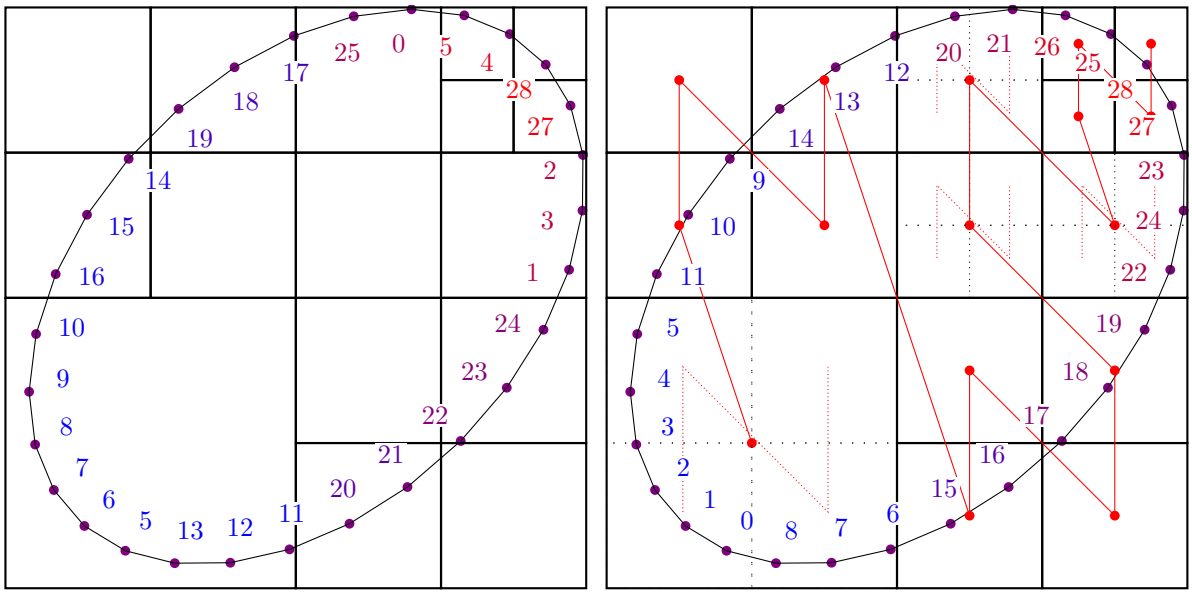


Figure 3.6: *Zorder of front points*: *(a)*: A front with vertices $v_i \in \boldsymbol{\mathcal{V}}$ marked with their indices $i \in \mathbb{N}_{|\boldsymbol{\mathcal{V}}|}$ in the set $\boldsymbol{\mathcal{V}}$. *(b)*: Indices of vertices in the cache which are arranged according to the Morton curve. Indices of vertices in the leaf cells with halo, children are ordered with the z-order of their children. This ordering takes care of the refinement operation.

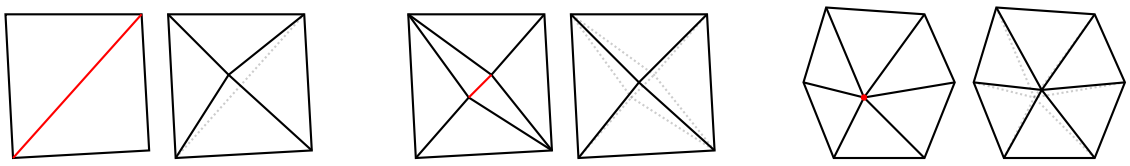


Figure 3.7: **Regridding** : *(a)*Split an edge when the edge size a is more than specified a_{max}, resulting in creation of two new triangles and a vertex. *(b)* Collapse an edge which is smaller than specified a_{min} resulting in the collapsing of two triangles and a vertex. *(c)* Vertex smoothing.

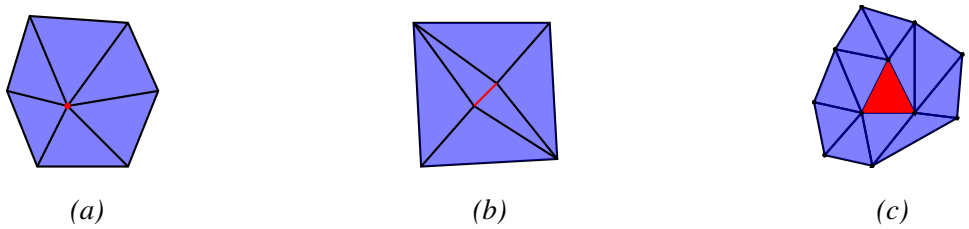


Figure 3.8: **Valence** : *(a) Valence of the vertex*: The valence triangles of a vertex is the set of triangles that share the vertex. *(b) Valence of the edge*: The valence triangles of an edge is the set of triangles that share either of the vertices of the edge. *(c) Valence of the triangle*: The valence triangles of a triangle is the set of triangles that share at least one of the vertex of the triangle.

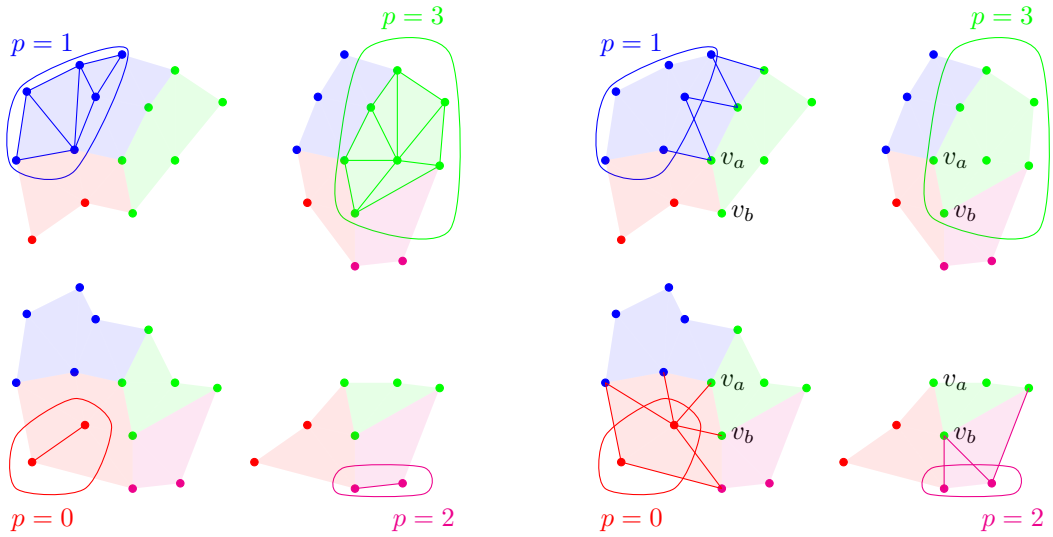
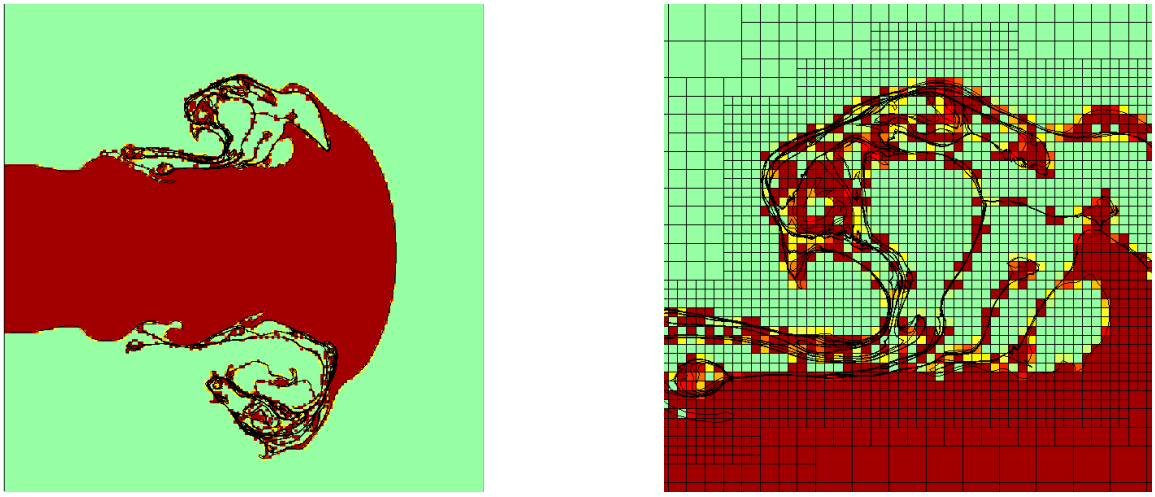
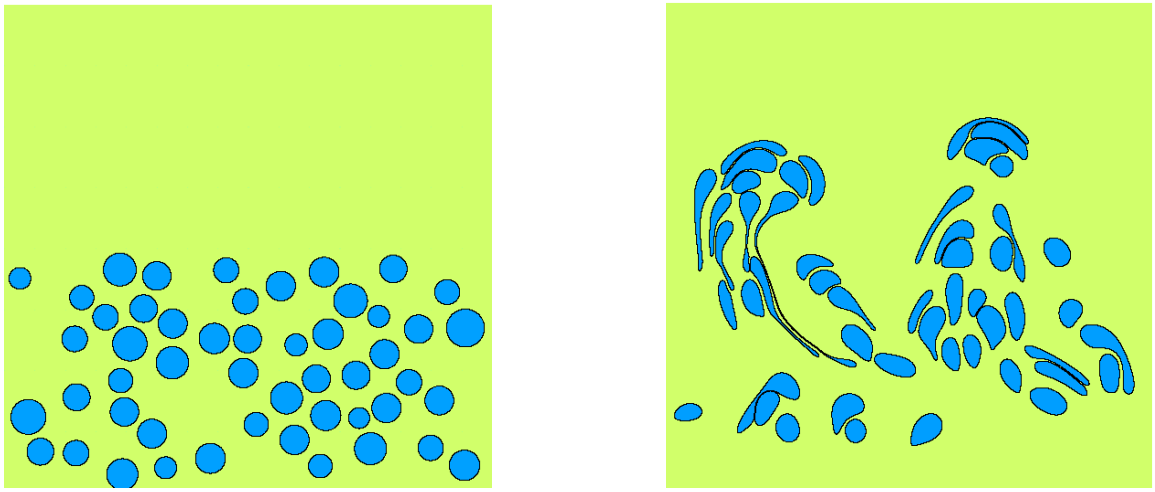


Figure 3.9: Splitting of edge operation in parallel to avoid race condition: *(a)* In the first step, the edge operation (split/collapse) is carried out only on the *inside edge* ($\{^{\cup}l\}_p \setminus \texttt{cut}(\Pi_{\mathcal{V}})$) which completely lies inside the domain. *(b)* In the following step, the edge operation is carried out on local edges, which is also a cut-edge ($\{^{\cup}l\}_p \bigcap \texttt{cut}(\Pi_{\mathcal{V}})$). **NOTE**: The second step has to be further split into two steps so that two processors don't split/merge edges of the same triangle which contains *triple point* (the points that are shared by t v_a, v_b)



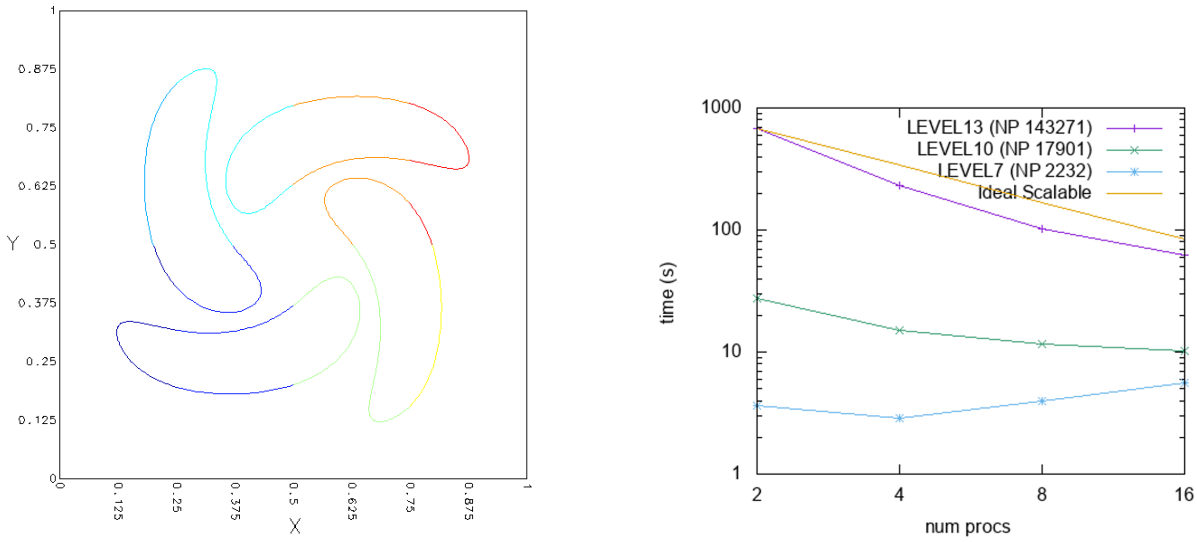
(a) *(b)*

Figure 3.10: *(a)*: A snapshot of a 2D atomization simulation implemented using parallel front tracking method. In the simulation, a jet of radius $R = 1/12\ m$ is injected with a pulsatile velocity $U = 0.1 + 0.05\sin(2\pi t/T)\ ms^{-1}$ with time period $T = 0.1\ s$. The Reynold's number is $Re = 5800$, surface tension $\sigma = 3 \times 10^{-5}$, density ratio is $\rho_1/\rho_0 = 2.84$, viscosities are $\mu_1 = 2.\ U\ R/(\rho_1\ Re), \mu_0 = 2.\ U\ R/(\rho_0\ Re)$. *(b)*: Shows the AMR capability. The heaviside used in the simulation is simply the volume fraction calculated from the front, which is discussed in the Ch:4 Sec:4.2.2.



(a) *(b)*

Figure 3.11: Two snapshots of a cluster of rising bubbles in 2D. In front tracking the topology change is not automatic, as is the case with VoF.



(a) *(b)*

Figure 3.12: Advection Test case: *(a)*: Four circles are stretched with a predefined velocity field. *(b)*: Scalability test run on a workstation, for different refinement levels.

Chapter 4

Balanced Surface Tension

4.1 Introduction

This chapter explains the implementation of a *balanced* surface tension that eliminates *spurious current* and conserves momentum. Well-balanced surface tension [59] [60] models discretize terms in the Navier-Stokes equation such that in static droplet and bubble test cases, the Laplace equilibrium is achieved, i.e., both the pressure and the surface tension discretization terms are in balance so that spurious or parasitic currents are avoided. The desirable properties while spatially discretizing surface tension terms are

- Well-balancedness: The pressure gradient balances the discrete surface tension in static configurations such that:
$$-\nabla_h p + \boldsymbol{f}_h^\sigma = 0 \tag{4.1}$$
- Discrete momentum conservation, Which is equivalent to
$$-\boldsymbol{f}_h^\sigma = \nabla_h \cdot \boldsymbol{T}_h^\sigma \tag{4.2}$$
where $\boldsymbol{T}_h^\sigma$ is an appropriately defined capillary tensor.

Earlier approaches to surface tension discretization using CSF [33] suffered from *parasitic current* or *spurious current* in stationary bubble or droplet test cases [61] [62]. In those particular test cases where the Laplace pressure difference is balanced by the surface tension, the NS equation can be written as

$$[[p]] = -\sigma(\boldsymbol{x}^s)\kappa(\boldsymbol{x}^s) \quad \forall \ \boldsymbol{x}^s \in \Gamma, \tag{4.3}$$

which can be expressed in the spatiotemporal discretization equation [63],

$$\nabla_h(p^n - \sigma\kappa_h^n H_h^n) = 0. \tag{4.4}$$

In the case of the spherical droplet at equilibrium, it reduces to [63]

$$p_{eqb} = \sigma\kappa_{eqb}H_h + C. \tag{4.5}$$

In the front tracking method, the heaviside H_h in the above equation can be derived using Eq:4.15. The two aspects of a well-balanced discretization requires that *(i)* the numerical evaluation of curvature is constant, and *(ii)* the gradient operators used in both terms (Eq:4.15) are similar or compatible [60] [59]. Well-balanced and momentum-conserving discretization

methods are implemented in VoF [60] and levelset [59] solvers. Implementation in [59] is similar to the front tracking code in [60] which employs an integral formulation of surface tension on the a portion of the interface reconstructed using a cubic spline interpolation of marker points. In front tracking, the advection of markers produces small amplitude, high-frequency errors in the curvature. In this chapter, we employ few Laplacian smoothing cycles to take care of the the curvature fluctuations (Refer Sec:4.2.4).

4.2 Numerical Implementation

The Navier-Stokes equations (Eq:2.9-2.13) are solved using the projection method implemented in Basilisk [29] and its extension to compressible multiphase flow [64] is explained in Sec:5.3 in the next chapter. Spatial discretization of momentum equation is done on an Marker-And-Cell staggered grid with AMR capability [29]. Discretization of surface tension term uses volume integral formulation similar to [62].

4.2.1 Interface

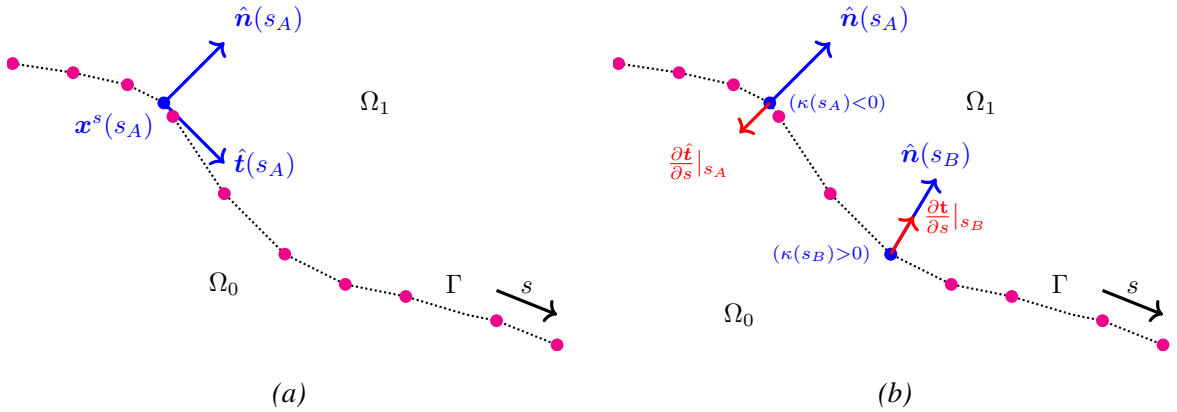


Figure 4.1: *(a)* parameter s, arc length $\theta(s)$, tangent $\boldsymbol{t}$, normal $\boldsymbol{t}$ *(b)* positive ($\kappa_a < 0$)and negative curvature($\kappa_b > 0$)

The 2D interface $\Gamma(t)$ at time t is the locus of interfacial points $\boldsymbol{x}^s(s,t)$ for arc length $s \in [0, S]$ and $t \in [0, T]$. In discretized computation, we would rather have a discrete set of marker points $\boldsymbol{\mathcal{V}} := \{\boldsymbol{x}_0^s, \boldsymbol{x}_1^s, ..\}$ at any time step t^n,. If we assume, the vertices are ordered cyclically, then the set of front element can be considered as $\boldsymbol{\mathcal{E}} := \{(\boldsymbol{x}_0^s, \boldsymbol{x}_1^s), (\boldsymbol{x}_1^s, \boldsymbol{x}_2^s), ..\}$. Let us define a polynomial interpolation function $M_i(m) : [0,1] \mapsto \mathbb{R}^2$ corresponding to each element $e_i := (\boldsymbol{x}_i^s, \boldsymbol{x}_{i+1}^s) \in \boldsymbol{\mathcal{E}}$, that maps parameter $m \in [0,1]$ to points on the interface with all the polynomials satisfying $M_i(1) = M_{i+1}(0) = \boldsymbol{x}_{i+1}^s$. Then, the interface can be considered as

$$\Gamma = \left\{ M_i(m) \mid m \in [0,1] \text{ and } i \in \mathbb{N}_{|\boldsymbol{\mathcal{V}}|} \right\} \tag{4.6}$$

The tangent at interfacial point $\boldsymbol{x}_i^s$ is

$$\hat{\boldsymbol{t}}\big|_{\boldsymbol{x}_i^s} = \left. \frac{dM_i/dm}{\|dM_i/dm\|} \right|_{t=0} \tag{4.7}$$

For C^1 (tangent) continuity

$$\left.\frac{dM_i/dm}{\|dM_i/dm\|}\right|_{t=1} = \left.\frac{dM_{i+1}/dm}{\|dM_{i+1}/dm\|}\right|_{t=0}$$

The direction of s is chosen such that points to the left and right of $\hat{\boldsymbol{t}}(\boldsymbol{x}^s)$ belongs to Ω_1 and Ω_0 respectively. So from the above convention, normal to the interface is obtained by rotating $\hat{\boldsymbol{t}}(s)$ counter clock-wise 90 degrees.

$$\hat{\boldsymbol{n}} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \hat{\boldsymbol{t}}$$

The arc length from $\boldsymbol{x}_0^s$ to $\boldsymbol{x}_i^s$ along the interface is

$$s_i = \sum_{j=0}^{i-1} \int_0^1 \left\| \frac{dM_j(m)}{dm} \right\| dm$$

For any point $\boldsymbol{x}^s \in \Gamma$ on the interface corresponding to the element $e_i := (\boldsymbol{x}_i^s, \boldsymbol{x}_{i+1}^s)$, all the geometric parameters can be expressed in terms of the local parameter $t \in [0, 1]$, or the arc length $s \in [s_i, s_{i+1}]$ or the coordinate $\boldsymbol{x}^s$ itself depending on the convenience. Here, let us define them in terms of t

$$\boldsymbol{x}^s(m) := M_i(m) \tag{4.8}$$

$$s(m) := s_i + \int_0^t \left\| \frac{dM_i(m)}{dm} \right\| dm \tag{4.9}$$

$$\hat{\boldsymbol{t}}(m) = \frac{dM_i(m)/dm}{\|dM_i(m)/dm\|} \tag{4.10}$$

$$\kappa(m)\hat{\boldsymbol{n}}(m) = \frac{d\hat{\boldsymbol{t}}(m)}{ds(m)} = \frac{1}{\|dM_i(m)/dm\|} \frac{d\hat{\boldsymbol{t}}(m)}{dm} \tag{4.11}$$

Advection of the Interface

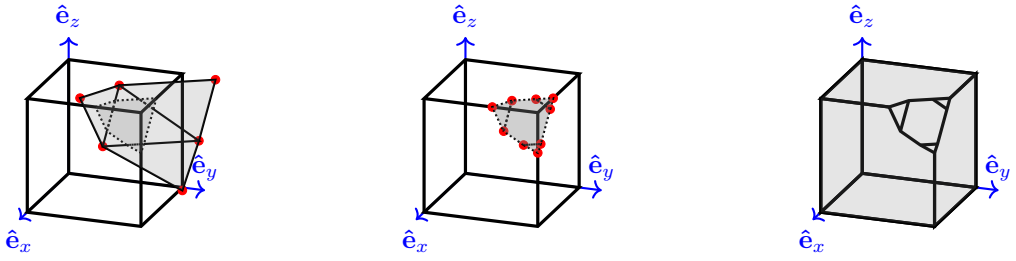
The marker points are advected using explicit integration ($\boldsymbol{x}^s \leftarrow \boldsymbol{x}^s + \Delta t\ \boldsymbol{u}^s(\boldsymbol{x}^s)$) where marker velocity ($\boldsymbol{u}^s(\boldsymbol{x}^s)$) is calculated from grid velocity ($\boldsymbol{x}_c$) using linear interpolation of velocity components.

4.2.2 One fluid formulation

In one-fluid formulation, we solve the set of discretized equations of both fluids, considering them as "one-fluid", allowing both the set of NS equations corresponding to both liquids into a single set of equations. The fluid properties such as viscosity and density of the "one-fluid" are determined by averaging the fluid properties of individual components. In this work, we follow the arithmetic averaging [29] [32] [53] as

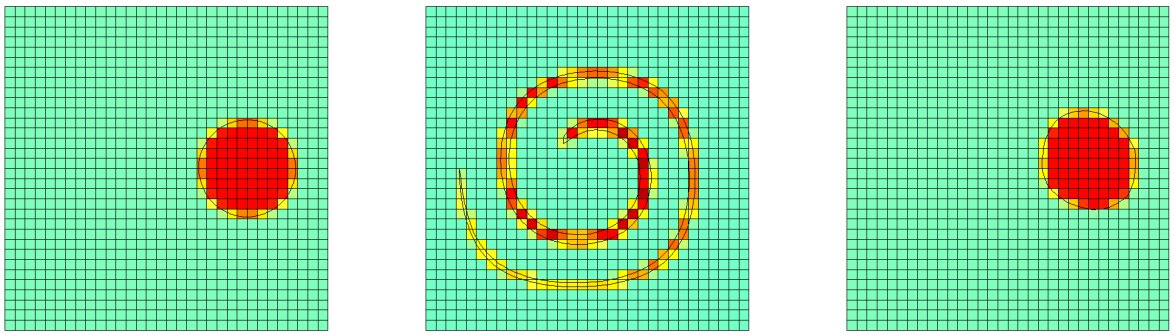
$$\rho(\boldsymbol{x}) = \rho_1 H(\boldsymbol{x}) + \rho_2(1 - H(\boldsymbol{x})) \tag{4.12}$$

$$\mu(\boldsymbol{x}) = \mu_1 H(\boldsymbol{x}) + \mu_2(1 - H(\boldsymbol{x})) \tag{4.13}$$



(a) *(b)* *(c)*

Figure 4.2: Heaviside $H_h(\boldsymbol{x})$ in each control volume is taken as the volume fraction in the control volume. *(a)* A cubic CV and all the triangular facets that intersect them *(b)* Polygons: Subset of each triangle inside the cube *(c)* Void Fraction: Subset of cubic control which belongs to Ω_0



(a) *(b)* *(c)*

Figure 4.3: Void fraction evaluated from an interface front at three different timesteps. The interface is advected with a velocity field (given by stream function $\psi(x,y) = \frac{1}{\pi}\sin^2(\pi x)\sin^2(\pi x)\cos(\pi\frac{t}{T})$). The algorithm 11 can correctly calculate even if the interface is highly stretched such that there are cells with front elements of opposite orientation (*(b)*).

where $H(\boldsymbol{x})$ is the heaviside function which is given by

$$H(\boldsymbol{x}) = \begin{cases} 0 & \forall\, \boldsymbol{x} \in \Omega_0^o \;\; (\text{interior points of } fluid-0, \Omega_0^o = \Omega_1 \setminus \Gamma) \\ 1 & \forall\, \boldsymbol{x} \in \Omega_1^o \;\; (\text{interior points of } fluid-1, \Omega_1^o = \Omega_1 \setminus \Gamma) \end{cases} \tag{4.14}$$

which is discontinous at the interface i.e $\forall\, \boldsymbol{x} \in \Gamma$.

There are many ways to determine the heaviside function. In most of the FT multiphase flow solvers [53] [2], the heavy side function is determined by solving the following Poisson equation

$$\nabla^2 H_h = \nabla \cdot \boldsymbol{G}_h \tag{4.15}$$

where $H_h(\boldsymbol{x}_c)$ is the discretized heaviside evaluated in a cell c, and

$$\boldsymbol{G}_h(\boldsymbol{x}_c) = \sum_{e \in \boldsymbol{\mathcal{E}}} \hat{\boldsymbol{n}}_e \delta_h(\boldsymbol{x_c} - \boldsymbol{x}_e^s) \frac{A_e}{|\Omega_c|} \tag{4.16}$$

where $e \in \boldsymbol{\mathcal{E}}$ represent front elements, $\hat{\boldsymbol{n}}_e$ is the normal evaluated at the centroid of the element, A_e is the area of the element, $|\Omega_c|$ is the volume of the control volume corresponding to cell c. The solution of Eq:4.15 doesn't guarantee H_h falls in $[0,1]$ and can also result in erronous

heaviside value in the cells far from the interface [39]. Inorder to avoid these errors, special care has to be taken in the Poisson solver like avoiding cells farther from interface [39]. This method of solving for heaviside requires additional care while two interfaces approaches.

However, in this work, we take the void fraction as the heaviside, which is calculated directly from the facets like in [62].

$$H_h(\boldsymbol{x}) = \frac{1}{\Delta V}\int_{d\Omega} H(\boldsymbol{x})dv = f \tag{4.17}$$

where f is the fraction of volume occupied by $fluid-1$ in the computation cell. Fig:4.3 illustrates an example where the heaviside is evaluated as the volume fraction occupied by reference fluid. [62] also used the same approach in 2D front tracking code. A general algorithm (for both 2D and 3D) to evaluate void fraction in a control volume is explained in detail in the appendix (Sec:A.5). A similar algorithm is described in [65] which is implemented in [53]. This method avoidis overshoots and undershoots while evaluating heaviside as long as there is no invalid interface intersections. However, since C^0 continuity of the interface is used in the evaluation, an additional filtering or Gaussian smoothing algorithm for smoother heaviside is recommended before evaluating density and viscosity. The algorithm described doesn't require stitching of polygons as in [65].

4.2.3 Balanced FT in 2D

In the staggered stencil, the finite volume integral of the x component of the momentum equation is carried out on the face-centered control volume Ω_{cx} (4.4),

$$\hat{\boldsymbol{e}}_x \cdot \boldsymbol{f}_\sigma(\boldsymbol{x}_{cx}) = \frac{1}{|\Omega_{cx}|}\int_{\Omega_{cx}} dV \int_\Gamma \delta(\boldsymbol{x}^s - \boldsymbol{x})\hat{\boldsymbol{e}}_x \cdot \boldsymbol{F}_\sigma(\boldsymbol{x}^s)dA \tag{4.18}$$

$$= \frac{1}{|\Omega_{cx}|}\int_{\Omega_{cx}\cap\Gamma} \sigma\kappa\hat{\boldsymbol{n}} \tag{4.19}$$

Referring to Fig:4.4 and Eq:4.19, and using Frenet-Serret relation, the integral of $-\nabla p + \boldsymbol{f}_\sigma$ can be expressed as

$$\hat{\boldsymbol{e}}_x \cdot \int_{\Omega_{cx}} (-\nabla p + \boldsymbol{f}_\sigma)dv = -\int_C^D p(y)\,dy + \int_B^A p(y)\,dy + \int_{B'}^{C'} \frac{\partial \sigma\hat{\boldsymbol{t}}}{\partial s}\cdot\hat{\boldsymbol{e}}_x\,ds \tag{4.20}$$

By taking care of Laplace pressure jump at B' and C', the above equation gives

$$\begin{aligned}\frac{1}{|\Omega_{cx}|}\int_{\Omega_{cx}} (-\frac{\partial p}{\partial x} + \boldsymbol{f}_{\sigma,x})dv &= -\frac{1}{h^2}\Big(|C'D|\,p_{i,j} + |CC'|\,(p_{i,j} + \sigma\kappa|_{C'})\Big) \\ &\quad + \frac{1}{h^2}\Big(|BB'|\,p_{i-1,j} + |B'A|\,(p_{i-1,j} - \sigma\kappa|_{B'})\Big) \\ &\quad + \frac{1}{h^2}\Big(\sum_{e_i}\int_{m_0(e_i)}^{m_1(e_i)} \frac{\partial\sigma\hat{\boldsymbol{t}}}{\partial s}\cdot\hat{\boldsymbol{e}}_x\,ds\Big),\end{aligned} \tag{4.21}$$

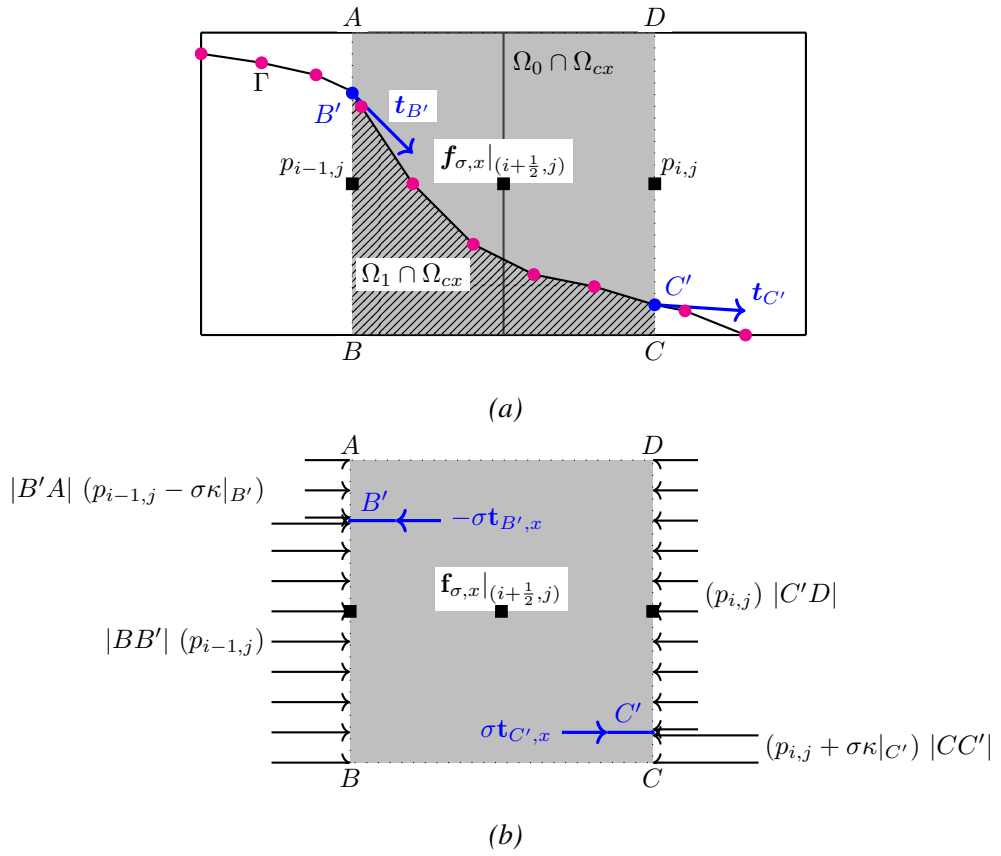


Figure 4.4: Finite volume integral: *(a)* In the MAC-staggered stencil, pressure $p^n_{i,j}$ is evaluated at the centroid of the control volume Ω_c corresponding to the cell $c := (i, j, l)$ (where i and j are integer index corresponding to x and y direction and l is the refinement level of cell). The x component of the momentum equation is integrated over the face-centered control volume Ω_c and the volume averaged x component of surface tension $\boldsymbol{f}_{\sigma,x}$ is calculated at the face center. *(b)* Integral of $\int_{\Omega_{cx}} \left(-\frac{\partial p}{\partial x} + \boldsymbol{f}_{\sigma,x}\right) dv$ over the control volume Ω_{cx}

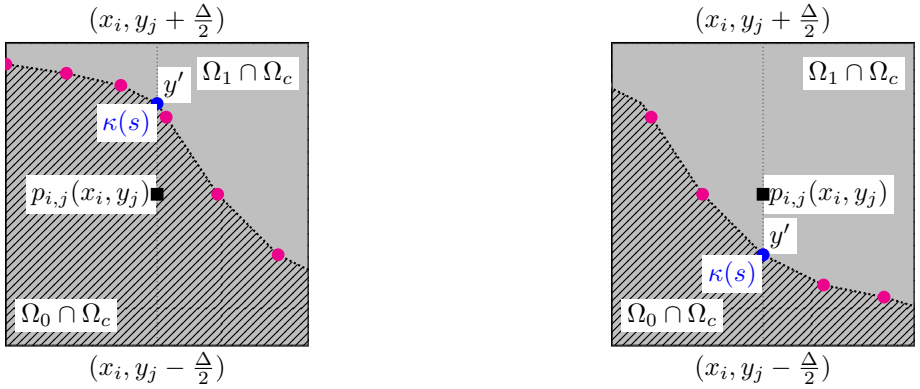
where the third term is the summation of elements $e_i := (\boldsymbol{x}^s_i, \boldsymbol{x}^s_{i+1}) \in \boldsymbol{\mathcal{E}}$ which intersects with the control volume Ω_{cx}, and m_0 and m_1 are found using Liang-Barsky algorithm ($0 \leq m_0 \leq m_1 \leq 1$) while cutting line segment $(\boldsymbol{x}^s_i, \boldsymbol{x}^s_{i+1})$ by the control volume. This term can be rewritten as

$$\sum_{e_i} \int_{m_0(e_i)}^{m_1(e_i)} \frac{\partial \sigma \hat{\boldsymbol{t}}}{\partial s} \cdot \hat{\boldsymbol{e}}_x \, ds = \sum_{e_i} [m_1(e_i) - m_0(e_i))](\sigma_{i+1}\hat{\boldsymbol{t}}_{i+1} - \sigma_i \hat{\boldsymbol{t}}_i) \cdot \hat{\boldsymbol{e}}_x \tag{4.22}$$

We finally have the discretised equation for Eq:4.4

$$\begin{aligned} \frac{1}{|\Omega_{cx}|} \int_{\Omega_{cx}} (-\frac{\partial p}{\partial x} + \boldsymbol{f}_{\sigma,x}) dv \quad &= \quad -\frac{1}{h}\left(p_{i,j} - p_{i-1,j}\right) - \frac{1}{h}\left(p^{cap,x}_{i,j} - p^{cap,x}_{i-1,j}\right) \\ &\quad + \sum_{e_i} [m_1(e_i) - m_0(e_i))](\sigma_{i+1}\hat{\boldsymbol{t}}_{i+1} - \sigma_i \hat{\boldsymbol{t}}_i) \cdot \hat{\boldsymbol{e}}_x \end{aligned} \tag{4.23}$$

where the capillary pressure correction in the x-component of the momentum equation ($p^{cap,x}_{i,j}$)



(Case A) (Case B)

Figure 4.5: Capillary pressure correction to $p_{i,j}^{cap,x}$. Two cases where the interface intersects the right face of the control volume Ω_{cx}, above (x_i, y_j) as in Case A and below (x_i, y_j) as in Case B, are represented here.

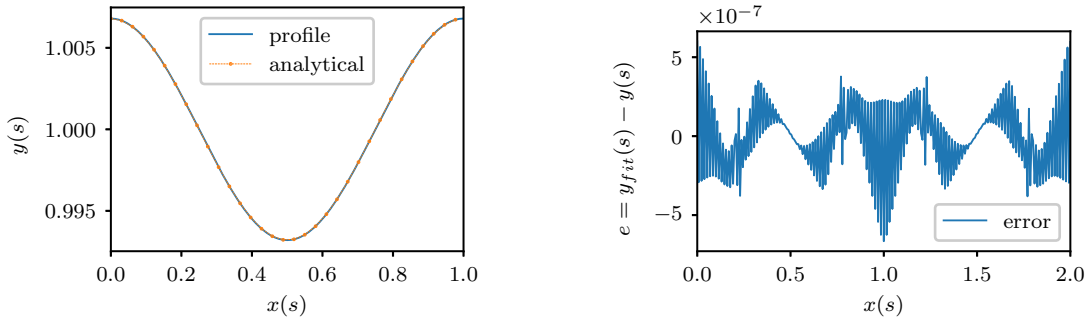
can be written as

$$p_{i,j}^{cap,x} = \begin{cases} \sigma(y')\kappa(y')(\frac{1}{2} - \frac{|y'-y_i|}{h}) & \text{if } y' \geq y_i. \text{ Case A in Fig:4.5} \\ -\sigma(y')\kappa(y')(\frac{1}{2} - \frac{|y'-y_i|}{h}) & \text{if } y' < y_i. \text{ Case B in Fig:4.5} \\ 0 & \text{if } \Gamma \text{ doesn't intersect the right face of } \Omega_{cx} \end{cases} \tag{4.24}$$

where $y' \in (y - \frac{h}{2}, y + \frac{h}{2})$ is the point at which the interface intersect the right face of the Ω_{cx}, and $\sigma(y')\kappa(y')$ is interpolated from the marker points. Similarly $p_{i,j}^{cap,y}$ can be derived.

4.2.4 Smoothing of curve

In front tracking, rather immersed boundary methods in general, advection of markers produces small amplitude, high frequency errors in the marker positions. Thus, interface forces involving higher-order marker position derivatives also have errors that can create numerical instability. So we employ a smooth- ing algorithm, which ensures that all the terms involved in the computation of the interface forces are smooth and have second-order convergence.



(a) (b)

Figure 4.6: (a)Instance of the profile of an initially perturbed damping wave compared with the analytical solution of Prosperetti. (b) High-frequency error in the numerical solution when solved with a front-tracking solver without smoothing.

Fig:4.6 shows the fluid-fluid interface and Fig:4.9(a) shows the error from the analytical

solution, which is a smooth sine wave. This small high-frequency error gives rise to errors in calculating curvature and their derivatives and potentially makes the integration of the system of equations in time unstable.

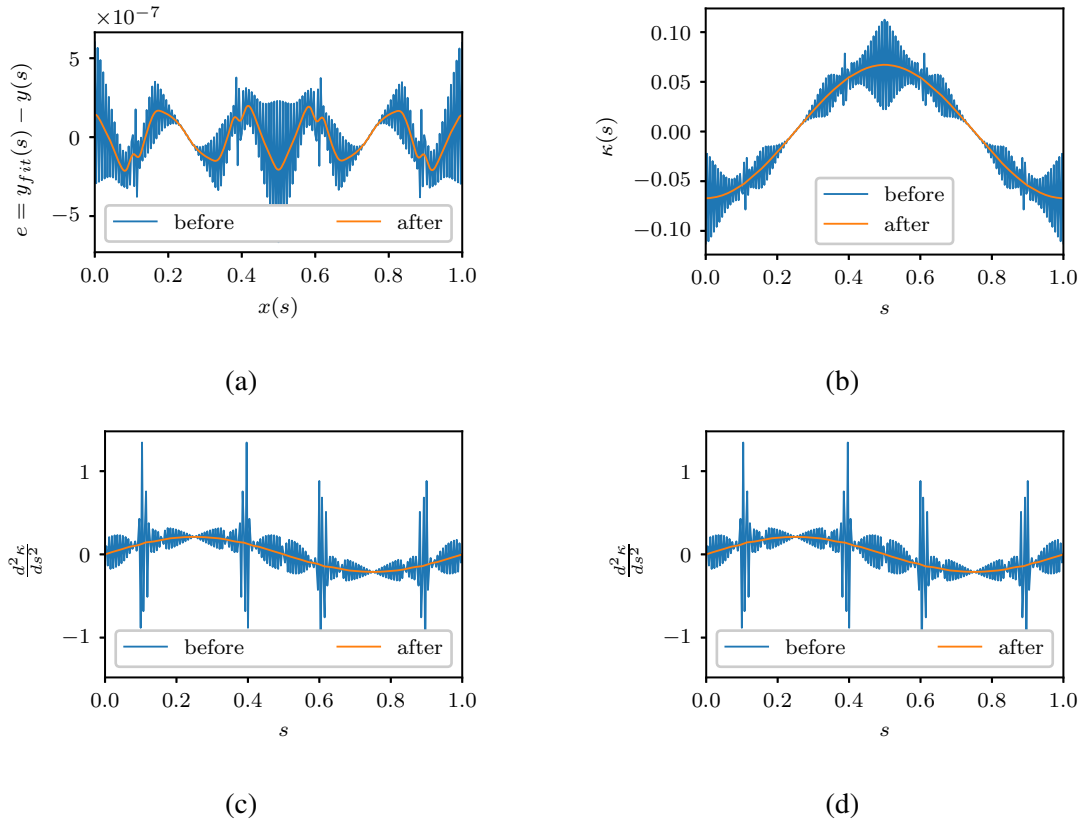


Figure 4.7: (a) error in interface position (b) curvature (c) the first derivative of curvature and (c) second derivative of curvature before and after a few cycles of smoothing cycles

There are different approaches to make the interface smooth. Here, we do a Laplacian smoothing [66] [58], where we do pseudo-time step integration of type

$$\boldsymbol{u}(s) = -c_{VN}\frac{d^2\kappa(s)}{ds^2}\boldsymbol{n}(s) \tag{4.25}$$

with constant c_{VN} can be determined from Von-Neumann stability analysis of the differential equation. c_{VN} can be determined as $\frac{1}{8(\Delta s)^4}$. In Fig:4.9 it is evident how few number of cycles of smoothing can kill the high-frequency errors.

4.3 Testcases

This section deals with the results of benchmarking test cases done with the front-tracking multiphase solver described in this chapter. Moreover, they are compared with previous studies.

4.3.1 Static Droplet

When a perfectly circular droplet (or spherical droplet in 3D) of heavier fluid is kept in the bulk of lighter fluid in the absence of gravity and other body forces, it will remain at static equilibrium with higher pressure inside the drop, which is balanced by the Laplace pressure. However, in

the direct numerical simulation of this multiphase problem, due to the inconsistency of the terms in the discretization of the Navier-Stokes equation, the unbalanced stresses manifest as unwanted velocity , which is called spurious current or parasitic current in literature.

(a) (b)

Figure 4.8: (a) Computational setup for spurious current testcase in Sec:4.3.1. (b) damping capillary wave in Sec4.3.2

In the subsection, we look into the parasitic current produced in a static droplet test case to look into the well-balancedness of the discretization of NS. A circular droplet of diameter, D, is kept at the center of a square computational domain of side L such that $D/L = 0.4$. The computation is simplified by simulating only $1/4$ of the domain with a quadrant circle placed at the origin. The density and viscosity of the liquid that composes the liquid are ρ_l and μ_l, respectively, and that of the surrounding gas is ρ_g and μ_g The domain is divided to $2^5 \times 2^5$ square control volumes such that there are 12.8 grid points per diameter.

The non-dimensional numbers involved in the problem are

$$\text{Density ratio } r_\rho = \frac{\rho_l}{\rho_g}, \text{Viscosity ratio } r_\mu = \frac{\mu_l}{\mu_g}, \text{Laplace number } La = \frac{\rho_l \sigma D}{{\mu_l}^2}$$

The time scale involved in the problem is capillary time scale $T_\sigma = \sqrt{\frac{\rho_l D^3}{\mu}}$ and viscous time scale $T_\mu = \frac{\rho_l D^2}{\mu}$.

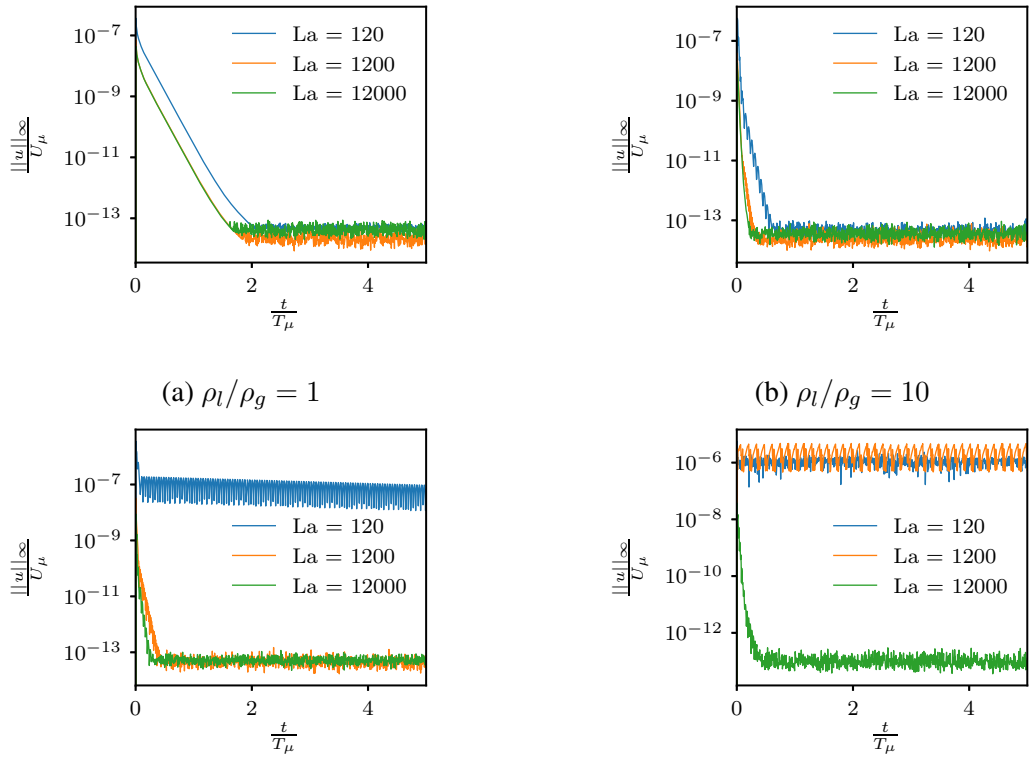
4.3.2 Capillary wave

A well-established test case used to look into the robustness and study the order of convergence of spatial discretization of a multiphase flow involving viscous stresses and capillary forces is the damped oscillation of a capillary wave [67] [68] [69][20] [63][24][70]. The test case involves studying the evolution of an initially perturbed two-fluid interface and comparing it with the interface initializing a small perturbation on

An initially perturbed interface that lies at the center of the square box: The wavelength of the wave is equal to the side length of the square. The ratio of the amplitude of perturbation to the wavelength is 0.01. The densities and viscosities of fluids on both sides of the interface are equal. Let $Oh = 1/\sqrt{3000}$, the non-dimensional viscosity $\epsilon = \nu k^2/\omega_0 \approx 6.472 \times 10^{-2}$. The square box is divided into 64x64 cells. Surface forces at the interface in the perturbed interface dampen the wave with time. The evolution of amplitude a/λ is plotted with non-dimensional time in Fig:4.13 and is compared with the analytical solution of Prosperetti [68] [71].

Case-A

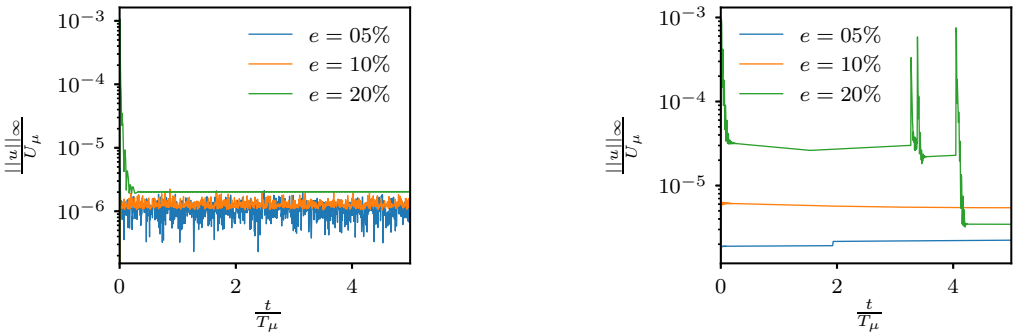
The evolution of the amplitude of a capillary wave is shown in Fig:4.13 in the test case with dimensional numbers listed in Table4.1.



(a) $\rho_l/\rho_g = 1$ (b) $\rho_l/\rho_g = 10$

(c) $\rho_l/\rho_g = 100$ (d) $\rho_l/\rho_g = 1000$

Figure 4.9: Decaying spurious current study for a static droplet test case with different ratios and Laplace numbers explained in Sec:4.3.1. The x-axis represents time, which is non-dimensionalised with viscous time scale and the y-axis is the maximum of the spurious velocity in the domain, which is non-dimensionalized with a viscous velocity scale.



(a)$La = 120$ (c)$La = 12000$

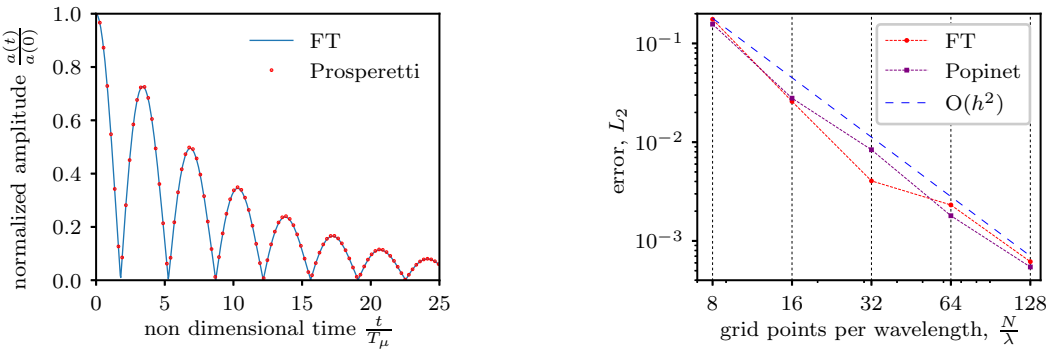
Figure 4.10: Eccentricity introduced

$\frac{\mu_l}{\mu_g}$	$\frac{\rho_l}{\rho_g}$	La	$\frac{a_0}{\lambda}$
1	1	3000	0.01

Table 4.1: Nondimensional parameters for damping capillary wave test case A

Case-B

The evolution of the amplitude of a capillary wave is shown in Fig:4.12 in the test case with dimensional numbers listed in Table4.2.

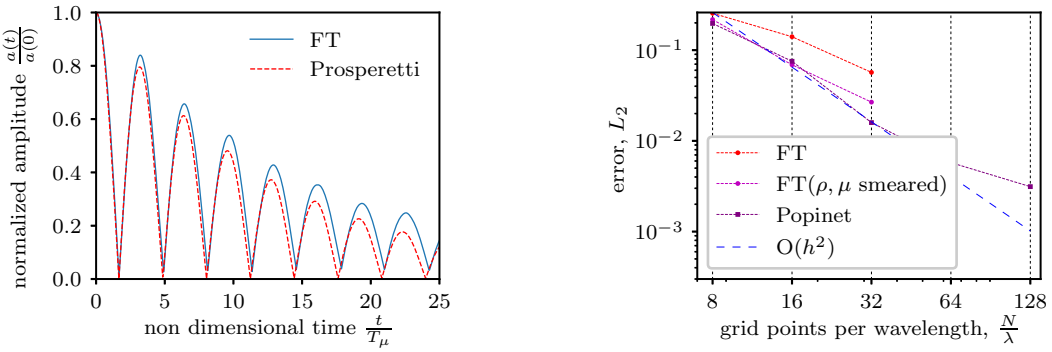


(a) (b)

Figure 4.11: (a) Evolution of normalized amplitude of an initially perturbed wave as mentioned in *Case-A* of Sec:4.3.2. (b) The spatial convergence of L_2 error

$\frac{\mu_l}{\mu_g}$	$\frac{\rho_l}{\rho_g}$	La	$\frac{a_0}{\lambda}$
55.72	850	3000	0.01

Table 4.2: Nondimensional parameters for damping capillary wave test *Case-B* of Sec:4.3.2



(a) (b)

Figure 4.12: (a) Evolution of normalized amplitude of an initially perturbed wave as mentioned in testcase of Sec:4.3.2. (b) The spatial convergence of L_2 error.

Spatial Convergence

Let us look into the error between the exact solution and the DNS solution for the current front-tracking solver for different resolution of the grid and look into the converge. We define the L_2 error as

$$L_2 = \frac{1}{\lambda}\sqrt{\frac{\Omega_0}{25}\int_{t=0}^{25/\Omega_0}(h - h_{exact})^2 dt}$$

which is the difference in the amplitude of the damped wave between the DNS and the exact solution, which is averaged over a time period equals to $25/\omega_0$. The frequency Ω_0 is obtained from

$${\omega_0}^2 = \frac{\sigma k^3}{2\rho} \text{ where } k = \frac{2\pi}{\lambda} \text{ is the wave number}$$

Nondimensional parameters for damping capillary wave test case B Convergence of L_2 is studied by doing? the simulation for different grid resolution $\lambda/\delta = 8, 16, 32, 64, 128$. The error and convergence study is compared with [60].

Study / $\frac{\lambda}{\Delta_h}$	8	16	32	64	128
current study	0.17561	0.0258687	0.0040554	0.00231449	0.000616792
Popinet[60]	0.1568	0.0279	0.00838	0.0018	0.000545
Popinet&Zaleski	0.3018	0.0778	0.0131	0.0082	0.00645

Table 4.3: L_2 error for different / $\frac{\lambda}{\Delta_h}$ (*Case-A* of Sec:4.3.2).

Study / $\frac{\lambda}{\Delta_h}$	8	16	32
current study	0.255214	0.140576	0.056885
Popinet[60]	0.1971	0.0754	0.0159

Table 4.4: L_2 error for different / $\frac{\lambda}{\Delta_h}$ (*Case-B* of Sec:4.3.2).

4.3.3 Oscillating Droplet

Like the damping capillary wave test case in the above subsection 4.3.2, A small perturbation is introduced on the circular interface separating two inviscid liquids, and the system is let evolve in time [72] [44]. Due to the surface tension, the interface oscillates. Theoretically, the droplet should oscillate without damping as both liquids are inviscid. However, the direct numerical simulation shows that the perturbation numerically disappears in time.

An inviscid circular drop made up of a liquid of viscosity ρ_l with diameter, D is kept inside another inviscid liquid of density ρ_g. The surface tension for the pair of liquids is σ. A sinusoidal perturbation of amplitude ϵ_0 is introduced on the interface at $t = 0$.

$$r(\theta) = \frac{D}{2} + \epsilon_0 cos(n\theta)$$

For this current test case, n is taken as 2. Theoretically, the perturbation gives rise to an undamped oscillation of frequency

$$\omega_n^2 = \frac{(n^3 - n)\sigma}{(\rho_l + \rho_g)(D/2)^3}$$

$\frac{\rho_l}{\rho_g}$	La	$\frac{\epsilon}{D}$
1000	∞	0.05

Table 4.5: Nondimensional parameters for oscillating inviscid drop

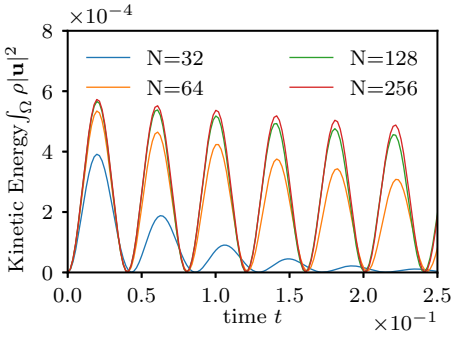


Figure 4.13: Evolution of normalized amplitude of an oscillating inviscid drop of small amplitude

Chapter 5

Compressible Multiphase Flow using Front Tracking

5.1 Introduction

This chapter explains the implementation of a compressible multiphase flow solver using the front tracking method. The general algorithm regarding *all-mach* multiphase flow solver follows [64], who have implemented the solver in AMR-based solver Basilisk [29]. [64] is a VoF-based finite volume solver that considers the interfacial tension and the compressibility of both fluids.

5.2 Compressible Flow Solver

The compressible flow solver aims to solve the evolution of the interface and the subspaces that occupy each fluid, the flow field variables, and the intrinsic properties of fluids, which, unlike an incompressible flow, can vary. For a system of two-component fluid occupying a d-dimensional computational domain $\Omega \subset \mathbb{R}^d$ ($d \in \{2, 3\}$) with computational surface $\partial\Omega$, the objective of this solver is to look into the following,

$$\begin{aligned} \Gamma(t), \Omega_0(t), \Omega_1(t) \quad &\forall \quad t \in [0, T] \subset \mathbb{R} \\ \boldsymbol{u}_i(\boldsymbol{x}, t) \quad &\forall \quad t \in [0, T] \text{ and } \boldsymbol{x} \in \Omega_i(t) \setminus \Gamma(t) \\ p_i(\boldsymbol{x}, t), \rho_i(\boldsymbol{x}, t), e_i(\boldsymbol{x}, t) \quad &\forall \quad t \in [0, T] \text{ and } \boldsymbol{x} \in \Omega_i(t) \setminus \Gamma(t) \end{aligned}$$

where $\Gamma(t)$ is an oriented closed surface that represents the interface separating two fluids, say $fluid - 0$ and $fluid - 1$, which respectively occupy the domains $\Omega_0(t) \setminus \Gamma(t)$ and $\Omega_1(t) \setminus \Gamma(t)$ such that $\Omega_0(t) \cap \Omega_1(t) = \Gamma(t)$ and $\Omega_0(t) \cup \Omega_1(t) = \Omega(t)$. The index $i \in \{0, 1\}$ in the above description represents the i-th component of the system. $\boldsymbol{u}$ stands for the velocity field. The thermodynamic properties of fluid, pressure, density, and specific internal energy are represented by p, ρ, and e, respectively. We do not consider other scalars and vectors that represent flow fields or properties of either component because they are constant for both components or are derivable from the variables listed above.

5.2.1 Governing Equations

We can say that the following ten spatiotemporal variables can fully represent the system of two-component compressible fluids

$$f_i(\boldsymbol{x},t), \rho_i(\boldsymbol{x},t), \boldsymbol{u}_i(\boldsymbol{x},t), p_i(\boldsymbol{x},t), e_i(\boldsymbol{x},t) \;\; \forall \;\; \boldsymbol{x} \;\in\; \Omega_i \text{ and } t \in [0,T] \tag{5.1}$$

If $H(\boldsymbol{x},t)$ is the heaviside with $H(\boldsymbol{x},t) = 0 \; \forall \; \boldsymbol{x} \in \Omega_0(t)$ and $H(\boldsymbol{x},t) = 1 \; \forall \; \boldsymbol{x} \in \Omega_1(t)$ then the local heaviside f_i is given by

$$f_i(\mathbf{x},\mathbf{t}) = \begin{cases} 1- & H(\mathbf{x},\mathbf{t}) \; \forall \; \mathbf{x} \in \boldsymbol{\Omega_0}(\mathbf{t}) \\ & H(\mathbf{x},\mathbf{t}) \; \forall \; \mathbf{x} \in \boldsymbol{\Omega_1}(\mathbf{t}) \end{cases} \tag{5.2}$$

The Navier-Stokes equations that define the governing system are

$$\frac{\partial}{\partial t}\rho_i + \boldsymbol{\nabla} \cdot (\rho_i \boldsymbol{u}_i) \;=\; 0 \tag{5.3}$$

$$\frac{\partial}{\partial t}(\rho_i \boldsymbol{u}_i) + \boldsymbol{\nabla} \cdot (\rho_i \boldsymbol{u}_i \boldsymbol{u}_i) \;=\; -\boldsymbol{\nabla} p_i + \boldsymbol{\nabla} \cdot \boldsymbol{\tau}_i \tag{5.4}$$

$$\frac{\partial}{\partial t}\big(\rho_i(e_i + \frac{1}{2}\boldsymbol{u}_i^2)\big) + \boldsymbol{\nabla} \cdot \big(\rho_i(e_i + \frac{1}{2}\boldsymbol{u}_i^2)\, \boldsymbol{u}_i\big) \;=\; -\boldsymbol{\nabla} \cdot (\boldsymbol{u}_i p_i) + \boldsymbol{\nabla} \cdot (\boldsymbol{\tau}_i \boldsymbol{u}_i) \tag{5.5}$$

where $\boldsymbol{\tau}_i = 2\mu \boldsymbol{\nabla} \frac{1}{2}(\boldsymbol{u} + \boldsymbol{u}^T) - \frac{2}{3}\mu(\boldsymbol{\nabla} \cdot \boldsymbol{u}_i)\boldsymbol{I}$ is the viscous stress tensor and $(e_i + \frac{1}{2}|\boldsymbol{u}_i|^2)$ is the specific total energy of the fluid. The evolution of component spaces, $\Omega_i(t)$ and interface, $\Gamma(t)$ are solved by integrating the following evolution equation of the heaviside function,

$$\frac{\partial}{\partial t} f_i + \boldsymbol{u}_i \boldsymbol{\nabla} \cdot f_i \;=\; 0 \tag{5.6}$$

The equation of state (*EOS*) model used in this work is the stiffened gas EOS model,

$$\rho_i e_i = \frac{p_i + \mathit{\Gamma}_i \mathit{\Pi}_i}{\mathit{\Gamma}_i - 1} \tag{5.7}$$

where $\mathit{\Gamma}_i$ and $\mathit{\Pi}_i$ are fitting parameters specific to each component of fluids.

The 5 pair of equations listed above (Eq:5.3 - Eq:5.7) gives a closure to the system. The above system of equations can be numerically solved, provided sufficient boundary conditions on $\partial\Omega$ and all the initial conditions at $t = 0$ and the jump boundary conditions at $\Gamma(t)$ for each pair of variables in 5.1. Given that there is no mass diffusion and slip at the boundary, we can say that there is no velocity jump at the interface,

$$[[\boldsymbol{u}]]_\Gamma = \mathbf{0}, \tag{5.8}$$

where $[[\;]]_\Gamma$ represents the jump in the variable when we move across the interface Γ from Ω_0 to Γ_1. Say, for a variable ϕ, the jump is

$$[[\phi]]_\Gamma = \lim_{s \to 0^+} \left(\phi(\boldsymbol{x}^s - s\hat{\boldsymbol{n}}(\boldsymbol{x}^s)) - \phi(\boldsymbol{x}^s + s\hat{\boldsymbol{n}}(\boldsymbol{x}^s))\right) \; \forall \; \boldsymbol{x}^s \in \Gamma(t) \tag{5.9}$$

where $\hat{\boldsymbol{n}}(\boldsymbol{x}^s)$ is the normal defined at the interface point $\boldsymbol{x}^s$, $\boldsymbol{x}^s - s\hat{\boldsymbol{n}}$ and $\boldsymbol{x}^s + s\hat{\boldsymbol{n}}$ are points in the subspaces Ω_1 and Ω_0 respectively just interior to Γ (since, by definition, the normal $\hat{\boldsymbol{n}}$ points towards the interior of Ω_0).

In the presence of surface tension, the pressure jump can be written as the sum of jumps in Laplace pressure and viscous stress, which gives

$$[[p]]_\Gamma = -\sigma\kappa + \hat{\boldsymbol{n}} \cdot [[\boldsymbol{\tau}]]_\Gamma \cdot \hat{\boldsymbol{n}} \tag{5.10}$$

The absence of energy transfer across the interface gives

$$[[\boldsymbol{\nabla} e \cdot \boldsymbol{n}_\Gamma]]_\Gamma = 0 \tag{5.11}$$

Jump in local heaviside functions f_i is inherently defined in their definition (Eq:5.2).

Apart from the governing equations (Eq:5.3 -Eq:5.7), the All-Mach algorithm uses some other equations derived from Eq:5.3-5.7. The projection step in Navier-Stokes solver uses the following pressure evolution equation (Eq:A.52)

$$\frac{dp_i}{dt}\left(\frac{\Gamma_i}{\rho_i c_i^2} - \frac{\beta_i^2 T_i}{\rho_i c_{p_i}}\right) = \frac{\beta_i}{\rho c_{p_i}}\Phi - \boldsymbol{\nabla}\cdot\boldsymbol{u} \tag{5.12}$$

to solve for velocity and pressure at t^{n+1}. In the Eq:5.12, $\Phi = \tau_i : \boldsymbol{u}_i$ represents the viscous dissipation c and represents the speed of the sound. For the specific model of the equation of state

$$c_i^2 = \frac{dp_i}{d\rho_i} = \Gamma_i \frac{p_i + \Pi_i}{\rho_i} \tag{5.13}$$

The equation Eq:5.12 can be approximated in the absence of viscous dissipation as

$$\left(\frac{1}{\rho c^2}\right)_e \frac{dp_i}{dt} = \left(\frac{1}{\rho c^2}\right)_e \left(\frac{\partial p_i}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla} p_i\right) = -\boldsymbol{\nabla}\cdot\boldsymbol{u} \tag{5.14}$$

where $\left(\frac{1}{\rho c^2}\right)_e = \left(\frac{\Gamma_i}{\rho_i c_i^2} - \frac{\beta_i^2 T_i}{\rho_i c_{p_i}}\right)$. The derivation of Eq:5.14 is explained in Sec:A.6. Some of the above equations can be decoupled, and the total number of equations and, thus, the variables that define the system uniquely can be reduced. Since $f_0(\boldsymbol{x}, t) = 1 - H(\boldsymbol{x}, t)$ and $f_1(\boldsymbol{x}, t) = H(\boldsymbol{x}, t)$, we can express the pair of equations in Eq:5.6 into a single equation for both subdomains as

$$\frac{\partial}{\partial t} H + \boldsymbol{u}\cdot\boldsymbol{\nabla} H = 0 \qquad \forall\, \boldsymbol{x} \in \Omega(t) \tag{5.15}$$

The index on the velocity is dropped because the velocity has no jump near the interface.

5.2.2 Monolithic Approach

The algorithm is not completely monolithic, as solving the system of equations of both components involves solving both the monolithic equation (where both the fluids are considered as *one-fluid*) and solving separately for each component. For example, the projection algorithm used in the solution of the Navier-Stokes equation is carried out by assuming both fluids as a single fluid while advection of scalars (Eq:5.31) are carried out separately. For this reason, each control cells store variables for both the components, and whenever both equations are solved using a single equation: the phase average values of the variables in the cells are used, and the contribution of the interface is added to the equation using the appropriate jump condition

mentioned in section:5.2.1.

The solver stores and solves for the discrete set of variables evaluated at the center of control volumes c, referred as $\{^{\cup}\phi\}$

$$\{^{\cup}\phi\}_c := \{f, p\} \cup \{^{\cup} f_i\phi_i\}$$

where p is the pressure evaluated at the center of each cell and $\{^{\cup} f_i\phi_i\}$ is the set of component wise contribution of conserved variables

$$\{^{\cup} f_i\phi_i\} := \bigcup_{i\in\{0,1\}} \{ f_i\rho_i,\ f_i\rho_i\boldsymbol{u}_i,\ f_i\rho_i E_i\} \tag{5.16}$$

for each control volume cell. E_i in the above equation represents the total specific energy, $E_i = e_i + \frac{1}{2}\boldsymbol{u}_i \cdot \boldsymbol{u}_i$, of the i-th component. The complete set corresponding to all the control volumes $\boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})$, is $\left\{^{\cup}\{^{\cup}\phi\}_c\right\}$.

As we move further in the chapter, we will be coming across some formulations that use *averaged* or *unified* form of conserving variables in some monolithic expressions, discretized or semi-discretized equations which are $\{\rho, \rho\boldsymbol{u}, \rho E\}$.

$$\phi := \sum_{i\in\{0,1\}} f_i\phi_i \qquad \forall\ \phi \in \{\rho, \rho\boldsymbol{u}, \rho E\}. \tag{5.17}$$

Many equations, like advection equations and interface advection in front tracking, use *averaged velocity*, which is assumed to be continuous across the interface and is given by

$$\boldsymbol{u} := \frac{\sum_i f_i\rho_i\boldsymbol{u}_i}{\sum_i f_i\rho_i}. \tag{5.18}$$

So, in short, the all-mach algorithm aims to find a discrete solution set $\left\{^{\cup}\{^{\cup}\phi\}_c\right\}^{n+1}$ at time step $t^{n+1} := t^n + \Delta t^n$ given the discrete solution set $\left\{^{\cup}\{^{\cup}\phi\}_c\right\}^{n}$ at time t^n and a time step size Δt^n that satisfy all the stability criteria imposed by spatiotemporal discretization

$$\left\{^{\cup}\{^{\cup}\phi\}_c\right\}^{n+1} \xleftarrow{Algo{:}9\texttt{ALLMACH}} \left\{^{\cup}\{^{\cup}\phi\}_c\right\}^{n}, \Delta t^n. \tag{5.19}$$

In the following section, the governing equations will be either written for each component identified with subscript i or written into a single equation where the subscript i is dropped.

Notation

The subscripts $i \in \{0, 1\}$ are used to identify the fluid component separately. The subscript $j \in \{0, .., D-1\}$ represents the direction in the Euclidian space. The computational model is primarily used to solve compressible two-phase flows in either axisymmetric flows or three-dimensional flows. In the future analysis in this chapter, the following notations will be followed when analyzing the integration of the governing equation for a control volume. For a given time step t^n, the computational volume is discretized by a quad/octree $\boldsymbol{\mathcal{T}}(t^n)$ which has a set of control volumes called leaves of the tree $\boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})$. A control cell $c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})$ has its associated control volume, which is the square/cubic volume Ω_c. The control surface $\partial\Omega_c$ is the union of all the six faces $\partial\Omega_c = \cup_f \partial\Omega_c^{\mathcal{F}}$. Variables associated with a face are identified with superscript $\mathcal{F}$ as in $\phi^{\mathcal{F}}$. The six control faces can be uniquely identified by the outward normal, $\boldsymbol{n}^{\mathcal{F}} \in \{\pm\boldsymbol{e}_j\ \forall\ j \in \{0, 1, 2\}\}$. For any primary variable $\phi \in \{f, \rho, \rho\boldsymbol{u}, \rho_i(e_i + \frac{1}{2}\boldsymbol{u}_i \cdot \boldsymbol{u}_i)\}$ in the

discretized (or semi-discretized) equations, the discretized value represents the value of ϕ at the center of the control volume. ϕ might also be represented as $\phi_{\mathcal{I}}$, $\phi_{\mathcal{I},\mathcal{J}}$ or $\phi_{\mathcal{I},\mathcal{J},\mathcal{K}}$ if the values of the neighboring cells are present in the equation. In time discretisation, ϕ^n or $\phi(t^n)$ represents the value at time step t^n. $\phi^{\mathcal{F}}$ is used to represent face center value, and ϕ_i is used to identify the two fluids separately.

5.2.3 Space and Time Discretization

For the time discretization, the continuous set $[0, T]$ is discretized into $\{t^n\} = \{t^n \mid n \in \{0, .., N\},\ t^{n+1} > t^n,\ t^0 = 0,\ t^N = T\}$. The time step size defined $\Delta t^n = t^{n+1} - t^n$ satisfies the CFL conditions and other stability criteria demanded by the scheme. As discussed in the above chapter, the computational volume, Ω is discretized into many square/cubical control volumes using quadtree or octree, i.e $\Omega = \cup_{c \in \boldsymbol{\mathcal{L}}(t^n)} \Omega_c$ [29]. The quadtree or octree ,$\boldsymbol{\mathcal{T}}(t^n)$, used to discretize the control volume is time-dependent and is modified in each time step to accommodate different length scales of the flow. The spatial solution of primary variables f_i, ρ_i, $\rho_i \boldsymbol{u}_i$ and $\rho_i e_i$ are found for the discretized set of all points, which are the centroids of the control volume, $\{\boldsymbol{x}_c\}^n = \{\boldsymbol{x} \mid \boldsymbol{x}$ is the centroid of $\Omega_c \,\forall\, c \in \boldsymbol{\mathcal{L}}(t^n)\}$. The fluxes of the above-mentioned variables $\boldsymbol{F}^{\mathcal{F}}_{f_i \phi_i}$ are evaluated at the center of the control faces $\mathcal{F}$ of cells. Control faces are the union of faces of control volumes, c i.e $\{\mathcal{F} \mid \mathcal{F}$is a control face of $c \,\forall\, c \in \boldsymbol{\mathcal{L}}\}$.

For discretized control volumes, f_i represents the fractional volume of $i-$th fluid in the control volume and takes values $[0, 1]$, unlike $\{0, 1\}$ in the continuous framework. Both f_0 and f_1 are represented as $1 - f$ and f respectively, where f is the volume fraction of the reference fluid (which is $fluid - 1$ in our case) in the control volume. f is the volume average of continuous heaviside function (Eq:A.21).

5.2.4 Interface Representation

The domains $\Omega_i(t)$ and the interface $\Gamma(t)$ are interdependent they can be represented by a single scalar. For example, a heaviside function (Eq:4.14) can be used to identify the domain.

- **VoF**: In the VoF method, the volume fraction of reference fluid f in the control volume is stored as a scalar, and the cells are identified empty ($f = 0$), mixed ($0 \leq f \leq 1$) or full ($f = 1$) depending on the value of f. The interface is reconstructed from the volume fraction f using PLIC algorithm [**?**] [29].
- **Front Tracking**: In the front tracking algorithm, discrete connected marker points constitute the interface. The volume fraction of each component can be found using Algo:11.

5.3 Numerical Method

Using the continuity of velocity across the interface, we can combine the momentum the equation in both the fluid domains into a single one and the implicit time integration of this unified momentum equation is written in the following semi-discrete equation

$$\frac{\rho^{n+1}\boldsymbol{u}^{n+1} - \rho^n \boldsymbol{u}^n}{\Delta t^n} + A^{n+1}(\rho \boldsymbol{u}) = -\boldsymbol{\nabla} p^{n+1} + \boldsymbol{\nabla} \cdot \mu \left(\boldsymbol{\nabla} \boldsymbol{u}^{n+1} + (\boldsymbol{\nabla} \boldsymbol{u}^{n+1})^T \right) + \boldsymbol{f}_\sigma^{n+1} \quad (5.20)$$

where $\boldsymbol{u}$ is the continuous velocity, $\rho = f_0\rho_0 + f_1\rho_1$ the average density of the control volume, p is the pressure in the unified formulation of Navier-Stokes, equation $A^{n+1}(\rho\boldsymbol{u})$ is the time integral of $\boldsymbol{\nabla}\cdot\rho\boldsymbol{u}\boldsymbol{u}$ as in Eq:5.31 and $\boldsymbol{f}_\sigma^{n+1}$ is the surface tension which takes into account the Laplace pressure jump at the interface (Eq:5.10). Since the above implicit equation cannot be solved directly we first predict a velocity at t^{n+1} using p^n instead as

$$\frac{\rho^{n+1}\boldsymbol{u}_p^{n+1} - \rho^n\boldsymbol{u}^n}{\Delta t^n} + A^{n+1}(\rho\boldsymbol{u}) = -\boldsymbol{\nabla}p^n + \boldsymbol{\nabla}\cdot\mu\left(\boldsymbol{\nabla}\boldsymbol{u}_p^{n+1} + (\boldsymbol{\nabla}\boldsymbol{u}_p^{n+1})^T\right) + \boldsymbol{f}_\sigma^{n+1} \quad (5.21)$$

Evaluation of the predicted velocity , $\boldsymbol{u}_p^{n+1}$, is carried out in two steps. In the first step, we evaluate $\boldsymbol{u}^{(adv)}$ followed by $\boldsymbol{u}_p^{n+1}$ as explained in the following equations, where we split Eq:5.21 into two as

$$\frac{\rho^{n+1}\boldsymbol{u}^{(adv)} - \rho^n\boldsymbol{u}^n}{\Delta t^n} = -A^{n+1}(\rho\boldsymbol{u}) \quad (5.22)$$

$$\frac{\rho^{n+1}\boldsymbol{u}_p^{n+1} - \rho^{n+1}\boldsymbol{u}^{(adv)}}{\Delta t^n} = -\boldsymbol{\nabla}p^n + \boldsymbol{\nabla}\cdot\mu\left(\boldsymbol{\nabla}\boldsymbol{u}_p^{n+1} + (\boldsymbol{\nabla}\boldsymbol{u}_p^{n+1})^T\right) + \boldsymbol{f}_\sigma^{n+1} \quad (5.23)$$

In the advection step, we evaluate advect variables f, ρ, $\rho\boldsymbol{u}$, $\rho(e + \frac{1}{2}|\boldsymbol{u}|^2)$. For the reason of stability, the advection for each variable is consistent and uses the same method. (Further explanation is given in section:5.3.1). The equation Eq:5.23 is solved for $\boldsymbol{u}_p^{n+1}$ using a multigrid solver.

Now we find $\boldsymbol{u}^{n+1}$ using the equation

$$\frac{\rho^{n+1}\boldsymbol{u}^{n+1} - \rho^n\boldsymbol{u}^n}{\Delta t^n} + A^{n+1}(\rho\boldsymbol{u}) = -\boldsymbol{\nabla}p^{n+1} + \boldsymbol{\nabla}\cdot\mu\left(\boldsymbol{\nabla}\boldsymbol{u}_p^{n+1} + (\boldsymbol{\nabla}\boldsymbol{u}_p^{n+1})^T\right) + \boldsymbol{f}_\sigma^{n+1} \quad (5.24)$$

Subtracting Eq:5.21 from Eq:5.27 gives

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}_p^{n+1} + \frac{\Delta t^n}{\rho^{n+1}}\left(\boldsymbol{\nabla}p^n - \boldsymbol{\nabla}p^{n+1}\right) = \boldsymbol{u}^* - \frac{\Delta t^n}{\rho^{n+1}}\boldsymbol{\nabla}p^{n+1} \quad (5.25)$$

where

$$\boldsymbol{u}^* = \boldsymbol{u}_p^{n+1} + \frac{\Delta t^n}{\rho^{n+1}}\boldsymbol{\nabla}p^n \quad (5.26)$$

Eq:5.25 coupled with $\boldsymbol{u}^{n+1}$ and p^{n+1} is solved with the pressure evolution equation (refer Eq:5.12)

$$\left(\frac{1}{\rho c^2}\right)_e\left(\frac{p^{n+1} - p^{(adv)}}{\Delta t^n}\right) = -\boldsymbol{\nabla}\cdot\left(\boldsymbol{u}^* - \frac{\Delta t^n}{\rho^{n+1}}\boldsymbol{\nabla}p^{n+1}\right) \quad (5.27)$$

where $\left(\frac{1}{\rho c^2}\right)_e$ is the phase averaged value of $\left(\frac{\Gamma_i}{\rho_i c_i^2} - \frac{\beta_i^2 T_i}{\rho_i c_{p_i}}\right)$ and $p^{(adv)}$ is calculated numerically from $\boldsymbol{u}^{(adv)}$, $E^{(adv)}$. Further explanation of the projection algorithm is given in section:5.3.3.

The complete algorithm is split into four steps advection (Section:5.3.1), prediction (Section:5.3.2), projection (Section:5.3.3) and energy updation (Section:5.3.4 which are discussed below.

5.3.1 Advection

This subsection discusses how the advection terms in the governing equation corresponding to variables ρ_i, $\rho_i \boldsymbol{u}_i$ and $\rho_i E_i$ are integrated in time and also discusses how we advect the volume fraction of components in the cell f_i are advected. The algorithm [64] uses a consistent method in calculating flux for variables ($\{\rho_i, \rho_i \boldsymbol{u}_i, \rho_i E_i\}$) consistent with that of volume fraction advection which is very crucial in the stability [73] [74] of the scheme especially for flows consisting of fluid components with high-density ratios. The advection step is carried out in directional split where the order of splitting is changed cyclically in every timestep. After each time directional split, we find a new volume fraction f^* in each cell using the Weymouth-Yue [18] algorithm. By the end of the last directional split advection, we will have the volume fraction corresponding to f_i^{n+1} in the cell. The complete advection algorithm is discussed below in Algo:8.

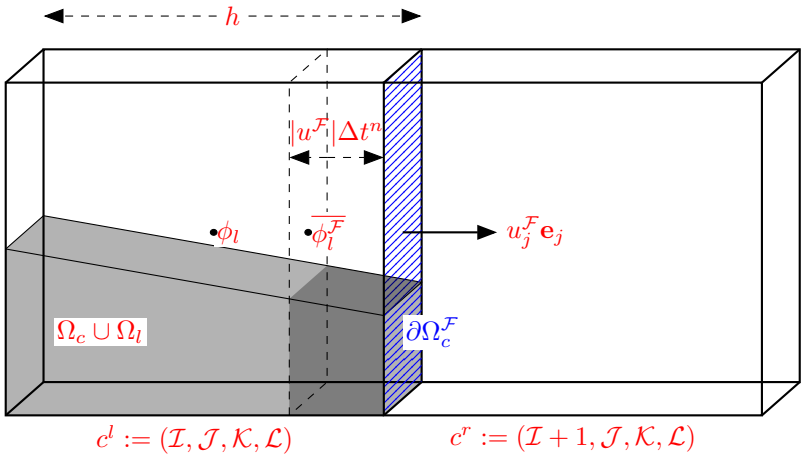


Figure 5.1: **flux at a face** $\partial\Omega_c^{\mathcal{F}}$: The flux of scalar $f_i\phi_i$ through the face $\partial\Omega^{\mathcal{F}}$ which is shared by cells c^l and c^r (marked in dashed blue) with normal $\pm\boldsymbol{e}^j$ is evaluated as a multiple of $\overline{\phi_i^{\mathcal{F}}}$ and $\boldsymbol{F}_{f_i}^{\mathcal{F}} \cdot \boldsymbol{n}^{\mathcal{F}}$, where $\overline{\phi_i^{\mathcal{F}}}$ is the interpolated value of ϕ_i at a distance $|u^{\mathcal{F}}|\Delta t^n$ upstream of the face (Eq:5.38). If we take the fluid component, i is colored in grey, then $|\boldsymbol{F}_{f_i}^{\mathcal{F}}|$ is the volume under the reconstructed piecewise interface, which fluxes out (marked in dark grey) normalized with the volume of cube h^3.

The evolution equation of a variable ϕ can be written in the conservative form

$$\frac{\partial \phi}{\partial t} + \boldsymbol{\nabla} \cdot (\boldsymbol{u}\phi) = S_\phi \tag{5.28}$$

Eq:5.28 on time integration gives

$$\phi^{n+1} = \phi^{(adv)} + \Delta t^n S_\phi^{n+1} \tag{5.29}$$

where

$$\phi^{(adv)} = \phi^n - A^{n+1}(\phi) \tag{5.30}$$

where $A^{n+1}(\phi)$ is the time integral of $\boldsymbol{\nabla} \cdot (\phi \boldsymbol{u})$ as in Eq:5.31 or Eq:5.32.

In the advection step, we evaluate $\phi^{(adv)}$ for the variables heaviside (f), density (ρ), momentum ($\rho\boldsymbol{u}$) and specific total energy ($\rho e + \frac{1}{2}\rho|\boldsymbol{u}|^2$) in their corresponding evolution equations (Eq:5.3, Eq:5.4, Eq:5.5 and Eq:5.6) respectively. For the reason of consistency, the advection scheme used for all the above-mentioned variables are the same. The volume average of the

time integral of advection term in the control volume Ω_c,

$$A^{n+1}(\phi) := \frac{1}{|\Omega_c|}\int_{t^n}^{t^{n+1}}\int_{\Omega_c} \boldsymbol{\nabla}\cdot(\phi\boldsymbol{u})dvdt = \frac{1}{|\Omega_c|}\int_{t^n}^{t^{n+1}}\int_{\partial\Omega_c} \phi\boldsymbol{u}\cdot\boldsymbol{n}dA\,dt \tag{5.31}$$

where $|\Omega_c| = h^D$ is the volume of control volume. For a discretized cubic control volume, the above integral can be written as a summation of all the cubic faces

$$A^{n+1}(\phi) = \frac{1}{h^3}\sum_f \int_{t^n}^{t^{n+1}}\int_{\partial\Omega_c^{\mathcal{F}}} \phi(t)\boldsymbol{u}\cdot\boldsymbol{n}(t)dA\,dt \approx \frac{1}{h^3}\sum_f \int_{t^n}^{t^{n+1}} \phi^{\mathcal{F}}(t)\boldsymbol{u}^{\mathcal{F}}(t)\cdot\boldsymbol{n}^{\mathcal{F}}h^2\,dt = \sum_f \boldsymbol{F}_\phi^{\mathcal{F}}\cdot\boldsymbol{n}^{\mathcal{F}} \tag{5.32}$$

where the $\phi^{\mathcal{F}}$ and $\boldsymbol{u}_n^{\mathcal{F}} = (\boldsymbol{u}^{\mathcal{F}}\cdot\boldsymbol{n}^{\mathcal{F}})\boldsymbol{n}^{\mathcal{F}}$ are the face-centered values of ϕ and outward normal velocity, respectively. For $\boldsymbol{n}^{\mathcal{F}} = \pm\boldsymbol{e}_j$, $\boldsymbol{u}^{\mathcal{F}}\cdot\boldsymbol{n}^{\mathcal{F}} = \pm u_j^{\mathcal{F}}$ which gives

$$\boldsymbol{F}_\phi^{\mathcal{F}} = \boldsymbol{e}_j\left(\frac{1}{h}\int_{t^n}^{t^{n+1}} \phi(t)u_j^{\mathcal{F}}(t)dt\right) \tag{5.33}$$

Eq:5.31 can be extended to vectors like momentum as

$$A^{n+1}(\rho\boldsymbol{u}) := \sum_{j\,\in\,\{0,..,D-1\}} A^{n+1}(\rho u_j)\boldsymbol{e}_j \tag{5.34}$$

$\boldsymbol{F}_\phi^{\mathcal{F}}$ is the discretized surface average of outward flux calculated by Bell-Collela-Glaz scheme [75] which can be written as In the single formulation ϕ is written as $f_0\phi_0 + f_1\phi_1$ which gives advection term as the sum of advection of each phase

$$A^{n+1}(\phi) = \sum_{i\in\{0,1\}} A^{n+1}(f_i\phi_i) = \sum_f F_{f_i\phi_i}^{\mathcal{F}} \tag{5.35}$$

In order to evaluate $F_{f_i\phi_i}$ we need to integrate the infinitesimal flux $f_i^{\mathcal{F}}(t)\phi_i^{\mathcal{F}}(t)\boldsymbol{u}^{\mathcal{F}}(t)dt$ as

$$\boldsymbol{F}_{f_i\phi_i}^{\mathcal{F}} = \boldsymbol{e}_j\left(\frac{1}{h}\int_{t^n}^{t^{n+1}} f_i^{\mathcal{F}}(t)\phi_i^{\mathcal{F}}(t)u_j^{\mathcal{F}}(t)dt\right) = \boldsymbol{e}_j\left(\overline{\phi_i^{\mathcal{F}}}(t^n)\frac{1}{h}\int_{t^n}^{t^{n+1}} f_i^{\mathcal{F}}(t)u_j^{\mathcal{F}}(t)dt\right) = \overline{\phi_i^{\mathcal{F}}}(t^n)\boldsymbol{F}_{f_i}^{\mathcal{F}} \tag{5.36}$$

where $u^{\mathcal{F}}(t) = \boldsymbol{u}^{\mathcal{F}}(t)\cdot\boldsymbol{n}^{\mathcal{F}}$ is the outward normal component of velocity at the face center and $F_{f_i}^{\mathcal{F}}$ is the fraction of fluid-i that fluxed out of the control volume at face $\mathcal{F}$ during the time Δt^n. $F_{f_i}^{\mathcal{F}}$ is evaluated using the reconstructed piece-wise interface in geometric-VOF [29] (In Fig:5.1). Alternatively, using the piecewise front segments in FT. In the integration, instead of using the face-centered value, $\phi_i^{\mathcal{F}}(t^n)$ we use the upstream value $\overline{\phi_i^{\mathcal{F}}}(t^n)$ which is calculated using BCG second order upwind [75].

Flux

Given a face $\mathcal{F}$ shared by cells c^l and c^r. The cell c^r is the face neighbor of cell c^l in the direction $\boldsymbol{e}_j$. The domain associated with face $\mathcal{F}$ is $\partial\Omega_c^{\mathcal{F}}$. We interpolate the velocity at the face center $u_j^{\mathcal{F}}$ from the cell-centered value $\boldsymbol{u}$. Then we determine the cell $c \in \{c^l, c^r\}$ which is the upwind cell ($c = c^l$ if $u_j^{\mathcal{F}} \geq 0$ and $c = c^r$ if $u_j^{\mathcal{F}} < 0$). The flux of each component is the corresponding volume (normalized with h^3. Eq:5.36) occupied by each fluid between

Algorithm 7: `FLUX`

Data: $\boldsymbol{u}^n, f^*, \{f_i\phi_i\}^n, \mathcal{F}, j$
Result: $\{\boldsymbol{F}^{\mathcal{F}}_{f_i\phi_i}\}$

/* This algorithm calculates fluxes of primary conservative variables of both fluids, $\{f_i\phi_i\}^n := \{f_i, f_i\rho_i, f_i\rho_i\boldsymbol{u}_i, f_i\rho_i E_i\}$ for at the cell faces during each directional split advection. The flux corresponding to each quantity listed above is respectively $\{\boldsymbol{F}^{\mathcal{F}}_{f_i}, \boldsymbol{F}^{\mathcal{F}}_{f_i\rho_i}, \boldsymbol{F}^{\mathcal{F}}_{f_i\rho_i\boldsymbol{u}_i}, \boldsymbol{F}^{\mathcal{F}}_{f_i\rho_i E_i}\}$ Refer to section:5.3.1. f^* is the volume fraction of reference fluid before each sweep. The face $\mathcal{F}$ is shared by cells c^l and c^r. c^r the face neighbor cell of c^l in the direction $\boldsymbol{e}_j$ (Fig:5.1). */

Identify the cell c upstream to the face $\mathcal{F}$ /* $c = c^r$ if $u^{\mathcal{F}}_j < 0$ and $c = c^r$ if $u^{\mathcal{F}}_j \geq 0$ */
if $0 < f^* < 1$ then /* f^* in the cell c (mixed cell) */
  Reconstruct interface $\boldsymbol{m} \cdot \boldsymbol{x} = \alpha$ in the cell c.
  $v \leftarrow$ volume between the face $\mathcal{F}$ and plane $|u^{\mathcal{F}}_j|\Delta t^n$ upstream under $\boldsymbol{m} \cdot \boldsymbol{x} = \alpha$ in the cell c.
  $v \leftarrow \frac{v}{h^d}$ /* Normalize with cell volume */
end
else if $f^* = 0$ then $v \leftarrow 0$ /* empty cell */
else if $f^* = 1$ then $v \leftarrow 1$ /* full cell */
Evaluate $\left.\frac{\partial \phi_i}{\partial x_j}\right|_{j=0}$ using Eq:5.38
Evaluate $\overline{\phi^{\mathcal{F}}_i}$ using 5.37
for *each* ϕ do
  $\boldsymbol{F}^{\mathcal{F}}_{f_1} \leftarrow sign(u^{\mathcal{F}}_n)\, v\, \boldsymbol{e}_j$ /* Flux of reference component ($fluid - 1$) */
  $\boldsymbol{F}^{\mathcal{F}}_{f_0} \leftarrow sign(u^{\mathcal{F}}_n)\, (1 - v)\, \boldsymbol{e}_j$ /* Flux of non-reference component ($fluid - 0$) */
  $\boldsymbol{F}^{\mathcal{F}}_{f_i\phi_i} \leftarrow \overline{\phi^{\mathcal{F}}_i}\, \boldsymbol{F}^{\mathcal{F}}_{f_i}$
end

the face $\mathcal{F}$ and a plane parallel to it, which is at a distance $|u_j^{\mathcal{F}}|\Delta t$ upwind to it. If the cell c is either empty or full, the calculation of the flux of both components is straightforward. For an empty cell ($f^* = 0$), flux of the reference component ($\boldsymbol{F}_{i_1}^{\mathcal{F}}$) and non-reference ($\boldsymbol{F}_{f_0}^{\mathcal{F}}$) are respectively $\mathbf{0}$ and $(u_j^{\mathcal{F}}\Delta t)/h^3\boldsymbol{e}_j$. Similarly for full cell ($f^* = 1$), the fluxes $\boldsymbol{F}_{i_1}^{\mathcal{F}}$ and $\boldsymbol{F}_{f_0}^{\mathcal{F}}$ are respectively $(u_j^{\mathcal{F}}\Delta t)/h^3\boldsymbol{e}_j$ and $\mathbf{0}$. For mixed cells ($0 \leq f^* \leq 1$), we need to reconstruct the discrete interface in the cell c, say, in the form $\boldsymbol{m}\cdot\boldsymbol{x} = \alpha$ .If v is the volume bounded between the above-mentioned planes, under the discrete interface $\boldsymbol{m}\cdot\boldsymbol{x} = \alpha$ in the cell, then $\boldsymbol{F}_{f_1}^{\mathcal{F}} = sign(u_j^{\mathcal{F}})v/h^3\boldsymbol{e}_j$ and $\boldsymbol{F}_{f_0}^{\mathcal{F}} = sign(u_j^{\mathcal{F}})(u_j^{\mathcal{F}}\Delta t - v)/h^3\boldsymbol{e}_j$.

Algorithm 8: `ADVECTION`

Data: $\{^{\cup}\boldsymbol{u}\}^n, \{^{\cup}f\}^n, \{^{\cup}\{^{\cup}f_i\phi_i\}\}^n$ and Δt^n
/* $\{^{\cup}f_i\phi_i\} = \{f_i\rho_i, f_i\rho_i\boldsymbol{u}_i, f_i\rho_i e_i + \frac{1}{2}\rho_i\boldsymbol{u}_i\cdot\boldsymbol{u}_i \mid i \in \{0,1\}\}$ */
Result: $\{^{\cup}f\}^{n+1}, \{^{\cup}\{^{\cup}f_i\phi_i^{(adv)}\}\}$ /* Calculate $f_i\phi_i^{(adv)}$ as in Eq:5.35 */

for *each cell* $c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})$ do /* with control volume Ω_c */
  if $f^n \geq \frac{1}{2}$ then $c_f^n \leftarrow 1$ else $c_f^n \leftarrow 0$ /* Weymouth and Yue [18] */
end
for *each direction* $j \in \mathbb{N}_D$ do /* Change the order every iteration of time step */
  for *each face* $\mathcal{F}$ *in* j*-th direction* do /* $\mathcal{F}$ is shared by cells c_l and c_r as in Fig:5.1 */
    $\{\boldsymbol{F}_{f_i\phi_i}^{\mathcal{F}}\} \leftarrow$ `FLUX`$(\boldsymbol{u}^n, f^*, \{\phi_i^n\}, \mathcal{F}, i, j)$
  end
  for *each cell* $c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})$ do /* with cubic control volume Ω_c */
    $f^* \leftarrow f^* + c_f^n\frac{\Delta t^n}{\Delta x_j}(u_j^{\mathcal{F}_r} - u_j^{\mathcal{F}_l})$ /* Weymouth and Yue [18] */
  end
end
for *each cell* $c \in \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{T}})$ do /* with control volume Ω_c */
  $f^{n+1} \leftarrow f^*$
  for *each* $f_i\phi_i \in \{^{\cup}f_i\phi_i\}$ do
    $f_i\phi_i^{(adv)} \leftarrow f_i\phi_i^n$
    for *each face* $\mathcal{F}$ *of cell* c do /* with normal $\boldsymbol{n}_{\mathcal{F}} = \pm\hat{\mathbf{e}}_j$ */
      $f_i\phi_i^{(adv)} \leftarrow f_i\phi_i^{(adv)} + \boldsymbol{F}_{f_i\phi_i}^{\mathcal{F}}\cdot\boldsymbol{n}^{\mathcal{F}}$
    end
  end
end

For the face shared by the cells $c_l(\mathcal{I},\mathcal{J},\mathcal{K},\mathcal{L})$ and $c_r(\mathcal{I}+1,\mathcal{J},\mathcal{K},\mathcal{L})$ (the blue colored face in Fig:5.1),

$\overline{\phi_i^{\mathcal{F}}}$ is evaluated as

$$\overline{\phi_i^{\mathcal{F}}}(t^n) = \begin{cases} \phi_i(t^n)\big|_{c^l} \pm \left(\frac{h}{2} - |u^{\mathcal{F}}|\Delta t^n\right)\frac{\partial\phi_i}{\partial x_j}\big|_{c^l} & \text{if } u_j^{\mathcal{F}} \geq 0 \\ \phi_i(t^n)\big|_{c^r} \mp \left(\frac{h}{2} - |u^{\mathcal{F}}|\Delta t^n\right)\frac{\partial\phi_i}{\partial x_j}\big|_{c^r} & \text{if } u_j^{\mathcal{F}} < 0 \end{cases} \tag{5.37}$$

where the subscript c^l and c^r are the corresponding values in the cells sharing the face. In case of inward flux at any face, the $\overline{\phi_i^{\mathcal{F}}}$ is calculated by interpolating from the cell-centered value of the face neighbor where the evaluation of the slope at the center of the cell $c(\mathcal{I},\mathcal{J},\mathcal{K},\mathcal{L})$ in the

x direction ($j = 0$) is illustrated below

$$\left.\frac{\partial \phi_i}{\partial x_j}\right|_{j=0} = \begin{cases} \frac{1}{h} g(\phi_{i_{\mathcal{I}-1}}, \phi_{i_{\mathcal{I}}}, \phi_{i_{\mathcal{I}+1}}) & \text{if } f_{i_{\mathcal{I}-1}}, f_{i_{\mathcal{I}}}, f_{i_{\mathcal{I}+1}} > 0 \\ \frac{1}{h}(\phi_{i_{\mathcal{I}+1}} - \phi_{i_{\mathcal{I}}}) & \text{if } f_{i_{\mathcal{I}-1}} = 0 \text{ and } f_{i_{\mathcal{I}+1}} > 0 \\ \frac{1}{h}(\phi_{i_{\mathcal{I}}} - \phi_{i_{\mathcal{I}-1}}) & \text{if } f_{i_{\mathcal{I}-1}} > 0 \text{ and } f_{i_{\mathcal{I}+1}} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.38}$$

where the function $g(\phi_{i_{\mathcal{I}-1}}, \phi_{i_{\mathcal{I}}}, \phi_{i_{\mathcal{I}+1}})$ is given by

$$g(\phi_{i_{\mathcal{I}-1}}, \phi_{i_{\mathcal{I}}}, \phi_{i_{\mathcal{I}+1}}) = \begin{cases} \min\left(\theta(\phi_{i_{\mathcal{I}}} - \phi_{i_{\mathcal{I}-1}}),\ \frac{1}{2}(\phi_{i_{\mathcal{I}+1}} - \phi_{i_{\mathcal{I}-1}}),\ \theta(\phi_{i_{\mathcal{I}+1}} - \phi_{i_{\mathcal{I}}})\right) & \text{if } \phi_{i_{\mathcal{I}-1}} > \phi_{i_{\mathcal{I}}} > \phi_{i_{\mathcal{I}+1}} \\ \max\left(\theta(\phi_{i_{\mathcal{I}}} - \phi_{i_{\mathcal{I}-1}}),\ \frac{1}{2}(\phi_{i_{\mathcal{I}+1}} - \phi_{i_{\mathcal{I}-1}}),\ \theta(\phi_{i_{\mathcal{I}+1}} - \phi_{i_{\mathcal{I}}})\right) & \text{if } \phi_{i_{\mathcal{I}-1}} < \phi_{i_{\mathcal{I}}} < \phi_{i_{\mathcal{I}+1}} \\ 0 & \text{Otherwise} \end{cases} \tag{5.39}$$

The outward flux calculated at each face in Eq:5.37 is taken as 0 if outward normal component of face centered velocity ($u_n^{\mathcal{F}}$)

Interface Advection

Evolution equation corresponding to Eq:5.28 for time integration of volume fraction f_i is the conservative form of Eq:5.6 given by

$$\frac{\partial}{\partial t} f_i + \boldsymbol{\nabla} \cdot (\boldsymbol{u} f_i) \quad = \quad f_i \boldsymbol{\nabla} \cdot \boldsymbol{u} \tag{5.40}$$

where the *averaged* velocity $\boldsymbol{u}_i$ is given by Eq:5.18. Weymouth-Yue [18] suggested the dilatation term $f_i \boldsymbol{\nabla} \cdot \boldsymbol{u}$ is approximated by $c_{f_i}^n \boldsymbol{\nabla} \cdot \boldsymbol{u}$ where

$$c_{f_i}^n = \begin{cases} 0 \text{ if } f_i^n < \frac{1}{2} \\ 1 \text{ if } f_i^n \geq \frac{1}{2} \end{cases} \tag{5.41}$$

This modification makes sure the f is within $[0, 1]$ after each split advection and the net volume conserves for both fluids after d number of split advection (where d is dimension of Eulerian space) if $\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0$ (incompressible flow). If we say f_i^*, f_i^{**} are the volume fraction of i-th component in the cell c before and after each directional split in direction j, we can write

$$f_i^{**} = f_i^* - \left(\boldsymbol{F}_{f_i}^{\mathcal{F}_r} - \boldsymbol{F}_{f_i}^{\mathcal{F}_l}\right) \cdot \boldsymbol{e}_j + c_f^n \frac{\Delta t^n}{\Delta x_j} \left(u_n^{\mathcal{F}_r} - u_n^{\mathcal{F}_l}\right) \tag{5.42}$$

where $\mathcal{F}_l$ and $\mathcal{F}_l$ are the *left* and *right* faces of cell c in the direction $\boldsymbol{e}_j$. The complete interface advection algorithm is given in Algo:8.

So, in the VoF method, the interface is inherently advected if we integrate f_i

$$\Gamma(t^{n+1}) \xleftarrow{\text{Algo:8 ADVECTION}} (\Gamma(t^n), \boldsymbol{u}^n, \Delta t^n) \tag{5.43}$$

In the front tracking method, the heaviside is taken as the volume fraction and is calculated using Algo:11 `F2V3D` and the $\phi^{(adv)}$ is calculated in similar fashion (Eq:5.36, Algo:8). The marker points, $\boldsymbol{x}^s \in \Gamma(t^n)$, are explicitly advected using locally interpolated velocity at marker

locations i.e $\frac{d}{dt}\boldsymbol{x}^s = \boldsymbol{u}(\boldsymbol{x}^s)$ and thus the we have $\Gamma(t^{n+1})$

$$\Gamma(t^{n+1}) \xleftarrow{\boldsymbol{x}^s(t^{n+1}) \leftarrow \boldsymbol{x}^s(t^n) + \Delta t^n \boldsymbol{u}^n(\boldsymbol{x}^s(t^n))} (\Gamma(t^n), \boldsymbol{u}^n, \Delta t^n) \tag{5.44}$$

However f_i^{n+1} from the Weymouth-Yue [18] advection will be inconsistent with the volume fraction calculated using Algo:F2V3D($\Gamma(t^{n+1})$). It may be taken care of by adjusting the marker points ($\boldsymbol{x}^s \in \Gamma(t^{n+1})$) by integrating the following equation in the pseudo time step

$$\frac{d}{d\tau}\boldsymbol{x}^s = C_{VN} \;\; \hat{\boldsymbol{n}}(\boldsymbol{x}^s) \sum_{c \in \boldsymbol{\mathcal{L}}} \delta_h(\boldsymbol{x}_c - \boldsymbol{x}^s)(f_1^{n+1} - \hat{f}_1^{n+1})$$

where f_1^{n+1} is the volume fraction of reference fluid after Weymouth-Yue [18] advection, $\hat{f}_1^{n+1}$ is the volume fraction of reference fluid calculated using Algo:11 (`F2V3D`) before every pseudo timestep iteration, δ_h is the discrete Dirac delta function and C_{VN} is a constant that satisfies Von-Neumann stability. This idea is not pondered in this thesis and is left for future analysis.

Density Advection

Since $S_\phi = 0$ in Eq:5.29 for $\phi = \rho$ which gives

$$(f_i\rho_i)^{n+1} = (f_i\rho_i)^{(adv)} \tag{5.45}$$

$$\rho^{n+1} = \sum_i (f_i\rho_i)^{n+1} \tag{5.46}$$

Velocity Advection

The 'average' or 'unified' advected velocity is found (Eq:5.22) as

$$\boldsymbol{u}^{(adv)} = \frac{1}{\rho^{n+1}} \left((\rho\boldsymbol{u})^n - \Delta t^n A^{n+1}(\boldsymbol{\rho u}) \right) \tag{5.47}$$

where $A^{n+1}(\boldsymbol{\rho u}) = \sum_i f_i^{n+1} A^{n+1}(\boldsymbol{\rho u})$ and ρ^{n+1} is found using Eq:5.46.

Thus, after the advection step, we would have found the volume fraction, (f_i^{n+1}), density $(f_i\rho_i)^{n+1}$ and thus $\Gamma(t^{n+1})$ and $\Omega_i(t^{n+1})$. We have also evaluated $(f_i\rho_i\boldsymbol{u}_i)^{(adv)}$ and $(f_i\rho_i E_i)^{(adv)}$ from which we will eventually find $(f_i\rho_i\boldsymbol{u}_i)^{n+1}$ and $(f_i\rho_i E_i)^{n+1}$ which will be discussed in the following sections.

5.3.2 Prediction

Predicted velocity at t^{n+1}, $\boldsymbol{u}_p^{n+1}$ is found by solving the equation

$$\frac{\rho^{n+1}\boldsymbol{u}_p^{n+1} - \rho^{n+1}\boldsymbol{u}^{(adv)}}{\Delta t^n} = -\boldsymbol{\nabla} p^n + \boldsymbol{\nabla} \cdot \mu \left(\boldsymbol{\nabla}\boldsymbol{u}_p^{n+1} + (\boldsymbol{\nabla}\boldsymbol{u}_p^{n+1})^T \right) + \boldsymbol{f}_\sigma^{n+1} \tag{5.48}$$

using multigrid solver in [29]. The *averaged* viscosity is taken as the harmonic average, $\mu := 1/\sum_i (f_i/\mu_i)$. In Eq:5.48, the surface tension, $\boldsymbol{f}_\sigma^{n+1}$ is evaluated from the interface location Γ^{n+1} in FT and from the local heavisides f_i^{n+1} in VOF. We have discussed the implementation of surface tension evaluation in front tracking in Section:**??**. Surface tension implementation in VOF is discussed in the following section.

Surface Tension in VOF

Surface Tension using void fraction in VOF is implemented using *Continuum Surface Force* (CSF) Model by [33]. $\boldsymbol{f}_\sigma^{n+1}$ is the volumetric average of surface tension integral calculated at

the center of a control cell, say c, which can be written as

$$\boldsymbol{f}_{\sigma}^{n+1}(\boldsymbol{x}) = \frac{1}{|\Omega_c|}\int_{\Omega_c}\left[\int_{\Gamma}\sigma(\boldsymbol{x}^s)\,\kappa(\boldsymbol{x}^s)\,\hat{\boldsymbol{n}}(\boldsymbol{x}^s)\;dA\right]\;\delta(\boldsymbol{x}'-\boldsymbol{x}^s)dv(\boldsymbol{x}') \tag{5.49}$$

where $\sigma(\boldsymbol{x}^s)$, $\kappa(\boldsymbol{x}^s)$ and $\hat{\boldsymbol{n}}(\boldsymbol{x}^s)$ are respectively surface tension, curvature and normal defined on a surface point $\boldsymbol{x}^s$ on $\Gamma(t^{n+1})$, δ is the Dirac delta function and $V_{\Omega_c} = h^3$ is the volume of control volume. Using the CSF model for constant surface tension, we evaluate the above expression using the discrete cell-centered curvature and heaviside.

$$\boldsymbol{f}_{\sigma}^{n+1}(\boldsymbol{x}) \approx \sigma\kappa_h(\boldsymbol{x})\boldsymbol{\nabla}_h C(\boldsymbol{x}) \tag{5.50}$$

For achieving *balanced surface tension* [60] implementation, we use face-centered cells for discretization momentum and face-centered value $\sigma\kappa$ in the equation Eq: is calculated by averaging the corresponding value in the cells that share that particular face. The cell-centered value of κ is found using height function [60]. For an axisymmetric simulation, the curvature can be calculated as

$$\kappa = \frac{h_r^{''}}{(1+h_r^{'2})^{3/2}} - \frac{1}{h_r(1+h_r^{'2})^{1/2}} \tag{5.51}$$

where $h_r(z)$ is the height calculated in the radial direction and $h_r^{'} = d/dz\;(h_r)$ and $h_r^{''} = d^2/dz^2(h_r)$. Stable algorithms for calculating curvature using the height function is discussed [60]. May also refer to the Appendix Sec:A.4.2.

Stability

The stability criteria imposed by the discretization scheme of surface tension is given by ([33] [76])

$$\Delta t^n \leq \sqrt{\frac{(\rho_0+\rho_1)h^3}{4\pi\sigma}}, \tag{5.52}$$

where h is the dimension of the smallest cubic cells in the AMR grid.

5.3.3 Projection

In this projection step we solve p^{n+1} and $\boldsymbol{u}^{n+1}$ as mentoined in Eq:5.27. When we rearrange the terms, it gives

$$-\Delta t^n\boldsymbol{\nabla}\cdot\left(\frac{1}{\rho^{n+1}}\boldsymbol{\nabla}p^{n+1}\right) + \left(\frac{1}{\rho c^2}\right)_e p^{n+1} = -\boldsymbol{\nabla}\cdot\boldsymbol{u}^* - \left(\frac{1}{\rho c^2}\right)_e\frac{p^{(adv)}}{\Delta t^n} \tag{5.53}$$

which is in the form of a Helmholtz equation

$$\boldsymbol{\nabla}\cdot\alpha_p\boldsymbol{\nabla}p^{n+1} - \beta_p^2\,p^{n+1} = \gamma_p \tag{5.54}$$

where $\alpha_p = \frac{1}{\rho^{n+1}}$, $\beta_p^2 = \frac{1}{\Delta t^n}(\frac{1}{\rho c^2})_e$ and $\gamma_p = \boldsymbol{\nabla}\cdot\boldsymbol{u}^* + \left(\frac{1}{\rho c^2}\right)_e\frac{p^{(adv)}}{\Delta t^n}$ The above discrete Helmholtz equation is solved in the AMR grid using in-built multigrid-based solver in basilisk.

In the above equation $\left(\frac{1}{\rho c^2}\right)_e = \left(\frac{\Gamma_i}{\rho_i c_i^2} - \frac{\beta_i^2 T_i}{\rho_i c_{p_i}}\right)$. In the context of simulation of cavitation, we can take gas as ideal with $\beta_i = 1/T_i$, and for water we have $\Gamma_i = 1$ and $\beta_i \approx 0$ and

thus in both the cases $\left(\frac{1}{\rho c^2}\right)_e \approx \left(\frac{1}{\rho_i c_i^2}\right)$ is a good approximation. So for empty and full cells $(1/\rho c^2)_e$ can be readily evaluated as $(1/\rho_0 c_0^2)$ and $(1/\rho_1 c_1^2)$ respectively where the speed of sound in the individual component c_i are evaluated using the equation of state for reference and non-reference fluid (Eq:A.47) gives

$$\left(\frac{1}{\rho c^2}\right)_e = \left(\frac{1}{\rho_i c_i^2}\right) = \frac{1}{\Gamma_i}\ \frac{1}{p + \Gamma_i \Pi_i} \quad \text{if } f_i = 1 \tag{5.55}$$

and for a mixed cell with $0 < f < 1$, we evaluate

$$\left(\frac{1}{\rho c^2}\right)_e = \frac{\frac{1}{\Gamma-1}}{p\left(\frac{1}{\Gamma-1}+1\right) + \frac{\Gamma\Pi}{\Gamma-1}}$$

where the cell averaged value of $\frac{1}{(\Gamma-1)}$ and $\frac{\Gamma\Pi}{(\Gamma-1)}$ are found using summing the the individual contribution of each component

$$\frac{1}{\Gamma-1} = \sum_{i=0}^{1} f_i \frac{1}{\Gamma_i - 1} \quad \text{and} \quad \frac{\Gamma\Pi}{\Gamma-1} = \sum_{i=0}^{1} f_i \frac{\Gamma_i \Pi_i}{\Gamma_i - 1} \tag{5.56}$$

$$\tag{5.57}$$

$p^{(adv)}$ in Eq:5.53 is evaluated using the approximate evaluation

$$p^{(adv)} \approx \rho E^{(adv)} - \frac{1}{2}\rho \boldsymbol{u}\cdot\boldsymbol{u}^{(adv)} - \frac{\Gamma\Pi}{\Gamma-1} \tag{5.58}$$

where

$$\rho E^{(adv)} = \sum_{i=0}^{1} f_i \rho_i E_i^{(adv)}, \quad \rho\boldsymbol{u} = \sum_{i=0}^{1} (f_i\rho_i\boldsymbol{u})^n,$$

$$\boldsymbol{u}^{(adv)} = \frac{\sum_{i=0}^{1} f_i\rho_i\boldsymbol{u}_i^{(adv)}}{\sum_{i=0}^{1} f_i\rho_i}, \quad \frac{\Gamma\Pi}{\Gamma-1} = \sum_{i=0}^{1} f_i \frac{\Gamma_i\Pi_i}{\Gamma_i - 1}$$

The above Helmholtz equation reduces to a Poisson equation in the limit $c \to \infty$ for an incompressible flow.

The solution of Hemhotz equation Eq:5.53 is p^{n+1} from which you can find velocity $\boldsymbol{u}^{n+1}$ using Eq:5.25 and Eq:5.26. Finally you can find $f_i\rho_i\boldsymbol{u}_i$ by mutliplying $\boldsymbol{u}$ with $f_i\rho_i$.

5.3.4 Energy Evolution

By the end of projection step, section5.3.3, the discrete solution set $\mathcal{V}^{n+1}$ at time t^{n+1} are updated except for the variables $f_i\rho_i E_i$. In order to update the total specific energy, you have to complete the energy evolution equation (Refer to Eq: and Eq:)

$$(f_i\rho_i E_i)^{n+1} = (f_i\rho_i E_i)^{(adv)} + f_i^{n+1}\Delta t^n \left[-\boldsymbol{\nabla}\cdot\left(\boldsymbol{u}^{n+1}p_i^{n+1}\right) + \boldsymbol{\nabla}\cdot\left(\boldsymbol{\tau}_i^{n+1}\cdot\boldsymbol{u}^{n+1}\right)\right] \tag{5.59}$$

where $\boldsymbol{\tau}_i^{n+1} = \mu_i\left[\nabla\boldsymbol{u}^{n+1} + (\nabla\boldsymbol{u}^{n+1})^T\right]$, $p_0^{n+1} = p^{n+1} - f^{n+1}\sigma\kappa^{n+1}$ and $p_1^{n+1} = p^{n+1} + (1 - f^{n+1})\sigma\kappa^{n+1}$.

Now, all the variables corresponding to t^{n+1} are found. The complete algorithm of the all-mach algorithm is shown in Algo:9.

Algorithm 9: `ALLMACH`

Data: $p^n, {\rho_i}^n, (\rho_i \boldsymbol{u})^n, (\rho_i e + \frac{1}{2}\rho_i \boldsymbol{u}\cdot\boldsymbol{u})^n, \Delta t$

Result: $p^{n+1}, {\rho_i}^{n+1}, (\rho_i \boldsymbol{u})^{n+1}, (\rho_i e + \frac{1}{2}\rho_i \boldsymbol{u}\cdot\boldsymbol{u})^{n+1}$

while $t^n < T$ do

- Determine Δt^n using stability criteria
- Calculate f^{n+1} and $f_i\phi_i^{(adv)}$ for all cells (Algo:8 (`ADVECTION`), Eq:5.31)
- Calculate surface tension $\boldsymbol{f}_\sigma^{n+1}$ at each cell centers (Eq:5.50)
- Calculate predicted velocity $\boldsymbol{u}_p^{n+1}$ for each cells (Eq:5.48)
- Calculate $\boldsymbol{u}^*$ for each cells (Eq:5.26)
- Calculate $p^{(adv)}$ using (Eq:5.58)
- Calculate p^{n+1} (Eq:5.53, Eq:5.54)
- Calculate $\rho\boldsymbol{u}^{n+1}$ (Eq:5.25)
- Calculate ρE^{n+1} (Eq:5.59)

end

5.4 Test Cases

5.4.1 Weakly Non-Linear Collapse of Bubble

In this section, we discuss a test case to check the non-linearity of the compressible two-phase flow system. In this test case, a bubble initially at equilibrium with ambient pressure $p_{\infty,0}$ is suddenly applied with a pressure perturbation $\Delta p > 0$ at $t = 0^+$ such that the bubble collapses. Let, at the initial equilibrium ($t \leq 0$), the bubble has an equilibrium radius R_0 with non-condensible gas content with equilibrium gas pressure $p_{G,0}$ which satisfy the Laplace pressure jump $p_{G,0} = p_{\infty,0} + 2\sigma/R_0$ where σ is the surface tension. Let us use the characteristic velocity scale as the collapse velocity $U_c = \sqrt{\frac{\Delta p}{\rho_L}}$ and the length scale $L_c = R_0$ which gives the time scale as $T_c = R_0/U_c$.

Let us define a non-dimensional number using pressure perturbation as

$$P = \frac{p_\infty}{p_{\infty,0}} = \frac{p_{\infty,0} + \Delta p}{p_{\infty,0}},$$

which is the ratio of new ambient pressure to the old (with $P > 1$) and in a violent collapse $P \gg 1$. However, for this particular test case, we use $P = 10$ which corresponds to a weak non-linear collapse. The Weber Number (We), Reynolds Number (Re) and Mach number (Ma) are defined as

$$We = \frac{\Delta p R_0}{\sigma},\ Re = \frac{R_0\sqrt{\Delta p\ \rho_{L,0}}}{\mu_L},\ \text{and}\ Ma = \frac{U_c}{c_{L,0}} = \frac{1}{c_{L,0}}\sqrt{\frac{\Delta p}{\rho_{L,0}}} \tag{5.60}$$

where $\rho_{L,0}$ is the density of surrounding liquid at initial equilibrium and μ_L is the liquid viscosity. The gas density at $p_{G,0}$ is $\rho_{G,0}$. Parameters of gas in it's EOS equation (Eq:5.7) are $\Gamma_G = \gamma_G = 1.4$ and $\Pi_G = 0$. Meanwhile, the EOS equation of liquids are $\Gamma_L = 5.5$ and $\Pi_L = \frac{1}{\Gamma_L}\frac{\Delta p}{Ma^2} - p_\infty$ (Refer Eq:A.47 and Eq:5.60). There are two more non-dimensional numbers, viscosity ratio $\frac{\mu_G}{\mu_L}$ and density ratio $\frac{\rho G,0}{\rho_{L,0}}$, which along with P, Re, We and Ma uniquely represents the system. The set of non-dimensional numbers used in this test case are

listed in 5.1.

The result is compared with the solution of Rayleigh-Plesset equation [77] and Keller-Miksis equation [78], and also with [64]. For consistency of comparison of result with solution of Rayleigh-Plesset equation, we initialize the liquid pressure ($p(r, 0^+)\ \forall\ r > R_0$) as predicted by R-P equation in the incompressible limit of liquid $p(r, t = 0^+) = p_{\infty,0} + \Delta p\ (1 - \frac{R_0}{r})$.

$P = \frac{p_{\infty,0}+\Delta p}{p\infty,0}$	$\frac{\rho_{G,0}}{\rho_{L,0}}$	$\frac{\mu_G}{\mu_L}$	Re	We	Ma
10	10^{-3}	10^{-2}	10	10	0.1

Table 5.1: Non-dimensional parameters used for the weakly collapse of a spherical bubble collapse test case in Sec:5.4.1.

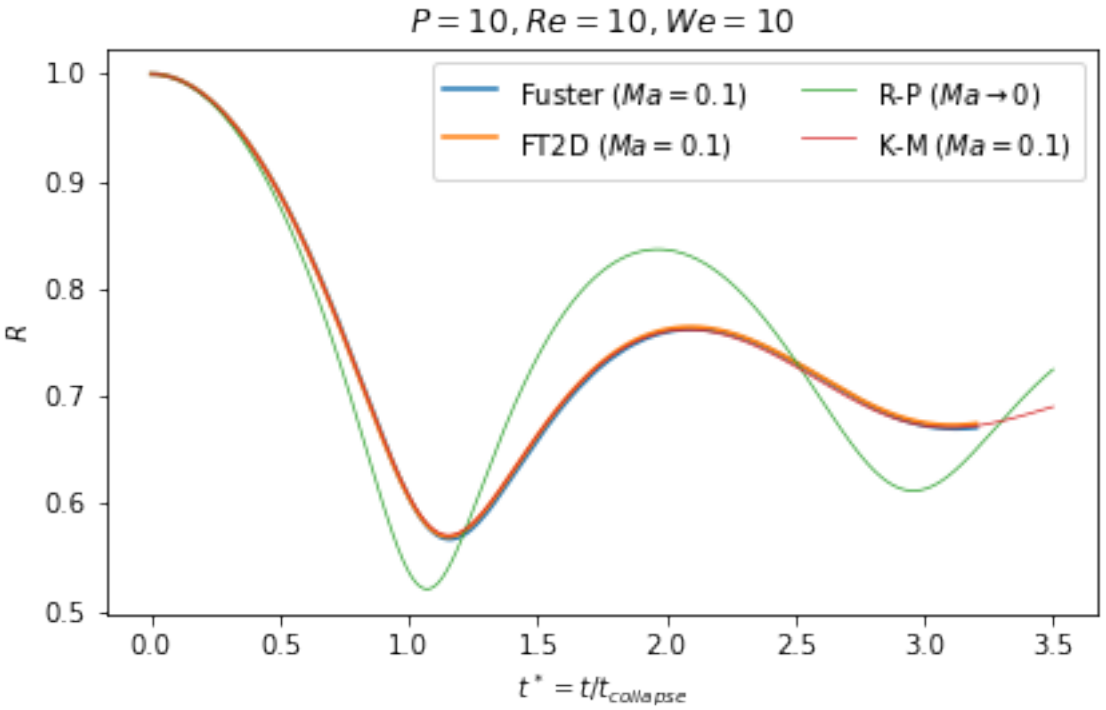


Figure 5.2: Evolution of non-dimensional radius $R^*(t^*) = \frac{R(t^*)}{R_0}$ with non-dimensional time $t^* = tU/R_0 = \sqrt{\frac{\Delta p}{\rho_L}}\frac{1}{R_0}\ t$.

Chapter 6

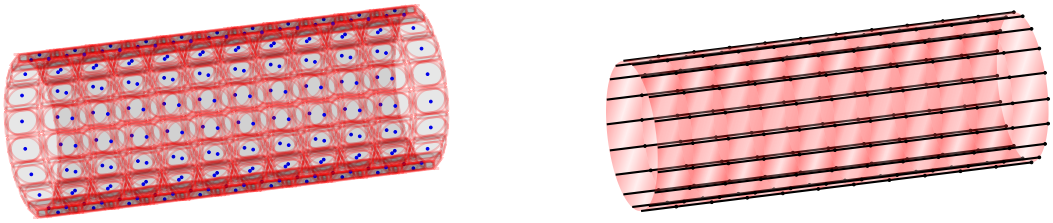
Cavitation of Micro-Bubbles in Blood Vessel

The immersed boundary method, originally implemented by [30]were used to simulate the interaction of heart valves with blood flow.

The most simplified model (regarding the vessel) is assuming the vessel is rigid, and you can look into the cavitation dynamics of the bubble to an acoustic perturbation. The expansion ratio of the bubble in the rigid tube is reduced (compared to a bubble in an infinite bulk of liquid) [8].The rigid tube assumption is an inadequate approximation to a blood capillary vessel because, in reality, they are highly compliable.

6.1 Immersed Boundary Method: Fiber Mechanics

[]This section details how to model the FSI interaction of blood vessel wall with compressible fluids. In fig:6.1, its represents a thin-walled, linearly arranged array of epithelial cell walls can be represented as



(a) *(b)*

Figure 6.1: The blood capillary wall is comprised of linearly arrayed epithelial cells can be considered as arrays of fibers.

6.1.1 Governing Equations

A bundle of fibers that represents a thin membrane Γ in $\mathbb{R}^3$ can be represented by a continuous mapping of $(q, r) \in \mathbb{R}^2$. The membrane Γ is immersed in a fluid that occupies $\Gamma \subset \mathbb{R}^3$. If the fluid is incompressible, and the membrane has the same density as that of the fluid, then the governing equations for the coupled system can be expressed in Euler-Lagrangian formulation as given in section:2.3 of the chapter:2.

6.1.2 Membrane Force (Fibers Mechanics)

The force imparted by the membrane on the fluid, $\boldsymbol{f}$, can be calculated from the deformation on the membrane Γ.

If the membrane is assumed to be comprised of continuous bundles of independent fibers, we may assume the local strain energy density functional is independent of cross-fibre strain. If the paramter r represent a unique fibre and s represents the arc length along the fibre then $\overline{\mathcal{E}}(r,s,t)$ can be written in the form

$$\overline{\mathcal{E}}(r,s,t) = \overline{\mathcal{E}}(r,s,t)\big(\frac{\partial}{\partial s}\boldsymbol{x}^s(r,s,t), \frac{\partial^2}{\partial s^2}\boldsymbol{x}^s(r,s,t), .., ..\big) \tag{6.1}$$

If we write the local energy density as the sum of extensional elastic energy and bending energy, we can express it as

$$\overline{\mathcal{E}}(r,s,t) = \frac{1}{2}k_t\left[\left\|\frac{\partial \boldsymbol{x}^s}{\partial s}\right\| - 1\right]^2 + \frac{1}{2}k_b\left\|\frac{\partial^2 \boldsymbol{x}^s}{\partial s^2}\right\|^2, \tag{6.2}$$

where k_t and k_b are the coefficients of elasticity [] . There are other formulations for elastic energy density like [], etc. If the elastic energy model follows the Eq:6.5, the force density can be written as (refer section:6.1.3 for the derivation)

$$\overline{\boldsymbol{F}}(r,s,t) = \frac{\partial}{\partial s}(T\hat{\boldsymbol{t}}) - k_b\frac{\partial^4}{\partial s^4}\boldsymbol{x}^s \tag{6.3}$$

where $T = k_t(\|\partial \boldsymbol{x}^s/\partial s\| - 1)$ is the tension in the discrete fibre element and $\hat{\boldsymbol{t}} = (\partial \boldsymbol{x}^s/\partial s)/(\|\partial \boldsymbol{x}^s/\partial s\|)$ is the tangent along the fibre at $\boldsymbol{x}^s$

6.1.3 Discretisation of Membrane Force Density

By definition, force density in the discrete form, $\boldsymbol{F}_{h_i} := \boldsymbol{F}(r,s_i,t)$, at the Lagrangian point $\boldsymbol{x}_i^s := \boldsymbol{x}^s(r,s_i,t)$ on a fiber designated by r at time t can be written as

$$\overline{\boldsymbol{F}}_i = -\frac{1}{\Delta r \Delta s}\frac{\partial E_h}{\partial \boldsymbol{x}_i^s} \tag{6.4}$$

The elastic energy functional, using the Eq:6.5, can be written in the discretized form as

$$E_h[(.,.,t)] = \sum_i \Delta r\left\{\sum_j \Delta s\left\{\frac{1}{2}k_t\left[\left\|\frac{\boldsymbol{x}_{j+1}^s - \boldsymbol{x}_j^s}{\Delta s}\right\| - 1\right]^2 + \frac{1}{2}k_b\left\|\frac{\boldsymbol{x}_{j+1}^s - 2\boldsymbol{x}_j^s + \boldsymbol{x}_{j-1}^s}{\Delta s^2}\right\|^2\right\}\right\} \tag{6.5}$$

For the above discrete energy functional, the force density can be written as the sum of elastic tension force and bending force density

$$\overline{\boldsymbol{F}}_i = \overline{\boldsymbol{F}}_{t_i} + \overline{\boldsymbol{F}}_{b_i} \tag{6.6}$$

where the tension force density $\overline{\boldsymbol{F}}_{t_i}$ and bending force density $\overline{\boldsymbol{F}}_{b_i}$ at the point s_i on the fiber identified by $'r'$ are given by

$$\overline{\boldsymbol{F}}_{t_i} = \frac{1}{\Delta s}\sum_j k_t \left[\left\|\frac{\boldsymbol{x}^s_{j+1}-\boldsymbol{x}^s_j}{\Delta s}\right\| - 1\right]\frac{\boldsymbol{x}^s_{j+1}-\boldsymbol{x}^s_j}{\|\boldsymbol{x}^s_{j+1}-\boldsymbol{x}^s_j\|}(\delta_{i,j+1}-\delta_{i,j}) \tag{6.7}$$

$$\overline{\boldsymbol{F}}_{b_i} = -\frac{1}{\Delta s^2}\sum_j k_b \left[\frac{\boldsymbol{x}^s_{j+1}-2\boldsymbol{x}^s_j+\boldsymbol{x}^s_{j-1}}{\Delta s^2}\right](\delta_{i,j+1}-2\delta_{i,j}+\delta_{i,j-1}) \tag{6.8}$$

which in its continuous is expressed in Eq:6.3

6.2 Length Scales and Time Scales

This section discusses length scales, time scales involved in the physics involved in a targeted drug delivery simulation. It also discusses the non-dimensional numbers relavant in the context. Furthermore assumptions taken are elaborated ?

6.2.1 Length Scales

Capillary Vessel Radius $R_{V,0}$

In targeted drug delivery we aims to enchance the porosity of blood capillaries also called as capillary vessels which are the smallest vessels among the supplying networks of the circulatory system and they have diameters ranging from 5-10 μm.

$$\mathcal{O}(R_{V,0}) \approx 10\ \mu m \tag{6.9}$$

Rather than just being a supply channel capillaries are also the site at which the exchange of gases (O_2, CO_2), proteins, etc happens. Depending on the organs, the capillaries have different materials to exchange and thus different structure of walls; contniuous, fenestrated and sinusoidal [?]. (move this sentence from here to ..)

Blood capillary Length, L_V

The length of capillary vessel has influence in the oscillation of confined bubbles [Oguz and Prosperetti]

Bubble Radius

The coated bubble used for enhancing ultrasound image contrast and targeted drug delivery have equilibrium radius usually ranging from $2\mu m$ to $5\mu m$ (Marmottant etal (2005), Unger etal 2003, De Jong etal 2009, John S Allen etal 2001, Hynynen 2001)

$$\mathcal{O}(R_0) \approx 2.5\ \mu m \tag{6.10}$$

Wavelength of sound in the gas

The wavelength of sound in gas (distance covered by sound during the time $2\pi/\omega$)

$$\lambda_G = \frac{2\pi}{\omega}\sqrt{\frac{\gamma R_G T_\infty}{M_G}} \tag{6.11}$$

where iω is the frequency of driving pressure wave. If $\lambda >> R_0$ we can assume that the pressure within the bubble is uniform. For air at $T_\infty = 300K$, $M_G = 28.96g/mol - K$, $\gamma_G = 1.4$.

$$\mathcal{O}(\lambda_G) \approx 2 \times 10^{-3} m \tag{6.12}$$

and we have $\lambda_G >> R_0$.

Thermal Penetration Depth

$$\delta_t = \sqrt{\frac{D_t}{\omega}} \tag{6.13}$$

where D_t is the thermal diffusivity.

6.2.2 Time Scales

Driving Frequency, ω

ω is the frequency of applied acoustic wave. (Stable frequency range)

Natural Frequency in bulk, Ω_0

The natural frequency of bubble in bulk is a measure commonly encountered in study and analysis of gas bubble cavitation.

$$\omega = \sqrt{\frac{3k}{\rho_L R_0^2}(p_{\infty,0} + \frac{2\sigma}{R_0}) - \frac{2\sigma}{\rho_L R_0^3}} \tag{6.14}$$

6.2.3 Non dimensional numbers

Mach Number

The mach in the context of cavitation is defined as the ratio of characteristic velocity of the cavity interface to the speed of the sound in liquid. Characteristic velocity of interface can be defined either as ωR_0 or $\sqrt{p_a/\rho_L}$ where p_a is the amplitude of acoustic pressure excitation.

$$Ma = \frac{1}{c_L}\sqrt{\frac{p_a}{\rho_L}} \tag{6.15}$$

Reynolds Number

$$Re = \frac{R_0}{\mu_L}\sqrt{p_a \rho_L} \tag{6.16}$$

Weber Number

Ratio of ambient pressure to Laplace pressure

$$We_0 = \frac{p_{\infty,0} R_0}{2\sigma} \tag{6.17}$$

Laplace Number

$$La = \frac{\rho_L R_0 \sigma}{\mu^2} \tag{6.18}$$

Ratio of radii ..

$$\frac{R_0}{R_{V,0}} \tag{6.19}$$

Eccentricity

$$\frac{e}{R_{V,0}} \tag{6.20}$$

where e is the distance between undeformed bubble's center and axis of vessel.

Density ratio

$$\frac{\rho_{L,0}}{\rho_{G,0}} \tag{6.21}$$

Viscosity ratio

$$\frac{\mu_L}{\mu_G} \tag{6.22}$$

Frequency Ratio

$$\frac{\omega}{\omega_0} \tag{6.23}$$

6.3 Cavitation in Blood Vessel: Axi-Symmetric Simulation

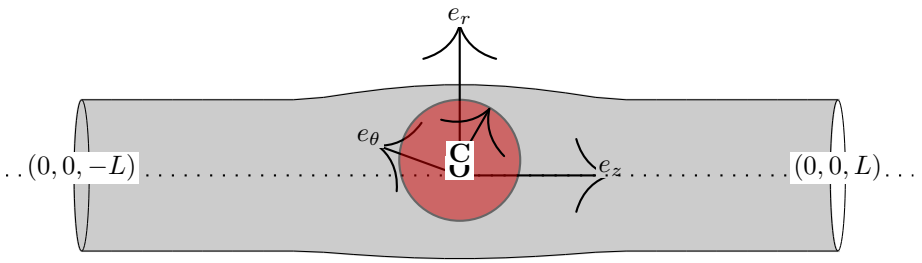


Figure 6.2: Vessel of resting radius r_v which has length $2 * l_v$. Bubble of initial radius $r_b = (r(t = 0))$ placed at $\mathbf{O}_c(e, 0, 0)$.

6.4 Non-dimensional numbers

Cavitation of microbubble inside blood capillary depends on applied pressure field, vessel compliance factors, blood matrix rheology and surrounding tissue properties. We simplify the problem by assuming blood and surrounding tissue as Newtonian fluids of similar properties.

Exciting acoustic pressure:

$$p_\infty(t) = p_0 - p_a sin(\omega t)$$

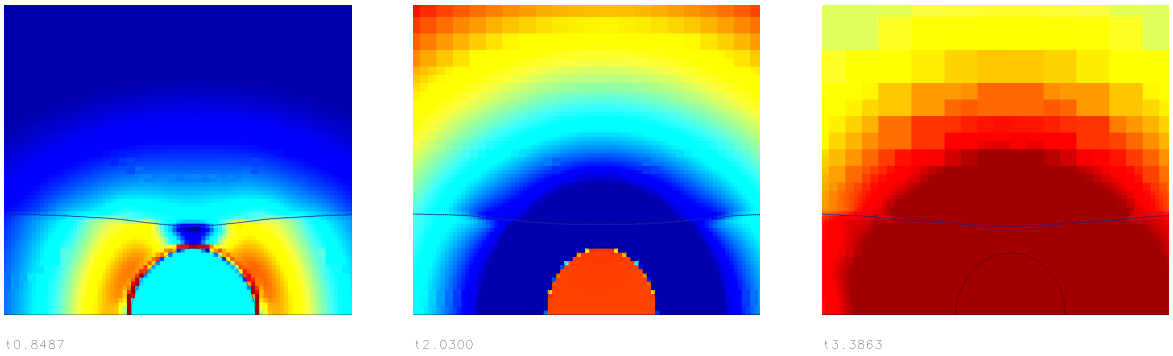
Bubble response:

$$\frac{R(t)}{R_0} = f\left(\frac{p_a}{p_0}, Re, We, K, \frac{\omega}{\omega_N}, E_1, E_2, \frac{e}{R_{v0}}, \frac{R_0}{R_{v0}}\right)$$

$$\begin{aligned} \text{where } Re &= \frac{\rho_{l_0} U_0 R_0}{\mu_L} \text{ where } U_0 = \sqrt{p_a/\rho_{l_0}} \\ We &= \frac{\rho_{l_0} {U_0}^2 R_0}{\sigma} \\ K &= \frac{\rho_{g_0} {c_{g_0}}^2}{\rho_{l_0} {c_{l_0}}^2} \\ E_1 &= \frac{k_t D \omega^2}{p_a {c_{l_0}}^2} \\ E_2 &= \frac{k_b \omega^2}{p_a D {c_{l_0}}^2} \end{aligned}$$

where E_1 and E_2 are two non-dimensional numbers involving linear and bending elasticity of the membrane. e is the eccentricity.

Radii Ratio $r = \frac{R_b}{R_v} < 1$ Eccentricity $e = \frac{E}{R_v} < 1 - r$



(a) *(b)* *(c)*

Figure 6.3: Collapse of bubble inside a vessel: Pressure contours plotted for different time. Immersed boundary and gas-liquid interface are also plotted. Non-spherical collapse is visible. ($\frac{p_\infty}{p_{\infty 0}} = 9, We = 10, Re = 10, Ma = 0.1, \frac{\mu_1}{\mu_2} = 100, \frac{\rho_1}{\rho_2} = 1000, \frac{R_{v0}}{R_0} = 1.2, \frac{k_t}{(p_\infty - p_{\infty 0})R_0} = 10000$)

Chapter 7

Conclusion

7.1 Conclusion

This thesis has focused on implementing a scalable Eulerian-Lagrangian solver capable of capturing compressible multiphase flows and fluid-structure interaction of membrane with fluid. We have explained the distributed parallel algorithms that optimize data distribution and communication.

In Chapter 2, we discussed about Eulerian-Lagrangian methods involving free surface and thin membranes. The chapter covers governing equations of E-L problems and routines involved in a E-L problem implented in an adaptive mesh refinement.

In Chapter 3, we focused on developing algorithms for the partitions of datasets that optimize intra-grid and inter-grid communications among AMR grids and Front grids. The routines in Eulerian-Lagrangian methods can be categorized as Eulerian, Eulerian-Lagrangian, and Lagrangian. Some of these operations alter the associated datasets, with the cost and frequency of subsequent load balancing emerging as pivotal factors in selecting a graph partitioning method. All these routines, active on partitioned subsets, necessitate MPI communications. Ensuring uniformity in both size and computational load across each partitioned dataset while simultaneously minimizing communication requirements stands as a paramount goal for any MPI parallel computing algorithm. This thesis focuses on partitioning the surface mesh in a manner ensuring that each processor possesses ownership of the marker points situated within its domain. This arrangement facilitates a localized relationship between marker points and leaf cells, ensuring they both reside on the same processor. With an increasing refinement level, there is a corresponding increase in the number of vertices and elements. Scalability is directly correlated to uniformity in which vertices are distributed among the processor. Scalability is also directly correlated to the number of elements per processor, provided the uniformity of distribution is not affected significantly.

We've developed a front-tracking solver that ensures both well-balanced and momentum conservation, which is discussed in Chapter 4. This solver incorporates a Laplacian smoothing routine, addressing the loss of surface smoothness during advection. The solver is also extended to compressible multiphase flows which is discussed in Chapter 5.

The author also aimed to make the AMR-based parallel front tracking code an open source project. The general architecture of source code is such that it is reusable and can be extendable to other Eulerian-Lagrangian solvers with source terms near the embedded Lagrangian surface or membrane.

7.2 Future Works

Even though most of the algorithms were designed for both 3D and 2D models, the thesis still needs to establish regridding and topology changes in 3D which. Parallel regridding algorithms in Chapter 3 are yet not completely implemented. The current version of the code avoids regridding some edges which are in the edge-cut. However, this version can simulate 3D problems involving less surface deformation but fails otherwise by introducing discontinuity in the tangent plane. Similarly, we have yet to consider the problem of parallel topology changes in 3D. Both these problems are fundamental aspects of a multiphase flow solver involving Lagrangian meshes and thus can be considered a primary future extension of this thesis.

This thesis has yet to focus on the aspect of volume conservation and higher-order accuracy interpolation during the advection of the interface. Peskin's interpolation and bilinear/trilinear velocity interpolation do not guarantee the volume conservation of the fluid components. The reason is that, say in an incompressible flow, even though the discrete velocity field is divergence-free in each grid cell, the interpolated velocity may not be necessarily divergence-free [32] [79]. The works by Peskin and Printz [79] and McDermott and Pope [80], focused on schemes that minimise error in the divergence of the interpolated velocity at the marker points. The implementation of better advection schemes like [80], [56] or [81] in the current solver are straight forward, and these methods are promising in reducing the error during advection schemes. In addition to these, the current regrid algorithms and Gaussian smoothing algorithm can also be improved with the aim of volume conservation [56] [57] [58].

A half-edge mesh [54] is promising in parallel remeshing in shared and distributed memory architecture and can thus be implemented to achieve better scalability.

Simulation of cavitation inside blood vessels involving both fluid-structure interaction and compressible flow are not yet carried out in their expected depth. We have started some work regarding the analysis of membrane stress and the natural frequency of oscillation of bubbles inside compliable vessels, which requires extensive work and is thus avoided in this thesis. This can also be considered as an extension of this thesis.

The updated documentation of the code and results used in this thesis will be available in the git repository `https://github.com/basilkottilingal/FT/`. The author is planning to publish this work comprehensively

- A-B Kottilingal, S Zaleski. Scalable parallel front tracking method in structured adaptive mesh refinement (AMR) grid. [Work in Progress]

Appendix A

Appendix

A.1 Database For Front Tracking

The computational complexity of the front tracking method arises from the basic fact that a separate mesh data of the fluid interface is required . Since they are independent from the Eulerian grid, interpolation of data has to be done between the AMR grid and the Lagrangian grid. The front represents the interface which is collection of connected triangular facets. The vertices in all the triangles are ordered in clock-wise with respect to outer normal so that "inside" and "outside" of each facets can be uniquely identified.

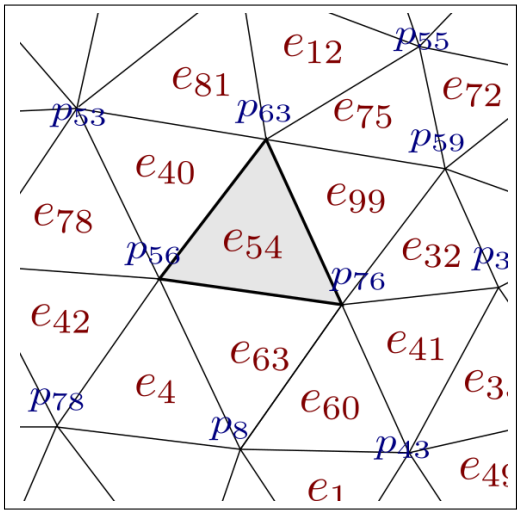


Figure A.1: Front grid maintains two list.(a) List of front-points $\mathbf{P} = \{p_{53}, p_{63}, p_{55}, p_{59}..\}$. (b) List of front-elements $\mathbf{E} = \{e_{81}, e_{12}, e_{54}, e_{59}..\}$. Each front-points store the coordinates of them, while each front-element store the integer-indices of its vertices (front-points) in CCW order and also the integer-indices of its neighbors (front-elements)

Elements	Corners	Neighbors
e_{54}	p_{76}, p_{63}, p_{56}	e_{99}, e_{40}, e_{63}
..	..	..
..	..	..

There are two stacks of data: the first one is the list of marker points also referred as "front-points" and the second one is the list of triangular facets also referred as "front-elements". A front-point or a front-element is identified by an integer. The stack has "Prev" and "Next" which are integers. (Refer Fig:A.1 and Fig:A.2)

Implementation of Sets as a Linked List

There are two types of data structures used in this chapter: linked-list (refer section:) and tree (refer section:). The data structures of frontpoints $\boldsymbol{\mathcal{V}}$ and frontelments $\boldsymbol{\mathcal{E}}$ (refer section:) are examples for linked-lists . When a linked-list , $\boldsymbol{\mathcal{V}}$, is used to implement the set $\{^{\cup}v\}$, there are $m' = |\boldsymbol{\mathcal{V}}|$ number of nodes allocated for objects ($m' \geq m$) out of which $m = |\{^{\cup}v\}|$ nodes are occupied and constitutes the set $\{^{\cup}v\}$. An object, v, is accessed from $\boldsymbol{\mathcal{V}}$ using an indexing-mapping or indexing-function , $\boldsymbol{\mathcal{V}}[i]$ that maps index i from full-index-set $\bar{\boldsymbol{\mathcal{I}}}_{\boldsymbol{\mathcal{V}}}$. We used the square bracket here for index-mapping to distinguish from other mappings.

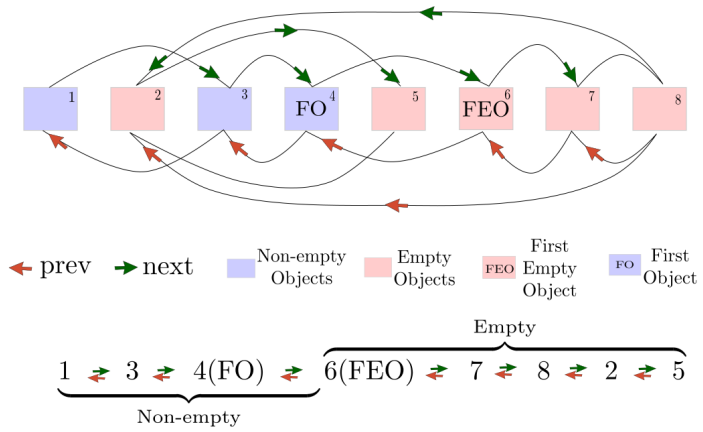


Figure A.2: The stack used to store integer indices of front-points and front-elements. Blue squares are filled part of the stack and the red squares are the non-filled.

A.1.1 Linked lists Iterators and Caches

This section discusses on the implementation of a linked list S corresponding to a set of objects of s

Simple arrays, $\mathcal{S} := \{(\mathtt{i}, \mathtt{S[i]}) \mid \mathtt{0} \leq \mathtt{i} < \mathtt{100}\}$, which doesn't need an insertion or deletion can be implemented by allocating a linear memory array like here,

```
struct s{
  datatype DATA;
};
s S[100]; // A linear array of s
```

These kind of linear array structure doesn't allow to delete objects which are placed in between.

Linked lists

If you modify the graph as

$$\mathcal{S} := \{(\mathtt{i}, \mathtt{S[ITERATOR(i)]}) \mid \mathtt{0} \leq \mathtt{i} < \mathtt{NOBJECTS} < \mathtt{100}\} \tag{A.1}$$

where the ITERATOR iterates through a LINKEDLIST of the S

$$\mathtt{ITERATOR(i)} \leftarrow \begin{cases} \mathtt{HEAD} & \text{if } \mathtt{i} = \mathtt{0} \\ \mathtt{NEXT[ITERATOR(i-1)]} & \text{if } \mathtt{0} < \mathtt{i} < \mathtt{NOBJECTS} \end{cases} \tag{A.2}$$

and the LINKEDLIST is a connected data structure, where an index i is connected to two others using PREV and NEXT, like

$$\mathtt{LINKEDLIST} := \{(\mathtt{i},\ \mathtt{(NEXT[i],\ PREV[i])}) \mid \mathtt{0} \leq \mathtt{i} < \mathtt{100}\} \tag{A.3}$$

Linked list allows you to delete from any index position. Here is a simple implementation of such a linked list where you can store a maximum of 100 objects of s which also allows to to insert or delete an instance of s

```
struct s{
  datatype DATA;
};
s S[100];
/* A linked list */
int NEXT[100];
int PREV[100];
int HEAD, NOBJECT; //  HEAD , NOBJECT in [0, 100)
```

In the above implemenetation, you can start from the head of the list (`HEAD`) and iterate through the list using `PREV[i]`.

```
int ITERATOR = HEAD;
for (int i=0; i<NOBJECTS; ++i) {
  s _s = S[i]
  // Do something with '_s'
  // ...
  ITERATOR = PREV[ITERATOR];
}
```

Refer Fig:A.3 which has implemented the above snippets. You can modify the above code to implement dynamic memory allocation. These kind of linked list [ParisCode], are advantageous for the following reasons

1. *Insert or Delete object*: Suitable for surface regrid operations
2. *integer indices for objects*: Integer indices for objects like vertex and neighbor helps to communicate across processors in parallel computing.

Caches

A cache is a mapping of an index `i` to `ITERATOR(i)` for faster iteration through a linked list. A cache $\mathcal{C}_{\mathcal{S}}$ of the graph $\mathcal{S}$ ..

$$\texttt{CACHE} := \{(\texttt{i, ITERATOR(i)}) \mid \texttt{0} \leq \texttt{i} < \texttt{NOBJECTS}\} \tag{A.4}$$

```
int CACHE[100];
int ITERATOR = HEAD;
for (int i=0; i<NOBJECTS; ++i) {
  CACHE[i] = ITERATOR;
  ITERATOR = PREV[ITERATOR];
}
//Now you can iterate through the list using CACHE
for (int i=0; i<NOBJECTS; ++i) {
  s _s = S[CACHE[i]]
  // Do something with '_s'
  // ...
}
```

Unlike the above snippet, the implementation is using memory addresses and are meant for very large databases.

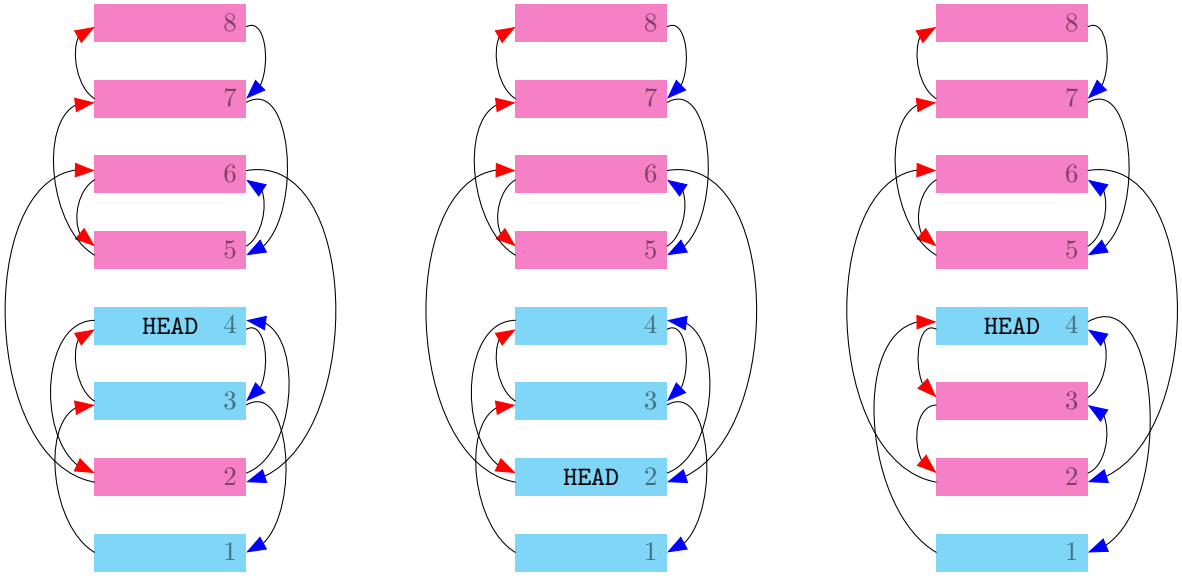


Figure A.3: *(a)*: A linked list used to store indices of frontpoints and frontelements. and are respectively the used and unused objects of the linked list. The integer index of each cell, represent the integer index of either the frontelement in $\mathcal{N}$ or the frontpoint in $\mathcal{V}$. Each objects is connected using `PREV` ⟶ and `NEXT` ⟶. All the objects previous to the `HEAD` including `HEAD` are the used part of the linked list whihc comprises the set $\{^{\cup}s\}$ and all objects next to `HEAD` are the unused part of the linked list. *(b)*: Adding an object to the list happens at the `HEAD`. `(c)`: While you can delete an object from any occuppied part of the linked list.

A.2 Morton Curve

Similar to the definition of SFC $\mathcal{Z}$ as in 2.4.5, let's define a map corresponding to each cells of parent Octree $\boldsymbol{\mathcal{O}}^+$

$$\boldsymbol{\mathcal{Z}}^+ : \mathbb{N}_{|\boldsymbol{\mathcal{O}}^+|} \mapsto \{^{\cup}c\}^+ \tag{A.5}$$

In this thesis, it employs [29] for implementing AMR grid that employs a *Morton curve (z-order indexing)* as the SFC to traverse through the discrete sets of cells. Let's use the notation $\boldsymbol{\mathcal{Z}}^+$ for the Morton curve. Using Morton curve, you can traverse through, the children $c' = (2i + i', 2j + j', 2k + k', l + 1)$ of a internal cell $c = (i, j, k, l)$ (only if all the children are leaves), using

$$\begin{aligned} \boldsymbol{\mathcal{S}}_z^+ [c'] &= \boldsymbol{\mathcal{S}}_z^+ [c] + M(c') + 1 \\ \text{given} \quad c' &= (2i + i', 2j + j', 2k + k', l + 1) \in \{^{\cup}c\} \; \forall \; i', j', k' \in \{0, 1\} \end{aligned} \tag{A.6}$$

where $M(c) \in \mathbb{N}_{2^D}$ is the relative position of a cell among its siblings defined by the Morton curve which can defined as

$$M(\ (i, j, k, l)\ ) = 4\ (i \ mod\ 2) + 2\ (j \ mod\ 2) + (k \ mod\ 2)$$

Eq:A.6 can be generalised to define the Morton curve as a SFC $\boldsymbol{\mathcal{Z}}^+ : \mathbb{N}_{|\boldsymbol{\mathcal{O}}^+|} \mapsto \{^{\cup}c\}^+$ which satisfies the following conditions

1. $$\boldsymbol{\mathcal{Z}}^{+-1}[c_0] = 0, \tag{A.7}$$

   where c_0 is the root node of AMR tree.

2. For the children $c' = (2i+i', 2j+j', 2k+k', l+1)$ and $c'' = (2i+i'', 2j+j'', 2k+k'', l+1)$ of an internal cell $c = (i, j, k, l)$ we have

$$\boldsymbol{\mathcal{S}}_z^+[c'] = \begin{cases} 1 \ + \ \boldsymbol{\mathcal{S}}_z^+[c] & \text{if} \quad M(c') = 0 \\ 1 \ + \ \boldsymbol{\mathcal{S}}_z^+[c''] + \left| \{^{\cup}d_c\}^+{}_{c''} \right| & \text{if} \quad M(c') - M(c'') = 1 \end{cases} \tag{A.8}$$

   where $\{^{\cup}d_c\}^+{}_{c''}$ is the set of all children of c''.

The above definitions can be used to iterate through the entire parent tree by starting from the root cell and every other iteration is either to its, sibling, or daughter with $M() = 0$ or ..

A.3 MAC Staggered Grid

Staggered Grid used in the discretization of NS equation.

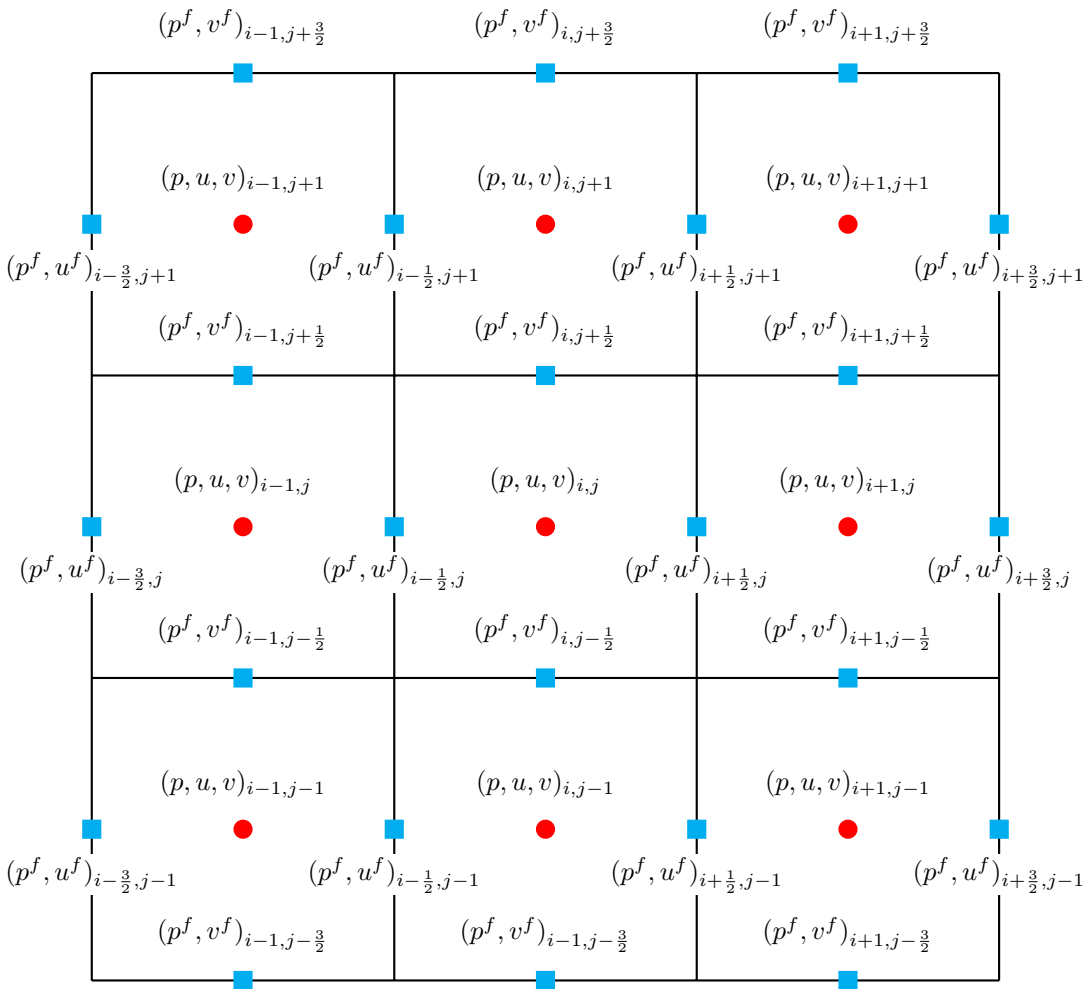


Figure A.4: Grids used for Discretrising NS

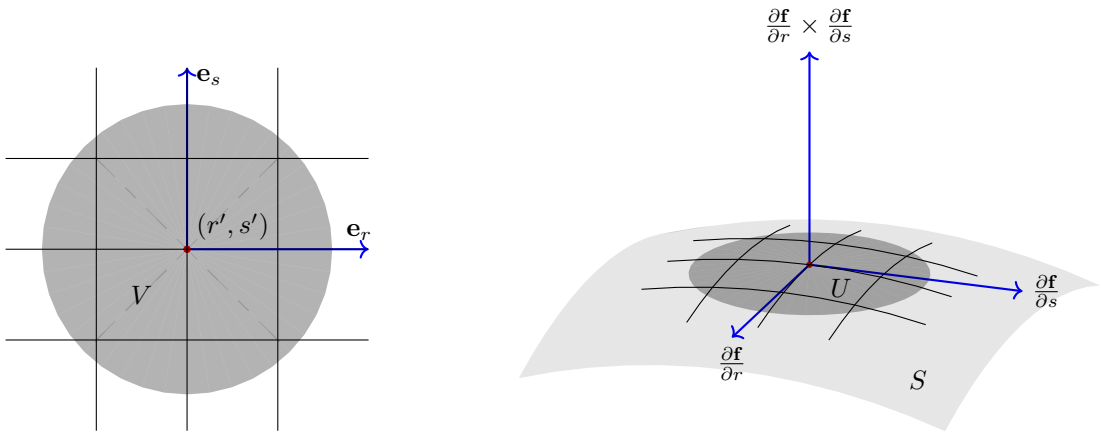
A.4 Differential Geometry, Surface Derivatives and Surface Integrals

A.4.1 A regular surface with Local parametrization

A surface S in Eulclidian space $S \subset \mathbb{R}^3$ is a regular surface if. every point of S has an open neighborhood $U \subset S$ for which there is an open subset V of $\mathbb{R}^2$ and a hemomorphism $\mathbf{f} : V \to U$ such that

1. fuction $\mathbf{f}$ is C^∞ smooth

2. for each point (r, s) of V, the two partial derivatives $\frac{\partial \mathbf{f}}{\partial r}$ and $\frac{\partial \mathbf{f}}{\partial s}$ are linearly independent as elements of $\mathbb{R}^3$



(a) (b)

Figure A.5: The neighborhood V of (r', s'), $(V \subset \mathbb{R}^2)$ is mapped to U which is the neighborhood of $\mathbf{X}(r', s')$ by a smooth function. The directional derivatives of f should be linealy independent i.e $|\frac{\partial \mathbf{f}}{\partial r} \times \frac{\partial \mathbf{f}}{\partial s}| \neq 0$

Both $\frac{\partial \mathbf{f}}{\partial r}$, $\frac{\partial \mathbf{f}}{\partial s}$ are basically the tangents of smooth surface S at $\mathbf{X}(r', s')$ any points in the inifintesimal patch U, is a linear combination of the mentioned tangent vectors. The unit normal vector is

$$\mathbf{n}(r', s') = \pm \frac{\frac{\partial \mathbf{f}}{\partial r} \times \frac{\partial \mathbf{f}}{\partial s}}{|\frac{\partial \mathbf{f}}{\partial r} \times \frac{\partial \mathbf{f}}{\partial s}|} \tag{A.9}$$

A.4.2 Fundamental forms and Curvatures

First fundamental forms, $E = \frac{\partial \mathbf{f}}{\partial r} \cdot \frac{\partial \mathbf{f}}{\partial r} = ||\frac{\partial \mathbf{f}}{\partial r}||^2$, $F = \frac{\partial \mathbf{f}}{\partial r} \cdot \frac{\partial \mathbf{f}}{\partial s}$, $G = \frac{\partial \mathbf{f}}{\partial s} \cdot \frac{\partial \mathbf{f}}{\partial s}$, $EG - F^2 = ||\frac{\partial \mathbf{f}}{\partial s} \times \frac{\partial \mathbf{f}}{\partial s}||^2$
Second Fundamental forms, $L = \frac{\partial^2 \mathbf{f}}{\partial r^2} \cdot \mathbf{n}$, $M = \frac{\partial^2 \mathbf{f}}{\partial r \partial s} \cdot \mathbf{n}$, $N = \frac{\partial^2 \mathbf{f}}{\partial s^2} \cdot \mathbf{n}$,
Gaussian (geometric mean) $K = \frac{LN - M^2}{EG - F^2}$ and mean curvatures $H = \frac{1}{2} \frac{GL - 2FM + EN}{EG - F^2}$

A.4.3 Surface Gradient

Say $(r' + \delta r, s' + \delta s)$ in the neighborhood V of (r', s') lies on the same surface contour of ϕ passing through $\mathbf{X}(r', s')$ lying on surface S, then

$$d\phi = \frac{\partial \phi}{\partial r}\delta r + \frac{\partial \phi}{\partial s}\delta s = 0 \tag{A.10}$$

which gives

$$-\frac{\delta r}{\frac{\partial \phi}{\partial s}} = \frac{\delta s}{\frac{\partial \phi}{\partial r}} = c' \tag{A.11}$$

The tangent vector to S which is also tangent to iso-contour passing through $\mathbf{X}(r', s')$ is

$$\mathbf{t}_\phi = -c'\frac{\partial \phi}{\partial s}\frac{\partial \mathbf{f}}{\partial r} + c'\frac{\partial \phi}{\partial r}\frac{\partial \mathbf{f}}{\partial s} \tag{A.12}$$

The surface gradient $\nabla_s \phi$ is a local tangent vector normal to $\mathbf{t}_\phi$, which is

$$\begin{aligned}
\nabla_s \phi = c''\mathbf{n} \times \mathbf{t}_\phi &= c'c''\frac{1}{EG - F^2}\left(\frac{\partial \mathbf{f}}{\partial r} \times \frac{\partial \mathbf{f}}{\partial s}\right) \times (-\frac{\partial \phi}{\partial s}\frac{\partial \mathbf{f}}{\partial r} + \frac{\partial \phi}{\partial r}\frac{\partial \mathbf{f}}{\partial s}) \\
&= c'''\left[-\frac{\partial \mathbf{f}}{\partial r}(-\frac{\partial \phi}{\partial s}F + \frac{\partial \phi}{\partial r}G) + \frac{\partial \mathbf{f}}{\partial s}(-\frac{\partial \phi}{\partial s}E + \frac{\partial \phi}{\partial r}F)\right]
\end{aligned}$$

for any arbitrary point $(r' + dr, s' + ds)$ in the neighborhood V of (r', s')

$$\begin{aligned}
d\phi &= \frac{\partial \phi}{\partial r}dr + \frac{\partial \phi}{\partial s}ds = \nabla_s \phi \cdot d\mathbf{x} \\
&= c'''\left[-\frac{\partial \mathbf{f}}{\partial r}(-\frac{\partial \phi}{\partial s}F + \frac{\partial \phi}{\partial r}G) + \frac{\partial \mathbf{f}}{\partial s}(-\frac{\partial \phi}{\partial s}E + \frac{\partial \phi}{\partial r}F)\right] \cdot \left(\frac{\partial \mathbf{f}}{\partial r}dr + \frac{\partial \mathbf{f}}{\partial s}ds\right) \\
&= -c'''(EG - F^2)\left(\frac{\partial \phi}{\partial r}dr + \frac{\partial \phi}{\partial s}ds\right)
\end{aligned}$$

which gives $c''' = \frac{-1}{EG-F^2}$ and thus

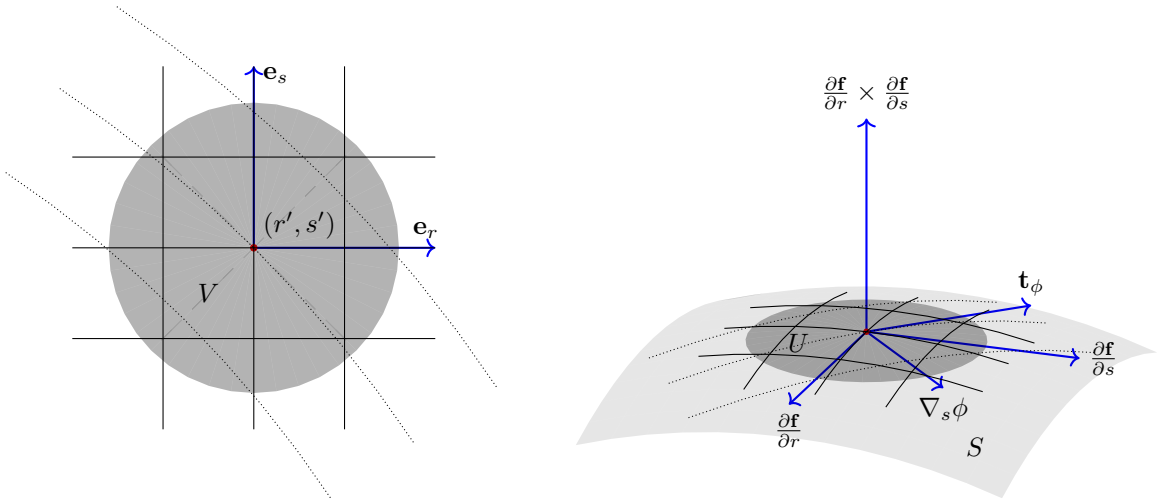
$$\nabla_s \phi = \frac{1}{EG - F^2}\left[\frac{\partial \mathbf{f}}{\partial r}(\frac{\partial \phi}{\partial r}G - \frac{\partial \phi}{\partial s}F) + \frac{\partial \mathbf{f}}{\partial s}(-\frac{\partial \phi}{\partial r}F + \frac{\partial \phi}{\partial s}E)\right] \tag{A.13}$$

Surface divergence of a vector $\mathbf{F}$ is (fixme)

$$\nabla_s \phi = \frac{1}{EG - F^2}\left[\frac{\partial \mathbf{f}}{\partial r}(\frac{\partial \phi}{\partial r}G - \frac{\partial \phi}{\partial s}F) + \frac{\partial \mathbf{f}}{\partial s}(-\frac{\partial \phi}{\partial r}F + \frac{\partial \phi}{\partial s}E)\right] \tag{A.14}$$

A.4.4 Surface of Revolution

In cylindrical coordinates the points in $\mathbb{R}^3$ are represented as (z, r, θ) with orthonormal basis vectors $\mathbf{e}_z$, $\mathbf{e}_r$ and $\mathbf{e}_\phi$ (fixme: r coincides with r of hemomorphism map) For surfaces having axial symmetry we can take $(r, s) := (z, \theta)$ where z and θ are axial coordinate and polar angle and (with $\theta \in [0, 2\pi)$) the mapping $\mathbf{f} = (z, a(z)\cos\theta, a(z)\sin\theta)$. The tangents and normal ($\partial\mathbf{f}/\partial z$, $\partial\mathbf{f}/\partial\theta$, $\pm\partial\mathbf{f}/\partial z \times \partial\mathbf{f}/\partial\theta$) for any point on the surface S can be represented in the



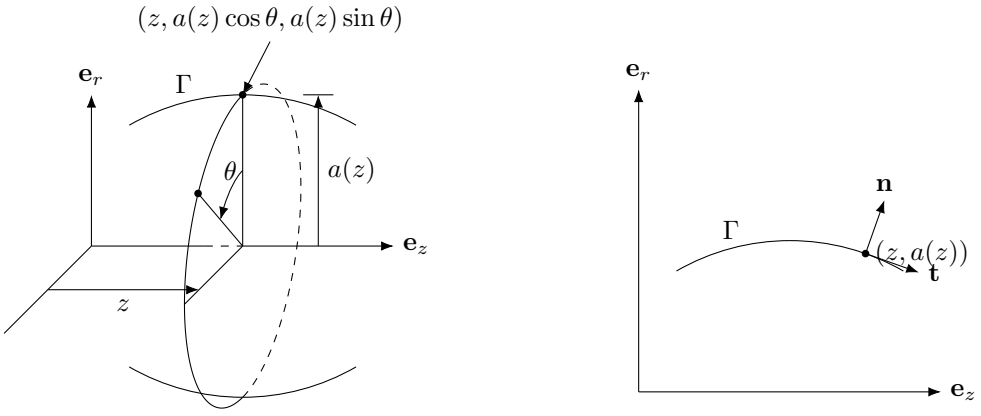
(a) (b)

Figure A.6: The neighborhood V of (r', s'), $(V \subset \mathbb{R}^2)$ is mapped to U which is the neighborhood of $\mathbf{X}(r', s')$ by a smooth function. The directional derivatives of f should be linealy independent i.e $|\frac{\partial \mathbf{f}}{\partial r} \times \frac{\partial \mathbf{f}}{\partial s}| \neq 0$

orthonormal basis as

$$\{\mathbf{e}_z + a' \cos\theta \mathbf{e}_r + a' \sin\theta \mathbf{e}_\theta,\ -a \sin\theta \mathbf{e}_r + a \cos\theta \mathbf{e}_\theta,\ \pm(aa' \mathbf{e}_z - a \cos\theta \mathbf{e}_r - a \sin\theta \mathbf{e}_\theta)\} \tag{A.15}$$

where $a' = da/dz$. We have the first fundamental forms $E = 1 + a'^2$, $F = 0$, $G = a^2$, $EG - F^2 = a^2(1 + a'^2)$,



(a) (b)

Figure A.7: (a) Surfaces of revolution. (b) Analysis in a $z - r$ plane

For cases of bubble cavitation simulation, consider only the normal towards $fluid - 1$ as defined in Chpater.2 and Chapter.3, we have unique orthonormal vectors composed out of

tangents and normal as

$$\left\{ \frac{1}{\sqrt{1+a'^2}} \left(\mathbf{e}_z + a' \cos\theta \mathbf{e}_r + a' \sin\theta \mathbf{e}_\theta\right), \; -\sin\theta \mathbf{e}_r + \cos\theta \mathbf{e}_\theta, \right. \\ \left. \frac{1}{\sqrt{1+a'^2}} \left(a' \mathbf{e}_z - \cos\theta \mathbf{e}_r - \sin\theta \mathbf{e}_\theta\right) \right\} \tag{A.16}$$

Curvature

For a surface of revolution, curvature which is the sum of principal curvatures is given by $\kappa = 2H = \frac{GL-2FM+EN}{EG-F^2}$ We have second fundamentals $L = -a''/\sqrt{1+a'^2}$, $M = 0$ $N = a/\sqrt{1+a'^2}$ which gives

$$\kappa = \frac{-a''}{\left(1+a'^2\right)^{3/2}} + \frac{1}{a} \frac{1}{\left(1+a'^2\right)^{1/2}} \tag{A.17}$$

Surface Gradient

Using Eq:A.13, we have the surface gradient of an arbitrary scalar as

$$\nabla_s \phi = \frac{1}{a^2 \left(1+a'^2\right)} \left\{ \left(\mathbf{e}_z + a' \cos\theta \mathbf{e}_r + a' \sin\theta \mathbf{e}_\theta\right) \left(a^2 \frac{\partial \phi}{\partial z} - 0\right) \right. \\ \left. + \left(-a \sin\theta \mathbf{e}_r + a \cos\theta \mathbf{e}_\theta\right) \left(-0 + \left(1+a'^2\right) \frac{\partial \phi}{\partial \theta}\right) \right\} \tag{A.18}$$

Axi Symmetric

For axi-symmetric case we have the equation is independent of θ, so we can take $\theta = 0$. We also have $\frac{\partial \phi}{\partial \theta} = 0$. So we have orthormal tangents and normal

$$\{\mathbf{t}, \mathbf{e}_\theta, \mathbf{n}\} = \{\frac{1}{\sqrt{1+a'^2}} \mathbf{e}_z + \frac{a'}{\sqrt{1+a'^2}} \mathbf{e}_r, \; \mathbf{e}_\theta, \; -\frac{a'}{\sqrt{1+a'^2}} \mathbf{e}_z + \frac{1}{\sqrt{1+a'^2}} \mathbf{e}_r\} \tag{A.19}$$

and surface derivative

$$\nabla_s \phi = \frac{1}{\left(1+a'^2\right)} \left(\mathbf{e}_z + a' \mathbf{e}_r\right) \frac{\partial \phi}{\partial z} + \frac{1}{a} \left(\mathbf{e}_\theta\right) \frac{\partial \phi}{\partial \theta} = \frac{1}{\sqrt{1+a'^2}} \frac{\partial \phi}{\partial z} \mathbf{t} \tag{A.20}$$

For problems in $\mathbb{R}^3$, we can represent

A.5 Volume fraction from the Front (Front2Vof Algorithm)

This section discusses in detail the algorithm to evaluate volume fraction from the front. Volume fraction in a cell, f, is defined as

$$f = \frac{1}{h^3}\int_{\Omega_c} H(x,y,z)dv \tag{A.21}$$

(where $H(x,y,z)$ is the heaviside function defined in 4.14) is the volume fraction in the control volume occupied by $fluid-1$. Let us define the following before explaining the algorithm,

- Ω_c as the cubic control volume. For simplification, we take the control volume has one vertex at the origin, as $\Omega_c = \{(x,y,z) \mid 0 \leq x,y,z \leq h\})$. The interior of the control volume (the interior of the closed set Ω_c) is $\Omega_c^{\mathrm{o}} = \Omega_c \setminus \partial\Omega_c$
- We define the top face of the control volume as $\partial\Omega_{c;z=h} = \{(x,y,z) \mid 0 \leq x,y \leq h \text{ and } z = h\}$ with its interior $\partial\Omega^{\mathrm{o}}_{c;z=h}$.
- We define the top-right edge of the control volume as the intersection of the top and right faces, which gives $\partial\Omega_{c;z=h} \cap \partial\Omega_{c;y=h} = \{(x,y,z) \mid 0 \leq x \leq h \text{ and } y,z = h\}$ with its interior $\partial\Omega^{\mathrm{o}}_{c;z=h} \cap \partial\Omega^{\mathrm{o}}_{c;y=h}$
- we define Γ as the oriented C^0 surface (connected triangular edges/facets) that represents the fluid-fluid interface.
- The subsets of Γ, *(i)* $\Gamma \cap \Omega_c$, *(ii)* $\Gamma \cap \partial\Omega_{c;z=h}$ and *(iii)* $\Gamma \cap \partial\Omega_{c;z=h} \cap \partial\Omega_{c;y=h}$ are the portion of the interface that intersects the control volume, the top face, and the top-right edge, respectively.

The algorithm defined in the following section does not take into consideration two special cases

- **Special Case 1**: When the cell is empty

$$f = 0 \text{ when } \Omega_c \subseteq \Omega_0$$

- **Special Case 2**: When the cell is full

$$f = 1 \text{ when } \Omega_c \subseteq \Omega_1$$

and in both cases $\Omega_c^{\mathrm{o}} \cap \Gamma = \emptyset$. These cases are specially identified and treated. For the rest of the cases with mixed cells ($\Omega_c^{\mathrm{o}} \cap \Gamma \neq \emptyset$), the algorithm is defined below.

We can say the volume belonging to $fluid-1$ as the algebraic summation of volume under an infinitesimal patch of area dA and it is a projection on the bottom plane. The volume fraction of the cell can be evaluated by integrating the portion of the surface inside the control volume as

$$f = \frac{v}{h^3} = \frac{1}{h^3}\int_{\Omega_c\cap\Gamma} (z\,\hat{\boldsymbol{n}}\cdot\boldsymbol{e}_z)\,dA$$

where v is the volume under the surface $\int_{\Omega_c\cap\Gamma}$. Evaluation of f used in the above equation fails if there are some interior points on the top face with $H(\boldsymbol{x}=1)$ i.e $\partial\Omega^{\mathrm{o}}_{c;z=h} \cap \Omega_1^{\mathrm{o}} \neq \emptyset$ So, we rewrite the above equation as

$$f = \frac{1}{h^3}\int_{\Omega_c\cap\Gamma} (z\,\hat{\boldsymbol{n}}\cdot\boldsymbol{e}_z)\,dA + \frac{1}{h^2}\int\limits_{\partial\Omega_{c;z=h}} H(x,y,h)dx\,dy \quad \text{given } \Omega_c^{\mathrm{o}} \cap \Gamma \neq \emptyset \tag{A.22}$$

The second term is the fraction of area on the top face $\Omega_{c;z=h}$ belonging to $fluid - 1$. There are 3 cases as follows

$$\frac{1}{h^2}\int\limits_{\partial\Omega_{c;z=h}} H(x,y,h)dx\,dy \begin{cases} = 0 & \text{if } \partial\Omega^{\mathrm{o}}_{c;z=h} \cap \Gamma = \emptyset \text{ with } \partial\Omega_{c;z=h} \subset \Omega_0 \\ & \text{(empty top face cell)} \\ \in (0,1) & \text{if } \partial\Omega^{\mathrm{o}}_{c;z=h} \cap \Gamma \neq \emptyset \\ & \text{(mixed top face cell)} \\ = 1 & \text{if } \partial\Omega^{\mathrm{o}}_{c;z=h} \cap \Gamma = \emptyset \text{ with } \partial\Omega_{c;z=h} \subset \Omega_1 \\ & \text{(full top face cell)} \end{cases} \tag{A.23}$$

The full top face cell can be identified separately from the full top face cell case with the negative sign of $\frac{v}{h^3}$. We can combine all three cases using

$$f = F_v\left(\frac{1}{h^3}\int_{\Omega_c\cap\Gamma}(z\,\hat{\boldsymbol{n}}\cdot\boldsymbol{e}_z)\,dA + \frac{a}{h^2}\right) \tag{A.24}$$

where $\frac{a}{h^2}$ is the area fraction on the top face, calculated only in the case of mixed top face cell, and it is taken as 0 otherwise. The function F_v is defined as

$$F_v(v) = \begin{cases} v & \text{if } v \geq 0 \\ 1+v & \text{if } v < 0 \end{cases} \tag{A.25}$$

Let's now see how the area fraction is calculated for the case where the top face is a mixed-face cell. The area fraction can be calculated as the area under the curve with the correction term similar to Eq:A.22. $\partial\Omega_{c;z=h} \cap \Gamma$

$$\frac{a}{h^2} = \frac{1}{h^2}\int_{\Gamma\cap\partial\Omega_{c;z=h}} (y\ \hat{\boldsymbol{n}}_\perp \cdot \boldsymbol{e}_y)\ dl + \int_{\Gamma\cap\partial\Omega_{c;z=h}\cap\partial\Omega_{c;y=h}} H(x,h,h)dx \quad \text{given } {}^{\partial\Omega^{\circ}_{c;z=h}\cap\Gamma\neq\emptyset} \tag{A.26}$$

Taking into consideration all cases of empty, mixed, and full fractions on the top-right edge

$$\frac{a}{h^2} = F_v\left(\frac{1}{h^2}\int_{\Gamma\cap\partial\Omega_{c;z=h}} (y\ \hat{\boldsymbol{n}}_\perp \cdot \boldsymbol{e}_y)\ dl + \frac{e}{h}\right) \tag{A.27}$$

where e is the portion of the edge on the top-right edge that belongs to $fluid - 1$. $\hat{\boldsymbol{n}}_\perp$ is the normal to the infinitesimal patch dl which can be found as $\hat{\boldsymbol{n}}_\perp = (\hat{\boldsymbol{n}} - (\hat{\boldsymbol{n}} \cdot \boldsymbol{e}_z)\boldsymbol{e}_z)/(||\ .\ ||_2)$ The correction term, the edge fraction on the top-right edge, in eq:A.26 can be evaluated by summing the points that are intersected by the surface and the top-right face.

$$\frac{e}{h} = F_v\left(\frac{1}{h}\sum_{\Gamma\cap\partial\Omega_{c;z=h}\cap\partial\Omega_{c;y=h}} x\ sign(\hat{\boldsymbol{n}} \cdot \boldsymbol{e}_x)\right) \quad \text{given } \partial\Omega^{\circ}_{c;z=h} \cap \partial\Omega^{\circ}_{c;y=h} \cap \Gamma \neq \emptyset \tag{A.28}$$

We can write the equation to evaluate the volume fraction into a single equation as

$$f = F_v\left(\frac{1}{h^3}\int_{\Omega_c\cap\Gamma} (z\ \hat{\boldsymbol{n}} \cdot \boldsymbol{e}_z)\ dA + F_v\left(\frac{1}{h^2}\int_{\Gamma\cap\partial\Omega_{c;z=h}} (y\ \hat{\boldsymbol{n}}_\perp \cdot \boldsymbol{e}_y)\ dl + F_v\left(\frac{1}{h}\sum_{\Gamma\cap\partial\Omega_{c;z=h}\cap\partial\Omega_{c;y=h}} x\ sign(\hat{\boldsymbol{n}} \cdot \boldsymbol{e}_x)\right)\right)\right) \tag{A.29}$$

Intuitively, you can write the equation for a 2D square control volume $\Omega_c = \{(x,y)\,|\,0 \leq x,y \leq 1\}$ and oriented smooth curve Γ

$$f = F_v\left(\frac{1}{h^2}\int_{\Gamma} (y\ \hat{\boldsymbol{n}}_\perp \cdot \boldsymbol{e}_y)\ dl + F_v\left(\frac{1}{h}\sum_{\Gamma\cap\partial\Omega_{c;y=h}} x\ sign(\hat{\boldsymbol{n}} \cdot \boldsymbol{e}_x)\right)\right) \tag{A.30}$$

The evaluation of the volume fraction described above is illustrated with an example in A.8

Volume fraction in the cells with no surface intersection, $\Gamma \cap \Omega_c = \emptyset$, is either 0 or 1. volume fraction in such cell at time step $(n+1)$ can be determined using

$$f^{n+1} = \begin{cases} 1 & \text{if } f^n \geq \frac{1}{2} \\ 0 & \text{if } f^n < \frac{1}{2} \end{cases} \tag{A.31}$$

provided the CFL condition CFL $< \frac{1}{2}$. In all other cells, the volume fraction is determined using the Eq:A.22. The correction terms $\frac{a}{h^2}$ and $\frac{e}{h}$ need to be evaluated only when $\Gamma \cap \partial\Omega_{c;z=h} \neq \emptyset$ and $\Gamma \cap \partial\Omega_{c;z=h} \cap \partial\Omega_{c;y=h} \neq \emptyset$ respectively.

The direction used in volume integration, area integration, and edge summation are $\boldsymbol{e}_z$,

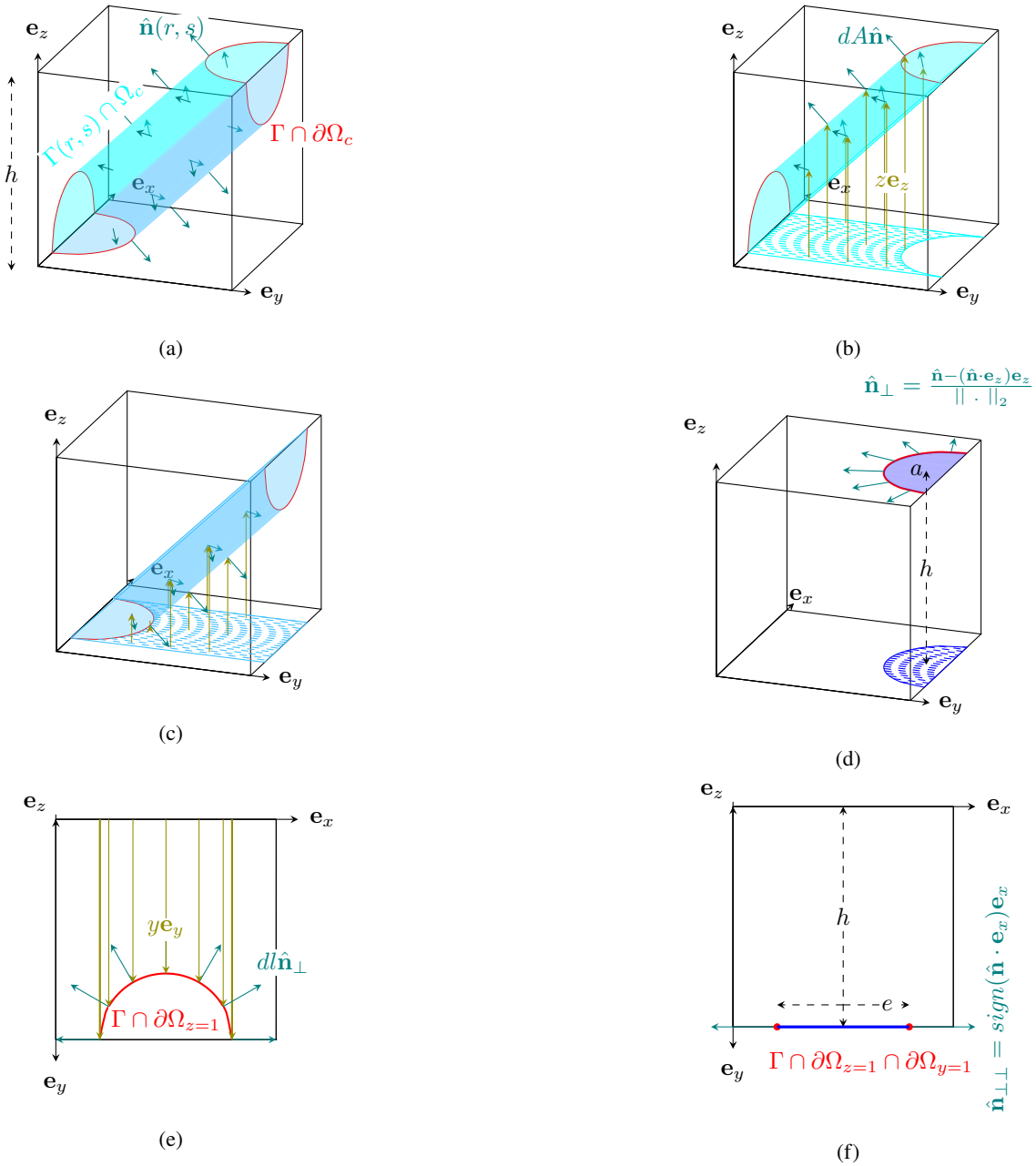
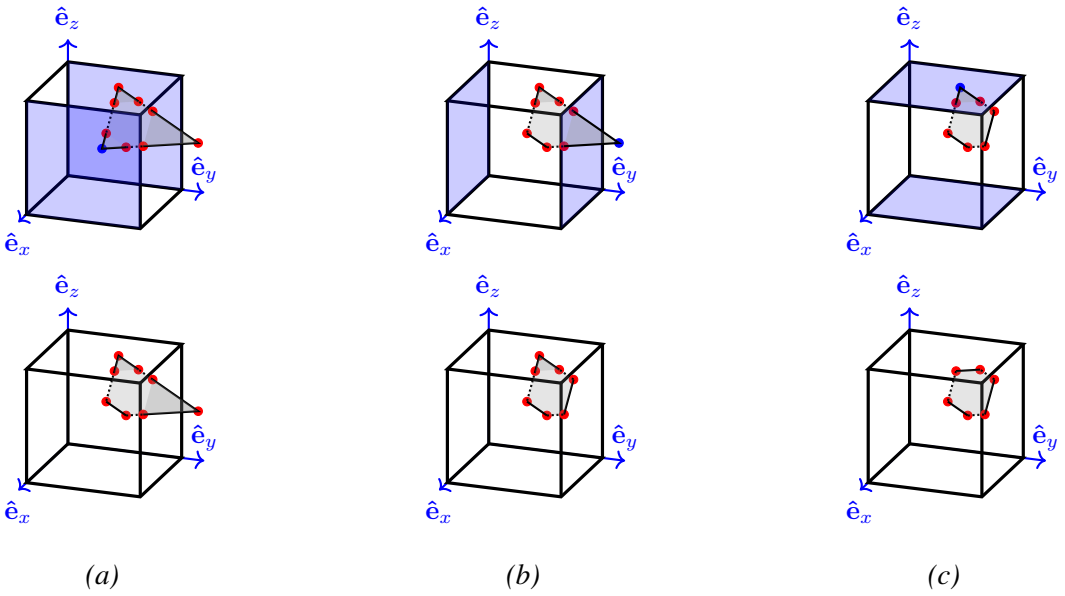


Figure A.8: Calculating void fraction inside a cubic cell intercepted by a front: *(a)* A cylinder $\Gamma := \{(x,y,z) \mid F(x,y,z) = 0\}$ where $F(x,y,z) = (x-0.5)^2 + (y-z)^2 - 0.3^2$ intersects the cubic control volume $\Omega_c := \{(x,y,z) \mid 0 \leq x,y,z \leq h\}$. We define the fluid occupying the cylinder as $fluid-1$ and the one outside as $fluid-0$. Volume fraction, f defined as $f = \frac{1}{h^3}\int_{\Omega_c} H(x,y,z)dv$ (where $H(x,y,z)$ is the heaviside function defined in 4.14) is the volume fraction in the control volume occupied by $fluid1$.It can be evaluated as $f = \frac{a}{h^2} + \frac{1}{h^3}\int_{\Omega_c\cap\Gamma}(z\ \hat{\boldsymbol{n}}\cdot\boldsymbol{e}_z)\ dA$ where a is the area on the top face, $\partial\Omega_{c;z=h}(=\{(x,y,z) \mid 0 \leq x,y \leq 1, z = h\})$. The area on the top face that is inside the cylinder, a, is colored blue in *(d)*. The volume $\int_{\Omega_c\cap\Gamma}(z\ \hat{\boldsymbol{n}}\cdot\boldsymbol{e}_z)\ dA$ is the algebraic sum of the volume subtended between an infinitesimal patch dA on Γ and its projection onto the bottom plane $z = 0$. It is represented in *(b)* and *(c)*. The domain on which the volume integration is done, $\Omega_c \cap \Gamma$, is the portion of the surface that lies inside the control volume.*(b)* shows the volume integral with $\hat{\boldsymbol{n}}\cdot\boldsymbol{e}_z > 0$. *(c)* shows the volume integral with $\hat{\boldsymbol{n}}\cdot\boldsymbol{e}_z < 0$. The integration is split for the sake of explanation. *(d)* shows the area that lies inside the cylinder and is cut by the top face of the cube. The area fraction on the top face, $\frac{a}{h^2} = \frac{e}{h} + \frac{1}{h^2}\int_{\Gamma\cap\partial\Omega_{c;z=h}}(y\ \hat{\boldsymbol{n}}_\perp\cdot\boldsymbol{e}_y)\ dl$ where e is the portion of the edge on the top-right edge that is inside the cylinder. It is represented in the blue line segment in *(g)*). The domain over which the area integral is done, $\Gamma\cap\partial\Omega_{c;z=h}$, is the portion of the surface that intersects the top face of the control volume. *(f)* shows the area integral. *(g)* shows the portion of the edge that lies inside the cylinder and is cut by the top right edge of the cube. The edge fraction can be evaluated as $\frac{e}{h} = \frac{1}{h}\sum_{\Gamma\cap\partial\Omega_{c;z=h}\cap\partial\Omega_{c;y=h}} x\ sign(\hat{\boldsymbol{n}}\cdot\boldsymbol{e}_x)$

$\boldsymbol{e}_y$ and $\boldsymbol{e}_x$ respectively. The selection of these directions is arbitrary, and the evaluation of volume fraction can be done by arbitrarily choosing any other set, say $\{\boldsymbol{e}_{d0}, \boldsymbol{e}_{d1}, \boldsymbol{e}_{d2}\}$ with $\boldsymbol{e}_{di} \in \{\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z\}$. The above algorithm can be used even when there are multiple fronts intersecting the same control volume.

Clipping Algorithm



(a) *(b)* *(c)*

Figure A.9: Clipping of triangle by a cube: Edges of polygons are successively divided to get the polygon. Firstly cut by the faces of the cube which lie on the planes *(a)* $x = \frac{h}{2} \pm \frac{h}{2}$ if they intersect the planes, then by the faces of the cube which lie on the planes *(b)* $y = \frac{h}{2} \pm \frac{h}{2}$ and *(c)* $z = \frac{h}{2} \pm \frac{h}{2}$ respectively.

As discussed earlier we need to identify the domains $\Gamma \cap \Omega_c$, $\Gamma \cap \partial\Omega_{c;z=h}$ and $\Gamma \cap \partial\Omega_{c;z=h} \cap \partial\Omega_{c;y=h}$ for carrying out the integrations in Eq:A.29. Let us define

- $P_i(v_0, v_1, .., v_{N_v-1})$ is a convex polygon with N_v vertices and the vertices v_0, .. v_{N_v-1} are ordered cyclically.
- Set of points on the Polygon $\theta(P)$ is defined as

$$\theta(P(v_0, .., v_{N_v-1})) = \left\{ \boldsymbol{x} \;\middle|\; \boldsymbol{x} = \sum_{i=0}^{Nv-1} \left(\prod_{j=0}^{i-1} c_j \prod_{j=i}^{N_v-2} (1 - c_j) \right) v_i.\boldsymbol{X} \text{ with } \substack{0 \le c_j \le 1 \\ \forall j \in \{0,..N_v-2\}} \right\}$$

  with $\theta^o(P(v_0, v_1, .., v_{N_v-1}))$ as its interior. The definition of θ and θ^ocan be extended to a set of points on triangles $(\theta(T(v_0, v_1, v_2)))$ and points on line segments $(\theta(E(v_0, v_1)))$ by taking $N_v = 3$ and $N_v = 2$ respectively in the definition.
- In the context of front tracking, the interface is represented by a collection of discrete triangles, $\{T\}$. That gives

$$\Gamma = \bigcup_{T_i \in \{T\}} \theta(T_i)$$

Using algorithm `CLIP3D` (Algo:10, also represented in Fig:A.9), we can find a polygon which is the subset of respective triangular facet, θ_i, which represents the portion of the triangle that lies inside the cube. In this algorithm, all edges of the triangle are divided by the planes $x = \frac{h}{2} \pm \frac{h}{2}$ if they cut. New points are inserted at the point of intersection of edges and planes, making the triangle a polygon. Then all the vertices v_j of the polygon that neither lies between nor lies on the planes, i.e., $|x - \frac{h}{2}| > \frac{h}{2}$, are deleted. This is repeated in another direction to get

the final polygon. The triangle that intersects the cubic control volume will result in a polygon with a number of vertices more than zero. ($N_v > 0$).

Algorithm 10: CLIP3D : Clipping of triangle by a cube

Data: A triangle, T with ordered vertices (v_0, v_1, v_2), and a cube, $\Omega_c = \{(x, y, z) \mid 0 \leq x, y, z \leq h\}$

Result: A polygon P which is the intersection of both T and cube

Initialise P as polygon with $N_v = 3$ and vertices as v_0, v_1, v_2

for *each vertices, v_j of polygon P* $\forall j \in \{0, 1, .., N_v - 1\}$ do

  $S_{v_j} \leftarrow 0$ /* $S_{v_j} = 0$ means $v_j.\boldsymbol{X}$ doesn't lie on $\partial\Omega_c$ */

end

for *each direction,i* $\forall i \in \{0, 1, 2\}$ do

  for *each vertices, v_j of polygon P* $\forall j \in \{0, 1, .., N_v - 1\}$ do

    if *($|v_j.\boldsymbol{X}[i] - \frac{h}{2}| > \frac{h}{2}$)* then

      $I_{v_j} \leftarrow 0$ /* Vertex lies outside */

    else

      $I_{v_j} \leftarrow 1$ /* Vertex lies between or on the planes */

      if *($v_j.\boldsymbol{X}[i] == h$)* then

        $S_{v_j} \leftarrow S_{v_j}|2^i$ /* $S_{v_j}\&2^i == 1$ means $v_j.\boldsymbol{X}$ lie on $\partial\Omega_{c;\boldsymbol{x}[i]=0}$ */

      else if *($v_j.\boldsymbol{X}[i] == h$)* then

        $S_{v_j} \leftarrow S_{v_j}|2^{i+3}$ /* $S_{v_j}\&2^{i+3} == 1$ means $v_j.\boldsymbol{X}$ lie on $\partial\Omega_{c;\boldsymbol{x}[i]=h}$ */

      end

      /* '&' and '|' means bitwise AND and bitwise OR respectively */

    end

  end

  for *each edges of P with end vertices v_a, v_b* do

    if *($(v_a.\boldsymbol{X}[i] - \frac{h}{2} \pm \frac{h}{2})(v_b.\boldsymbol{X}[i] - \frac{h}{2} \pm \frac{h}{2}) < 0$)* then

      /* if the edge is split by faces of the cube $\boldsymbol{x}[i] = \frac{h}{2} \mp \frac{h}{2}$ */

      Insert vertex v_c on the edge between v_a and v_b such that

      $v_c.\boldsymbol{X}[i] \leftarrow \frac{h}{2} \mp \frac{h}{2}$ /* setting the coordinates of the new vertex v_c */

      $v_c.\boldsymbol{X}[k] \leftarrow v_a.\boldsymbol{X}[k] + (\frac{h}{2} \mp \frac{h}{2} - v_a.\boldsymbol{X}[i])\frac{v_b.\boldsymbol{X}[k] - v_a.\boldsymbol{X}[k]}{v_b.\boldsymbol{X}[i] - v_a.\boldsymbol{X}[i]}$ $\forall k \in \{0, 1, 2\} \setminus \{i\}$

      $I_{v_c} \leftarrow 1$

      $N_v \leftarrow N_v + 1$

      if *($v_c.\boldsymbol{X}[i] == 0$)* then

        $S_{v_j} \leftarrow S_{v_j}|2^i$ /* $S_{v_c}\&2^i == 1$ means $v_c.\boldsymbol{X}$ lie on $\partial\Omega_{c;\boldsymbol{x}[i]=0}$ */

      else if *($v_c.\boldsymbol{X}[i] == h$)* then

        $S_{v_j} \leftarrow S_{v_j}|2^{i+3}$ /* $S_{v_c}\&2^{i+3} == 1$ means $v_c.\boldsymbol{X}$ lie on $\partial\Omega_{c;\boldsymbol{x}[i]=h}$ */

      end

    end

  end

  for *each vertices v_j of P* $\forall j \in \{0, 1, .., N_v - 1\}$ do

    if *($I_{v_j} == 0$)* then

      Remove v_j from the polygon, P

      $N_v \leftarrow N_v - 1$

    end

  end

end

/* If triangle and cube intersects $\theta(T) \cap \Omega_c^o \neq \emptyset$, CLIP3D return a convex polygon, P with $N_v > 0$ with ordered vertices v_j */

The CLIP3D algorithm also identifies whether the vertices of P lie on any specific face of the cubic control volume. Vertex, v_j of the polygon, P that lies on the face, $\partial\Omega_{c;\boldsymbol{x}[i]=0}$ have $S_{v_j}\&2^i = 1$. Vertex, v_j of the polygon, P that lies on the face, $\partial\Omega_{c;\boldsymbol{x}[i]=h}$ have $S_{v_j}\&2^{i+3} = 1$. Inherently, it can also identify whether a vertex lies on an edge or on a corner of the cube. So vertices lying on the top face satisfies $S_{v_j}\&2^5 = 1$, vertices on top-right face satisfies $S_{v_j}\&2^5 = 1 = S_{v_j}\&2^4$.

Let's say $\{P\}$, $\{E\}$ and $\{V\}$ as the list of polygons, edges, and vertices defined as follows,

$$\{P\} = \{P_i(v_0, v_1, .., v_{N_v-1}) \mid P_i = \texttt{CLIP3D}(T_i) \text{ with} N_v \geq 3 \text{ and } \forall T_i \in \{T\}\} \tag{A.32}$$

$$\{E\} = \{E_j(v_j, v_{j+1}) \mid E_j(v_j, v_{j+1}) \text{ is an edge of } P_i \in \{P\} \text{ with } 2^5\&S_{v_j} = 1 = 2^5\&S_{v_{j+1}}\} \tag{A.33}$$

$$\{V\} = \{v_j \mid v_j \text{ is a vertex of } P_i \in \{P\} \text{ with } 2^4\&S_{v_j} = 1 = 2^5\&S_{v_j}\} \tag{A.34}$$

where $\theta(P_i)$ is the set of all points on the polygon. The subsets of Γ required in the Eq:A.29 can thus be written using the sets of $\{P\}$, $\{E\}$ and $\{V\}$

$$\Gamma \cap \Omega_c = \bigcup_{P_i \in \{P\}} \theta(P_i) \tag{A.35}$$

$$\Gamma \cap \partial\Omega_{c;z=h} = \bigcup_{E_i \in \{E\}} \theta(E_i) \tag{A.36}$$

$$\Gamma \cap \partial\Omega_{c;z=h} \cap \partial\Omega_{c;y=h} = \bigcup_{v_i \in \{V\}} \{v_i.\boldsymbol{X}\} \tag{A.37}$$

Front to VOF algorithm

Complete algorithm to evaluate void fraction in a leaf cell $c \in \boldsymbol{\mathcal{L}}$ is explained in Algo:11 (F2V3D) and represented in Fig:A.10.

Intialisation of volume fraction

The above algorithm can be used effectively in calculating the initial volume fraction f^0 at $t = 0$ in all mixed cells ($\{P\} \neq \emptyset$). For other cells, we can initialize f^0 with the parametrized or implicit function used to initialize the front. Let us say if the triangular mesh is initialized from an implicit function $F(x, y, z) = 0$ with $F(x, y, z) = 0 \; \forall \; (x, y, z) \in \Gamma$, then, the empty or full cells can be initialized with the sign of the function $F(\boldsymbol{x}_c)$ at the center of the control volume ($\boldsymbol{x}_c$)

$$f^0(\boldsymbol{x}_c) = \frac{1}{2}\left(1 + sign(F(\boldsymbol{x}_c))\right) \quad \forall \quad c \in \boldsymbol{\mathcal{L}} \tag{A.38}$$

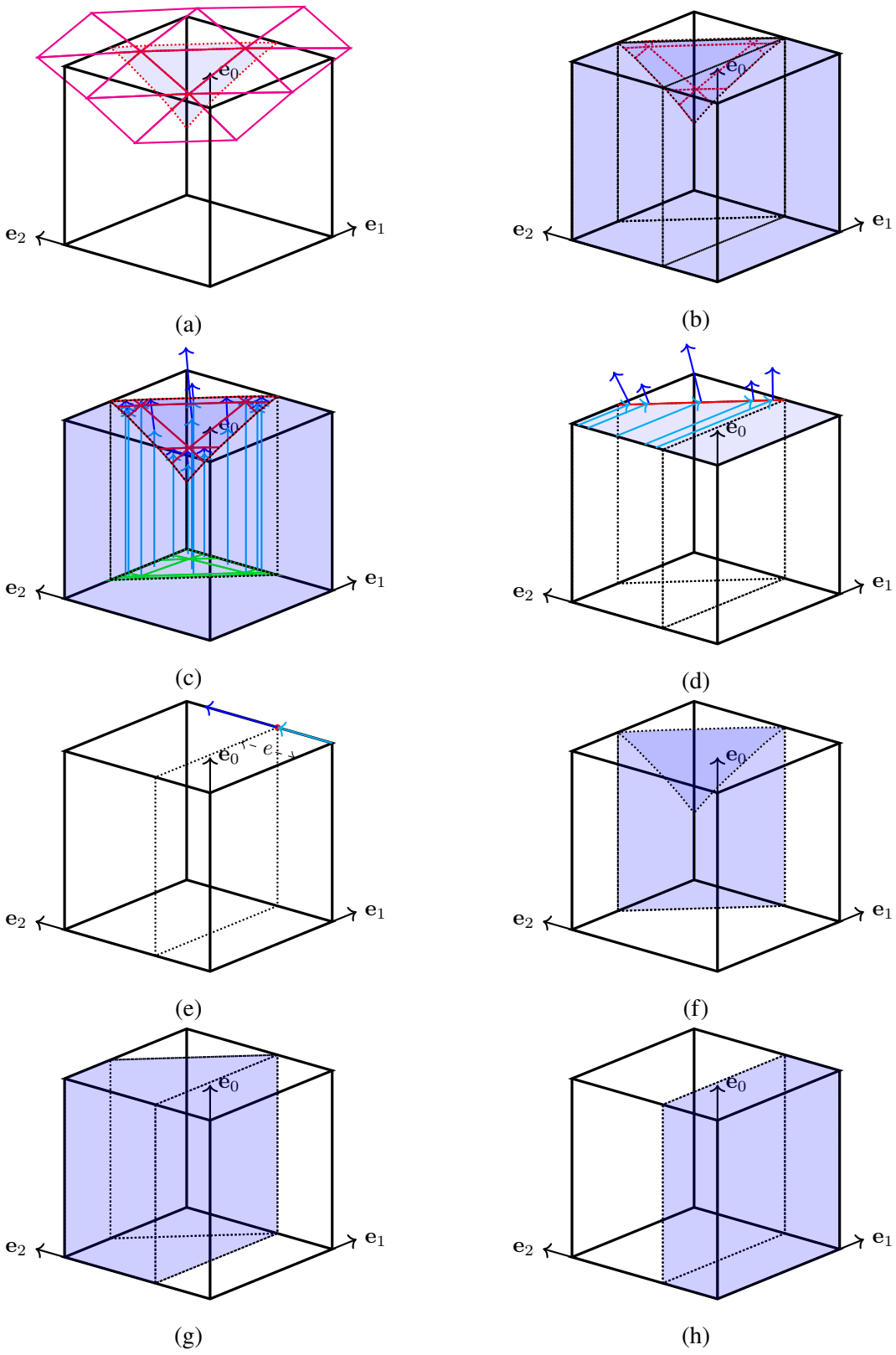


Figure A.10: (a) The cubic control volume and the front elements (triangles) in the neighborhood which intersects with the control volume (cube). (b) The polygons ($\{P\}$) are the subsets of triangles ($\{T_i\}$) is marked in red. Volume belonging to $fluid-1$ is filled blue. (c) Volume under the polygon. (d) Area under the edges on the top face. (e) Edge portion on the top-front edge belonging to $fluid-1$. Sets $\{P\}$, $\{E\}$ and $\{V\}$ are marked red in (c), (d) and (e) respectively. The subset of the cube that belongs to fluid 1 is in a filled blue color. The volume fraction calculated in (c) is corrected by the area fraction in (d), which is also corrected by the edge fraction in (e). Volume corresponding to each is represented in filled blue in (f), (g), and (h) respectively.

Algorithm 11: `F2V3D`: Finding void fraction in each computational cell from the front

Data: facets, $\{T\}$ and cubic control volume, Ω_c and volume fraction at time step n, f^n
Result: volume fraction f^{n+1} in the control volume
Find set of polygons, $\{P\}$ that intersects C.V using Eq:A.32 /* Using Algo:10 */
if $\{P\} = \emptyset$ then
  f^{n+1} is found using Eq:A.31
else
  $v \leftarrow \frac{1}{h^3} \sum_{P_i \in \{P\}} A(P_i)\ (\hat{\boldsymbol{n}}(P_i) \cdot \boldsymbol{e}_z)\ (\boldsymbol{C}(P_i) \cdot \boldsymbol{e}_z)$ /* Fractional volume under polygons */
  /* $A(P_i)$, $\hat{\boldsymbol{n}}(P_i)$ and $\boldsymbol{C}(P_i)$ are area, normal and area centroid of P_i */
  if $\{E\} \neq \emptyset$ then
    $a \leftarrow \frac{1}{h^2} \sum_{E_i \in \{E\}} L(E_i)\ (\hat{\boldsymbol{n}}_\perp(E_i) \cdot \boldsymbol{e}_y)\ (\boldsymbol{C}(E_i) \cdot \boldsymbol{e}_z)$ /* Fractional area under edges */
    /* $L(E_i)$, $\hat{\boldsymbol{n}}_\perp(E_i)$ and $\boldsymbol{C}(E_i)$ are length, normal and centroid of E_i */
    if $\{V\} \neq \emptyset$ then
      $e \leftarrow \frac{1}{h} \sum_{v_i \in \{V\}} (v_i.\boldsymbol{X} \cdot \boldsymbol{e}_x)\ (\hat{\boldsymbol{n}}(v_i) \cdot \boldsymbol{e}_x)$ /* Edge fraction with $H(\boldsymbol{x}) = 1$ on top-right edge */
      /* $n(v_i)$ surface normal at v_i */
    else
      $e \leftarrow 0$
    end
    $a \leftarrow a + F_v(e)$ /* Correction of area fraction. Eq:A.29 */
  else
    $a \leftarrow 0$
  end
  $v \leftarrow v + F_v(a)$ /* Correction of volume fraction. Eq:A.29 */
  $f^{n+1} \leftarrow F_v(v)$
end

A.5.1 Filtering the color function

The Front2VOF algorithm results in a sharp interface which requires filtering for better results [65] [82]. It can be done by applying a discrete filter on the stencil to get a smoother color function $\tilde{c}$ from the void fraction c with a weightage of $1/8$, $1/16$, $1/32$ and $1/64$ to the void fraction c in the center cell, face neighbors, edge neighbors and vertex neighbors respectively.

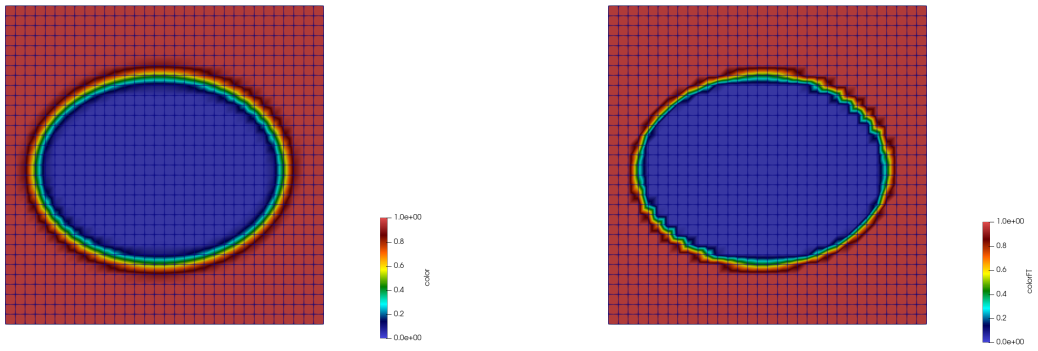
[65] [82] also uses an alternative approach with double Gaussian filtering. This method uses the filtering equation

$$G(\mathbf{r}) = A \exp\left(-\left[(\mathbf{r}\cdot\mathbf{t})^2/2\sigma_t^2 + (\mathbf{r}\cdot\mathbf{n})^2/2\sigma_n^2\right]\right) \tag{A.39}$$

where $\mathbf{r}$ is the distance from the center of the stencil, and the coefficient A is a normalizing factor. The filtering window in the tangential and normal direction of the surface are chose as $\sigma_t = h\sqrt{3}$, and $\sigma_n = \sigma_t/4$ respectively.

A.5.2 Test case to compare Front2VOF and Poisson Solver

Inorder to compare the current Front2VOF algorithm and the Poisson solver [39], we compare the evolution of kinetic energy in an oscillating 3D droplet testcase where an initial ellipsoid (Fig:**??**) with small eccentricity in one axis is allowed to oscillate in a viscous fluid. The test case is presented in [53]. The results from Front2VOF method implemented by [82] is compared with that of [53] which utilises the Poisson Solver to evaluate the color function.



(a) The color function with the Poisson approach.

(b) The color function with the Front2VOF approach.

Figure A.11: Color function in the cells calculated from the front using *(a)* Poisson method of [39] *(b)* Direct intergation from the interface [65] [82] which uses a method similar to the current work.

[width=.35]./plots/mesh$_bubble_image_1.png$

Figure A.12: Test case of the oscillating bubble/droplet: initial mesh with $D/\Delta x = 19.2$.

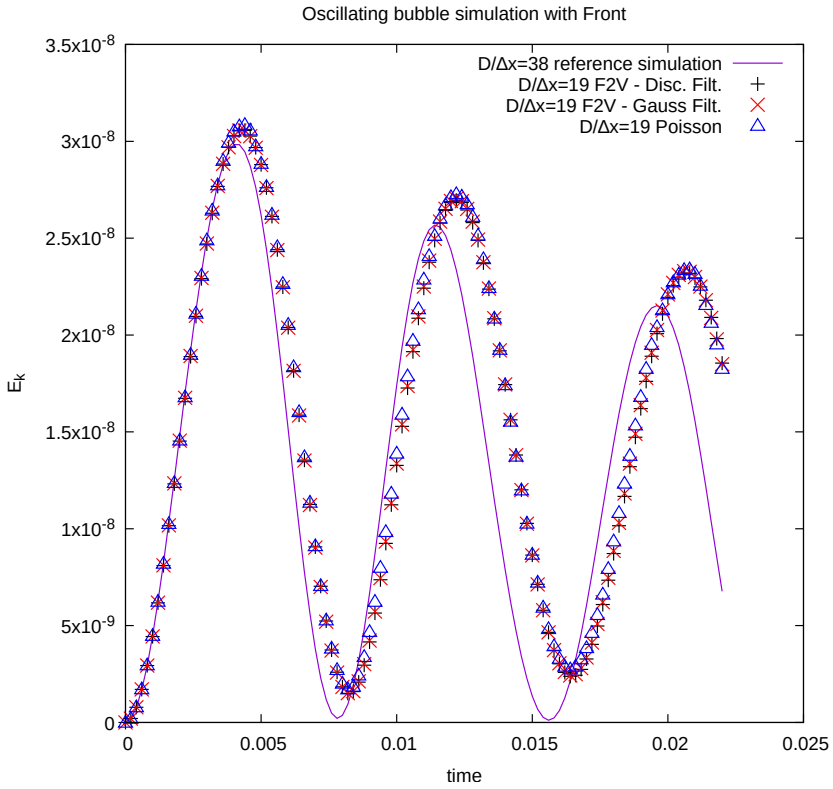
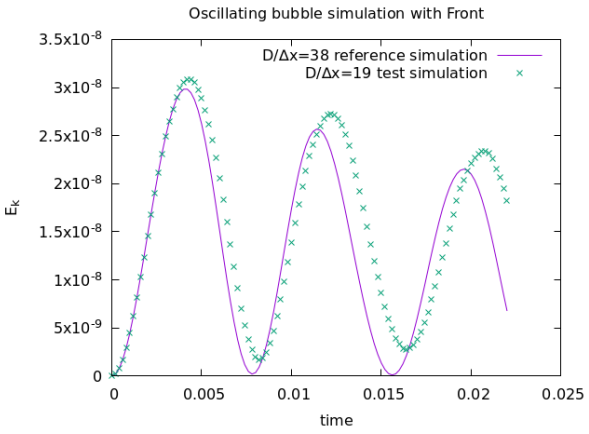
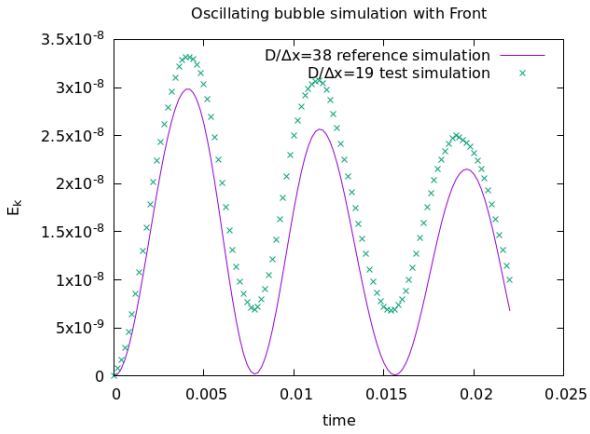


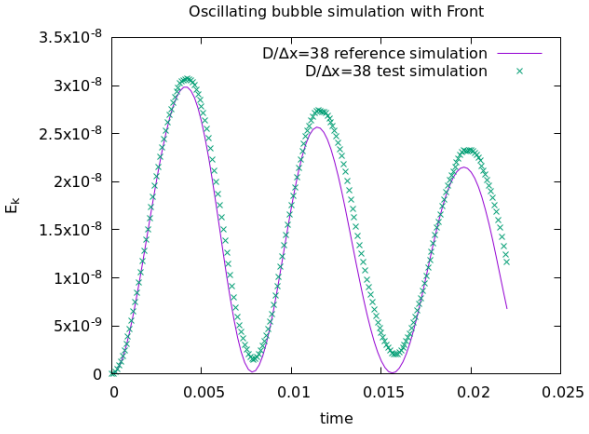
Figure A.13: Comparison of different filterings on the evolution of the kinetic energy over time. Comparison of the evolution of kinetic energy calculated by front tracking solver which uses Front2VOF algorithm (with filtering) with that [53] which uses the Poisson solver [39] for color function evaluation. (Reproduced from [82] with permission)



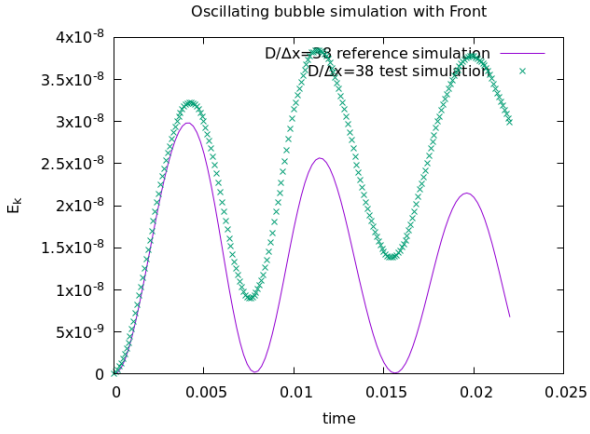
(a) $D/\Delta x = 19$, Poisson.



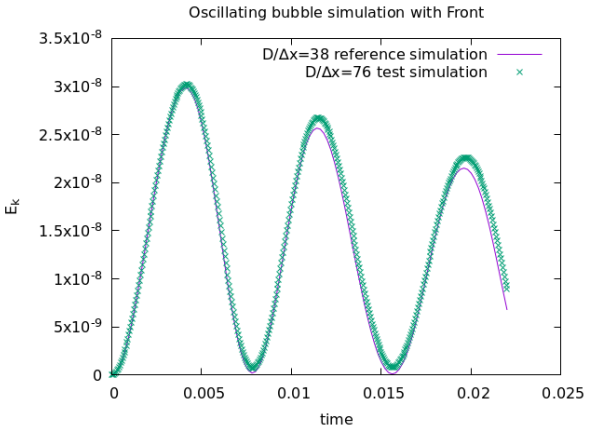
(b) $D/\Delta x = 19$, Front2VOF.



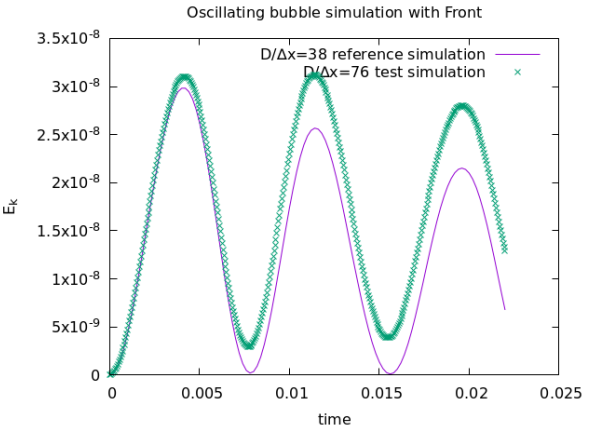
(c) $D/\Delta x = 38$, Poisson.



(d) $D/\Delta x = 38$, Front2VOF.



(e) $D/\Delta x = 76$, Poisson.



(f) $D/\Delta x = 76$, Front2VOF.

Figure A.14: Evolution of the kinetic energy over time. Comparison of the evolution of kinetic energy calculated by front tracking solver which uses Front2VOF algorithm (without filtering) with that [53] which uses the Poisson solver [39] for color function evaluation. (Reproduced from [82] with permission)

A.6 Pressure Equation

Dot product of momentum equation (sigle formualtion) and velocity (continuous across the interface)

$$u_i\left(\frac{\partial}{\partial t}(\rho u_i)+\frac{\partial}{\partial x_j}(\rho u_i u_j)\right) = u_i\left(-\tfrac{\partial p}{\partial x_i}+\tfrac{\partial \tau_{ji}}{\partial x_j}+f_{\sigma_i}\right) \tag{A.40}$$

$$\frac{\partial}{\partial t}(\frac{1}{2}\rho u_i u_i)+\frac{\partial}{\partial x_j}\left((\frac{1}{2}\rho u_i u_i)u_j\right) = \tfrac{\partial}{\partial x_j}\left(u_i(-p\delta_{ij}+\tau_{ij}+\tau_{\sigma_{i,j}})\right) - \left(-p\delta_{ij}+\tau_{ij}+\tau_{\sigma_{i,j}}\right)\tfrac{\partial u_i}{\partial x_j} \tag{A.41}$$

$$\tag{A.42}$$

energy equation (single formulation)

$$\frac{\partial}{\partial t}(\rho e+\frac{1}{2}\rho u_i u_i)+\frac{\partial}{\partial x_j}\left((\rho e+\frac{1}{2}\rho u_i u_i)u_j\right) = \frac{\partial}{\partial x_j}\left(u_i(-p\delta_{ij}+\tau_{ij}+\tau_{\sigma_{i,j}})\right) \tag{A.43}$$

The difference

$$\frac{\partial}{\partial t}(\rho e)+\frac{\partial}{\partial x_j}(\rho e u_j) = \Phi - p\frac{\partial u_j}{\partial x_j} \tag{A.44}$$

and using continuity equation

$$\frac{\partial \rho}{\partial t}+u_j\frac{\partial \rho}{\partial x_j} = -\rho\frac{\partial u_j}{\partial x_j} \tag{A.45}$$

the Eq:A.46 can be written as in the non conservative form as

$$\rho\left(\frac{\partial e}{\partial t}+u_j\frac{\partial e}{\partial x_j}\right) = \Phi - p\frac{\partial u_j}{\partial x_j} \tag{A.46}$$

Equation of speed of sound

$$c^2 = \frac{dp}{d\rho} \tag{A.47}$$

equation of state $\rho = \rho(p,e)$

$$d\rho = \left(\frac{\partial \rho}{\partial p}\right)_e dp + \left(\frac{\partial \rho}{\partial e}\right)_p de = \frac{\gamma}{c^2}dp - \frac{\rho\beta}{c_p}de \tag{A.48}$$

Eq:A.49 can be rewritten using eq:A.46 and A.45 as

$$-\rho\frac{\partial u_j}{\partial x_j} = \frac{\gamma}{c^2}\left(\frac{dp}{dt}\right) - \frac{\rho\beta}{c_p}\frac{1}{\rho}\left(\Phi - p\frac{\partial u_j}{\partial x_j}\right) \tag{A.49}$$

$$\frac{\gamma}{\rho c^2}\left(\frac{dp}{dt}\right) - \frac{1}{\rho}\frac{\rho\beta}{c_p}\frac{1}{\rho}\left(+\frac{p}{\rho c^2}\frac{dp}{dt}\right) - \frac{\beta}{\rho c_p}\Phi = -\frac{\partial u_j}{\partial x_j} \tag{A.50}$$

$$\frac{dp}{dt}\left(\frac{\gamma}{\rho c^2}-\frac{\beta p}{\rho^2 c_p c^2}\right) = \frac{\beta}{\rho c_p}\Phi - \frac{\partial u_j}{\partial x_j} \tag{A.51}$$

$$\frac{dp}{dt}\left(\frac{\gamma}{\rho c^2}-\frac{\beta^2 T}{\rho c_p}\right)=\frac{\beta}{\rho c_p}\Phi-\frac{\partial u_j}{\partial x_j} \tag{A.52}$$

For liquid $\gamma = 1$ and $\beta \approx 0$, for ideal gases $\beta = 1/T$ and in the absence of viscous dissipation, we have

$$\frac{dp}{dt}\left(\frac{1}{\rho c^2}\right)=-\frac{\partial u_j}{\partial x_j} \tag{A.53}$$

A.7 Rayleigh-Plesset Equation

Rayleigh-Plesset Equation

$$\frac{1}{\rho_L}(p_B(t) - p_\infty(t)) = R\frac{d^2R}{dt^2} + \frac{3}{2}(\frac{dR}{dt})^2 + \frac{4\nu_L}{R}\frac{dR}{dt} + \frac{2\sigma}{\rho_L R}$$

Non dimensional numbers

(Use characteristic velocity scale as collapse velocity $U_c = \sqrt{\frac{\Delta p}{\rho_L}}$ and length scale R_0 and time scale R_0/U_c)

(1) ratio of new ambient pressure to old

$$P = \frac{p_\infty}{p_{\infty,0}} = \frac{p_{\infty,0} + \Delta p}{p_{\infty,0}}$$

(with $P > 0$). In violent collapse $P \gg 1$

(2) Weber Number, We

$$We = \frac{\Delta p R_0}{\sigma}$$

(5) Reynolds Number

$$Re = \frac{R_0\sqrt{\Delta p\, \rho_L}}{\mu_L}$$

(6) Non-dim time

$$t^* = tU/R_0 = \sqrt{\frac{\Delta p}{\rho_L}}\frac{1}{R_0}\, t$$

(7) Non-dim Radius

$$R^*(t^*) = \frac{R(t^*)}{R_0}$$

(8) Non-dim Interface velocity

$$U^*(t^*) = \frac{dR^*}{dt^*} = \sqrt{\frac{\rho_L}{\Delta p}}\frac{dR}{dt}$$

We alse have

$$\frac{dR}{dt} = \sqrt{\frac{\Delta p}{\rho_L}}\frac{dR^*}{dt^*}\ ; \quad \frac{d^2R}{dt^2} = \frac{1}{R_0}\frac{\Delta p}{\rho_L}\frac{d^2R^*}{dt^{*2}}\ ; \quad \frac{p_{\infty,0}}{\Delta p} = \frac{1}{P-1}\ ; \quad \frac{p_\infty}{\Delta p} = \frac{P}{P-1}$$

R-P (Non-dimensionalised)

$$R^*\frac{dU^*}{dt^*} + \frac{3}{2}U^{*2} + \frac{4}{Re}\frac{U^*}{R^*} + \frac{2}{We}\frac{1}{R^*} = \left(\frac{1}{P-1} + \frac{2}{We}\right)\left(\frac{1}{R^*}\right)^{3\gamma} - \frac{P}{P-1}$$

which gives

$$\frac{d}{dt^*}\begin{bmatrix} R^* \\ U^* \end{bmatrix} = \begin{bmatrix} U^* \\ \frac{1}{R^*}\left[\left(P - 1 + \frac{2}{We}\right)\left(\frac{1}{R^*}\right)^{3\gamma} - P - \frac{3}{2}U^{*2} - \frac{4}{Re}\frac{U^*}{R^*} - \frac{2}{We}\frac{1}{R^*}\right] \end{bmatrix}$$

In the above equation

$$p_G(t^*) = \left[\left(\frac{1}{P-1} + \frac{2}{We}\right)\left(\frac{1}{R^*}\right)^{3\gamma}\right]\Delta p$$

$$p_\infty(t^*) = \frac{P}{P-1}\,\Delta p$$

A.8 Keller-Miksis Equation (Weekly Compressible Liquid)

Keller-Miksis equation (Dimensional)

$$\left(1-\frac{\dot{R}}{c_{L,0}}\right)R\frac{d^2R}{dt^2} + \left(1-\frac{1}{3}\frac{\dot{R}}{c_{L,0}}\right)\frac{3}{2}(\frac{dR}{dt})^2 = \left(1+\tfrac{\dot{R}}{c_{L,0}}\right)\tfrac{1}{\rho_{L,0}}\left(p_L(R,t)-p_\infty(t)\right) + \tfrac{R}{\rho_{L,0}c_{L,0}}\tfrac{d}{dt}\left(p_L(R,t)-p_\infty(t)\right) \quad \text{(A.54)}$$

Since the liquid is (slghtly) compressible the expression involves speed of sound at ambient. ALso the liquid density is noy constant and K-M eqn involves the liquid density at ambient.

KM Non- Dimensional (in collapse) In the case of collapse, apart from the initially defined non-dim numbers P, Re and We, we need one more non-dim number, the Mach number.

$$Ma = \frac{U_c}{c_{L,0}} = \frac{1}{c_{L,0}}\sqrt{\frac{\Delta p}{\rho_{L,0}}}$$

$$\begin{aligned}(1-Ma\,U^*)\,R^*\frac{dU^*}{dt^*} + \left(1-\frac{1}{3}Ma\,U^*\right)\frac{3}{2}U^{*2} = \\ (1+Ma\,U^*)\left\{\left(\frac{1}{P-1}+\frac{2}{We}\right)\left(\frac{1}{R^*}\right)^{3\gamma} - \frac{2}{We}\frac{1}{R^*} - \frac{4}{Re}\frac{1}{R^*}U^* - \frac{P}{P-1}\right\} \\ +Ma\,U^*\left\{-3\gamma\left(\frac{1}{P-1}+\frac{2}{We}\right)\left(\frac{1}{R^*}\right)^{3\gamma} + \frac{2}{We}\frac{1}{R^*} + \frac{4}{Re}\frac{1}{R^*}U^*\right\} - Ma\,\frac{4}{Re}\frac{dU^*}{dt^*}\end{aligned} \quad \text{(A.55)}$$

which gives

$$\begin{aligned}\frac{dU^*}{dt^*} = \tfrac{1}{R^*(1-Ma\,U^*)+Ma\,\frac{4}{Re}}\Big\{&(1+(1-3\gamma)Ma\,U^*)\left(\tfrac{1}{P-1}+\tfrac{2}{We}\right)\left(\tfrac{1}{R^*}\right)^{3\gamma} - \\ &\tfrac{2}{We}\tfrac{1}{R^*} - \tfrac{4}{Re}\tfrac{1}{R^*}U^* - (1+Ma\,U^*)\,\tfrac{P}{P-1} - \left(1-\tfrac{1}{3}Ma\,U^*\right)\tfrac{3}{2}U^{*2}\Big\}\end{aligned} \quad \text{(A.56)}$$

Bibliography

[1] B Dollet, P Marmottant, and V Garbin. Bubble dynamics in soft and biological matter. *Annual Review of Fluid Mechanics*, 51:331–355, 2019.

[2] S O Unverdi and G Tryggvason. A front-tracking method for viscous, incompressible, multi-fluid flows. *Journal of computational physics*, 100(1):25–37, 1992.

[3] C Peskin. The immersed interface method. *Acta Numerica*, 11:479–517, 2002.

[4] L Rayleigh. Viii. on the pressure developed in a liquid during the collapse of a spherical cavity. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 34(200):94–98, 1917.

[5] E A Neppiras. Acoustic cavitation. *Physics reports*, 61(3):159–251, 1980.

[6] S H Bloch, P A Dayton, and K W Ferrara. Targeted imaging using ultrasound contrast agents. *IEEE Engineering in Medicine and Biology Magazine*, 23(5):18–29, 2004.

[7] S Martynov, E Stride, and N Saffari. The natural frequencies of microbubble oscillation in elastic vessels. *The Journal of the Acoustical Society of America*, 126(6):2963–2972, 2009.

[8] S Qin and K W Ferrara. Acoustic response of compliable microvessels containing ultrasound contrast agents. *Physics in Medicine & Biology*, 51(20):5065, 2006.

[9] R J Price, D M Skyba, S Kaul, and T C Skalak. Delivery of colloidal particles and red blood cells to tissue through microvessel ruptures created by targeted microbubble destruction with ultrasound. *Circulation*, 98(13):1264–1267, 1998.

[10] J L Bull. The application of microbubbles for targeted drug delivery. *Expert opinion on drug delivery*, 4(5):475–493, 2007.

[11] C C Coussios and R A Roy. Applications of acoustics and cavitation to noninvasive therapy and drug delivery. *Annu. Rev. Fluid Mech.*, 40:395–420, 2008.

[12] H R Guzmán, A J McNamara, D X Nguyen, and M R Prausnitz. Bioeffects caused by changes in acoustic cavitation bubble density and cell concentration: a unified explanation based on cell-to-bubble ratio and blast radius. *Ultrasound in medicine & biology*, 29(8):1211–1222, 2003.

[13] P Prentice, A Cuschieri, K Dholakia, M Prausnitz, and P Campbell. Membrane disruption by optically controlled microbubble cavitation. *Nature physics*, 1(2):107–110, 2005.

[14] E C Everbach and C W Francis. Cavitational mechanisms in ultrasound-accelerated thrombolysis at 1 mhz. *Ultrasound in medicine & biology*, 26(7):1153–1160, 2000.

[15] A F Prokop, A Soltani, and R A Roy. Cavitational mechanisms in ultrasound-accelerated fibrinolysis. *Ultrasound in medicine & biology*, 33(6):924–933, 2007.

[16] C S Peskin. Numerical analysis of blood flow in the heart. *Journal of computational physics*, 25(3):220–252, 1977.

[17] C W Hirt and B D Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *Journal of computational physics*, 39(1):201–225, 1981.

[18] G D Weymouth and D K Yue. Conservative volume-of-fluid method for free-surface simulations on cartesian-grids. *Journal of Computational Physics*, 229(8):2853–2865, 2010.

[19] M Rudman. Volume-tracking methods for interfacial flow calculations. *International journal for numerical methods in fluids*, 24(7):671–691, 1997.

[20] D Gueyffier, J Li, A Nadim, R Scardovelli, and S Zaleski. Volume-of-fluid interface tracking with smoothed surface stress methods for three-dimensional flows. *Journal of Computational physics*, 152(2):423–456, 1999.

[21] D L Youngs. Time-dependent multi-material flow with large fluid distortion. *Numerical methods for fluid dynamics*, 1982.

[22] S Osher and J A Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[23] S Osher and R P Fedkiw. Level set methods: an overview and some recent results. *Journal of Computational physics*, 169(2):463–502, 2001.

[24] M Sussman and E G Puckett. A coupled level set and volume-of-fluid method for computing 3d and axisymmetric incompressible two-phase flows. *Journal of computational physics*, 162(2):301–337, 2000.

[25] B Bunner and G Tryggvason. Dynamics of homogeneous bubbly flows part 1. rise velocity and microstructure of the bubbles. *Journal of Fluid Mechanics*, 466:17–52, 2002.

[26] B Bunner and G Tryggvason. Dynamics of homogeneous bubbly flows part 2. velocity fluctuations. *Journal of Fluid Mechanics*, 466:53–84, 2002.

[27] C Kuan, J Sim, E Hassan, and W Shyy. Parallel eulerian-lagrangian method with adaptive mesh refinement for moving boundary computation. In *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 370, 2013.

[28] M N Farooqi, D Izbassarov, M Muradoğlu, and D Unat. Communication analysis and optimization of 3d front tracking method for multiphase flow simulations. *The International Journal of High Performance Computing Applications*, 33(1):67–80, 2019.

[29] S Popinet and collaborators. Basilisk. http://basilisk.fr, 2013–2020.

[30] C S Peskin. *Flow patterns around heart valves: a digital computer method for solving the equations of motion*. Yeshiva University, 1972.

[31] J Glimm, J W Grove, X Li, and N Zhao. Simple front tracking. *Contemporary mathematics*, 238(2):133–149, 1999.

[32] G Tryggvason, R Scardovelli, and S Zaleski. *Direct numerical simulations of gas–liquid multiphase flows*. Cambridge university press, 2011.

[33] J U Brackbill, D B Kothe, and C Zemach. A continuum method for modeling surface tension. *Journal of computational physics*, 100(2):335–354, 1992.

[34] M J Berger and J Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of computational Physics*, 53(3):484–512, 1984.

[35] A M Roma. *A multilevel self adaptive version of the immersed boundary method*. New York University, 1996.

[36] J Holke. Scalable algorithms for parallel tree-based adaptive mesh refinement with general element types. *arXiv preprint arXiv:1803.04970*, 2018.

[37] M Bader. *Space-filling curves: an introduction with applications in scientific computing*, volume 9. Springer Science & Business Media, 2012.

[38] P J Frey and H Borouchaki. Geometric surface mesh optimization. *Computing and visualization in science*, 1(3):113–121, 1998.

[39] G Tryggvason, B Bunner, A Esmaeeli, D Juric, N Al-Rawahi, W Tauber, J Han, S Nas, and Y Jan. A front-tracking method for the computations of multiphase flow. *Journal of computational physics*, 169(2):708–759, 2001.

[40] T Brochu and R Bridson. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4):2472–2493, 2009.

[41] A Guéziec, G Taubin, F Lazarus, and B Hom. Cutting and stitching: Converting sets of polygons to manifold surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):136–151, 2001.

[42] S Shin and D Juric. Modeling three-dimensional multiphase flow using a level contour reconstruction method for front tracking without connectivity. *Journal of Computational Physics*, 180(2):427–470, 2002.

[43] J Pan, T Long, L Chirco, R Scardovelli, S Popinet, and S Zaleski. An edge-based interface tracking (ebit) method for multiphase-flows simulation with surface tension. *arXiv preprint arXiv:2309.00338*, 2023.

[44] D Torres and J Brackbill. The point-set method: front-tracking without connectivity. *Journal of Computational Physics*, 165(2):620–644, 2000.

[45] D Darmana, N G Deen, and J Kuipers. Parallelization of an euler–lagrange model using mixed domain decomposition and a mirror domain technique: Application to dispersed gas–liquid two-phase flow. *Journal of Computational Physics*, 220(1):216–248, 2006.

[46] B Nkonga and P Charrier. Generalized parcel method for dispersed spray and message passing strategy on unstructured meshes. *Parallel Computing*, 28(3):369–398, 2002.

[47] C Kuan. *Parallel processing of Eulerian-Lagrangian, cell-based adaptive method for moving boundary problems*. PhD thesis, University of Michigan, 2013.

[48] S Schamberger and J Wierum. Graph partitioning in scientific simulations: Multilevel schemes versus space-filling curves. In *International Conference on Parallel Computing Technologies*, pages 165–179. Springer, 2003.

[49] P Frey and P George. *Mesh Generation*. Hermes Science Publishing, 2008.

[50] H Samet. *The design and analysis of spatial data structures*, volume 85. Addison-wesley Reading, MA, 1990.

[51] D Khan, A Plopski, Y Fujimoto, M Kanbara, G Jabeen, Y J Zhang, X Zhang, and H Kato. Surface remeshing: A systematic literature review of methods and research directions. *IEEE transactions on visualization and computer graphics*, 28(3):1680–1713, 2020.

[52] W Yue, Q Guo, J Zhang, and G Wang. 3d triangular mesh optimization in geometry processing for cad. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 23–33, 2007.

[53] W Aniszewski, T Arrufat, M Crialesi-Esposito, S Dabiri, D Fuster, Y Ling, J Lu, L Malan, S Pal, R Scardovelli, et al. Parallel, robust, interface simulator (paris). *Computer Physics Communications*, 263:107849, 2021.

[54] M McGuire. The half-edge data structure. *Website: http://www. flipcode. com/articles/article halfedgepf. shtml*, 2000.

[55] M Hussain, Y Okada, and K Niijima. A fast and memory-efficient method for lod modeling of polygonal models. In *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings*, pages 137–142. IEEE, 2003.

[56] C Gorges, F Evrard, B van Wachem, and F Denner. Reducing volume and shape errors in front tracking by divergence-preserving velocity interpolation and parabolic fit vertex positioning. *Journal of Computational Physics*, 457:111072, 2022.

[57] P Lindstrom and G Turk. Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):98–115, 1999.

[58] G Taubin. Curve and surface smoothing without shrinkage. In *Proceedings of IEEE international conference on computer vision*, pages 852–857. IEEE, 1995.

[59] M O Abu-Al-Saud, S Popinet, and H A Tchelepi. A conservative and well-balanced surface tension model. *Journal of Computational Physics*, 371:896–913, 2018.

[60] S Popinet. An accurate adaptive solver for surface-tension-driven interfacial flows. *Journal of Computational Physics*, 228(16):5838–5866, 2009.

[61] B Lafaurie, C Nardone, R Scardovelli, S Zaleski, and G Zanetti. Modelling merging and fragmentation in multiphase flows with surfer. *Journal of computational physics*, 113(1):134–147, 1994.

[62] S Popinet and S Zaleski. A front-tracking algorithm for accurate representation of surface tension. *International Journal for Numerical Methods in Fluids*, 30(6):775–793, 1999.

[63] Y Renardy and M Renardy. Prost: a parabolic reconstruction of surface tension for the volume-of-fluid method. *Journal of computational physics*, 183(2):400–421, 2002.

[64] D Fuster and S Popinet. An all-mach method for the simulation of bubble dynamics problems in the presence of surface tension. *Journal of Computational Physics*, 374:752–768, 2018.

[65] D KOFFI BI, S Zaleski, B Kottilingal, G Tryggvason, S Costanzo, R Scardovelli, Y Ling, and J Lu. Exact computation of the color function from piecewise linear interfaces. *Bulletin of the American Physical Society*, 67, 2022.

[66] D A Field. Laplacian smoothing and delaunay triangulations. *Communications in applied numerical methods*, 4(6):709–712, 1988.

[67] H Lamb. *Hydrodynamics*. University Press, 1924.

[68] A Prosperetti. Motion of two superposed viscous fluids. *The Physics of Fluids*, 24(7):1217–1223, 1981.

[69] D Gerlach, G Tomar, G Biswas, and F Durst. Comparison of volume-of-fluid methods for surface tension-dominant two-phase flows. *International Journal of Heat and Mass Transfer*, 49(3-4):740–754, 2006.

[70] M Herrmann. A balanced force refined level set grid method for two-phase flows on unstructured flow solver grids. *Journal of computational physics*, 227(4):2674–2706, 2008.

[71] L Cortelezzi and A Prosperetti. Small-amplitude waves on the surface of a layer of a viscous liquid. *Quarterly of Applied Mathematics*, 38(4):375–389, 1981.

[72] D E Fyfe, E S Oran, and M Fritts. Surface tension and viscosity with lagrangian hydrodynamics on a triangular mesh. *Journal of Computational Physics*, 76(2):349–384, 1988.

[73] M Rudman. A volume-tracking method for incompressible multifluid flows with large density variations. *International Journal for numerical methods in fluids*, 28(2):357–378, 1998.

[74] T Arrufat, M Crialesi-Esposito, D Fuster, Y Ling, L Malan, S Pal, R Scardovelli, G Tryggvason, and S Zaleski. A mass-momentum consistent, volume-of-fluid method for incompressible flow on staggered grids. *Computers & Fluids*, 215:104785, 2021.

[75] J B Bell, P Colella, and H M Glaz. A second-order projection method for the incompressible navier-stokes equations. *Journal of computational physics*, 85(2):257–283, 1989.

[76] W C Elmore, W C Elmore, and M A Heald. *Physics of waves*. Courier Corporation, 1985.

[77] M S Plesset. The dynamics of cavitation bubbles. 1949.

[78] J B Keller and M Miksis. Bubble oscillations of large amplitude. *The Journal of the Acoustical Society of America*, 68(2):628–633, 1980.

[79] C S Peskin and B F Printz. Improved volume conservation in the computation of flows with immersed elastic boundaries. *Journal of computational physics*, 105(1):33–46, 1993.

[80] R McDermott and S B Pope. The parabolic edge reconstruction method (perm) for lagrangian particle advection. *Journal of Computational Physics*, 227(11):5447–5491, 2008.

[81] M Tavares, D Koffi-Bi, E Chénier, and S Vincent. A two-dimensional second order conservative front-tracking method with an original marker advection approach based on jump relations. *Comput. Phys*, 27(5):1550–1589, 2020.

[82] D KOFFI BI, S Zaleski, B Kottilingal, G Tryggvason, S Costanzo, R Scardovelli, Y Ling, and J Lu. Exact computation of the color function from piecewise linear interfaces. *[Unpublished Work]*.