Pontifícia Universidade Católica
do Rio de Janeiro

Bruno de Barros Mendes Kassar

Numerical simulation of multiphase flows with enhanced curvature computation by point-cloud sampling

TESE DE DOUTORADO

Thesis presented to the Postgraduate Program in Engenharia Mecânica at PUC–Rio in partial fulfillment of the requirements for the degree of Doctor in Engenharia Mecânica.

Advisor : Profa. Angela Ourivio Nieckele
Co–Advisor: Dr. João Neuenschwander Escosteguy Carneiro

Rio de Janeiro
July 2016

Bruno de Barros Mendes Kassar

Numerical simulation of multiphase flows with enhanced curvature computation by point-cloud sampling

Thesis presented to the Postgraduate Program in Mechanical Engineering of the Departamento de Engenharia Mecânica, Centro Técnico Científico da PUC-Rio as partial fulfillment of the requirements for the degree of Doutor.

Rio de Janeiro, July 28, 2016

Bruno de Barros Mendes Kassar

Engenheiro Mecânico e Mestre em Engenharia Mecânica pela PUC–Rio

Ficha Catalográfica

Kassar, Bruno de Barros Mendes

Numerical simulation of multiphase flows with enhanced curvature computation by point-cloud sampling / Bruno de Barros Mendes Kassar; advisor: Angela Ourivio Nieckele; co–advisor: João Neuenschwander Escosteguy Carneiro. — 2016.

152 f.: il. color. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Mecânica, 2016.

Inclui bibliografia

1. Engenharia Mecânica – Teses. 2. Escoamento multifásico. 3. Método dos Volumes Finitos. 4. VOF. 5. Curvatura. I. Nieckele, Angela Ourivio. II. Carneiro, João Neuenschwander Escosteguy. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Mecânica. IV. Título.

CDD: 621

To my grandfather, Dr. José Nobre Mendes,
in memoriam

Acknowledgments

First of all, I would like to thank my advisors Professor Angela Nieckele and Dr. João Carneiro for all the support, all the time spent with me over the codes, discussing results and planning each step of this work. I would like to thank them for the careful revision of this thesis manuscript and Dr. Carneiro for suggesting the name of the method: PC-VOF. Also, I wish to thank them for the friendship and support during the *International Conference on Multiphase Flow* (ICMF 2016), in Florence.

I would like to thank CAPES and CNPQ for financial support.

Professor Luiz Eduardo Bittencourt for helping me during my early introduction to OpenFOAM® and for the countless times discussing over `interFoam` code. Professor Igor de Paula for the discussions, in the early beginning of this work, about flow instability, for running the experiment of stratified flow transition specially for our discussions and for accepting to be member of the jury. Professor Marcio Carvalho, who advised me during the MSc., for the advices since I started pursuing the doctorate; for his suggestions and discussions during the thesis proposal and for accepting to be member of the jury. Professors Ivan Menezes and Anderson Pereira for the friendship and discussions on the optimization problems. Professors Waldemar Celes and Hélio Lopes for many ideas shared regarding curvature computation and field initialization. Dr. Luis Cristóvão Gomes Coelho for the support during the whole process of this work. I would like to thank the support team and specially Walter Habitzreuter for helping me on the setup of Linux environments in the remote machines that I used to run the simulations, at the Tecgraf Institute. I would like to thank Professor Francisco Ricardo for showing interest in this work and for the discussions on curvature computation during Cobem 2015.

I would like to thank the jury for accepting the invitation, specially the ones who are not from Rio, Professor Aristeu and Dr. Fuster.

I would like to acknowledge the influence of Fernandes Antonio, my first Physics teacher. My interest in Mechanics and Physics started in his remarkable classes, more than fifteen years ago at Colégio Santo Agostinho. Also, Dr. José Nazar for his friendship and support since my early decision to pursue the doctorate!

I wish also to acknowledge and thank Professor Marcelo Gattass for the support, opportunity and influence in my career!

I wish to thank my friends and colleagues, who were kind and patient with me during these years of doctorate. And finally, I would like to thank my sister, Fernanda, for her friendship and my parents, Maria Lucia and Raphael, for everything!

Abstract

Kassar, Bruno de Barros Mendes; Nieckele, Angela Ourivio (Advisor); Carneiro, João Neuenschwander Escosteguy (Co-Advisor). **Numerical simulation of multiphase flows with enhanced curvature computation by point-cloud sampling** . Rio de Janeiro, 2016. 152p. Doctoral Thesis — Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro.

Volume of Fluid (VOF) is a widely employed method for multiphase flows prediction for its simplicity, good mass conservation characteristics and natural handling of topologically complex interfaces. For surface tension dominated flows, however, literature has shown that accuracy in surface tension estimations is still an issue, what may cause parasitic currents and inaccurate prediction of pressure jump condition across interfaces. It occurs mainly due to abrupt changes in the volume fraction field across the interfaces, which takes to inaccurate estimates of interfacial curvatures. Therefore, different approaches have been proposed to mitigate this problem including height-functions, volume fraction smoothing, parabolic fittings, among others. This work proposes a novel approach for curvature estimation in VOF, but not limited to it, that sheds a new light on this persistent problem. The idea is to sample the interfaces with clouds of points and normals at the 0.5 level isosurface of the volume fraction field and to compute the curvature for each point of the cloud by a Computer Graphics technique (normal fitting). The curvatures are then projected onto the Eulerian grid in a Front-Tracking fashion. The new method was implemented in the standard OpenFOAM VOF solver (interFoam) resulting in improvements on the pressure jump estimations and in significant reduction of spurious currents. Numerical simulations were performed and results compared to benchmark data showing the feasibility of the idea.

Keywords

Multiphase flow; Finite Volume Method; Volume of Fluid; Curvature.

Resumo

Kassar, Bruno de Barros Mendes; Nieckele, Angela Ourivio; Carneiro, João Neuenschwander Escosteguy. **Simulação numérica de escoamentos multifásicos com aprimoramento no cálculo da curvatura pela amostragem por nuvem de pontos** . Rio de Janeiro, 2016. 152p. Tese de Doutorado — Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro.

Volume of Fluid (VOF) é um método amplamente empregado na predição de escoamentos multifásicos devido à sua simplicidade, boas características de conservação de massa e natural tratamento de interfaces topologicamente complexas. No entanto, para escoamentos dominados por tensão interfacial, a literatura tem mostrado que a precisão nas estimativas da tensão interfacial ainda é um problema em questão, que pode levar a correntes parasíticas e previsão imprecisa da condição de salto de pressão através das interfaces. Isto ocorre principalmente devido às variações abruptas do campo de fração volumétrica através das interfaces, que leva a imprecisão no cálculo das curvaturas interfaciais. Portanto, diferentes abordagens têm sido apresentadas para mitigar este problema, incluindo funções-altura, suavização da fração volumétrica, ajuste parabólico, entre outros. Este trabalho propõe uma nova abordagem para estimativa de curvatura em VOF, mas não limitado a este, que lança uma nova luz a este problema persistente. A ideia é amostrar a interface por nuvens de pontos e normais na isosuperfície de nível 0.5 do campo de fração volumétrica e calcular a curvatura para cada ponto da nuvem por uma técnica de Computação Gráfica (ajuste de normais). As curvaturas são, então, projetadas na malha Euleriana de maneira tal como no método Front-Tracking. O novo método foi implementado no código padrão de VOF do OpenFOAM (interFoam) resultando em melhorias nas estimativas de salto de pressão e em significativa redução das correntes espúrias. Simulações numéricas foram realizadas e resultados comparados a dados de referência demonstrando a viabilidade da ideia.

Palavras–chave

Escoamento multifásico; Método dos Volumes Finitos; VOF; Curvatura.

Contents

List of Figures

List of Tables

Nomenclature

Ca	capillary number
Eo	Eötvös number
La	Laplace number
Re	Reynolds number
We	Weber number
ΔS_j	boundary faces of cell j
Δt	time step
$\forall$	volume
$\forall_j$	volume of cell j
Γ	general diffusion coefficient
γ	Level Set signed distance function
κ	interface curvature
$\boldsymbol{\tau}$	viscous stress tensor
$\boldsymbol{I}$	identity matrix
μ	dynamic viscosity
Ω	domain region
Ω_i	region occupied by fluid component i
ϕ	general dependent variable
ρ	specific mass
σ	surface tension coefficient
$\overrightarrow{V}$	velocity vector
$\overrightarrow{x}$	position vector
$\overrightarrow{\nabla}$	gradient operator
$\overrightarrow{\nabla}\cdot$	divergent operator
$\overrightarrow{f_\sigma}$	surface tension force
$\overrightarrow{g}$	gravity acceleration
$\overrightarrow{S_f}$	area vector of face f
C_α	interface compression coefficient

d	diameter
D_k	3D bubble diameter (length) in k direction
E	bubble aspect ratio
g	gravity acceleration magnitude
h	mesh spacing
p	pressure
$p_{\rho g h}$	modified pressure
Q	least-squares plane objective function
S_C	constant term of the source term
S_f	area of face f
S_P	linear coefficient of the source term
t	time variable
t_f	final simulation time
V_t	bubble terminal rising velocity

"Ho visto un angelo nel marmo ed ho scolpito fino a liberarlo."
(I saw an angel in the marble and carved until I set him free.)

Michelangelo Buonarroti.

1
Introduction

Multiphase flows are flows in which different fluids coexist interacting with each other. It can be constituted by different, miscible or not, fluids or the same fluid in different phases – gas, liquid or solid. It is characterized by the existence of interfaces between fluids/phases, with discontinuities of flow properties across these interfaces. For liquid-liquid flows, the term *phase* will still be used even though not meaning phase of matter, but fluid component.

Multiphase flows are often found in nature and industry. Ocean waves, rain, clouds, sandstorms are some examples of multiphase flows in nature, just to name a few. In industry, a wide variety of examples can be found, such as fluidized beds, refrigeration, pipeline transport of oil and gas, flow of slurries or pulverized particles, pressurized water nuclear reactors; boiling water and so on (Ishii & Hibiki, 2011). Figures[1] 1.1 and 1.2 display examples of multiphase flows in nature and industry, respectively. Figure 1.1(a) shows dispersed air bubbles in water waves. Figure 1.1(b) presents drops of water bouncing on the free surface due to surface tension. At the same time it causes ripples on the free surface. In Fig.1.2, drops of oil are immersed in the continuous water phase, what is found frequently in oil and gas industry.

The main difficulty in the prediction of multiphase flows is to keep track of the fluid phases along the domain. Depending on several parameters such as geometry, fluid properties and flow regime, the phases can be arranged in different configurations.

1.1(a): Air bubbles dispersed in water waves.

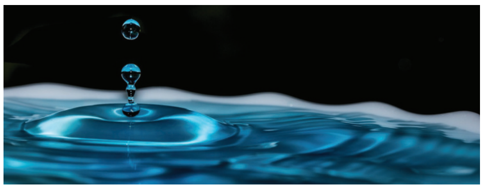

1.1(b): Bouncing droplet.

Figure 1.1: Flow examples in nature.

[1]Images under Creative Commons CC0 license. Downloaded from pixabay.com.

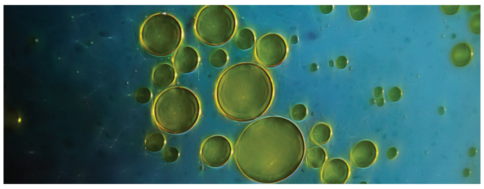


Figure 1.2: Oil drops immersed in water.

A common characterization of multiphase flows in the literature is by its phases geometric arrangement and can be generally divided in *dispersed flows* and *separated flows*. Dispersed flows consist of finite particles, drops or bubbles spread in a continuous phase. The dispersed entities often have length scales which are much smaller than the characteristic length scales of the flow. On the other hand, separated flows are identified by different continuous phases separated by large scale interfaces (Brennen, 2005). Combination of these two configurations often exists.

In pipe flows, the phases geometric arrangement are usually designated *flow patterns* (or *flow regimes*). Figure 1.3 illustrates some of the flow patterns found in horizontal and vertical pipes, including stratified, slug, churn, annular flows, among others.

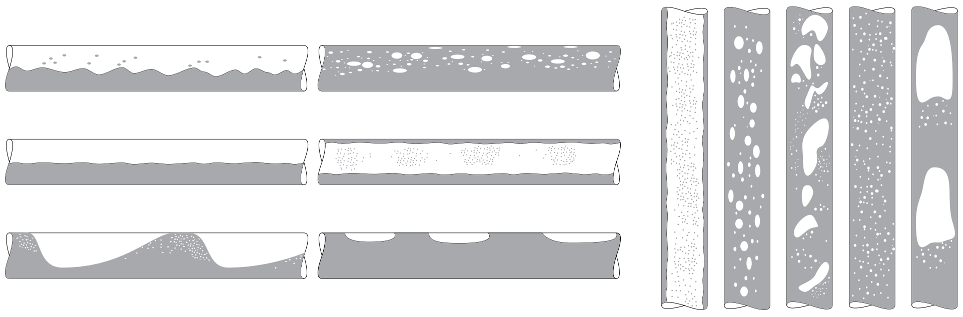


Figure 1.3: Flow patterns in vertical and horizontal pipes.

Exchange rates of mass, momentum and energy are affected by the geometric distribution of the flow components and interfacial shapes (Brennen, 2005). These exchanges are influenced by the area available for transfer between the phases, which clearly depends on the interface shape. Interaction between phases is another factor, as well as the interface evolution. Each phase behavior is dependent on the other phase, characterizing a two-way coupling.

Since the interaction between fluid components strongly depends on the geometric characteristics of these interfaces, for each pattern different forces may be more relevant than others. In this way, with respect to the geometric distribution, different formulations may be better suited than others.

Section 1.1 will present some of the most discussed methods available in the literature to predict multiphase flows. Algorithms developed to calculate two-phase flows must be capable of solving and keeping track of the interface position as accurately as possible, since this position is part of the solution.

1.1 Two Phase Flow Simulation

Acoording to Prosperetti & Tryggvason (2007), there are two main classes of models to simulate multiphase flows: *Multi-Fluid Model* and *One-Fluid Model*. In the first class, the conservation equations for each phase are obtained through a phase average procedure as described by Ishii & Hibiki (2011). Therefore, a complete set of conservation equations – of mass, momentum and energy – are solved, one set for each phase. The phases are coupled with each other by interfacial interaction terms – representing the exchange of mass, momentum and heat at the interfaces – and are usually defined with the aid of empirical or semi-empirical correlations. This type of approach is well suited for dispersed flows, although it is not limited to it. For each fluid component, the quantities are determined in an average way, thus there is no need for high level of grid refinement in order to resolve the interfaces of the smallest bubbles or drops. On the other hand, *One-Fluid Model* solves only one set of conservation equations for the whole domain. It is usually done by keeping track of the interfaces between the fluid components, as depicted in Fig. 1.4. In general, there is no need for empirical correlations to determine the interface position. For each phase region, the appropriate values of fluid properties are used. Coupling between phases is established by the proper consideration of surface tension forces. This approach is better suited for separate flows, since prediction of dispersed flows may require extremely fine grids, what may become computationally unfeasible.

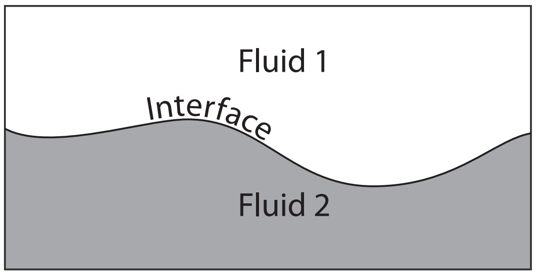


Figure 1.4: Illustration of two immiscible fluids.

This work focuses on the *One-Fluid Model* approach, and it is aimed to enhance the accuracy in curvature predictions. In the *One-Fluid* class, many methods are found in literature to track interfaces. They can represent the interface explicitly or implicitly. A few examples are shown next.

1.1.1 Front-Tracking Method

The *Front-Tracking Method* (Unverdi & Tryggvason, 1992) employs a volumetric fixed grid to solve the flow field and a moving front mesh to keep track of the interface. The interface is explicitly represented by this additional moving mesh. The moving front is depicted in Fig.1.5, where points are connected by straight lines, also referred as surface elements. In 3D, the elements are polygons, usually triangles connecting the front points. The points are advected in a Lagrangian mode following the local velocity field, giving rise to new shapes for the interface at each time step. The surface tension is computed on the front mesh and, then projected onto the fixed grid. To couple both meshes –fixed grid and moving front – information from one is transferred to the other by interpolation procedures that guarantee conservation of the transferred quantities.

The description of the front, especially in three-dimensional flows, needs very careful consideration. From a computational point of view, the *front* takes into account the marker points, their connectivity and the physics related to the presence of an interface, i.e. the interfacial jump conditions (Tryggvason *et al.*, 2011). One advantage of this approach is that the interface always remains sharp. However, the grid may stretch or shrink, resulting in the need for dynamic addition and deletion of points (with proper connectivity updates) of the interface mesh. To keep track of the front objects when the interface propagates and deforms is a challenge, and depends strongly on the data structure used to describe them. Great difficulties arise when multiple interfaces interact with each other causing merge or breakup. In these cases, special treatment of the front mesh must be performed. Main difficulties in this case consist of remeshing the fronts and of knowing the exact instant of time to break or merge the fronts.

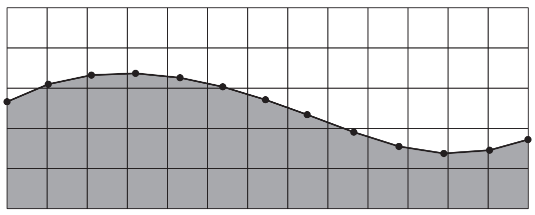


Figure 1.5: Representation of the interface in the Front-Tracking method.

1.1.2 Marker and Cell Method

The first methodology devoted to handle the interface implicitly was introduced by Harlow & Welch (1965), and it is known as *Marker and Cell* method (MAC). Marker particles are initially distributed through all cells containing fluid. Then, by employing a Lagrangian approach, the particles are displaced along the flow domain based on the underlying velocity field computed at the Eulerian grid. The interface is reconstructed in cells that fill two requirements: contain particles and are adjacent to empty cells. This method requires to dynamically remove and add fresh particles when stretching happens, a similar issue presented in Front-Tracking. In principle, coalescence and breakup are not a constrain to use this method, as the interface is not explicitly advected. Figure 1.6 displays the particles spread throughout the fluid domain. The interface is implicitly lying at the boundary between regions with and without particles and is represented as a dotted line. The MAC method was originally devised for free-surface flows, where the *massless* particles did not participate in the calculations explicitly, being used mainly for identifying the different regions in the flow, with boundary conditions applied at the free surface. Only one fluid was considered, with the rest of the domain (above the free-surface) considered fully passive. Daly (1967) extended the original method to handle both fluids, whereas the particles "carry" different properties depending on the fluid properties they initially represent. Hence, the marker particles in this case are also used to compute densities and viscosities in each cell. Some extensions of the method were proposed later (Hirt & Shannon, 1968; Harlow *et al.*, 1976).

The main problem of the MAC method (and its extensions) is that, in addition to solving the equations governing the fluid flow, a large number of particles needs to be tracked, which is computationally expensive.

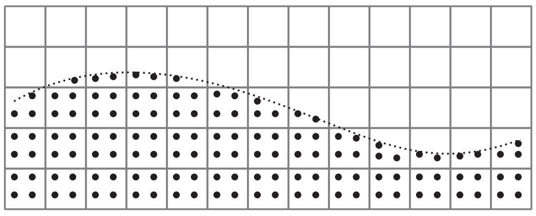


Figure 1.6: Representation of the fluid domain in Marker and Cell method.

1.1.3 Advection of Marker Functions: Volume of Fluid and Level Set Methods

The high cost associated with tracing the particles of the MAC method led to the development of methods that advect a marker (or color) function in an

Eulerian grid in substitution of tracking the marker particles. Two families of this type of method are *Volume of Fluid* (VOF) and *Level Set* (LS) methods.

The *Volume of Fluid* (VOF) method handles the interface implicitly, by tracking the interface indirectly using a scalar marker α that indicates the volume fraction of the reference phase in each cell. This marker α is equal to unity in regions fully occupied by the reference phase and zero in regions empty of this phase. Cells where α value lies between 0 and 1, correspond to cells crossed by the interface, as depicted in Fig. 1.7.

Across the interface, a pressure jump due to surface tension must be imposed. This effect is accounted in the flow formulation by adding a force due to the surface tension as a source term in the momentum equation at the interface cells, as will be discussed in Chapter 3. In VOF, surface tension is most commonly taken into account by using the CSF – *Continuous Surface Force* – method proposed by Brackbill *et al.* (1992). In CSF method, the interface curvature is computed by taking the divergence of a normal field in the vicinities of the interface. This normal field is based on the gradient of the marker function α and represents the interface normal. For flows in which surface tension plays an important role, the accuracy on its prediction strongly influences the flow evolution, and for this reason several works found in the literature have focused on this issue, as discussed in Chapter 2.

At each time step, the α field is advected with the flow in an Eulerian way. After the advection of α is performed, the interface position inside the cell can be determined, what is referred to as *interface reconstruction*. Many methods of surface reconstruction have been developed over the years and Prosperetti & Tryggvason (2007) present a deep discussion on it. There are also several schemes to advect the α field.

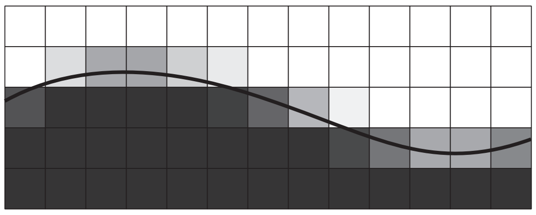
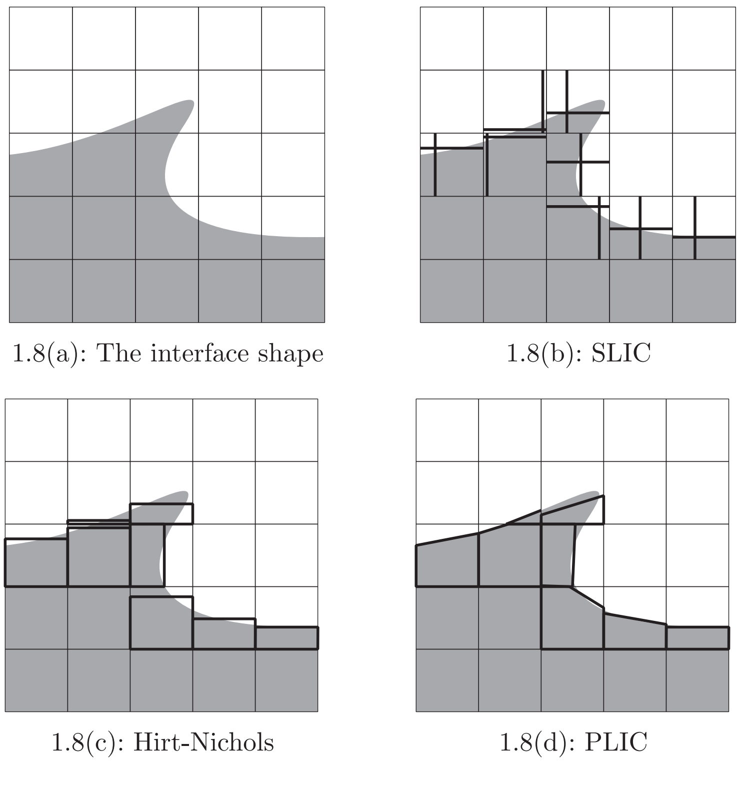


Figure 1.7: Representation of the volume fraction α field in Volume of Fluid method.

Figure 1.8 illustrates the first early types of surface reconstruction methods that were created. The actual interface is shown in Fig. 1.8(a). One of the first methods developed for a 2D situation, named SLIC – *Simple Line Interface Calculation* – method, illustrated in Fig. 1.8(b), was introduced by Noh & Woodward (1976). It consists in splitting the interface in two straight lines. One is parallel to the x-axis for the y-axis advection and the other parallel to y-axis for x-axis advection.

Hirt & Nichols (1981) tried to improve the original SLIC method using only one of the lines (either aligned with x or y axis depending on the case) to approximate the interface for advection in both x and y directions. Fig. 1.8(c) depicts their approach. The choice for the line orientation was defined by a rough estimate of the interface normal vector, taking into account not only the value of α in the concerned cell, but also in the neighboring cells.

The natural extension of Hirt & Nichols scheme is the PLIC – *Piecewise Linear Interface Calculation* – method, by Rider & Kothe (1998), in which the interface approximation was not limited only to axis aligned lines (or planes in 3D). The lines are perpendicular to the estimated interface normals so that they are a better approximation to the interface than Hirt & Nichols scheme, as depicted in Fig. 1.8(d). Although it is a better approximation, the line segments are not connected across cell boundaries, as the reader may observe in the picture. These discontinuities are well observed in regions with high normal variation, i.e. high curvature.


1.8(a): The interface shape

1.8(b): SLIC

1.8(c): Hirt-Nichols

1.8(d): PLIC

Figure 1.8: Illustration of different method of surface reconstruction in VOF method.

The transport equation of the volume fraction function is a pure convective equation, further it is a step function, therefore the false diffusion problem becomes critical, since it is very difficult to predict sharp interfaces. Numerical diffusion occurs specially in lower order schemes, such as *Upwind* schemes. So higher order schemes would seem reasonable to be employed, however they often are unstable. Therefore, in order to achieve a stable method that can be able to produce sharp interfaces with monotonic decay of the volume fraction, many techniques have been proposed. In Gopala & van Wachem (2008), a comparison between several volume fraction advection methods is presented: Flux-corrected transport (FCT) (Boris & Book, 1973), Lagrangian-PLIC (van Wachem & Schouten, 2002), CICSAM – *Compressive Interface Capturing Scheme for Arbitrary Meshes* – (Ubbink, 1997) and Inter-gamma compressive schemes (Jasak & Weller, 1995). Gopala & van Wachem (2008) conclude that the FCT method, besides being restricted to structured meshes, is non-mass conservative for practical flows cases. The Lagrangian-PLIC scheme is fairly accurate, for the tests performed, can be employed for relatively large time steps (as long as $\mathbf{Co} < 1$), however it is also restricted to structured meshes. CICSAM and inter-gamma schemes, which are compressive VOF schemes, have shown to be accurate (in mass conservation and in keeping the interface sharp) for all performed tests. Both schemes are not limited to structured meshes, however they require low Courant numbers, i.e. $\mathbf{Co} < 0.01$ in order to maintain interface sharpness. Much work has also been devoted to compress the interface and improve its prediction through high order or blended schemes (Ghobadian (1991), Pericleous & Chan (1994), Darwish (1993) and Ubbink (1997)).

A very good characteristic of the VOF method is that is mass conservative, while its drawback is the abrupt changes of the α field across interfaces. Although slightly abrupt changes are desirable for the sake of interface sharpness maintenance, it deteriorates the accuracy of normal estimates and, consequently, curvature estimates. Inaccurate curvature computations are known to generate *spurious or parasitic currents* and often generate unphysical pressure fields, with the presence of ripples in the interface region (Francois *et al.*, 2006; Klostermann *et al.*, 2013).

The *Level Set* (LS) method, introduced by Osher & Sethian (1988), also tracks the interface implicitly. However, unlike previous methods, the interface is identified by a smooth *distance function* γ from the interface, as illustrated in Fig. 1.9. The interface lies in the zeroth level and its evolution is performed by advecting the γ field at each time step.

Instead of working with the distance values itself – always greater or equal to zero – one might use negative values on one side of the interface and positive on the other. So, in order to identify the fluid component, one just needs to check the sign of the γ field.

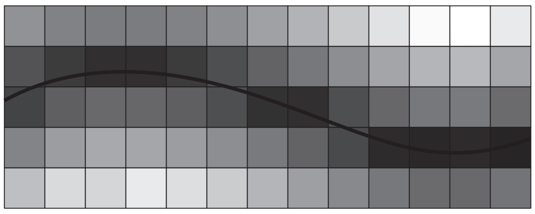
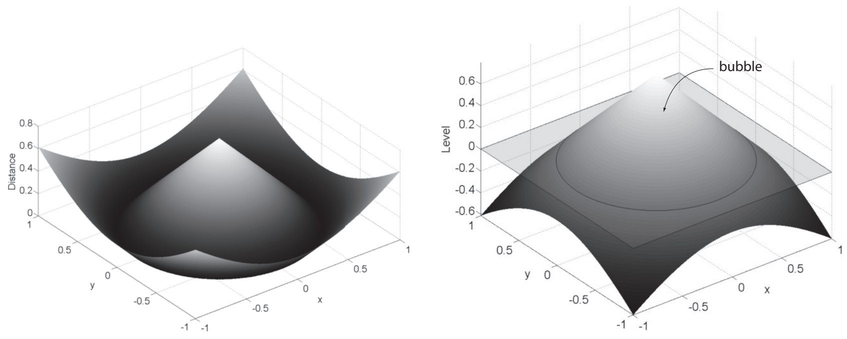


Figure 1.9: Representation of the interface as a distance field.

Figure 1.10 shows the distance function and the signed distance function for a 2D circular interface delimiting a bubble/drop from the continuous phase. The circle is centered at $(0, 0)$ with radius equal to 0.8. The horizontal plane in Fig.1.10(b) lies in level $\gamma = 0$. The portion of the domain with values above the zeroth plane belongs to the bubble/drop, while the portion bellow belongs to the continuous phase. Portions on the zeroth plane contain the interface.



1.10(a): Distance Function 1.10(b): Signed Distance Function

Figure 1.10: Illustration of Level Set field for a circular interface. (a) the distance function.(b) the signed distance function.

The advection equation for the γ function is very similar to the advection equation of the volume fraction function α of the VOF model. The main advantage of the method in relation to VOF method is that γ is a smooth variable and not a step variable like α, therefore, it does not suffer from accuracy in curvature estimates as much as VOF. The advection of γ accurately preserves the zero level-set, which identifies the interface position. However, off the interface, the level-set field does not remain a distance function once it passes through the advection procedure. Besides, the magnitude of $\nabla\gamma$ can become too large or too small around the interface, leading to inaccurate computation of variables that depend on γ, i.e. normals and curvatures. It also may suffer from loss of mass in flows with significant vorticity or with high deformation of the interface. In order to mitigate these problems, the distance function must be reinitialized in every time step after the advection process. A number of extensions to the original method have been proposed. For example, Sussman *et al.* (1994) have proposed an iterative approach to reinitialize the distance function γ by employing an artificial time integration. Sussman & Fatemi (1999) introduced ways to enhance mass conservation, while Sussman & Puckett (2000) developed a hybrid VOF-LS method in order to take advantage of the benefits of both methods.

Despite the presented issues with VOF, one of its greatest advantages is that it presents good mass conservation properties, which is intrinsic to its formulation. Furthermore, many of the VOF methods are simpler than LS and hybrid VOF-LS in their numerical implementation. For both reasons, the VOF method is one of the most widely used interface tracking methods. However, as pointed out by several authors (Williams, 2000; Sussman, 2003; Cummins *et al.*, 2005; Francois *et al.*, 2006) a key issue in VOF methods is the accurate computation of curvature, which will be the main focus of the present work.

1.2 Objectives

This work proposes a novel approach in the VOF framework to deal with curvature computation, that increases accuracy in simulations, reduces the so-called *spurious currents* and alleviates ripples in the pressure field across interfaces. The proposed method was inspired by techniques used in *Computer Graphics* (CG). The idea of the new methodology is to sample the interface by clouds of points interpolated from the VOF field and to geometrically compute the curvatures based on these points. The curvatures (more precisely, the surface tension forces) are computed at each point and are projected onto the fixed grid in a Front-Tracking manner. Further details are presented in Chapter 5.

The new approach was implemented in the standard OpenFOAM® VOF solver, `interFoam` (Weller, 2008). In order to correctly initialize the fields for the simulations in OpenFOAM® framework, a new tool for field initialization was also developed. In order to demonstrate the accuracy of the proposed methodology, several benchmark problems are solved and compared against reference data.

1.3 Thesis Organization

A literature review is presented in Chapter 2, where many efforts in accuracy improvements in VOF are presented. Mathematical formulation of multiphase flows with VOF are presented in Chapter 3. Numerical aspects are discussed in Chapter 4, where a method for accurate initialization of the fields is presented in Section 4.3. The proposed methodology for curvature computation is described in Chapter 5. Chapters 6 and 7 present and discuss results with the proposed methodology, as well as compare them to benchmark data and to `interFoam` results. Finally, the main conclusions and recommendations for future work are present in Chapter 8.

2
Literature Review

Multiphase flows present, in general, a level of complexity in its numerical description greater than single phase flows, specially because, besides velocity and pressure fields, the interface dynamics is also part of the solution. The main issue is to keep track of the phases. The choice of a numerical method to be employed depends strongly on the flow pattern that is to be simulated. For separated flows, or disperse flows with the disperse phase well described by the mesh resolution, a One Fluid approach (Prosperetti & Tryggvason, 2007) is a reasonable choice as it does not require empirical correlations and the interfaces can be tracked explicitly.

Topologically complex interfaces are usually found in problems of breaking waves, coalescence and breakup of drops and bubbles. These complex interfaces can be determined by explicit and implicit methods. Complex interface determination is a major challenge for methods which explicitly represent the (highly deformed) interfaces, since the surface is described by a moving mesh that is advected along with the flow changing not only its shape, but also its topology. This is the case of Front-Tracking type methods (Tryggvason *et al.*, 2001), which require re-meshing in cases of breakup and coalescence of bubbles or drops. Further, these methods must be able to identify the correct time step to perform fronts merging or splitting. Besides these complications, insertion and deletion of surface elements must be foreseen, in order to keep interfaces well represented.

In methods that represent interfaces implicitly, such as MAC, VOF and Level-Set, the topological problem is handled naturally, once the surface mesh does not evolve, but instead, (if needed) it is reconstructed at every time step. Thus, in problems where topological changes are significant, methods with implicit interface representation are often preferable for their simplicity.

Determination of a two phase flow field coupled with the interface definition depends directly on the surface tension acting at the interface. Accurate computation of surface tension requires correct estimation of surface normals and curvatures (Francois *et al.*, 2006). Typically, normals and curvatures are better represented in explicit methods than in implicit ones. Besides, there are possible simplifications that can be used, as is the case of Front-Tracking method. By using the Stokes theorem (Aris, 1990), the surface tension force may be simplified so that its computation can be performed without the need of explicitly computing the interface curvature. Instead, a cyclic integral along the surface elements edges is performed, as described in Tryggvason *et al.* (2001). In methods that deal with the interface implicitly, however, this last approach is unfeasible once the interface is not represented by surface elements. Therefore, the curvature must be computed from the available information that identifies the interfaces. For instance, in Level Set method the curvature is computed from its signed distance field and in VOF method, it is computed based in the volume fraction field, that, as pointed before in Section 1.1.3, presents accuracy issues regarding curvature computation.

At the present work, the VOF method was selected to solve the flow phase flow field. Thus, the literature review presented in the next sections are focuses on the interface treatment. As stated above the interface determination is essential and critical for an accurate determination of complex interfaces.

2.1 Surface Tension Improvements for VOF method

The surface tension force $\overrightarrow{f_\sigma}$ is a force that acts at the interface and depends on the interface curvature κ, the surface tension coefficient σ and the surface normal vector $\hat{n}$. It acts on the direction of $\hat{n}$. In One-Fluid methods, $\overrightarrow{f_\sigma}$ is modeled by means of Dirac delta (δ) in order to concentrate it at the interface. It is given by (Prosperetti & Tryggvason, 2007)

$$\overrightarrow{f_\sigma} = \sigma\ \kappa\ \delta(n)\ \hat{n} \tag{2.1}$$

where n is a normal variable to the interface, with $n = 0$ at the interface.

The most common approach to compute $\overrightarrow{f_\sigma}$ is the so-called CSF – *Continuum Surface Force* – model, proposed by Brackbill *et al.* (1992). Its original idea is to consider the interface to have finite thickness (on the length scale of the mesh spacing) and to transform the superficial force (related to the pressure jump across the interface) into a continuum volumetric force concentrated at the interface region. It results in replacing the δ function by the gradient of the marker function and by taking the gradient direction as the normal field of the surface. In the case of VOF method, the marker function is the volume fraction α, so $\overrightarrow{f_\sigma^{CSF}}$ is given by (Tryggvason *et al.*, 2011)

$$\overrightarrow{f_\sigma^{CSF}} = \sigma\ \kappa\ \overrightarrow{\nabla}\alpha \tag{2.2}$$

while the interface curvature is given by the divergence of the normal to the interface (Prosperetti & Tryggvason, 2007) and it results in

$$\kappa = \vec{\nabla} \cdot \hat{n} = \vec{\nabla} \cdot \left(\frac{\vec{\nabla}\alpha}{|\vec{\nabla}\alpha|} \right). \tag{2.3}$$

Another common method is referred to as GFM –*Ghost Fluid Model*– (Liu *et al.*, 2000), where the surface force is considered sharp, instead of smeared around the interface, as in CSF. This method is mostly used in Level Set implementations, because it requires the knowledge of the distance field to the interface. Kang *et al.* (2000) presents results with simulations performed with CSF and GFM methods and shows CSF approach is more stable than GFM. As CSF approach smears the interface, it has the effect to (artificially) damp out *spurious currents* with more intensity than GFM.

2.1.1 Smoothing VOF field

As already mentioned in Section 1.1.3, the α field presents abrupt changes across the interface, so that normals and curvatures computed via finite differences of the α field suffer from loss of accuracy in the transition region (Williams *et al.*, 1998). With this in mind, Williams *et al.* (1998) and Williams (2000) proposed to convolve the volume fraction field with smoothing kernels in order to obtain a smooth α field in the interface region, and then compute the interface normal field based on this smoothed volume fraction. From this normal field, the curvature field is evaluated by a discrete divergence operator.

Another approach in the same path of volume fraction smoothing is presented by Lafaurie *et al.* (1994), where a Laplacian filter transforms the VOF field α into a smoother field. The cell-centered volume fraction α field is first projected onto the face centers (by linear interpolation of the two neighboring cells to each face) giving rise to the face-centered α_f field, and then, a smooth cell-centered $\tilde{\alpha}$ is computed by

$$\tilde{\alpha} = \frac{\sum_f \alpha_f S_f}{\sum_f S_f} \tag{2.4}$$

where S_f is the area of face f and the summations are over all faces of each cell. This procedure can be performed many times, until the desired level of smoothing is achieved, i.e. until spurious currents are consistently reduced. The problem with this approach is exactly the choice of the number of times the VOF field should be filtered, because each time the filter is applied, the high curvature regions tend to lose its intensity. Hoang *et al.* (2013) tested this filter in OpenFOAM® –`interFoam` solver (Weller, 2008)– for microchannel flows, and found that reduction of one order of magnitude on spurious currents are obtained by applying the Laplacian filter twice, and that further filtering do not decrease these currents significantly. Samkhaniani & Ansari (2016) recently employed this filter –twice as recommended by Hoang *et al.* (2013)– in `interFoam`, where subcooled boiling flows were studied. In their problem, the spurious currents create artificial extra heat convection in the vicinity of the interface, producing unphysical mass transfer. The application of the Laplacian filter was able to reduce spurious currents by one order of magnitude, however it was not able to completely fade it out.

2.1.2 Parabolic Reconstruction

Renardy & Renardy (2002) addressed the problem of accurate curvature computation with VOF by representing the interface as piecewise quadratic equations fitted on the VOF field. The algorithm is named PROST – *Parabolic Reconstruction of Surface Tension*. It consists on fitting quadratic equations for every cell crossed by the interface, taking the neighboring cells into consideration. It is relevant to notice that the referred work requires orthogonal structured meshes. In 3D domains, the interface is described by piecewise paraboloids for each interface cell and the curvature κ is analytically computed from the paraboloid equations.

2.1.3 Coupling VOF with Level Set

Some researchers took advantage of the fact that the Level-Set field γ is smooth throughout the whole domain and, thus, more adequate for normal and curvature computation than the VOF field. Thus, Sussman & Puckett (2000) coupled the Level-Set method with VOF in what they called CLSVOF –*Coupled Level Set/Volume-of-Fluid* – method where the γ values at each time step are derived from both γ and α values from the previous time step. The curvatures and normals are obtained directly via finite differences on the (smooth) γ field. The normals are computed by $\hat{n} = \vec{\nabla}\gamma/|\vec{\nabla}\gamma|$ and the curvatures by taking the divergence of this normal field, i.e. $\kappa = \vec{\nabla} \cdot \hat{n}$. In this way, CLSVOF benefits from VOF's ability to conserve mass and Level-Set's accuracy on normal and curvature computation.

Albadawi *et al.* (2013) implement a simple coupled VOF with LS method (S-CLSVOF) in OpenFOAM® in order to improve accuracy in surface tension computations. Their work concludes that the standard VOF implementation in

OpenFOAM® –`interFoam`– is not suitable for problems in which surface tension effects play a significant role. Therefore it is extended with LS method. Bubble growth and detachment problems were simulated and while qualitatively both `interFoam` and S-CLSVOF are able to predict the complete process of bubble growth and detachment, `interFoam` fails to predict the correct detachment time. The coupled method, however, predicts the bubble volume and time of detachment with errors less than 3%.

2.1.4 Reconstructed Distance Function

Cummins *et al.* (2005) presented a new method to compute curvature in a VOF framework based on what they called RDF – *Reconstructed Distance Function*. Inspired in the Level Set method, they proposed to define a distance function from the reconstructed interface (reconstructed with PLIC) to be used for curvature computation. This RDF function Φ is computed for all interface cells and for cells around it, up to a normal distance of $3\Delta x$ from the interface, where Δx is the mesh spacing. The value Φ_{ij} of the RDF for each considered cell (i, j) is a weighted average of the Euclidean distances from the center of the cell (i, j) to the neighboring PLIC segments that represent the interface. The referred work compares the accuracy and robustness of curvature estimates based on the proposed RDF against those obtained with two other methods: the already mentioned convolved VOF (CV) function (Williams, 2000) and a technique named *Height Function* (HF) (Sussman, 2003; Helmsen & Colella, 1997). HF is described in section 2.1.5. Results obtained using height function were superior compared to both convolved VOF and RDF. However, when interfacial length scale is not well resolved by the grid, CV and RDF methods exhibit greater robustness over the HF approach.

2.1.5 Height-Function

In order to obtain second-order accurate curvatures, Sussman (2003) computed a curvature based on the volume fraction field, instead of using γ, that would not give formally second-order accuracy. This method is based on reconstructing a *Height Function* (HF) directly from the volume fractions, using a stencil of neighboring cells as described in Helmsen & Colella (1997). In Sussman (2003), the orientation of the interface is determined by a normal computed based on the the level set function as $\hat{n} = \vec{\nabla}\gamma / |\vec{\nabla}\gamma|$.

To better explain the method, it is considered that the interface surface is horizontal rather than vertical (for a vertical interface, an analogous procedure is employed. For each cell (i, j) containing the interface, a 3×7 block of neighboring cells is used to compute the “height” as a result of the summation over the α field along the three vertical columns of 7 cells, for $i - 1$, i and $i + 1$. This gives the “height” value at $(i - 1, j)$, (i, j) and $(i + 1, j)$. With these values, the first and second derivatives of the HF (which is second order accurate) are determined at cell (i, j), giving rise to the curvature. The authors have considered one of the phases a gas with constant pressure, without being explicitly solved for.

Sussman *et al.* (2007) extended Sussman (2003) treating the gas as a second incompressible fluid. In Sussman *et al.* (2007), the level set function is used only to compute interface normals and to compute density and viscosity to be used in the Navier Stokes equations. The curvature is computed in the very same way as in Sussman (2003).

Sussman & Ohta (2009) continued the research on curvature estimation based on a height function (HF). They derived the height function from the LS field instead of the VOF field, as in previous works (Sussman, 2003; Sussman *et al.*, 2007). They found it easier to implement and more accurate. The authors make an important consideration that when one discretizes the curvature for surface tension purposes, that curvature should represent an approximation to the curvature *on the interface*, in the same way the height function approximation does (height function reconstructed from either γ or α). The curvature computed directly from the level set field, i.e., from the formula $\kappa = \overrightarrow{\nabla} \cdot (\overrightarrow{\nabla}\gamma / |\overrightarrow{\nabla}\gamma|)$ gives an estimate at the cell center instead of on the surface itself, what is not desired.

Originally, the HF technique was devised based on orthogonal meshes and no method existed for non structured meshes. This limitation was present, until recently, when Ito *et al.* (2014) extended the HF method for two-dimensional unstructured meshes. Later, Ivey & Moin (2015) extended the HF technique for 3D unstructured meshes. Their approach enables the computation of interface normals and curvatures in three-dimensional unstructured non-convex polyhedral grids. The accuracy obtained with this enhancement is compared to the accuracy in the traditional HF technique in orthogonal meshes.

Francois *et al.* (2006) devised a new balanced-force algorithm for modeling interfacial flows with surface tension. In their work, the momentum equation time discretization is split into a “predictor” and a “corrector” steps. The predictor step solves for the velocities in an intermediate time step, while the corrector step corrects the pressure jump and then corrects the velocity field for the next time step. This procedure recovers an exact balance between pressure jump and surface tension. Since the authors estimate curvature with HF, difficulties arise when the curvature radius becomes too small compared to the grid size. In this limit, the results are inconsistent. However, Popinet (2009), using the Gerris solver (Popinet, 2003), generalized the HF method enabling second-order accuracy even with low mesh resolutions. In the same work, Popinet proposes to enhance mesh resolution by octree subdivision (Samet, 1984), what enables to capture small and large scales efficiently. It is worth to notice that octree subdivision requires Cartesian grids. A common test case of code validation is the problem of a bubble or drop at rest in a stationary fluid. The exact solution is given by Laplace's law, which states that the pressure jump across the surface is balanced by the surface tension and it results in a spherical shape. Until recently, all methods with implicit interface representation resulted in non-equilibrium solutions due to spurious currents. In the worst scenario, these parasitic currents can be strong enough and cannot be neglected. The work of Popinet (2009) claims to be the first to achieve static equilibrium for this problem. Fuster *et al.* (2009) presented many simulations of primary atomization performed with Gerris showing that this kind of problem, where the characteristic length scales are spread over several orders of magnitude, is possible to be solved due to Gerris' capability of adaptive mesh refinement.

Deshpande *et al.* (2012) have made an analysis of the performance of `interFoam` in a number of different test cases. In particular, it has been shown that, for surface tension driven flows, curvatures converged to different values than the analytical solutions suggest. Furthermore, non-negligible spurious currents have also been observed. A time-step constraint has been proposed to work around this problem in the test cases used by the authors.

Klostermann *et al.* (2013) have investigated the problem of a single rising bubble with `interFoam`. The test cases of Hysing *et al.* (2009) were simulated and results were compared to a number of different interface capturing codes. The authors conclude that the parasitic currents, and specially curvature estimation (together with CSF), in `interFoam` are probably interconnected topics that should be further investigated, as well as the surface compression scheme.

Cano-Lozano *et al.* (2015) performed tests comparing both `interFoam` and Gerris for rising bubble flows dominated by surface tension. They concluded that `interFoam` is not suitable for this type of flows as it presents non-negligible spurious currents, while Gerris showed absence of spurious currents. These parasitic currents may be reduced by increasing spatial resolution in `interFoam`, however at the cost of a considerable increment in CPU time. They conclude that Gerris is well suitable for surface tension dominated flows due to its *Adaptive Mesh Refinement* technique and surface tension treatment by both HF and balanced-force algorithm (Francois *et al.*, 2006).

Octree subdivision is one of the strategies of mesh refinement in AMR – *Adaptive Mesh Refinement* – methods, that subdivide the grids in regions where the solution lacks accuracy. These methods define a threshold for the error in the flow solution, and where this limit is reached, the mesh is refined. Another approach for grid refinement is the SAMR – *Structured Adaptive Mesh Refinement* – (Berger & Oliger, 1984; Berger & Colella, 1989). Pivello *et al.* (2014) employs the SAMR algorithm proposed by Berger & Rigoutsos (1991) to refine the Eulerian grid in their Front-Tracking Method where the criteria for refinement is based on the interface position and on the vorticity gradient. In general, mesh refinement technique enables to capture flow phenomena in a wide range of length scales efficiently, since the mesh is refined only where needed.

2.2 Interface Surface Reconstruction and VOF Advection

Concerning surface reconstruction, most of the recent works found in the literature use PLIC – Piecewise Linear Interface Calculation (Rider & Kothe, 1998). Cummins *et al.* (2005), Francois *et al.* (2006), Sussman & Ohta (2009) and Popinet (2009) use PLIC as a technique to reconstruct and advect the interface in structured orthogonal meshes. The PLIC algorithm consists of two steps: (i) an interface reconstruction by planes in each interfacial cell and (ii) a geometrical computation of volume fluxes across the grid faces (Rider & Kothe, 1998; Garrioch & Baliga, 2006). A recent article by Ito *et al.* (2013) proposes a new PLIC algorithm applied to unstructured meshes. It is accurate even for unstructured meshes with highly-irregular cell arrangements. In their work, the authors use meshes of quadrilaterals and triangles.

An alternative approach to VOF without the need for interface reconstruction, as in PLIC, is to solve the transport of volume fraction by FCT – *Flux Corrected Transport.* FCT was introduced by Boris & Book (1973) defining the computation of fluxes in a way that guarantees boundedness in the solution of hyperbolic equations. Zalesak (1979) improved the technique and extended it to more than one dimension without the need for time splitting. It is known that high order schemes for numerical integration of transport equations suffer from oscillations. On the other hand, low order schemes suffer from numerical diffusion. The main idea of FCT is to compute fluxes with both low and high order schemes and weight them in a way that minimizes both oscillations and numerical diffusion (Zalesak, 1979).

In 2004, OpenFOAM® –*Open Field Operation and Manipulation*– was released as an open source C++ suite of libraries and solvers for the solution of continuum mechanics problems (OpenFOAM, 2014), which includes CFD – *Computational Fluid Dynamics*– problems. In 2007, OpenFOAM® version 1.4 was released introducing a new algorithm for two-phase interface tracking based on FCT theory (Zalesak, 1979; Rudman, 1997). This algorithm, called MULES –*Multidimensional Universal Limited Explicit Solver*– was implemented by Henry Weller (Weller, 2008) and is well described in Damián (2013). In the current version of OpenFOAM® (v2.3.0) used in this work, just released on February 2014, MULES was improved. As it is an explicit method, it would come at the cost of a limit in the maximum Courant number ($\mathbf{Co} = \delta t|\overrightarrow{V}|/\delta x$). However, in the current version, a new semi-implicit variant of MULES is introduced. It first executes an implicit predictor step before constructing an explicit correction. This approach maintains boundedness and stability at an arbitrarily large Courant number. The main goal is to transport the phase fraction with VOF method keeping the interface sharp and guaranteeing boundedness of the volume fraction field without the need for the costly, yet more accurate, PLIC surface reconstruction.

So far, PLIC surface reconstruction is not yet available in standard OpenFOAM® releases, but it has already been developed and presented as `voFoam` in Maric *et al.* (2013). The current update of `voFoam` project is that it is still to be released as an open source code.

3
Mathematical Formulation

Isothermal fluid flows are modeled by the balance equations of mass and linear momentum (Panton, 2005). In the present work, the VOF method, which is a type of *One-Fluid Model* (Prosperetti & Tryggvason, 2007) was selected to determine the flow field.

The conservation of mass, in the absence of mass transfer, is given by

$$\frac{\partial \rho}{\partial t} + \overrightarrow{\nabla} \cdot (\rho \overrightarrow{V}) = 0 \tag{3.1}$$

where ρ is the fluid density, t the time variable, $\overrightarrow{V}$ the velocity vector and $\overrightarrow{\nabla}\cdot$ the divergent operator. For incompressible flows, the above reduces to enforce the velocity field to be a divergence-free vector field

$$\overrightarrow{\nabla} \cdot \overrightarrow{V} = 0 \tag{3.2}$$

The linear momentum balance can be written as

$$\frac{\partial (\rho \overrightarrow{V})}{\partial t} + \overrightarrow{\nabla} \cdot (\rho \overrightarrow{V} \overrightarrow{V}) = -\overrightarrow{\nabla} p + \overrightarrow{\nabla} \cdot \boldsymbol{\tau} + \rho \overrightarrow{f} \tag{3.3}$$

where $\rho \overrightarrow{f}$ is a body force acting on the fluid (such as the gravitational force, where $\overrightarrow{f}$ would be the gravitational acceleration). The term $\overrightarrow{\nabla} p$ is the pressure gradient, the pressure force per unit volume, and $\boldsymbol{\tau}$ is the viscous stress tensor.

For *Newtonian* fluids, the viscous stress tensor is given by

$$\boldsymbol{\tau} = \mu \left[\nabla \overrightarrow{V} + \left(\nabla \overrightarrow{V} \right)^T - \frac{2}{3} \left(\nabla \cdot \overrightarrow{V} \right) \boldsymbol{I} \right] \tag{3.4}$$

where μ is the fluid viscosity and $\boldsymbol{I}$ is the identity tensor.

3.1
Two-Phase Formulation

The concept of the *One Fluid Model* is to solve only one set of equations for both phases, considering the fluid with variable properties. The key component of this type of model is to identify the regions occupied by each fluid.

Consider a two-phase system, where hypothetical phases 1 and 2 are separated by an interface, as shown in Fig.3.1 Let the interface S be represented by every point that satisfy a function $F(\overrightarrow{x}, t)$ equal to a constant. For instance, let it be at the zero level of F (Tryggvason *et al.*, 2011).

$$F(\overrightarrow{x}, t) = 0 \tag{3.5}$$

where $\overrightarrow{x}$ is the position vector.

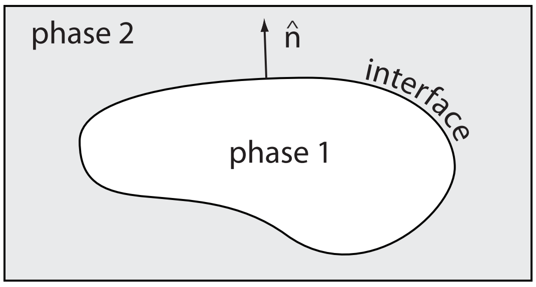


Figure 3.1: Two phases separated by an interface.

Given the interface is at $F = 0$ and that it separates both phases, one of the phases is at $F < 0$ and the other at $F > 0$. It is usual convention to let phase 1 be at $F > 0$ and phase 2 be at $F < 0$. Besides let the normal at the interface be pointing from phase 1 to phase 2. Thus, the unit normal at the interface is given by

$$\hat{n} = -\frac{\overrightarrow{\nabla} F}{|\overrightarrow{\nabla} F|} \tag{3.6}$$

The interface can move basically through two processes: (1) mass transfer between the phases and (2) fluid motion. Notice that only its normal direction $V_S = \overrightarrow{V_S} \cdot \hat{n}$ is necessary to be specified, i.e. any tangential movement does not change its shape. Considering no mass transfer, the interface moves with the fluid velocity. That is to say its material derivative is zero (Tryggvason *et al.*, 2011)

$$\frac{DF}{Dt} = \frac{\partial F}{\partial t} + \overrightarrow{V} \cdot \overrightarrow{\nabla} F = 0 \tag{3.7}$$

Using the interface normal defined in Eq.(3.6), it becomes

$$\frac{\partial F}{\partial t} - \overrightarrow{V} \cdot \hat{n} |\overrightarrow{\nabla} F| = 0 \tag{3.8}$$

$$\frac{\partial F}{\partial t} - V_S |\overrightarrow{\nabla} F| = 0 \tag{3.9}$$

Consider a control volume containing a portion of the interface, as depicted in Fig.3.2. Taking its thickness as zero, no mass can accumulate inside the control volume. Therefore, the inflow mass is equal the outflow. That is to say

$$\rho_1(\overrightarrow{V}_1 \cdot \hat{n} - V_S) = \rho_2(\overrightarrow{V}_2 \cdot \hat{n} - V_S) = \dot{m} \tag{3.10}$$

where $\dot{m}$ is the net mass flow per unit area across the interface. If there is no mass transfer, $\dot{m} = 0$, so

$$V_S = \overrightarrow{V}_1 \cdot \hat{n} = \overrightarrow{V}_2 \cdot \hat{n} \tag{3.11}$$

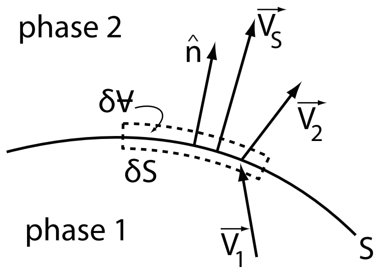


Figure 3.2: Thin control volume containing the interface.

As noted in Tryggvason *et al.* (2011), mass conservation places a restriction on the normal velocity, Eq.(3.11), but none on the tangential velocity. For viscous flows, it is observed experimentally that there is no slip between the phases, so that the tangential components are the same. So, at the interface

$$\overrightarrow{V}_S = \overrightarrow{V}_1 = \overrightarrow{V}_2 \quad \text{at } F = 0 \tag{3.12}$$

In order to determine the motion of the interface , a further condition is necessary (Tryggvason *et al.*, 2011). A momentum balance on the control volume of Fig.3.2 that is moving with velocity $\overrightarrow{V}_S$ results in

$$-\oint_{\delta S} \rho \overrightarrow{V}(\overrightarrow{V} \cdot \hat{n} - V_S) ds + \oint_{\delta S} \hat{n} \cdot \boldsymbol{T} ds + \int_S \overrightarrow{f_\sigma} ds = 0 \tag{3.13}$$

where the tensor $\mathbf{T}$ has a normal contribution of pressure p and the viscous contribution $\boldsymbol{\tau}$, $\mathbf{T} = (-p\, \boldsymbol{I} + \boldsymbol{\tau})$.

The first two integrals are over the edges of the control volume $\delta\forall$ and the last integral is over the interface. For flows with no mass transfer, Eq.(3.11) is valid, therefore the first term drops resulting in

$$\oint_{\delta S} \hat{n} \cdot \boldsymbol{T} ds + \int_S \overrightarrow{f_\sigma} ds = 0 \tag{3.14}$$

Considering, once again, the thickness of the control volume converging to zero, the boundary δS lies on the surface S. Thus, the cyclic integral of the stress results in the jump of the surface traction $\boldsymbol{T}$ across S. That is, the jump in $(-p\boldsymbol{I} + \boldsymbol{\tau}) \cdot \hat{n}$ should be balanced by the surface tension force (Prosperetti & Tryggvason, 2007). It results in

$$[(p_1 - p_2)\boldsymbol{I} + (\boldsymbol{\tau}_2 - \boldsymbol{\tau}_1)] \cdot \hat{n} = -\overrightarrow{\nabla} \cdot [(\boldsymbol{I} - \hat{n}\hat{n})\sigma] = -(\boldsymbol{I} - \hat{n}\hat{n}) \cdot \overrightarrow{\nabla}\sigma + \sigma\kappa\hat{n} \quad (3.15)$$

where the term $(\boldsymbol{I} - \hat{n}\hat{n})$ is the projector on the plane tangent to the interface, σ is the surface tension coefficient and κ is twice the local mean curvature of the surface given by

$$\kappa = \overrightarrow{\nabla} \cdot \hat{n} \quad (3.16)$$

For constant surface tension coefficient, the following arises

$$[(p_1 - p_2)\boldsymbol{I} + (\boldsymbol{\tau}_2 - \boldsymbol{\tau}_1)] \cdot \hat{n} = \sigma\kappa\hat{n} \quad (3.17)$$

To account for the jump in the surface traction, the surface tension is added to the linear momentum equation Eq.(3.3) as an additional source term. This is the way the boundary condition of pressure jump across the interface is treated in One-Fluid method. This effect is concentrated at the interface by means of the delta $\delta(n)$ function. Taking the surface tension effect into account, Eq.(3.3) is written for two-phase flows as

$$\frac{\partial(\rho\overrightarrow{V})}{\partial t} + \overrightarrow{\nabla} \cdot (\rho\overrightarrow{V}\overrightarrow{V}) = -\overrightarrow{\nabla}p + \rho\overrightarrow{f} + \overrightarrow{\nabla} \cdot \boldsymbol{\tau} + \overrightarrow{f_\sigma} \quad (3.18)$$

where $\overrightarrow{f_\sigma}$ is the interface tension force per unit volume, acting at the interface region. (Brackbill *et al.*, 1992)

$$\int_{\forall} \overrightarrow{f_\sigma} dv = \int_S \sigma\kappa(\overrightarrow{x_s})\ \hat{n}(\overrightarrow{x_s})\ ds \quad (3.19)$$

where the integral on the right hand side is over the interface S and the integral on the left hand side is over the volume in the vicinity of the interface (transition region) with a thickness that goes to zero. The parameter σ is the interface tension coefficient (considered constant here), κ is the interface curvature, $\hat{n}$ is the interface normal vector and $\overrightarrow{x_s}$ is a position vector on the interface S. According to Brackbill *et al.* (1992), the integral over the interface can be converted into an integral over the volume by using a delta function that takes zero value off the interface. So, the above can be stated as

$$\int_S \sigma\kappa(\overrightarrow{x_s})\ \hat{n}(\overrightarrow{x_s})\ ds = \int_\forall \sigma\kappa(\overrightarrow{x})\ \hat{n}(\overrightarrow{x})\ \delta\left[\hat{n}(\overrightarrow{x_s}) \cdot (\overrightarrow{x} - \overrightarrow{x_s})\right]\ dv \tag{3.20}$$

where the delta function is evaluated at a normal coordinate from the interface. Here $\overrightarrow{x_s}$ is the closest point from $\overrightarrow{x}$ at the interface. The surface tension force $\overrightarrow{f_\sigma}$ can be stated simply by

$$\overrightarrow{f_\sigma} = \sigma\kappa\ \hat{n}\ \delta(n) \tag{3.21}$$

where n is the normal coordinate from the interface to $\overrightarrow{x}$.

Chapter 4 presents the discretization employed for the momentum equation, Eq.(3.18), in `interFoam` and Chap.5 brings more details concerning curvature.

3.2 VOF Two-Phase Formulation

A marker function to keep track of the fluid phases is defined such that it is equal to one in every point inside the reference phase and zero in the other phase. Without loss of generality, phase 1 will be considered the reference phase in this description. Considering Ω to be the whole domain, Ω_1 the region occupied by phase 1 and Ω_2 the region occupied by phase 2, the marker function can be stated as

$$\tilde{\alpha}(\overrightarrow{x}, t) = \begin{cases} 1 & , \quad \overrightarrow{x} \in \Omega_1 \text{ at time } t \\ 0 & , \quad \overrightarrow{x} \in \Omega_2 \text{ at time } t \end{cases} \tag{3.22}$$

It is clearly a non continuous function, with abrupt change across the interface between the phases. One may notice, however that there is a more convenient way to express this indicator. By integrating $\tilde{\alpha}$ in a volume region $\delta\forall_i$ around point $\overrightarrow{x_i}$, the volume fraction α of the reference phase arises as follows

$$\alpha(\overrightarrow{x_i}, t) = \int_{\delta\forall_i} \tilde{\alpha}(\overrightarrow{x}, t) d\forall \tag{3.23}$$

If the region is fully filled with phase 1 at time t, $\alpha(\overrightarrow{x_i}, t) = 1$. If, on the other hand, is is completely filled of phase 2, $\alpha(\overrightarrow{x_i}, t) = 0$. If both phases are present, $0 < \alpha(\overrightarrow{x_i}, t) < 1$.

The fluid properties such as ρ and μ can represented as the mixture property such as

$$\rho(\overrightarrow{x},t) = \tilde{\alpha}(\overrightarrow{x},t)\ \rho_1 + (1-\tilde{\alpha}(\overrightarrow{x},t))\ \rho_2 \tag{3.24}$$
$$\mu(\overrightarrow{x},t) = \tilde{\alpha}(\overrightarrow{x},t)\ \mu_1 + (1-\tilde{\alpha}(\overrightarrow{x},t))\ \mu_2 \tag{3.25}$$

where subscript 1 refers to *phase 1* and 2 to *phase 2*. Therefore, away from the interface, where the marker function takes up values of 0 or 1, the physical properties associated to each individual phase are recovered. Analogously, in an infinitesimal region around point $\overrightarrow{x}$, the mixture propeties are given as a function of the volume fraction

$$\rho(\overrightarrow{x},t) = \alpha(\overrightarrow{x},t)\ \rho_1 + (1-\alpha(\overrightarrow{x},t))\ \rho_2 \tag{3.26}$$
$$\mu(\overrightarrow{x},t) = \alpha(\overrightarrow{x},t)\ \mu_1 + (1-\alpha(\overrightarrow{x},t))\ \mu_2 \tag{3.27}$$

The governing equation for the volume fraction advection will be derived from the conservation of mass equation. In its conservative form, the conservation of mass is given by

$$\frac{\partial \rho}{\partial t} + \overrightarrow{\nabla} \cdot (\overrightarrow{V} \rho) = 0 \tag{3.28}$$

Substituting the ρ definition for the mixture into this equation, the following arises

$$\frac{\partial[\rho_1\tilde{\alpha} + \rho_2(1-\tilde{\alpha})]}{\partial t} + \overrightarrow{\nabla} \cdot \left\{\overrightarrow{V}[\rho_1\tilde{\alpha} + \rho_2(1-\tilde{\alpha})]\right\} = 0$$
$$(\rho_1-\rho_2)\frac{\partial\tilde{\alpha}}{\partial t} + (\rho_1-\rho_2)\overrightarrow{\nabla}\cdot\left(\overrightarrow{V}\tilde{\alpha}\right) + \rho_2\overrightarrow{\nabla}\cdot\overrightarrow{V} = 0$$

For incompressible flows, $\overrightarrow{\nabla}\cdot\overrightarrow{V} = 0$. Integrating the above in the infinitesimal region around point $\overrightarrow{x}$, and considering ρ_1 and ρ_2 as constants, it results in

$$(\rho_1-\rho_2)\frac{\partial\int_{\delta\forall}\tilde{\alpha}d\forall}{\partial t} + (\rho_1-\rho_2)\int_{\delta\forall}\overrightarrow{\nabla}\cdot\left(\overrightarrow{V}\tilde{\alpha}\right)d\forall = 0 \tag{3.29}$$

Applying Eq.(3.23) into the above (and formally, $\rho_1 - \rho_2 \neq 0$), the VOF advection equation is obtained

$$\frac{\partial\alpha}{\partial t} + \overrightarrow{\nabla}\cdot\left(\overrightarrow{V}\alpha\right) = 0. \tag{3.30}$$

4 Numerical Formulation

By examining the conservation equations of mass and linear momentum, Eqs. 3.1 and 3.18, one may observe that they can be represented by a general conservation equation, as discussed in Patankar (1980). Equation 4.1 presents the general form

$$\frac{\partial(\rho\phi)}{\partial t} + \vec{\nabla} \cdot \overrightarrow{J} = S \tag{4.1}$$

where the variable ϕ is the dependent variable, $\vec{J}$ is the total flux of ϕ, and S a source term.

The total flux has a convective and diffusive contributions and is given by

$$\vec{J} = \rho \, \overrightarrow{V} \phi - \Gamma \, \vec{\nabla} \phi \tag{4.2}$$

where Γ is the appropriate diffusion coefficient.

The source term S may be linearized as follows:

$$S = S_C + S_P \, \phi \tag{4.3}$$

where S_C is the source term constant and S_P is the linear coefficient with respect to ϕ. One issue to consider is that S_P must always be negative to guarantee stability of the numerical model (Patankar, 1980).

The development of the present work was performed within the framework of an open source CFD tool, OpenFOAM® (Weller *et al.*, 1998), which employs the Finite Volume Method to discretize conservation equations with the general form of Eq.(4.1). The Finite Volume Method consists in subdividing the computational domain in a number of control volumes and integrating the conservation equations in space and time in each one of them.

Figure 4.1 depicts two adjacent finite volumes separated by the face f between them. One of the volumes is referred as the owner of the face and its center is referred to by C_j. The control volume adjacent to j through face f is the volume j', whose center is referred to by $C_{j'}$. Volume j' is called the neighbor cell. The vector $\overrightarrow{S_f}$ is the area vector of the face. It is normal to the face and its magnitude is equal to the face area. The face unit normal $\hat{S}_f$ is given by $\hat{S}_f = \overrightarrow{S_f}/|\overrightarrow{S_f}|$.

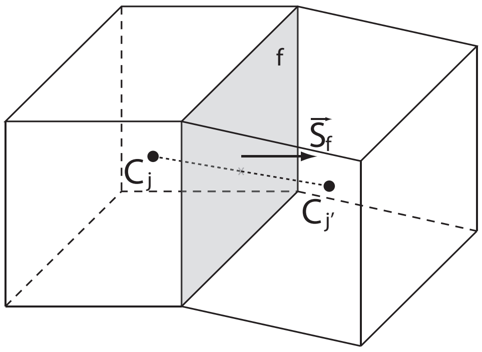


Figure 4.1: Two adjacent finite volumes through face f.

The integral of Eq.(4.1) is obtained by exchanging the integration order depending on the term in consideration, as shown bellow

$$\int_{\Delta\forall}\int_{\Delta t}\frac{\partial\rho\phi}{\partial t}dt\ d\forall + \int_{\Delta t}\int_{\Delta\forall}\vec{\nabla}\cdot\vec{J}\ d\forall\ dt = \int_{\Delta t}\int_{\Delta\forall} S\ d\forall\ dt \tag{4.4}$$

where $\Delta\forall$ refers to the owner volume, and Δt the time step.

The first term can be easily integrated by considering $(\rho\ \phi)$ constant inside the control volume. It can be written after diving by $\Delta\ t$ as

$$\frac{\rho^{n+1}\phi^{n+1} - \rho^n\phi^n}{\Delta t}\Delta\forall \tag{4.5}$$

where n indicates a time instant t and $n+1$ corresponds to the time instant $t+\Delta t$.

Applying the Gauss theorem (Aris, 1990) on the second term on the LHS of Eq. (4.4), it can be transformed from a volume integral into a surface integral, i.e.

$$\int_{\Delta\forall}\vec{\nabla}\cdot\vec{J}\ d\forall = \oint_{\Delta S}\vec{J}\cdot\hat{S}\ dS = \sum_{f\in\Delta S}\overrightarrow{J_f}\cdot\overrightarrow{S_f} \tag{4.6}$$

where ΔS is the element of area and $\hat{S}$ the surface unit normal vector. $\overrightarrow{J_f}$ is the flux at the control volume face, with a convective and diffusive contribution,

$$\overrightarrow{J_f}\cdot\overrightarrow{S_f} = (\rho\vec{V})_f\cdot\overrightarrow{S_f}\ \phi_f - (\Gamma\vec{\nabla}\ \phi)_f\cdot\overrightarrow{S_f} \tag{4.7}$$

and it can be evaluated by several different schemes, like *Upwind, Central Difference* (Verteege & Malalasekera, 2007), QUICK (Leonard, 1979), among many others.

The time integration of flux and source term of Eq. (4.4) can be performed by different schemes, such as Euler schemes or Crank-Nicholson (Patankar, 1980), as

$$\int_{\Delta t} \phi \, dt = \phi^{n+1} f + \phi^{n} (1 - f)) \Delta t \tag{4.8}$$

where $f = 0$ or $f = 1$ correspond, respectively, to Euler explicit or implicit schemes and $f = 1/2$ refers to Crank-Nicholson scheme. An example of OpenFOAM® code illustrating the transient treatment is presented at the Appendix (A.3).

In the next sections, a brief description of the OpenFOAM® solver is presented, so that the new interface treatment, which is being proposed in this work (presented in Chapter 5), can be better understood.

4.1 OpenFOAM

OpenFOAM® framework was selected to be employed at the present work, because it is an open source software package designed to solve generic continuum mechanics problems. OpenFOAM® is being widely used by the scientific community and it counts with a growing community of users and code contributors. The main distribution comes with many application tutorials for different problems such as:

- Basic CFD: potential flows, pure diffusion, convection etc;
- Combustion;
- Compressible Flows;
- Incompressible Flows;
- Multiphase flows;
- DNS - Direct Numerical Simulation – and LES – Large Eddy Simulation;
- Heat Transfer;
- Electromagnetics;
- Chemical Reactions;
- Stress Analysis;
- and even Financial applications.

With this wide collection of types of standard solvers, the programmer may create new solvers based on an existing one. It is very common to find users who just need to add a new term to the transport equation of an existing solver. It can usually be done by modifying a single line of the original code with ease.

OpenFOAM® was originally named as FOAM – Field Operation And Manipulation – and the first formally published article is the work of Weller *et al.* (1998). It consists of a set of libraries and solvers for partial differential equations written in C++ language (Stroustrup, 2013), that enables programmers to easily develop solvers on top of it. It explores many features of C++ language in order to accomplish this goal. Object-orientation allows declaration of classes that mimic mathematical objects such as scalars, vectors, tensors etc. Operator overloading enables readability of the codes, that can look very similar to its mathematical representations. Differential equations can be coded in a way that greatly resembles the way it would be written in paper, as illustrated in Appendix A.3.

4.1.1 Mesh Generation

Standard OpenFOAM® distribution offers several mesh generation tools. One of them is the `blockMesh` utility, to generate parametric meshes with grading and with possible curved edges. The domain is assembled by contiguous blocks that are subdivided according to prescribed numbers of subdivisions. All structured meshes in this work were generated with `blockMesh`. Besides providing its own mesh generators, OpenFOAM® also provides mesh converters to enable meshes modeled in other software to be imported. For unstructured meshes, one option used in this work was to generate meshes in GMSH application (Geuzaine & Remacle, 2009) and, then convert them to OpenFOAM® by calling `gmshToFoam` utility.

The mesh is defined by sets of *points*, *faces*, *cells* and *boundaries*. *Point* is a unique 3D position in space that identify corners of the *faces*. A *face* is an ordered circular list of points, i.e. the last point is connected to the first to close a loop. The connection between two adjacent points in the list is identified as an edge. It is important to enforce the points of a face to be coplanar. There are two types of faces: *internal* and *boundary* faces. *Internal faces* are connections between two (exactly two) adjacent cells. One of the cells is called the face owner (face normal points outside from it) and the other is called the neighbor cell (face normal points into the neighbor cell). *Boundary faces* are faces attached only to the owner cell and its normals point outside of the domain. These faces belong to their respective boundary patches, which identify the boundary condition for the fields.

4.1.2 General Solver Information

The OpenFOAM® platform employs co-located meshes, i.e. all variables are stored at the cell centers. Although it can bring stability problems (Patankar, 1980), a substantial reduction on memory requirements is gained and the discretization coefficients of the three momentum equation components become the same. Besides, OpenFOAM® does not rely on structured meshes. It employs the Finite Volume Method on unstructured grids. All information of cell connectivity is stored in each cell and other topological elements, such as faces, edges and vertices.

The system of algebraic equations can be solved by many different methods in OpenFOAM®. Preconditioned Conjugate Gradient (for symmetric matrices), Preconditioned Bi-conjugate Gradient, Gauss-Seidel, Diagonal Incomplete-Cholesky (for symmetric matrices) (Saad, 2003), Generalized Geometric-Algebraic Multi-Grid (Briggs *et al.*, 2000) among others. The multigrid solver uses the principle of generating a fast solution (using one of the available linear solvers) on a coarse mesh, mapping the solution on the fine mesh as an initial guess in order to obtain a fine solution.

Besides all presented features of OpenFOAM®, it also benefits from being fully parallel. OpenFOAM® applications can run in parallel on distributed processors in a MPI – Message Passing Interface – environment (Quinn, 2003; Squyres, 2012). The method used is called *domain decomposition*, in which the geometry and associated fields are broken into pieces and assigned to separate processors. After each processor solves part of the problem, the master processor (or master node) maps the solutions in the whole domain.

4.2 Standard OpenFOAM VOF solver: interFoam

In the VOF method (Hirt & Nichols, 1981), the α field (volume fraction of the reference phase) is a phase indicator that keeps track of the fluid phases and is advected with the velocity field, Eq.(3.30). Figure 4.2 shows three possible situations for cells positioning with respect to the interface. When a cell is fully occupied by *phase 1*, its value is 1. Where $\alpha = 0$, it means there is no amount of *phase 1* present; it is the same as to say it is fully occupied by the other fluid component, *phase 2*. Cells that present α values $\in (0, 1)$ have a mixture of both phases, what indicates the interface region (the dark gray region in the figure).

In spite of the fact that the interface region has a finite thickness due to its representation through the volume fractions, the interface can be identified as the isosurface of a constant α value. If one takes the midpoint in the α range $[0, 1]$, i.e. $\alpha = 0.5$, to be the interface position, the interface may be treated as sharp even though it is smeared in the α field representation.

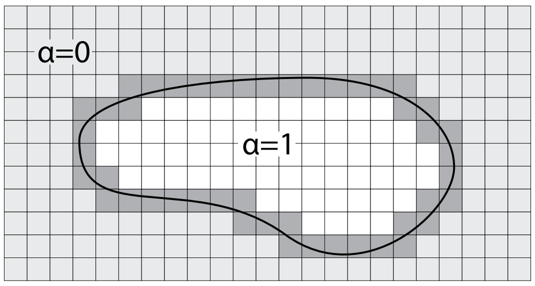


Figure 4.2: The three cases of cell positioning with respect to the interface.

In theory, the volume fraction should vary from 0 to 1 (and vice versa) within one cell thickness. However, due to numerical diffusion during its advection, the interface thickness may consist of several cells as illustrated in Fig.4.3. As discussed in Chapter 2, there are several approaches which are employed in order to keep the interface as sharp as possible. Such approaches involve the application of high order or blended discretization schemes for the convective transport of α. The degree of smearing of the interface will highly depend on the scheme used, and many examples are given by Davis (1994), Ghobadian (1991), among others. Ubbink & Issa (1999) presented a bounded, compressive discretization scheme for the volume fraction equation called CICSAM. Weller (2008) developed an alternative compressive scheme that adds to the equation of α advection an artificial "counter gradient" transport of α towards the interface in order to keep the interface sharp.

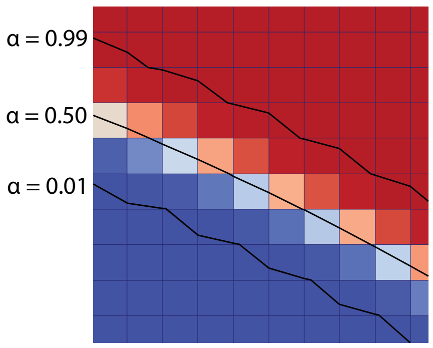


Figure 4.3: Interface smearing.

Since surface tension is a force that acts on the interface surface, it is desirable to have interfaces as sharp as possible, aiming to reduce inaccuracies due to application of surface tension in off-interface regions. Thus, the proper choice of cells where to apply this force is crucial. However, extremely abrupt changes in α in the transition region brings difficulties in normal and curvature estimations as described in Chapter 2. The main focus of the present work is to improve the interface curvature estimation and therefore lead to a better definition of the surface tension force. A detailed description of the methodology in presented at Chapter 5. In the next sections, the transport of the volume fraction field as performed by the standard `interFoam` solver in OpenFOAM® will be highlighted.

4.2.1 Transport of VOF field

In `interFoam`, the advection of α field is performed algebraically, without the need of surface reconstruction. The algorithm to solve α transport is called MULES – Multi-Dimensionsal Limiter for Explicit Solution – and is based on the FCT theory (Zalesak, 1979; Rudman, 1997) to evaluate the face fluxes with limiters. In order to maintain interface sharpness, Weller (2008) adds an "artificial" compression term to the α field transport equation Eq.(3.30). With this compression term, it results in

$$\frac{\partial\, \alpha}{\partial t} + \vec{\nabla} \cdot \left(\vec{V} \alpha\right) + \vec{\nabla} \cdot \left[\vec{V_c}\, \alpha\, (1-\alpha)\right] = 0 \tag{4.9}$$

where the compressibility term is concentrated within the interface region. The term $\alpha\,(1-\alpha)$ ensures it is only active in this region. The "compression velocity" $\vec{V_c}$ is given by

$$\vec{V_c} = \min\left[C_\alpha |\vec{V}|, \max\left(|\vec{V}|\right)\right] \frac{\vec{\nabla}\alpha}{|\vec{\nabla}\alpha|} \tag{4.10}$$

where C_α is the compression coefficient, usually of order 1. The max operator is performed in the whole domain and the min operator is performed locally (at the faces surrounding the concerned cell). Basically, if one desires to turn compressibility off, it is sufficient to set C_α to zero. Larger values of C_α can also be useful, depending on the situation (Weller, 2008). In this work, C_α was kept equal to unity.

According to Wardle & Weller (2013), MULES algebraic advection scheme for the volume fraction is not as accurate as geometric advection schemes, such as PLIC (Rider & Kothe, 1998). However, since it does not require surface reconstruction, it's implementation is much simpler, it is more efficient computationally and it is mass conservative, unlike PLIC, for example (Gopala & van Wachem, 2008). A negative consequence of this last advection scheme, is the possibility of increase in spurious currents, which can be minimized by keeping a low Courant **Co** number. **Co** is computed at the faces centers by

$$\mathbf{Co} = \frac{|\overrightarrow{V}_f \cdot \overrightarrow{S_f}|}{\overrightarrow{d}_{PN} \cdot S_f}\Delta t \tag{4.11}$$

where Δt is the time-step and $\overrightarrow{d}_{PN}$ is the distance between the centers of the cells neighboring the face. In this way, the time step at each step is adjusted based on a maximum user provided Courant number $\mathbf{Co}_{max}$ as follows (Damián, 2013)

$$\Delta t = \min\left\{\frac{\mathbf{Co}_{max}}{\mathbf{Co}}\Delta t^{n-1}, \left(1+\lambda_1\frac{\mathbf{Co}_{max}}{\mathbf{Co}}\right)\Delta t^{n-1}, \lambda_2\Delta t^{n-1}, \Delta t_{max}\right\} \tag{4.12}$$

where $\lambda_1 = 0.1$ and $\lambda_2 = 1.2$ are two hard-coded constants. In `interFoam`, the desired Courant number is achieved by sub dividing the simulation time-step in sub-cycles to maintain the maximum Courant number at the interface low. As pointed by Weller (2008), Courant should be kept $\mathbf{Co}_{max} \leq 0.25$ at the interface, otherwise unboundedness tends to occur rather fast. Gopala & van Wachem (2008) recommend $\mathbf{Co}_{max} < 0.3$, Berberovic *et al.* (2009) recommend the use of $\mathbf{Co}_{max}$ around 0.2. Damián (2013) goes even further and points out that for some situations it was required $\mathbf{Co}_{max} = 0.1$.

In the Finite Volume Method, Eq.(4.9) is integrated over cell j resulting in

$$\int_{\forall_j}\frac{\partial\alpha}{\partial t}d\forall + \int_{\forall_j}\overrightarrow{\nabla}\cdot(\overrightarrow{V}\alpha)d\forall + \int_{\forall_j}\overrightarrow{\nabla}\cdot\left[\overrightarrow{V_c}\alpha(1-\alpha)\right]d\forall = 0 \tag{4.13}$$

Using the Gauss theorem on the second and third terms and Explicit Euler time discretization, just to simplify the description, its discretized version for cell j follows

$$\frac{\alpha_j^{n+1} - \alpha_j^n}{\Delta t}\forall_j + \sum_{f\in\Delta S_j}(F_\alpha + \lambda F_{\alpha c})^n \tag{4.14}$$

where α_j^{n+1} is the value of α at the center of cell j at the next time step, α_j^n is its value at the current time step, the summation is performed over all faces that belong to the boundary ΔS_j of cell j, $\forall_j$ is the j^{th} cell volume, F_α is the face flux due to the velocity field and $F_{\alpha c}$ is the compressive flux. The λ coefficient is a delimiter to concentrate the compressive advection term only at the interface region. It is equal to one at the interface and zero away from it. The fluxes F_α and $F_{\alpha c}$ at the faces f are given by

$$F_\alpha = \phi_f \alpha_{fUP} \tag{4.15}$$

$$F_{\alpha c} = \phi_f \alpha_f + \phi_{cf} \alpha_{cf}(1 - \alpha_{cf}) - F_\alpha \tag{4.16}$$

where ϕ_f is the volumetric flux given by

$$\phi_f = \overrightarrow{V_f} \cdot \overrightarrow{S_f}, \tag{4.17}$$

$\overrightarrow{S_f}$ is the face area vector and $\overrightarrow{V_f}$ is the velocity vector evaluated at face f. It is interpolated between the values of the velocity field at the centers of the cells j and j', which are the indexes of the pair of neighboring cells adjacent to face f. If, for instance, the grid spacing is uniform and $\overrightarrow{V_f}$ is computed by linear interpolation (others can be employed), $\overrightarrow{V_f} = (\overrightarrow{V_j} + \overrightarrow{V_{j'}})1/2$. The term $\alpha f UP$ is the α value evaluated at face f by Upwind scheme

$$\alpha_{fUP} = \begin{cases} \alpha_j & \text{, if } \phi_f \geq 0 \\ \alpha_{j'} & \text{, if } \phi_f < 0. \end{cases} \tag{4.18}$$

The face centered term α_f is evaluated by a combination of Upwind and Central Differencing schemes as follows

$$\alpha_f = \alpha_{fUP} + \lambda_\alpha(\alpha_{fCD} - \alpha_{fUP}) \tag{4.19}$$

where λ_α is a blending coefficient that depends on the method used. In this work it was used the van Leer (1974) scheme. The variable $\alpha f_C D$ is the α value at face f center evaluated by Central Differencing scheme as follows

$$\alpha_{fCD} = \frac{1}{2}(\alpha_j + \alpha_{j'}) \tag{4.20}$$

The compressive flux ϕ_{cf} at face f center is given by

$$\phi_{cf} = \min\left[C_\alpha \frac{|\phi_f|}{|\overrightarrow{S_f}|}, \max\left(\frac{|\phi_f|}{|\overrightarrow{S_f}|}\right)\right](\hat{n}_f \cdot \overrightarrow{S_f}) \tag{4.21}$$

where the max operator is performed in all faces of the grid and the min operator is performed locally. The parameter C_α is the interface compression coefficient, a value that can be adjusted by the user. In this work, $C_\alpha = 1$ for all presented results. The interface unit normal vector is given by

$$\hat{n}_f = \frac{\overrightarrow{n_f}}{|\overrightarrow{n_f}|} \quad , \quad \overrightarrow{n_f} = \frac{(\overrightarrow{\nabla}\alpha)_j + (\overrightarrow{\nabla}\alpha)_{j'}}{2} \tag{4.22}$$

where $(\overrightarrow{\nabla}\alpha)_j$ is the gradient of α at the center of cell j. It is worth to mention here that, in the method proposed in this work, instead of using the gradient of α, the normal $\hat{n}_f$ is obtained by linear interpolation of the interface normals at the cell centers computed by the method explained in Chap.5. Moving back to `interFoam` approach, the cell centered α gradient computed at cell j is given by

$$\overrightarrow{\nabla}\alpha_j = \frac{1}{\forall_j}\sum_{f\in\Delta S_j}\overrightarrow{S_f}\alpha_f \tag{4.23}$$

where f is the index at the faces around cell j and α_f is the value of α interpolated to face f center by linear interpolation of the α values at the centers of cells j and j', where j' in this context is the id of the cell neighboring cell j through face f. See Fig. 4.1, where two neighboring cells are depicted. The face centered α value in the compressive term α_{cf} is also a combination of Upwind and Central Differencing schemes

$$\alpha_{cf} = \alpha_{fUP} + \lambda_{\alpha c}(\alpha_{fCD} - \alpha_{fUP}) \tag{4.24}$$

where $\lambda_{\alpha c}$ is a blending factor given by the `interfaceCompression` scheme

$$\lambda_{\alpha c} = \min\left\{\max\left[1-\max\left(\sqrt{1-4\alpha_j(1-\alpha_j)}, \sqrt{1-4\alpha_{j'}(1-\alpha'_j)}\right), 0\right], 1\right\} \tag{4.25}$$

4.2.2 Momentum and Mass Transport

The velocity and pressure fields are determined by the solution of the momentum equation, Eq. (3.18) coupled with the mass conservation Eq.(3.2) through PISO algorithm (Issa, 1986). For convenience, the pressure in `interFoam` is treated as a modified pressure $p_{\rho gh}$ (Berberovic *et al.*, 2009) defined by

$$p_{\rho gh} = p - \rho\overrightarrow{g}\cdot\overrightarrow{x} \tag{4.26}$$

where $\overrightarrow{g}$ is the gravity acceleration and $\overrightarrow{x}$ the position vector. The pressure gradient combined with the body force due to gravity is then written in terms of the modified pressure as

$$-\overrightarrow{\nabla}p + \rho\overrightarrow{g} = -\overrightarrow{\nabla}p_{\rho gh} - \overrightarrow{g}\cdot\overrightarrow{x}\overrightarrow{\nabla}\rho, \tag{4.27}$$

Thus, the VOF momentum equation, for two incompressible *Newtonian* fluids, as implemented in `interFoam`, for laminar flows, is written as

$$\begin{aligned}\frac{\partial(\rho\vec{V})}{\partial t}&+\vec{\nabla}\cdot(\rho\vec{V}\vec{V}) = \\ &\vec{\nabla}\cdot(\mu\vec{\nabla}\vec{V})+\vec{\nabla}\mu\cdot\vec{\nabla}\vec{V}-\vec{\nabla}p_{\rho gh}-\vec{g}\cdot\vec{x}\vec{\nabla}\rho+\vec{f_\sigma}\end{aligned} \tag{4.28}$$

Note that the last three terms in Eq.(4.28) correspond to the source term of the general equation Eq. (4.4).

The surface tension volumetric force in Eq.(4.28) is computed by the CSF – *Continuum Surface Force*– method (Brackbill *et al.*, 1992)

$$\vec{f_\sigma} = \sigma\kappa\frac{\vec{\nabla}\alpha}{[\alpha_1-\alpha_2]} \tag{4.29}$$

where $[\alpha_1-\alpha_2]$ is the jump in the "color function", i.e. $[\alpha_1-\alpha_2]=1$. Thus, the surface tension force term results in

$$\vec{f_\sigma} = \sigma\kappa\vec{\nabla}\alpha \tag{4.30}$$

where the curvature κ is computed by taking the divergence of the surface normal vector

$$\kappa = \vec{\nabla}\cdot\left(\frac{\vec{\nabla}\alpha}{|\vec{\nabla}\alpha|}\right) \tag{4.31}$$

resulting in a surface force as

$$\vec{f_\sigma} = \sigma\vec{\nabla}\cdot\left(\frac{\vec{\nabla}\alpha}{|\vec{\nabla}\alpha|}\right)\vec{\nabla}\alpha \tag{4.32}$$

The pressure is determined in an indirect way so that the continuity equation is satisfied. OpenFOAM$^{®}$ also has more than one algorithm to solve the pressure-velocity coupling. In particular, the PIMPLE algorithm, which is an adaptation of the SIMPLE – *Semi-Implicit Method for Pressure-Linked Equations* – algorithm (Patankar, 1980), is recommended for steady state flows. For transient problems, the PISO – *(*Pressure Implicit with Splitting Operators) – algorithm (Issa, 1986) is recommended. Both algorithms determine the pressure by combining momentum and continuity equations in order to obtain a velocity field that also satisfies the conservation of mass. The velocity is first predicted (based on the pressure field at the previous time step) by solving the momentum equation Eq.(4.28). First, for every cell of the grid, Eq.(4.28) is integrated and then discretized. Its volume integral over cell j, with volume $\forall_j$ and boundary ΔS_j is given by

$$\begin{aligned}\int_{\forall_j} \frac{\partial \rho \overrightarrow{V}}{\partial t}\, d\forall + \oint_{\Delta S_j} \left(\rho \overrightarrow{V}\overrightarrow{V}\right) \cdot \hat{S}\, dS = \\ + \oint_{\Delta S_j} \left(\mu \overrightarrow{\nabla}\overrightarrow{V}\right) \cdot \hat{S}\, dS + \int_{\forall_j} \overrightarrow{\nabla}\overrightarrow{V} \cdot \overrightarrow{\nabla}\mu\, d\forall \\ - \int_{\forall_j} \overrightarrow{\nabla} p_{\rho g h}\, d\forall - \int_{\forall_j} \overrightarrow{g} \cdot \overrightarrow{x}\overrightarrow{\nabla}\rho\, d\forall + \int_{\forall_j} \sigma\kappa\overrightarrow{\nabla}\alpha\, d\forall\end{aligned} \tag{4.33}$$

and its discretized version, based on the code analysis and the works of Deshpande *et al.* (2012), Klostermann *et al.* (2013) and Damián (2013) follows

$$\begin{aligned}\frac{\rho_j^{n+1}\overrightarrow{V_j^{n+1}} - \rho_j^n\overrightarrow{V_j^n}}{\Delta t}\forall_j + \sum_{f\in\Delta S_j} \left(\rho_f\phi_f\right)^n \overrightarrow{V_f^*} = \\ + \sum_{f\in\Delta S_j} \mu_f^{n+1}\left(\overrightarrow{\nabla}\overrightarrow{V^*}\right)_f \cdot \overrightarrow{S_f} + \left(\overrightarrow{\nabla}\overrightarrow{V_j^n} \cdot \overrightarrow{\nabla}\mu_j^{n+1}\right)\forall_j \\ + \mathcal{R}\left\{\left[\left(\sigma\kappa^{n+1}\right)_f(\overrightarrow{\nabla}\alpha^{n+1})_f - (\overrightarrow{g}\cdot\overrightarrow{x})_f(\overrightarrow{\nabla}\rho^{n+1})_f - (\overrightarrow{\nabla}p_{\rho g h}^n)_f\right]|\overrightarrow{S_f}|\right\}\end{aligned} \tag{4.34}$$

where n and $n+1$ refer respectively to the current and next time step. The operator $\mathcal{R}$ is the reconstruct operator implemented in OpenFOAM®. It receives a field given as faces fluxes and reconstructs it to a cell-centered field. This equation is solved to predict the velocity field $\overrightarrow{V^*}$ that is used as an initialization of the PISO correction loops. Besides, in a given time step, it is solved after the advection of α, i.e. after the solution of Eq.(4.14), so that the α field at the next time step is already known. The fluid density and dynamic viscosity are given by

$$\rho_i^{\tilde{n}} = \alpha_i^{\tilde{n}}\rho_1 + (1-\alpha_i^{\tilde{n}})\rho_2 \tag{4.35}$$

$$\mu_i^{\tilde{n}} = \alpha_i^{\tilde{n}}\mu_1 + (1-\alpha_i^{\tilde{n}})\mu_2 \tag{4.36}$$

where $\tilde{n}$ can be either n or $(n+1)$, i can refer either to cell j or face f and α_f is computed by linear interpolation between the values of α at the centers of the cells adjacent to face f, i.e. cells j and j' (Fig.4.1), such as α_{fCD} in Eq.(4.20).

After solving Eq.(4.34), the pressure and velocity fields are corrected by `nCorrectors` iterations of PISO, where `nCorrectors` is a variable controlled by the user. The PISO iterations are performed by first correcting the pressure field by solving the following equation (Deshpande *et al.*, 2012)

$$\sum_{f\in\Delta S_j}\left(\frac{1}{A_j}\right)_f\left(\overrightarrow{\nabla}p_{\rho g h}^{m+1}\right)_f|\overrightarrow{S_f}| = \sum_{f\in\Delta S_j}\phi_f^* \tag{4.37}$$

where ϕ_f^* is the flux computed by the velocity field $\overrightarrow{V^*}$ from the previous iteration, starting with the one predicted by Eq.(4.34), and A_j is given by (Deshpande *et al.*, 2012)

$$A_j = \left(\frac{\rho_j^{n+1}\forall_j}{\Delta t} + \sum_{f \in \Delta S_j} \frac{1}{2}(\rho\phi)_f^n(1+\Theta_j(f)w) + \sum_{f \in \Delta S_j} \mu_f^{n+1}\Theta_j(f)\frac{|\overrightarrow{S_f}|}{|\overrightarrow{C_j} - \overrightarrow{C_{j'}}|} \right)\frac{1}{\forall_j} \tag{4.38}$$

where $|\overrightarrow{C_j} - \overrightarrow{C_{j'}}|$ is the distance between the centers of the adjacent cells j and j'. The multiplier $\Theta_j(f)$ indicates whether the normal at face f points towards or outwards cell j. It solely depends on the mesh construction itself. In OpenFOAM®, a cell can either be the owner or neighbor to an adjacent face. If cell j owns face f, the face normal points outwards from the cell. If, on the contrary, the cell is a neighbor of the face, the face normal points towards the cell. The multiplier $\Theta_j(f)$ is given by

$$\Theta_j(f) = \begin{cases} 1 & \text{, if cell } j \text{ owns face } f \\ -1 & \text{, if cell } j \text{ is neighbor to face } f \end{cases} \tag{4.39}$$

and the term w is given by

$$w = \begin{cases} (1-\lambda_V) & \text{, if } \phi_f \geq 0 \\ -(1-\lambda_V) & \text{, if } \phi < 0 \end{cases} \tag{4.40}$$

where λ_V can be chosen among many options, such as λ_α in Eq.(4.19). After the new corrected pressure is obtained by solving Eq.4.37, the velocity field is updated by means of updating the flux ϕ at iteration $m+1$

$$\phi_f^{m+1} = \phi_f^* - \left(\frac{1}{A_j}\right)\left(\overrightarrow{\nabla} p_{\rho gh}^{m+1}\right)_f |\overrightarrow{S_f}| \tag{4.41}$$

More detailed descriptions of the algorithm implemented in `interFoam` can be found in Deshpande *et al.* (2012), Klostermann *et al.* (2013) and Damián (2013).

The only modification in the standard `interFoam` performed in this work is in the way the curvature κ and the interface normal $\hat{n}$ are computed. Other aspects of the solver are kept exactly the same as in standard `interFoam` solver. The discussion on curvature and normal computations is presented in Chapter 5.

4.3 Field Initialization

OpenFOAM® provides `setFields` functionality, which enables the user to initialize fields for a given shape, such as spheres, boxes, cylinders etc. This functionality is the default way to initialize fields in OpenFOAM®, however it does not guarantee the correct initial volume/area of the prescribed geometry.

To explain the algorithm, a practical problem be used. Consider a volume fraction field – the α field of VOF for instance – representing an ellipse is to be initialized, such as in Fig.4.4. The application `setFields` solves this problem as follows. It traverses each cell of the computational grid and tests whether the center of the cell lies inside the shape or not. If it does, it assigns the cell with $\alpha = 1$, otherwise it assigns $\alpha = 0$. Figure 4.4 depicts the result of `setFields` algorithm for the 2D ellipse. Notice the staircase pattern on the border of the ellipse.

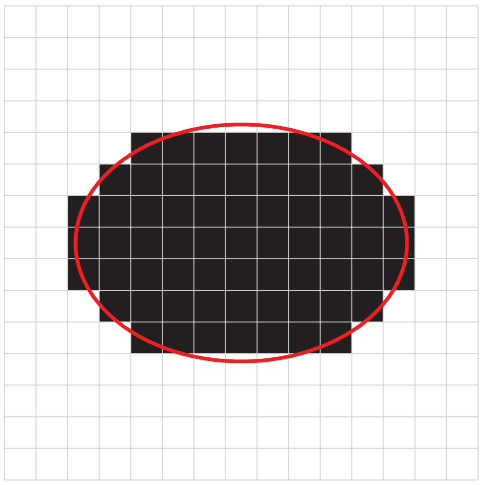


Figure 4.4: VOF field initialized by setFields algorithm.

This initialization can be undesired if one wishes an accurate representation of the given geometry or simply does not desire the staircase pattern. In order to avoid this pattern, artificial smoothing has been employed by some users by projecting the initialized field from cell centers to face centers and then projecting it back to cell centers. Besides the laborious efforts, this procedure does not guarantee preservation of the volume (or area) of the shape anyway.

In order to avoid these drawbacks in field initialization, it was convenient, to come up with a simpler solution capable of generating the desired accurate initial field. The main problem is the following: given a cell (of any type: hexahedral, tetrahedral, prismatic etc) and a shape (sphere, cone, cylinder, etc), what is the volume of the intersection between them? This problem seems straightforward, but not!

The explanation will be carried out in 2D, although the method was implemented in OpenFOAM® for both 2D and 3D simulations. The first attempt was to discretize the cell with quadtree (octree in 3D), but it only seemed reasonable for rectangular (cube in 3D) cells. Therefore, a tool that could abstract the type of either the cell and the shape was necessary.

The approach implemented was the so called Monte Carlo Integration, which is a particular type of Monte Carlo method (Ross, 2013). The idea is to generate uniformly distributed random points inside the cell and count the number of points that lie inside and outside the given shape. In order to answer whether or not a point lies inside a shape, each type of geometry has to implement its own query. It is usually very simple for simple primitive types, and some were implemented in this work, such as: cylinder, sphere, box, oriented box and ellipse. More complex objects can be created as a combination of these simple primitive types, or new primitive types can be easily added. This methodology was named `setFracFields` and the integration algorithm is presented next:

```
box := cell bounding box (xmin,xmax,ymin,ymax,zmin,zmax);
np  := initialize point counter with zero;
npin:= initialize inside point counter with zero;
while not converged
 p := generate random point in box;
 while (p is not inside cell)
  p := generate random point in box;
 end while
 np := np + 1;
 if p is inside shape
  npin := npin + 1;
 end if
end while
if converged and np > 0
 vol := cell volume;
 alpha := vol * (npin / np);
else
 alpha := 0;
end if
```

The pseudo code to generate the random point inside the cell box follows bellow:

```
p := (xmin + (xmax-xmin) * runif(),
      ymin + (ymax-ymin) * runif(),
      zmin + (zmax-zmin) * runif());
```

where `runif` generates a uniformly distributed random variable $\in [0, 1]$. The main loop stops when convergence in volume fraction is achieved within a tolerance or a maximum number of points is reached. Practical situations showed that convergence should be checked after a minimum number of points is generated. In this work, convergence is checked only after the generation of at least 100 points. Besides, this algorithm is only applied to cells crossed by the interface of the shape. It can be efficiently checked by testing if the cell vertices lie inside or outside the shape. If all lie inside, volume fraction is set to $\alpha = 1$, if all lie outside $\alpha = 0$, otherwise the Monte Carlo integration algorithm is performed. For rough meshes this naive test is not enough, so additional points on the cell edges should be checked.

Regarding the shape of the cells, Fig.4.5 displays the two situations that may occur. In Fig.4.5(a), the cell bounding box coincides with the cell itself. And in Fig.4.5(b) the bounding box does not coincide. In this case, random points generated outside the cell (identified with an $\times$) are discarded and not accounted in the total number of points. Only points depicted with hollow circles and solid circles are taken into account. The α value for the cell with volume v is given by

$$\alpha = v \frac{n_\bullet}{n_\circ + n_\bullet} \tag{4.42}$$

where $n_\bullet$ is the number of points in the interception between the shape and the cell; and $(n_\circ + n_\bullet)$ is the total number of points generated inside the cell.

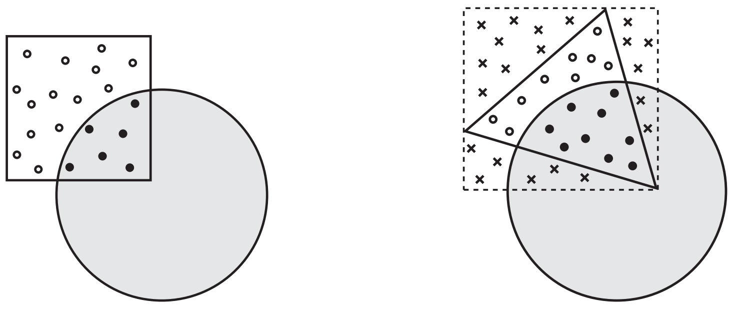

4.5(a): Square cell 4.5(b): Triangular cell

Figure 4.5: A circle interceting two types of cell.

Figure 4.6 displays the field initialized by Monte Carlo Integration. Notice the field fraction values in cells crossed by the interface.

This functionality was described for the volume fraction field but can be applied for a generic field ϕ, where ϕ is either a: scalar, vector or tensor field. The user just needs to prescribe two default values: ϕ_{IN} and ϕ_{OUT}, respectively the values inside and outside the shape. For every cell, the volume fraction α

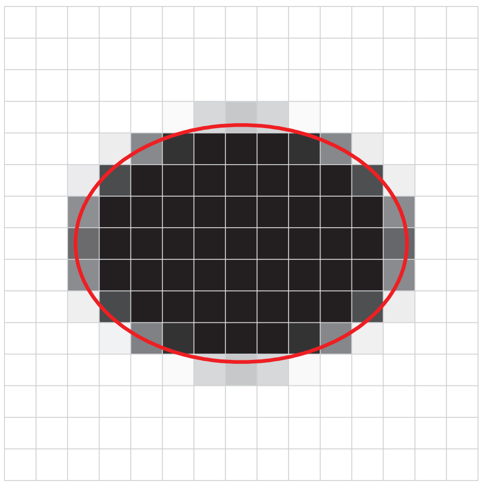


Figure 4.6: VOF field initialized by Monte Carlo Integration in `setFracFields`.

is precomputed and the value ϕ is computed by the linear interpolation given by Eq.(4.43).

$$\phi = \alpha\phi_{IN} + (1 - \alpha)\phi_{OUT} \tag{4.43}$$

Practical situations have shown that the initialized volume $\forall$ converges with an increase in the number of random points, what can be controlled by the user. $\forall$ is computed as a summation over all cells of the domain by

$$\forall = \sum_{N} \alpha_i v_i \tag{4.44}$$

where α_i is the volume fraction assigned to cell i and v_i is the volume of cell i. The user can control the accuracy of the initialization by adjusting the minimum and maximum number of generated points per cell, and the tolerance used for volume fraction convergence checking.

5 Interface Treatment

This chapter is devoted to present in detail the formulation proposed at the present work to enhance the curvature estimation, employing a point-cloud method, based on a Computer Graphics technique. First, the interface is sampled by a cloud of points, then its normals and curvatures are obtained based on the cloud. The following sections describe each step of the procedure in detail.

5.1 Point-Cloud Construction

The point-cloud is obtained straight from the cell-centered VOF field α, that represents the volume fraction of the reference fluid phase in each cell (Kassar & Nieckele, 2015). It is a cell-centered field, i.e. the α values are stored at cell centers. Initially, this field is projected onto the vertices of the grid by an inverse distance interpolation in order to obtain the vertex centered α_P field, where subscript P stands for vertex-point. The value of α_P at any vertex P_i is given by Eq. (5.1), where the values of α at all cells neighboring P_i are taken into account, as depicted in Fig. 5.1.

$$\alpha_{P_i} = \frac{\sum_{j=1}^{N_i} w_{ij}\ \alpha_j}{\sum_{j=1}^{N_i} w_{ij}} \qquad \text{, where} \qquad w_{ij} = \frac{1}{|P_i - C_j|} \tag{5.1}$$

where the index i refers to the i^{th} grid vertex P_i, j refers to the j^{th} cell, w_{ij} is the weighting factor of the j^{th} adjacent cell to vertex P_i and C_j is the j^{th} cell center.

Besides the α field, the cell-centered $\overrightarrow{\nabla}\alpha$ is also projected onto the grid vertices, following the same procedure:

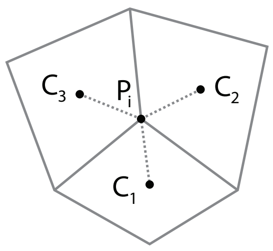


Figure 5.1: Vertex P_i and its three neighboring cells with centers C_1, C_2 and C_3.

$$\vec{\nabla}\alpha_{P_i} = \frac{\sum_{j=1}^{N_i} w_{ij}\vec{\nabla}\alpha_j}{\sum_{j=1}^{N_i} w_{ij}} \tag{5.2}$$

where $\vec{\nabla}\alpha_j$ is the α gradient computed at the cell centers. It computed by

$$\vec{\nabla}\alpha_j = \frac{1}{\forall_j}\sum_f \vec{S_f}\alpha_f \tag{5.3}$$

where f is the index at the faces around cell j and α_f is the value of α interpolated to face f center by linear interpolation of the α values at C_j and at the center of the cell neighboring cell j through face f. See Fig. 4.1, where two neighboring cells are depicted.

It is well documented in the literature that the gradient of α lacks accuracy for curvature computations. This drawback is attributed to abrupt changes in the values of α across the interface. Thus, $\vec{\nabla}\alpha$ here is actually used just to obtain the interface normal vectors senses. Once α and $\vec{\nabla}\alpha$ fields are projected onto the grid vertices, the point-cloud computation may proceed. In fact, we are looking for a set of points equipped with normal vectors to sample the interface.

5.2 Interface Points Sampling

Taking into account that the values of α represent the volume fraction of the reference fluid component at a given region, one may observe that the possible values of α range from 0% to 100%, or from 0 to 1. For now, fluid 1 is considered as the reference fluid component. Regions assigned with $\alpha = 1$ are fully filled with fluid component 1 and regions fully filled with fluid component 2 are assigned with $\alpha = 0$. If one travels through a path in the domain from a point where $\alpha = 1$ to a point where $\alpha = 0$, it is reasonable to say that along the way, the traveler will cross the interface between fluid phases 1 and 2. See Fig.

5.2. At the critical point where the path crosses the interface, the volumetric fractions must be the same. Thus, it can considered that the value of α should be 0.5 at the interface. More formally, we may state: *The interface between two fluid components lies at the isosurface of* $\alpha = 0.5$. Referring once again to the path traversal, it is reasonable to say that, for a traveler to cross the interface along a path, it is sufficient that it travels from a point with $\alpha \leq 0.5$ to a point with $\alpha \geq 0.5$.

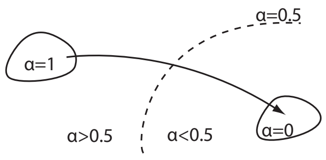


Figure 5.2: Interface between two fluid phases at $\alpha = 0.5$.

After the α field is projected by Eq. (5.1), the values of α are available at the grid vertices. One may travel from each vertex towards every neighboring vertex through the connecting edges looking for the interface. Whenever the value $\alpha = 0.5$ is reached, a new sampling point is stored in the point-cloud. In order to find all sampling points lying on the edges, one just needs to look for edges whose endpoints lie at different sides of the interface. This condition occurs whenever one of the endpoints has $\alpha \leq 0.5$ while the other has $\alpha \geq 0.5$. Edges that attend this condition are marked as *interfacial edges.* The sampling point position is obtained by interpolating the values of α at these edges endpoints.

In this work, two α functions were tested: a linear and a cubic one. The linear function just requires the α values at the endpoints and the cubic is enriched by α derivatives at the endpoints. At this stage, the derivatives we have at hand are $\overrightarrow{\nabla}\alpha$ at the endpoints. The derivative on the edge line is taken as the projection of $\overrightarrow{\nabla}\alpha$ at the edge direction vector. Figure 5.3 shows a triangular cell crossed by an interface and its bottom edge shows two possibilities to represent α: by linear or cubic functions. α_0 and α_1 are the values of α, respectively, at the vertices P_0 and P_1. P_L is the point on the edge where the linear α function is 0.5. P_C is the point for which the cubic α function is equal to 0.5.

For an edge starting at P_0 and ending at P_1, its points can be obtained by a line parametrization with $t \in [0, 1]$

$$P(t) = P_0 + t(P_1 - P_0) \tag{5.4}$$

then, on the edge, one may write α as a function of t.

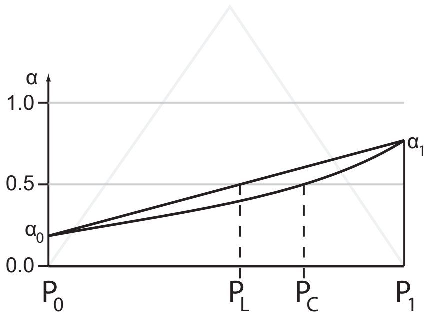


Figure 5.3: Linear and cubic variation of α along and edge from P_0 to P_1.

5.2.1 Linear interpolation of volume fraction

The α function varies linearly along the edge and is defined by

$$\alpha_L(t) = \alpha_0 + t(\alpha_1 - \alpha_0) \tag{5.5}$$

When the edge is an *interfacial edge*, the parameter t_L for which $\alpha_L(t_L) = 0.5$ is given by

$$t_L = \begin{cases} (0.5 - \alpha_0)/(\alpha_1 - \alpha_0) & , \quad \alpha_0 \neq \alpha_1 \\ 0.5 & , \quad \text{otherwise.} \end{cases} \tag{5.6}$$

5.2.2 Cubic interpolation of volume fraction

The α_C function is an adjust of a Cubic Hermite Spline (Isaacson & Keller, 1994) to α values and derivatives at P_0 and P_1 and is given by

$$\alpha_C(t) = (2\,t^3 - 3\,t^2 + 1)\alpha_0 + (t^3 - 2\,t^2 + t)\alpha_0' + (-2\,t^3 + 3\,t^2)\alpha_1 + (t^3 - t^2)\alpha_1' \tag{5.7}$$

where α_0' and α_1' are the derivatives of α with respect to t respectively at $t = 0$ and $t = 1$. These are taken as the projection of $\vec{\nabla}\alpha$ at the corresponding points onto the edge and normalized by the edge length. The value t_C that satisfies $\alpha_C(t_C) = 0.5$ is one of the real roots of the cubic polynomial that lies in the interval $[0, 1]$ and can be solved by the algorithm proposed in Numerical Recipes in C, pp.183 (Press *et al.*, 1992). For completeness, this algorithm is explained in Appendix A.1

5.3 Normal Vectors Sampling

Normal vector estimation is performed in two steps. A first rough estimate based on $\overrightarrow{\nabla}\alpha$ interpolations and a local geometric computation based on the sampling points with their neighboring points.

5.3.1 First step: rough estimate

This step is performed during the sampling points computation by a linear interpolation of the values of $\overrightarrow{\nabla}\alpha$ at each interfacial edge endpoints. The parameter on the edge is the same obtained for the point interpolation and it can be either t_L or t_C. So, the normal value at an interfacial edge, whose interface point lies at a point on the edge represented by parameter t is given by

$$\overrightarrow{n} = (\overrightarrow{\nabla}\alpha)_{P_0} + t\left[(\overrightarrow{\nabla}\alpha)_{P_1} - (\overrightarrow{\nabla}\alpha)_{P_0}\right] \qquad \therefore \qquad \hat{n} = \frac{\overrightarrow{n}}{|\overrightarrow{n}|} \tag{5.8}$$

5.3.2 Second step: geometric refinement

There are many methods available in the literature for normal estimation of point clouds. In this work, three methods were tested and the one with superior results was selected. Two of them are similar and rely on local triangulation, the third one relies on a minimization problem. For now, the discussion must proceed in three dimensions. The reader should notice that two dimensional simulations in OpenFOAM® are, indeed, a particular case of the 3D simulation.

Fan Triangulation

The first two methods are based on the same principle: approximate the normal at point P_0 as the average of the normals of the triangles surrounding P_0. Figure 5.4 displays the interfacial edge E_0 with the interpolated point P_0 identifying the position where the interface crosses it; and also its neighboring points. This set of points constitute the local vicinity of P_0. Further, to be more precise, it is the first layer of neighboring points.

The set of surrounding triangles is built by a triangulation of the first layer of neighboring points just like a triangle fan around P_0.

First, the neighboring points must be sorted. We choose to sort them in counter-clockwise direction for a viewer pointed by the normal at P_0, as depicted in Fig.5.5. At this stage, the normal at P_0 at hand is the first rough

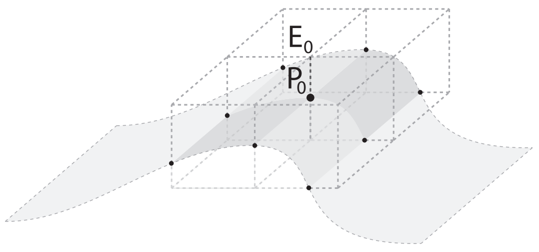


Figure 5.4: An interface point on an edge and its direct neighboring points.

estimate described in section 5.3.1. It is, indeed, the gradient of α interpolated to point P_0.

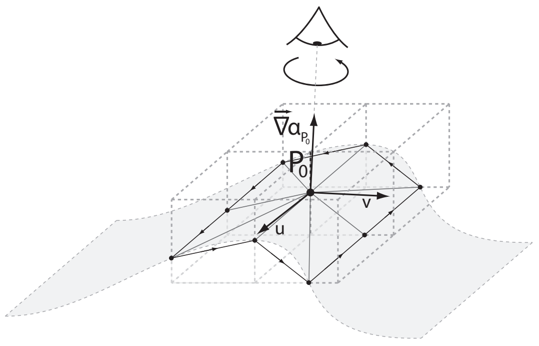


Figure 5.5: Sorting the first layer of neighboring points.

A local frame (u, v) is defined such that u and v are orthonormal vectors and normal to $(\vec{\nabla}\alpha)_{P_0}$. More formally, u and v satisfy

$$u \times v = \frac{(\vec{\nabla}\alpha)_{P_0}}{|(\vec{\nabla}\alpha)_{P_0}|} \tag{5.9}$$

Once the surrounding triangles are obtained, as shown in Figure 5.6, their normals are computed. Let's take, for instance, one of the triangles $P_0P_1P_2$, where the indexes are already sorted. Its normal $\hat{n}_1$ is given by

$$\vec{n_1} = (P_1 - P_0) \times (P_2 - P_0) \qquad \therefore \qquad \hat{n}_1 = \frac{\vec{n_1}}{|\vec{n_1}|} \tag{5.10}$$

Generally, the normal of each of the N surrounding triangles is given by

$$\vec{n_i} = (P_i - P_0) \times (P_{i^+} - P_0) \qquad \therefore \qquad \hat{n}_i = \frac{\vec{n_i}}{|\vec{n_i}|} \tag{5.11}$$

where i^+ is the index of the point next to i. It is given by

$$i^+ = \begin{cases} i+1 & \text{if } i < N \\ 1 & \text{if } i = N \end{cases} \tag{5.12}$$

In Figure 5.6, for example, the point next to P_8 is P_1.

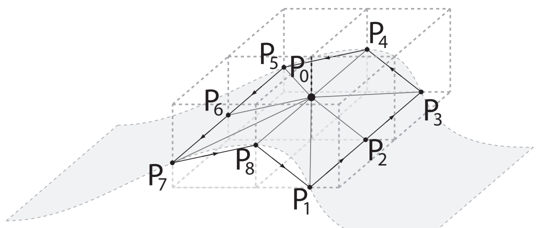


Figure 5.6: Triangle fan around interfacial point P_0.

The normal at P_0 can be obtained by averaging the normals of its surrounding triangles. Many options are available, such as weighting by triangles areas, angles, edges lengths and so on. In this work, two options were tested: an unweighted average and an average weighted by triangles areas. The first method will be called TriFAN and the later TriWEI.

By examining the normal definition, Eq. (5.10), it can be seen that the non-normalized vector is proportional to the triangle area. The unweighted average normal at P_0 in the TriFAN method is given by the average of the surrounding triangles unit normals. On the other hand, the normal by TriWEI is given by the average of the surrounding triangles normals without normalization; i.e. the average norm is obtained weighted by triangle areas.

The respective normals of P_0 ($\hat{n}_{TFAN}$ and $\hat{n}_{TWEI}$) are given by

$$\vec{n}_{TFAN} = \sum_{i=1}^{N} \hat{n}_i \qquad \therefore \qquad \hat{n}_{TFAN} = \frac{\vec{n}_{TFAN}}{|\vec{n}_{TFAN}|} \tag{5.13}$$

and

$$\vec{n}_{TWEI} = \sum_{i=1}^{N} \vec{n}_i \qquad \therefore \qquad \hat{n}_{TWEI} = \frac{\vec{n}_{TWEI}}{|\vec{n}_{TWEI}|} \tag{5.14}$$

Least-Squares Plane Fitting

Another approach in computing P_0 normal vector is to find the normal vector of the plane that minimizes the Euclidean distance to the point cloud in a least-squares sense. We found that increasing the number of neighboring points there is a reduction on the average error in normal estimates. Thus, the point cloud comprises not only the direct neighboring points to P_0, but also a second layer of neighboring points, as depicted in Figure 5.7. This second layer is obtained by traversing a new layer of neighboring grid cells that contain interfacial edges.

First, a plane is defined by a center-point C and a unit normal vector $\hat{w}$, as shown in Fig.5.8. The Euclidean distance, depicted in Fig.5.8, from a point

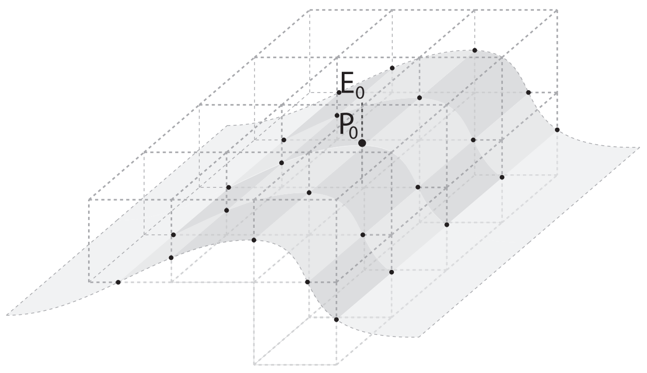


Figure 5.7: An interface point on an edge and its first and second degrees neighboring points.

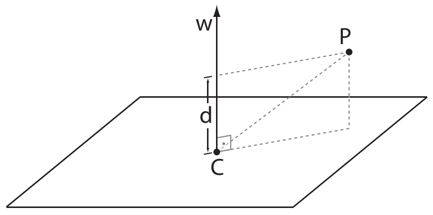


Figure 5.8: A plane defined by the center-point C and normal vector $\hat{w}$ and a point P with distance d from the plane.

P to the plane $(C, \hat{w})$ is given by

$$d = |(P - C) \cdot \hat{w}| \tag{5.15}$$

So, the squared distance is

$$d^2 = [(P - C) \cdot \hat{w}]^2 \tag{5.16}$$

The plane that best fits a set of N points in a least-squares way is such that minimizes the sum of the squared distances to each point. Therefore, the plane that minimizes the following objective function

$$Q(C, \overrightarrow{w}) = \sum_{i=0}^{N} [\overrightarrow{w} \cdot (P_i - C)]^2 \tag{5.17}$$
$$\text{subject to } |\overrightarrow{w}| = 1$$

is sought for the least-squares solution, $\overrightarrow{\nabla} Q = \overrightarrow{0}$. By taking the gradient of Q and equaling to zero, the following holds

$$\sum_{i=0}^{N} [\overrightarrow{w} \cdot (P_i - C)] = 0. \tag{5.18}$$

Dividing it by $N+1$ gives

$$\overrightarrow{w} \cdot \left[\tfrac{1}{N+1} \sum_{i=0}^{N} (P_i - C) \right] = 0. \tag{5.19}$$

As the centroid $\bar{P}$ of the point set is given by

$$\bar{P} = \frac{1}{N+1} \sum_{i=0}^{N} P_i \tag{5.20}$$

Eq. (5.19) reduces to

$$\overrightarrow{w} \cdot (\bar{P} - C) = 0. \tag{5.21}$$

It is to say that the centroid of the point set lies on the plane. So the plane center may taken as this point, $C = \bar{P}$. Thus, the only unknown that remains is the plane unit normal vector.

Consider p_i as the coordinate of point P_i with respect to the centroid $\bar{P}$, i.e.

$$p_i = P_i - \bar{P}. \tag{5.22}$$

The objective function may be rewritten as follows

$$Q(\overrightarrow{w}) = \sum_{i=0}^{N} (\hat{w} \cdot p_i)^2 . \tag{5.23}$$

To account for the constraint $|\overrightarrow{w}| = 1$, one may define the function $G(\overrightarrow{w}) = (\overrightarrow{w})^2 - 1$, and force it to be zero. In the least square solution by the Method of Lagrange Multipliers, we have $\nabla Q = \lambda \nabla G$, for a real number λ, which is an eigenvector problem. It can be further reduced to a singular value problem, as discussed in Shakarji (1998). The normal vector $\overrightarrow{w}$ is taken as the singular vector that corresponds to the smallest singular value of the 3×3 matrix $M^T M$, where M is the $(N+1) \times 3$ matrix of p_i components (x_i, y_i, z_i) given by

$$M = \begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix} \tag{5.24}$$

The normal vector of P_0 is, then taken as

$$\hat{n}_{LSQR} = \overrightarrow{w}. \tag{5.25}$$

where $\overrightarrow{w}$ is a unit vector. As noted in Shakarji (1998), the eigenvectors of $M^T M$ are also singular vectors of M. So, instead of computing the matrix $M^T M$, this work finds $\overrightarrow{w}$ by taking the SVD – Singular Value Decomposition (Golub & Van Loan, 1996)– of M and choosing the singular vector that corresponds to the smallest singular value of M.

5.4 Curvature Computation

Curvature measures the rate of bending of curves or surfaces. In curves, the curvature is defined by the rate of variation of the tangent vector $\overrightarrow{t}$ with respect to the arc length s.

$$\kappa_{\text{curve}}(s) = \left| \frac{d\overrightarrow{t}(s)}{ds} \right|. \tag{5.26}$$

Equation (5.26) shows that the curvature is uniquely defined at any point on a curve. On surfaces, however, at a given position, there are infinite directions where the tangent vector can point to. Any vector perpendicular to the normal vector at a given position is tangent to the surface. Therefore, any point on a surface can present infinite curvatures associated to it. Two of these directions, however, indicate the maximum and minimum bending of the surface. These are called the *principal directions* and the curvatures associated to them are the *principal curvatures*, usually represented by κ_1 and κ_2 (Kreyszig, 1991).

The average between the principal curvatures is called the *mean curvature*. It is often found in the literature represented by H and is given by

$$H = \frac{\kappa_1 + \kappa_2}{2}. \tag{5.27}$$

Twice the value of H gives the value of the curvature κ, that is part of the surface tension force term present in the conservation of momentum Eq.

(3.18). Here it will be referred just by *curvature* and will be denoted by the symbol κ. In terms of the principal curvatures, it is given by

$$\kappa = \kappa_1 + \kappa_2 \tag{5.28}$$

5.5 Curvature Computation on a Point-Cloud

Many works can be found in the literature for curvature computations on a point-cloud with or without prescribed normals.

Yang & Qian (2007) adjust a *moving least-squares* surface (Levin, 2003) to the point cloud equipped with normals and, then, compute the curvatures with closed formulas. This procedure is computationally expensive and depends on a good estimate of a scale parameter, that gives rise to further complexity and may depend on empirical experience.

There are authors who triangulate the points, with methods such as Delaunay Triangulation (de Berg *et al.*, 2008), and extract the curvatures from the triangle meshes with methods such as the one proposed by Batagelo & Wu (2007).

Seibert *et al.* (2010) define a finite number of planes containing the normal vector of a point in the cloud. These planes sweep a 360° rotation around this normal. For each plane, an osculating circle is fitted to the neighboring points on the vicinity of this plane, giving rise to a value of curvature for each direction. With these values of curvatures, κ_1 and κ_2 are computed. This approach is considered computationally expensive once many fitting problems must be solved for every point in the point cloud. Besides, it requires a high density of points on the point cloud.

These works are mainly focused in Computer Graphics applications such as image and scanned data processing. Besides curvatures, they aim at finding accurate representations of the surfaces. The focus of this work, however, is not on a good representation of the surface itself, but on a good representation of its curvatures. Therefore, the method proposed by Cheng & Zhang (2009) was the one chosen and gave feasible results. It basically fits linear functions to represent the normal vector in the vicinity of a given point. From these functions, the curvatures may be computed by analytic differentiation. The authors argue that the minimization of the algebraic distance between the points of the cloud and a surface – in a least squares sense – does not necessarily result in the minimization of the error in the curvature estimates. According to them, for the sake of curvature estimation, fitting functions to normal vectors in local regions is more accurate and robust than fitting high order surfaces to point positions. It makes sense, indeed, if one notices that curvatures are second-order derivatives of the surface positions and measure directly the variation of normals itself. The next section will discuss the method proposed by Cheng & Zhang (2009) in more details.

5.6 Curvature Estimation by Normal Fitting

For every point E_0 in the cloud, the curvature should be estimated based on its neighboring points. A local system $(\hat{u}, \hat{v}, \hat{w})$ is defined, where $\hat{w}$ is equal to the normal vector at $\overrightarrow{E}_0$, i.e. $\hat{w} = \hat{n}_0$. The vectors $\hat{u}$ and $\hat{v}$ are such that $\hat{u} \times \hat{v} = \hat{w}$. These vectors give the basis for a plane with the origin at point $\overrightarrow{E_0}$. It will be called the *local frame* at point E_0.

One could write functions to represent the normal vector of the surface around point E_0 in this local frame as functions of the parameters u and v. The normal vectors $\hat{n}_i$ of every point on the vicinity of E_0 are projected onto the local frame to give rise to its components n_{ui} and n_{vi}, where $n_{ui} = \hat{n}_i \cdot \hat{u}$ and $n_{vi} = \hat{n}_i \cdot \hat{v}$. Notice that the projection of $\hat{n}_0$ onto the local frame gives rise to $n_{u0} = 0$ and $n_{v0} = 0$. Figure 5.9 illustrates the local frame around point E_0 and the neighboring point E_i with normal vector $\hat{n}_i$ projected onto the plane, giving rise to the 2D vector (n_{ui}, n_{vi}).

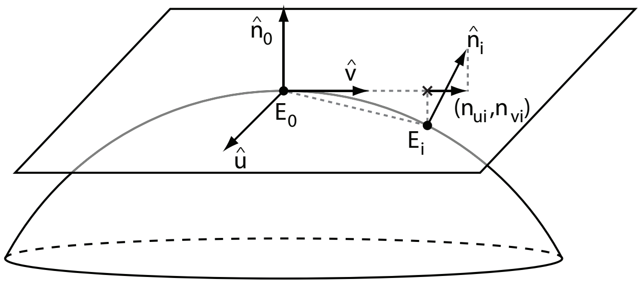


Figure 5.9: Local frame around interface point E_0.

The surface normal vector components in the local frame of E_0 may be described by the following functions (Cheng & Zhang, 2009)

$$n_u(u, v) = \nu_1 + w_{11}\ u + w_{12}\ v + \vartheta(u^2 + v^2) \tag{5.29}$$

$$n_v(u, v) = \nu_2 + w_{21}\ u + w_{22}\ v + \vartheta(u^2 + v^2) \tag{5.30}$$

where w_{11}, w_{12}, w_{21} and w_{22} are the derivatives of the normal vector, $(\nu_1, \nu_2) = (n_{u0}(0, 0), n_{v0}(0, 0))$ and ϑ denotes a higher order term. The high order terms may be neglected resulting in linear functions.

In order to obtain the parameters w_{11}, w_{12}, w_{21} and w_{22}, one may solve two independent unconstrained minimization problems (Cheng & Zhang, 2009). One minimization for the u component and another for the v, such as

$$\min \sum_{i=0}^{N} (\nu_1 + w_{11}\ u_i + w_{12}\ v_i - n_{ui})^2 \qquad \text{and} \tag{5.31}$$

$$\min \sum_{i=0}^{N} (\nu_2 + w_{21}\ u_i + w_{22}\ v_i - n_{vi})^2 \tag{5.32}$$

where N is the total number of points in the vicinity of E_0 and n_{ui} and n_{vi} are the coordinates of the projected normal vector $\hat{n}_i$ onto the local frame. These minimization problems give rise to the same 3×3 symmetric matrix, that needs to be inverted. It was observed that the solution of these two minimization problems gives good approximations to the curvature. Once the values of w_{11}, w_{12}, w_{21} and w_{22} are obtained, the curvature may be computed by taking the eigenvalues of the Weingarten curvature matrix $\boldsymbol{W}$.

$$\boldsymbol{W} = -\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \tag{5.33}$$

The curvatures k_1 and k_2, where $k_1 \geq k_2$, are the eigenvalues of $\boldsymbol{W}$ itself. Then, the curvature κ_{E_0} at point E_0 is computed by Eq.(5.28). As $\boldsymbol{W}$ is a 2×2 matrix, the eigenvalue problem can be solved by closed formula. The method employed in this work to solve for the eigenvalues of this specific matrix is described in details in Appendix A.2.

Once all interface sampling points are filled with curvatures and normals, this information is projected onto the grid of cells, where the momentum conservation Eq.(3.18) is solved.

5.7 Projection of Surface Traction onto the Finite Volumes Grid

As the curvatures are computed on each sampling point, they must be projected onto the Finite Volumes cell centers in order to be accounted in the momentum equation. The procedure employed in this work is similar to the one employed in Front Tracking Method and described in Tryggvason *et al.* (2001). Consider a general quantity ϕ to be projected from the point cloud to the fixed grid of cells. The projection method is based on the fact that the quantity attributed to the interface points must be conserved when projected onto the cells. So it is to say that the integral of the quantity over the interface is the same as its integral over the cell volumes. In a mathematical form, it can be written as

$$\int_{\Delta s} \phi_s ds = \int_{\Delta\forall} \phi_v d\forall \tag{5.34}$$

and applying for the present situation,

$$\int_{\Delta s} \sigma\ \kappa\ \hat{n}\ ds = \int_{\Delta\forall} \sigma\ \kappa\ \delta(n)\hat{n}\ d\forall \tag{5.35}$$

where Δs is the interface surface domain; $\Delta\forall$ is the grid volumetric domain; ϕ_s is the value of the quantity on the surface; ϕ_c is the value of the quantity in the cell volumes.

The reader may notice that, so far, there was no concern about the interface area. Indeed, in order to fulfill Eq.(5.34), the area of interface that belongs to each point must be computed. The computational cost of area computation for each point may be reduced if one takes advantage on the volumetric grid topology. Notice that any sampling point lays on a grid edge. Therefore, one may compute the area of interface enclosed by each cell. Figure 5.10 shows the interface enclosed by a cell and subdivided into four sub-areas. Each sub-area is assigned to its adjacent sampling point, as highlighted.

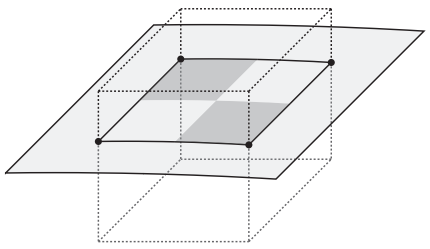


Figure 5.10: The area of interface enclosed by a cell. Each of the four portions are assigned to their respective points.

This area is computed as follows: Consider a cell with N sampling points – already equipped with normal vectors. First, the centroid and the average normal of the N points are computed. Then, the points are sorted around the centroid in counter-clockwise sense for a viewer pointed by the normal, similarly as already discussed in Fig.5.5. After sorting, a polygon arises and its area is computed by simple triangulation around any corner. Notice that this method for area computation requires the polygon to be convex. As the grid cells are convex – it is our requirement – the polygons are also convex. This area is, then subdivided equally in as many points as there are and each sub-area is added to each point. Figure 5.11 depicts one sampling point surrounded by four cells and the areas attributed to it.

At this stage, every sampling point i carries more information than just its position. It is a local representation of the interface surface itself, so it can be called a surface element. This concept is often used in Computer Graphics,

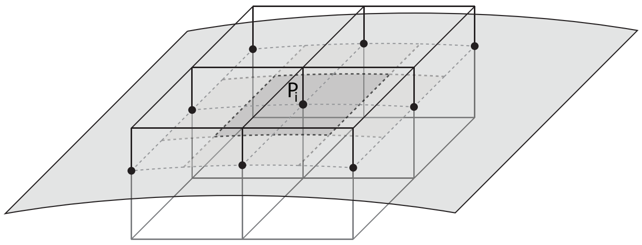


Figure 5.11: An interface point P_i and the area assigned to it as a contribution from its four neighboring cells.

and is referred by *surfel*, as in Pfister *et al.* (2000). Figure 5.12 depicts the idea of *surfel* in this work. It is a representation of the surface with the following information:

1. position P_i,
2. normal vector $\hat{n}_i$,
3. curvature κ_i,
4. area A_i.

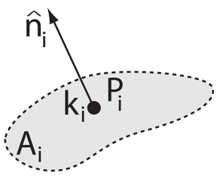


Figure 5.12: A surface element, or *surfel*.

With these information, we are now capable of projecting the curvatures onto the grid cells. This procedure is described in Tryggvason *et al.* (2001) and is detailed here for both completeness and to expose slight differences.

First of all, the objective of this procedure is to obtain the interface traction force per unit volume in each grid cell. We start by looking at this force at the sampling points, that can be interpreted as the centers of elements of surface with area A_i, normal vector $\hat{n}_i$ and curvature κ_i. Therefore, for each element of surface i (sampling point P_i), the interfacial traction force $\overrightarrow{F^s_{\sigma_i}}$ is given by

$$\overrightarrow{F^s_{\sigma_i}} = \sigma\ \kappa_i\ A_i\ \hat{n}_i \tag{5.36}$$

This force shall be spread through the adjacent cells in a way that satisfies Eq.5.34.

For each interfacial point P_i, a characteristic length h_i is determined as

$$h_i = 2|e_i| \tag{5.37}$$

where $|e_i|$ is the length of the edge where P_i lies in. Then, all neighboring cells whose centroids C_j lie inside a regular $2h_i \times 2h_i \times 2h_i$ cube centered at P_i are selected. For 2D simulations, instead of a cube, a $2h_i \times 2h_i$ square is employed. For each selected j cell, the distance vector $\overrightarrow{r_{ij}}$ from its centroid C_j to P_i is computed as

$$\overrightarrow{r_{ij}} = C_j - P_i. \tag{5.38}$$

Then, based on $\overrightarrow{r_{ij}}$, a weight w_{ij} is attributed to cell j.

$$w_{ij} = d_{ij_x} d_{ij_y} d_{ij_z} \tag{5.39}$$

where d_{ij_k} is a function of r_{ij_k} and h_i

$$d_{ij_k} = \begin{cases} (h_i - |r_{ij_k}|)/h_i & , |r_{ij_k}| < h_i \\ 0 & , |r_{ij_k}| \geq h_i \end{cases} \tag{5.40}$$

For 2D simulations, $d_{ij_z} = 1$. Therefore, the contribution of volumetric force provided by *surfel* P_i to cell j is given by

$$\overrightarrow{f^{v}_{ij}} = \frac{w_{ij}}{\sum w_{ij}} \, \sigma \, \kappa_i \, \frac{A_i}{\forall_j} \hat{n}_i \tag{5.41}$$

The total volumetric interfacial force $\overrightarrow{f_{\sigma j}}$ at cell j results from the sum of all neighboring *surfel* contributions $\overrightarrow{f^{v}_{ij}}$. This term is explicitly used in the momentum equation as a source term.

$$\overrightarrow{f_{\sigma j}} = \frac{1}{\forall_j} \sum_{i}^{N} \frac{w_{ij}}{\sum w_{ij}} \, \sigma \, \kappa_i \, A_i \, \hat{n}_i \tag{5.42}$$

where N is the total number of neighboring points. Of course, in the code implementation, it is only needed to traverse the points in the vicinity of cell j, i.e. points for which $w_{ij} > 0$.

`interFoam` requires the surface tension term to be provided at face centers, so the cell-centered interfacial force field $\overrightarrow{f_{\sigma j}}$ is projected onto the face centers by central-differencing interpolation and projected onto the face normal $\hat{S}_f$.

5.8 Curvature Computation Algorithm Overview

In summary, the proposed algorithm for curvature computation follows:

1. Compute the $\vec{\nabla}\alpha$ field at cell centers, based on the raw cell-centered α field
2. Project α and $\vec{\nabla}\alpha$ fields from cell centers to mesh vertices
3. For each edge crossed by an interface, compute the parameter t on the edge line where the interfacial point lies
   (a) By linear interpolation Eq.(5.5), or
   (b) By spline interpolation Eq.(5.7).
4. Compute the interfacial points, Eq.(5.4), and the first normal estimates, Eq.(5.8), on the edges crossed by interface
5. Improve normal estimates by one of the methods:
   (a) Fan triangulation (TFAN or TWEI) or
   (b) Least-squares fitting of the neighboring points (LSQR)
6. Compute curvatures at the interfacial points by normal fitting (Section 5.6)
7. Compute surface area corresponding to each interfacial point
8. Project surface tension force from interfacial points to cells on the fixed grid.
9. Interpolate surface tension force from cell centers to face centers and project onto face normals.

Unless otherwise noticed, the default methods for normal computation and point cloud sampling in PC-VOF are, respectively, least-squares planes (Sec.5.3.2) and linear interpolation of α field (Sec.5.2.1).

6
Verification and Validation Tests

To demonstrate the performance of the methodology proposed at the present work, a few simple and basic preliminary results are examined here. Initially, the performance of the different point interpolation and normal computational methods is evaluated in a test case comprising a circular interface immersed in a fluid with constant prescribed velocity field. In particular, the curvature computed by the different methods are analyzed. Then, the effect of solving the velocity field coupled with the circular interface evaluation is discussed. The classical problem of an oscillating bubble is examined and the predictions are compared with the exact solution. Finally, the problem of an initially square interface, that evolves to circular due to surface tension force, is studied.

6.1
Curvature Accuracy

The purpose of this section is to investigate the performance of the different methods for point interpolation (Linear and Cubic Spline) and normal computation (LSQR, TriFAN and TriWEI) presented in Chapter (5), with regard to the accuracy on the curvature estimation of circular interfaces.

Figure 6.1 illustrates the domain considered and the initial interface position. The domain size in the $x - y$ directions is 1.5×1. The circle is centered at $(0.5, 0.5)$ and its radius measures $r_0 = 0.2$. So the exact curvature along the whole interface is $k_{exact} = 1/0.2 = 5$.

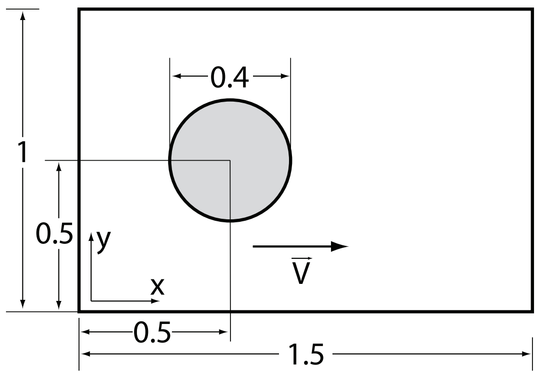


Figure 6.1: Setup for the static and moving circular interface.

Initially, the flow field was set equal to zero and the circle (α field) initialization was performed in two steps: first, the method explained in Sec.4.3 was employed and a short relaxation simulation during 0.5 unit of time (dimensionless) was performed with gravity turned off. All properties were set to unity ($\rho_1 = \rho_2 = \mu_1 = \mu_2 = \sigma = 1$). Then, the circle was set to motion in a uniform velocity field $\overrightarrow{V} = [U, 0, 0]^T$ for a period of 0.5 of time, where $U = 1$.

Different mesh resolutions were employed in order to observe its role on curvature values. The mesh spacing h was set to $1/32, 1/64, 1/128$ and $1/256$. The time-step was limited by (Popinet, 2009)

$$\Delta t \leq \sqrt{\frac{\rho\ h^3}{\pi\ \sigma}} \tag{6.1}$$

The following quantities were obtained at $t = 0$, and at $t = 0.5$:

- average radius error
- average normal error
- average curvature
- curvature standard deviation

The average radius error is a measurement of the error in the point sampling interpolation, that can either be linear or cubic. It was obtained by

$$Err_{ri} = \frac{|r_i - r_0|}{r_0} \times 100 \tag{6.2}$$

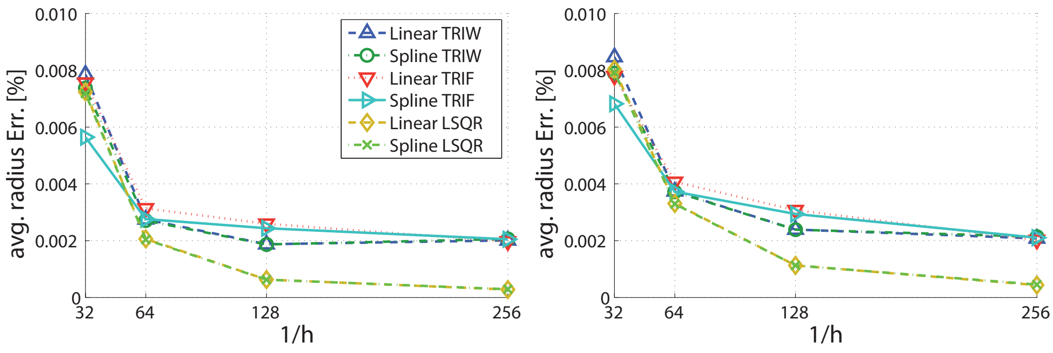
where i is the sampling point index, r_i is the distance of the ith sampling point to the circle center and r_0 is the exact value of the radius, i.e. 0.2. And the error in normal estimates is computed by

$$Err_{ni} = |\hat{n}_i - \hat{n}_{i0}| \tag{6.3}$$

where $\hat{n}_i$ is the unit normal computed at point i and $\hat{n}_{i0}$ is the expected unit normal vector for the point. It is the normalized vector from the circle center to the sampling point position. The circle center at each time step is given by $(0.5 + U\ t, 0.5)$.

The following results compare the six possible combinations of the methods: two options for the point interpolation – Linear and Cubic Spline – and three options for normal computation – LSQR, TriFAN and TriWEI. Here, the last ones are referred as TRIF and TRIW, respectively.

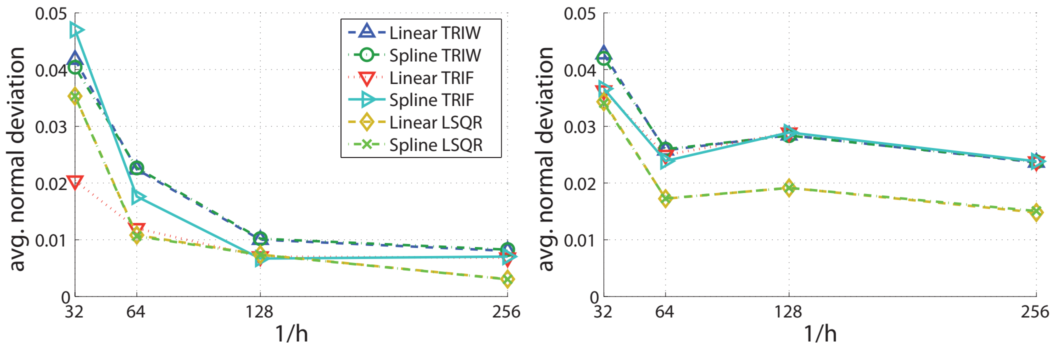
Figure 6.2 shows the average error in the positions of the point cloud at $t = 0$ ($U = 0$) and at $t = 0.5$ ($U = 1$). This error is measured by the error in the distance to the circle center, i.e. the error of the radius. All methods were able to determine the average radius with a very small error. Results for Linear LSQR and Spline LSQR combinations show the best results for both instants of time. Also, as expected, convergence with mesh refinement is observed for all six combinations. Further, the average error obtained with zero and non-zero prescribed velocity is practically the same for all cases.



6.2(a): $t = 0$ $(U = 0)$ 6.2(b): $t = 0.5$ $(U = 1)$

Figure 6.2: Average radius deviation.

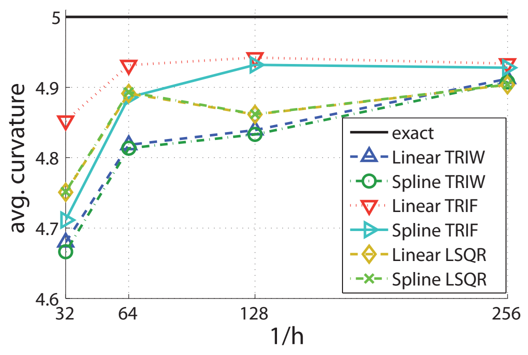
Average normal deviation by Linear LSQR and Spline LSQR also provide best results, as depicted in Fig. 6.3. Here, a slight deterioration on the normal estimation is noticed with the non-zero velocity field after the interface advection. It is due to small perturbations in the point cloud positions after the advection, that is performed by the original `interFoam` code (MULES algorithm). It is also perceivable that, after the interface advection, the difference in normal errors between LSQR and the the triangulation methods increases. It indicates that LSQR method is less sensible to perturbations on the positions of the point cloud.



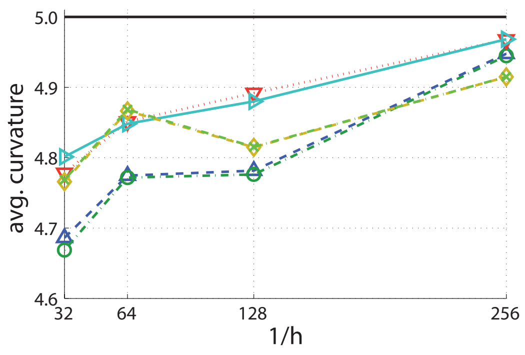
6.3(a): $t = 0$ $(U = 0)$ 6.3(b): $t = 0.5$ $(U = 1)$

Figure 6.3: Average unit normal deviation.

Average curvature is presented in Fig.6.4, where the triangulation methods shows slightly better results than LSQR, however, as it will be shown next, the standard deviation of curvature for LSQR is significantly smaller than the other methods.



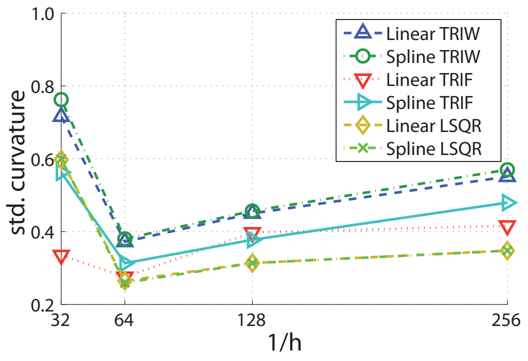
6.4(a): $t = 0$ $(U = 0)$



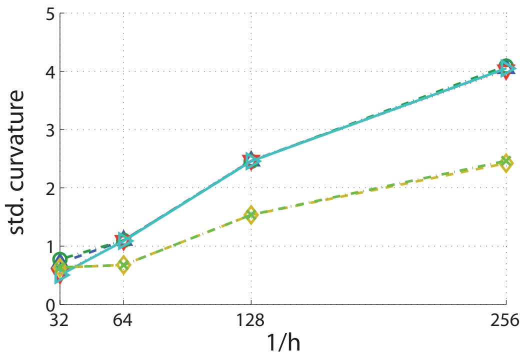
6.4(b): $t = 0.5$ $(U = 1)$

Figure 6.4: Average curvature.

Figure 6.5 displays the standard deviation in curvature estimates for the six methods. As it can be seen, LSQR combinations results in the lowest standard deviations. In practice, what occurs is that this method is more stable than the others. Since the triangulation methods rely only on a small number of neighboring points, they are more sensible to errors in the points estimations. On the other hand, LSQR takes the normal in a least squares sense, what



6.5(a): $t = 0$ $(U = 0)$



6.5(b): $t = 0.5$ $(U = 1)$

Figure 6.5: Curvature standard deviation.

dampens out the errors in position estimates. By looking at these results and by practical experience, LSQR method was chosen as the default method for normal estimates in this work. Depending on the situation, the triangulation methods can provide a locally much higher or much lower values of curvatures, what can cause local perturbations on the interface. In an extreme level, it may cause unphysical interface breakup events.

Between linear and cubic spline interpolation, one may notice that for LSQR normal computation, there is no improvement in using spline interpolation. Besides, for TriFAN and TriWEI, spline interpolation results in higher standard deviations (for both cases, zero or non-zero velocity field). Also, it results in worse curvature estimates in coarse meshes. Since the spline interpolation needs the gradient of α, these results allow us to say that the gradient of volume fraction $\overrightarrow{\nabla}\alpha$ does not enrich the α interpolation. So, it can be also stated that $\overrightarrow{\nabla}\alpha$ will not precisely determine the normal. This conclusion is also supported by the literature, that shows that $\overrightarrow{\nabla}\alpha$ provides inaccurate normal estimates, as discussed in Chapter 2. Therefore, the default method chosen for point interpolation is the linear method. Anywhere in this work, where PC-VOF is mentioned, the method adopted for point and normal estimations are respectively, linear interpolation and least squares fitting, unless otherwise specified.

6.2 Circle Moving in an Initially Prescribed Velocity Field

Popinet (2009) describes a validation test case for a circular interface immersed in a initially prescribed velocity field in order to evaluate the coupling between advection, solution of the momentum equation and curvature computation. Therefore, the Navier-Stokes equations are solved and the α field is advected. With this test it is possible to observe the influence of advection in the perturbation of the interface and also in the accuracy of curvature computation.

Here, the problem configuration was defined exactly as employed by Popinet (2009) to allow for comparison between the methodologies. The same domain as illustrated in Fig. (6.1) is employed. The circle also has the same diameter, i.e. $D = 0.4$. As in the previous example, both phases consist of the same fluid properties: $\rho = \rho_1 = \rho_2$ and $\mu = \mu_1 = \mu_2$. An initial velocity field is set to $\overrightarrow{V} = [U, 0, 0]^T$, with $U = 1$. Boundary conditions on the left and right patches are set to cyclic and on top and bottom patches set to zero gradients for pressure, velocity and α.

This problem is governed by three parameters, Reynolds (**Re**), Weber (**We**) and Laplace (**La**) numbers, defined as

$$\mathbf{Re} = \frac{\rho\, U D}{\mu^2} \qquad ; \qquad \mathbf{We} = \frac{\rho\, U^2 D}{\sigma} \qquad ; \qquad \mathbf{La} = \frac{\sigma\, \rho\, D}{\mu^2}. \tag{6.4}$$

Popinet (2009) presents results for simulations performed for various Laplace **La** numbers, where the values were set to $\mathbf{La} = 120, 1200, 12000$ and ∞. The **We** number was kept constant and equal to $\mathbf{We} = 0.4$. Further, the density of both phases were set equal to $\rho = 1$, and the surface tension was $\sigma = 1$. Thus, the fluids viscosity μ were determined from the Laplace number.

To evaluate the performance of the point cloud method, the spurious velocities obtained with the present methodology (denominated here as PC-VOF (LSQR) and PC-VOF (TriFAN)) are compared with the results obtained with `interFoam`, as well as those presented by Popinet (2009). PC-VOF (LSQR) refers to PC-VOF method with normals computed by least-squares-planes and PC-VOF (TriFAN) refers to normals computed by fan triangulation. Popinet (2009) presented a method combining an adaptive quad/octree spatial discretisation, geometrical Volume-Of-Fluid interface representation, balanced-force CSF with height-function curvature estimation.

The spurious velocities are estimated through the RMS velocity V_{RMS}, given by

$$V_{RMS} = \sqrt{\frac{1}{\int_\Omega dV} \int_\Omega \overrightarrow{V'} \cdot \overrightarrow{V'} dV} \tag{6.5}$$

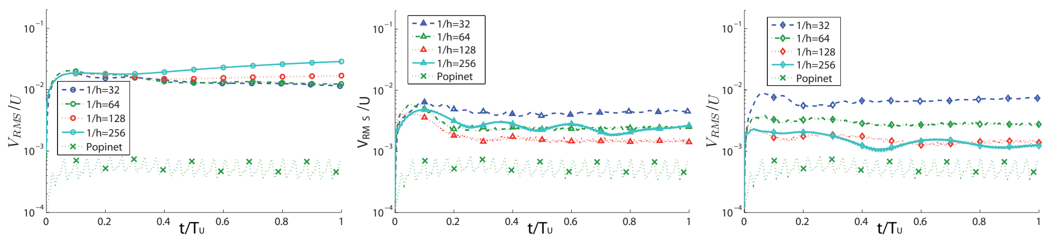
where $\overrightarrow{V'}$ is the fluctuation

$$\overrightarrow{V'} = \overrightarrow{V} - \langle \overrightarrow{V} \rangle \qquad ; \qquad \langle \overrightarrow{V} \rangle = \frac{1}{\int_\Omega dV} \int_\Omega \overrightarrow{V} dV \tag{6.6}$$

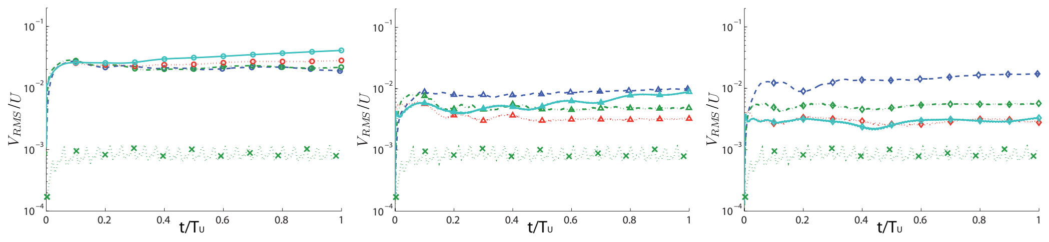
and $\langle \overrightarrow{V} \rangle$ is the mean velocity.

Also, the error in the average curvature avg($\langle \kappa \rangle$) are presented for the various Laplace numbers and four different mesh refinements. The domain is refined in regular $1.5n \times n$ meshes, where $n = 32,\ 64,\ 128$ and 256. The time step is limited according to Eq.(6.1)

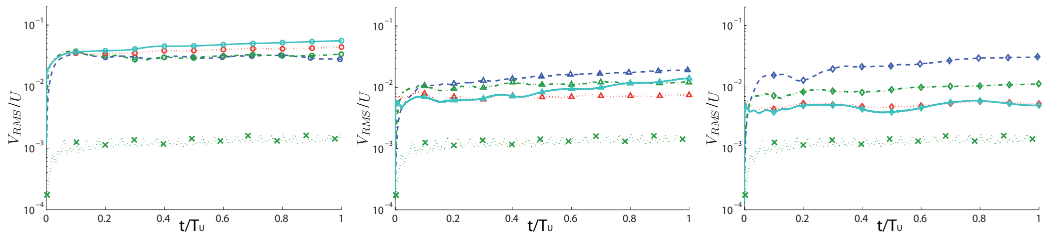
Figures 6.6 shows values of V_{RMS} for simulations performed by `interFoam`, PC-VOF (TFAN) and PC-VOF (LSQR) for the various Laplace numbers. Results from Popinet (2009) are also added for the respective Laplace number. The abscissae show a dimensionless time t/T_U, where $T_U = D/U$. The ordinate scale was kept the same to facilitate comparison. It can be clearly seen that the spurious currents resulting from PC-VOF (TFAN) and PC-VOF (LSQR) are equivalent and smaller than the ones obtained with `interFoam`. Further, it can also be observed that for all cases obtained here with OpenFOAM®, smaller spurious currents are obtained as the mesh is refined. The results from Popinet (2009) show the smallest spurious currents. The author mentions that the initial mesh resolution of 64x64 was used. However, it is unclear which level of adaptive mesh refinement has been used, such that comparisons with present results should be made with care.



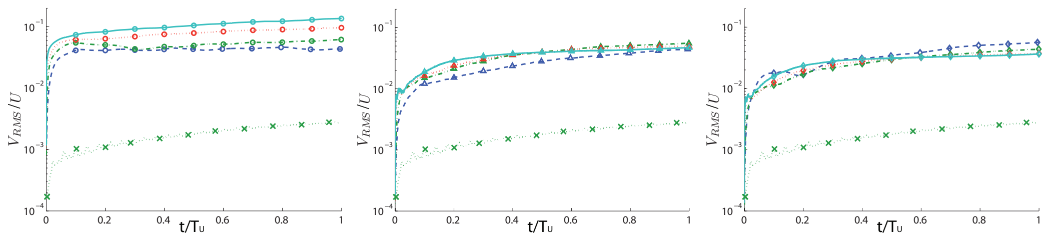
6.6(a): **La** = 120: `interFoam`, PC-VOF (TFAN) and PC-VOF (LSQR)



6.6(b): **La** = 1200: `interFoam`, PC-VOF (TFAN) and PC-VOF (LSQR)



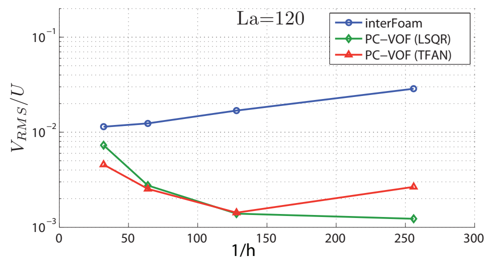
6.6(c): **La** = 12000: `interFoam`, PC-VOF (TFAN) and PC-VOF (LSQR)



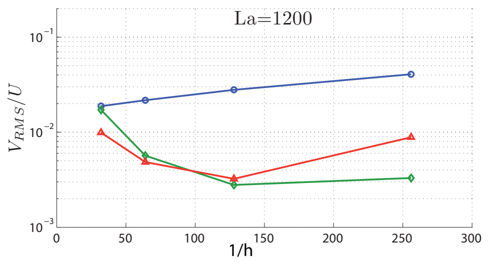
6.6(d): **La** = ∞: `interFoam`, PC-VOF (TFAN) and PC-VOF (LSQR)

Figure 6.6: V_{RMS} values for simulations performed with `interFoam`.

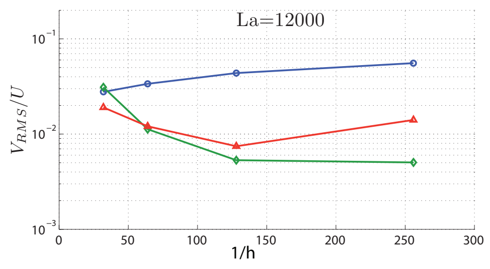
The decays of V_{RMS} with mesh refinement at the last time step are plotted at Figure 6.7 for the different Laplace numbers (**La** $= 120$, 1200, 12000 and ∞). It can be observed that `interFoam` consistently increases the RMS velocity with mesh refinement, resulting in an undesired behavior. Between PC-VOF TFAN and LSQR, LSQR presents better results for all Laplace numbers. For **La** $= 1200$, it slightly increases its RMS velocity for the finest mesh, however it is still better than TFAN, which presents slight divergence.



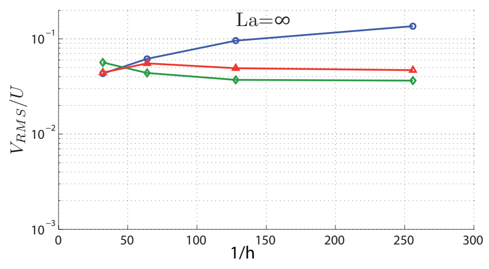
6.7(a): $La = 120$



6.7(b): $La = 1200$



6.7(c): $La = 12000$

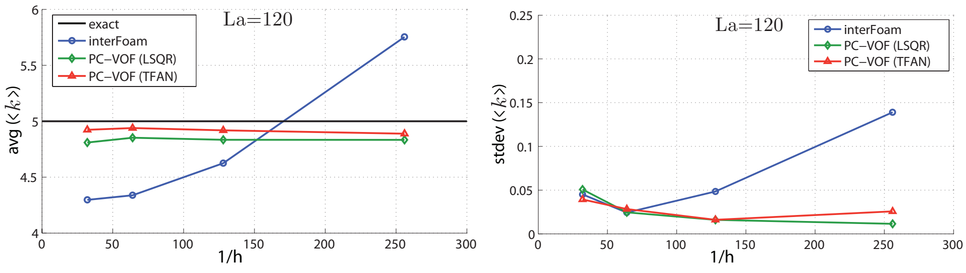


6.7(d): $La = \infty$

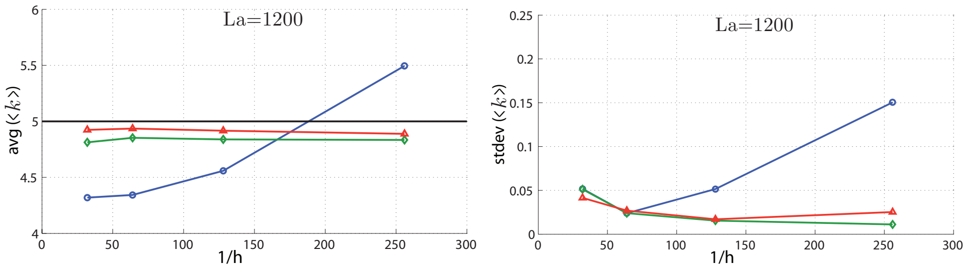
Figure 6.7: V_{RMS} mesh convergence for `interFoam` and PC-VOF with normals computed by TFAN and LSQR.

Figure 6.8 shows the average curvature avg($\langle\kappa\rangle$) and its standard deviation stdev($\langle\kappa\rangle$) obtained with `interFoam`, PC-VOF (TFAN) and PC-VOF (LSQR), for **La** $= 120$, 1200, 12000 and ∞. At each graph of avg($\langle\kappa\rangle$), the exact curvature is also plotted.

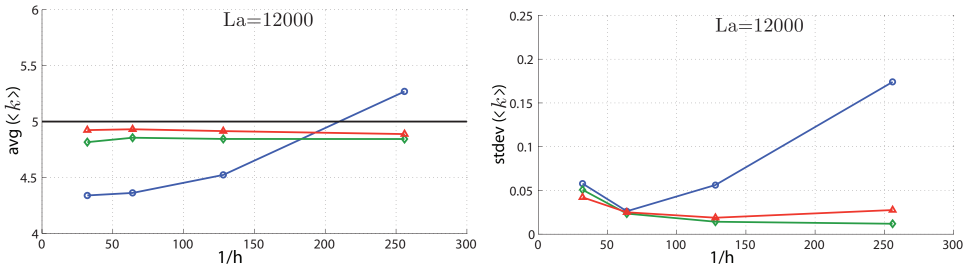
The three methods present similar behavior for the average curvature as the Laplace number increases. Average curvature in `interFoam` presents divergence with mesh resolution compared to both PC-VOF results. PC-VOF (TFAN) shows a best estimation of avg($\langle\kappa\rangle$). Besides not converging on the average curvature, `interFoam` also presents a more preeminent standard deviation compared to PC-VOF results. Between both PC-VOF alternatives, LSQR is the one that results in less spreading of the average curvature, what is desirable in a sense that makes it more stable than TFAN.



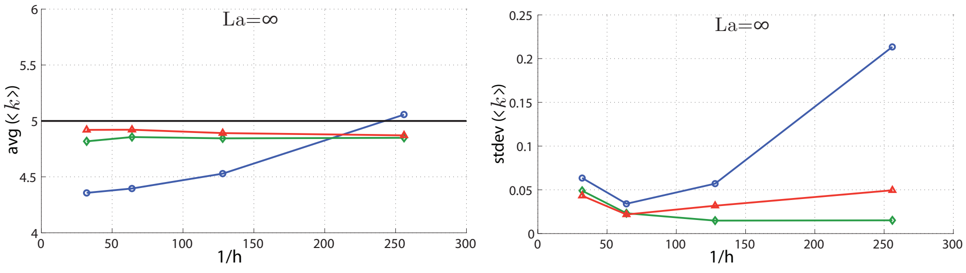
6.8(a): **La** = 120



6.8(b): **La** = 1200



6.8(c): **La** = 12000



6.8(d): **La** = ∞

Figure 6.8: Mesh convergence regarding average curvature avg($\langle\kappa\rangle$) and its standard deviation stdev($\langle\kappa\rangle$) with `interFoam` and PC-VOF with normals computed by TFAN and LSQR.

6.3 Oscillating Drop

Rayleigh (1879) derived a linear theory for cylindrical jets oscillating in the plane perpendicular to its axis. When the amplitude is small enough compared to its radius, the linear theory is valid and the frequency of oscillation ω_n, in rad/s is given by

$$\omega_n = \sqrt{\frac{\sigma(n_\omega^3 - n_\omega)}{\rho\, r_0^3}} \tag{6.7}$$

where σ is the surface tension, ρ is the jet fluid density, r_0 is the jet undisturbed radius and n_ω corresponds to the mode of oscillation.

The shape of the cross-section of the jet in polar coordinates is given by

$$r(\theta) = r_0 + r_\epsilon \cos{(n_\omega \theta)} \tag{6.8}$$

where r_ϵ is the perturbation amplitude, that should be a small quantity in comparison with r_0. Rayleigh (1879) observed that for large values of r_ϵ, the results expected from the linear theory diverged from experimental data, due to non-linear effects. Considering the jet axis direction z coordinate is much greater than r_0, one can approach this problem to 2D, on the plane perpendicular to the jet axis. This reduces the 3D jet oscillation to a 2D drop oscillation problem.

Fyfe *et al.* (1988) presents the linear theory taking into account the effects of an external fluid. The frequency, then, becomes

$$\omega_n = \sqrt{\frac{\sigma(n_\omega^3 - n_\omega)}{(\rho_d + \rho_e) r_0^3}} \tag{6.9}$$

where ρ_d is the density of the drop fluid and ρ_e is the density of the external fluid. This problem was also studied by Fuster *et al.* (2009) and by de Melo (1995) in his MSc. thesis. Here the setup described in de Melo (1995) will be followed. It consists of a 2D kerosene drop immersed in air oscillating in $n_\omega = 2$ oscillation mode. The problem properties follow:

- drop density $\rho_d = 820\ kg/m^3$,
- air density $\rho_e = 1.3\ kg/m^3$,
- interface tension $\sigma = 0.03\ N/m$,
- non viscous flow, i.e. $\mu_d = \mu_e = 0\ Pa.s$
- undisturbed drop radius $r_0 = 0.0125\ cm$.

The oscillation period τ is given by

$$\tau = \frac{2\pi}{\omega_n} \tag{6.10}$$

Considering the problem parameters above, the oscillation period value is $\tau = 0.593ms$. As the oscillation mode is $n_\omega = 2$, the drop oscillates in ellipsoidal shapes.

The drop is initialized as in de Melo (1995) as an ellipse such that the ratio between its major and minor axes is equal to 1.4 and its area A is equal to its undeformed area, i.e. the area of the circle with radius r_0. It gives

$$\frac{a}{b} = \frac{(r_0 + r_\epsilon)}{(r_0 - r_\epsilon)} = 1.4 \quad \text{and} \quad A = \pi(r_0 + r_\epsilon)(r_0 - r_\epsilon) = \pi r_0^2$$

where a and b are respectively the horizontal and vertical drop diameters. The domain consists of a square of side equal to $0.2cm$. Figure 6.9 displays the initial drop shape.

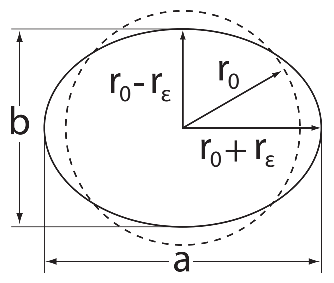


Figure 6.9: Initial drop shape.

Velocity and pressure boundary conditions at the four boundary patches were set to zero gradient.

The simulation time step Δt was limited by a stability condition taking into account the capillary spurious velocities, as discussed in Deshpande *et al.* (2012). The time step is, thus, limited by

$$\Delta t \leq \tau_\sigma = \frac{1}{2}\left[C_2\tau_\mu + \sqrt{(C_2\tau_\mu)^2 + 4C_1\tau_\rho^2}\right] \tag{6.11}$$

where C_1 and C_2, as discussed in Deshpande *et al.* (2012), are adjusted in order to control spurious currents growth in `interFoam`. They are adjusted to $C_1 = 0.01$ and $C_2 = 10$. The time scales τ_ρ and τ_μ are given by

$$\tau_\rho = \sqrt{\frac{\rho\, h^3}{\sigma}} \quad ; \quad \tau_\mu = \frac{\mu\, h}{\sigma} \tag{6.12}$$

where h is the mesh spacing and ρ was taken as the drop density. As this flow is inviscid, $\tau_\mu = 0$ and τ_σ reduces to $\tau_\sigma = \sqrt{C_1}\tau_\rho = \tau_\rho/10$.

Time evolution of drop diameters aspect ratio a/b obtained by the different solvers is compared with the exact solution in Fig. 6.10 for the finest grid refinement ($1/h = 256$). Predictions obtained by `interFoam` are compared with the solution obtained by three PC-VOF methods. Although it has been shown in section (6.1) that the Least Square approach is better than the triangulation ones, one more test was performed to confirm it. Here the normals were computed by LSQR, TriFAN and TriWEI and the points were computed by linear interpolation.

Figure 6.11 illustrates the effect of mesh resolution on the time evolution of the ratio a/b. One may observe, once again, superior results for PC-VOF with normal computations by least-squares planes. Both the amplitude and period of oscillation converge with mesh refinement. Simulations with PC-VOF (TFAN) show convergence for oscillation period, however the amplitude increases resulting in unfeasible results. Simulations with PC-VOF (TWEI) shows increasing amplitudes, also resulting in unfeasible results. Simulations with standard `interFoam` seem to improve amplitude with mesh refinement, but the oscillation period converges to a wrong value, greater than the exact solution.

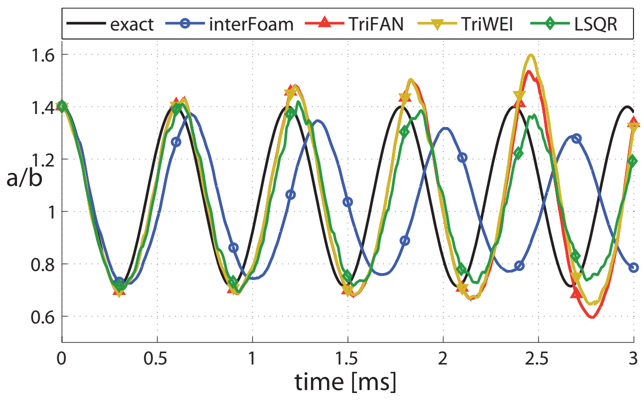
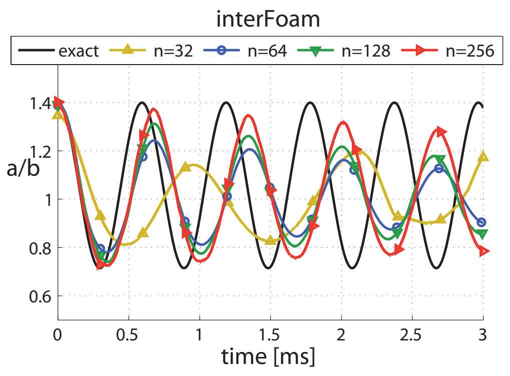
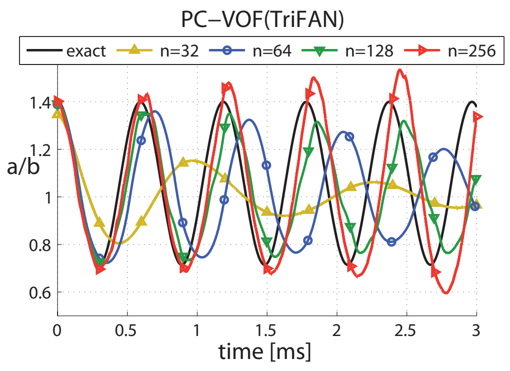


Figure 6.10: Drop diameter aspect ratio for simulations performed by `interFoam` and PC-VOF with normals computed by TFAN, TWEI and LSQR for the finest grid resolution.

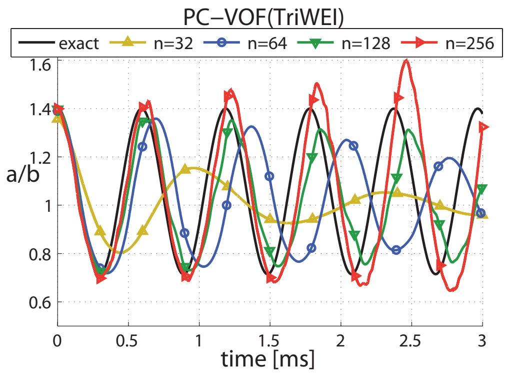
Results for both triangulation methods become unfeasible once their amplitudes keep increasing at each oscillation period. This is due to oscillation in local curvatures computed by these methods, that arise due to noise in the sampling points positions. Once the advection of α field is performed, noise in the sampling points can also increase, resulting in unfeasible curvatures in both TRIFAN and TRIWEI triangulation methods. LSQR method is less sensible to the point cloud noise once it makes a sort of averaging among many neighboring points. Besides the smoothing resulted from the LSQR method, the surface tension forces computed at the sampling points are also smoothed during the procedure of projection from the point cloud to the Eulerian grid, described in Section 5.7. In this way, the method combination chosen to be the default in PC-VOF is linear computation of sampling points and least-squares computation of normals. That is, unless otherwise noticed, PC-VOF stands for PC-VOF (Linear LSQR).



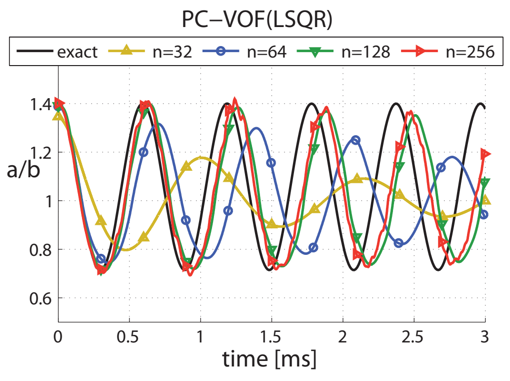
6.11(a): `interFoam`



6.11(b): PC-VOF (TFAN)



6.11(c): PC-VOF (TWEI)



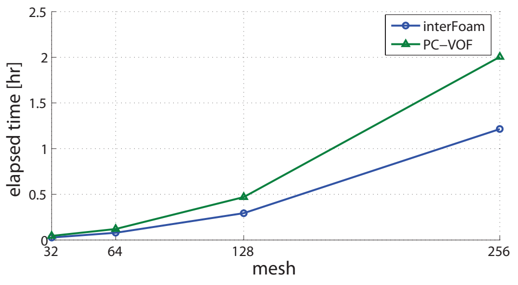
6.11(d): PC-VOF (LSQR)

Figure 6.11: Drop diameter aspect ratio for simulations performed by `interFoam` and PC-VOF with normals computed by TFAN, TWEI and LSQR.

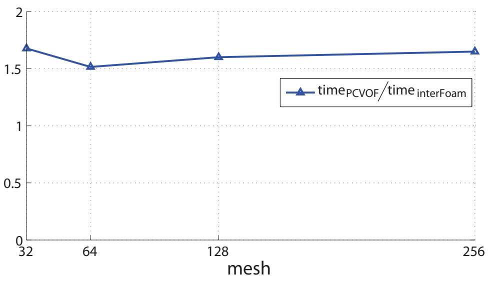
6.4 Execution Time

In order to compare PC-VOF execution time against `interFoam`, the oscillating drop case presented in Section 6.3 was employed. This section provides a comparison between execution time with `interFoam` and with PC-VOF with point sampling by linear interpolation and normal computation by least-squares plane fitting (LSQR). Simulations were performed in serial processing in Ubuntu 14.04.4 LTS with CPU Intel®Core™ i7-5960X @ 3.00GHz. Four mesh refinements were employed, the same presented in Sec. 6.3, i.e. $1/h = 32, 64, 128$ and 256. The time step was fixed in $\Delta t = 10^{-7}s$ for all mesh refinements and the simulations were carried out until the final time $t_f = 10^{-3}s$.

Figure 6.12(a) displays the total time taken by `interFoam` and PC-VOF to perform the simulations for each mesh refinement in hours and Fig.6.12(b) displays the time taken by PC-VOF divided by the time taken by `interFoam`.



6.12(a): Elapsed time.



6.12(b): Time fraction.

Figure 6.12: Execution time.

Time ratio results in Fig.6.12(b) show that PC-VOF takes an average of 65% more time than standard `interFoam` for this problem.

6.5
Square Interface

This section studies a square interface that, due to the interfacial surface force, has its shape evolved to a circle. The environmental fluid is *fluid 1* and the fluid inside the square is *fluid 2*. Densities and viscosities were set to unity ($\rho_1 = \rho_2 = \mu_1 = \mu_2 = 1$) and gravity set to zero ($\overrightarrow{g} = [0,0,0]^T$). The surface tension coefficient was tested for four values, $\sigma = 100$, 10, 1 and 0.1. The domain size is 10×10 and the interface is initialized as a square of size equal to 4, as depicted in Fig.6.13.

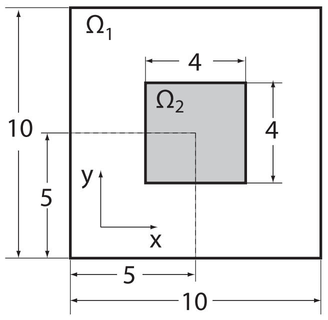


Figure 6.13: Square initial configuration.

The simulation was run until steady state was achieved. Figure 6.14 displays the initial and final interface shapes for PC-VOF (Linear LSQR) and `interFoam` with regular and irregular (triangular) meshes, for $\sigma = 10$. The employed regular mesh was 80×80 and the irregular/unstructured one was equivalent in the number of cells (triangular cells), i.e. with approximately 1600 triangles. Time step was limited by a maximum Courant number $\mathbf{Co}_{max} = 0.1$. Boundary conditions for the volume fraction, velocity and pressure were set to zero gradient on all boundary patches.

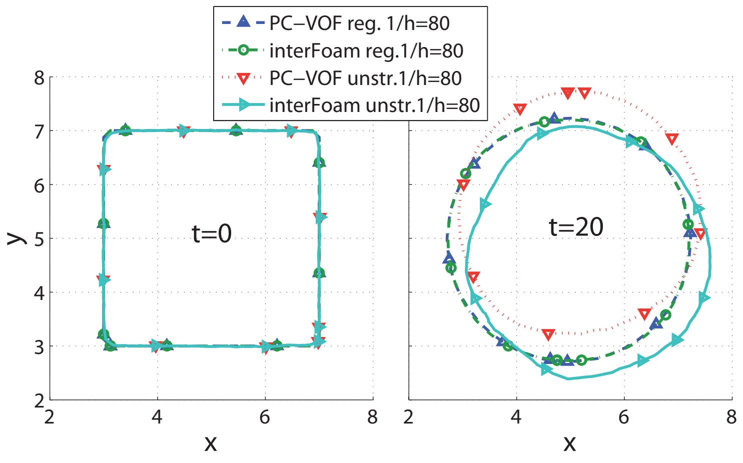
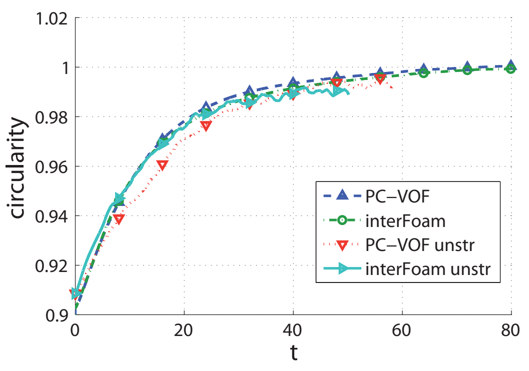


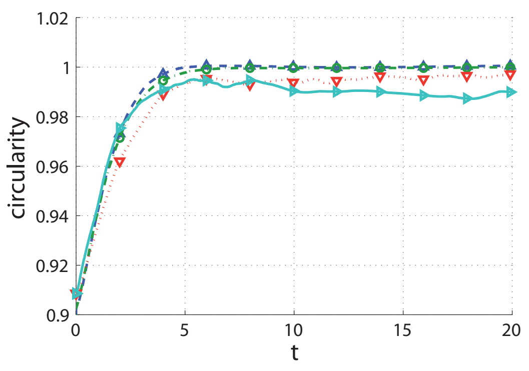
Figure 6.14: Square initial and final shapes for $\sigma = 10$.

It can be observed that in unstructured mesh `interFoam` produces an interface that deviates from a circle. It can be perceived visually. In constrast, PC-VOF tends to a circular interface for both structured and unstructured meshes.

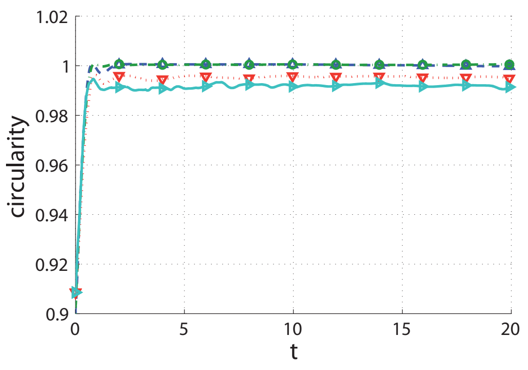
Figure 6.15 shows the evolution of circularity for various values of the surface tension coefficient. It is defined as the ratio between the perimeter of the area-equivalent circle and the perimeter of the interface. For a perfectly circular interface, the circularity is equal to unity.



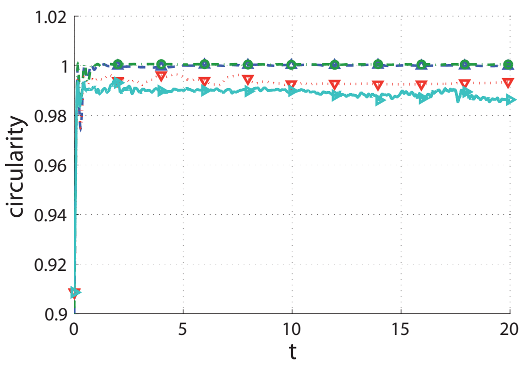
6.15(a): $\sigma = 0.1$



6.15(b): $\sigma = 1$



6.15(c): $\sigma = 10$

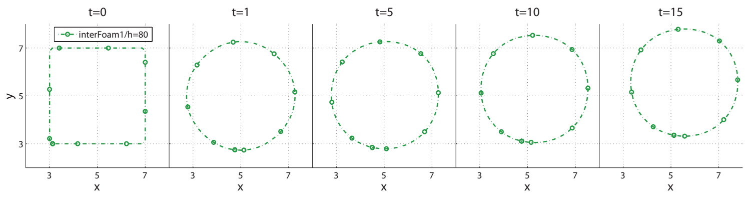


6.15(d): $\sigma = 100$

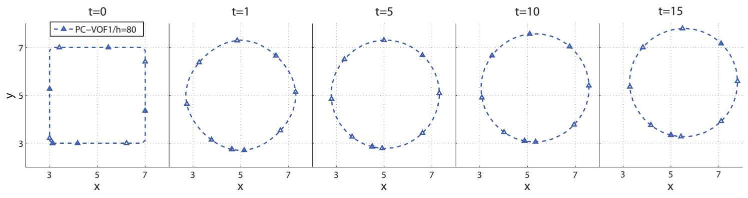
Figure 6.15: Circularity evolution for different values of surface tension coefficient σ.

For $\sigma = 0.1$, it is necessary a longer simulation time to achieve steady state. Besides, for the unstructured mesh, the interface derived away from the domain during the simulation. Therefore, the circularities are displayed only up to this point for unstructured meshes. It can be observed that both PC-VOF and `interFoam` recover the unit circularity for regular mesh. In unstructured mesh, PC-VOF remains closer to unity than `interFoam`.

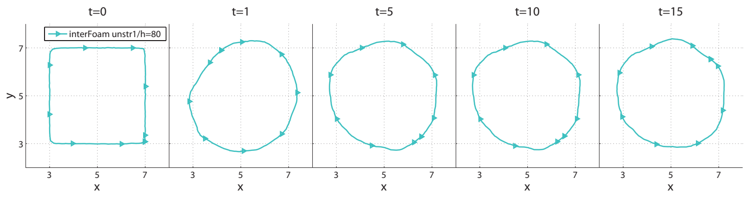
For $\sigma = 100$, Fig.6.16 presents the evolution of the interface shape for each situation for $t = 0, 1, 5, 10$ and 15. It can be observed that for regular mesh, both PC-VOF and `interFoam` were able to obtain a circle as a converged solution. The final shape for `interFoam` in unstructured mesh



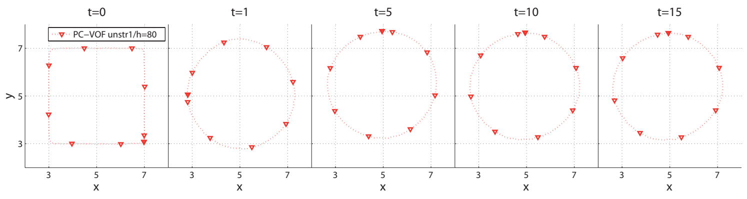
6.16(a): `interFoam` reg.



6.16(b): PC-VOF reg.



6.16(c): `interFoam` unstr.



6.16(d): PC-VOF unstr.

Figure 6.16: Shape evolution of square for $\sigma = 100$.

deviates from a circle, while a nice circle was obtained with PC-VOF. These results demonstrate that the new interface treatment improves the curvature accuracy, and handles unstructured meshes without problems.

7 Bubble in a liquid column

Hysing *et al.* (2009) proposed two configuration cases of a single rising two dimensional bubble for quantitative validation of incompressible multiphase flow solvers. Reference results generated by three different codes were provided for comparison. Adelsberger *et al.* (2014) later extended the benchmark to three dimensional cases and provided results also for three different solvers, including a previous version of `interFoam`. Section 7.1 presents a comparison of `interFoam` and PC-VOF against the two dimensional benchmark data and Section 7.3 deals with the three dimensional configuration.

7.1 Two-Dimensional Bubble in a Liquid Column

Hysing *et al.* (2009) proposed a benchmark configuration for a two-dimensional bubble rising in a liquid column. Codes by three independent research groups were tested in two different test cases. The first one involved slight bubble deformations, and all codes presented good agreement with each other. In the second test case, a higher density ratio was chosen and the different codes agreed well up to the point of breakup, so that reference values could be established up to this point.

Klostermann *et al.* (2013) employed this benchmark configuration to evaluate the standard `interFoam` solver implemented in OpenFOAM® version 1.5.1 through version 2.1.0. This section is devoted to employ the very same benchmark cases to evaluate `interFoam` in its version 2.3.0 and PC-VOF, developed on top of it. The methods for normal computation and point cloud sampling in PC-VOF are, respectively, least-squares planes (Sec.5.3.2) and linear interpolation of α field (Sec.5.2.1). These are taken as the default methods in PC-VOF in this work. Some of the results presented in this section were also presented in Kassar *et al.* (2016).

The initial configuration is depicted in Fig. 7.1. The liquid column is considered as *fluid 1* and *fluid 2* is the bubble. The dimensions may be taken as in meters and all other quantities are also in the SI – International System of Units. Regarding the boundary conditions, the left and right patches are assigned with slip conditions and the bottom and top walls are assigned with zero velocity.

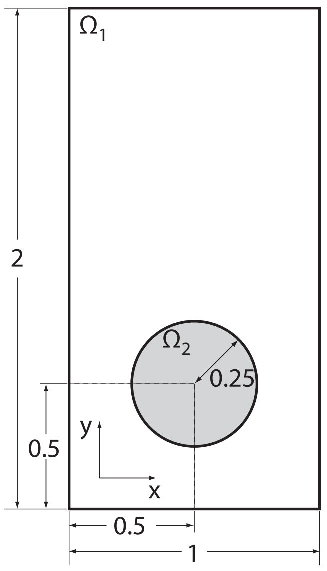


Figure 7.1: Bubble initial configuration.

The dimensionless numbers that govern the problem are: Reynolds number **Re**, Eötvös number (**Eo**), capillary number (**Ca**) and Weber number (**We**). The liquid, *fluid 1*, is the reference fluid, and it is identified by the subscript 1, while subscript 2 refers to the bubble.

The Reynolds number is the ratio of inertial to viscous forces and is defined by

$$\mathbf{Re} = \frac{\rho_1 V_t\ d}{\mu_1} \tag{7.1}$$

where d is the bubble diameter and $V_t = \sqrt{g\ d}$ is its terminal rising velocity, ρ_1 and μ_1 are the density and viscosity of *fluid 1*.

The Eötvös number quantifies the ratio between body forces and surface tension forces and is given by

$$\mathbf{Eo} = \frac{\Delta\rho\ g\ d^2}{\sigma} = \frac{(\rho_1 - \rho_2)\ g\ d^2}{\sigma} \tag{7.2}$$

where σ is the interface tension coefficient and g is gravity magnitude (towards negative y). The Weber number **We** corresponds to the ratio of inertial forces to surface tension effects, while the capillary number **Ca** gives the ratio between viscous forces and surface tension effects, and are given by

$$\mathbf{We} = \frac{\rho_1 V_t^2 d}{\sigma} \tag{7.3}$$

$$\mathbf{Ca} = \frac{\mu_1 V_t}{\sigma} = \frac{\mathbf{We}}{\mathbf{Re}} \tag{7.4}$$

Two test cases were proposed by Hysing *et al.* (2009), denoted by TC1 and TC2. In TC1, the surface tension plays an important role on the flow and in TC2, the buoyancy is the dominant force. Values of fluid properties are given in the Table 7.1. In the same table, the dimensionless numbers that govern the problem are included.

Case	ρ_1	ρ_2	μ_1	μ_2	σ	g	V_t	**Re**	**Eo**	**Ca**	**We**
TC1	1000	100	10	1.0	24.50	0.98	0.7	35	9	0.286	10
TC2	1000	1	10	0.1	1.96	0.98	0.7	35	124.9	3.571	125

Table 7.1: Test cases parameters.

By observing the values in Table 7.1, one may notice that TC1 is a case in which surface tension plays a significant role, while TC2 is more influenced by gravitational and inertial forces.

Such as proposed by Hysing *et al.* (2009), some quantities were monitored in order to perform quantitative comparison with benchmark data. Namely: circularity C, center of mass position $\overrightarrow{x_c}$ and mean rise velocity $\overrightarrow{V_c}$.

Circularity C, Eq.(7.5), is given by the ratio between the perimeter of the area-equivalent circle with radius r_{b0} and the perimeter of the bubble P_b. It gives somehow a measurement for the bubble shape, enabling quantitative comparison. For a perfectly circular bubble, the circularity is equal to unity and decreases as the bubble is deformed. It is given by

$$C = \frac{2\ \pi\ r_{b0}}{P_b} \tag{7.5}$$

The center of mass $\overrightarrow{x_c}$ is given by

$$\overrightarrow{x_c} = \frac{\int_{\Omega_2} \overrightarrow{x} d\ \forall}{\int_{\Omega_2} d\ \forall} \tag{7.6}$$

where Ω_2 denotes the region that the bubble occupies. It is identified by cells for which $\alpha \geq 0.5$

The mean rise velocity is given by

$$\overrightarrow{V_c} = \frac{\int_{\Omega_2} \overrightarrow{V} d\forall}{\int_{\Omega_2} d\forall}. \tag{7.7}$$

Four grid refinements $1/h$ with regular quadrangular meshes are used: $1/h = 40, 80, 160$ and 320, where h is the grid spacing in x and y directions, i.e. $h = \Delta x = \Delta y$. Besides, for each quad mesh, a triangular mesh was also tested with equivalent number of cells. The triangular meshes are actually composed by triangular prisms, i.e. triangles extruded in the z direction (as in OpenFOAM® 2D domains are a special case of 3D) and were generated by GMSH (Geuzaine & Remacle, 2009). And example of the quad. and tri. grids for $1/h = 20$ is presented in Fig.7.2.

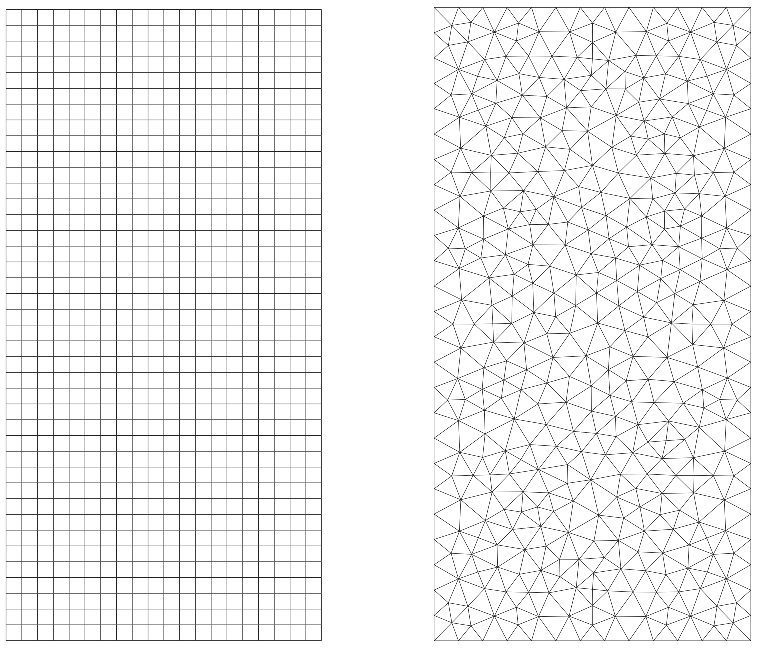

7.2(a): Quads 7.2(b): Triangles

Figure 7.2: Quadrangular mesh and equivalent triangular for $1/h = 20$.

Before the simulation of the rising bubble takes place, an initialization simulation is performed with gravity set to zero. This simulation results in pressure, α and velocity fields under static equilibrium. Besides, it provides results of spurious currents and pressure jump value across the interface. These values were obtained, checked against theoretical results and are presented in the following discussion.

7.1.1 Zero Gravity Condition

The volume fraction field α is first initialized to meet the configuration of Fig.7.1. The method of field initialization presented in Sec.4.3 was employed. For this situation, the value of gravity acceleration is set to $[0\ 0\ 0]^T$.

The pressure jump across the interface is given by the Laplace-Young equation

$$\Delta p = \sigma\kappa \tag{7.8}$$

In the case of a 2D circular bubble, the curvature is given by $1/r_0$, where r_0 is the bubble radius. So the pressure jump is given by $\Delta p = \sigma/r_0$.

Test Case 1

In TC1, the theoretical pressure jump is given by $\Delta p = 24.5/0.25 = 98Pa$. The simulation time was set equal to 3 s, as in Klostermann *et al.* (2013). It is enough time to reach steady condition in pressure jump and in spurious currents. Here, the pressure jump Δp for the circular bubble is computed numerically as

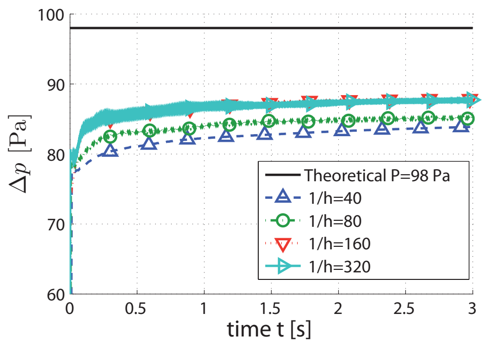
$$\Delta p = \langle p_2 \rangle - \langle p_1 \rangle \tag{7.9}$$

were $\langle p_1 \rangle$ and $\langle p_2 \rangle$ are respectively the average pressure in the continuous phase (*fluid 1*) and the average pressure inside the bubble (*fluid 2*).

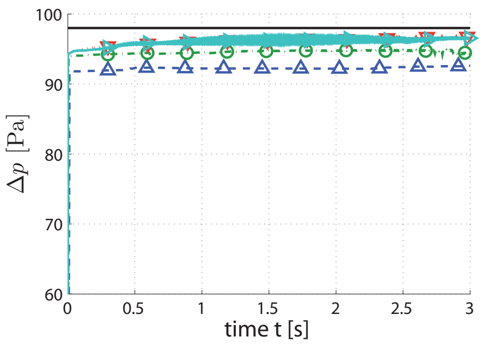
$$\langle p_1 \rangle = \frac{\int_{\hat{\Omega}_1} p \, d\forall}{\int_{\hat{\Omega}_1} d\forall} \quad , \quad \langle p_2 \rangle = \frac{\int_{\hat{\Omega}_2} p \, d\forall}{\int_{\hat{\Omega}_2} d\forall}. \tag{7.10}$$

where $\hat{\Omega}_1$ is considered to be region for which $\alpha < 0.01$ and $\hat{\Omega}_2$ is identified by cells for which $\alpha \geq 0.99$. This is done to avoid integrating over interface cells region.

Figure 7.3 displays the evolution of Δp with simulation time, respectively, for `interFoam` and PC-VOF with regular quadrangular meshes. Mesh independent results are obtained for both solvers. One may notice that superior results are reached with PC-VOF. The pressure jump with PC-VOF converges to a value much closer to the expected value according to Young-Laplace Eq.(7.8) than `interFoam`.



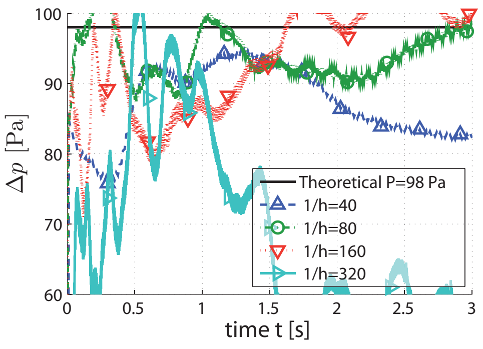
7.3(a): `interFoam`



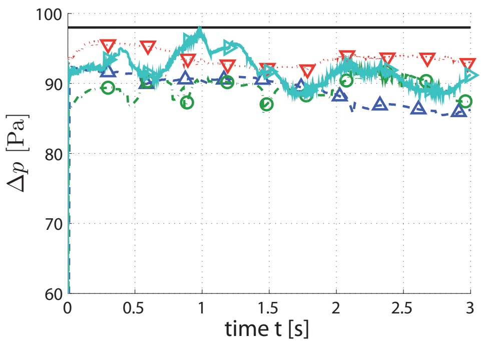
7.3(b): PC-VOF

Figure 7.3: Pressure jump in TC1 for regular quads meshes.

Figure 7.4 presents the evolution of Δp with time respectively for `interFoam` and PC-VOF with triangular meshes. It can be seen that for triangular meshes, the results with PC-VOF present oscillations, but these values converge to a slightly more accurate pressure jump than `interFoam` predicted even in regular quads meshes. Simulations performed with `interFoam` in triangular meshes present oscillations that increase with mesh refinement. Note also that with triangular meshes, the `interFoam` simulated bubble lost its circular shape in these simulations.



7.4(a): `interFoam`



7.4(b): PC-VOF

Figure 7.4: Pressure jump in TC1 for triangular meshes.

Convergence with mesh refinement for Δp values at the final simulation time $t = 3\ s$ is presented in Fig.7.5 for the four situations: `interFoam` and PC-VOF in quadrangular and triangular meshes.

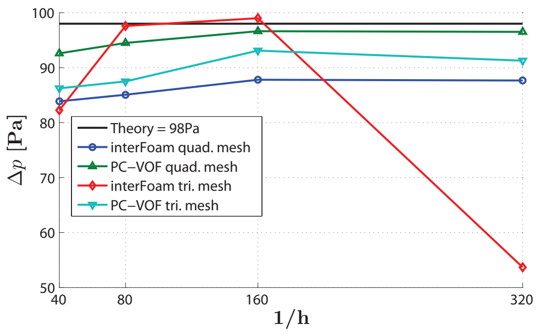


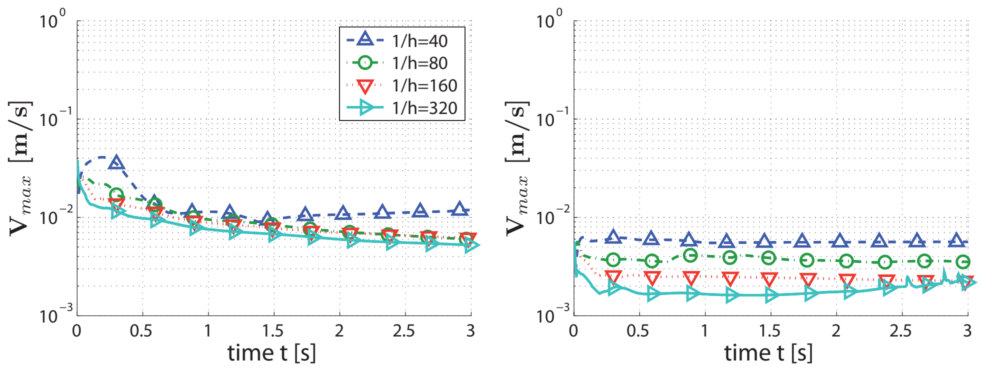
Figure 7.5: Pressure jump convergence in TC1.

Notice that PC-VOF converges in both quadrangular and triangular meshes, while `interFoam` only converges in quadrangular meshes. Besides, PC-VOF in triangular meshes converges to a value slightly closer to the theoretical result than `interFoam` in quads meshes.

Concerning spurious currents (V_{max}), Fig.7.6 displays its evolution with simulation time for, respectively, `interFoam` and PC-VOF with quads meshes. The ordinate scales were kept the same to facilitate comparison. The reader may observe considerable reduction in the spurious currents with PC-VOF method. It can also be noted that, as the mesh is refined, `interFoam` reaches a constant value of the maximum spurious velocity, while, with PC-VOF, a reduction of its value is still being obtained.

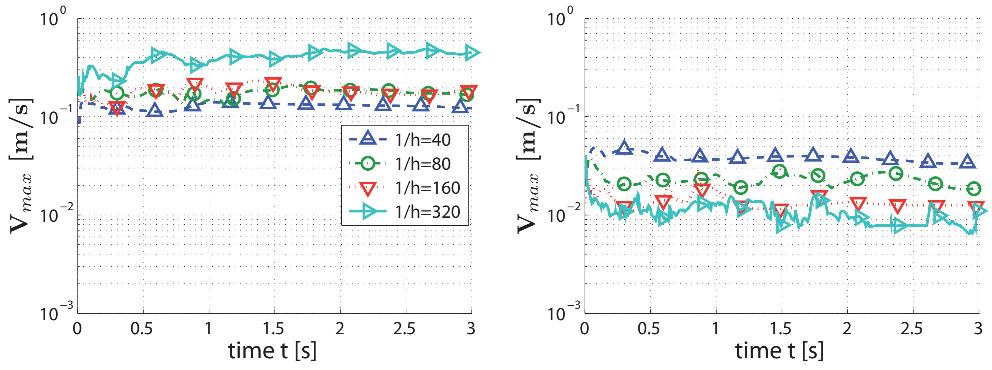
Figure 7.7 displays the evolution of spurious currents for triangular meshes. Notice that ordinate scales in (a) and (b) figures do not match due to differences in order of magnitude. However, the scale in Fig.7.7(b) is the same as in Fig.7.6.

On the one hand, spurious currents with `interFoam` in triangular grids do not show convergence with grid refinement. On the other hand, in PC-VOF with triangular grids, oscillations are found, specially for the finest grids, but the values lie in range comparable to the situations with quads grids and better convergence is presented.



7.6(a): `interFoam` 7.6(b): PC-VOF

Figure 7.6: Spurious currents in TC1 for quadrangular meshes.



7.7(a): `interFoam` 7.7(b): PC-VOF

Figure 7.7: Spurious currents in TC1 for triangular meshes.

Figure 7.8 depicts the mesh convergence in spurious currents for the four situations. The reader should notice that PC-VOF method presents a convergence that decreases monotonically for the tested quads meshes while `interFoam` diverges for triangular meshes.

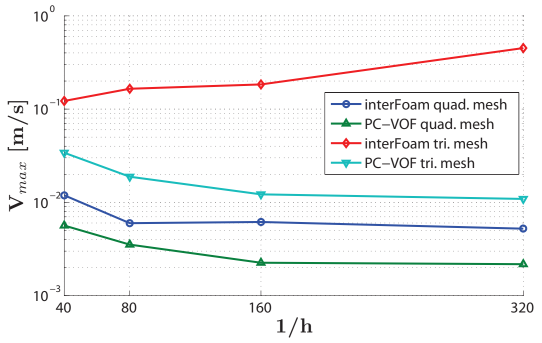
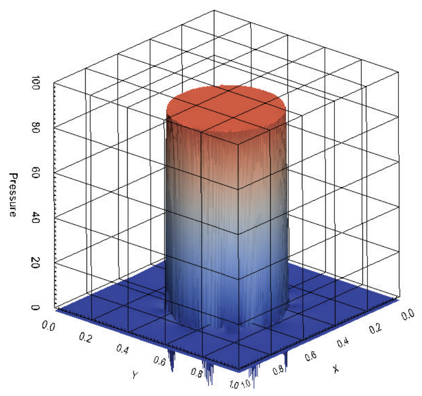
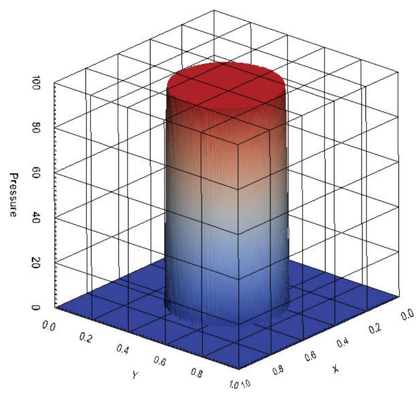


Figure 7.8: Spurious currents convergence with mesh resolution in TC1.

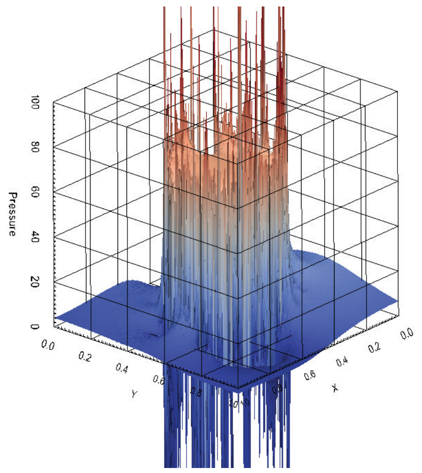
The shapes of the pressure fields at the last instant of simulation ($t = 3\ s$) are presented for each case for the finest mesh, i.e. $1/h = 320$. Figures 7.9 and 7.10 show, respectively, an isometric view and a lateral view of the pressure fields. One may notice that `interFoam` presents ripples on the pressure across the interface. For triangular mesh, the ripples grow to a higher degree. On the other hand, PC-VOF presents reduced ripples in the triangular mesh and neglected ripples in quads mesh.



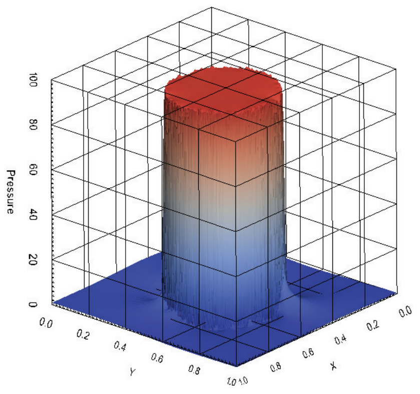
7.9(a): `interFoam` quad.mesh



7.9(b): PC-VOF quad.mesh

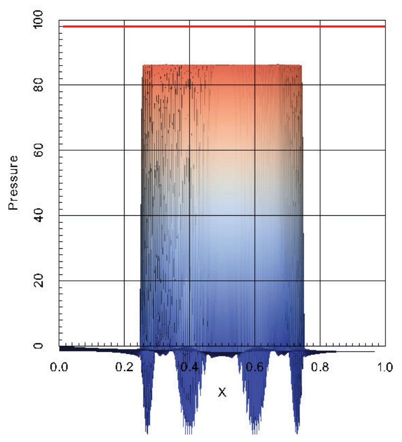


7.9(c): `interFoam` tri.mesh

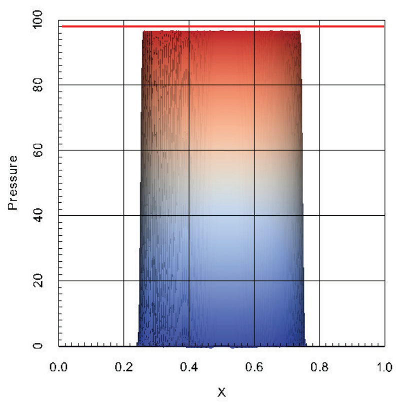


7.9(d): PC-VOF tri.mesh

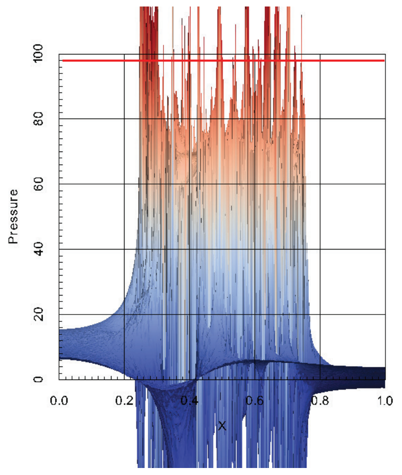
Figure 7.9: Pressure fields for TC1 test case at $t = 3\ s$. Exact value is $\Delta p = 98 Pa$.



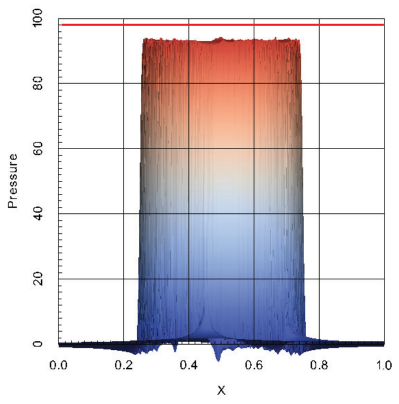
7.10(a): `interFoam` quad.mesh



7.10(b): PC-VOF quad.mesh



7.10(c): `interFoam` tri.mesh



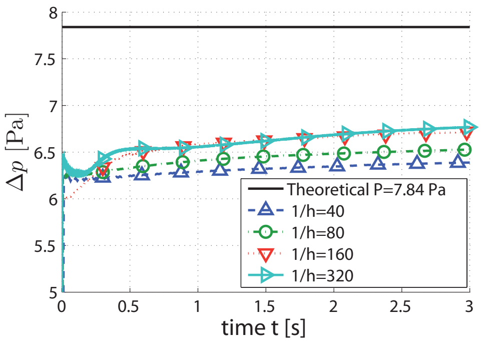
7.10(d): PC-VOF tri.mesh

Figure 7.10: Pressure fields for TC1 test case at $t = 3\ s$. Exact value is $\Delta p = 98 Pa$.

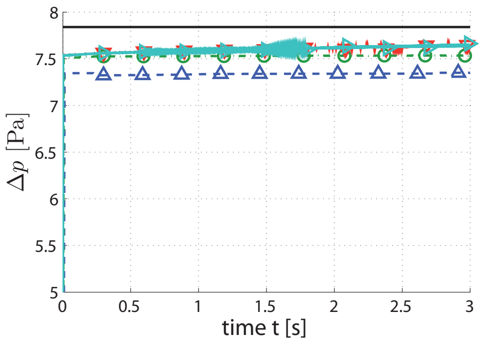
Test Case 2

In test case 2, interfacial tension forces play a less significant role than in test case 1. The same results presented for TC1 are presented for TC2 without gravity, as well. The pressure jump in this case is given by $\Delta p = 1.96/0.25 = 7.84\ Pa$

Figure 7.11 displays the evolution of the pressure jump with both methods in quads meshes, while the pressure jump for triangular meshes is depicted in Fig.7.12.

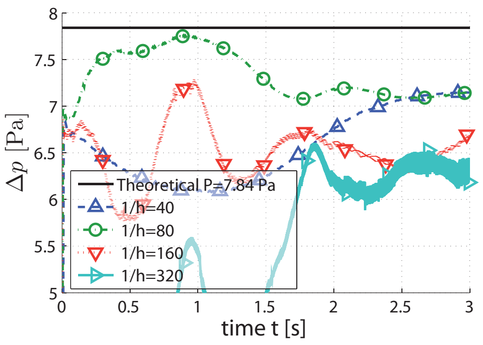


7.11(a): `interFoam`

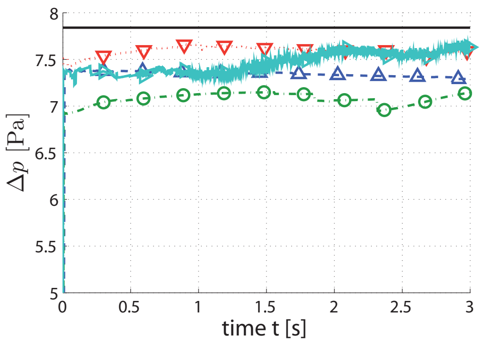


7.11(b): PC-VOF

Figure 7.11: Pressure jump in TC2 for structured quads meshes.



7.12(a): `interFoam`



7.12(b): PC-VOF

Figure 7.12: Pressure jump in TC2 for triangular meshes.

The reader may notice that best agreements with Laplace-Young Eq.(7.8) are obtained by PC-VOF for either quadrangular and triangular meshes. It can also be seen that for all PC-VOF cases, as the mesh is refined, the predicted value approaches the exact solution, reaching a mesh independent solution. The results obtained with the triangular meshes are more unstable, however are still quite superior than `interFoam`. `interFoam` not only presents larger errors, but also, even in quads meshes, produces instability on the solutions. The results deteriorate significantly when triangular mesh is employed with `interFoam`.

To better visualize the effect of mesh refinement in the obtained solutions, the convergence of pressure jump for the four configurations is displayed in Fig. 7.13. The values for these plots were acquired at the corresponding plots at time = 3 s. Analyzing the figure, it is clear that PC-VOF (with both quadrangular and triangular meshes) shows superior results in Δp than `interFoam` and `interFoam` diverges with triangular meshes.

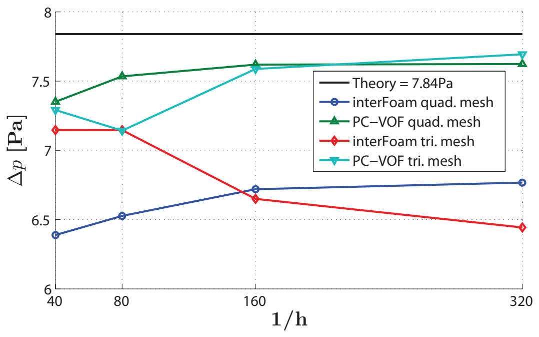
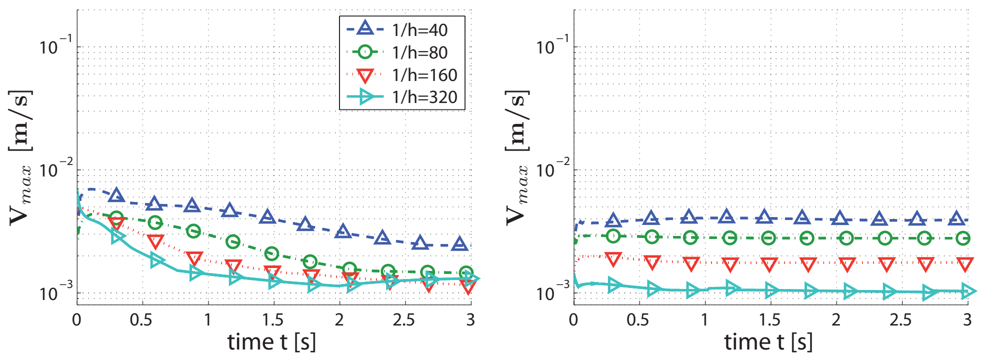


Figure 7.13: Pressure jump convergence in TC2.

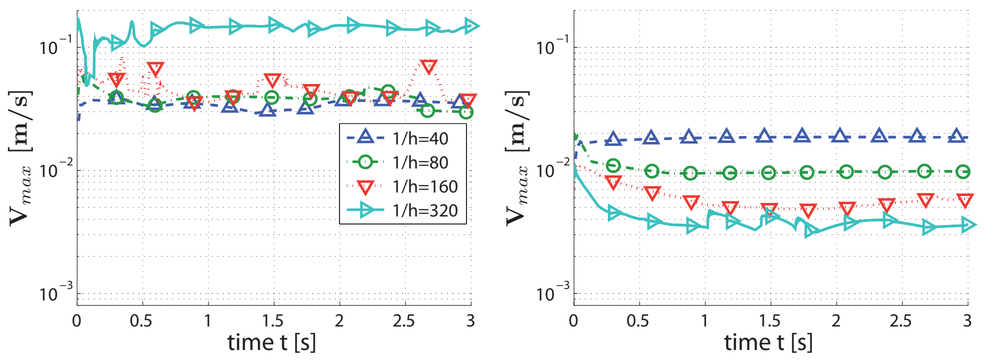
Spurious currents evolutions during simulations with quads meshes are presented in Fig.7.14 while with triangular meshes in Fig.7.15. In this test case, in which surface tension play a less significant role, `interFoam` and PC-VOF, both with quads meshes, provide spurious currents of the same order. As the mesh is refined, the spurious currents are reduced with PC-VOF, while it becomes mesh independent with `interFoam`. For this surface tension dominated flow, the spurious currents of triangular PC-VOF are much lower than for `interFoam`. Note that for the finest mesh, `interFoam` spurious currents are increased.



7.14(a): `interFoam` 7.14(b): PC-VOF

Figure 7.14: Spurious currents in TC2 for quadrangular meshes.

Figure 7.16 shows the mesh dependence of spurious currents for the four situations. Once again, PC-VOF shows monotonicity on spurious currents



7.15(a): `interFoam` 7.15(b): PC-VOF

Figure 7.15: Spurious currents in TC2 for triangular meshes.

decay with mesh refinement. For triangular meshes, `interFoam` diverges for the finest grid.

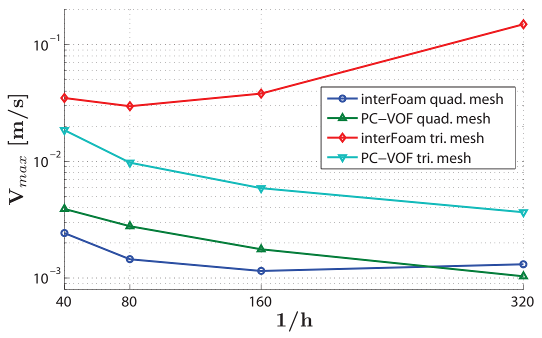
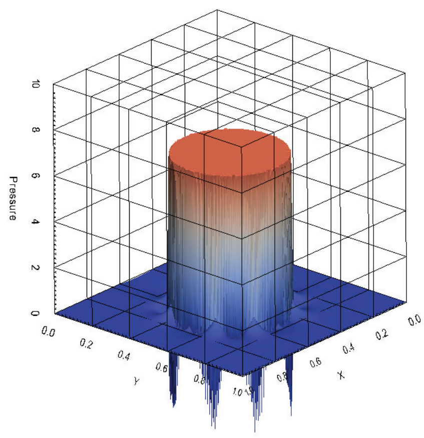
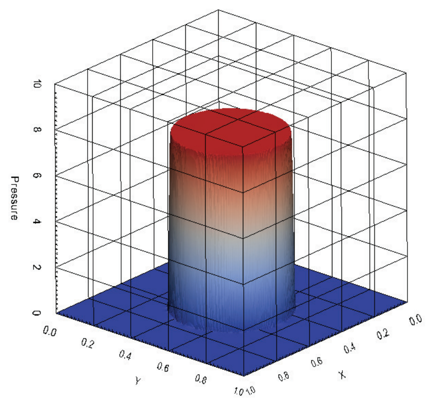


Figure 7.16: Spurious currents convergence with mesh resolution in TC2.

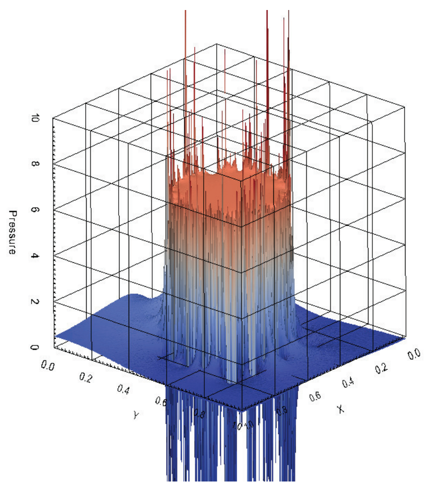
The shapes of the pressure fields at the last instant of simulation time ($t = 3\ s$) are presented for each case for the finest mesh, i.e. $1/h = 320$. Figures 7.17 and 7.18 show, respectively, an isometric view and a lateral view of the pressure fields. One may notice that the results obtained with `interFoam`, such as happened in TC1, presents ripples on the pressure across the interface. For the triangular mesh, the ripples grow to a higher degree, as well. On the other hand, PC-VOF presents reduced ripples in the triangular mesh and no ripples in the quadrangular mesh.



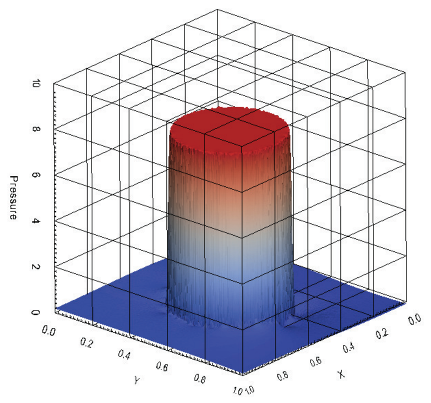
7.17(a): `interFoam` quad.mesh



7.17(b): PC-VOF quad.mesh

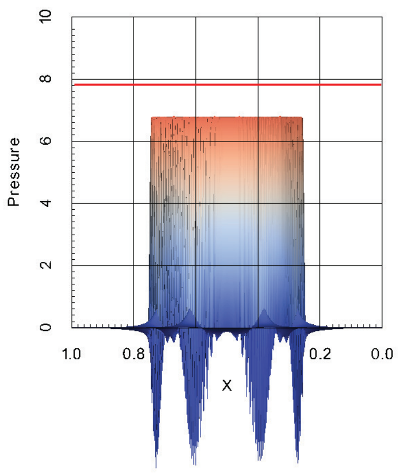


7.17(c): `interFoam` tri.mesh

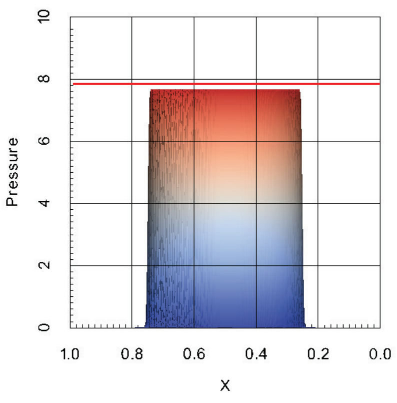


7.17(d): PC-VOF tri.mesh

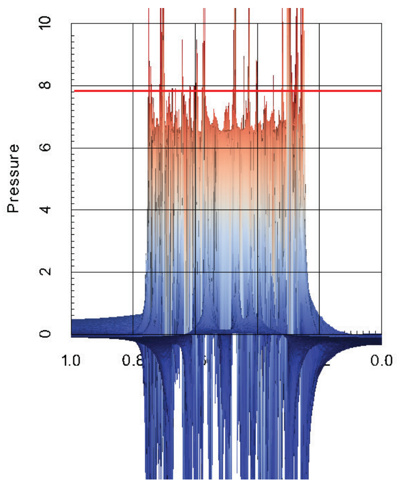
Figure 7.17: Pressure fields for TC2 test case at $t = 3\ s$. Exact value is $\Delta p = 7.84\ Pa$.



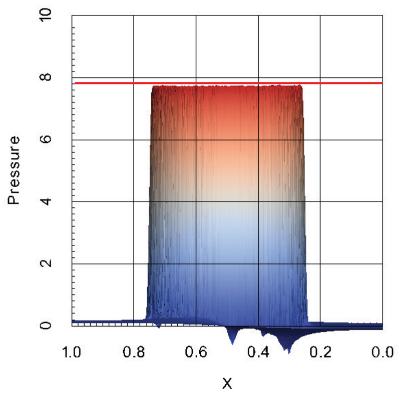
7.18(a): `interFoam` quad.mesh



7.18(b): PC-VOF quad.mesh



7.18(c): `interFoam` tri.mesh



7.18(d): PC-VOF tri.mesh

Figure 7.18: Pressure fields for TC2 test case at $t = 3\ s$. Exact value is $\Delta p = 7.84\ Pa$.

7.1.2 Rising Condition

In the rising condition, the gravity is activated and set to $\overrightarrow{g} = [0 \quad -0.98]^T$ for both test cases. Besides proposing the benchmark test cases TC1 and TC2, Hysing *et al.* (2009) also provide results for both cases simulated by three different codes:

– TP2D (Turek, 1999)
– FreeLIFE (Parolini & Burman, 2005)
– MooNMD (John & Matthies, 2004; Ganesan *et al.*, 2007)

All codes are based on the finite element method. The first two employ an Eulerian Level Set method, while the third employs an arbitrary Lagrangian-Eulerian moving grid approach. The three agree well in results and form a benchmark set of data to evaluate other solvers. The rising condition was simulated for a period of 3 s of time, such as in Klostermann *et al.* (2013) and Hysing *et al.* (2009) and results for rising velocity, center of mass vertical position and circularity are compared to benchmark data and presented bellow.

Test Case 1

The bubble shape evolution with time steps was obtained for each solver – `interFoam` and PC-VOF– for quadrangular and triangular meshes. Figure (7.19) presents the bubble shape for TC1 simulated with `interFoam` in quadrangular meshes for four different time steps: $t = 0,\ 1,\ 2$ and $3\ s$. At $t = 3\ s$, benchmark data is available and presented for comparison. Following the same procedure, Figs.7.20, 7.21 and 7.22 present results respectively for PC-VOF in quadrangular meshes, `interFoam` in triangular meshes and PC-VOF in tri. meshes.

Analyzing the time evolution of the flow, it can be observed that, as the bubble starts moving upward, it begins to deform. A depression is formed in its lower part, and its width increases. Similar solution was obtained with both solver for quadrangular meshes. For the triangular case, PC-VOF was also able to capture the bubble behavior as time evolves. On the other hand, by examining Fig. 7.21 for `interFoam` triangular, a deterioration of the bubble shape occurs as time evolves, and as the mesh is refined.

For the final time instant $t = 3\ s$, the present results can be compared with the three benchmark data. One may observe superior results for PC-VOF, compared to `interFoam`, with respect to the final position and shape of the bubble. For the quadrangular mesh, a slight improvement is observed, while for the triangular mesh, the improvement on the solution obtained with PC-VOF

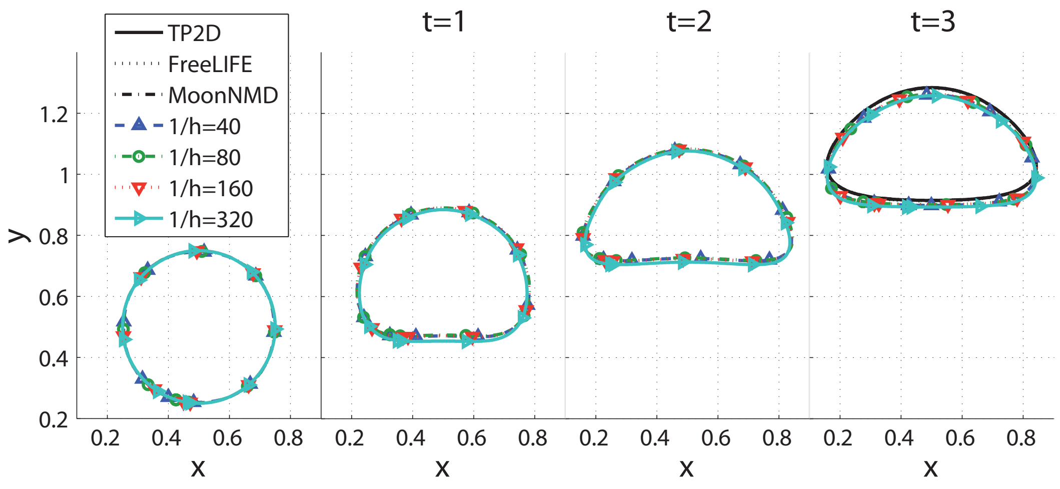


Figure 7.19: TC1 bubble shapes with `interFoam` in quadrangular meshes.

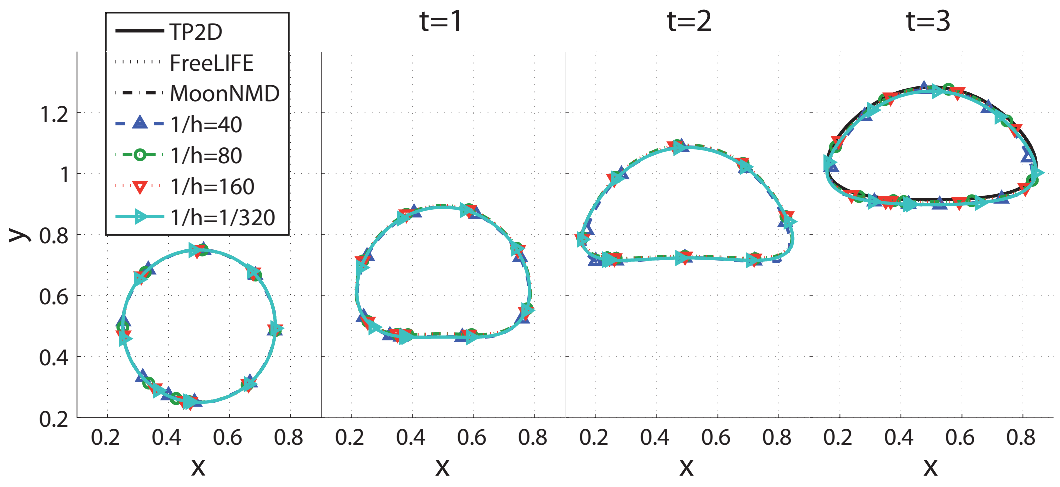


Figure 7.20: TC1 bubble shapes with PC-VOF in quadrangular meshes.

is clear. This behavior may find an explanation with the aid of the previously observed effects for the static case. The magnitude of the parasitic currents has been observed to be significantly higher in `interFoam` than in PC-VOF. As also observed by Klostermann *et al.* (2013), the parasitic currents are of the order of 5% of the maximum bubble rise velocity for `interFoam` in the quadrangular meshes, and 15-60%(!) for the triangular ones.

It is not the purpose of the present thesis to improve the advection scheme in `interFoam`. However, a recent approach taken by Roenby *et al.* (2016) may be an interesting choice to couple with the PC-VOF method, since it improves advection and also relies on the knowledge of the determination of the interface position from the α field.

Circularity during simulation time is presented for each solver – `interFoam` and PC-VOF– for quadrangular and triangular meshes in Fig. 7.23. In

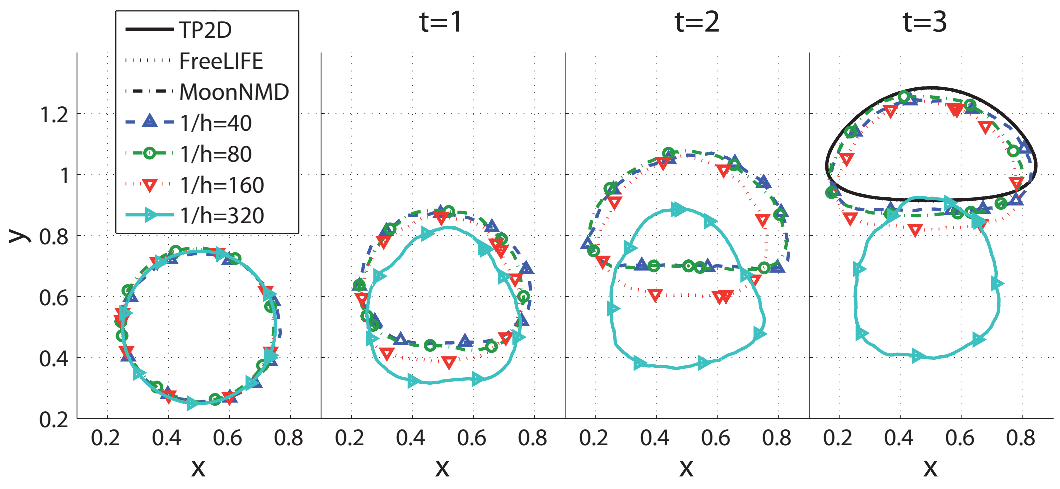


Figure 7.21: TC1 bubble shapes with `interFoam` in triangular meshes.

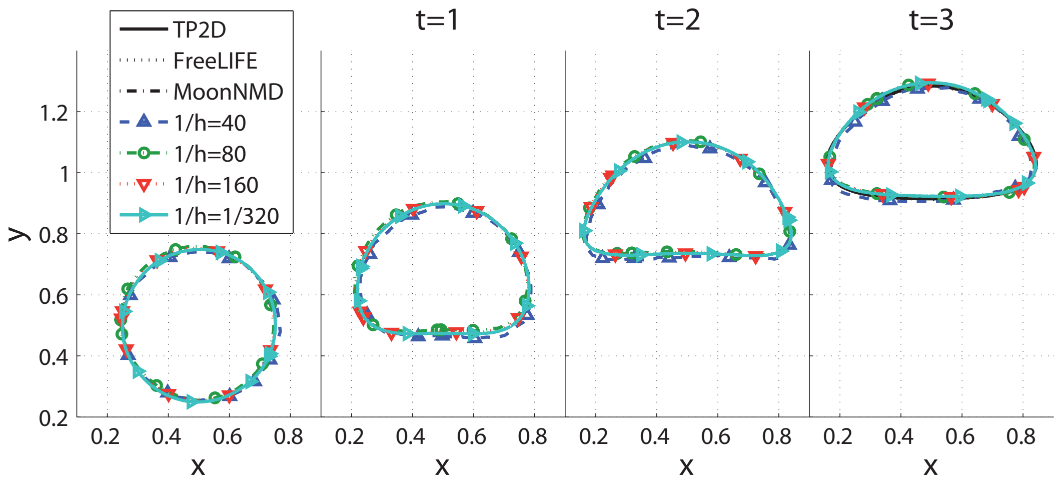
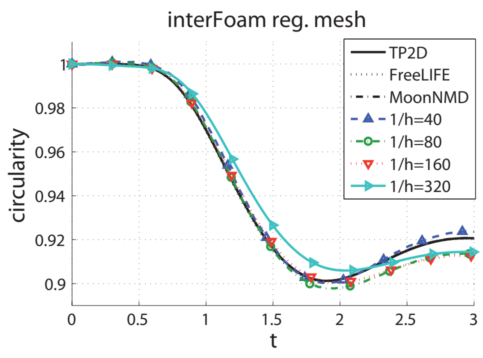


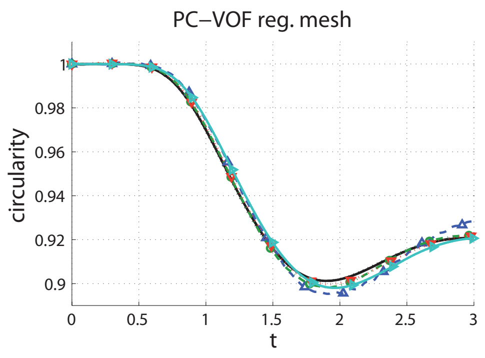
Figure 7.22: TC1 bubble shapes with PC-VOF in triangular meshes.

those figures the three benchmark data (which are nearly coincident) are also included. Once again, PC-VOF provides results that converge to benchmark data for quadrangular meshes as opposed to `interFoam`, which converges to a slightly lower circularity value. For triangular meshes, `interFoam` does not follow the benchmark pattern of circularity, while PC-VOF follows the benchmark circularity with great adherence!

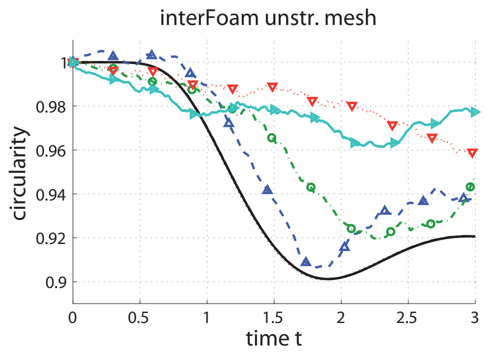
The center of mass y coordinate, computed by Eq.(7.6), during the simulation is presented for each solver in Fig. 7.24. For quadrangular meshes, both methods follow the benchmark data, with PC-VOF converging to a closer value. For triangular meshes, the predicted centers of mass obtained with `interFoam` are lower, indicating lower rising velocities. At the same time, results of PC-VOF in tri. meshes show great adherence to benchmark data, once again. Note that the solution of `interFoam` in triangular meshes diverge



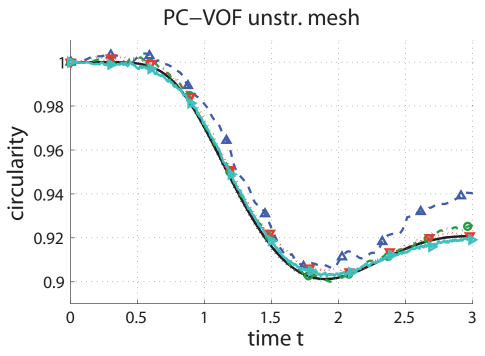
7.23(a): `interFoam` quad. mesh



7.23(b): PC-VOF quad. mesh



7.23(c): `interFoam` tri. mesh



7.23(d): PC-VOF tri. mesh

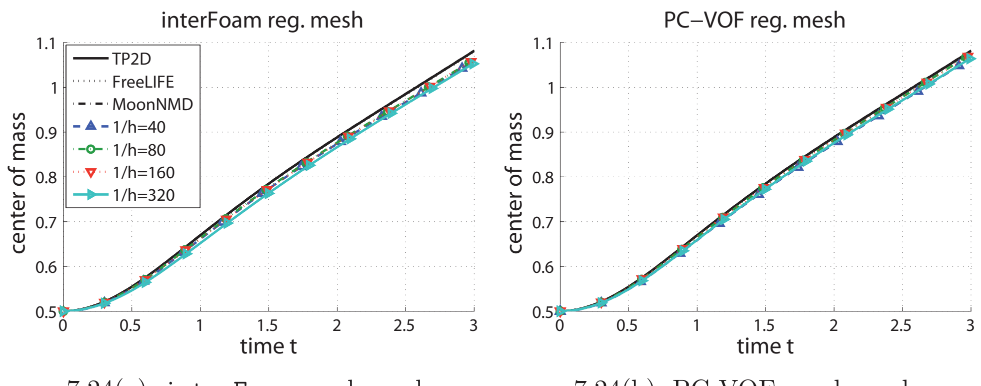
Figure 7.23: TC1 circularity.

from the benchmark, specially when the mesh is refined.

The bubble rise velocity y component, computed by Eq.(7.7), during the simulation is presented for each solver in Fig. (7.25) and compared to benchmark data. As noted for the center of mass evolution in time, the velocity evolution in time also provides the same conclusions. PC-VOF reaches a bubble terminal rise velocity closer to benchmark data than `interFoam`, for both kinds of meshes. For triangular meshes, `interFoam` diverges for the finest grid resolution. Nevertheless, PC-VOF shows a greater adherence to benchmark data.

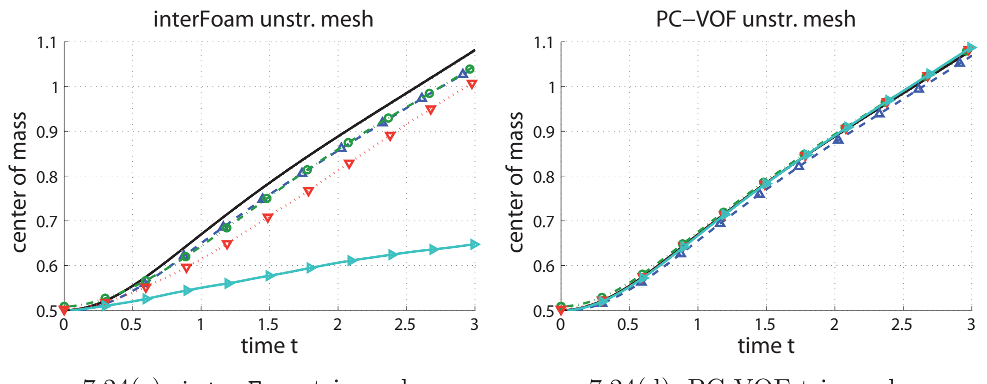
At the last instant of simulation time ($t = 3\ s$), the values of circularity, rise velocity and center of mass were compiled and plotted against mesh resolution to enable convergence checking in Figs. 7.28, 7.27 and 7.28. Benchmark data at the same instant t is also plotted for comparison (and made constant for all mesh resolutions for convenience, as convergence checking of benchmark data is not being evaluated, but taken as a reference value). Benchmark data are plotted in black lines with different patterns.

Figure 7.26 displays the convergence of circularity with mesh refinement. PC-VOF results in circularity that converges "exactly" to benchmark data for both types of meshes. In quads meshes, `interFoam`, even for the finest



7.24(a): `interFoam` quad. mesh

7.24(b): PC-VOF quad. mesh



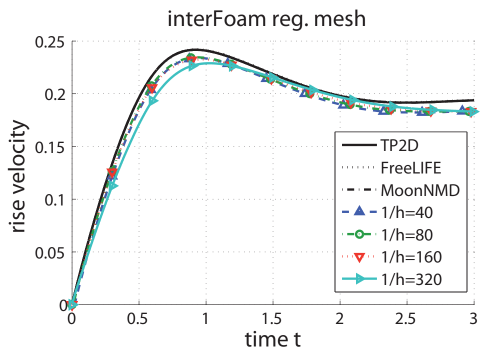
7.24(c): `interFoam` tri. mesh

7.24(d): PC-VOF tri. mesh

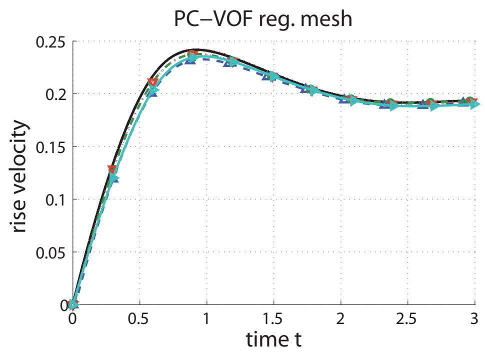
Figure 7.24: TC1 bubble center of mass evolution.

grid, does not approach benchmark data as closely. For triangular meshes, `interFoam` diverges from benchmark data with mesh refinement.

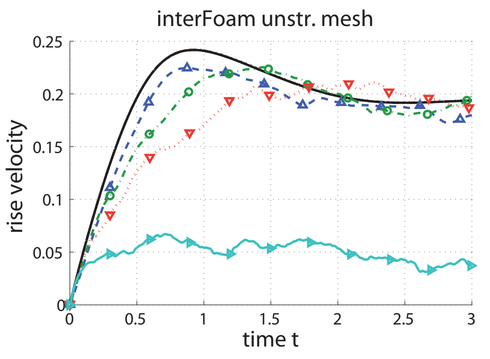
Figure 7.27 displays the rise velocity convergence with mesh refinement, while Figure 7.28 displays the y coordinate of the bubble center of mass convergence with mesh refinement. The same conclusions for the terminal velocity are applicable to the final position of the center of mass. Both methods in quadrangular meshes behave similarly, however PC-VOF shows a slight improvement in the final position compared to `interFoam`. For triangular meshes, `interFoam` results in perceivable divergence from benchmark data, however PC-VOF, once again, converges to benchmark data.



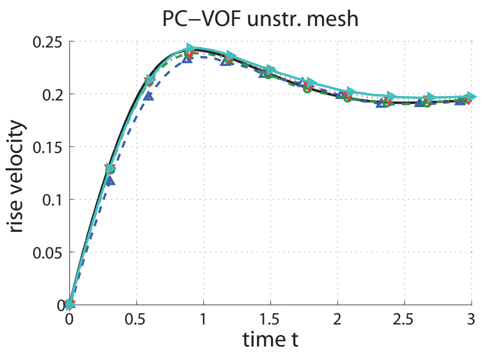
7.25(a): `interFoam` quad. mesh



7.25(b): PC-VOF quad. mesh



7.25(c): `interFoam` tri. mesh



7.25(d): PC-VOF tri. mesh

Figure 7.25: TC1 bubble rise velocity evolution.

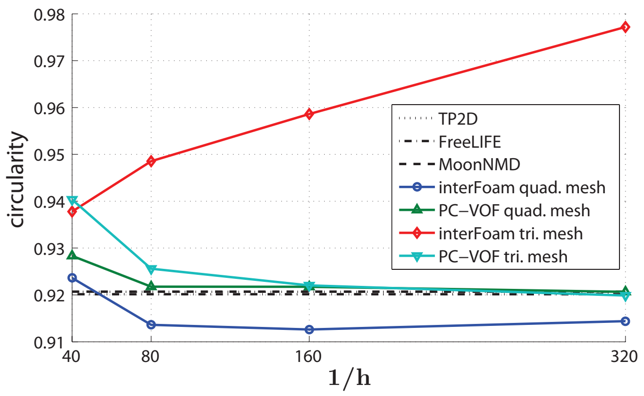


Figure 7.26: TC1 bubble circularity at $t = 3$ s convergence with mesh refinement.

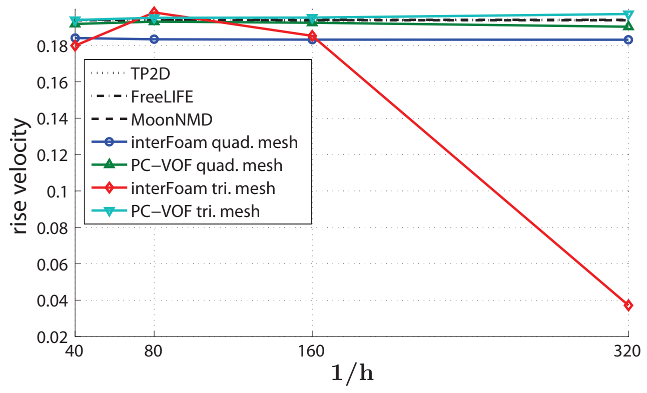


Figure 7.27: TC1 bubble rise velocity at $t = 3$ s convergence with mesh refinement.

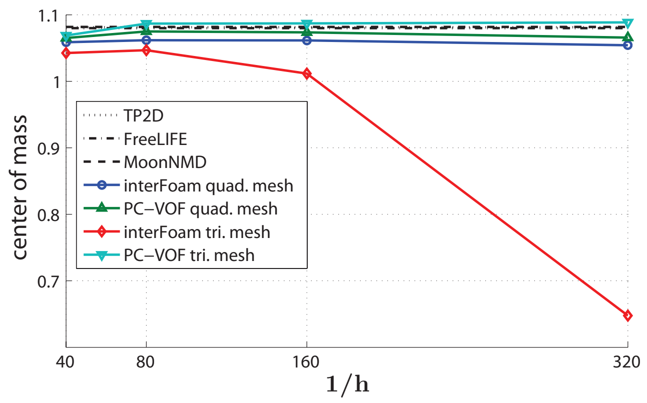


Figure 7.28: TC1 bubble center of mass y coordinate at $t = 3$ s convergence with mesh refinement.

Test Case 2

Following the same order presented for TC1, Figs. 7.29 and 7.30, present the bubble shape evolution with time for `interFoam` and PC-VOF in quadrangular meshes, respectively. Figures 7.31 and 7.32 correspond to `interFoam` and PC-VOF in triangular meshes. For the last step ($t = 3\ s$), the three benchmark bubble shape data are included in the graphs.

For this case, due to the ten times greater Capillary number and Weber number in relation to case TC1, it can be observed that as time evolves, the bubble displays a more intense deformation. It begins to elongate in the extremities, indicating that it will eventually break. In fact, the benchmark solution of TP2D predicts a bubble breakup.

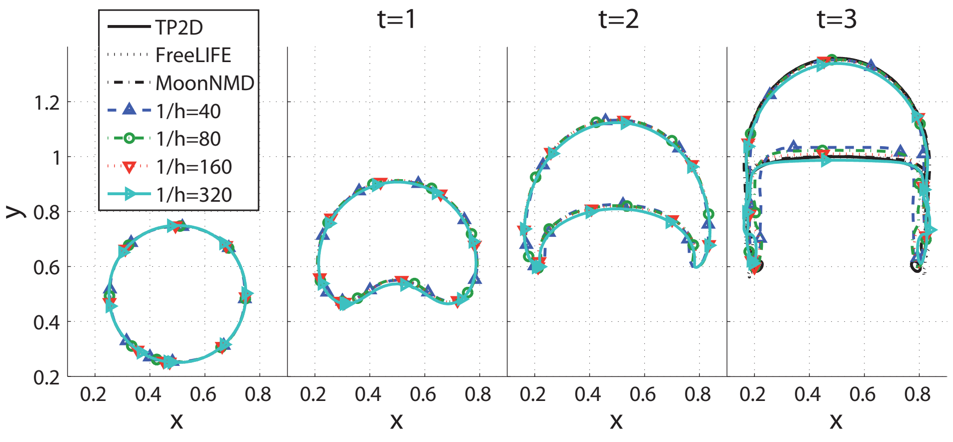


Figure 7.29: TC2 bubble shapes with `interFoam` in quadrangular meshes.

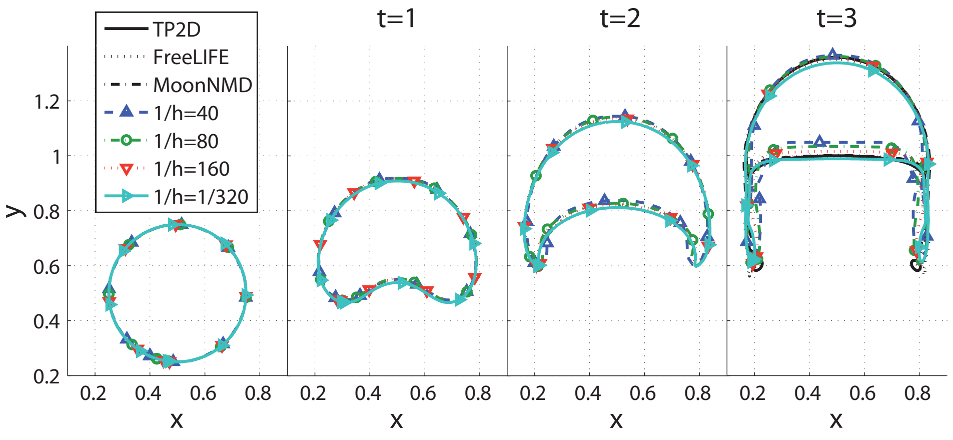


Figure 7.30: TC2 bubble shapes with PC-VOF in quadrangular meshes.

Once in this test case, with gravity turned on, surface tension plays a less significant role, compared to TC1, the results of both `interFoam` and PC-VOF are supposed to be closer to each other than in TC1. The results above confirm this statement and it is also an evidence that the only part of `interFoam` solver

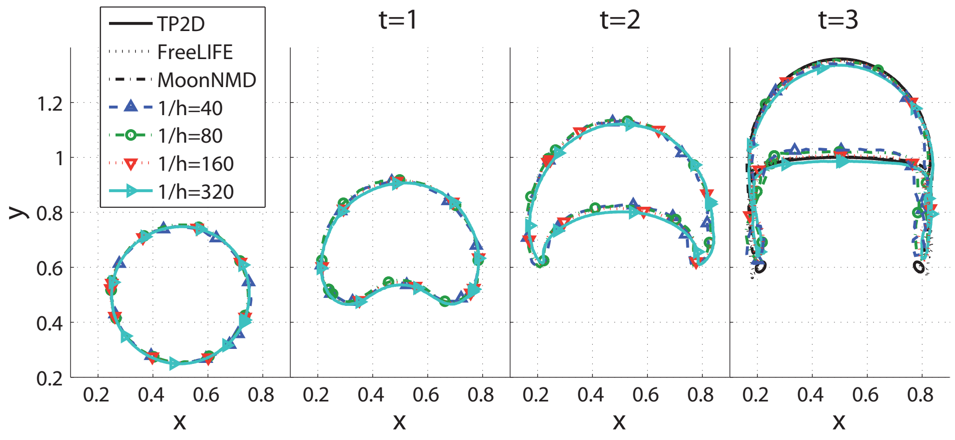


Figure 7.31: TC2 bubble shapes with `interFoam` in triangular meshes.

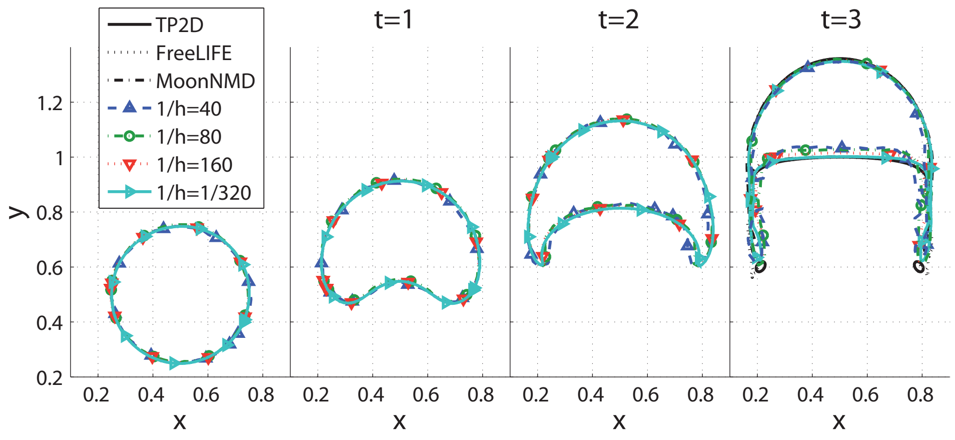
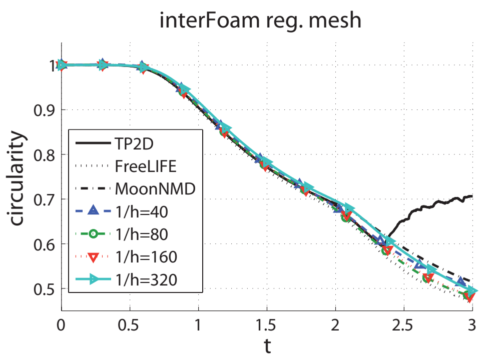


Figure 7.32: TC2 bubble shapes with PC-VOF in triangular meshes.

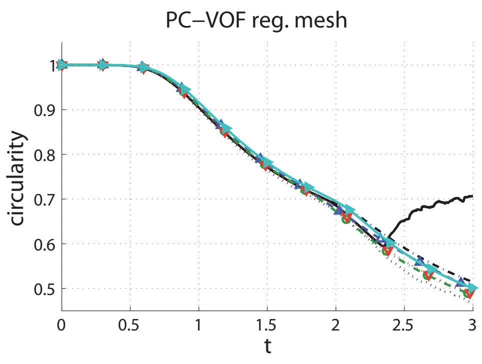
changed by this work was the surface tension computation. For quadrangular and triangular meshes, both methods agree very well with benchmark data and the results are very close to each other.

As mentioned, in the present test case, surface tension plays a less significant role than in the previous test case. This is indicated by the lower surface tension forces (higher **Eo** and **Ca** numbers) as well as higher viscosity and density ratios. Therefore, it is expected that the improved surface tension force computation does not lead to significant improvements in these results, compared to `interFoam`.

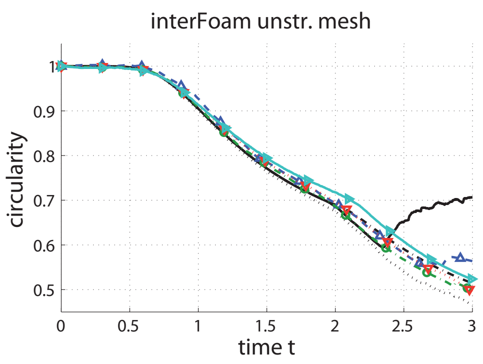
The evolution of the bubble circularity with time is presented for each solver – `interFoam` and PC-VOF– for quadrangular and triangular meshes in Fig. 7.33. TP2D shows a clear deviation from the other benchmark results, due to the bubble breakup that occurred in this solver. Both `interFoam` and PC-VOF follow the same tendencies for quadrangular and triangular meshes with good adherence to the results of FreeLIFE and MoonNMD, in which breakup does not occur.



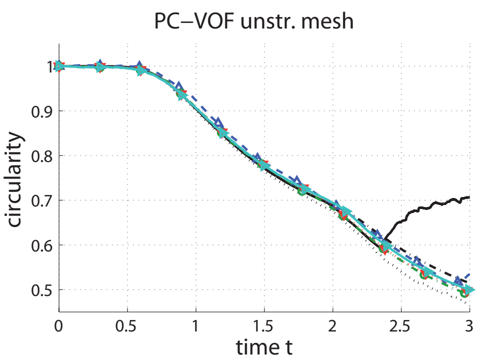
7.33(a): `interFoam` quad. mesh



7.33(b): PC-VOF quad. mesh



7.33(c): `interFoam` tri. mesh



7.33(d): PC-VOF tri. mesh

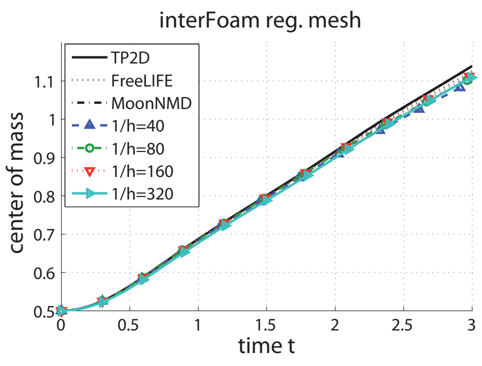
Figure 7.33: TC2 circularity.

The center of mass y coordinate, computed by Eq.(7.6), during the simulation is presented for each solver in Fig. 7.34 and the reader may notice, once again, that both methods agree well with benchmark data. Results obatined with PC-VOF, however, present a greater adherence to benchmark. It can be observed more clearly for the triangular meshes.

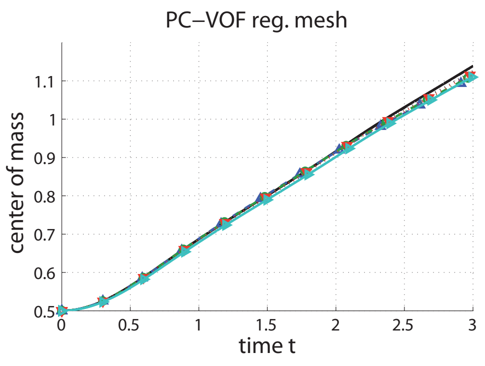
The bubble rise velocity y component, computed by Eq.(7.7), during the simulation is presented for each solver in Fig. 7.35 and compared to benchmark data. Both `interFoam` and PC-VOF provide similar results and differences between them are almost unnoticeable. Terminal velocity presents convergence towards benchmark data for all four situations: `interFoam` with quad. and tri. meshes and PC-VOF with quad. and tri. meshes.

As discussed for TC1, convergence with mesh resolution is presented for TC2 at the last instant of simulation time ($t = 3\ s$) for rise velocity, center of mass and circularity for `interFoam` and PC-VOF in quadrangular and triangular meshes.

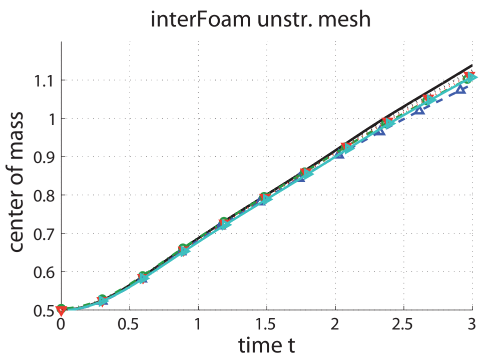
Figure 7.36 displays the rise velocity convergence with mesh refinement. Results of `interFoam` and PC-VOF are comparable and converge to benchmark data. The spread in the terminal velocity in the benchmark data is such that both `interFoam` and PC-VOF are lying in the very same region of benchmark



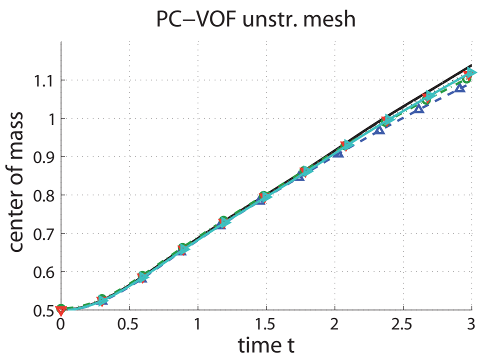
7.34(a): `interFoam` quad. mesh



7.34(b): PC-VOF quad. mesh



7.34(c): `interFoam` tri. mesh



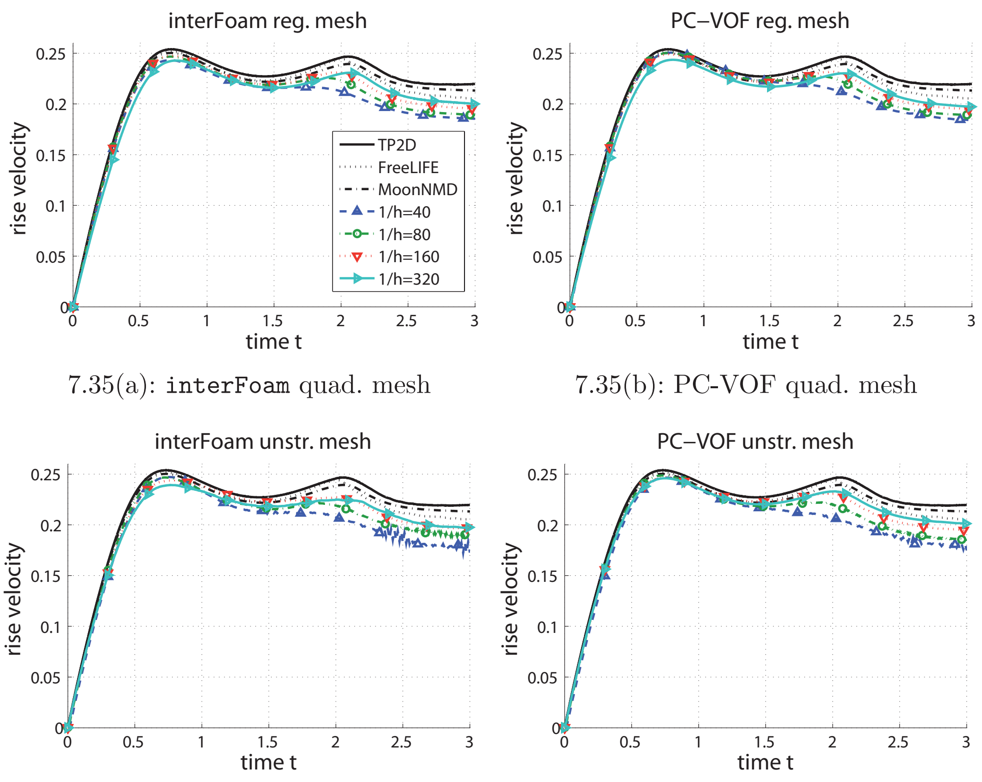
7.34(d): PC-VOF tri. mesh

Figure 7.34: TC2 bubble center of mass evolution.

results.

The mesh refinement effect on the y coordinate of the bubble center of mass is shown in Figure 7.37. Results for the center of mass converge for both `interFoam` and PC-VOF to a similar level slightly bellow the benchmark. Again, there is a spread in the benchmark data for this quantity, and both `interFoam` and PC-VOF provide similar results close to the benchmark.

Finally, Fig. 7.38 displays the convergence of circularity with mesh refinement. As noticed for the other quantities (terminal velocity and final center of mass position), both `interFoam` and PC-VOF follow the same tendencies. PC-VOF in quad. and tri. meshes and `interFoam` in quad. meshes result in circularities converging to a value in the strip bounded by FreeLIFE and MoonNMD results.



7.35(a): `interFoam` quad. mesh 7.35(b): PC-VOF quad. mesh

7.35(c): `interFoam` tri. mesh 7.35(d): PC-VOF tri. mesh

Figure 7.35: TC2 bubble rise velocity evolution.

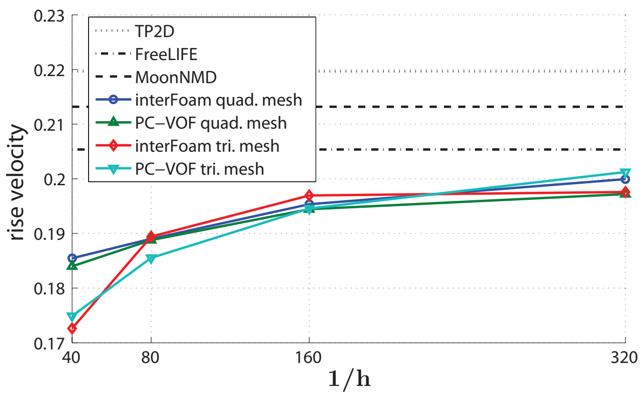


Figure 7.36: TC2 bubble rise velocity at $t = 3$ s convergence with mesh refinement.

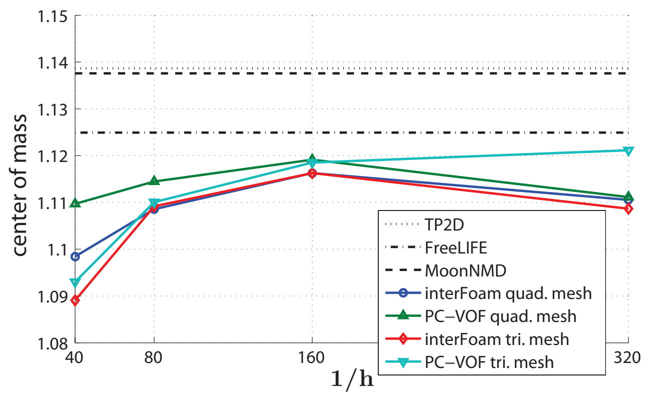


Figure 7.37: TC2 bubble center of mass y coordinate at $t = 3$ s convergence with mesh refinement.

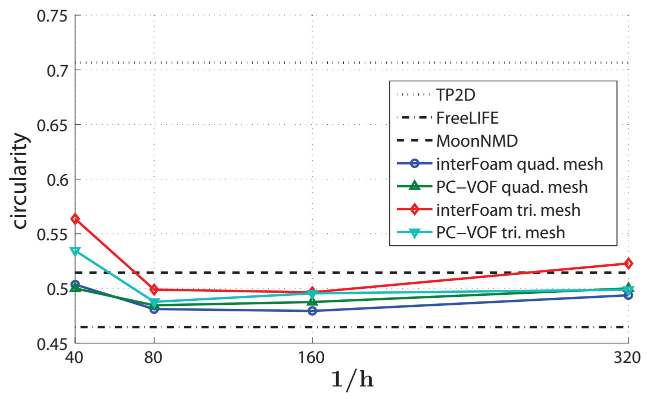


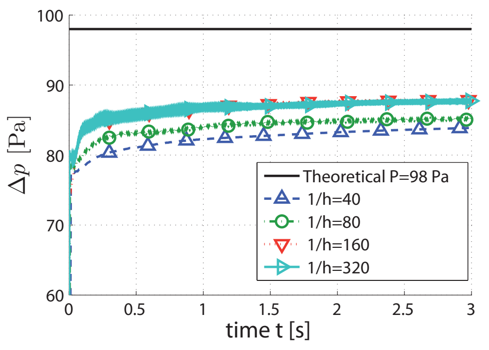
Figure 7.38: TC2 bubble circularity at $t = 3$ s convergence with mesh refinement.

7.2 Initialization Performance

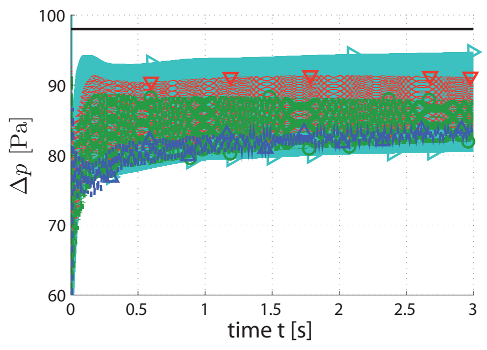
This section is devoted to test the performance of the initialization algorithm proposed in Section 4.3, the Monte Carlo integration method, denoted here by `setFracFields`. The 2D bubble test cases, TC1 and TC2, in zero gravity condition, presented in Section 7.1 will be used for this discussion. The method employed in the α field initialization in Section 7.1 was the `setFracFields`, that is the default method in this work (used in all results presented in this work). Here, these results are compared to the ones generated by simulations initialized by the standard OpenFOAM® initialization method: `setFields`. Results of spurious currents and pressure jump are displayed in the following discussion.

7.2.1 Test Case 1

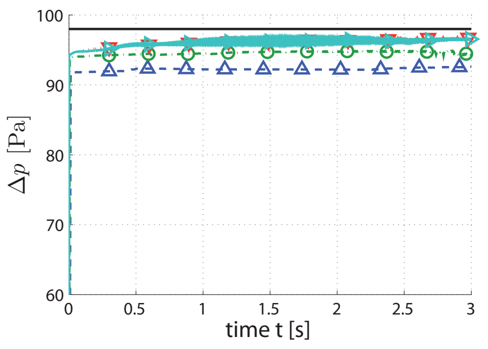
Figure 7.39 displays the evolution in time of the pressure jump Δp, computed by Eq.(7.9), for `interFoam` and PC-VOF initialized with `setFields` and `set-FracFields` for the TC1 test case. It can be clearly observed that initialization



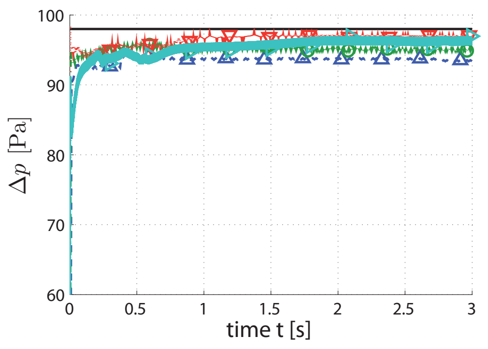
7.39(a): `interFoam setFracFields`



7.39(b): `interFoam setFields`



7.39(c): PC-VOF `setFracFields`



7.39(d): PC-VOF `setFields`

Figure 7.39: Pressure jump in TC1 for for different initialization methods.

performed by `setFracFields` increases stability of the solution compared to `setFields`. It is also very clear that `interFoam` is more sensible to field initialization, while PC-VOF is more independent and more stable for both types of initialization.

Figure 7.40 displays the behavior of Δp with respect to mesh refinement. It can be be observed that PC-VOF shows similar behavior in the pressure

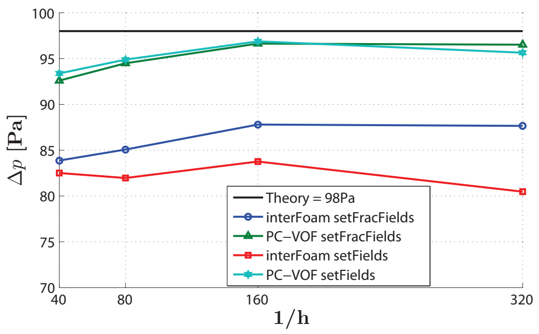
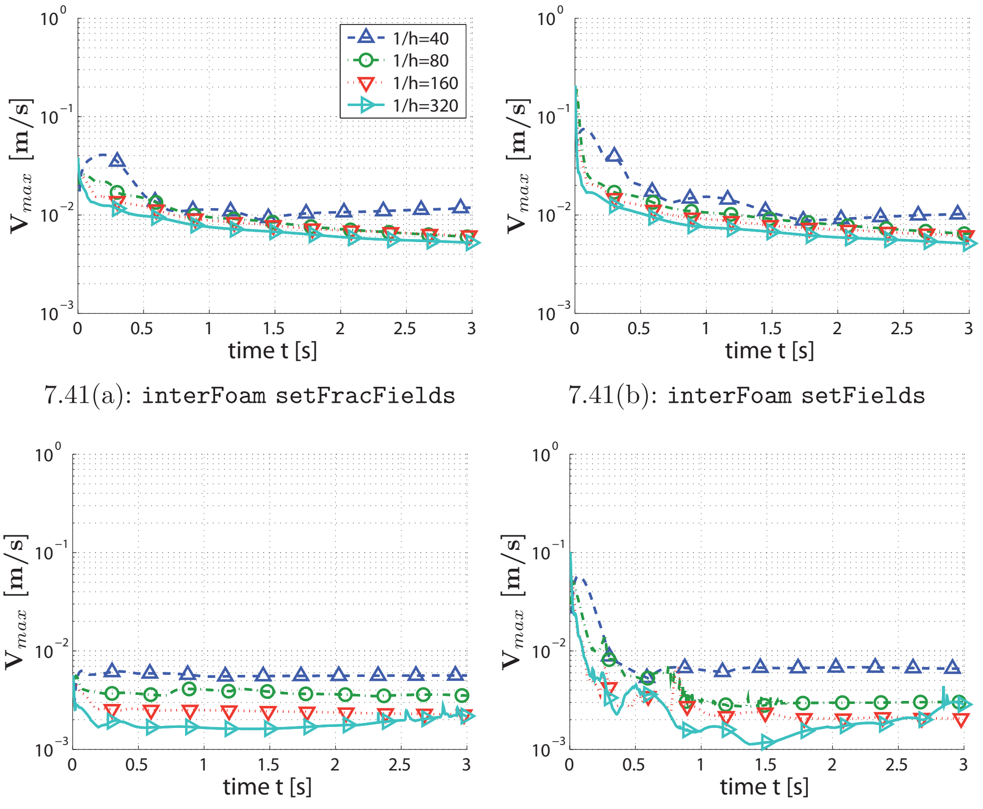


Figure 7.40: Pressure jump Δp behavior with mesh refinement in TC1.

jump with mesh refinement, while `interFoam` clearly presents a persistent increase in error of Δp estimates for `setFields` initialization compared to its results for Monte-Carlo initialization.

Figure 7.41 displays the evolution in time of the spurious currents, measured by the maximum magnitude of the velocity field, for `interFoam` and PC-VOF initialized with `setFields` and `setFracFields`. It is clear that spurious currents start from a higher level for `setFields` initialized simulations compared to Monte-Carlo initialization. This result was already expected once the interface produced by `setFields` presents a stepwise pattern. On the other hand, the interface produced by `setFracFields`, i.e. the Monte-Carlo integration, presents a smooth behavior, with its shape already much closer to the final circular interface. The interface initialized by `setFields` methods requires to change its stepwise shape in order to recover the equilibrium circular shape. At the final time steps, both initialization methods tend to converge to similar levels of parasitic currents. Just for the finest grid, PC-VOF displays a slight increase in its spurious currents in the `setFields` case.

Figure 7.42 displays the behavior of the parasitic currents at the final simulation time, $t = 3s$, for both `interFoam` and PC-VOF with both types of initialization methods. It is clear that PC-VOF produces better decays of spurious currents compared to `interFoam`, regardless of the initialization employed. Besides, `interFoam` converges to very similar values of spurious currents notwithstanding the initialization method. In conclusion, `setFrac-Fields` initialization produces consistently better results in both `interFoam` and PC-VOF for spurious currents and pressure jump, with the pressure jump in `interFoam` being more sensible to the initialization method.



7.41(a): `interFoam setFracFields` 7.41(b): `interFoam setFields`

7.41(c): PC-VOF `setFracFields` 7.41(d): PC-VOF `setFields`

Figure 7.41: Parasitic currents evolution in TC1 for different initialization methods.

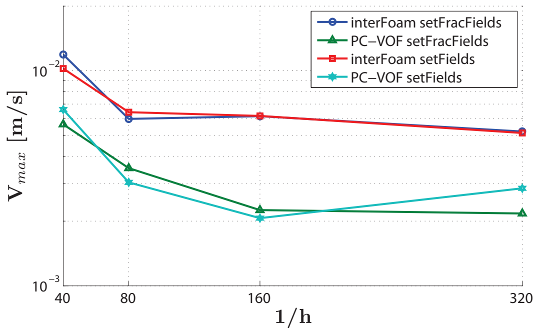
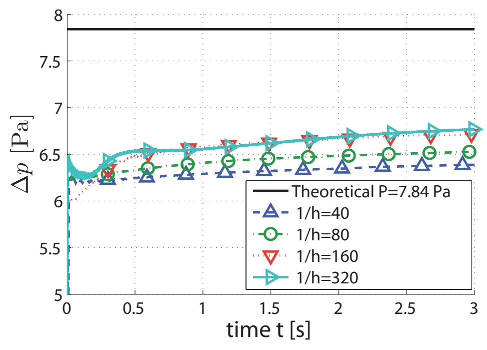


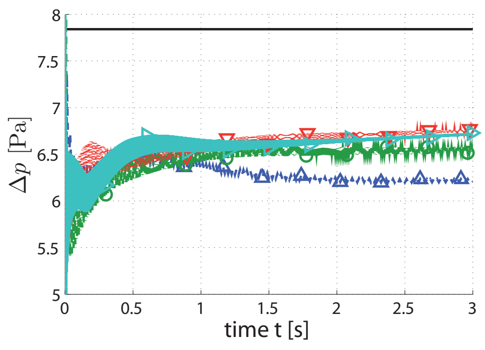
Figure 7.42: Spurious currents convergence with mesh resolution in TC1.

7.2.2 Test Case 2

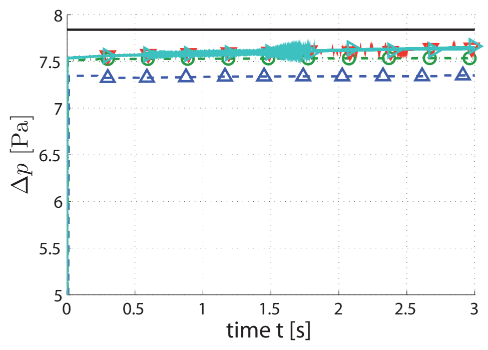
Figure 7.43 displays the evolution in time of the pressure jump Δp, computed by Eq.(7.9), for `interFoam` and PC-VOF initialized with `setFields` and `setFracFields`. Once again, `setFracFields` displays pressure jump with



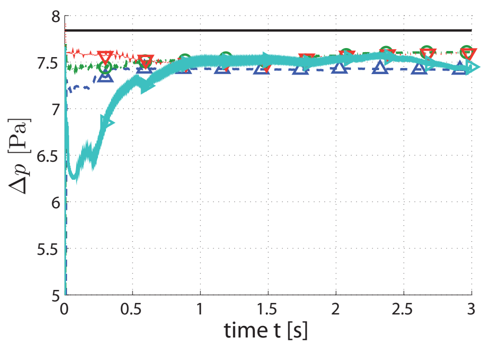
7.43(a): `interFoam setFracFields`



7.43(b): `interFoam setFields`



7.43(c): PC-VOF `setFracFields`



7.43(d): PC-VOF `setFields`

Figure 7.43: Pressure jump in TC2 for for different initialization methods.

reduced oscillations both both solvers.

Figure 7.44 displays the behavior of Δp with respect to mesh refinement. It can be observed that, when Monte-Carlo integration is employed, both solvers converge to a value of Δp slightly closer to the analytical solution compared to `setFields` results.

Figure 7.45 displays the evolution in time of the spurious currents, for the TC2 test case.

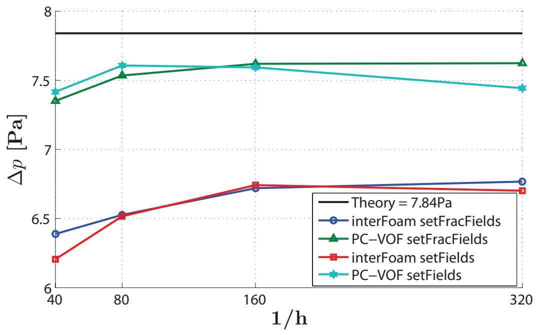
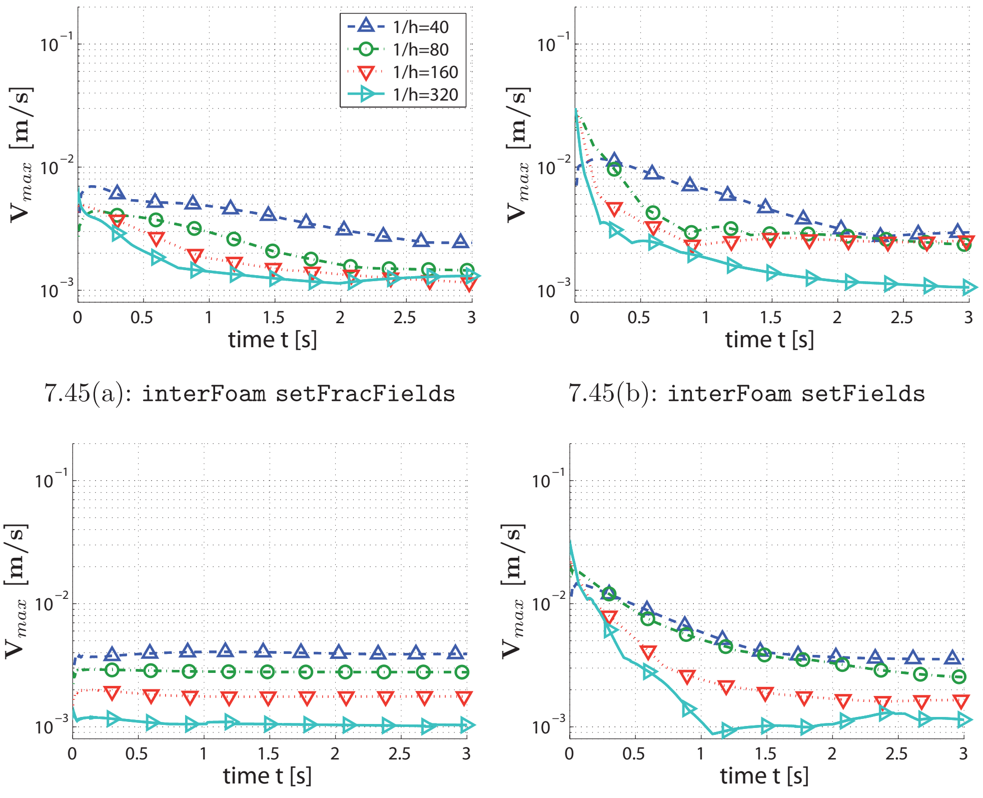


Figure 7.44: Pressure jump Δp behavior with mesh refinement in TC2.



7.45(a): `interFoam setFracFields`

7.45(b): `interFoam setFields`

7.45(c): PC-VOF `setFracFields`

7.45(d): PC-VOF `setFields`

Figure 7.45: Parasitic currents evolution in TC2 for different initialization methods.

Here it is very clear that both methods, when initialized with `setFields`, start at a higher level of spurious currents compared to the results for initialization performed with Monte-Carlo integration. These results allows to conclude that simulations initialized by `setFracFields`, i.e. the Monte-Carlo integration, require shorter periods of simulation time to reach stabilization in spurious currents. On the other hand, simulations initialized by `setFields` takes longer to stabilize.

Figure 7.46 displays the parasitic currents decay with mesh refinement for the four situations. As already mentioned, in this test case, the interface tension plays a less significant role in comparison to TC1, so that spurious currents converge to similar levels regardless of the solver or the initialization method.

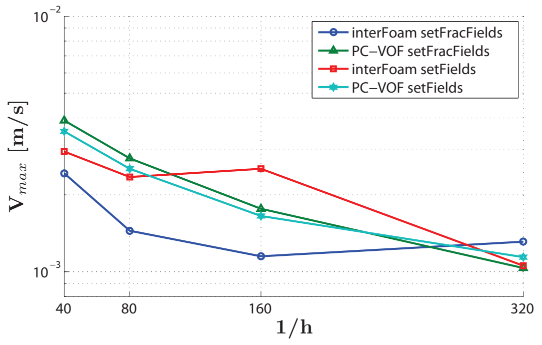


Figure 7.46: Spurious currents convergence with mesh resolution in TC2.

7.3 Three-Dimensional Bubble in a Liquid Column

Adelsberger *et al.* (2014) extended the 2D benchmark proposed by Hysing *et al.* (2009) to 3D for rising bubbles. The three-dimensional domain and initial bubble configuration is presented in Fig.7.47.

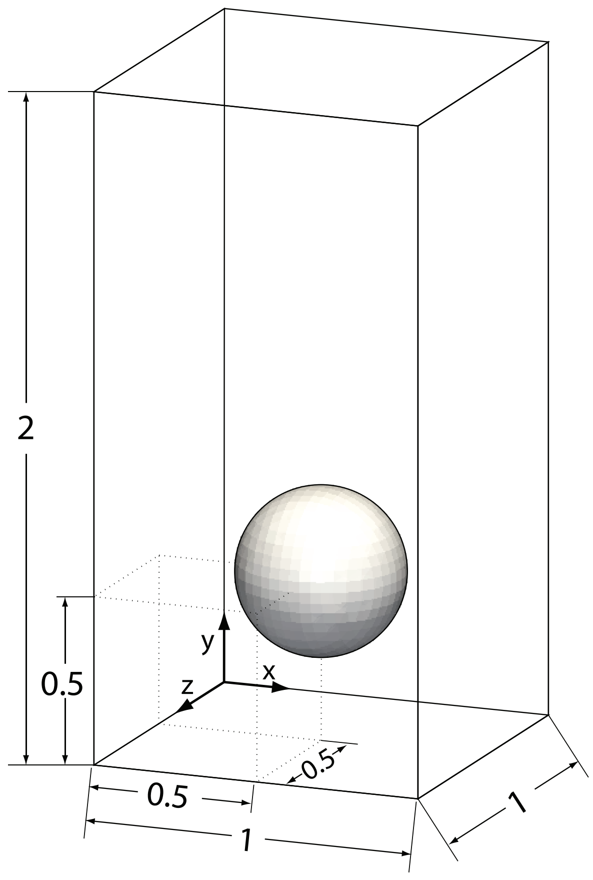


Figure 7.47: Bubble initial configuration in the 3D domain.

Adelsberger *et al.* (2014) presents results for the following three different solvers:

- DROPS (Gross & Reusken, 2011),
- NaSt3D (Croce *et al.*, 2010),
- `interFoam` version 2.2.2.

DROPS and NaSt3D are based on Level Set method. The version of `interFoam` used in Adelsberger *et al.* (2014) differs from the one used in this work (version 2.3.0), so results of both will be displayed in this work. As already discussed in Sec.7.1, Test Case 2 does not present significant differences between PC-VOF and `interFoam`, what is is due to the fact that surface tension does not play a significant role. Therefore, for this three-dimensional case, our attention will be focused only on TC1.

For convenience, the fluid properties are once again shown in Table 7.2:

ρ_1	ρ_2	μ_1	μ_2	σ	g
1000	100	10	1.0	24.50	0.98

Table 7.2: TC1 parameters.

The only difference between the 2D and 3D benchmarks is on the boundary conditions. Adelsberger *et al.* (2014) assume no-slip boundary conditions on the walls, so $\overrightarrow{V} = \overrightarrow{0}$ on all walls. For pressure and α fields, the boundary condition is set to zero gradient on all walls. The droplet is initialized as a sphere with radius $r_0 = 0.25$ and with its center positioned at $[0.5, 0.5, 0.5]^T$. Once again, simulations are performed for a period of time of 3 s. Adelsberger *et al.* (2014) presents results for `interFoam` for a mesh resolution ($128 \times 256 \times 128$), i.e. for a regular mesh with spacing $h = 1/128$. In this work, three mesh resolutions were employed with $1/h = 32,\ 64$ and 128. The time-step $\Delta\ t$ was fixed, being limited by a stability condition taking into account the capillary spurious velocities (Deshpande *et al.*, 2012), as already discussed in Sec.6.3. For each mesh, the time step Δt employed is shown in Table 7.3.

$1/h$	32	64	128
Δt	10^{-2}	5×10^{-3}	3×10^{-3}

Table 7.3: Times steps, in seconds, for each mesh resolution.

Adelsberger *et al.* (2014) examined the evolution of the bubble through 4 parameters: (i) rising velocity, (ii) its center of mass y coordinate; (iii) its diameters in x, y and z directions, respectively D_x, D_y and D_z; and (iv) its sphericity Ψ. Aiming to compare the results generated in this work by PC-VOF and `interFoam` (2.3.0) with the benchmark data, the same variables were computed here.

Sphericity Ψ, as defined in Wadell (1935), is given by the ratio between the area of the equivalent sphere with the same volume $\forall_b$ of the bubble and the bubble area A_b. Given the area of a sphere of radius r is $4\pi r^2$ and its volume is $4/3\pi r^3$, sphericity is given by

$$\Psi = \frac{\pi^{1/3}(6\forall_b)^{2/3}}{A_b} \tag{7.11}$$

As many articles performing experimental measurements (Liu *et al.*, 2015; Aoyama *et al.*, 2016) present the bubble aspect ratio E, which is the ratio between the bubble height to its width, it will also be presented here, for convenience. In this work, it is given by

$$E = \frac{D_y}{\frac{1}{2}(D_x + D_z)} \tag{7.12}$$

where the term $1/2(D_x + D_z)$ is the average bubble width.

7.3.1 Comparison to Benchmark Data

Figure 7.48 displays the rise velocity comparison with benchmark data for both `interFoam` and PC-VOF for the finest mesh, i.e. $1/h = 128$. All results agree well, with PC-VOF being closer to data of DROPS and NaStD than `interFoam`. Both versions of `interFoam` result in the lowest values for rising terminal velocity.

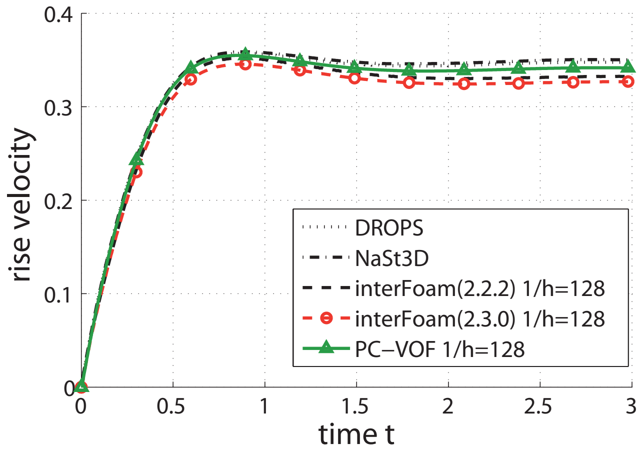


Figure 7.48: 3D bubble rise velocity.

Figure 7.49 displays the time evolution of the y coordinate of the center of mass. As expected, again, PC-VOF results are closer to DROPS and NaSt3D than both versions of `interFoam`, since the center of mass position is directly dependent on the rising velocity.

Comparison of the bubble sphericity Ψ and aspect ratio E obtained with the different solvers during the simulation time are shown in Fig. 7.50. Great adherence to the provided benchmark data is achieved for both PC-VOF and `interFoam`. Once again, PC-VOF shows the tendency to generate closer results to the Level Set solvers than `interFoam`.

Figure 7.51 displays the bubble height D_y and widths D_x and D_z during simulation time. PC-VOF also shows closer agreement with the benchmark for D_y and D_z. A slightly different result was obtained for D_x, with a close agreement of PC-VOF with the benchmark data at the first time steps, but at larger time instants, `interFoam` results are closer to the benchmark data.

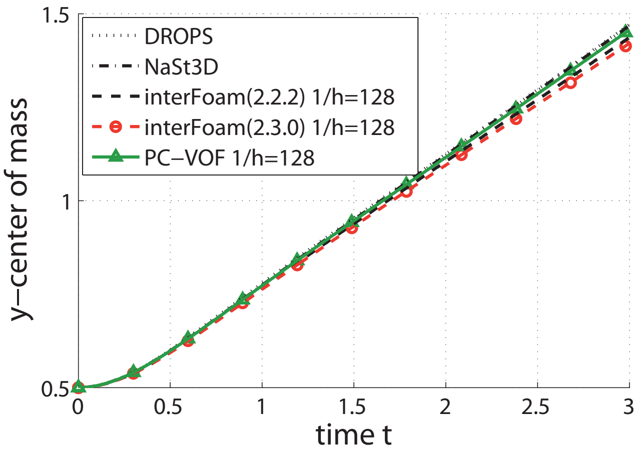
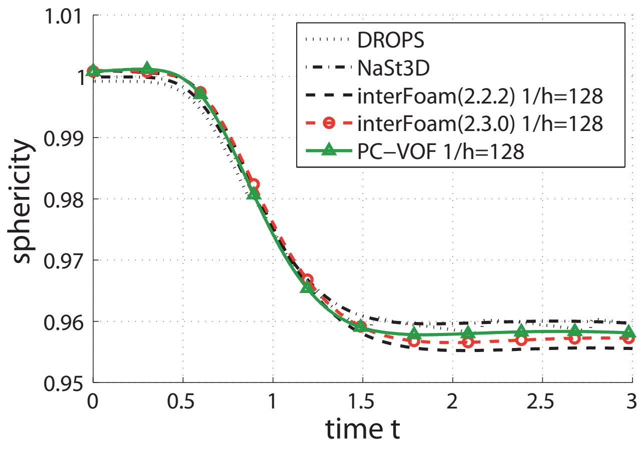
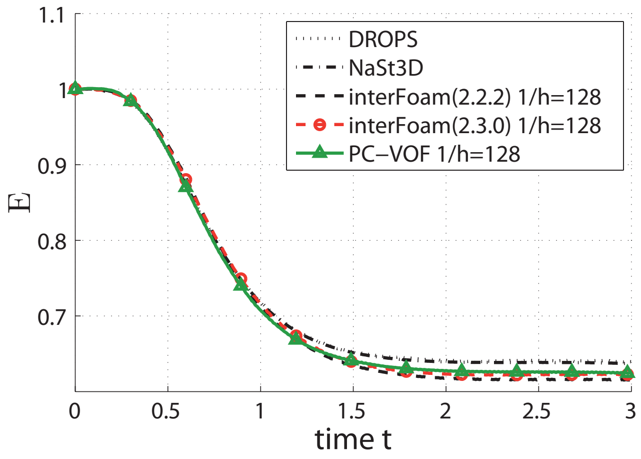


Figure 7.49: 3D bubble center of mass y coordinate.

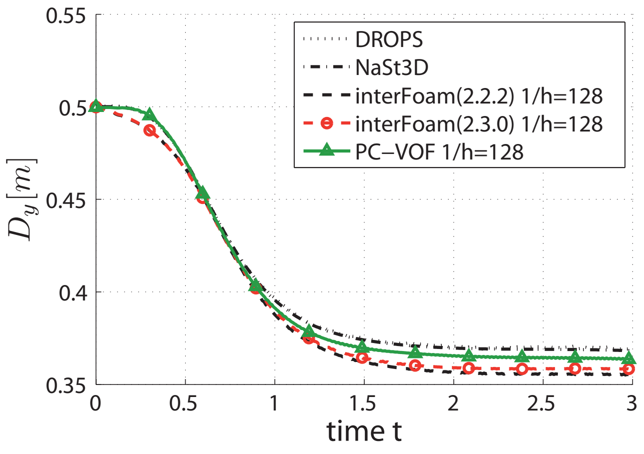


7.50(a): Sphericity Ψ

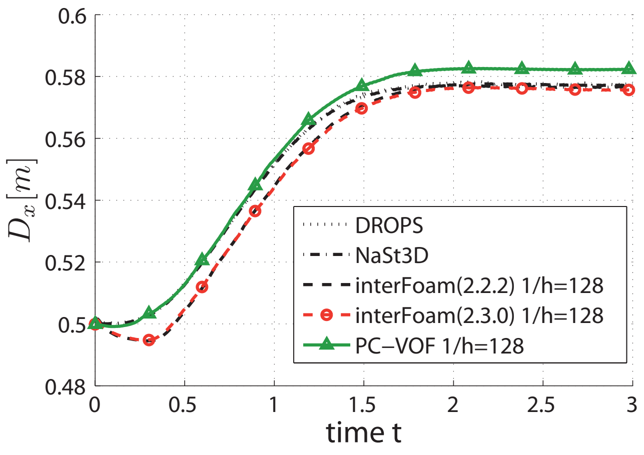


7.50(b): Aspect ratio E

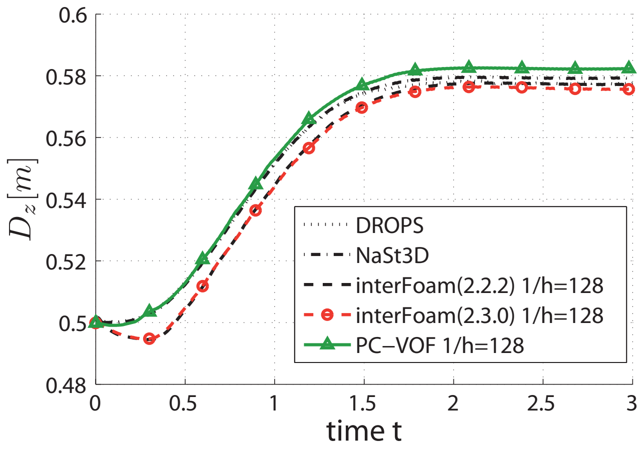
Figure 7.50: 3D bubble sphericity and aspect ratio during simulation.



7.51(a): height D_y



7.51(b): width D_x



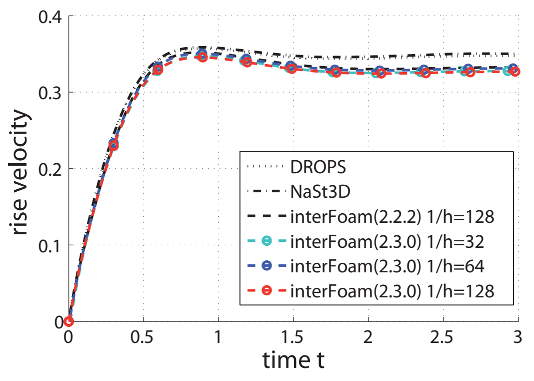
7.51(c): width D_z

Figure 7.51: 3D bubble diameters during simulation time.

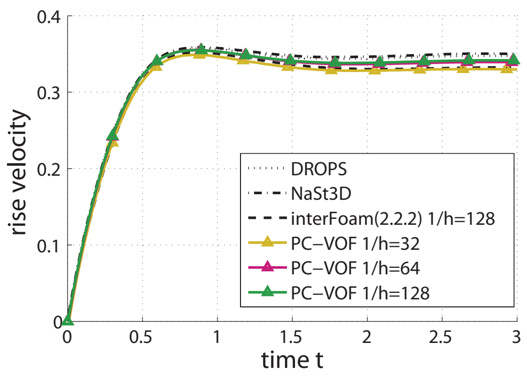
7.3.2 Mesh Convergence

As mentioned above, three mesh refinements were employed in these simulations. Results for the other two coarsest meshes are presented next, in order to allow the evaluation of mesh convergence.

Bubble rising velocity is presented in Fig. 7.52, while the bubble center of mass y coordinate is presented in Fig. 7.53 and bubble sphericity is shown in Fig. 7.54. Results for all meshes are shown in all figures. Examining these figures, it can be seen that both PC-VOF and `interFoam` converge to a level close to benchmark data.

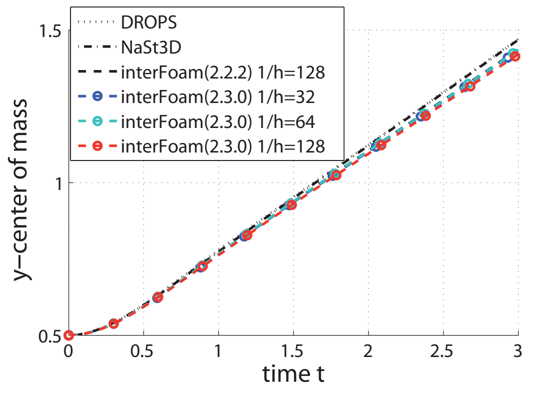


7.52(a): `interFoam`

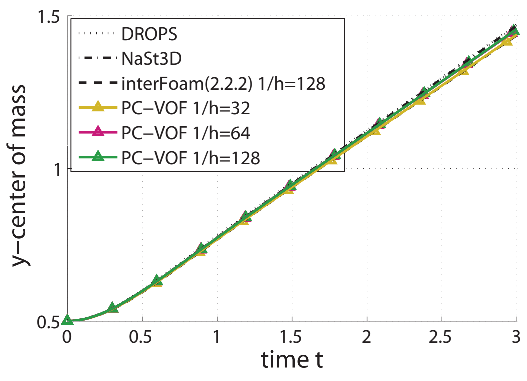


7.52(b): PC-VOF

Figure 7.52: 3D bubble rise velocity



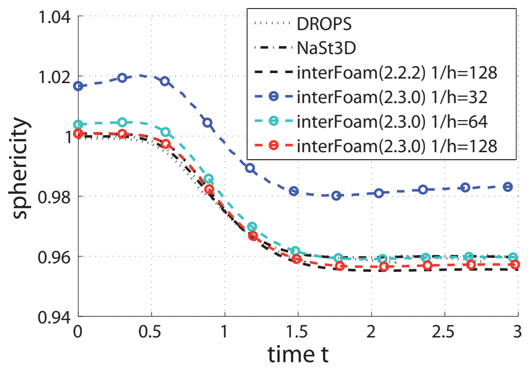
7.53(a): `interFoam`



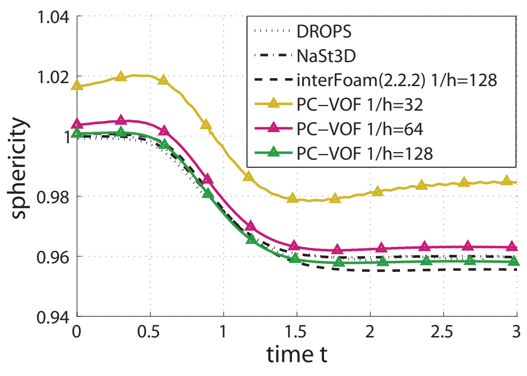
7.53(b): PC-VOF

Figure 7.53: 3D bubble center of mass vertical coordinate.

The aspect ratio E is presented for various mesh resolutions for `interFoam` and PC-VOF in Fig. 7.55. Convergence is observed for both methods. Results for `interFoam` clearly converge to results provided for `interFoam` (2.2.2) and results from PC-VOF converge to a level between the Level Set methods and `interFoam` results.

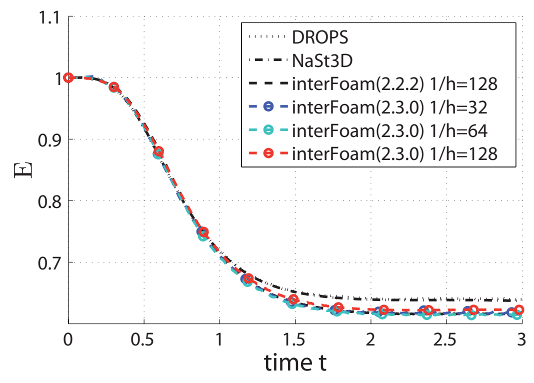


7.54(a): `interFoam`

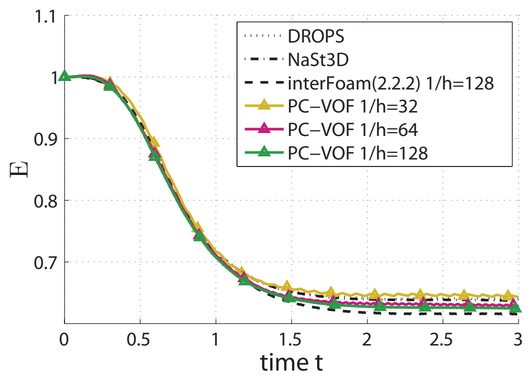


7.54(b): PC-VOF

Figure 7.54: 3D sphericity for varying mesh resolutions.



7.55(a): `interFoam`

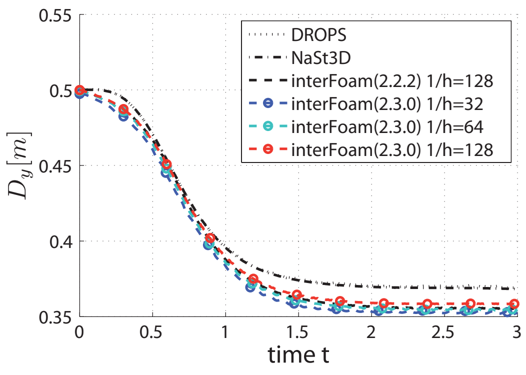


7.55(b): PC-VOF

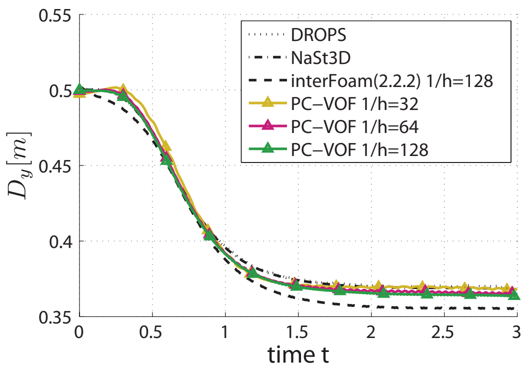
Figure 7.55: 3D rising bubble aspect ration E for varying mesh resolutions.

Bubble diameters for all employed meshes are presented in Fig. 7.56. Again, agreement to benchmark data as the mesh is refined is observed for both PC-VOF and `interFoam`. It is possible to observe that, at the very beginning of the simulation, `interFoam` decreases its D_x and D_z, following `interFoam` results provided by Adelsberger *et al.* (2014). On the other hand, PC-VOF follows the same tendency of the Level Set methods DROPS and NaSt3D.

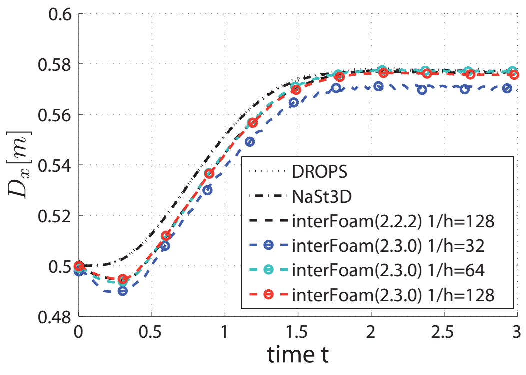
Finally, four screen-shots of the bubble simulated with PC-VOF and `interFoam` are presented in Fig. 7.57 for $t = 0, 1, 2$ and 3. Results for PC-VOF and `interFoam` shapes are aligned side by side to facilitate comparison. Observe that the bubble simulated by `interFoam` ascends slightly slower than PC-VOF. This issue was also apparent in the two-dimensional case presented in Section 7.1.



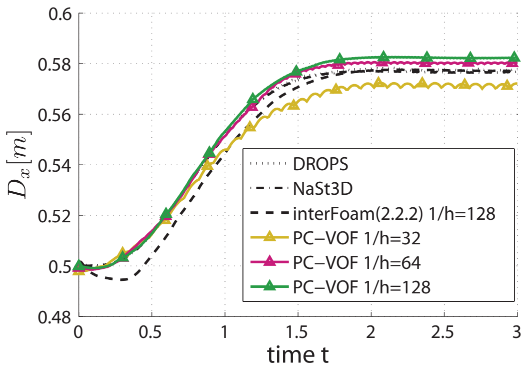
7.56(a): `interFoam` D_y



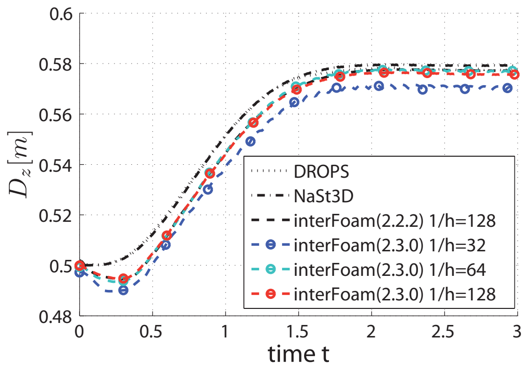
7.56(b): PC-VOF D_y



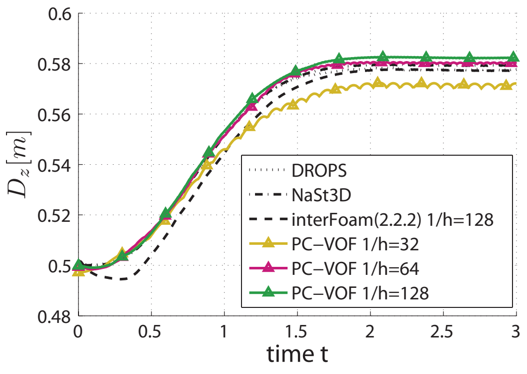
7.56(c): `interFoam` D_x



7.56(d): PC-VOF D_x



7.56(e): `interFoam` D_z



7.56(f): PC-VOF D_z

Figure 7.56: 3D bubble diameters for varying mesh resolutions.

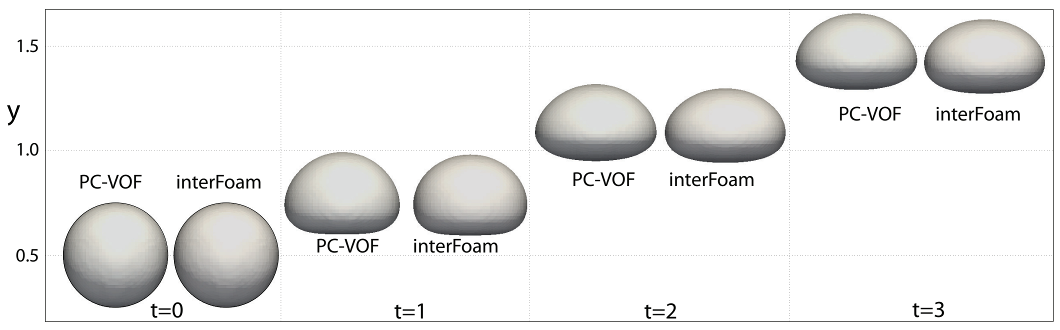


Figure 7.57: Bubble pictures at different time steps. PC-VOF and `interFoam` side by side.

8 Final Remarks

This work proposes and presents PC-VOF, a new approach for the problem of curvature computation in the VOF framework. It demonstrates the proposed methodology is feasible and makes a link between Computer Graphics and multiphase flows, so that further improvements in curvature computations in the CG area may also be applied to Computational Fluid Dynamics.

PC-VOF can have many variants (different methods for points, normals and curvatures computations) and can be applicable not only to VOF, but also to Level Set methods and any other methods that rely on marker functions to implicitly represent the interfaces. PC-VOF recovers sharpness of the interfaces, once it computes the curvatures at the interface positions, not at cells centers – which is the case of CSF approaches. The present approach does not require a specific type of mesh (regular, structured) to be implemented. It just relies on the Eulerian grid topology to determine the neighboring points, needed to perform triangulations, area computations and solving the presented minimization problems.

This work was focused primarily on the implementation and feasibility testing of the novel idea, as it shows indeed. Therefore, the code so far is not at its most efficient form and many optimizations may be implemented specially regarding memory allocation (that, still was not an issue, but can be improved). As the curvature computations are performed just at the interface points, and in practical situations the amount of points is considerably lower than the number of cells or edges in the fixed grid, the memory required for allocation – proportional to the number of points – , is not a limiting factor.

Curvature computation for point clouds is a wide research field in CG, so that many other methods can be tested in PC-VOF in future works. For this work, several approaches for curvature and normal computation were tested and the most efficient ones, that provided the most feasible results, were chosen. Normal computation by least-squares planes has shown to be the most stable and accurate (in most cases) method among the possibilities tested (i.e. it presented better performance than simple triangulation and triangulation weighted by triangles areas). Points sampling by linear interpolation also presented better results than by spline interpolation, once in spline interpolations the derivatives are based on interpolations of the inaccurate gradients of the volume fraction.

Mesh adaptivity allows for solving complex problems in which different flow length scales are present. Thus, for future works, it is convenient to implement PC-VOF in mesh adaptive refinement solvers, such as `interDyMFoam` (OpenFOAM, 2014), which applies mesh refinement at the interface region. Once the point clouds at each time step are freshly created from the available α field, PC-VOF can be promptly applied to adaptive mesh refinement solvers.

Once PC-VOF builds a point cloud and also a connection between them, based on the fixed grid topology, curvature computation by cyclic integrals over the edges of the surface elements, such as performed in Front-Tracking method (Tryggvason *et al.*, 2001), can also be employed. The author tested this approach for curvature computation, however, due to the noise on the point cloud, results were unfeasible. In order for this approach to work, the author believes some sort of filtering could be employed on the point cloud to mitigate noise. One idea is to approximate the surface elements by local B-spline surfaces to improve smoothness.

The proposed initialization algorithm by Monte Carlo Integration (`setFracFields`) has proven to produce better results than the ones produced by `setFields` initialization. Besides, simulations initialized by `setFracFields` take less time to reach convergence compared to the ones initialized by the standard OpenFOAM® initialization algorithm `setFields`.

The implementation of the new approach was based on standard OpenFOAM® VOF solver, `interFoam` (Weller, 2008) and significant improvements are demonstrated in Chapters 6 and 7. Compared to standard `interFoam`, a significant reduction in both spurious currents and in ripples on the pressure field is obtained. At the same time, pressure jump achieves values much closer to the expected ones with the novel approach. The aforementioned improvements justify the use of this methodology even at the cost of an increase in computational time, as discussed in Section 6.4. Moreover, these results show the feasibility of the method, which opens new possibilities for the persistent issue of curvature accuracy in VOF framework.

A
Appendix

A.1
Cubic Equation Real Roots

Consider a cubic equation of the form

$$t^3 + at^2 + bt + c = 0 \tag{A.1}$$

where a, b and c are real numbers. In order to obtain the real roots, one should follow these steps (Press *et al.*, 1992). Compute

$$Q = \frac{a^2 - 3b}{9} \qquad \text{and} \qquad R = \frac{2a^3 - 9ab + 27c}{54}$$

If $R^2 < Q^3$, then there are three real roots that can be obtained as follows. First compute $\theta = \arccos(R/\sqrt{Q^3})$, then the real roots t_1, t_2 and t_3 are given by

$$\begin{aligned} t_1 &= -2\sqrt{Q}\ \cos\left(\frac{\theta}{3}\right) - \frac{a}{3} \\ t_2 &= -2\sqrt{Q}\ \cos\left(\frac{\theta + 2\pi}{3}\right) - \frac{a}{3} \\ t_3 &= -2\sqrt{Q}\ \cos\left(\frac{\theta - 2\pi}{3}\right) - \frac{a}{3} \end{aligned}$$

Otherwise, in case $R^2 \geq Q^3$, there is only one real root and two complex conjugate ones. Their calculation is as follows. Compute

$$A = -\mathrm{sgn}(R)\left[|R| + \sqrt{R^2 - Q^3}\right]^{1/3} \qquad \text{and} \qquad B = \begin{cases} Q/A & \text{, if } A \neq 0 \\ 0 & \text{, if } A = 0 \end{cases}$$

where sgn is the sign function

$$\operatorname{sgn}(x) = \begin{cases} 1 & , \text{if } x \geq 0 \\ -1 & , \text{if } x < 0 \end{cases}$$

The roots are, then given by

$$\begin{aligned} t_1 &= (A+B) - \frac{a}{3} \\ t_2 &= -\frac{1}{2}(A+B) - \frac{a}{3} + i\frac{\sqrt{3}}{2}(A-B) \\ t_3 &= -\frac{1}{2}(A+B) - \frac{a}{3} - i\frac{\sqrt{3}}{2}(A-B) \end{aligned}$$

where t_1 is the real root.

A.2
Eigenvalues of an n by n Matrix

For a general $n \times n$ matrix $\boldsymbol{A}$, a nonzero vector $\overrightarrow{v}$ in $\mathbb{R}^n$ is an eigenvector of $\boldsymbol{A}$ if $\boldsymbol{A}\overrightarrow{v}$ is multiple of $\overrightarrow{v}$. That is (Anton & Rorres, 2001)

$$\boldsymbol{A}\overrightarrow{v} = \lambda\overrightarrow{v} \tag{A.2}$$

where λ is the corresponding eigenvalue. It results in

$$(\boldsymbol{A} - \lambda\boldsymbol{I})\overrightarrow{v} = 0 \tag{A.3}$$

where $\boldsymbol{I}$ is the identity matrix. Since $\overrightarrow{v}$ is a nonzero vector, the matrix $(\boldsymbol{A}-\lambda\boldsymbol{I})$ must be singular. That is to say its determinant must be zero

$$\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0. \tag{A.4}$$

A.2.1
2 by 2 matrix

Consider the following 2×2 matrix

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{A.5}$$

The determinant of $(\boldsymbol{A} - \lambda \boldsymbol{I})$ is given by

$$\begin{aligned}\det(\boldsymbol{A} - \lambda \boldsymbol{I}) =& (a-\lambda)(d-\lambda) - bc = 0\\ & \lambda^2 - \lambda(a+d) + ad - bc = 0\end{aligned}$$

where $a+d$ is the trace of $\boldsymbol{A}$ and $ad-bc$ its determinant D. Thus the above can be rewritten as

$$\lambda^2 - \lambda T + D = 0 \tag{A.6}$$

The eigenvalues of $\boldsymbol{A}$ are, thus, the roots of Eq.(A.6).

$$\lambda = \frac{T}{2} \pm \sqrt{\frac{T^2}{4} - D} \tag{A.7}$$

A.3 OpenFOAM Code Examples

Consider the following transient diffusion equation as in OpenFOAM manual (Greenshields, 2015)

$$\frac{\partial \phi}{\partial t} = \Gamma \nabla^2 \phi. \tag{A.8}$$

It can be coded as

```
solve
(
 fvm::ddt(phi) ==
 Gamma * fvm::laplacian(phi)
);
```

where `fvm` *namespace* means the Laplacian term is implicitly discretized, so that the above code discretizes time in an Implicit Euler scheme. If one desired to use, for instance, an Explicit Euler scheme, Eq.(A.8) would be coded like this

```
solve
(
 fvm::ddt(phi) ==
 Gamma * fvc::laplacian(phi)
);
```

where `fvc` *namespace* means the Laplacian term is explicitly discretized. The Crank & Nicolson (1996) scheme would be implemented as

```
solve
```

```
(
 fvm::ddt(phi) ==
 Gamma *0.5*(fvm::laplacian(phi) + fvc::laplacian(phi))
);
```

Bibliography

Adelsberger, Jutta, Esser, Patrick, Griebel, Michael, Groß, Sven, Klitz, Margit, & Rüttgers, Alexander. 2014. 3D incompressible two-phase flow benchmark computations for rising droplets. *In: Proceedings of the 11th World Congress on Computational Mechanics (WCCM XI), Barcelona, Spain, 2014.*

Albadawi, A., Donoghue, D.B., Robinson, A.J., Murray, D.B., & Delauré, Y.M.C. 2013. Influence of surface tension implementation in Volume of Fluid and coupled Volume of Fluid with Level Set methods for bubble growth and detachment. *International Journal of Multiphase Flow*, **53**, 11 – 28.

Anton, Howard, & Rorres, Chris. 2001. *Algebra Linear com Aplicacoes.* Bookman.

Aoyama, S., Hayashi, K., Hosokawa, S., & Tomiyama, A. 2016. Shapes of ellipsoidal bubbles in infinite stagnant liquids. *International Journal of Multiphase Flow*, **79**, 23 – 30.

Aris, R. 1990. *Vectors, Tensors and the Basic Equations of Fluid Mechanics.* Dover Books on Mathematics. Dover Publications.

Batagelo, Costa Harlen, & Wu, Shin-Ting. 2007. Estimating curvatures and their derivatives on meshes of arbitrary topology from sampling directions. *The Visual Computer*, **23**(9), 803–812.

Berberovic, Edin, van Hinsberg, Nils P., Jakirlic, Suad, Roisman, Ilia V., & Tropea, Cameron. 2009. Drop impact onto a liquid layer of finite thickness: Dynamics of the cavity evolution. *PHYSICAL REVIEW E*, **79**(3, 2).

Berger, M., & Rigoutsos, I. 1991. An algorithm for point clustering and grid generation. *IEEE Transactions on Systems, Man, and Cybernetics*, **21**(5), 1278–1286.

Berger, M. J., & Colella, P. 1989. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, **82**(May), 64–84.

Berger, Marsha J, & Oliger, Joseph. 1984. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, **53**(3), 484 – 512.

Boris, J.P., & Book, D.L. 1973. Flux-corrected transport. I. SHASTA, a fluid transport algorithm that works. *Journal of Computational Physics*, **11**(1), 38–69. cited By (since 1996)724.

Brackbill, J.U, Kothe, D.B, & Zemach, C. 1992. A continuum method for modeling surface tension. *Journal of Computational Physics*, **100**(2), 335 – 354.

Brennen, Christopher E. 2005. *Fundamentals of Multiphase Flow*. Cambridge: Cambridge University Press.

Briggs, William L., Henson, Van Emden, & McCormick, Steve F. 2000. *A Multigrid Tutorial (2Nd Ed.)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Cano-Lozano, J.C., Bolaños-Jiménez, R., Gutiérrez-Montes, C., & Martínez-Bazán, C. 2015. The use of Volume of Fluid technique to analyze multiphase flows: Specific case of bubble rising in still liquids. *Applied Mathematical Modelling*, **39**(12), 3290 – 3305.

Cheng, Zhang-Lin, & Zhang, Xiaopeng. 2009. Estimating differential quantities from point cloud based on a linear fitting of normal vectors. *Science in China Series F: Information Sciences*, **52**(3), 431–444.

Crank, J., & Nicolson, P. 1996. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Advances in Computational Mathematics*, **6**(1), 207–226.

Croce, Roberto, Griebel, Michael, & Schweitzer, Marc Alexander. 2010. Numerical simulation of bubble and droplet deformation by a level set approach with surface tension in three dimensions. *International Journal for Numerical Methods in Fluids*, **62**(9), 963–993.

Cummins, Sharen J., Francois, Marianne M., & Kothe, Douglas B. 2005. Estimating curvature from volume fractions. *Computers & Structures*, **83**, 425 – 434. Frontier of Multi-Phase Flow Analysis and Fluid-Structure Frontier of Multi-Phase Flow Analysis and Fluid-Structure.

Daly, Bart J. 1967. Numerical Study of Two Fluid Rayleigh-Taylor Instability. *Physics of Fluids*, **10**(2), 297–307.

Damián, Santiago Márques. 2013. *An Extended Mixture Model for the Simultaneous Treatment of Short and Long Scale Interfaces.* PhD Thesis, Universidad Nacional del Litoral, Santa Fe, Argentina.

Darwish, M. S. 1993. A New High-Resolution Scheme Based on the Normalized Variable Formulation. *Numerical Heat Transfer, Part B: Fundamentals*, **24**(3), 353–371.

Davis, Stephen F. 1994. Flux difference splittings and limiters for the resolution of contact discontinuities. *Applied Mathematics and Computation*, **65**(1), 3 – 18.

de Berg, Mark, Cheong, Otfried, van Kreveld, Marc, & Overmars, Mark. 2008. *Computational Geometry: Algorithms and Applications.* 3rd edn. Springer-Verlag Berlin Heidelberg.

de Melo, José Ronaldo Chaves. 1995. *Simulação numérica de escoamentos bifásicos com interfaces.* MSc Thesis, Pontifífica Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil.

Deshpande, Suraj S, Anumolu, Lakshman, & Trujillo, Mario F. 2012. Evaluating the performance of the two-phase flow solver interFoam. *Computational Science & Discovery*, **5**(1), 014016.

Francois, Marianne M., Cummins, Sharen J., Dendy, Edward D., Kothe, Douglas B., Sicilian, James M., & Williams, Matthew W. 2006. A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework. *Journal of Computational Physics*, **213**(1), 141 – 173.

Fuster, Daniel, Agbaglah, Gilou, Josserand, Christophe, Popinet, Stéphane, & Zaleski, Stéphane. 2009. Numerical simulation of droplets, bubbles and waves: state of the art. *Fluid Dynamics Research*, **41**(6), 065001+.

Fuster, Daniel, Bague, Anne, Boeck, Thomas, Le Moyne, Luis, Leboissetier, Anthony, Popinet, Stephane, Ray, Pascal, Scardovelli, Ruben, & Zaleski, Stephane. 2009. Simulation of primary atomization with an octree adaptive mesh refinement and VOF method. *International Journal of Multiphase Flow*, **35**(6), 550–565.

Fyfe, D.E, Oran, E.S, & Fritts, M.J. 1988. Surface tension and viscosity with lagrangian hydrodynamics on a triangular mesh. *Journal of Computational Physics*, **76**(2), 349 – 384.

Ganesan, Sashikumaar, Matthies, Gunar, & Tobiska, Lutz. 2007. On spurious velocities in incompressible flow problems with interfaces. *Computer Methods in Applied Mechanics and Engineering*, **196**(7), 1193 – 1202.

Garrioch, S. H., & Baliga, B. R. 2006. A PLIC volume tracking method for the simulation of two-fluid flows. *International Journal for Numerical Methods in Fluids*, **52**(10), 1093–1134.

Geuzaine, C., & Remacle, J. F. 2009. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*.

Ghobadian, A. 1991. *Development of a method for numerical simulation of flows with moving interfaces*. Tech. rept. National Power Report TEC/L/0077/M91.

Golub, Gene H., & Van Loan, Charles F. 1996. *Matrix Computations (3rd Ed.)*. Baltimore, MD, USA: Johns Hopkins University Press.

Gopala, Vinay R., & van Wachem, Berend G.M. 2008. Volume of fluid methods for immiscible-fluid and free-surface flows. *Chemical Engineering Journal*, **141**(1–3), 204 – 221.

Greenshields, Christopher J. 2015 (Dec.). *OpenFOAM - The Open Source CFD Toolbox - Programmers's Guide*. 3.0.1 edn. CFD Direct Ltd., United Kingdom.

Gross, Sven, & Reusken, Arnold. 2011. *Numerical Methods for Two-Phase Incompressible Flows*. Springer Series in Computational Mathematics, vol. 40. Springer.

Harlow, F.H., Amsden, A.A., & Nix, J.R. 1976. Relativistic fluid dynamics calculations with the particle-in-cell technique. *Journal of Computational Physics*, **20**(2), 119 – 129.

Harlow, Francis H., & Welch, J. Eddie. 1965. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *Physics of Fluids (1958-1988)*, **8**(12), 2182–2189.

Helmsen, J., & Colella, P. 1997 (Jan). *Non-convex profile evolution in two dimensions using volume of fluids*. Tech. rept. Lawrence Livermore National Lab., CA (United States).

Hirt, C. W., & Nichols, B. D. 1981. Volume of fluid /VOF/ method for the dynamics of free boundaries. *Journal of Computational Physics*, **39**(Jan.), 201–225.

Hirt, C.W, & Shannon, J.P. 1968. Free-surface stress conditions for incompressible-flow calculations. *Journal of Computational Physics*, **2**(4), 403 – 411.

Hoang, Duong A., van Steijn, Volkert, Portela, Luis M., Kreutzer, Michiel T., & Kleijn, Chris R. 2013. Benchmark numerical simulations of segmented two-phase flows in microchannels using the Volume of Fluid method. *Computers & Fluids*, **86**, 28 – 36.

Hysing, S., Turek, S., Kuzmin, D., Parolini, N., Burman, E., Ganesan, S., & Tobiska, L. 2009. Quantitative benchmark computations of two–dimensional bubble dynamics. *International Journal for Numerical Methods in Fluids*, **60**(11), 1259–1288. doi: 10.1002/fld.1934.

Isaacson, E., & Keller, H.B. 1994. *Analysis of Numerical Methods*. Dover Books on Mathematics. Dover Publications.

Ishii, Mamoru, & Hibiki, Takashi. 2011. *Thermo-Fluid Dynamics of Two-Phase Flow*. second edn. Springer.

Issa, R.I. 1986. Solution of the Implicity Discretized Fluid Flow Equations by Operator-Splitting. *Journal of Computational Physics*, **62**, 40–65.

Ito, Kei, Kunugi, Tomoaki, Ohshima, Hiroyuki, & Kawamura, Takumi. 2013. A volume-conservative {PLIC} algorithm on three-dimensional fully unstructured meshes. *Computers & Fluids*, **88**(0), 250 – 261.

Ito, Kei, Kunugi, Tomoaki, Ohno, Shuji, Kamide, Hideki, & Ohshima, Hiroyuki. 2014. A high-precision calculation method for interface normal and curvature on an unstructured grid. *Journal of Computational Physics*, **273**, 38 – 53.

Ivey, Christopher B., & Moin, Parviz. 2015. Accurate interface normal and curvature estimates on three-dimensional unstructured non-convex polyhedral meshes. *Journal of Computational Physics*, **300**, 365 – 386.

Jasak, H., & Weller, H.G. 1995 (February). *Interface-tracking capabilities of the InterGamma differencing scheme*. Tech. rept. CFD research group, Imperial College, London.

John, Volker, & Matthies, Gunar. 2004. MooNMD –a Program Package Based on Mapped Finite Element Methods. *Comput. Vis. Sci.*, **6**(2), 163–170.

Kang, Myungjoo, Fedkiw, Ronald P., & Liu, Xu-Dong. 2000. A Boundary Condition Capturing Method for Multiphase Incompressible Flow. *Journal of Scientific Computing*, **15**(3), 323–360.

Kassar, Bruno de Barros Mendes, & Nieckele, Angela Ourivio. 2015. Curvature estimation on fluid interfaces by point-cloud sampling. *In: Proceedings of the 23rd ABCM International Congress of Mechanical Engineering. Rio de Janeiro, Brazil.*

Kassar, Bruno de Barros Mendes, Carneiro, João Neuenschwander Escosteguy, & Nieckele, Angela Ourivio. 2016. Numerical simulation of a single rising bubble by a VOF method with enhanced curvature estimation based on point-cloud sampling. *In: Proceedings of the 9th International Conference on Multiphase Flow. Firenze, Italy.*

Klostermann, J., Schaake, K., & Schwarze, R. 2013. Numerical simulation of a single rising bubble by VOF with surface compression. *International Journal for Numerical Methods in Fluids*, **71**(8), 960–982.

Kreyszig, Erwin. 1991. *Differential Geometry*. 1st edn. USA: Dover Publications.

Lafaurie, B., Nardone, C., Scardovelli, R., Zaleski, S., & Zanetti, G. 1994. Modelling merging and fragmentation in multiphase flows with SURFER. *J. Comput. Phys.*, **113**, 134–147.

Leonard, B.P. 1979. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Computer Methods in Applied Mechanics and Engineering*, **19**(1), 59 – 98.

Levin, D. 2003. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization*, **3**.

Liu, Liu, Yan, Hongjie, & Zhao, Guojian. 2015. Experimental studies on the shape and motion of air bubbles in viscous liquids. *Experimental Thermal and Fluid Science*, **62**, 109 – 121.

Liu, Xu-Dong, Fedkiw, Ronald P., & Kang, Myungjoo. 2000. A Boundary Condition Capturing Method for Poisson's Equation on Irregular Domains. *Journal of Computational Physics*, **160**(1), 151 – 178.

Maric, Tomislav, Marschall, Holger, & Bothe, Dieter. 2013 (May). *voFoam - A geometrical Volume of Fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using OpenFOAM*. Comments: voFoam - geometrical unsplit VoF algorithm on unstructered meshes".

Noh, W. F., & Woodward, P. 1976. SLIC (simple line interface calculation). *Proceedings, Fifth International Conference on Fluid Dynamics, Lecture Notes in Physics*, **59**, 330–340.

OpenFOAM. 2014 (Feb.). *OpenFOAM - The Open Source CFD Toolbox - User Guide*. 2.3.0 edn. OpenFOAM Foundation.

Osher, Stanley, & Sethian, James A. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, **79**(1), 12 – 49.

Panton, R.L. 2005. *Incompressible Flow*. Wiley.

Parolini, N, & Burman, E. 2005. A finite element level set method for viscous free-surface flows. *Pages 416–427 of:* Primicerio, M and Spigler, R and Valente, V (ed), *Applied and Industrial Mathematics in Italy*. Series on Advances in Mathematics for Applied Sciences, vol. 69. Italian Soc Appl & Ind Math. 7th Conference on Applied and Industrial Mathematics in Italy, Venice, Italy, Sep. 20-24, 2004.

Patankar, Suhas V. 1980. *Numerical Heat Transfer and Fluid Flow*. Hemisphere Publishing Corporation.

Pericleous, Kyriacos A, & Chan, Koon. 1994. The SEA method for free-surface problems with heat transfer and change of phase. *Journal of Fluid Engineering*, **185**(June), 227–236.

Pfister, Hanspeter, Zwicker, Matthias, van Baar, Jeroen, & Gross, Markus. 2000. Surfels: Surface Elements As Rendering Primitives. *Pages 335–342 of: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Pivello, M.R., Villar, M.M., Serfaty, R., Roma, A.M., & Silveira-Neto, A. 2014. A fully adaptive front tracking method for the simulation of two phase flows. *International Journal of Multiphase Flow*, **58**, 72 – 82.

Popinet, Stéphane. 2003. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Journal of Computational Physics*, **190**(2), 572 – 600.

Popinet, Stéphane. 2009. An accurate adaptive solver for surface-tension-driven interfacial flows. *Journal of Computational Physics*, **228**(16), 5838 – 5866.

Press, William H., Teukolsky, Saul A., Vetterling, William T., & Flannery, Brian P. 1992. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press.

Prosperetti, Andrea, & Tryggvason, Grétar. 2007. *Computational Methods for Multiphase Flow*. Cambridge University Press.

Quinn, Michael J. 2003. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group.

Rayleigh, Lord. 1879. On the Capillary Phenomena of Jets. *Proceedings of the Royal Society of London*, **29**, 71–97.

Renardy, Yuriko, & Renardy, Michael. 2002. PROST: A Parabolic Reconstruction of Surface Tension for the Volume-of-Fluid Method. *Journal of Computational Physics*, **183**(2), 400 – 421.

Rider, William J., & Kothe, Douglas B. 1998. Reconstructing Volume Tracking. *Journal of Computational Physics*, **141**(2), 112 – 152.

Roenby, J., Bredmose, H., & Jasak, H. 2016. A Computational Method for Sharp Interface Advection. *ArXiv e-prints*, Jan.

Ross, Sheldon. 2013. Chapter 12 - Markov Chain Monte Carlo Methods. *Pages 271 – 302 of:* Ross, Sheldon (ed), *Simulation (Fifth Edition)*, fifth edition edn. Academic Press.

Rudman, Murray. 1997. Volume-Tracking Methods for Interfacial Flow Calculations. *International Journal for Numerical Methods in Fluids*, **24**(7), 671–691.

Saad, Y. 2003. *Iterative Methods for Sparse Linear Systems*. 2nd edn. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Samet, Hanan. 1984. The Quadtree and Related Hierarchical Data Structures. *ACM Comput. Surv.*, **16**(2), 187–260.

Samkhaniani, N., & Ansari, M. R. 2016. Numerical simulation of bubble condensation using CF-VOF. *Progress in Nuclear Energy*, **89**(MAY), 120–131.

Seibert, Helmut, Hildenbrand, Dietmar, Becker, Meike, & Kuijper, Arjan. 2010. Estimation of Curvatures in Point Sets Based on Geometric Algebra. *Pages 12–19 of: VISIGRAPP 2010. Proceedings.*

Shakarji, Craig M. 1998. Least-Squares Fitting Algorithms of the NIST Algorithm Testing System. *Pages 633–641 of: Journal of Research of the National Institute of Standards and Technology.*

Squyres, Jeffrey M. 2012. *The Archicture of Open Source Applications.* Vol. ii. Self published. Chap. 15.

Stroustrup, Bjarne. 2013. *The C++ Programming Language.* 4th edn. Addison-Wesley Professional.

Sussman, M., Smereka, P., & Osher, S. 1994. A Level Set Approach for Computing Solutions to Incompressible Two-phase Flows. *Journal of Computational Physics*, **114**, 146–159.

Sussman, M., Smith, K.M., Hussaini, M.Y., Ohta, M., & Zhi-Wei, R. 2007. A sharp interface method for incompressible two-phase flows. *Journal of Computational Physics*, **221**(2), 469 – 505.

Sussman, Mark. 2003. A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *Journal of Computational Physics*, **187**(1), 110 – 136.

Sussman, Mark, & Fatemi, Emad. 1999. An Efficient, Interface-Preserving Level Set Redistancing Algorithm and Its Application to Interfacial Incompressible Fluid Flow. *SIAM J. Sci. Comput.*, **20**(4), 1165–1191.

Sussman, Mark, & Ohta, Mitsuhiro. 2009. A Stable and Efficient Method for Treating Surface Tension in Incompressible Two-Phase Flow. *SIAM Journal on Scientific Computing*, **31**(4), 2447–2471.

Sussman, Mark, & Puckett, Elbridge Gerry. 2000. 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.

Tryggvason, G., Bunner, B., Esmaeeli, A., Juric, D., Al-Rawahi, N., Tauber, W., Han, J., Nas, S., & Jan, Y.-J. 2001. A front tracking method for the computations of multiphase flow. *J. Comput. Phys.*, **169**, 708–759.

Tryggvason, Grétar, Scardovelli, Ruben, & Zaleski, Stéphane. 2011. *Direct Numerical Simulations of Gas-Liquid Multiphase Flows*. Cambridge, New York: Cambridge University Press.

Turek, S. 1999. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approache*. Lecture notes in computational science and engineering. Springer-Verlag.

Ubbink, O, & Issa, RI. 1999. A method for capturing sharp fluid interfaces on arbitrary meshes. *Journal of Computational Physics*, **153**(1), 26–50.

Ubbink, Onno. 1997. *Numerical prediction of two fluid systems with sharp interfaces*. PhD Thesis, Imperial College, London.

Unverdi, Salih Ozen, & Tryggvason, Grétar. 1992. A front-tracking method for viscous, incompressible, multi-fluid flows. *Journal of Computational Physics*, **100**(1), 25 – 37.

van Leer, Bram. 1974. Towards the ultimate conservative difference schemes, II. Monotonicity and conservation combined in a second order scheme. *Journal of Computational Physics*, **14**, 361–367.

van Wachem, B. G. M., & Schouten, J. C. 2002. Experimental validation of 3-D lagrangian VOF model: Bubble shape and rise velocity. *AIChE Journal*, **48**(12), 2744–2753.

Verteege, H. K., & Malalasekera, W. 2007. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. 2nd edn. SPearson, Prentice Hall.

Wadell, Hakon. 1935. Volume, Shape, and Roundness of Quartz Particles. *The Journal of Geology*, **43**(3), 250–280.

Wardle, Kent E., & Weller, Henry G. 2013. Hybrid Multiphase CFD Solver for Coupled Dispersed/Segregated Flows in Liquid-Liquid Extraction. *International Journal of Chemical Engineering*.

Weller, H. G., Tabor, G., Jasak, H., & Fureby, C. 1998. A Tensorial Approach to Computational Continuum Mechanics Using Object-oriented Techniques. *Computers in Physics*, **12**(6), 620–631.

Weller, Henry G. 2008 (May). *A New Approach to VOF-based Interface Capturing Methods for Incompressible and Compressible Flow*. Tech. rept. TR/HGW/04. OpenCFD Ltd., United Kingdom.

Williams, M. W., Kothe, D. B., & Puckett, E. G. 1998. Accuracy and Convergence of Continuum Surface Tension Models. *Pages 294–305 of: Fluid Dynamics at Interfaces.* Univ. Press.

Williams, Matthew Wayne. 2000. *Numerical methods for tracking interfaces with surface tension in 3-D mold-filling processes.* PhD Thesis, University of California, Davis.

Yang, Pinghai, & Qian, Xiaoping. 2007. Direct Computing of Surface Curvatures for Point-Set Surfaces. *Pages 29–36 of: SPBG'07.*

Zalesak, Steven T. 1979. Fully multidimensional flux-corrected transport algorithms for fluids. *Journal of Computational Physics*, **31**(3), 335 – 362.