Numerical Simulation of Immiscible Fluids with FEM Level Set Techniques

Dissertation
Doktors der Naturwissenschaften

Dem Fachbereich Mathematik der Universität Dortmund
vorlegt am 2007 von

Shu-Ren Hysing

Abstract

Hysing, S. 2007. Numerical simulation of immiscible fluids with FEM level set techniques. Doctoral dissertation.

Multiphase flows, including free surface and two-phase flows, are commonly encountered in many industrial applications. Effects of wave phenomena are for example important when designing boats, drop formation is essential for ink jet printers, and bubbles may come into play in chemical reactors and heat exchangers.

Numerical simulation of these phenomena is a complex and challenging task. The desire to have both high accuracy and computational speed often stands in direct contradiction to each other. The aim of this thesis was to describe a suitable methodology with potential to be both very accurate and also efficient. High resolution benchmarks were also developed in order to validate and quantify the performance of a code (TP2D) developed according to the presented approach.

The developed methodology combined a non-conforming finite element discretization with the level set method for tracking the interfaces. A semi-implicit approach to implementing surface tension forces was also derived, which allowed for significantly larger time steps in comparison with the traditional explicit approach.

The benchmarks were used to compare the developed code with two commercial CFD codes (Comsol Multiphysics and Ansys Fluent). The commercial codes did not show strong convergence towards the reference solution, in contrast to TP2D which was both faster and significantly more accurate. TP2D even computed a more accurate solution on the very coarsest grid compared to the best results of the commercial codes.

Keywords: two-phase flow, finite element method, interface tracking, level set method, benchmarking.

Knowing is not enough; we must apply.
Willing is not enough; we must do.

Bruce Lee

Acknowledgments

First, I would like to thank my main supervisor, Stefan Turek, for giving me the opportunity of immersing myself in applied high performance computing as a PhD student. Thank you for your confidence in me, optimism, patience, and showing a real passion for the field of numerical computation. I also thank Dmitri Kuzmin for very helpful discussions and guidance on the topic of convective stabilization techniques amongst others.

Thanks should also go to Nicola Parolini, Erik Burman, Sashikumaar Ganesan, and Lutz Tobiska for all their contributions and also patience with respect to the bubble benchmarks. Additionally all my colleagues at the institute for applied mathematics and numerics at the University of Dortmund should be thanked for their invaluable help and support.

It should be acknowledged that this work has been made possible by the financial contributions from the German Research foundation (DFG) under grants Paketantrag PAK178 (Tu102/27-1, Ku1530/5-1) and Sonderforschungsbereich SFB708 (TP B7).

Finally, I would like to deeply thank my family, in particular my mother Margareta Hysing and my father Wong Mau, for always believing in me, and teaching me independence and to trust in myself. And last but not least I thank my sister Shu-Chin who without fail always has been there for me.

- Shu-Ren

Contents

Chapter 1

Introduction

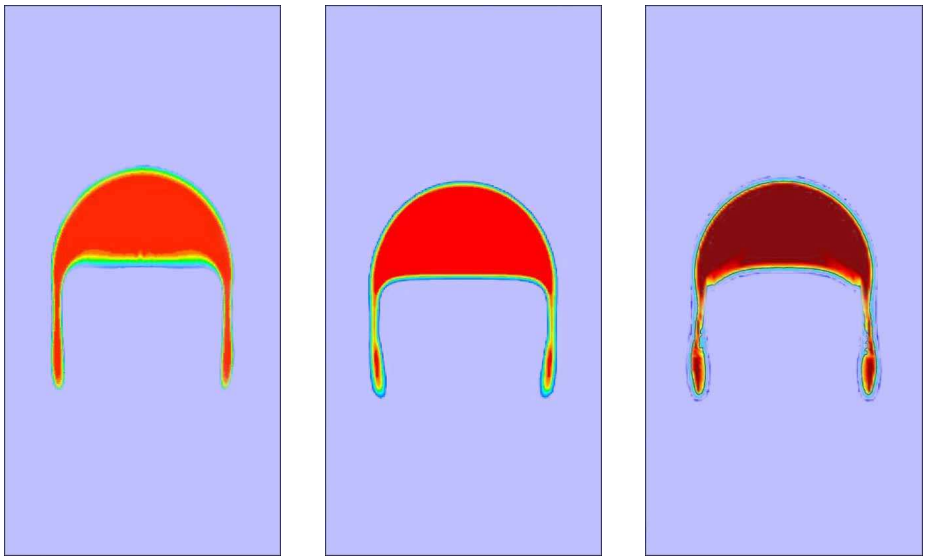
Fluid flow plays a major role in each and everyones daily lives, from the essential blood flowing through our veins and air breathed into our lungs, to the trivial cup of coffee and squeeze on the tube of toothpaste in the mornings. As long as these processes work we rarely give them a moments notice, it is only when the cup of coffee goes missing one morning that we start to ask questions. There are however many applications and processes for which it, due to for example safety reasons, is necessary to know and to be able to accurately predict the behavior of the involved fluids. Some examples of such vital processes include the flow of reactants in chemical gas-liquid reactors, boiling and movement of gases in reactors for power generation such as in nuclear reactors, and micro-capillary flow in biomedical devices.

Many of the previously mentioned examples can and do concern the flow of immiscible fluids, that is fluids which are incapable of mixing, such as oil and water. These types of flows are generally known as multiphase flows, of which the most common configuration involves two distinct fluids, so called two-phase flows. The different fluids are separated by a very thin interface region where surface tension effects and mass transfer due to chemical reactions may appear. Surface tension is caused by molecular force imbalances in the vicinity of the fluid interfaces. A molecule deep in the core of the fluid will experience a zero net force due to all surrounding molecules, whereas a molecule in the interface region is "missing" some neighbors, resulting in a net force imbalance.

To explain and to predict the behavior of fluids, one is faced with two choices: Firstly, one can make experiments and measurements on either a model scale or in full scale. This approach should ideally be able to give very accurate predictions if one is able to capture all relevant details. A major drawback of the experimental approach is that it often is prohibitively expensive and time consuming, and it sometimes even can be completely impossible or too dangerous to measure what one is after. This could for example be to examine and quantify details of the fuel ignition process inside a combustion engine. The second alternative then, is to resort to intellectual reasoning whereby suitable assumptions are made regarding the involved physical processes which then can be modeled by mathematical relations. The arising equation systems are usually very complex and can only in very rare cases be solved analytically. They will

more commonly be discretized and approximated numerically with subsequent application of appropriate solution techniques.

There exists a multitude of algorithms and methods to choose from for numerical simulation of immiscible fluids. Before making a choice and committing to a specific methodology or software it is important to ask what the quantities of interests are and how much accuracy will be required. Since one often has a very specific process or application to simulate in mind, the answers to these questions should really come rather easily. Unfortunately it is very hard to find dedicated scientific studies providing benchmarking, reference data, and selection criteria on which to apply these answers. This author can only count 11 such relevant publications pertaining to four different benchmark configurations [18, 43, 46, 51, 52, 53, 54, 68, 107, 108, 112]. Thus one is still more or less standing in the dark. With this in mind, it is the aim of this thesis to highlight an appropriate algorithm, capable of handling a wide spectrum to two-phase flow problems and investigate how to optimally modify it for both accuracy and computational efficiency. In addition to this, benchmark configurations for strict quantitative evaluation and comparison of interfacial flow codes will be presented and thoroughly investigated.



(a) Ansys CFX (b) Ansys Fluent (c) Comsol Multiphysics

Figure 1.1: Solution snapshots of a rising bubble computed with three different commercial CFD codes.

To get a feeling for what the current state of the art is and what modern computational fluid dynamics (CFD) tools are capable of let's briefly look at an example, in this case an initially circular air bubble rising in a water column. If the bubble is large and surface tension forces are not completely dominating, there will be significant topology change of the interface, which may even break up into smaller parts. Figure 1.1 depicts the solution of such a test case computed with the commercial software codes Ansys CFX, Ansys Fluent, and

Comsol Multiphysics. All three solutions show the same overall behavior, a significantly deformed shape with thin elongated trailing filaments. If one takes a closer look however, it is evident that the solutions differ somewhat, in particular with respect to the shape of the filaments. From a physical viewpoint the solutions are also questionable since such thick and elongated filaments rarely if ever appear in nature and experiments. It is more likely that the filaments should break off and form trailing satellite droplets. Let's leave this for now but come back to this discussion towards the end of the thesis.

This thesis is organized as follows. Chapter 2 discusses mathematical modeling of fluid flows with particular emphasis on two-phase flows. Chapter 3 continues with explaining the numerical discretization procedure of the derived equation systems and highlighting appropriate solution techniques. Interface tracking techniques with emphasis on the level set method and related algorithms is described in Chapter 4. This is followed with a chapter devoted to implementation of surface tension effects (Chapter 5), where an efficient method allowing for large time steps while still being fully implicit with respect to all interfaces is derived. Benchmark test cases for quantitative validation and comparison of two-phase flow codes are presented in Chapter 6 together with extensive studies, conducted in collaboration with two other research groups, which provide reference target values of the measured quantities. The last chapter of this thesis (Chapter 7) uses the established benchmarks to compare an academic code, based on the methodology described in Chapters 2–5, with two commercial CFD tools. The comparison indicates that although the commercial tools deliver plausible solutions in the picture norm, they leave a lot to desire with respect to accuracy and thus also overall computational efficiency. The thesis is concluded with a discussion on methodologies which may have a future potential to improve both accuracy and efficiency for numerical simulation of immiscible multiphase flows. Appendix A supplements the thesis by practically describing a suitable solver structure for the developed methodology.

Chapter 2

Mathematical modeling

To model physical phenomena such as heat transfer, wave propagation, structural stresses, and of course fluid dynamics, it is very common to pose the involved problems as partial differential equations. These equations can quite elegantly be derived by examining the rate of change of the involved quantities in an infinitesimal volume.

2.1 Conservation laws

Consider for the moment a prototypical property or quantity q, existing in an arbitrary domain Ω. For this quantity to be conserved within Ω, the rate of change of q must be equal to the normal flux crossing the total surface S (let $\hat{\mathbf{n}}$ denote the unit normal to S). With flux meaning the transport of q, in and out of Ω, with a vector field $\mathbf{u}$. Any generation from directional surface sources $\boldsymbol{Q}_S$ and scalar internal sources Q_V also contributes to an increased (or decreased) amount of q. Figure 2.1 illustrates these phenomena for the region Ω.

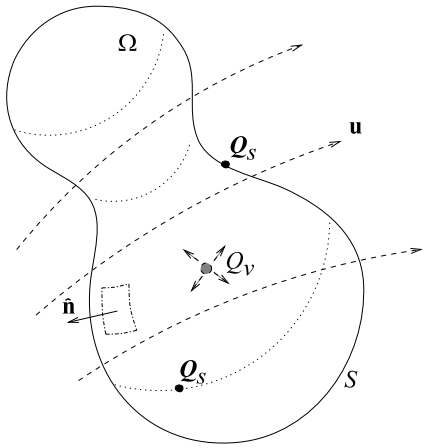


Figure 2.1: Illustration of a general conservation law.

A balance equation explicitly stating the conservation of q will have the form

$$\frac{\text{net increase of } q}{\text{time}} = \frac{\text{influx}}{\text{time}} + \frac{\text{internal generation}}{\text{time}} + \frac{\text{surface generation}}{\text{time}}$$

which is easy to translate into mathematical vector notation

$$\frac{\partial}{\partial t}\int_\Omega q\, d\Omega = -\int_S q\mathbf{u}\cdot\hat{\mathbf{n}}\, dS + \int_\Omega Q_V\, d\Omega - \int_S \boldsymbol{Q}_S\cdot\hat{\mathbf{n}}\, dS.$$

Application of Gauss' divergence theorem

$$\int_S \mathbf{F}\cdot\hat{\mathbf{n}}\, dS = \int_\Omega \nabla\cdot\mathbf{F}\, d\Omega$$

to the surface terms results in the following reformulation

$$\frac{\partial}{\partial t}\int_\Omega q\, d\Omega = -\int_\Omega \nabla\cdot q\mathbf{u}\, d\Omega + \int_\Omega Q_V\, d\Omega - \int_\Omega \nabla\cdot\boldsymbol{Q}_S\, d\Omega. \tag{2.1}$$

Let us consider the conservation of mass by substituting q with the density ρ. Under the assumption that the system is isolated it is possible to disregard any generation from internal and surface sources. What is left is the following balance equation

$$\frac{\partial}{\partial t}\int_\Omega \rho\, d\Omega = -\int_\Omega \nabla\cdot(\rho\mathbf{u})\, d\Omega$$

which must hold for a volume of any size. Thus, in the limit $\Omega \to 0$ the well known mass conservation or continuity equation will appear, which reads

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

Moving on to conservation of linear momentum which, from Newton's second law, is the amount of mass transported with a certain velocity. Specifically, momentum transport per unit volume is therefore expressed as the density ρ transported with the velocity field $\mathbf{u}$. If now the product of these quantities is substituted for q in the general conservation equation (2.1) then the flux of momentum over the boundaries will accordingly be represented by the quantity $\rho\mathbf{uu}$. Moreover, momentum is created and destroyed by forces acting internally and on the surface of Ω. Inside of Ω momentum is generated by the presence of forces, here labeled $\mathbf{f}$, while on the surface momentum is created by both pressure p and shear stresses, which combined are represented by the stress tensor $\boldsymbol{\tau}$. Having made these observations enables the following formulation for the conservation of momentum

$$\frac{\partial}{\partial t}\int_\Omega \rho\mathbf{u}\, d\Omega = -\int_\Omega \nabla\cdot(\rho\mathbf{uu})\, d\Omega + \int_\Omega \mathbf{f}\, d\Omega + \int_\Omega \nabla\cdot\boldsymbol{\tau}\, d\Omega.$$

This should again be valid for a volume of any size or shape with the consequential removal of the integral operations, thus yielding the partial differential equation

$$\frac{\partial}{\partial t}(\rho\mathbf{u}) = -\nabla\cdot(\rho\mathbf{uu}) + \mathbf{f} + \nabla\cdot\boldsymbol{\tau}.$$

The stress tensor $\boldsymbol{\tau}$ can, as stated above, be subdivided into two parts, the static pressure p and the shear stresses $\boldsymbol{\sigma}$. Consider for the moment Ω in the form of a unit cube where the coordinate axes are aligned with the surface normals. For a given coordinate axis, the pressure term will now only have one component which is acting directly against the surface normal, while the shear stress terms will have three different components, one in each coordinate direction. This enables a splitting of $\boldsymbol{\tau}$ into the two terms $p\mathbf{I}$ and $\boldsymbol{\sigma}$, that is

$$\frac{\partial}{\partial t}(\rho\mathbf{u}) = -\nabla\cdot(\rho\mathbf{u}\mathbf{u}) + \mathbf{f} + \nabla\cdot(-p\mathbf{I} + \boldsymbol{\sigma}) \tag{2.3}$$

where $\mathbf{I}$ denotes the diagonal unit matrix.

2.2 Navier-Stokes equations

Fluids can generally be classed as either Newtonian, such as plain water, or non-Newtonian for which the viscosity changes with the applied stress (for example toothpaste, corn starch solutions, and blood). For Newtonian fluids the shear stresses are assumed to be proportional to the velocity gradients, which can be expressed as

$$\sigma_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

where μ is the dynamic or molecular viscosity of the fluid.

Under the assumption that the fluid also is incompressible, meaning that the density ρ is constant, the continuity equation (2.2) reduces to $\nabla\cdot\mathbf{u} = 0$. By using this identity, the first term on the right hand side of the momentum equation (2.3) can be rewritten as $\nabla\cdot(\rho\mathbf{u}\mathbf{u}) = \mathbf{u}\cdot\nabla(\rho\mathbf{u}) + \rho\mathbf{u}(\nabla\cdot\mathbf{u}) = \rho(\mathbf{u}\cdot\nabla)\mathbf{u}$. After a little rearranging, what is left is what commonly is referred to as the Navier-Stokes equations

$$\rho\left(\frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) = -\nabla p + \nabla\cdot\left(\mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right) + \mathbf{f} \tag{2.4}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{2.5}$$

which mathematically describe flows of incompressible Newtonian fluids.

The convective term in (2.4) is nonlinear in $\mathbf{u}$ and makes the equations particularly challenging to solve, which also is why existence and uniqueness of a solution has yet to be proved. This proof is in fact so elusive that the Clay Mathematics Institute has announced it as one of the seven Millennium Prize Problems (awarding $1 million for its solution) [20]. The Navier-Stokes equations have despite these difficulties been used extensively and successfully to model and predict a multitude of fluid engineering applications, from optimizing flow around the Alinghi America's Cup racing boat [79] and Formula 1 racing cars [2, 6], to designing micro- and nano-scale devices such as bio-chips and miniature fuel cells [27, 63].

2.3 Boundary and initial conditions

The modeled applications and processes are usually very specific in nature and the studies can thus be confined to a smaller spatial sub region or domain $\Omega \subset \mathbb{R}^d$ and a specific temporal extent $[0, T]$. Both these restrictions will usually greatly simplify the modeling and also reduce the effort required to obtain a solution. It is particularly advantageous to utilize all existing symmetry axes to further shrink the computational domain. Sometimes it is also possible to transform a three dimensional model to two dimensions if the problem can be considered axisymmetric. The drawback of restricting the models spatially is that the equations now have to be supplied with suitable boundary conditions, which are supposed to describe all interactions between Ω and the rest of the non-modeled environment.

Boundary conditions usually come in two distinct variants, Dirichlet conditions which fixes the value of a quantity, and Neumann, also called natural, boundary conditions which specify the in- or out-flux. When modeling fluid flow, Dirichlet conditions uniquely set the velocities on a boundary, that is

$$\mathbf{u} = \mathbf{g} \quad on \quad \partial\Omega_D.$$

These conditions are usually prescribed at inlets to set the inflow velocities or at impermeable walls to fix the velocity of the fluid to that of the wall (which usually has zero velocity). One can alternatively prescribe the stress at a boundary with the following natural boundary condition

$$\hat{\mathbf{n}} \cdot \left(-p\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right) = h \quad on \quad \partial\Omega_N$$

where $\hat{\mathbf{n}}$ is the outward pointing unit normal. The special case of a homogenous Neumann condition ($h = 0$), or zero stress condition, is commonly used as an outflow boundary condition. A combination of Dirichlet and Neumann conditions may be used to specify a so called free-slip or symmetry condition

$$\hat{\mathbf{n}} \cdot \mathbf{u} = 0, \qquad \hat{\mathbf{t}} \cdot \hat{\mathbf{n}} \cdot \left(\mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right) = 0 \quad on \quad \partial\Omega_S.$$

Initial conditions, in the form of specified velocities $\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x})$, must be prescribed in addition to the boundary conditions for applications where the temporal evolution is of interest. For stationary calculations it is also advantageous to specify a good initial guess so that the nonlinear solver will converge faster. The equations together with fluid parameters, domain, boundary conditions, and initial conditions all together now uniquely specify the model problem.

2.4 Two-phase flows

The special case of modeling two-phase flows can be treated quite similar to single phase flows. The fact that two unique fluids are present can be handled either by using two separate sets of the Navier-Stokes equations, one for each fluid, or more conveniently using the same set of equations for both fluids but with variable density and viscosity fields, that is

$$\begin{aligned} \rho(\mathbf{x})\left(\frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) &= -\nabla p + \nabla\cdot\left(\mu(\mathbf{x})(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right) + \mathbf{f} \\ \nabla\cdot\mathbf{u} &= 0 \end{aligned}$$

where now $\rho(\mathbf{x})$ and $\mu(\mathbf{x})$ depend on the spatial coordinate $\mathbf{x} \in \Omega$. The latter approach requires a method to identify where each fluid is so that the correct value of the density and viscosity is chosen at each point. This will later on be referred to as interface tracking techniques.

Assuming that the two fluids are immiscible, there will exist interfaces $\Gamma := \partial\Omega_1 \cap \partial\Omega_2$ separating the different fluids or phases (see Figure 2.2). On these interfaces internal boundary conditions have to be prescribed to link the different fluids together. For incompressible fluids which do not lose or gain mass (for example, due to chemical reactions), the normal velocity on both sides of the interface must be equal

$$\hat{\mathbf{n}} \cdot \mathbf{u} = 0|_\Gamma.$$

Here $\hat{\mathbf{n}}$ is the unit normal at the interface pointing into Ω_1 and $[\mathsf{A}]|_\Gamma = A|_{\Omega_1 \cap \Gamma} - A|_{\Omega_2 \cap \Gamma}$ denotes the jump of a quantity A across the interface.

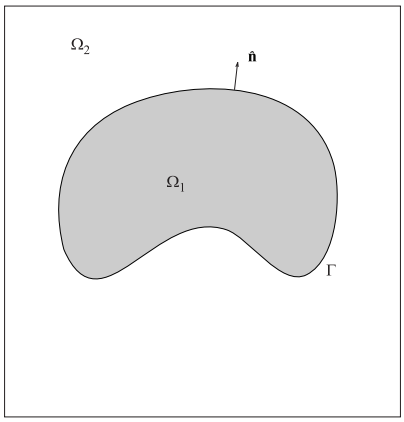


Figure 2.2: Illustration of a typical two-phase flow configuration.

Additionally the internal forces must, due to Newton's third law, be in equilibrium. However, as pointed out in the introduction, molecular force imbalances will arise between the two fluids. An additional force must thus exist to reestablish the equilibrium, and it is this force which is usually called surface tension.

2.5 Surface tension

Surface tension forces work towards minimizing the surface area of an interface. There are in general two ways to integrate surface tension effects into the Navier-Stokes equations. One can apply the force balance boundary condition

$$[-p\mathbf{I} + \mu(\nabla\mathbf{u} + (\nabla\mathbf{u})^T)]\big|_\Gamma \cdot \hat{\mathbf{n}} = \sigma\kappa\hat{\mathbf{n}}$$

directly to the interface Γ. This states that the difference between the combination of pressure and viscous forces must be equal to the product of a coefficient σ and the curvature of the interface κ. The surface tension coefficient σ mainly depends on the two fluids in contact, but can also vary with temperature and

chemical impurities on the interfaces (so called surfactants). In the following σ will be assumed to be constant.

The internal force boundary condition can also be rewritten as a volumetric force, and then takes the form

$$\mathbf{f}_{st} = \sigma\kappa\hat{\mathbf{n}}\delta(\Gamma, \mathbf{x}) \tag{2.6}$$

where $\delta(\Gamma, \mathbf{x})$ is the Dirac delta function localizing the surface tension force to the interface. This approach is generally favored by the research community since, from a numerical viewpoint, its inclusion as an explicit right hand side source term is quite straight forward. A detailed derivation of the interface conditions can be found in [94].

When an interface is in contact with a wall or solid boundary a so called contact angle is formed. This angle depends on several variables; the two fluids, the material and roughness of the solid boundary, and the magnitude and sign of the contact line velocity. Modeling of contact angle dynamics is complex and will not be discussed further. The interested reader is referred to references [35, 61, 93].

2.6 Nondimensionalization

Dimensionless numbers help to identify which physical effects are dominating, and also assist when classifying different model problems. The introduction of the length scale L, characteristic velocity U, and the subsequent scaling of the involved variables

$$x_i' = \frac{x_i}{L}\ , \quad \mathbf{u}' = \frac{\mathbf{u}}{U}\ , \quad p' = \frac{p}{\rho U^2}\ , \quad t' = \frac{tU}{L}$$

results in the following dimensionless form of the Navier-Stokes equations

$$\begin{aligned} \rho\left(\frac{U^2}{L}\frac{\partial \mathbf{u}'}{\partial t'} + \frac{U^2}{L}(\mathbf{u}'\cdot\nabla')\mathbf{u}'\right) &= -\frac{\rho U^2}{L}\nabla' p' + \nabla'\cdot\frac{U}{L^2}\left(\mu(\nabla'\mathbf{u}' + \nabla'\mathbf{u}'^T)\right) + \mathbf{f} \\ \nabla'\cdot\mathbf{u}' &= 0. \end{aligned}$$

Multiplication with the factor $L/\rho U^2$ and rearranging gives

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} &= -\nabla p + \nabla\cdot\left(\frac{\mu}{\rho U L}(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right) + \frac{L}{\rho U^2}\mathbf{f} \\ \nabla\cdot\mathbf{u} &= 0 \end{aligned} \tag{2.7}$$

where the prime notation has been dropped. The Reynolds number can now be identified as the inverse of the nondimensional group in front of the viscous term, that is

$$Re = \frac{\rho U L}{\mu}.$$

This scale factor is the most commonly used nondimensional number in fluid dynamics and relates inertial forces to viscous forces. A high Re number indicates that the nonlinear convective terms dominate over the viscous terms, and vice versa a low Reynolds number indicates that the flow is viscous and laminar.

The right hand side source term $\mathbf{f}$ in (2.7) includes all imposed forces of which two major effects relevant to two-phase flow applications will be considered, gravitational and surface tension forces. The gravitational and volume force formulation of the surface tension forces (2.6) result in the following source term

$$\mathbf{f} = \rho g + \sigma\kappa\hat{\mathbf{n}}\delta(\Gamma, \mathbf{x})$$

which with the introduced scaling yields

$$\mathbf{f}' = \frac{Lg}{U^2} + \frac{\sigma}{\rho U^2 L}\kappa'\hat{\mathbf{n}}\delta(\Gamma, \mathbf{x}).$$

Here it has been used that κ is proportional to $\partial^2 f/\partial x_i^2$ and thus scales as $\kappa' = L^2\kappa$. Two additional nondimensional groups can now be identified, namely the Froude and Weber numbers

$$Fr = \frac{U^2}{gL}, \qquad We = \frac{\rho U^2 L}{\sigma}$$

which relate inertial to gravitational effects (g is the gravitational constant) and inertial to surface tension effects. Another helpful nondimensional number which will be used further on is the Eötvös number, which is defined as

$$Eo = \frac{\rho g L^2}{\sigma}.$$

The Eötvös number is used to characterize the shapes of bubbles and drops by relating buoyancy forces to surface tension forces. One example of using these numbers to classify test configurations is the classical diagram of Clift, Grace, and Weber (see Figure 2.3), which predicts bubble shapes as a function of bubble Re and Eo numbers [19].

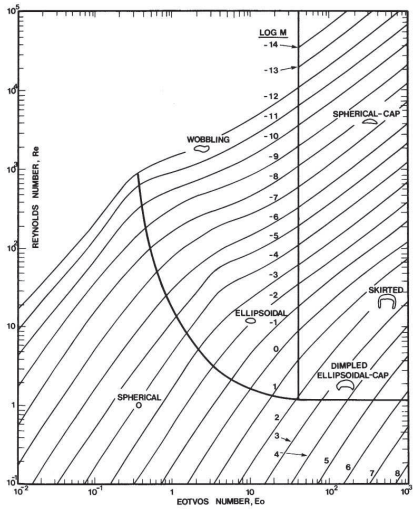


Figure 2.3: Shape regimes for bubbles and drops (from Clift et al. [19]).

2.7 General problem formulation

Assume that incompressible Newtonian flow of immiscible fluids in a d-dimensional region Ω is to be modeled. The task is then, given $\mathbf{u}_0 = \mathbf{u}(\mathbf{x}, 0)$ and $p_0 = p(\mathbf{x}, 0)$, to find the unknown velocity field $\mathbf{u}(t)$ and pressure $p(t)$ through solving

$$\begin{cases} \rho(\mathbf{x})\left(\dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) = -\nabla p + \nabla\cdot\left(\mu(\mathbf{x})(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right) + \mathbf{f} & \text{in } \Omega, \\ \nabla\cdot\mathbf{u} = 0 & \text{in } \Omega, \\ \mathbf{u} = \mathbf{g} & \text{on } \partial\Omega_D, \\ \hat{\mathbf{n}}\cdot\left(-p\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right) = h & \text{on } \partial\Omega_N, \\ [\hat{\mathbf{n}}\cdot\mathbf{u}]|_\Gamma = 0 & \text{on } \Gamma, \\ [-p\mathbf{I} + \mu(\nabla\mathbf{u} + (\nabla\mathbf{u})^T)]\big|_\Gamma\cdot\hat{\mathbf{n}} = \sigma\kappa\hat{\mathbf{n}} & \text{on } \Gamma. \end{cases}$$

The external boundary $\partial\Omega$ consists of two parts, the Dirichlet boundaries $\partial\Omega_D$, where the velocity field is uniquely specified, and Neumann boundaries $\partial\Omega_N$. Internally there may exist an arbitrary number of interfaces which are included in Γ.

The weak or variational form of the posed problem shall now be derived. First define a scalar product

$$(\mathbf{u}, \mathbf{w}) = \int_\Omega \mathbf{u}\cdot\mathbf{w}\, d\Omega = \int_\Omega u_1 w_1 + \ldots + u_d w_d\, d\Omega$$

and introduce a set of arbitrary test functions which constitutes a vector field $\mathbf{v} = [v_1\, v_2 \ldots v_d]$, where subscripts $1, \ldots, d$ denote the space dimension in question. Application of the test functions to the momentum equations, through the scalar product, yields

$$\begin{aligned} &\int_\Omega \rho(\mathbf{x})\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right)\cdot\mathbf{v}\, d\Omega = \\ &\quad -\int_\Omega \nabla p\cdot\mathbf{v}\, d\Omega + \int_\Omega \nabla\cdot\left(\mu(\mathbf{x})(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right)\cdot\mathbf{v}\, d\Omega + \int_\Omega \mathbf{f}\cdot\mathbf{v}\, d\Omega. \end{aligned} \quad (2.8)$$

By incorporating the pressure into the stress tensor in (2.8) and using the Gaussian theorem the term for the total stress can be written as

$$\begin{aligned} &\int_\Omega \nabla\cdot\left(-p\mathbf{I} + \mu(\mathbf{x})(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right)\cdot\mathbf{v}\, d\Omega = \\ &\qquad = -\int_\Omega \left(-p\mathbf{I} + \mu(\mathbf{x})(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right)\cdot\nabla\mathbf{v}\, d\Omega + \\ &\qquad\quad + \int_{\partial\Omega} \hat{\mathbf{n}}\cdot\left(-p\mathbf{I} + \mu(\mathbf{x})(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right)\cdot\mathbf{v}\, dS = \\ &\qquad = -\int_\Omega \left(-p\mathbf{I} + \mu(\mathbf{x})(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right)\cdot\nabla\mathbf{v}\, d\Omega + \int_{\partial\Omega_N} h\cdot\mathbf{v}\, dS \end{aligned}$$

where the Neumann boundary condition has been substituted into the boundary integral which arose naturally due to the partial integration. An advantage of

using the weak formulation in this way is that it reduces the requirements on the approximating functions to be once instead of twice differentiable.

The boundary conditions for the surface tension forces can be incorporated similarly to the Neumann boundary conditions but will in the following be included into the force term $\mathbf{f}$ as per (2.6). The weak formulation of the problem now reads:

Find $\mathbf{u}(t)$ and $p(t)$ such that

$$\left(\rho(\mathbf{x})\frac{\partial \mathbf{u}}{\partial t}, \mathbf{v}\right) + a\left(\mathbf{u}, \mathbf{u}, \mathbf{v}\right) - b\left(p, \mathbf{v}\right) + b\left(q, \mathbf{u}\right) = \left(\mathbf{f}, \mathbf{v}\right) + f(h, \mathbf{v}) \tag{2.9}$$

for velocity and pressure test functions, $\mathbf{v}$ and q, chosen in appropriate function spaces. Here the convective and diffusive terms are represented as

$$a\left(\mathbf{w}, \mathbf{u}, \mathbf{v}\right) = \int_{\Omega} \rho(\mathbf{x})(\mathbf{w} \cdot \nabla)\mathbf{u} \cdot \mathbf{v} + \mu(\mathbf{x})(\nabla \mathbf{u} + \nabla \mathbf{u}^T) \cdot \nabla \mathbf{v} \, d\Omega,$$

and the linear forms as

$$\begin{aligned} b\left(q, \mathbf{v}\right) &= \int_{\Omega} q \nabla \cdot \mathbf{v} \, d\Omega, \\ f(g, \mathbf{v}) &= \int_{\partial \Omega_N} g \cdot \mathbf{v} \, dS. \end{aligned}$$

The weak formulation of the Navier-Stokes equations (2.9) will, after time discretization, form the basis of the spatial finite element discretization presented in Chapter 3.

Chapter 3

Numerical treatment

Analytical solutions of the Navier-Stokes equations are very hard to come by, even for quite simple configurations. Take for example flow in a straight and perfectly smooth circular pipe, if the flow rate is low, the fluid behaves predictably and is classed as laminar. In this case it is possible to derive an analytical solution, the so called Hagen-Poiseuille flow profile [114]. However, as the flow rate increases perturbations start to appear and the flow becomes increasingly irregular. This chaotic and unpredictable flow behavior is what is usually meant by turbulence, a flow state for which no exact analytical solutions can be found without resorting to drastic simplifications.

To obtain general solutions to the Navier-Stokes equations one more commonly must apply numerical techniques. The equations are discretized both spatially into smaller subregions and temporally into time slices, which leads to a large set of coupled subproblems each of which should satisfy the discrete equations. This can in general be represented as a large matrix system which must be inverted in order to obtain the corresponding discrete and approximate solutions.

In the following chapter, the issues of appropriate discretization and subsequent solution techniques for the Navier-Stokes equations with emphasis on two-phase flow applications shall be addressed. The presented methodology builds on and shares many concepts with the FeatFlow software suite (short for Finite Element Analysis Tools for Flow) which has been developed to efficiently, robustly, and accurately simulate single phase laminar fluid flow [7, 102].

3.1 Temporal discretization

The first step in the numerical discretization process is to choose an appropriate time stepping scheme. It should not only be accurate in time, but also easy to realize and computationally robust and affordable (inexpensive) to use. A simple and flexible choice is the θ-scheme approach, which allows for the use of the single step Backward Euler and Crank-Nicolson schemes, and also multi-step schemes such as the strongly A-stable Fractional-step-θ-scheme. The θ-scheme applied to the Navier-Stokes equations results in the following general semi-discrete system for each time step:

Given $\mathbf{u}^n$ at time $t = t^n$ and time step $\Delta t = t^{n+1} - t^n$, then solve for $\mathbf{u} = \mathbf{u}^{n+1}$ and $p = p^{n+1}$

$$\begin{aligned} \rho\frac{\mathbf{u}-\mathbf{u}^n}{\Delta t} + \theta\left[\rho(\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla\cdot\left(\mu(\nabla\mathbf{u} + (\nabla\mathbf{u})^T)\right)\right] + \nabla p &= \mathbf{b}^{n+1} \\ \nabla\cdot\mathbf{u} &= 0 \end{aligned} \tag{3.1}$$

with right hand side given by

$$\begin{aligned} \mathbf{b}^{n+1} &= \theta\mathbf{f}^{n+1} + (1-\theta)\mathbf{f}^n \\ &\quad - (1-\theta)\left[\rho^n(\mathbf{u}^n\cdot\nabla)\mathbf{u}^n - \nabla\cdot\left(\mu^n(\nabla\mathbf{u}^n + (\nabla\mathbf{u}^n)^T)\right)\right]. \end{aligned} \tag{3.2}$$

The parameter θ is chosen according to the time stepping method, $\theta = 1$ for the first order Backward Euler scheme and $\theta = 1/2$ for the second order Crank-Nicolson scheme. The Fractional-step-θ-scheme (first proposed by Glowinski et al. [10]) is a multi-step scheme involving the following time stepping parameters

$$\theta = 1 - \frac{\sqrt{2}}{2}, \qquad \theta' = 1 - 2\theta, \qquad \alpha = \frac{1-2\theta}{1-\theta}, \qquad \beta = 1 - \alpha$$

where the whole time interval Δt is split into the following three sub time steps

$$\begin{aligned} \rho\frac{\mathbf{u}^{n+\theta}-\mathbf{u}^n}{\Delta t/3} + \alpha\theta N(\mathbf{u}^{n+\theta})\mathbf{u}^{n+\theta} + \theta\nabla p^{n+\theta} &= \\ = \quad \alpha\theta\mathbf{f}^{n+\theta} + \beta\theta\mathbf{f}^n - \beta\theta N(\mathbf{u}^n)\mathbf{u}^n & \\ \nabla\cdot\mathbf{u}^{n+\theta} = 0, & \end{aligned} \tag{3.3}$$

$$\begin{aligned} \rho\frac{\mathbf{u}^{n+1-\theta}-\mathbf{u}^{n+\theta}}{\Delta t/3} + \beta\theta' N(\mathbf{u}^{n+1-\theta})\mathbf{u}^{n+1-\theta} + \theta'\nabla p^{n+1-\theta} &= \\ = \quad \beta\theta'\mathbf{f}^{n+1-\theta} + \alpha\theta'\mathbf{f}^{n+\theta} - \alpha\theta' N(\mathbf{u}^{n+\theta})\mathbf{u}^{n+\theta} & \\ \nabla\cdot\mathbf{u}^{n+1-\theta} = 0, & \end{aligned} \tag{3.4}$$

$$\begin{aligned} \rho\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n+1-\theta}}{\Delta t/3} + \alpha\theta N(\mathbf{u}^{n+1})\mathbf{u}^{n+1} + \theta\nabla p^{n+1} &= \\ = \quad \alpha\theta\mathbf{f}^{n+1} + \beta\theta\mathbf{f}^{n+1-\theta} - \beta\theta N(\mathbf{u}^{n+1-\theta})\mathbf{u}^{n+1-\theta} & \\ \nabla\cdot\mathbf{u}^{n+1} = 0. & \end{aligned} \tag{3.5}$$

Here $N(\mathbf{w})\mathbf{u} = \left[\rho(\mathbf{w}\cdot\nabla)\mathbf{u} - \nabla\cdot\left(\mu(\nabla\mathbf{u} + (\nabla\mathbf{u})^T)\right)\right]$ represents, in compact form, the nonlinear convective and diffusive terms.

3.2 Spatial discretization

The next step is to apply a spatial discretization to the unknown variables $\mathbf{u}$ and p. Commonly employed discretization schemes are finite volume (FVM), finite difference (FDM), and finite element methods (FEM). A suitable choice is not always obvious but in [102] Turek showed how to devise a very robust and computationally efficient nonconforming finite element scheme to solve the Navier-Stokes equations on arbitrary domains. In the following, this methodology will be extended to be able to efficiently treat two-phase flows with immiscible fluids.

The starting point of constructing a finite element discretization is to first derive the weak or variational form of the equations. The weak formulation is obtained by multiplying the equations with arbitrary functions $\mathbf{v}$, the so called test function space, after which integration is performed over the whole domain Ω. Partial integration (by using using the Gaussian theorem) is usually also applied to terms involving 2nd and higher order derivatives, which reduces smoothness requirements of the involved variables. The corresponding weak formulation of the Navier-Stokes equations which will be used in the following was derived in Chapter 2 (Section 2.7).

The discretization step consists of subdividing or triangulating the domain Ω into smaller cells (in this case quadrilaterals in two dimensions and hexahedrals in three dimensions). The triangulation is denoted by $\mathcal{T}_h$ where $h(K)$ is the diameter or width of cell K.

The approximation of the velocity on each cell utilizes rotated multilinear (bilinear in 2D and trilinear in 3D) polynomial shape functions $\tilde{Q}_1$ defined by

$$\tilde{Q}_1(K) := \left\{ q \circ \psi_K^{-1} : \quad q \in span\left\langle 1, x_i, x_i^2 - x_{i+1}^2, i = 1, \ldots, d \right\rangle \right\}$$

where $\psi_K : \hat{R} \to K$ is the multilinear transformation from the reference element $\hat{R} = [-1, 1]^d$ to the cell K. The corresponding degrees of freedom are determined by either of the functionals

$$F_E^{(a)} := |E|^{-1} \oint_E v \; ds \qquad \text{or} \qquad F_E^{(b)} := v(m_E)$$

where $E \subset \partial\mathcal{T}_h$ denotes the cell edges/faces. Using the functional $F_E^{(a)}$ will result in degrees of freedom corresponding to the mean values over the edges, and using $F_E^{(b)}$ will similarly correspond to pointwise values on the edge/face midpoints m_E. Alternatively one may employ the non-parametric counterparts which, although computationally expensive, are better suited for grids with high anisotropies.

The pressure field is approximated by piecewise constant values on each cell, the so called Q_0 element, and sought in the space

$$L_h = \{ q_h \in L_0^2 : q_{h|K} \equiv \text{const}, \quad \forall \; K \in \mathcal{T}_h \}.$$

The Stokes element pair $\tilde{Q}_1 Q_0$, also called Rannacher-Turek elements [106], is very efficient from a computational viewpoint, allowing for fast geometric multigrid solvers to be constructed. It is also very robust with respect to grid anisotropies and fulfills all stability requirements of the LBB Babuska-Brezzi condition.

The test function space $\mathbf{v}$ is usually taken as the same function space which approximates the dependent variables (Galerkin formulation), but can also be modified to include certain properties, for example streamline diffusion or SUPG stabilization (Petrov-Galerkin formulation).

Discretization in both time and space now yields the following saddle point system which must be solved to advance the solution:

Given $\mathbf{u}^n$ and time step $\Delta t = t^{n+1} - t^n$, solve for $\mathbf{u} = \mathbf{u}^{n+1}$ and $p = p^{n+1}$

$$\begin{cases} S\mathbf{u} + \Delta t B p = \mathbf{b} \\ B^T \mathbf{u} = 0 \end{cases} \tag{3.6}$$

where the system or iteration matrix S is defined as

$$S = \left[M_\rho + \theta \Delta t N(\mathbf{u}) \right].$$

Here M_ρ denotes the density weighted mass matrix

$$M_\rho = \int_\Omega \rho(\mathbf{x}) \mathbf{u} \cdot \mathbf{v} \, d\Omega$$

arising from the discretization of the time derivative. The transport operator $N(\cdot)$ is given by

$$N(\mathbf{w}) = \int_\Omega \rho(\mathbf{x}) \left((\mathbf{w} \cdot \nabla)\mathbf{u} \right) \cdot \mathbf{v} + \mu(\mathbf{x}) \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right) \cdot \nabla \mathbf{v} \, d\Omega$$

and contains the nonlinear convective and diffusive contributions. B and B^T are simply discrete analogs of the gradient and divergence operators, and are used to include the pressure gradient and to enforce the incompressibility constraint. The discrete right hand side is finally given by

$$\mathbf{b} = \int_\Omega \mathbf{b}_h^{n+1} \cdot \mathbf{v} \, d\Omega,$$

where $\mathbf{b}_h^{n+1}$ is the discrete analog to equation (3.2). A suitable and efficient method to solve this system, with proper treatment of the pressure and continuity equation, will be discussed in the following section.

3.3 Discrete projection method

Due to the incompressibility constraint special care has to be taken to be able to solve the saddle point system (3.6). Standard direct solvers and iterative solvers, not sensitive to zeros on the diagonal, may be applied to invert the coupled system directly. However, direct solvers are generally slow and require large amounts of memory. Iterative solvers on the other hand require more and more iterations to converge as the systems grow in size.

Two-phase flow applications are most often time dependent and involve phenomena that occur on very small time scales (for example break up and coalescence). This will pose natural restrictions on the maximum allowable size of the time steps. With this in mind it is preferable to employ the discrete pressure Schur complement approach, a type of projection method, presented in [103].

The essence of this methodology is to split the momentum equations and the continuity equation from each other, leading to smaller Poisson type problems which separately can be solved very efficiently.

The discrete projection method applied to (3.6) involves the following steps to obtain the new velocity field $\mathbf{u}^{n+1}$ and pressure field p^{n+1}:

1. Solve the momentum equations for the approximate velocity field $\tilde{\mathbf{u}}$ (while ignoring the incompressibility constraint):

$$S\tilde{\mathbf{u}} \;=\; \mathbf{b}^{n+1} - \Delta t B p^n$$

2. Construct the pressure right hand side:

$$f_p \;=\; \frac{1}{\Delta t} B^T \tilde{\mathbf{u}}$$

3. Approximate exact pressure matrix $P^* = B^T S^{-1} B$ with $P = \; B^T M_{\rho,l}^{-1} B$ and solve the pressure Poisson equation:

$$Pq \;=\; f_p$$

4. Update the pressure field:

$$p^{n+1} \;=\; p^n + \alpha_R q + \alpha_D M_{p,l}^{-1} f_p$$

   where α_R and α_D are appropriately chosen damping parameters, and $M_{p,l}^{-1}$ is the discrete lumped pressure mass matrix.

5. Project the approximate velocity field to a divergence free space:

$$\mathbf{u}^{n+1} \;=\; \tilde{\mathbf{u}} - \Delta t M_{\rho,l}^{-1} B q$$

The key to efficiency lies in optimizing the two main computationally intensive parts, the matrix inversions in steps 1 and 3, as well as the finite element matrix assembly.

3.3.1 Momentum equations

An efficient method to solve the discretized momentum equations in the first step of the discrete projection method is to use a geometric multigrid approach. The idea behind multigrid is to assemble and iteratively solve the linear systems on a sequence of grids. This allows for the slowly converging low frequency errors on the finest grid to quickly be filtered out on the coarser grids (the low frequency error is seen as having a higher frequency on the coarser grids). A near linear efficiency can in this way be achieved in the optimal case (with linear meaning that the cost of solving the systems increase linearly with the number of degrees of freedom) [42]. This is in contrast to standard iterative solvers which require an increasing number of iterations to converge as the computational grids are refined.

A prototypical multigrid sweep to solve the system $A_k u_k = f_k$ involves the following steps:

1. Given an initial guess on grid level k, u_k^0, perform $j = 0, \ldots, m-1$ presmoothing steps to get a more accurate iterate
$$u_k^{j+1} = \mathcal{S}_k(u_k^j).$$
The smoothing operator $\mathcal{S}_k$ essentially computes a first approximation to $A_k u_k = f_k$.

2. The presmoothing steps should have "smoothed" the residual sufficiently so that the remaining error will be seen as having a high frequency on a coarser grid. Calculate the residual and restrict it to a coarser grid
$$r_{k-1} = \mathcal{I}_k^{k-1}(f_k - A_k u_k^m)$$
where $\mathcal{I}_k^{k-1}$ is the restriction operator from the fine grid k to the coarser grid $k-1$.

3. Solve the coarse grid system
$$A_{k-1} u_{k-1}^* = r_{k-1}$$
to obtain the correction u_{k-1}^*.

4. Prolongate the calculated correction to the fine grid and apply
$$u_k^{m+1} = u_k^m + \alpha_k \mathcal{I}_{k-1}^k u_{k-1}^*,$$
where α is a suitably chosen damping parameter and $\mathcal{I}_{k-1}^k$ is the prolongation or interpolation operator from grid $k-1$ to k.

5. Perform $l = 0, \ldots, n-1$ number of postsmoothing steps (as in step 1) to obtain the final solution u_k^{m+1+n}.

It is common to apply these steps recursively on a succession of grid levels to achieve a faster reduction of error. Appropriate algorithms now have to be chosen for the prolongation, restriction, smoother, and solver components to achieve full efficiency.

The most computationally expensive component is normally the smoother $\mathcal{S}$ which performs a number of typical iterative solver steps (for example Jacobi, SOR, or SSOR). Smoothers depending on incomplete LU factorizations (ILU-k) are generally excellent at smoothing, however, their cost can offset the benefits on regular grids.

The prolongation and restriction routines are normally constructed either as pure interpolation operators or alternatively as discrete L^2-projection operators. Another approach, which has been developed for highly anisotropic grids, is to embed appropriate weighting in the operators to properly account for the anisotropies [92]. This approach is potentially advantageous for two-phase flow simulations since the discontinuous density and viscosity fields can be interpreted as anisotropies.

The employed solvers are usually standard iterative solvers, alternatively direct solvers may be used on coarse grids if the number of degrees of freedom is sufficiently small. It is not necessary to decrease the residual error in the solving step by a large amount, usually a gain of one digit is sufficient, since the correction is applied iteratively.

3.3.2 Convective stabilization techniques

High Re-number flows, where the convective terms dominate over the diffusive, often require special numerical treatment. This is mainly due to the limited resolution (the number of cells) that can be afforded in the simulations. The unresolved subgrid effects will eventually cause oscillations and an unphysical solution behavior if not suppressed. A mechanism to handle this consists of locally adding small amounts of numerical diffusion to counterbalance the dominating convective terms.

A quite elegant approach to introduce artificial stabilization in the finite element context consists of modifying the test function space to $\mathbf{v}_{PG} := \mathbf{v}+\delta\nabla\mathbf{v}$, where $\delta = \delta(Re_h, h)$ is locally tuned for the correct amount of stabilization. This classical Petrov-Galerkin approach only introduces artificial diffusion in the flow or streamline directions, and is thus called streamline diffusion or streamline upwind Petrov-Galerkin (SUPG) [47]. The method is consistent (meaning that the stabilization disappears for the exact solution), linear with respect to the solution variables, and is usually fairly easy to implement.

One major drawback of the streamline diffusion approach is that it allows unphysical under- and over-shoots to appear (the method is not monotonicity preserving). Depending on the nature of the phenomena under study it can be very critical to limit or even completely suppress this behavior (for example when solving for turbulence variables or chemical reactions). An alternative is to use total variation diminishing (TVD) and flux-corrected transport (FCT) schemes. These schemes allow for very accurate solutions at the cost of introducing an additional nonlinearity. However, if treated properly, or if the problem is already nonlinear, the additional cost will be quite low. A very elegant and general approach to introduce TVD/FCT, in particular with respect to the FEM context, is to use the approaches developed by Kuzmin et al. [58, 59, 60].

Another recent stabilization approach is so called edge stabilization where contributions are added to each cell edge corresponding to the jump in either the dependent variable or its gradient [12]. The stabilizing terms can quite easily be modified to treat different effects, for example instabilities due to convection, incompressibility, and the Korn's inequality (which appears when using the Rannacher-Turek elements with the deformation stress tensor formulation). The edge stabilization method is essentially linear, like streamline diffusion, but has the advantages of generally being more accurate and involve a quite insensitive tuning parameter (it can sometimes even vary within magnitudes without destabilizing the solutions significantly [104]). The main disadvantages are that it is not monotonicity preserving, and that it requires increased computational effort due to the need for assembly over all cell edges. The edge based structure also leads to additional matrix storage requirements, and a corresponding increase in computational time required to invert the matrices.

3.3.3 Nonlinear iteration techniques

The convective term in the momentum equations unfortunately contains a nonlinearity which must be treated appropriately. An efficient way to solve the nonlinear model problem $A(u)u = f$ is to use the following iterative defect correction approach

$$u_j = u_{j-1} + \omega\ C^{-1} r_{j-1}, \qquad j = 0, 1, 2, \ldots$$

where $r_l = f - A^{-1}(u_l)u_l$ is the defect vector in iteration l, and ω is a damping parameter. The preconditioning matrix C should ideally be chosen so that it is cheap to construct and easy to invert while still allowing for fast convergence. Typically one will use $C = A$ for the standard fixed point approach while full Newton schemes require a more elaborate construction. The iterative procedure is stopped when a predefined convergence criterion has been reached, for example when the norm of the residual error $||r_j||$ has decreased sufficiently or the solution difference between iterates $||u_j - u_{j-1}||$ is small enough.

The iterative defect correction scheme applied to the Navier-Stokes equations leads to the following linearization of the convective term

$$(\mathbf{u}_j \cdot \nabla)\mathbf{u}_j \approx (\mathbf{u}_{j-1} \cdot \nabla)\mathbf{u}_j$$

where $\mathbf{u}_{j-1}$ is the solution from the previous step in the defect loop. An alternative for time dependent problems is to use extrapolation backwards in time by replacing

$$(\mathbf{u}^{n+1} \cdot \nabla)\mathbf{u}^{n+1} \quad \text{by either} \quad (\mathbf{u}^n \cdot \nabla)\mathbf{u}_j \quad \text{or} \quad ((2\mathbf{u}^n - \mathbf{u}^{n-1}) \cdot \nabla)\mathbf{u}_j.$$

It is even possible to treat the convective term completely explicit on the right hand side. Although these extrapolation techniques remove the nonlinearities, and thus are computationally favorable, they should be used with caution since the time step size may have to be reduced dramatically if the nonlinearity is strong.

3.3.4 Pressure Poisson equation

The solution of the pressure Poisson equation is the second major part of the discrete projection method. To handle this efficiently a geometric multigrid approach, similar to the one used for solving the momentum equations, can be applied. An important difference between single and multiphase flow is in the choice and construction of the lumped mass matrix, which is the main constituent of the approximate projection matrix $P = B^T M_{\rho,l}^{-1} B$. For multiphase flow applications the lumped mass matrix needs to be weighted with the discontinuous density field and there are number of ways of doing this leading to quite different results.

The two main approaches is either to first lump the standard mass matrix and then scale it with the density, or first assemble the full density weighted mass matrix exactly and lump it afterwards. The first approach is easy and cheap, the standard row-sum lumped mass matrix is computed and stored once, after which it is multiplied with a corresponding density field vector. The density field can either be evaluated pointwise in the nodes (or midpoints) alternatively averaging can be applied on the cells first, after which a mean density for each cell can be evaluated.

The second alternative is to construct the mass matrix exactly, resolving the density jumps on the elements accurately after which a lumping procedure can be applied. For a mass matrix constructed from the $\tilde{Q}_1$ basis functions one can in general not use the standard row-sum lumping procedure since negative diagonal entries may be created. This is due to the combination of the density jump and non-positiveness of parts of the basis functions which may create dominating off-diagonal entries. There are two other approaches available however, diagonal

Grid level	Pw.	Avg.+Pw.	Exact+HRZ
3	111	31	14
4	109	33	12
5	40	23	8
6	35	23	7
7	26	27	5

Table 3.1: Average number of multigrid iterations required to solve the pressure Poisson equation. (Pw.) denotes pointwise evaluation of the density, (Avg.+Pw.) density averaging before pointwise evaluation, and (Exact+HRZ) exact assembly with HRZ-lumping.

lumping where only the positive diagonal entries are kept, and HRZ-lumping (from Hinton, Rock, and Zienkiewicz [44]) where the diagonal entries are kept and the whole resulting local mass matrix is scaled so that the total mass of each element is preserved.

The different approaches were evaluated by simulating a standard rising bubble test problem (test case 2 described in Chapter 6) and recording the average number of multigrid iterations required to reduce the residual norm in the pressure Poisson equation to 10^{-10}. The results are shown in Table 3.1. It is clear that the exact assembly plus HRZ-mass lumping is by far the best choice, requiring on average around 10 iterations. For the variants where the standard lumped mass matrix is scaled by the density it is preferable to average the density over the cell. This approach did however take 2-3 times more iterations to converge compared to using the HRZ-lumped mass matrix. The most costly method is to just scale with the density evaluated directly in the nodes, requiring over 100 iterations to achieve convergence on the coarser grid levels.

3.3.5 Matrix assembly

Finite element methods invariably require expensive numerical integration to build the finite element matrices, this is the so called assembly step. For single phase flow it is possible to save most matrices in memory if the mesh is kept fixed throughout the simulation, then only the convective contributions need to be reassembled. However, for two-phase flow simulations the density and viscosity fields will change with time, thus all but the gradient matrices will need to be reassembled in each and every time step. For efficiency reasons it is clearly very important to fully optimize the assembly routines using good programming practice, such as precomputing as much as possible and lifting calculations from the inner element assembly loops.

Another particularity of two-phase flow simulations is that the assembly will include numerical integration with discontinuous coefficients and functions. Standard cubature rules developed for continuous functions (such as Gaussian quadrature) may in this case lead to large errors. Two approaches can be taken to remedy this, firstly the discontinuous functions can be regularized and artificially "smoothed", secondly one can split the region to be integrated into several patches, each patch corresponding to a continuous part of the function.

Regularized assembly. Applying regularization to the discontinuous coefficients is a very popular and common approach. It requires very little modification to an existing code and is thus easy to implement. Another advantage is that the assembly can be performed fully implicit with respect to any interfaces, which can potentially be a great advantage. Following the work of Tornberg [109] a regularized Heaviside or step function can be constructed as

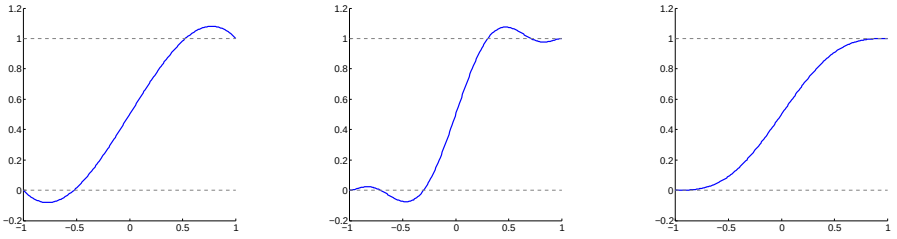
$$H_w(d) = \begin{cases} 1 & d > w, \\ \nu(d/w) & |d| \leq w, \\ 0 & d < w, \end{cases}$$

where the signed distance function $d = d(\Gamma, \mathbf{x})$ gives the minimum signed distance between the point $\mathbf{x}$ and the interface Γ. The regularization is fully determined by specifying the width of the regularized support region w and selecting the transition or smoothing function ν. The width should be chosen as a function of the mesh size $w = \mathcal{O}(h)$ so that the integrands converge in the limit $h \to 0$. A narrower transition region results in better accuracy but will require a higher quadrature rule to converge. Tornberg has shown that the use of transition functions with a high number of vanishing moments is advantageous since the analytical error of replacing the discontinuous Heaviside function with its regularized counterpart will accordingly scale with higher powers of w [109]. Two examples of suitable polynomial transition functions are

$$\begin{aligned} \nu^{2,0}(\xi) &= \frac{1}{2} + \frac{1}{8}(9\xi - 5\xi^3), \\ \nu^{4,1}(\xi) &= \frac{1}{2} + \frac{1}{256}(525\xi - 1225\xi^3 + 1323\xi^5 - 495\xi^7). \end{aligned}$$

The function $\nu^{2,0}$ has two vanishing moments but no continuous derivative, and $\nu^{4,1}$ has four vanishing moments and one continuous derivative. Special care must be taken when using these polynomials since they are not strictly positive (see Figures 3.1(a) and 3.1(b)), if used improperly they may locally generate unphysical coefficient values (for example negative density and viscosity values). Another common choice for the transition function, which is strictly positive but which error only scales as w^2, is the *sine* function (plotted in Figure 3.1(c))

$$\nu(\xi)^{sin} = \frac{1}{2}(1 + \xi + \frac{1}{\pi} sin(\pi\xi)).$$



(a) $\nu^{2,0}(\xi)$ (b) $\nu^{4,1}(\xi)$ (c) $\nu^{sin}(\xi)$

Figure 3.1: Transition functions approximating a Heaviside function.

Having chosen w, ν, and reconstructed H_w the regularized function f^{reg} is finally given as

$$f^{reg} = f_1 + (f_2 - f_1)H_w$$

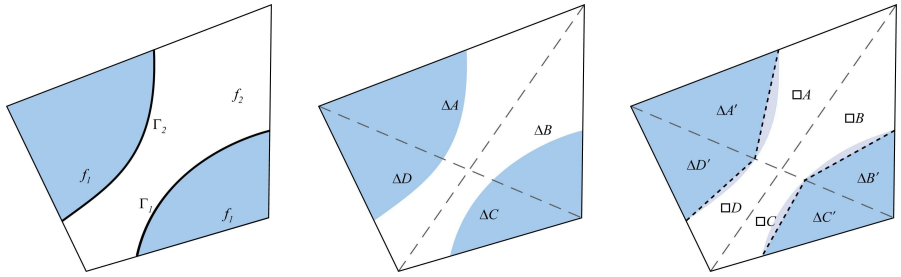
where f_i is the value of the discontinuous function in region or phase i. f^{reg} is clearly continuous, with smoothness properties according to the chosen transition function, and allows standard integration rules to be applied.

Patch assembly. The potentially more accurate way of integrating a discontinuous function is theoretically very simple but practically more complex to implement (especially in 3D). One must essentially identify and construct suitable subregions or patches, corresponding to the different function values, which then can be integrated with usual quadrature rules. The patch assembly procedure thus consists of applying the following steps to each cell containing an interface:

1. Reconstruct and identify all interfaces intersecting the cell.
2. Split the cell into smaller sub-regions by regular subdivision. (Optional)
3. Split the cell or sub-regions into polygonal patches over which the discontinuous function is constant or at least smooth.
4. Perform the integrations over all patches and sum up the results to form the full integral.

Since the interfaces normally are curved they do not allow for suitable patches to be formed and are thus approximated by lines in 2D and planes in 3D. This naturally results in a decrease in accuracy, hence the 2nd optional splitting step which can be performed recursively to attain arbitrary precision. The splitting is also helpful to eliminate ambiguities if several interface segments are present in the integration region.

The patch integration process applied to an arbitrary quadrilateral is illustrated in Figure 3.2. In step 1 (shown in Figure 3.2(a)) the interfaces are explicitly found and reconstructed. In this example two interfaces (Γ_1 and Γ_2) intersect the cell and separates the discontinuous function f into three regions with values f_1 or f_2. In the second step the quadrilateral is split into four triangles ($\Delta A - \Delta D$) via the diagonals (Figure 3.2(b)). This splitting is preferred to regular refinement into four quadrilaterals, since the lower right subcell then would contain two interface segments. The final patches over which integration is performed is depicted in Figure 3.2(c). Note that the interface curves have been approximated by straight line segments. In this case there are two ways of performing the integration, the first is to integrate normally over the patches, using f_2 for the quadrilateral patches $\square A - \square D$ and f_1 for the triangular ones $\Delta A' - \Delta D'$. Alternatively one can choose to integrate only over triangles, by first calculating the integral of f_1 over $\Delta A - \Delta D$ and then subtracting $f_1 - f_2$ integrated over the smaller patches $\Delta A' - \Delta D'$. Both methods will give the same result but the latter may be more convenient from an implementational viewpoint.



(a) Interface reconstruction (b) Cell splitting (c) Patch creation

Figure 3.2: Illustration of the patch assembly process.

A simple numerical test case was constructed in order to evaluate the different integration methods. The test considered the assembly of a mass matrix scaled with a discontinuous coefficient on a $[2 \times 2]$ square domain. The scaling coefficient f was given as

$$f(\mathbf{x}) = \begin{cases} 10 & |\mathbf{x}| < 0.6, \\ 1000 & |\mathbf{x}| \geq 0.6, \end{cases}$$

and was thus equal to 10 inside a circle with radius 0.6 and 1000 everywhere else. Assembly with regularization of the discontinuous function employed all three previously defined transition functions where the width spanned 1.5 cells ($w = 1.5h$). From Figure 3.3, showing the l_2 error for the assembled non-zero matrix entries, one can see that the *sine* ν^{sin} and low order polynomial $\nu^{2,0}$ transition functions were somewhat less accurate than the high order transition polynomial $\nu^{4,1}$, although all of them converged with more or less 2nd order. The simple 2x2 Gauss integration rule was also applied, where the discontinuous coefficient was evaluated directly in the cubature points, which resulted in errors of similar magnitude but with a slightly lower convergence order than for the regularized assembly. Lastly the patch integration process was tested and showed error levels a magnitude or so lower than the best of the other methods and also a better convergence behavior. Thus if high accuracy is needed then the patch assembly process is clearly the best choice.

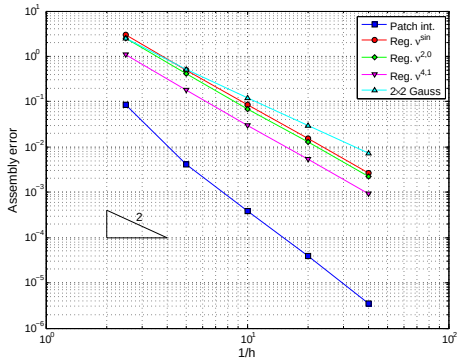


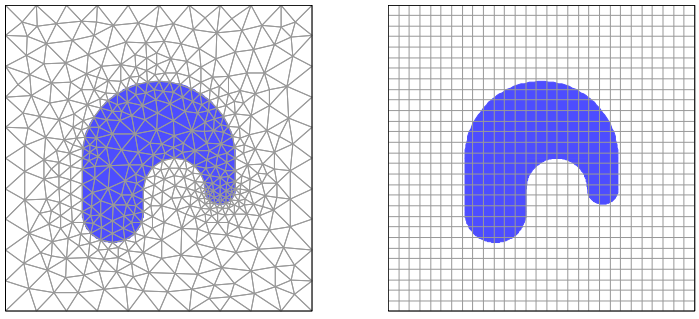
Figure 3.3: l_2 error in the assembly of a mass matrix scaled with a discontinuous coefficient.

Chapter 4

Interface tracking

Interface tracking methods can in general be divided into two main categories. The first category consists of Lagrangian methods where the interfaces are explicitly tracked, meaning that the cell edges always are aligned with the interfaces (Figure 4.1(a)). This approach preserves a sharp interface representation but does so at the cost of introducing additional algorithmic complexities to handle break up and merging of interface segments. The governing equations also have to be modified to account for the grid movement, and periodic remeshing may be necessary if the interfaces deform too much, which may introduce additional errors.

The second category consists of Eulerian interface tracking methods where the interfaces are allowed to intersect the cells arbitrarily (Figure 4.1(b)). This allows for fixed grids to be used which is more desirable from both computational and implementational points of view. In addition to this, coalescence and break up is, for most Eulerian methods, implicitly controlled by the grid resolution and is thus treated naturally. The drawback of using Eulerian methods is that the interfaces potentially are "smeared" across one or more cell widths requiring higher resolution to achieve the same accuracy as for the Lagrangian methods. In the following we will examine how the Eulerian methods can be applied to the posed two-phase flow problems.



(a) *Lagrangian* (b) *Eulerian*

Figure 4.1: Lagrangian and Eulerian interface descriptions.

4.1 Eulerian interface tracking methods

There are quite a number of Eulerian interface tracking methods and variants to choose from, each having its own distinct advantages and disadvantages. The most popular methods to date are the volume of fluid (VOF), front tracking, and level set methods.

The volume of fluid method originally introduced by Hirt and Nichols [69] implicitly tracks interfaces through the volume fraction function. Each cell is assigned a volume fraction between 0 and 1 to indicate the averaged relative amount of one of the fluids. A cell with a volume fraction of 0 or 1 will be completely filled with either of the fluids, while a cell with an intermediate volume fraction will have an interface segment intersecting it. The convection of the discontinuous volume fraction field usually requires special techniques but can be performed fully mass conserving, which is the main advantage of this approach. On the other hand, the use of an averaged volume fraction field does not give any information about the interfaces and their corresponding reconstructions are therefore not unique (one can for example use piecewise constant and linear (PLIC) techniques [110], or more advanced and expensive methods (such as the least squares ELVIRA and parabolic PROST methods [82, 84]) to reconstruct the interfaces). Both the reconstruction and convection steps makes the VOF method particularly suited for Cartesian tensor product grids, but the extension to unstructured grids, although possible, is quite rarely encountered.

The front tracking method essentially does the opposite of the VOF method. Here the front is tracked explicitly, usually with the help of massless marker particles indicating the position of the front. The convection of the particles is both easy and inexpensive. All that is required is an interpolation operator giving the velocity at the position of the particles. The opposite, to use information of the explicit front in the governing equations, requires more elaborate techniques. The common approach is to construct a smoothed delta function effectively smearing the front on the computational grid. Another drawback is that merging and break up of interfaces may cause difficulties, depending on the implementation approach. Additionally, marker particles must periodically be inserted or removed to maintain a nice representation of the front.

Many other methods exist, such as the phase field [29, 111], immersed interface [62], ghost fluid [28], and segment projection methods [109]. Some researchers have tried to combine two methods to get the best properties out of them. The CLSVOF method combines the level set and VOF methods to essentially create a mass conservative level set method [99]. The particle level set method can be seen as a diffuse front tracking approach used to correct the level set method [26]. In the following chapter the standard level set method is described in detail since it is easy to implement, the smooth level set field allows for accurate convection and discretization with high order finite elements, and distance function, normal, and curvature fields are easy to reconstruct globally.

4.2 Level set method

The level set method introduced by Osher and Sethian [72] has proved to be applicable in many diverse fields such as image processing, crystal growth, inverse problems, and of course multiphase flow. The main idea of the level set method is to embed the interface $\Gamma(t)$ (represented by a curve in 2D or surface in 3D) in a higher dimensional function ϕ, that is,

$$\Gamma(t) = \{\mathbf{x} \in \mathbb{R}^d \mid \phi(\mathbf{x}, t) = v_{ls}\},$$

where v_{ls} is the contour level or isosurface value implicitly representing the interface. The choice of v_{ls} is arbitrary but is usually taken as zero, since it then is possible to identify the different phases by just using the sign of ϕ. It is appropriate to prescribe (or at least initialize) ϕ as a signed distance function

$$\phi(\mathbf{x}, 0) = d(\Gamma, \mathbf{x}) = \begin{cases} dist(\Gamma, \mathbf{x}), & \mathbf{x} \in \Omega_1, \\ v_{ls}, & \mathbf{x} \in \Gamma, \\ -dist(\Gamma, \mathbf{x}), & \mathbf{x} \in \Omega_2, \end{cases}$$

where Ω_1 and Ω_2 denote the two regions that the fluids occupy. Two advantages of using a distance function is that it is smooth for the most part and simplifies construction of regularized Heaviside and delta functions. Methodologies using local grid adaptation and grid deformation can also make use of the distance function to identify where to refine the grid. Additionally normal and curvature fields are globally defined as

$$\hat{\mathbf{n}} = \frac{\nabla\phi}{|\nabla\phi|}, \qquad \kappa = -\nabla \cdot \hat{\mathbf{n}}. \tag{4.1}$$

An example of a level set function initialized as a signed distance function, where Γ is a circle, can be seen in Figure 4.2.

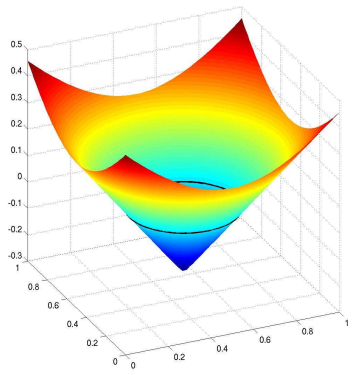


Figure 4.2: Level set description of a circle (represented by the solid black contour line).

The starting point for deriving an evolution equation for the level set field ϕ is to recognize that the following must hold for the moving interface (from here on taken as the zero level set)

$$\phi(\mathbf{x}(t), t) = 0.$$

Direct differentiation with the chain rule yields

$$\frac{\partial \phi}{\partial t} + \nabla \phi \cdot \frac{\partial \mathbf{x}(t)}{\partial t} = 0.$$

The speed with which the front propagates in the normal direction is given by $F = \hat{\mathbf{n}} \cdot \frac{\partial \mathbf{x}(t)}{\partial t}$. Using this and the definition of the normal vector in (4.1) results in the following evolution equation for ϕ

$$\frac{\partial \phi}{\partial t} + F|\nabla \phi| = 0$$

which must hold globally for all values of ϕ. The speed function F can depend on many variables such as mean curvature, external forces, but in our case will only depend on the fluid velocity, that is $F = \hat{\mathbf{n}} \cdot \mathbf{u}$, which gives

$$\frac{\partial \phi}{\partial t} + (\mathbf{u} \cdot \nabla)\phi = 0, \tag{4.2}$$

where the definition of the normal in (4.1) has been used once more.

What essentially has been achieved is a fully implicit formulation of the interface and its evolution at the cost of operating in a higher dimension. Additionally, geometrical properties such as interface normals and curvature are also implicitly defined and are possible to reconstruct globally. Note that in contrast to the VOF method, the interface is uniquely known and may be reconstructed if necessary.

4.3 Numerical treatment

The level set equation (4.2) is a pure hyperbolic transport problem and thus allows for application of standard solution tools, such as those used to treat the momentum equations in Chapter 3. Time discretization follows by applying the single step θ-scheme, which results in the problem formulation:

Given ϕ^n and the time step $\Delta t = t^{n+1} - t^n$, then solve for $\phi = \phi^{n+1}$

$$\frac{\phi - \phi^n}{\Delta t} + \theta(\mathbf{u} \cdot \nabla)\phi = b$$

with the right hand side

$$b = (\theta - 1)(\mathbf{u} \cdot \nabla)\phi^n.$$

Alternatively one may as before apply the Fractional-step-θ-scheme for additional stability which leads to a three step method analogous to equations (3.3)–(3.5).

It is only natural to choose the same discretization scheme in space as for the Navier-Stokes equations, the finite element method. Since the level set function is smooth for the most part a continuous representation is appropriate, differing from the non-conforming velocity ansatz functions. By using the same approximation order as for the velocities the choice falls upon continuous multilinear basis functions, the Q_1 space, which is described by

$$Q_1(K) = \left\{ q \circ F_K^{-1} : \quad q \in span\left\langle 1, x_i, x_i x_{\mathrm{mod}(i,d)+1}, i = 1, \ldots, d \right\rangle \right\}.$$

After discretization in space, the final form of the level set equation will read

$$\begin{aligned}[M_{(l)} + \Delta t\theta A]\phi^{n+1} &= b^n, \\ b^n &= [M_{(l)} - \Delta t(1-\theta)A]\phi^n,\end{aligned} \tag{4.3}$$

where $M_{(l)} = \int_\Omega v_1 v_2 \, d\Omega$ is the mass matrix which may be lumped if appropriate. A is the transport matrix, responsible for convecting the level set function and thus also implicitly the interface, and is given by

$$A = \int_\Omega (\mathbf{u} \cdot \nabla) v_1 v_2 \, d\Omega. \tag{4.4}$$

Here v_1 and v_2 denote the Q_1 basis functions and test functions, respectively. A is a trilinear form where the explicit velocity field $\mathbf{u}$ must be appropriately evaluated in the assembly step. Since $\mathbf{u}$ is changing A needs to be reassembled in each time step in contrast to the mass matrix which may be computed once and then reused. One can potentially save some effort by restricting the computations to a region around the interface, a so called narrow band approach, or alternatively if one sacrifices some accuracy the group finite element method, as suggested in [30, 60], is also a viable cost saving approach.

Since equation (4.2) is a hyperbolic transport equation without any diffusion, some form of artificial stabilization is needed. Edge stabilization initially seems to be an ideal choice, since it both allows for high accuracy and retains the linear character of the problem. The drawback is the increased matrix stencil adding to computational overhead. The stabilization options that are used in the following are the high resolution FEM-TVD schemes [58, 59, 60], although they render the problem nonlinear, the cost of solving the equations can for the most part be made almost negligible.

For the solution of equation (4.3) one may apply standard but efficient techniques, such as BiCGStab and multigrid schemes. Alternatively one can employ the iterative defect correction approach described earlier, which in fact also is necessary when using FEM-TVD. To take full advantage of all possibilities one should note that the preconditioning matrix C in the defect loop should closely resemble the iteration matrix

$$C = [M_{(l)} + \Delta t \, \theta A]$$

for fast convergence. It does not need to be exact, and can be approximated with just the mass matrix if the time steps are small enough. If one now also lumps the mass matrix the inversions are trivial and cost next to nothing. For larger time steps this approach usually works too but will require more iterations in the defect correction loop to converge.

4.4 Reinitialization

Even if the level set function is initialized as a distance function it will generally distort significantly with time, causing convergence difficulties and reducing accuracy for normal and curvature reconstructions. The velocity with which the level set field is transported will preserve the distance function property if

$$\nabla\phi \cdot \nabla F = 0 \tag{4.5}$$

is fulfilled, which means that the normal velocity is constant along the characteristics normal to the interface. This is unfortunately not the case for the velocity fields arising from fluid flow. To remedy this a so called extension velocity field may be constructed, which must both satisfy equation (4.5) and be identical to the original velocity field at the interface [1].

A more common approach is to periodically apply what is known as redistancing or reinitialization to the distorted level set field, and thus recover the distance function property. This is equivalent to solving the Eikonal equation

$$|\nabla\phi(\mathbf{x})| = F(\mathbf{x}) \tag{4.6}$$

with boundary condition $\phi(\mathbf{x}) = 0,\ \mathbf{x} \in \Gamma$. The speed function should be taken as unity, $F(\mathbf{x}) = 1$, in order to recover the pure distance function.

To be of practical interest the reinitialization methods must fulfill a number of requirements. Firstly, the chosen method should ideally not move the interface or zero level set, which would cause unphysical mass loss. Secondly, it should be as accurate as possible, since an accurate level set field will result in better normal and curvature fields. Thirdly, the computational overhead may not be significant, the computations should not be dominated by the redistancing step. It is worth to examine the various reinitialization algorithms in detail to identify suitable methods fulfilling these requirements [49].

4.4.1 Reinitialization methods

The fast sweeping method

The fast sweeping method described in the papers by Tsai et al. [100, 101] essentially consists of applying upwind type difference formulas, see subsection 4.4.2 later, while using Gauss-Seidel type of iterations, to update the distance function field. The key to fast sweeping is to update the points in a certain order that tries to follow the characteristics of the solution, that is the sweeping direction should ideally correspond to the real propagation of information. The basic method is described as follows

- Select sweeping direction and calculate the corresponding grid point order.
- Loop over all points in accordance with the above sequence and update the distance value for each point if the newly calculated distance is smaller than the previous one.
- Repeat this procedure for all sweeping directions.
- Repeat until convergence for all grid points has been achieved.

It is easy to see that the complexity of this algorithm is of order $\mathcal{O}(N)$, and has the potential to be very fast if the number of sweeps can be kept to a minimum. The sweeping orders may be chosen as follows for a two dimensional tensor product mesh

$$\begin{array}{ll} 1.\ i = 1:I, j = 1:J, & 2.\ i = I:1, j = 1:J \\ 3.\ i = I:1, j = J:1, & 4.\ i = 1:I, j = J:1 \end{array}$$

where i is the node position in the x-direction and j is the corresponding position in the y-direction. I and J are the maximum number of nodes in the x- and y-directions respectively.

Algebraic Newton method

This method utilizes an existing approximate distance field ψ by solving, for each grid point $\mathbf{x}_0$, the following equations

$$L(\mathbf{x}) = \begin{bmatrix} \psi(\mathbf{x}) \\ \nabla\psi(\mathbf{x}) \times (\mathbf{x} - \mathbf{x}_0) \end{bmatrix} = 0 \tag{4.7}$$

to locate the unknown point $\mathbf{x}$. The operator $L(\mathbf{x})$ specifies that $\mathbf{x}$ should lie on the zero distance curve and, additionally, that the vector pointing from the original point $\mathbf{x}_0$ to $\mathbf{x}$ should be parallel to the normal of the zero distance curve. Since each grid point $\mathbf{x}_0$ is only visited once the algorithmic complexity is of order $\mathcal{O}(N)$.

The system (4.7) can either be solved with a full Newton scheme, as described in Persson and Strang [80], or with the two step Newton scheme presented by Chopp [15] where the second order derivatives of the Jacobian have been omitted. The latter approach was tried but eventually dropped due to slow convergence.

For our purposes the two dimensional version of the operator and Jacobian will look like:

$$L(\mathbf{x}) = \begin{bmatrix} \psi(x, y) \\ (x - x_0)\psi_y - (y - y_0)\psi_x \end{bmatrix},$$

$$J(\mathbf{x}) = \frac{\partial L}{\partial \mathbf{x}} = \begin{bmatrix} \psi_x & \psi_y + (x - x_0)\psi_{xy} - (y - y_0)\psi_{xx} \\ \psi_y & -\psi_x - (y - y_0)\psi_{xy} + (x - x_0)\psi_{yy} \end{bmatrix}^T.$$

The typical iteration employed is $\mathbf{x}^{k+1} = \mathbf{x}^k - \delta J^{-1}(\mathbf{x}^k)L(\mathbf{x}^k)$, where δ is a relaxation parameter which can be adaptively adjusted to reduce the step size and to keep the updates from diverging. After each iteration convergence is checked by taking the residual norm of the operator $L(\mathbf{x})$. If convergence has been achieved the new distance is given by $\phi(\mathbf{x}_0) = |\mathbf{x} - \mathbf{x}_0|$.

The convergence properties of the algebraic Newton method depends on the smoothness of the given approximate distance field ψ. Should this field be non-smooth then the method will fail to find the exact distance in regions where the gradient is undefined (such as the corners of a square). Thus the algebraic Newton method can not be seen as a truly general method to compute distance functions for arbitrary distance fields, but the distance fields must belong to class C^2, that is have continuous second derivatives.

The fast marching method

The fast marching method originally devised by Sethian [88, 89] takes into account the characteristics of the solution, knowing that information will only propagate outward from the zero distance curve. Starting from there, each grid point is updated in order of increasing distance in an upwind fashion. Initialization of the fast marching algorithm is done with the following steps

- Tag all points on the cells intersecting the zero distance curve as *Accepted*, and calculate exact distance values to these points.
- Tag all grid points that lie in the neighboring cells to the boundary points as $Trial$, and compute initial approximate distance values to these.

- All other points lie in the *Unknown* set and should be given distance values that are bigger than the largest possible distance value.

After these initial steps the algorithm proceeds as follows

1. Find the point with the smallest distance value in the set of $Trial$ points.
2. Remove this point from the $Trial$ set and add it to the *Accepted* set.
3. Add all points of neighboring cells to the newly accepted point, that do not belong to the *Accepted* set to the $Trial$ set. Compute new distance values of all $Trial$ points that are neighbors to the newly accepted point.
4. Repeat the procedure until the $Trial$ set is empty.

The key to efficiency of fast marching is the realization of a heap structure for the $Trial$ set. This enables the sorting and finding of the minimum distance within the $Trial$ set (step 1) to be executed on average in $\mathcal{O}(logN)$ operations [87]. Thus the algorithmic complexity of the fast marching method is of order $\mathcal{O}(NlogN)$ [88]. Comparing the fast marching and fast sweeping methods one sees that fast marching costs $\mathcal{O}(logN)$ operations more to sort out the real propagation order, while fast sweeping assumes this is known in advance thus escaping this additional cost.

Brute force redistancing

The simple brute force method consists of subdividing or approximating the zero distance curve with linear line segments or just sampling points for an even easier version. Then the distance for each grid point to all approximated zero distance segments is computed, from which the minimum is taken as the new distance. This algorithm is obviously of order $\mathcal{O}(NM)$ with N being the number of grid points and M the number of zero distance segments. Thus if there are many interface segments the algorithm could approach quadratic costs while on the other hand for very few interface segments the algorithm is close to linear.

Hyperbolic PDE approach

The commonly used PDE based redistancing scheme uses the following time dependent hyperbolic equation to redistance a given approximate distance function ϕ_0 [98]

$$\frac{\partial \phi}{\partial t} = S(\phi_0)(1 - |\nabla \phi|). \tag{4.8}$$

This equation should be solved to the stationary limit together with homogeneous Neumann boundary conditions and the initial condition $\phi(\mathbf{x}, 0) = \phi_0(\mathbf{x})$.

$S(d)$ is a sign function introduced to propagate the information out from the zero distance curve. One does not commonly apply the exact sign function but a smoothed version, such as $S(d) = d^2/\sqrt{d^2 + \epsilon^2}$ where ϵ is proportional to the grid size or smoothing distance. This leads to a problem that the zero distance curve can wander off its initial position, resulting in loss of mass. Additional constraints can be introduced to minimize this effect such as done in a paper by Sussman and Fatemi [96].

Equation (4.8) can also be rewritten as

$$\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla \phi = S(\phi_0), \quad \mathbf{u} = S(\phi_0) \frac{\nabla \phi}{|\nabla \phi|}$$

taking on the guise of a normal, but nonlinear, convection and diffusion transport problem. Since there are a vast number of methods and discretization schemes to solve this problem (FDM, FEM, FVM, coupled with ENO methods, etc.), all resulting in different accuracies and efficiencies, this method will not be examined further.

4.4.2 Algorithmic components

Here the involved difference updates and the quadrilateral splitting used in the fast marching and fast sweeping methods are described in more detail.

Unstructured difference update

Both the fast marching method and the fast sweeping method depend on the ability to solve the Eikonal equation for a given grid point by using the distance values from the surrounding points. While this is rather straightforward given a strict Cartesian grid, it is not so easy if perturbed or unstructured grids are used. To manage this let's follow the general approach taken by Sethian and Vladimirsky [90] which allows for both first and second order updates. Another more geometrically oriented approach is taken by Kimmel et al. [56, 95], which however only allows for updates of first order accuracy.

To construct an approximation to the Eikonal equation for a grid point $\mathbf{x}$ using the surrounding points $\mathbf{x}_i, \cdots, \mathbf{x}_n$, first define the unit directional vector pointing from point $\mathbf{x}_i$ to point $\mathbf{x}$ as $\mathbf{P}_i = (\mathbf{x}-\mathbf{x}_i)/||\mathbf{x}-\mathbf{x}_i||$. Then let $v_i(\mathbf{x})$ be the value of the approximate directional derivative in direction $\mathbf{P}_i$ and point $\mathbf{x}$. Thus one can write that $v_i(\mathbf{x}) = \mathbf{P}_i \cdot \nabla \phi(\mathbf{x})$, and taking all directions i, substituting this into the Eikonal equation (4.6), and squaring gives the following

$$\mathbf{v}(\mathbf{x})^T (\mathbf{P}\mathbf{P}^T)^{-1} \mathbf{v}(\mathbf{x}) = F(\mathbf{x})^2 \tag{4.9}$$

where the matrix $\mathbf{P}$ is constructed by placing the directional vectors $\mathbf{P}_i$ as its rows and $\mathbf{v}(\mathbf{x})$ is simply a column vector with elements $v_i(\mathbf{x})$. Since $\mathbf{P}$ has to be nonsingular the updates are restricted to come from simplices, that is triangles in 2D and tetrahedra in 3D.

To solve equation (4.9) it is necessary to be able to write the directional derivatives $v_i(\mathbf{x})$ as functions of the unknown distance $\phi(\mathbf{x})$. First and second order difference approximations are given as follows:

$$v_i(\mathbf{x})^{\mathcal{O}(1)} = \frac{\phi(\mathbf{x})}{||\mathbf{x}-\mathbf{x}_i||} - \frac{\phi(\mathbf{x}_i)}{||\mathbf{x}-\mathbf{x}_i||},$$

$$v_i(\mathbf{x})^{\mathcal{O}(2)} = \frac{2\phi(\mathbf{x})}{||\mathbf{x}-\mathbf{x}_i||} - \frac{2\phi(\mathbf{x}_i)}{||\mathbf{x}-\mathbf{x}_i||} - \mathbf{P}_i \cdot \nabla \phi(\mathbf{x}_i).$$

Both of these formulas permit the rewriting of $v_i(\mathbf{x})$ into the form $v_i(\mathbf{x}) = a_i\phi(\mathbf{x}) + b_i$. Thus the Eikonal equation is finally reduced to a quadratic equation for the distance $\phi(\mathbf{x})$ in the form

$$(\mathbf{a}^T\mathbf{Q}\mathbf{a})\phi(\mathbf{x})^2 + (2\mathbf{a}^T\mathbf{Q}\mathbf{b})\phi(\mathbf{x}) + (\mathbf{b}^T\mathbf{Q}\mathbf{b}) = F(\mathbf{x})^2$$

where $\mathbf{Q}$ is defined as $\mathbf{Q} = (\mathbf{P}\mathbf{P}^T)^{-1}$ and the vectors $\mathbf{a}$ and $\mathbf{b}$ by their respective constituents a_i and b_i. This equation is then solved to find two roots of which the largest one is taken as a possible new distance for $\phi(\mathbf{x})$. This new distance can only be accepted if the update comes from within the polygon spanned by $\mathbf{x}, \mathbf{x}_i, \ldots, \mathbf{x}_n$ and if the calculated distance value is smaller than the old one. Thus the distance value is only updated if the vector components $\mathbf{Q}\mathbf{v}(\mathbf{x}) \approx \mathbf{Q}(\mathbf{a}\phi(\mathbf{x}) + \mathbf{b})$ are all positive and if $\phi(\mathbf{x}) < \phi^{old}(\mathbf{x})$.

Nodal point update and simplex construction

Given a point, A, to update with the unstructured update in a quadrilateral grid, the following procedure is applied:

- Loop over all quadrilaterals including grid point A as one of the defining vertices.
- Construct virtual triangles from grid point A and the other grid points of each quadrilateral. For an arbitrary quadrilateral with node numbering A to D the virtual triangles will have node combinations $A - B - C$, $A - C - D$, and $A - B - D$.
- Apply the unstructured difference update to grid point A for each virtual triangle in turn.

This splitting of the quadrilateral is quite effective to handle the imposed stability constraint of the unstructured update, that is to only update from acute triangles [56]. It is easy to see that the possibility to update point A from an acute angle is always present. This eliminates or at least reduces the need for splitting and unfolding of obtuse triangles described in references [56, 90]. The quadrilateral splitting procedure is illustrated in Figure 4.3.

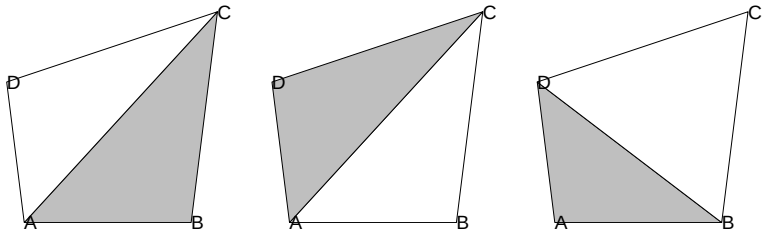


Figure 4.3: Simplex construction from a quadrilateral, the updating of point A comes from the grey triangle.

4.4.3 Numerical experiments

The described redistancing methods were applied to a practical benchmark test case in order to highlight their strengths and weaknesses. The test code was compiled with the Intel Fortran compiler and run on a 2.0 GHz Intel Core2Duo processor. In the following tables the column CPU denotes the actual required number of CPU seconds for a given method to converge for all nodes.

Test case

For the computational experiments a non trivial test case was chosen that both reflects typical features of an engineering computation, and also allows for an analytical solution. The test case is given by the following exact distance function field

$$\phi(\mathbf{x}) = \min\left(2.25 - y, \sqrt{x^2 + (y-1)^2} - 0.4\right).$$

The computational domain is a rectangle spanned by $x = [-1, 1],\ y = [0, 3]$. Both the zero distance curve and distance function field can be seen in Figure 4.4. This distance field contains both singularities, where the gradient is essentially undefined, and parts where the solution is smooth, giving a non-trivial test case.

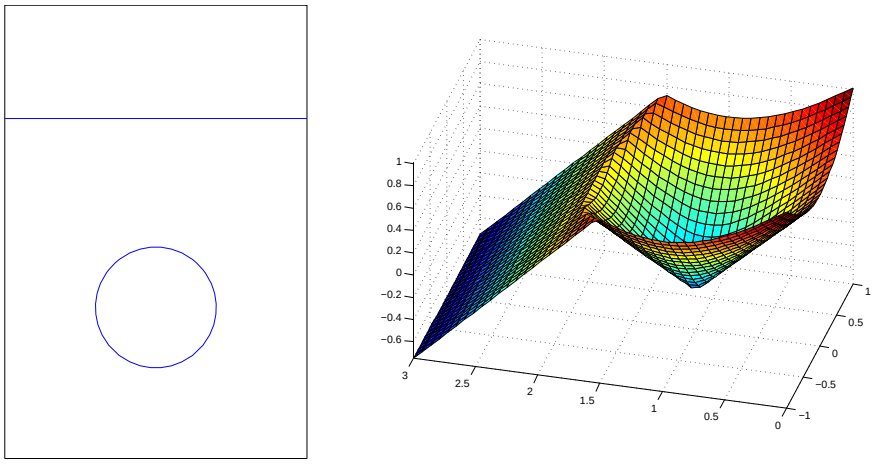


Figure 4.4: Interface curve and distance function field for the test case.

The basic coarse Cartesian grid consists of 24 conforming quadrilaterals with cell edge length $h = 0.5$, which correspond to grid level 1. Successive mesh levels are created by uniform subdivision, that is simply by splitting each cell into four new ones via its centroid coordinate. To test the unstructured capabilities of the solution algorithms, stochastic perturbations were introduced corresponding to 10% and 25% of the approximate mean cell edge length $h = 1/(\sqrt{N} - 1)$, where N is the number of nodes or vertices.

Error estimation

The computed solution vectors for all grid points were measured against the exact solutions in the following relative error norms

$$
\begin{aligned}
l_1 \textit{ error}: \qquad & ||e||_1 = \frac{\sum_{i=1}^{N} |\phi_{i,exact} - \phi_{i,computed}|}{\sum_{i=1}^{N} |\phi_{i,exact}|}, \\
l_2 \textit{ error}: \qquad & ||e||_2 = \left(\frac{\sum_{i=1}^{N} |\phi_{i,exact} - \phi_{i,computed}|^2}{\sum_{i=1}^{N} |\phi_{i,exact}|^2} \right)^{1/2}, \\
l_\infty \textit{ error}: \qquad & ||e||_\infty = \frac{\max_i |\phi_{i,exact} - \phi_{i,computed}|}{\max_i |\phi_{i,exact}|}.
\end{aligned}
$$

The error norms were only computed for points that did not belong to the cells intersecting the zero distance curve, or to points with a distance value greater than a prescribed maximum (taken as 0.35 in these tests), thus precluding the influence of singularities. In addition, convergence rates for the various methods in the different error norms were established as

$$\text{ROC} = log_{10}(\frac{||e^{l-1}||}{||e^l||})/log_{10}(\frac{h^{l-1}}{h^l})$$

where l is the level of refinement and h the mean cell diameter.

Results

The fast marching method. Tables 4.1 and 4.2 show the required CPU time, computed error norms and convergence orders for the fast marching method with the 1st and 2nd order difference updates, respectively. Only the most interesting results, from grid refinement levels 6 through 9 (corresponding to 24897 and 1575425 vertices), are included. It can be seen that the convergence rates very accurately approach the expected values with successive grid refinements. Also note that the grid distortion only causes a small reduction in accuracy, thus not really affecting the convergence rates at all. Comparing the results for the two difference update formulas one can see that the 2nd order approach only consumed a fraction more CPU time while at the same time giving much better results. The only reason not to use the 2nd order method is then if the gradients of the distance function at the interface nodes are unavailable, since they are required as additional initial values.

The fast sweeping method. Fast sweeping should ideally produce identical results to the marching method since it uses the same difference update formulas. This is also what calculations on non perturbed grids show. In the tests the same update order was used for distorted grids as for the purely Cartesian one. Unfortunately as the grids become more and more distorted the update procedure often fails, since one cannot be sure that the sweeping order is correct for an unstructured grid. The fast sweeping approach like standard iterative methods, requires an increasing number of sweeps to converge as the grid size decreases, which turned out to be particularly true for highly perturbed grids. The marching method was in fact faster, even on fully regular and unperturbed grids, and is therefore to be preferred over the sweeping method.

Level	CPU	$\|\|e\|\|_1$	ROC_1	$\|\|e\|\|_2$	ROC_2	$\|\|e\|\|_\infty$	ROC_∞
No mesh perturbation							
6	0.04	$0.212{\cdot}10^{-2}$	0.92	$0.455{\cdot}10^{-2}$	0.92	$0.237{\cdot}10^{-1}$	0.64
7	0.17	$0.109{\cdot}10^{-2}$	0.96	$0.235{\cdot}10^{-2}$	0.95	$0.132{\cdot}10^{-1}$	0.84
8	0.74	$0.553{\cdot}10^{-3}$	0.98	$0.119{\cdot}10^{-2}$	0.98	$0.710{\cdot}10^{-2}$	0.90
9	3.38	$0.280{\cdot}10^{-3}$	0.98	$0.604{\cdot}10^{-3}$	0.98	$0.368{\cdot}10^{-2}$	0.95
10% mesh perturbation							
6	0.05	$0.214{\cdot}10^{-2}$	0.94	$0.455{\cdot}10^{-2}$	0.93	$0.237{\cdot}10^{-1}$	0.66
7	0.20	$0.110{\cdot}10^{-2}$	0.96	$0.235{\cdot}10^{-2}$	0.95	$0.133{\cdot}10^{-1}$	0.83
8	0.88	$0.559{\cdot}10^{-3}$	0.98	$0.119{\cdot}10^{-2}$	0.98	$0.710{\cdot}10^{-2}$	0.90
9	4.00	$0.281{\cdot}10^{-3}$	0.99	$0.602{\cdot}10^{-3}$	0.99	$0.367{\cdot}10^{-2}$	0.95
25% mesh perturbation							
6	0.05	$0.219{\cdot}10^{-2}$	0.92	$0.451{\cdot}10^{-2}$	0.92	$0.235{\cdot}10^{-1}$	0.73
7	0.21	$0.113{\cdot}10^{-2}$	0.95	$0.233{\cdot}10^{-2}$	0.95	$0.132{\cdot}10^{-1}$	0.83
8	0.89	$0.573{\cdot}10^{-3}$	0.98	$0.118{\cdot}10^{-2}$	0.98	$0.702{\cdot}10^{-2}$	0.91
9	4.07	$0.289{\cdot}10^{-3}$	0.99	$0.597{\cdot}10^{-3}$	0.99	$0.362{\cdot}10^{-2}$	0.96

Table 4.1: Results for redistancing by the fast marching method with the 1st order difference update.

Level	CPU	$\|\|e\|\|_1$	ROC_1	$\|\|e\|\|_2$	ROC_2	$\|\|e\|\|_\infty$	ROC_∞
No mesh perturbation							
6	0.05	$0.569{\cdot}10^{-4}$	1.95	$0.122{\cdot}10^{-3}$	1.96	$0.106{\cdot}10^{-2}$	1.39
7	0.21	$0.146{\cdot}10^{-4}$	1.96	$0.314{\cdot}10^{-4}$	1.96	$0.296{\cdot}10^{-3}$	1.85
8	0.92	$0.367{\cdot}10^{-5}$	1.99	$0.786{\cdot}10^{-5}$	2.00	$0.778{\cdot}10^{-4}$	1.93
9	4.30	$0.925{\cdot}10^{-6}$	1.99	$0.198{\cdot}10^{-5}$	1.99	$0.195{\cdot}10^{-4}$	1.99
10% mesh perturbation							
6	0.06	$0.580{\cdot}10^{-4}$	1.96	$0.123{\cdot}10^{-3}$	1.97	$0.108{\cdot}10^{-2}$	1.50
7	0.23	$0.150{\cdot}10^{-4}$	1.95	$0.319{\cdot}10^{-4}$	1.95	$0.301{\cdot}10^{-3}$	1.84
8	1.00	$0.382{\cdot}10^{-5}$	1.97	$0.803{\cdot}10^{-5}$	1.99	$0.783{\cdot}10^{-4}$	1.94
9	4.79	$0.967{\cdot}10^{-6}$	1.98	$0.202{\cdot}10^{-5}$	1.99	$0.196{\cdot}10^{-4}$	2.00
25% mesh perturbation							
6	0.06	$0.636{\cdot}10^{-4}$	1.92	$0.130{\cdot}10^{-3}$	1.94	$0.110{\cdot}10^{-2}$	1.68
7	0.25	$0.168{\cdot}10^{-4}$	1.92	$0.342{\cdot}10^{-4}$	1.93	$0.311{\cdot}10^{-3}$	1.83
8	1.07	$0.434{\cdot}10^{-5}$	1.95	$0.878{\cdot}10^{-5}$	1.96	$0.795{\cdot}10^{-4}$	1.97
9	4.97	$0.111{\cdot}10^{-5}$	1.97	$0.223{\cdot}10^{-5}$	1.98	$0.203{\cdot}10^{-4}$	1.97

Table 4.2: Results for redistancing by the fast marching method with the 2nd order difference update.

Level	CPU	$\|\|e\|\|_1$	ROC_1	$\|\|e\|\|_2$	ROC_2	$\|\|e\|\|_\infty$	ROC_∞
No mesh perturbation							
6	0.35	$0.156{\cdot}10^{-3}$	1.92	$0.202{\cdot}10^{-3}$	1.96	$0.225{\cdot}10^{-3}$	2.01
7	1.72	$0.401{\cdot}10^{-4}$	1.96	$0.509{\cdot}10^{-4}$	1.99	$0.561{\cdot}10^{-4}$	2.00
8	8.95	$0.102{\cdot}10^{-4}$	1.98	$0.128{\cdot}10^{-4}$	1.99	$0.140{\cdot}10^{-4}$	2.00
9	51.98	$0.255{\cdot}10^{-5}$	2.00	$0.320{\cdot}10^{-5}$	2.00	$0.352{\cdot}10^{-5}$	2.00
10% mesh perturbation							
6	0.43	$0.152{\cdot}10^{-3}$	1.88	$0.197{\cdot}10^{-3}$	1.93	$0.263{\cdot}10^{-3}$	1.97
7	2.03	$0.398{\cdot}10^{-4}$	1.94	$0.504{\cdot}10^{-4}$	1.97	$0.651{\cdot}10^{-4}$	2.01
8	10.51	$0.101{\cdot}10^{-4}$	1.98	$0.127{\cdot}10^{-4}$	1.99	$0.162{\cdot}10^{-4}$	2.00
9	58.97	$0.251{\cdot}10^{-5}$	2.00	$0.316{\cdot}10^{-5}$	2.00	$0.405{\cdot}10^{-5}$	2.00
25% mesh perturbation							
6	0.45	$0.161{\cdot}10^{-3}$	1.88	$0.212{\cdot}10^{-3}$	1.94	$0.338{\cdot}10^{-3}$	1.97
7	2.05	$0.419{\cdot}10^{-4}$	1.94	$0.547{\cdot}10^{-4}$	1.95	$0.104{\cdot}10^{-2}$	–
8	10.55	$0.106{\cdot}10^{-4}$	1.98	$0.163{\cdot}10^{-4}$	1.74	$0.196{\cdot}10^{-2}$	–
9	59.82	$0.343{\cdot}10^{-5}$	1.63	$0.618{\cdot}10^{-3}$	–	$0.325{\cdot}10^{-0}$	–

Table 4.3: Results for the algebraic Newton method.

Algebraic Newton method. Table 4.3 presents the results for the algebraic Newton redistancing scheme with an approximate distance function given by a bilinear polynomial on each quadrilateral cell (the standard Q^1 finite element basis function). This distance field approximation, although a representative choice since one rarely can rely on having a nice analytical approximate distance function in practice, was quite challenging to use since it only is piecewise differentiable.

A convergence criteria of 10^{-9} and a maximum number of nonlinear iterations equal to 200 were used with the Newton method. The step size was also restricted not to exceed a distance equal to the mean cell diameter on the coarsest grid. The results show that the Newton method did not have any serious problems for grids with no or small perturbations, and converged with an order of 2. The approach did not however converge on the finer grids with 25% mesh perturbation. Regarding the CPU time, the Newton method was more than 10 times more demanding than the fast marching method.

Brute force method. Table 4.4 shows the results for brute force redistancing where the interface has been approximated by two straight line segments on each cell. Here, as expected, the accuracy is virtually independent of grid distortion, and the approach converges with an order of 2 in all norms. The required CPU times scales very unfavorably, rendering the method unpractical for larger calculations. The 10% increase in CPU time for the perturbed grids was due to the fact that the interface intersected more cells, resulting in more line segments than for the perfectly symmetric case.

Level	CPU	$\|\|e\|\|_1$	ROC_1	$\|\|e\|\|_2$	ROC_2	$\|\|e\|\|_\infty$	ROC_∞
No mesh perturbation							
6	0.57	$0.165 \cdot 10^{-3}$	1.98	$0.238 \cdot 10^{-3}$	2.01	$0.431 \cdot 10^{-3}$	2.01
7	4.61	$0.414 \cdot 10^{-4}$	1.99	$0.579 \cdot 10^{-4}$	2.04	$0.108 \cdot 10^{-3}$	2.00
8	36.58	$0.998 \cdot 10^{-5}$	2.05	$0.138 \cdot 10^{-4}$	2.06	$0.279 \cdot 10^{-4}$	1.95
9	294.71	$0.251 \cdot 10^{-5}$	1.99	$0.345 \cdot 10^{-5}$	2.00	$0.711 \cdot 10^{-5}$	1.97
10% mesh perturbation							
6	0.65	$0.169 \cdot 10^{-3}$	1.98	$0.245 \cdot 10^{-3}$	2.01	$0.505 \cdot 10^{-3}$	2.02
7	5.01	$0.422 \cdot 10^{-4}$	2.01	$0.591 \cdot 10^{-4}$	2.05	$0.121 \cdot 10^{-3}$	2.07
8	40.58	$0.101 \cdot 10^{-4}$	2.07	$0.140 \cdot 10^{-4}$	2.08	$0.326 \cdot 10^{-4}$	1.89
9	329.44	$0.253 \cdot 10^{-5}$	1.99	$0.350 \cdot 10^{-5}$	2.00	$0.770 \cdot 10^{-5}$	2.08
25% mesh perturbation							
6	0.65	$0.180 \cdot 10^{-3}$	1.96	$0.264 \cdot 10^{-3}$	1.99	$0.629 \cdot 10^{-3}$	2.04
7	5.06	$0.446 \cdot 10^{-4}$	2.01	$0.637 \cdot 10^{-4}$	2.05	$0.144 \cdot 10^{-3}$	2.13
8	40.70	$0.106 \cdot 10^{-4}$	2.08	$0.149 \cdot 10^{-4}$	2.10	$0.404 \cdot 10^{-4}$	1.83
9	328.60	$0.267 \cdot 10^{-5}$	1.99	$0.373 \cdot 10^{-5}$	2.00	$0.905 \cdot 10^{-5}$	2.16

Table 4.4: Results for the brute force method with the interface approximated by straight line segments.

4.4.4 Summary of reinitialization methods

From the tables it can clearly be seen that the fast marching method outperforms all other methods with respect to speed. Accuracy wise, marching with the 2nd order update is just as accurate as the other methods, and more notably is only marginally more expensive than with the first order update. The only drawback of the second order fast marching method is that one must compute and store arrays of the first derivatives causing some increase in memory consumption.

The fast sweeping method proved both slower than the marching method and was also quite unstable for highly perturbed grids. It is possible that these deficiencies could be overcome by more precise definitions of the sweeping order, but as it stands now this method is not competitive in a general context.

The algebraic Newton method is limited by the fact that it requires an approximate distance function for it to work. It is not possible to use the algorithm with a simple step function as initial distance field. The choice of an approximate distance function therefore influences the end results and necessitates careful tuning of the Newton scheme. For the considered benchmark, the timings were about a factor of ten slower than for the fast marching method, but this could vary somewhat depending on convergence criteria, the maximum number of allowed iterations, and the choice of relaxation parameter.

The last examined method, redistancing via brute force, does produce consistently good results for general grids but the required computational effort scales unfavorably for it to be generally interesting. The method could possibly prove useful to redistance a limited number of cells, such as the cells intersecting the initial zero distance curve. Brute force redistancing also shares a deficiency with the algebraic Newton method, that it can only produce the Euclidean distance, which will not be correct unless the domain is simply connected.

4.5 Normal and curvature field reconstruction

Geometric properties such as normals and curvature can easily be recovered globally from the level set function. Direct differentiation of the level set function ϕ gives

$$\hat{\mathbf{n}} = \frac{\nabla\phi}{|\nabla\phi|}, \quad \kappa = -\nabla \cdot \hat{\mathbf{n}}.$$

Although the finite element method supports this approach, it is not the best alternative since accuracy is lost when differentiating ϕ. The recovered normals will also be discontinuous at the cell edges if C^0-functions (such as Q_1 elements) are used, which causes ambiguous evaluations necessitating some form of averaging procedure. Furthermore, first order basis functions are not two times differentiable and thus some other approach must be found to reconstruct the curvature.

More accurate approaches include reconstruction via L^2-projection and so called patch recovery techniques. The L^2-projection approach minimizes the following variational forms

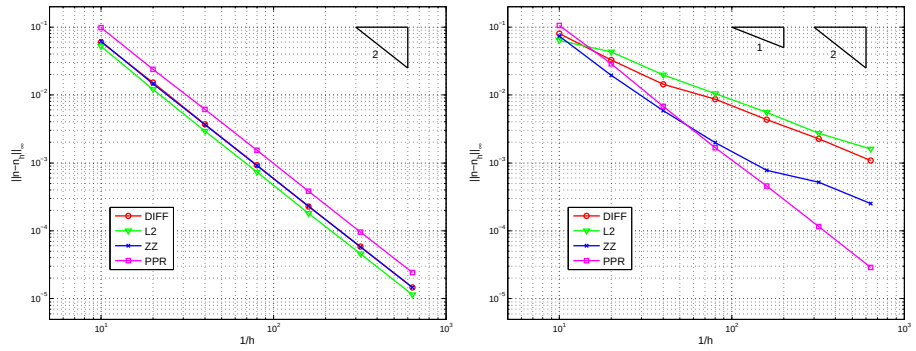
$$M_{(l)} n_{i,h} = B_i \phi_h, \quad M_{(l)} \kappa_h = -\sum_i (B_i n_{i,h}), \qquad i = 1, \ldots, n$$

where $M_{(l)}$ is the (possibly lumped) mass matrix, $n_{i,h}$ the recovered normal vector in direction i, and B_i the gradient matrix $B_i = \int \frac{\partial v_1}{\partial x_i} v_2 \, d\Omega$. An underscore h indicates the discrete analog of the continuous variable. Once recovered, the discrete normals and curvature now live in the corresponding finite element ansatz space (in our case Q_1 so they are continuous) and can be evaluated accordingly. If the mass matrix is lumped the inversion is trivial, otherwise appropriate solvers have to be applied.

The main idea behind patch based gradient recovery techniques is to use information available in the surrounding cells to construct a more accurate representation of the gradients. In the following the ZZ-patch recovery technique (from Zienkiewicz and Zhu [119, 120]) and the PPR-technique (short for polynomial preserving recovery [118]) will be explored. The idea of the ZZ-technique is to evaluate the gradients in superconvergent points of the cells neighboring the node of interest, after which a local least-squares fitting is applied. This construction has been shown to achieve better convergence on regular grids than otherwise possible [119]. The PPR-technique instead constructs a high order polynomial around the evaluation point from which the gradients can be recovered in a robust way.

The described gradient recovery techniques are applied to a test problem in order to assess the accuracy of the resulting normal and curvature fields. The test configuration considers a circle with radius 0.25 centered in a 2×2 domain. Errors can easily be calculated since the analytic normal and curvature fields around a circle are known. Figure 4.5(a) shows the maximum norm of the error of the normal reconstruction, which was evaluated pointwise along the circumference of the circle. The computations were performed for regular refinements of a perfect tensor product grid, and it is clear that all methods yield 2nd order convergence. However, when the grid nodes are stochastically perturbed by 20% one can see some clear differences (Figure 4.5(b)). The direct differentiation (DIFF) and L^2-projection (L2) methods now only converges with

1st order accuracy. The ZZ-technique converges with and order of 2 for the coarser grids but decreases to 1st order on as the grids are refined. The PPR approach is as robust as before achieving full 2nd order convergence.



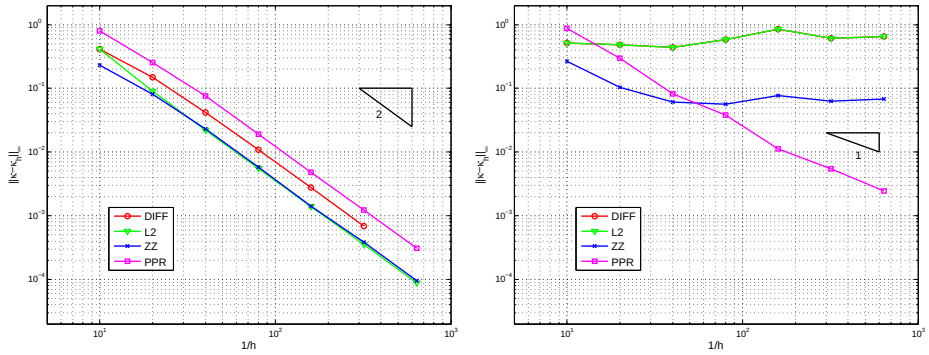
(a) No perturbation (b) 20% perturbation

Figure 4.5: Errors in normal reconstruction on a tensor product grid with and without 20% nodal perturbation.

The curvature is recovered in a 2nd step using the previously computed normals. In this way it is essential that the normals are recovered accurately in order to get a precise representation of the curvature. Figure 4.6(a) shows the results for the tensor product grids. Like for the normals, full 2nd order convergence is achieved. This is not so surprising since the recovery methods are the same, and the used grids are computationally favorable. The absolute values of the curvature errors are a magnitude higher than for the normals, but this is to be expected since some accuracy must be lost in the additional reconstruction step. When the grids are perturbed on the other hand the results are completely different (Figure 4.6(b)). Now the direct differentiation (DIFF) and L^2-projection (L2) methods completely stop converging (note that their respective curves overlap each other). The ZZ-patch recovery shows a quite low initial error but also stops converging as the grid is refined. The only method that converges is the PPR-technique although it shows the largest error on the coarsest grid. Clearly, if one works with anything but perfectly regular grids, the PPR-technique, although expensive, is the safest choice giving at least 1st order convergence in the curvature reconstruction.

4.6 Mass conservation

The last topic to examine within the field of interface tracking is mass conservation. The employed level set method is neither locally nor globally mass conserving. The amount of mass lost (or gained) during the corse of a simulation depends on many factors such as mesh density, time step, reinitialization method, and the scheme chosen to convect the level set field. Decreasing the mesh size and time steps sizes are two obvious ways to decrease mass loss. However, if this is not feasible (for example due to computational requirements) or if the problem in question calls for absolute mass conservation, an additional postprocessing step is needed.



(a) No perturbation (b) 20% perturbation

Figure 4.6: Errors in curvature reconstruction on a tensor product grid with and without 20% nodal perturbation.

In [94] Smolianski proposed a global mass correction step for the level set method which consists of lowering or raising the level set curve by addition of a suitable constant, that is

$$\phi^*(\mathbf{x}) = \phi(\mathbf{x}) + c_{mc}.$$

The mass correction constant defined as $c_{mc} = (A_{ref} - A(t))/l_\Gamma$, where A_{ref} is the reference mass/area, $A(t)$ the calculated mass/area, and l_Γ the length or circumference of the interface. This correction is to be applied after each time step and introduces to a very small change in the level set function which effectively eliminates accumulation of mass errors in time. An improvement of this approach was presented by Mut et al. [67] which applies this correction in a weighted sense so that nodes/cells with more or less mass are corrected appropriately. In this way the impact of the total mass correction is controlled better.

There are two other options to improve mass conservation worth mentioning. Firstly, one can, like for the phase field method introduce a Lagrange multiplier, which should enforce mass conservation in the equations themselves [29]. This is quite attractive from a mathematical point of view since the correction isn't as artificially imposed as for the method described above. Alternatively, one can employ the conservative level set method, by Olsson and Kreiss [73, 74], in which the distance function is substituted with a smoothed Heaviside like function. Additional source terms are also introduced to preserve the width of the smoothing region, and thus implicitly preserving the mass.

Chapter 5

Surface tension effects

As discussed earlier, surface tension effects arise due to the internal attraction of the sparsely distributed fluid molecules near an interface [114]. In contrast to the main bulk of the fluid, where the attractive forces are counterbalanced by other surrounding molecules, this results in net forces which work to minimize the corresponding areas or volumes of the involved interfaces. The magnitude of this effect will thus depend on the molecular composition of the involved fluids and also the shape of the interfaces.

In multiphase flow applications these surface tension or capillary effects can usually either be completely ignored, for example in large scale wave phenomena, or they have to be resolved very accurately, for example in small scale experiments involving bubbles and drops. In the former case there is not much to discuss, but in the latter some time and effort must be devoted to accurately and efficiently include these effects into the discrete systems.

5.1 Surface tension model

The surface tension forces can be mathematically modeled as interfacial boundary conditions. Assuming that no mass transfer between the fluid phases occur the following conditions apply at the $d-1$ dimensional interface $\Gamma \subset \Omega$

$$[\mathbf{u}]|_\Gamma = 0, \quad -\left[-p\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right]\big|_\Gamma \cdot \hat{\mathbf{n}} = \sigma\kappa\hat{\mathbf{n}}.$$

Here again $\hat{\mathbf{n}}$ denotes the interface normal and $[\mathsf{A}]|_\Gamma$ is the jump of property A across the interface. These conditions imply continuity of the velocity across the interface and also a jump in the normal stress proportional to the coefficient of surface tension σ and the curvature of the interface κ.

To apply these boundary conditions directly to the interfaces is often impractical. Fortunately, they can also be rewritten as source terms

$$\mathbf{f}_{st} = \sigma\kappa\hat{\mathbf{n}}\delta(\Gamma, \mathbf{x}) \tag{5.1}$$

where $\delta(\Gamma, \mathbf{x})$ is a Dirac delta function localizing the surface tension force to the interface between the different fluids. These additional source terms arise naturally when using a finite element discretization but have to be implemented

approximately when using finite differences or volumes. The finite difference counterpart is often called the immersed boundary method and has its roots in the early work on blood flow by Peskin [81]. This was later extended to finite volume/volume of fluid (VOF) calculations by Brackbill, Kothe, and Zemach [9] who dubbed it the continuum surface force method (CSF). The two latter approaches employ a regularization of the interface surface integrals to transform them to volume integrals. A significant advantage of this approach is that the evaluation of the surface tension forces may be done fully implicit in space by using a suitable regularization method, meaning that the interfaces never have to be reconstructed explicitly.

5.2 Explicit time integration

The most common way of discretizing the surface tension forces (5.1) in time is to use a fully explicit approach. The involved quantities are thus evaluated for the interface at the previous time step Γ^n and added to the right hand side as a source term. In the finite element context this is equivalent to

$$\mathbf{f}_{st} = \int_{\Gamma^n} \sigma\kappa^n\hat{\mathbf{n}}^n \cdot \mathbf{v}\, d\Gamma \tag{5.2}$$

$$= \int_{\Omega} \sigma\kappa^n\hat{\mathbf{n}}^n\delta(\Gamma^n, \mathbf{x}) \cdot \mathbf{v}\, d\Omega \tag{5.3}$$

where a superscript n denotes the old time level and $\mathbf{v}$ are the test functions.

A proper choice of time step size is not always clear. For efficiency reasons it is desirable to use a large time step, but if the phenomena under study occur very rapidly or involve high frequency fluctuations then the time step size must be correspondingly reduced. With a little reasoning it is possible to derive some upper bounds on the size of the time steps which should not be exceeded.

First consider the involved physical effects. It is desirable to achieve a balance between them so that neither will dominate and impose higher requirements on the temporal resolution. In the case of two-phase flows the capillary forces should be balanced against the viscous ones which, with help of the non-dimensional capillary number, can be expressed as

$$Ca = \frac{\mu U^{cap}}{\sigma} = \frac{\mu L}{\Delta t^{cap}\sigma} \approx 1 \qquad \Rightarrow \qquad \Delta t^{cap} \approx \frac{\mu L}{\sigma}.$$

Here U^{cap} and L denote characteristic velocity and length scales, respectively, and Δt^{cap} is our resulting physical capillary time scale.

Using the explicit form of the surface tension forces, equation (5.2) or (5.3), will unfortunately lead to the following discrete time step restriction [9]

$$\frac{c^{cap}\Delta t_h^{cap}}{h} < 1 \tag{5.4}$$

where c^{cap} is the phase velocity of the capillary waves given by

$$c^{cap} = \sqrt{\frac{\sigma\kappa}{\rho}}. \tag{5.5}$$

The largest magnitude of the curvature κ that can be resolved is inversely proportional to the size h of the cells containing an interface segment. Using this and inserting (5.5) in (5.4) yields the discrete numerical time step criterion

$$\Delta t_h^{cap} < \sqrt{\frac{\rho h^3}{\sigma}} \tag{5.6}$$

This constraint is very restrictive for high mesh densities (scaling like $h^{3/2}$) and also contra-productive to the requirement to resolve the discontinuities very accurately (with for example some form of local grid adaption). A small mesh size in the interface regions will thus impose a more or less global time step constraint severely limiting computational efficiency (any efficiency gains from locally adapting the grid might possibly be completely nullified by this time step restriction). In the following, a method circumventing this limitation, while at the same time preserving the scheme fully implicit with respect to all interfaces, will be examined.

5.3 Semi-implicit time integration

When working with the finite element method it is possible to employ partial integration to transfer the explicit calculation of the curvature to the test function space. This methodology was introduced by Dziuk [24] and later applied to flow calculations in the Arbitrary Eulerian Lagrangian (ALE) framework by Bänsch and coworkers [3, 4, 5], Sashikumaar and Tobiska [37], and Matthies [64]. Some work has also been done in the context of Eulerian fixed grids; with level sets by Groß et al. [41] and with front tracking by Minev and coworkers [66].

In the following, the new variation of implementing surface tension forces by Hysing [50] is presented. The approach combines the ideas of regularizing the surface integrals and implementing them semi-implicitly in time. This allows the capillary time step restriction to be circumvented while never explicitly having to reconstruct the interfaces, which always has been done previously. The first step in deriving this method is to introduce some definitions from differential geometry.

Definition 1 *The tangential gradient of a function f, which is differentiable in an open neighborhood of Γ, is defined by*

$$\underline{\nabla} f(x) = \nabla f(x) - (\hat{\mathbf{n}}(x) \cdot \nabla f(x))\hat{\mathbf{n}}(x), \quad x \in \Gamma.$$

Here ∇ denotes the usual gradient in $\mathbb{R}^d$.

Definition 2 *If f is two times differentiable in a neighborhood of Γ, then the Laplace-Beltrami operator of f is defined as*

$$\underline{\Delta} f(x) = \underline{\nabla} \cdot (\underline{\nabla} f(x)), \quad x \in \Gamma. \tag{5.7}$$

Lemma 1 *A theorem of differential geometry now states that*

$$\underline{\Delta} \mathbf{x}|_\Gamma = \kappa \hat{\mathbf{n}} \tag{5.8}$$

where κ is the mean curvature and $\mathbf{x}|_\Gamma$ is the identity mapping on Γ. Proof of this is for example given in Gallot et al. [34].

To transform the surface tension term (5.1) to its variational equivalent, which is the basis of a finite element discretization, multiply it with a suitably chosen test function space $\mathbf{v}$ and integrate over Ω, which yields

$$\mathbf{f}_{st} = \int_{\Omega} \sigma\kappa\hat{\mathbf{n}} \cdot \mathbf{v}\ \delta(\Gamma, \mathbf{x})\, d\Omega = \int_{\Gamma} \sigma\kappa\hat{\mathbf{n}} \cdot \mathbf{v}\, d\Gamma. \tag{5.9}$$

Here the delta function has been embedded in the integral to simplify the expressions. Using relations (5.7) and (5.8) with equation (5.9) and partial integration gives the following

$$\begin{aligned} \mathbf{f}_{st} &= \int_{\Gamma} \sigma\kappa\hat{\mathbf{n}} \cdot \mathbf{v}\, d\Gamma = \int_{\Gamma} \sigma(\underline{\Delta}\mathbf{x}|_{\Gamma}) \cdot \mathbf{v}\, d\Gamma \\ &= -\int_{\Gamma} \sigma\underline{\nabla}\mathbf{x}|_{\Gamma} \cdot \underline{\nabla}\mathbf{v}\, d\Gamma + \int_{\gamma} \sigma\partial_{\gamma}\mathbf{x}|_{\Gamma} \cdot \mathbf{v}\, d\gamma \end{aligned} \tag{5.10}$$

where the boundary term $\partial_{\gamma}\mathbf{x}|_{\Gamma} = \hat{\mathbf{n}}_{\gamma}$ is acting on the tangent line γ, given by the intersection between the interface Γ and the boundary of Ω, in the direction tangential to the interface. This term will only appear if the integration path does not posses a closed shape and $\mathbf{v}$ is non-vanishing on γ.

Following the work of Bänsch [3, 4] one can reduce the awkward time step restriction (5.6) by using a semi-implicit time discretization instead. This is accomplished by writing the new interface position as a function of the old position

$$(\mathbf{x}|_{\Gamma})^{n+1} = (\mathbf{x}|_{\Gamma})^{n} + \Delta t\, \mathbf{u}^{n+1} \tag{5.11}$$

where $\Delta t = t^{n+1} - t^{n}$ is the time step and $\mathbf{u}^{n+1}$ is the velocity field at the new time level. This results in the following representation of the surface tension effects

$$\mathbf{f}_{st} = -\int_{\Gamma^n} \sigma\underline{\nabla}(\mathbf{x}|_{\Gamma})^{n} \cdot \underline{\nabla}\mathbf{v}\, d\Gamma - \Delta t \int_{\Gamma^n} \sigma\underline{\nabla}\mathbf{u}^{n+1} \cdot \underline{\nabla}\mathbf{v}\, d\Gamma. \tag{5.12}$$

The resulting new term is linear with respect to the velocity at time level $n+1$ and can thus simply be assembled as a positive definite contribution to the iteration matrix. This approach is simpler to implement than the earlier work on implicit surface tension by Williams and Hochstein [45], which essentially required that the normals and curvature were evaluated implicitly as

$$\hat{\mathbf{n}}^{n+1} \approx \hat{\mathbf{n}}^{n} + \Delta t \left(\frac{\partial\hat{\mathbf{n}}}{\partial t}\right)^{n}, \qquad \kappa^{n+1} \approx \kappa^{n} + \Delta t \left(\frac{\partial\kappa}{\partial t}\right) \approx \kappa^{n} - \Delta t\, \nabla \cdot \left(\frac{\partial\hat{\mathbf{n}}}{\partial t}\right)^{n}.$$

The clear advantage that the semi-implicit discretization (5.12) has over a purely explicit one is that the additional term represents a diffusion operator working in the tangential direction of Γ. This results in a more physical implementation of capillary effects since an increased coefficient of surface tension now generates more interface diffusion, that is a stiffer system, instead of a larger destabilizing source term.

5.4 Fully implicit surface tension force

Having arrived at the semi-implicit expression in time for the surface tension force it would now be desirable to also combine it with the CSF framework. This would enable a fully implicit representation of the interfaces in space while at the same time retaining the stabilizing effect. To achieve this one must first go back to equations (5.9) and (5.10) but now keep the delta function in the expressions, that is

$$\begin{aligned} \mathbf{f}_{st} &= \int_\Omega \sigma\kappa\hat{\mathbf{n}}\cdot\mathbf{v}\ \delta(\Gamma,\mathbf{x})\,d\Omega &&= \int_\Omega \sigma(\underline{\Delta}\mathbf{x}|_\Gamma)\cdot(\mathbf{v}\delta(\Gamma,\mathbf{x}))\,d\Omega \\ &= -\int_\Omega \sigma\underline{\nabla}\mathbf{x}|_\Gamma\cdot\underline{\nabla}(\mathbf{v}\delta(\Gamma,\mathbf{x}))\,d\Omega &&= -\int_\Omega \sigma\underline{\nabla}\mathbf{x}|_\Gamma\cdot\underline{\nabla}\mathbf{v}\ \delta(\Gamma,\mathbf{x})\,d\Omega, \end{aligned}$$

where it has been assumed that $\delta(\Gamma,\mathbf{x})$ is constant in the tangential direction. Applying the semi-implicit time integration (5.11) now gives

$$\mathbf{f}_{st} = -\int_\Omega \sigma\underline{\nabla}(\mathbf{x}|_\Gamma)^n\cdot\underline{\nabla}\mathbf{v}\ \delta(\Gamma^n,\mathbf{x})\,d\Omega - \Delta t\int_\Omega \sigma\underline{\nabla}\mathbf{u}^{n+1}\cdot\underline{\nabla}\mathbf{v}\ \delta(\Gamma^n,\mathbf{x})\,d\Omega.$$

The last step is to substitute the singular Dirac delta function δ with its regularized counterpart $\delta_{\boldsymbol{\epsilon}}$, after which one arrives at the final expression for the surface tension force

$$\begin{aligned} \mathbf{f}_{st} = &-\int_\Omega \sigma\ \delta_{\boldsymbol{\epsilon}}(\Gamma^n,\mathbf{x})\ \underline{\nabla}(\tilde{\mathbf{x}}|_\Gamma)^n\cdot\underline{\nabla}\mathbf{v}\,d\Omega \\ &-\Delta t\int_\Omega \sigma\ \delta_{\boldsymbol{\epsilon}}(\Gamma^n,\mathbf{x}))\ \underline{\nabla}\mathbf{u}\cdot\underline{\nabla}\mathbf{v}\,d\Omega \end{aligned} \tag{5.13}$$

where $\tilde{\mathbf{x}}|_\Gamma$ is the extension of the interface over the support of $\delta_{\boldsymbol{\epsilon}}$ with width 2ϵ.

Assuming that it is possible to find an implicit way to construct the regularized delta function in equation (5.13), then the surface tension implementation will be fully implicit with respect to any interfaces and should not be bounded by the capillary time step restriction (5.6).

5.5 Regularization

The construction of the regularized Dirac delta function follows the same approach as for the regularization of the Heaviside function in Chapter 3. Existence of a distance function significantly simplifies the regularization procedure, and with $d(\Gamma,\mathbf{x})$ giving the minimum signed distance from $\mathbf{x}$ to Γ, one can write the delta function as

$$\delta(\Gamma,\mathbf{x}) = \delta(d(\Gamma,\mathbf{x})).$$

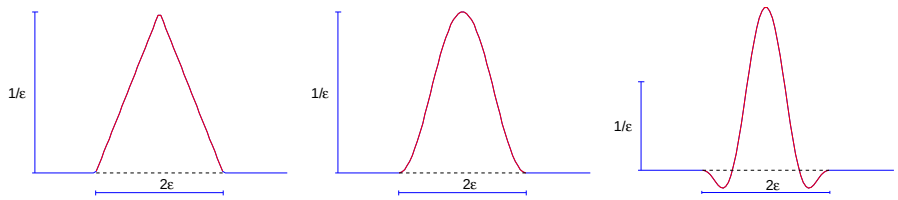
The regularized and continuous delta function δ_ϵ can now be defined as

$$\delta_\epsilon(x) = \begin{cases} \frac{1}{\epsilon}\varphi(x/\epsilon) & |x|\le\epsilon = mh, \\ 0 & |x|>\epsilon = mh, \end{cases}$$

where h is the mesh spacing, which together with the constant m defines the support ϵ of the regularized delta function, and φ is a characteristic function determining the kernel shape. There are several possible choices for the kernel function of which some common choices are:

- The linear hat function:
$$\varphi^1(\xi) = 1 - |\xi|$$
- The *cosine* approximation:
$$\varphi^2(\xi) = \frac{1}{2}(1 + cos(\frac{\pi\xi}{2}))$$
- Higher order polynomials, for example:
$$\varphi^3(\xi) = \frac{312}{512}(3 - 20\xi^2 + 42\xi^4 - 36\xi^6 + 11\xi^8)$$

Figure 5.1 shows a graphical representation of the kernel function for a fixed width 2ϵ. For a more in depth analysis of the ensuing errors when using this regularization approach the reader is referred the extensive work of Tornberg et al. [25, 109].



(a) $\delta_\epsilon(\varphi^1)$ (b) $\delta_\epsilon(\varphi^2)$ (c) $\delta_\epsilon(\varphi^3)$

Figure 5.1: Three examples of kernel functions for regularization of a Dirac delta function.

5.6 Numerical tests

In this section results from computational simulations are presented which examine the performance of the proposed surface tension implementation variant (5.13), which in the following is labeled CSF-LBI. The new approach is compared to the standard explicit method denoted by CSF (5.3).

5.6.1 Static bubble

This test case models a perfectly stationary circular bubble at equilibrium. According to the Laplace-Young law the pressure inside the bubble is equal to $p_{in} = p_{out} + \sigma/r$, where r is the radius of the bubble. Since everything is stationary the velocity should be zero everywhere, however, due to certain imbalances in the numerical method spurious velocity currents will be generated.

The test configuration consisted of a bubble with radius $r = 0.25$ positioned in the center of a unit square. The coefficient of surface tension and the viscosities were set to unity while the densities were given a magnitude of 10^4, which corresponds to a Laplace number of $La = (2r)\sigma\rho\mu^{-2} = 5 \cdot 10^3$. A fixed time step of $\Delta t = 0.01$ was used and the simulations were run until $t = 125$.

Tables 5.1 and 5.2 show the error of the dimensionless velocity for the different methods in the l^∞ and l^1 norms, defined as $max_i\ |\mathbf{u}_i\mu/\sigma|$ and $\frac{1}{N}\sum_{i=1}^{N}\ |\mathbf{u}_i\mu/\sigma|$, where N is the number of nodes. The new method proved somewhat more accurate than the standard CSF method which can be expected due to the additional diffusion in the interface region.

1/h	$\tilde{Q}_1 Q_0^{\mathrm{CSF}}$	$\tilde{Q}_1 Q_0^{\mathrm{CSF-LBI}}$
20	$7.4 \cdot 10^{-3}$	$6.9 \cdot 10^{-3}$
40	$3.9 \cdot 10^{-3}$	$3.7 \cdot 10^{-3}$
80	$2.0 \cdot 10^{-3}$	$1.8 \cdot 10^{-3}$
160	$9.8 \cdot 10^{-4}$	$8.1 \cdot 10^{-4}$
ROC $\approx$	1.0	1.0

Table 5.1: Errors and convergence rates (ROC) in the discrete l^∞ norm for the non-dimensional velocity $\mathbf{u}\mu/\sigma$.

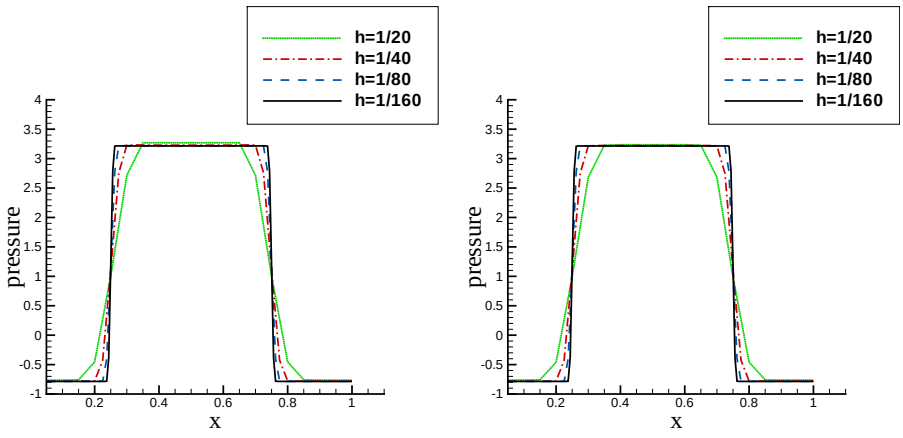
1/h	$\tilde{Q}_1 Q_0^{\mathrm{CSF}}$	$\tilde{Q}_1 Q_0^{\mathrm{CSF-LBI}}$
20	$6.7 \cdot 10^{-4}$	$5.8 \cdot 10^{-4}$
40	$1.9 \cdot 10^{-4}$	$1.6 \cdot 10^{-4}$
80	$5.2 \cdot 10^{-5}$	$4.1 \cdot 10^{-5}$
160	$1.4 \cdot 10^{-5}$	$1.0 \cdot 10^{-5}$
ROC $\approx$	1.9	2.0

Table 5.2: Errors and convergence rates (ROC) in the discrete l^1 norm for the non-dimensional velocity $\mathbf{u}\mu/\sigma$.

Table 5.3 shows how well the pressure field fulfilled the Laplace-Young law in both absolute and relative error norms. The results indicate that the $\tilde{Q}_1 Q_0$ approach with the standard CSF method does not perform equally well to the new CSF-LBI method. The error for the new approach was about a factor of 3 smaller than with the explicit implementation. Figure 5.2 shows pressure cut-lines at $y = 0.5$ for various levels of grid refinement and as can be seen the pressure approximation is quite sharp and non oscillating for both CSF methods even on fairly coarse grids.

1/h	$\tilde{Q}_1 Q_0^{\mathrm{CSF}}$	$\tilde{Q}_1 Q_0^{\mathrm{CSF-LBI}}$
	$\lvert p_{in} - p_{out} - \sigma/r \rvert$	
20	$4.8 \cdot 10^{-2}$	$6.1 \cdot 10^{-3}$
40	$1.2 \cdot 10^{-2}$	$3.5 \cdot 10^{-3}$
80	$3.1 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
160	$7.8 \cdot 10^{-4}$	$2.7 \cdot 10^{-4}$
	$\frac{\lvert p_{in} - p_{out} - \sigma/r \rvert}{(\sigma/r)} \cdot 100\%$	
20	1.2%	0.15%
40	0.31%	0.088%
80	0.078%	0.026%
160	0.020%	0.0066%

Table 5.3: Absolute (top) and relative (bottom) errors in the Laplace-Young law.



(a) CSF (b) CSF-LBI

Figure 5.2: Pressure cut-line ($y = 0.5$) for four different mesh sizes.

5.6.2 Oscillating bubble

In the second example the previous test configuration is made a little more difficult. The circle is in this case initially perturbed to an elliptical shape by scaling the semi-axes a factor 1.25 in the x-direction and 0.8 in the y-direction. The ellipse or bubble consists of the same fluid as in the surrounding unit square cavity. The fluid has a density of 10^4, viscosity 1, and a coefficient of surface tension equal to 0.1. Simulations for various levels of refinement of the initial 5×5 mesh were performed with a fixed time step $\Delta t = 5$ until $t = 1000$. At the final time the bubble is expected to have reached an equilibrium state, that is a stable circular shape. Theoretically the capillary time step restriction (5.6)

is already exceeded on the second mesh refinement (L3), as can be seen from Table 5.4, and instabilities can thus be expected to appear on this and finer meshes.

Mesh level	L1	L2	L3	L4	L5	L6	L7
Δt_h^{cap}	11.3	4.0	1.4	0.5	0.17	0.06	0.02

Table 5.4: Capillary time step restriction for the standard explicit CSF method applied to the oscillating bubble example.

Figure 5.3 shows the results for the standard CSF method at six representative times for levels 3-6. It is apparent that numerical oscillations start to appear and pollute the solution as the mesh is refined. The new CSF-LBI method on the other hand proved to be very stable, and as can be seen from Figure 5.4 only the very finest levels (at 80-250 times Δt_h^{cap}) show the onset of oscillations.

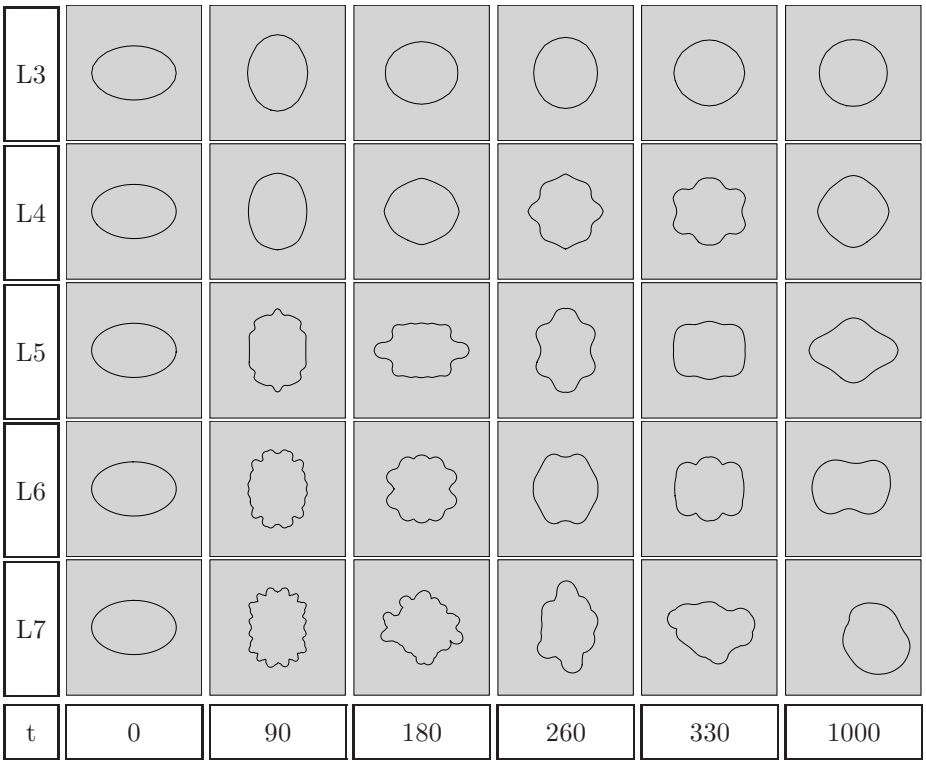


Figure 5.3: Evolution of an oscillating bubble with the standard explicit CSF method. Refinement levels 3-7.

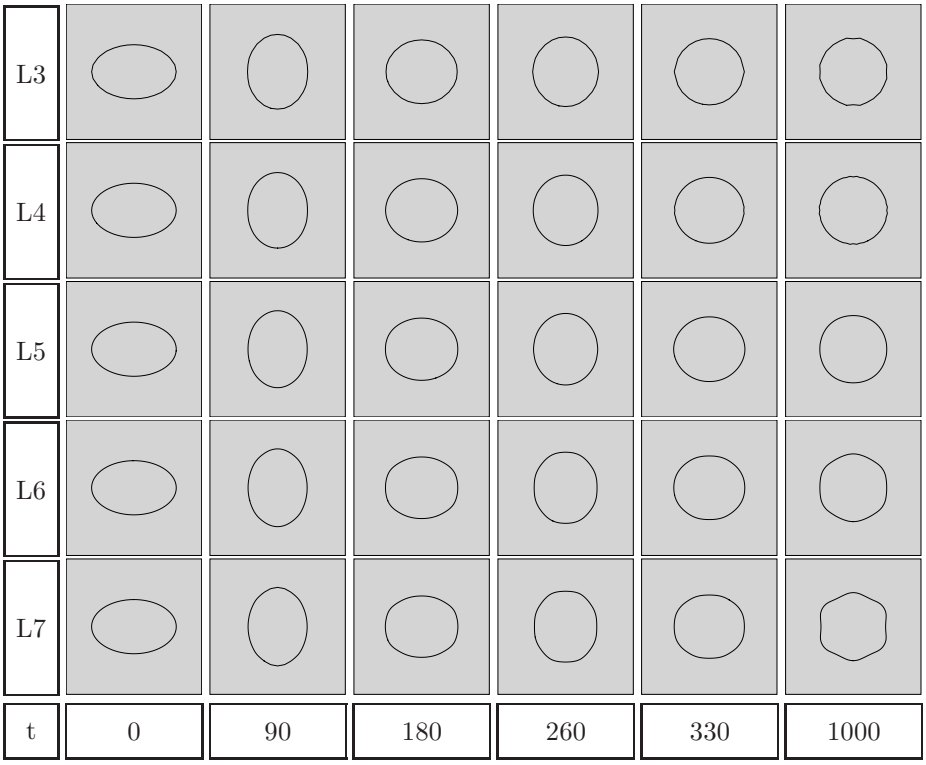


Figure 5.4: Evolution of an oscillating bubble with the new CSF-LBI method. Refinement levels 3-7.

5.6.3 Rising bubble

The final test example concerns a single bubble rising in a heavier fluid. The bubble is given an initial radius of $r_0 = 0.2$ and is placed at $[0.5; 0.5]$ in a rectangular domain with dimensions $[0; 1] \times [0; 2]$. No-slip conditions are applied on the horizontal walls and slip conditions on the vertical ones. Initially both fluids are at rest having a velocity of zero everywhere.

The physical parameters for the simulation are listed in Table 5.5 and correspond to a Reynolds number $Re = (2r_0)^{3/2} g_y^{1/2} \rho_1 \mu_1^{-1} = 71.6$ and Eötvös number $Eo = 4\rho_1 g_y r_0^2 \sigma^{-1} = 2.56$. According to Clift et al. [19] such a bubble is expected to assume an ellipsoidal shape. This is valid for fully three dimensional bubbles but not necessarily for the considered two dimensional one, it should however give an indication of the final shape that might be found. The computations were performed on a single 80×160 mesh where the time step Δt was varied between 0.0125 and 4 to test the stabilizing capabilities of the new surface tension implementation method. From (5.6) the estimate of the capillary time step restriction is $\Delta t_h^{cap} = 0.056$ indicating possible problems for the larger time step sizes.

ρ_1 (liquid)	10^4
ρ_2 (gas)	10^3
μ_1 (liquid)	1
μ_2 (gas)	1
g_y	$-8 \cdot 10^{-4}$
σ	0.5

Table 5.5: Physical parameters used in the rising bubble example.

Figure 5.5 shows the results for the standard CSF method at time steps $\Delta t = 0.5$ and $\Delta t = 1.0$, where it can be seen that severe oscillations pollute the solution with the larger time step. Further increases in time step size resulted in a complete breakdown of the solution process. It is interesting to note that it was still possible to perform the simulation although the capillary time step restriction was exceeded by almost a factor of ten. Figure 5.6 in contrast shows the results for the new CSF-LBI method where both time steps yielded the same qualitative solution. The method did in fact even work quite well up to $\Delta t = 2.0$, after which there also appeared too much distortion in the interface contour. Table 5.6 lists the averaged number of nonlinear iterations and linear multigrid steps for velocity and pressure with time step sizes $\Delta t = 0.5$ and $\Delta t = 0.25$. The new CSF-LBI method resulted in a slight decrease in the number of nonlinear iterations and multigrid steps while solving the momentum equations. The number of multigrid steps required to solve the pressure Poisson equation was however unaffected.

Δt	0.5		0.25	
	CSF	CSF-LBI	CSF	CSF-LBI
ANNL	4.1	3.6	3	2.6
AMGU	5.2	4.8	3	2.7
AMGP	5.3	5.2	4.6	4.6

Table 5.6: Average number of nonlinear iterations (ANNL), and linear multigrid steps for solving the velocity (AMGU) and the pressure (AMGP), for the rising bubble test case with time steps $\Delta t = 0.5$ and $\Delta t = 0.25$.

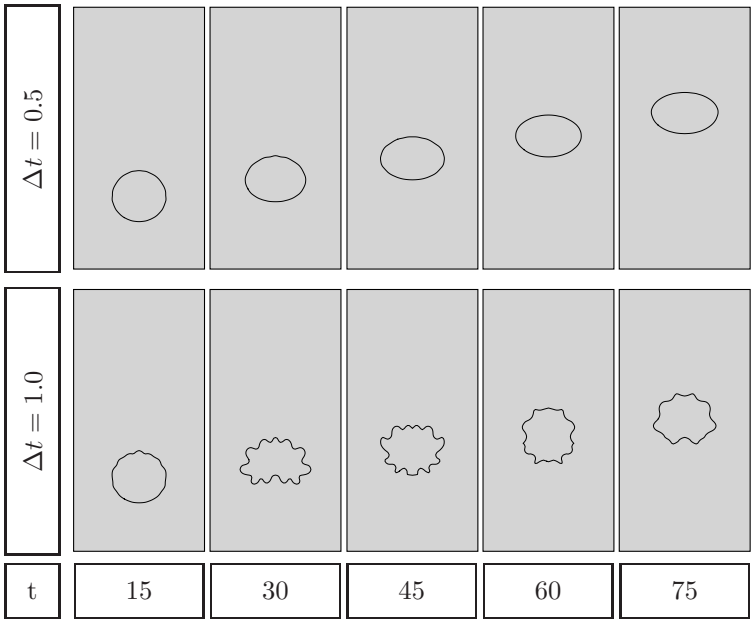


Figure 5.5: Evolution of a rising bubble with the standard CSF method with time steps $\Delta t = 0.5$ and $\Delta t = 1.0$.

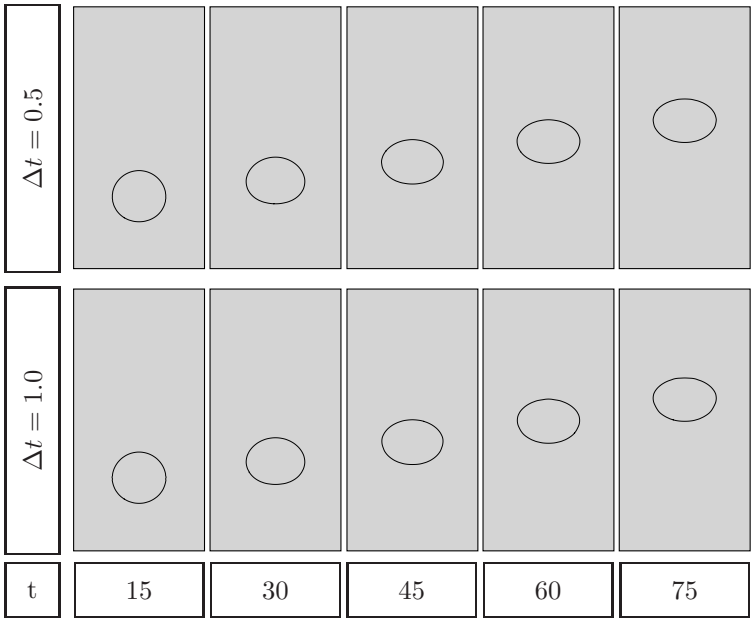


Figure 5.6: Evolution of a rising bubble with the new CSF-LBI method with time steps $\Delta t = 0.5$ and $\Delta t = 1.0$.

Chapter 6

Benchmarking

From Chapters 2–5 it is clear that numerical simulation of incompressible interfacial flows, such as two-phase flows with immiscible fluids, is a very complex matter involving many separate parts which must work together. Each year a significant amount of effort is invested by the research community in devising improved schemes, algorithms, and methods resulting in numerous publications on the topic. Considering this, it is perhaps somewhat surprising that no rigorous, that is *quantitative*, numerical benchmark configuration has been proposed for validation and comparison of interfacial flow codes to this date. This is in contrast to other fields of computational fluid dynamics for which dedicated benchmarks have been presented and accepted by the general CFD community [18, 46, 54, 68, 107, 108].

The most common approach to validate interfacial flow codes is to examine the "picture norm", that is qualitatively comparing the interface shape to either experiments or other numerical simulations. One frequently used test case for validation of two-phase flow codes is the classical dam break experiment by Martin and Moyce [65]. Although the dam break benchmark can be helpful for rough calibration in the early stages of code development, the somewhat imprecise experimental data and negligible influence of surface tension effects limits its subsequent use. When simulating bubbles and drops it is common to compare the computed bubble shapes to the experimentally established Clift, Grace, and Weber diagrams [19]. However, one should be careful not to only compare with experiments, since even if perfect agreement is reached one still cannot be certain that the solutions have satisfied the Navier-Stokes equations. Taking this into consideration it would be preferable to validate a code with well defined numerical benchmarks so that one can be sure that at least the posed mathematical problem is solved correctly.

Numerical benchmarks are on the other hand only helpful if they can be used for quantitative comparisons, mere visual inspection is rarely if ever enough to draw serious conclusions. To illustrate this, consider the bubble shapes shown in Figure 6.1. These shapes are calculated by six different codes with identical problem formulations. They should thus ideally give six identical solutions. Sadly this is clearly not the case. The shapes are quite similar but it is not possible to tell which solutions, if any, are really correct. In order to be able

to do this one must leave the "picture norm" behind and instead use some rigid metrics with which convergence directly can be measured. This shall be addressed in the following.

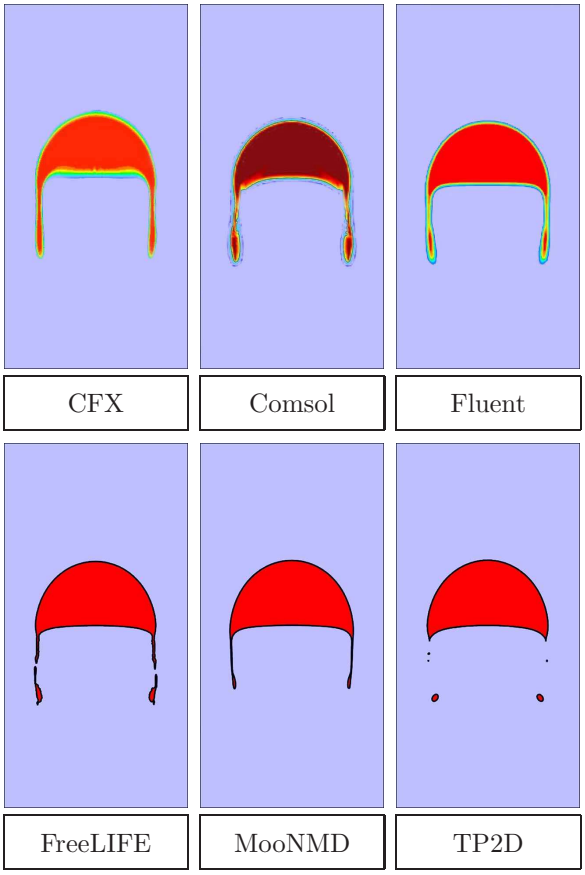


Figure 6.1: Simulation of a rising bubble with six different codes.

The rest of this chapter is devoted to the establishment of two new numerical benchmark configurations for quantitative comparison and validation of interfacial flow codes. Relevant benchmark quantities are defined to directly measure topological parameters, such as interface deformation, and also indirect ones, such as velocity measures. The task of the benchmarks is to track the evolution of an initially circular two-dimensional bubble rising in a liquid column. This configuration is simple enough to compute accurately yet also allows for very complex topology change, giving the interface tracking techniques of today an adequate challenge.

6.1 Definition of test cases

This section describes the governing equations, and defines the test cases and benchmark quantities to be used for validation of interfacial flow codes.

6.1.1 Configuration

The benchmarks consider isothermal, incompressible flows of immiscible fluids where the conservation of momentum and mass is described by the Navier-Stokes equations (2.4) and (2.5). Surface tension effects are important and may not be neglected. Their incorporation is up to the participants, and any of the methods described in Chapter 5 could for example be used.

The initial configuration, shown in Figure 6.2, is identical for both test cases and consists of a circular bubble of radius $r_0 = 0.25$ centered at $[0.5, 0.5]$ in a $[1 \times 2]$ rectangular domain. The density of the bubble is smaller than that of the surrounding fluid ($\rho_2 < \rho_1$) so that it will rise upwards. No-slip boundary conditions ($\mathbf{u} = 0$) are used at the top and bottom boundaries, whereas the free slip condition ($\hat{\mathbf{n}} \cdot \mathbf{u} = 0$, $\hat{\mathbf{t}} \cdot \hat{\mathbf{n}} \cdot \left(\mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T)\right) = 0$, $\hat{\mathbf{t}}$ - the tangential vector) is imposed on the vertical walls. The benchmarks are restricted to two dimensions since both computational complexity and time is greatly reduced.

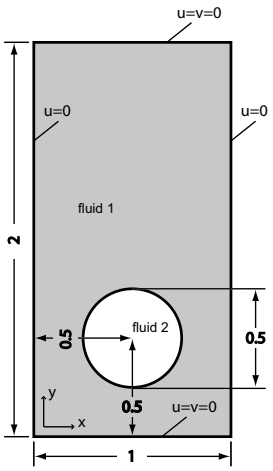


Figure 6.2: Initial configuration and boundary conditions.

6.1.2 Classification

The dimensionless Reynolds and Eötvös numbers are here specificly defined to relate to bubbles (and drops) as

$$Re = \frac{\rho_1 \sqrt{g} (2r_0)^{3/2}}{\mu_1}, \qquad Eo = \frac{4\rho_1 g r_0^2}{\sigma},$$

where a subscript 1 refers to the surrounding heavier fluid and 2 to the lighter fluid of the bubble. Moreover, r_0 is the initial radius of the bubble, and g is the gravitational constant. The density ratio ρ_1/ρ_2 and viscosity ratio μ_1/μ_2 finally help to fully classify the test cases.

6.1.3 Parameters

Table 6.1 lists the fluid and physical parameters which specify the test cases. The evolution of the bubbles should be tracked for 3 time units during which the defined benchmark quantities should be measured. The first test case models a rising bubble with $Re = 35$, $Eo = 10$, and both density and viscosity ratios equal to 10. According to the experimental studies by Clift et al. [19] such a bubble will end up in the ellipsoidal regime (see Figure 2.3). Assuming that this also is true for a two-dimensional bubble, it would mean that surface tension effects are strong enough to hold the bubble together and thus one should not expect any break up to occur in this test case.

Test Case	ρ_1	ρ_2	μ_1	μ_2	g	σ	Re	Eo	ρ_1/ρ_2	μ_1/μ_2
1	1000	100	10	1	0.98	24.5	35	10	10	10
2	1000	1	10	0.1	0.98	1.96	35	125	1000	100

Table 6.1: Physical parameters and dimensionless numbers defining the benchmark test cases.

The second and more challenging test case models a rising bubble with $Re = 35$, $Eo = 125$, and with large density and viscosity ratios (1000 and 100). This bubble lies somewhere between the skirted and dimpled ellipsoidal-cap regimes indicating that break up can possibly occur [19], which will present additional challenges to the different interface tracking algorithms.

6.1.4 Benchmark quantities

Visual comparison of the results, and in particular visualization of the bubble interface, is one obvious way to compare simulations. However, this does not allow us to rigorously determine how accurate our simulations really are and, perhaps more interestingly, *how much numerical effort is required to attain a certain accuracy*? The following quantities, which will be used to assist in describing the temporal evolution of the bubbles quantitatively, are therefore introduced.

Point Quantities. Positions of various points can be used to track the translation of bubbles. It is common to use the centroid or center of mass [16, 17, 71, 99], defined by

$$\mathbf{X}_c = (x_c, y_c) = \frac{\int_{\Omega_2} \mathbf{x}\, dx}{\int_{\Omega_2} 1\, dx}$$

where Ω_2 denotes the region that the bubble occupies. Other points could be the absolute top or bottom of a bubble [16].

Mass/Area. An obvious quantity to measure is mass/area conservation which ideally should be preserved perfectly. This is one of the most common measures that is used in validation of interfacial flow algorithms. Although it does give information regarding the potential accuracy of the simulation, it is not sufficient to determine whether the result is converged or even physically correct.

Circularity. The "degree of circularity", introduced by Wadell [113], can in two dimensions be defined as

$$\phi = \frac{P_a}{P_b} = \frac{\text{perimeter of area-equivalent circle}}{\text{perimeter of bubble}} = \frac{\pi d_a}{P_b}\,.$$

Here, P_a denotes the perimeter or circumference of a circle with diameter d_a which has an area equal to that of a bubble with perimeter P_b. For a perfectly circular bubble or drop the circularity will be equal to unity and decrease as the bubble is deformed.

Rise Velocity. The mean velocity with which a bubble is rising or moving is a particularly interesting quantity since it not only measures how the interface tracking algorithm behaves but also the quality of the overall solution. The mean bubble velocity is defined as

$$\mathbf{U}_c = \frac{\int_{\Omega_2} \mathbf{u}\; dx}{\int_{\Omega_2} 1\; dx}$$

where Ω_2 again denotes the region that the bubble occupies. A variant of this is simply to use the velocity at the centroid of the bubble $\mathbf{u}(\mathbf{X}_c)$. The velocity component in the direction opposite to the gravity vector is usually denoted as rise velocity V_c, for which the stationary limit is called terminal velocity. Both rise and terminal velocities are for example used in references [17, 97].

6.1.5 Error quantification

The temporal evolution of the computed benchmark quantities can be measured against suitable reference solutions to establish the following relative error norms

$$\begin{aligned}
l_1\ error: &\qquad ||e||_1 = \frac{\sum_{t=1}^{NTS} |q_{t,ref} - q_t|}{\sum_{t=1}^{NTS} |q_{t,ref}|},\\
l_2\ error: &\qquad ||e||_2 = \left(\frac{\sum_{t=1}^{NTS} |q_{t,ref} - q_t|^2}{\sum_{t=1}^{NTS} |q_{t,ref}|^2} \right)^{1/2},\\
l_\infty\ error: &\qquad ||e||_\infty = \frac{\max_t |q_{t,ref} - q_t|}{\max_t |q_{t,ref}|},
\end{aligned}$$

where q_t is the temporal evolution of quantity q.

The solution computed on the finest grid with the smallest time step is usually taken as a reference solution $q_{t,ref}$. Interpolation should be appropriately applied if there are more time steps or sample points (NTS) for the reference solution than the solutions q_t for which the error norms should be computed.

With the relative errors established and CPU times measured it is then easy to see how much effort is required to establish a certain accuracy. Additionally, convergence rates for the quantities can also be computed as

$$\text{ROC} = log_{10}(\frac{||e^{l-1}||}{||e^l||})/log_{10}(\frac{h^{l-1}}{h^l})$$

where l is the grid level and h the mean cell edge length.

6.2 Initial benchmark studies

Benchmarking and validation efforts have been initiated by the author with the aim of producing grid independent reference solutions for the proposed test cases. Extensive initial studies have also been carried out by the three research groups listed in Table 6.2, for which corresponding methods and codes are described in this section. As shall be seen in the following, the study found very good agreement for the first test case while the second proved significantly more challenging. The benchmark study has been submitted to IJNMF to both provide the research community with reference data and also elicit more participation from additional groups [51].

	Group and Affiliation	Code/Method
1	Uni. Dortmund, Inst. of Applied Math.	**TP2D**
	S. Turek, D. Kuzmin, S. Hysing	*FEM-Level Set*
2	EPFL Lausanne, Inst. of Analysis and Sci. Comp.	**FreeLIFE**
	E. Burman, N. Parolini	*FEM-Level Set*
3	Uni. Magdeburg, Inst. of Analysis and Num. Math.	**MooNMD**
	L. Tobiska, S. Ganesan	*FEM-ALE*

Table 6.2: Participating groups and methods.

Group 1: TP2D

The TP2D code (short for **T**ransport **P**henomena in **2D**) is the extension of the FeatFlow [102] incompressible flow solver to treat immiscible fluids with the level set method (as described in Chapters 2-5).

The benchmarks did not require artificial stabilization of the convective terms in the momentum equations due to the low flow velocities. Numerical stabilization was only used with the level set equation and then in the form of high order FEM-TVD. Reinitialization of the level set field was applied every 20 time steps and artificial mass conservation in the form of lowering/raising the level set field was used after every step. Although the mass conservation only involved adding a constant of magnitude 10^{-5} or less its inclusion was necessary, especially on coarser grids, in order to prevent accumulation of mass errors.

Surface tension effects were incorporated by straight line approximation of the interface contours and direct integration over these line segments instead of using the usual continuum surface force approach.

Group 2: FreeLIFE

The FreeLIFE (**Free**-Surface **Li**brary of **F**inite **E**lement) software is an incompressible flow solver for the solution of free-surface two-fluid problems. The software is based on the numerical solution of the Navier-Stokes equations with variable density and viscosity. In order to track the location of the interface between the two fluids, where discontinuities in the density and viscosity occur, a level set approach is adopted. The Navier-Stokes equations are therefore cou-

pled with an advection equation for the level set function whose zero level set defines the interface location [77, 78].

The spatial discretization is based on a piecewise linear finite-element approach. In particular, the Navier-Stokes problem is solved using P_1-isoP_2 elements for the velocity and P_1 for the pressure. The sub-grid topology associated with the P_1-isoP_2 element is also exploited for the solution of the level set transport equation, where the local sub-grid edge stabilization introduced in [13] has been adopted. In the simulations presented here, to be consistent with the method of group 1, a mass correction step has been added which consists in lowering/raising the level set function by a constant value in order to guarantee a global mass conservation.

The level set reinitialization is based on a new method proposed in [77] consisting of a local (L^2-projection based) reconstruction of the distance function in the neighborhood of the interface and a fast marching strategy for the far field [14].

The FreeLIFE software has been used for the simulation of a variety of test cases concerning laminar two-fluid flows. The results of these simulations have been presented and discussed in [22, 77, 78]. The method has been implemented in a finite element library which is restricted to two-dimensional problems. However, the proposed methodology is well suited for the solution of three-dimensional problems. This approach is currently being extended to three-dimensional problems in the framework of the library *LifeV*, a three dimensional finite element code developed in a joint collaboration between Ecole Polytechnique Fédérale de Lausanne (CMCS), Politecnico di Milano (MOX) and INRIA (BANG).

Group 3: MooNMD

MooNMD stands for **M**athematics and **o**bject **o**riented **N**umerics in **M**ag**D**eburg [55]. It is a program package based on mapped finite element methods for discretizing partial differential equations. In particular, it covers the solution of the incompressible Navier-Stokes equations by inf-sup stable isoparametric finite elements [39] and the solution of convection-diffusion equations by stabilized finite element methods. It has been extended to treat incompressible two-phase flows with capillary forces using the arbitrary Lagrangian-Eulerian (ALE) approach.

For the benchmarks, the velocity components were discretized on simplex grids by quadratic basis functions enriched with cubic bubble functions, and the pressure by discontinuous piecewise linear elements. In this way, high accuracy could be achieved and spurious velocities suppressed [38]. It is worth to mention, that no mass correction step was applied. Furthermore, the curvature was replaced by the Laplace-Beltrami operator which could then be integrated by parts and thus reduced the smoothness requirements [4, 24, 85].

For the time discretization, the second order, strongly A-stable fractional-step-theta scheme was used [83]. In each time step the interface was fully resolved by the mesh, meaning that the interface was always aligned with cell edges. Three to four different initial meshes were generated using the mesh generator Triangle [91], by fixing 200-900 degrees of freedom on the interface. The movement of the interface was done in a Lagrangian manner after which the inner mesh points were fitted to the new interface by an elastic mesh update, that is by solving a linear elasticity problem [35]. It turned out that no remeshing,

to improve the mesh quality, was needed for long periods. During such a period the number of degrees of freedom was fixed. However, remeshing had to be applied after some time depending on the degree of deformation, which changed the number of degrees of freedom dynamically during the simulations.

6.3 Results for test case 1

In test case 1 the bubble, being initially circular, is stretched horizontally and first develops a dimple as it rises, but after some time proceeds to assume a more stable ellipsoidal shape. Figure 6.3 shows a typical time evolution of the interface shape (computed by group 1 on a $h = 1/160$ grid).

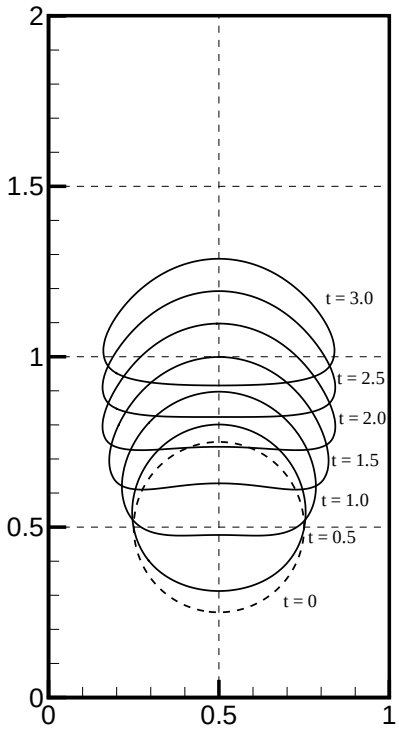


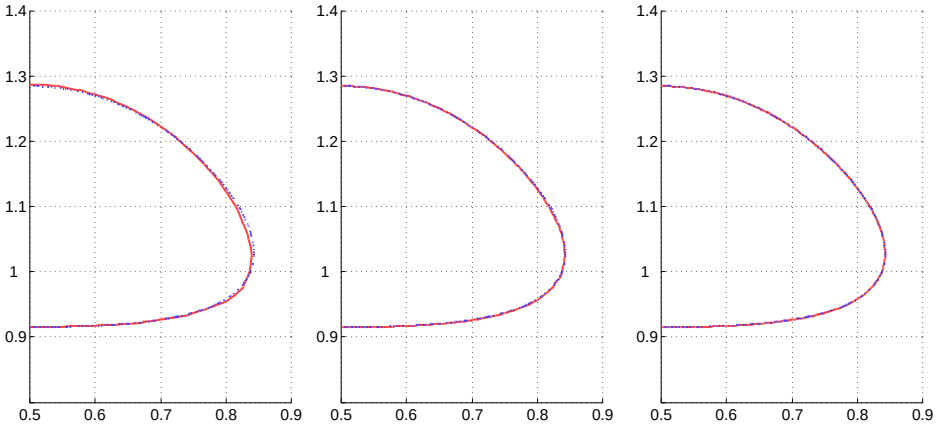
Figure 6.3: Typical interface evolution for test case 1.

6.3.1 Group 1: TP2D

The results for test case 1 computed with the TP2D code of group 1 are presented here. All computations were performed on rectangular tensor product grids with cell sizes $h = 1/[40, 80, 160, 320]$. The implicit 2nd order Crank-Nicolson scheme was used with the time step fixed to $\Delta t = h/16$. Table 6.3 shows the simulation statistics for the different grid levels where the number of elements is denoted by NEL, the total number of degrees of freedom by NDOF, and the total number of time steps by NTS. The time in seconds required for each computation is denoted by CPU which scaled by the number of time steps yields the factor CPU/TS. The Fortran 77 TP2D code was compiled with the PathScale v2.5 compiler suite and the computations were performed on servers with 2.4 GHz AMD Opteron processors.

$1/h$	NEL	NDOF	NTS	CPU	CPU/TS
40	3200	19561	1920	181	0.1
80	12800	77521	3840	1862	0.5
160	51200	308641	7680	20360	2.7
320	204800	1231681	15360	126373	8.2

Table 6.3: Simulation statistics for test case 1 and group 1 (TP2D).



(a) $h = 1/40$ (b) $h = 1/80$ (c) $h = 1/160$

Figure 6.4: Test case 1 bubble shapes for the TP2D code at time t=3. Coarse grid solutions (solid red) compared to the shape computed on the finest grid $h = 1/320$ (dashed blue).

In Figure 6.4 the symmetric right half of the coarse grid bubble shapes at the final time (t=3) are compared to the solution from the computation on the finest grid ($h = 1/320$). It is apparent that the solution on the coarsest grid $h = 1/40$ (Figure 6.4(a)) is already quite good but does visibly differ somewhat from the reference solution. The computation on a one level finer grid ($h = 1/80$ shown in Figure 6.4(b)) is clearly better and further refinements yield bubble shapes

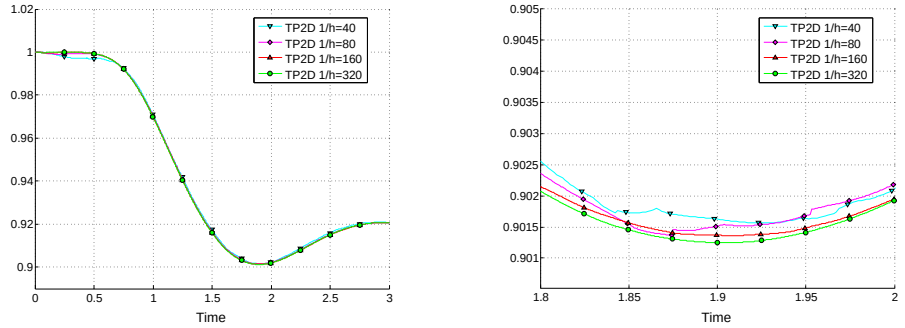
which are visually indistinguishable from the reference shape. Merely looking at the bubble shapes is not sufficient to say anything about the accuracy on the finer grids, and it is now that the previously defined benchmark quantities become particularly useful.

The relative error norms for the circularity, center of mass, and rise velocity are shown in Table 6.4 together with the estimated convergence rates (ROC). The reference solution is as before taken as the solution from the computation on the finest grid ($h = 1/320$). It is evident that all quantities converge with a more than linear convergence order, approaching quadratic convergence in the l_1 and l_2 norms. In the maximum norm the convergence order decreased to 1.16 for the circularity and 1.39 for the rise velocity.

$1/h$	$\|\|e\|\|_1$	ROC_1	$\|\|e\|\|_2$	ROC_2	$\|\|e\|\|_\infty$	ROC_∞
			Circularity			
40	1.00e-03		1.22e-03		2.89e-03	
80	3.01e-04	1.74	3.63e-04	1.75	9.67e-04	1.58
160	8.83e-05	1.77	1.10e-04	1.72	4.32e-04	1.16
			Center of mass			
40	2.65e-03		2.99e-03		3.56e-03	
80	9.64e-04	1.46	1.02e-03	1.55	1.14e-03	1.64
160	2.62e-04	1.88	2.71e-04	1.91	2.96e-04	1.95
			Rise velocity			
40	1.19e-02		1.29e-02		1.49e-02	
80	2.90e-03	2.04	3.07e-03	2.07	5.08e-03	1.55
160	7.73e-04	1.91	7.85e-04	1.97	1.94e-03	1.39

Table 6.4: Relative error norms and convergence orders for test case 1 and group 1 (TP2D).

The following figures depict the time evolution of the benchmark quantities for test case 1 and group 1 (TP2D). From Figure 6.5(a), which shows the circularity, it is quite hard to discern any significant differences between the different grids. Only for the coarsest grid ($h = 1/40$) is it possible to see some deviations; the circularity drops too quickly up until t=0.7, after which the correct solution behavior is recovered. A close up around the point of minimum circularity is shown in Figure 6.5(b) from where it is possible to see the convergence behavior. Most notable is that there are irregularities or small jumps in the curves for the two coarsest grids which is due to the reinitialization procedure which was applied every 20 time steps. The minimum circularity converges towards a value of 0.9013 around t=1.90 (see Table 6.5).



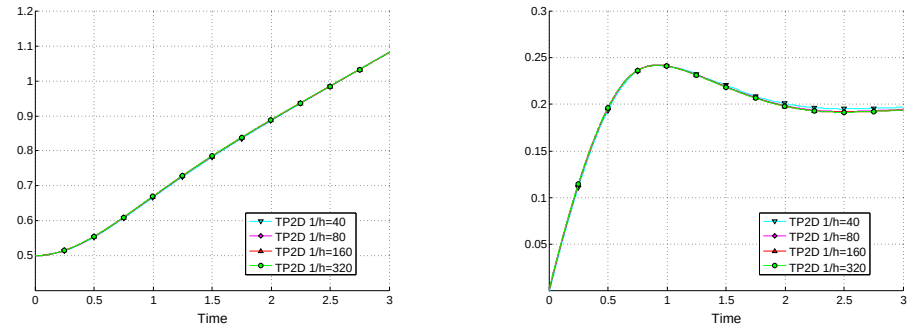
(a) Circularity (b) Close-up of the circularity

Figure 6.5: Temporal evolution of the circularity (TP2D Case 1).

$1/h$	40	80	160	320
ϕ_{min}	0.9016	0.9014	0.9014	0.9013
$t\|_{\phi=\phi_{min}}$	1.9234	1.8734	1.9070	1.9041
$V_{c,max}$	0.2418	0.2418	0.2419	0.2417
$t\|_{V_c=V_{c,max}}$	0.9141	0.9375	0.9281	0.9213
$y_c(t=3)$	1.0818	1.0810	1.0812	1.0813

Table 6.5: Minimum circularity and maximum rise velocity, with corresponding incidence times, and the final position of the center of mass for test case 1 and group 1 (TP2D).

Both the center of mass, shown in Figure 6.6(a), and the mean rise velocity of the bubble, shown in Figure 6.6(b), converge very nicely. From Table 6.5 one can see that the maximum rise velocity of $V_{c,max} = 0.2417$ is attained quite early at time t=0.92. The center of mass of the bubble can asymptotically be described as a linear function of time and approaches $y_c = 1.0813$ towards the end of the simulation.



(a) Center of mass (b) Rise velocity

Figure 6.6: Center of mass and rise velocity (TP2D Case 1).

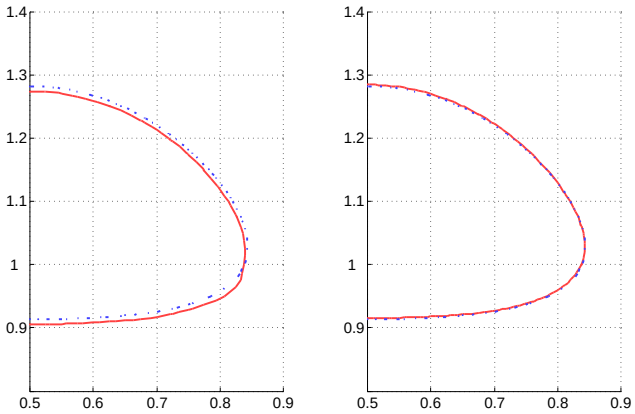
6.3.2 Group 2: FreeLIFE

The following results are computed with the FreeLIFE code of the second group. The computations were performed on simplex cells created by subdivision of regular quadrilaterals with element mesh sizes $h = 1/[40, 80, 160]$. The time step was chosen as $\Delta t = h/2$. Statistics and timings for the computations can be seen in Table 6.6.

$1/h$	NEL	NDOF	NTS	CPU	CPU/TS
40	6400	14145	240	257	1.1
80	25600	55485	480	4299	8.5
160	102400	219765	960	108846	113.2

Table 6.6: Simulation statistics for test case 1 and group 2 (FreeLIFE).

In Figure 6.7 the bubble shapes at the final time (t=3) with the different grid resolutions are compared. Although the interface contour from the solution on the coarsest grid seems to be offset in the y-direction the overall shape is correct, and when the grid is refined once it is not anymore possible to distinguish between the two bubbles (Figure 6.7(b)).



(a) $h = 1/40$ (b) $h = 1/80$

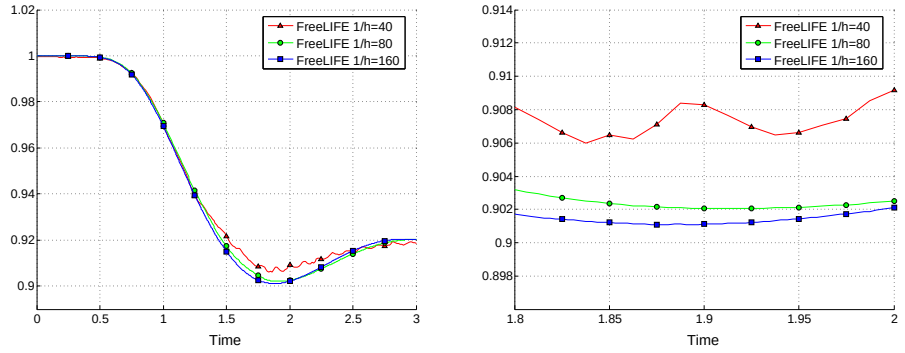
Figure 6.7: Test case 1 bubble shapes for the FreeLIFE code at time t=3. Coarse grid solutions (solid red) compared to the shape computed on the finest grid $h = 1/160$ (dashed blue).

A quantitative convergence analysis has been performed computing the relative errors for the circularity, center of mass, and rise velocity together with the estimated convergence rates (ROC), as defined in section 6.1.5. Here, the solution from the finest grid ($h = 1/160$) is taken as the reference solution. As can be seen from Table 6.7 the method gives a convergence order approaching 1.5 for the circularity and 2 for the rise velocity. The center of mass shows a good convergence behavior of about 3 in the l_1 and l_2 norms and 2 in the l_∞ norm.

$1/h$	$\|e\|_1$	ROC_1	$\|e\|_2$	ROC_2	$\|e\|_\infty$	ROC_∞
			Circularity			
40	2.61e-03		3.63e-03		8.09e-03	
80	1.05e-03	1.31	1.36e-03	1.41	2.51e-03	1.69
			Center of mass			
40	7.85e-03		8.14e-03		7.74e-03	
80	9.42e-04	3.06	1.25e-03	2.70	1.72e-03	2.17
			Rise velocity			
40	1.78e-02		1.95e-02		3.30e-02	
80	3.99e-03	2.16	5.54e-03	1.82	1.00e-02	1.72

Table 6.7: Relative error norms and convergence orders for test case 1 and group 2 (FreeLIFE).

Figure 6.8 depicts the circularity for the three different grid levels. Although the solution on the coarsest grid is highly oscillating the results converge toward the solution corresponding to the finest grid ($h = 1/160$). The maximum deformation of the bubble is reached at time t=1.88 where the circularity attains its minimum value of 0.9011 (see Table 6.8 and Figure 6.8(b)).



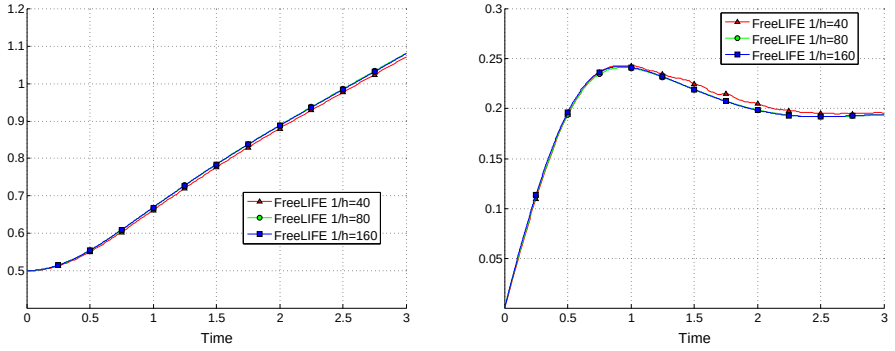
(a) Circularity (b) Close-up of the circularity

Figure 6.8: Temporal evolution of the circularity (FreeLIFE Case 1).

The time evolutions of the center of mass and mean rise velocity can be seen in Figures 6.9(a) and 6.9(b), respectively. Both these quantities seem to converge although the curve from the simulation on the coarsest grid deviates somewhat from the other two. The rise velocity reaches a maximum value of 0.2421 at time t=0.9313 and center of mass of the bubble reaches a height of 1.08 at the end of the simulation (see Table 6.8).

$1/h$	40	80	160
ϕ_{min}	0.9060	0.9021	0.9011
$t\|_{\phi=\phi_{min}}$	1.8375	1.9125	1.8750
$V_{c,max}$	0.2427	0.2410	0.2421
$t\|_{V_c=V_{c,max}}$	0.9000	0.9375	0.9313
$y_c(t=3)$	1.0715	1.0817	1.0799

Table 6.8: Minimum circularity and maximum rise velocity, with corresponding incidence times, and the final position of the center of mass for test case 1 and group 2 (FreeLIFE).



(a) Center of mass (b) Rise velocity

Figure 6.9: Center of mass and rise velocity (FreeLIFE Case 1).

6.3.3 Group 3: MooNMD

Simulations of test case 1 with the MooNMD code of the third group were performed with 200, 300, 600, and 900 degrees of freedom resolving the interface (denoted by NDOF$_{int}$). The computations were run on a server with a 2.16 GHz Intel processor, for which simulation statistics are given in Table 6.9.

NDOF$_{int}$	NEL(t=0)	NDOF	NTS	CPU	CPU/TS
200	595	17846	3000	11034	3.7
300	2640	24002	6000	25110	4.2
600	5534	50048	6000	58349	9.7
900	8066	72836	6000	180819	30.1

Table 6.9: Simulation statistics for test case 1 and group 3 (MooNMD).

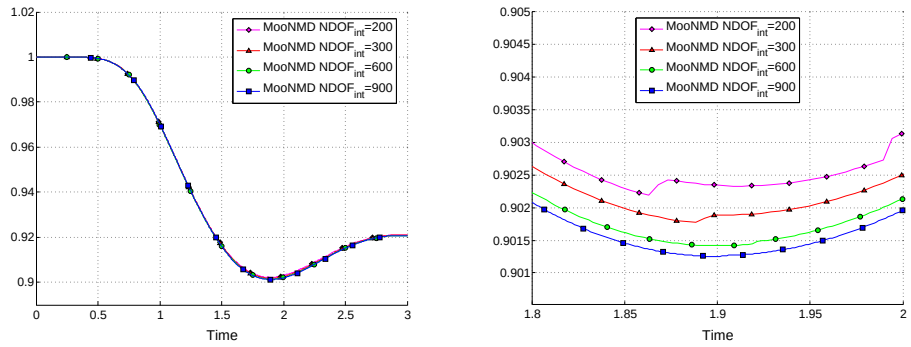
Table 6.10 shows the computed error norms and convergence orders for MooNMD. Since the finer meshes are not obtained by uniform refinement of the coarse mesh, but with the help of a mesh generator, h was replaced in the

formula for calculating the convergence rates by h^*, the edge length of the interface at $t = 0$. This is indicated in Table 6.10 by the notation ROC*. The center of mass and the rise velocity approach a convergence order of 3 and 2.2 in the l_1 and l_2 norms. In the l_∞ norm the convergence order decreases for the rise velocity. The circularity had the overall lowest convergence order of 1.3.

NDOF$_{int}$	$\|e\|_1$	ROC$_1^*$	$\|e\|_2$	ROC$_2^*$	$\|e\|_\infty$	ROC$_\infty^*$
			Circularity			
200	4.40e-04		5.99e-04		1.19e-03	
300	2.60e-04	1.30	3.40e-04	1.40	6.55e-04	1.47
600	1.07e-04	1.28	1.41e-04	1.27	2.90e-04	1.18
			Center of mass			
200	5.07e-04		7.91e-04		1.53e-03	
300	1.79e-04	2.57	2.87e-04	2.50	5.82e-04	2.38
600	1.66e-05	3.43	2.11e-05	3.76	3.85e-05	3.92
			Rise velocity			
200	2.87e-03		3.70e-03		5.96e-03	
300	1.18e-03	2.20	1.54e-03	2.17	2.48e-03	2.16
600	2.33e-04	2.34	3.10e-04	2.31	1.28e-03	0.95

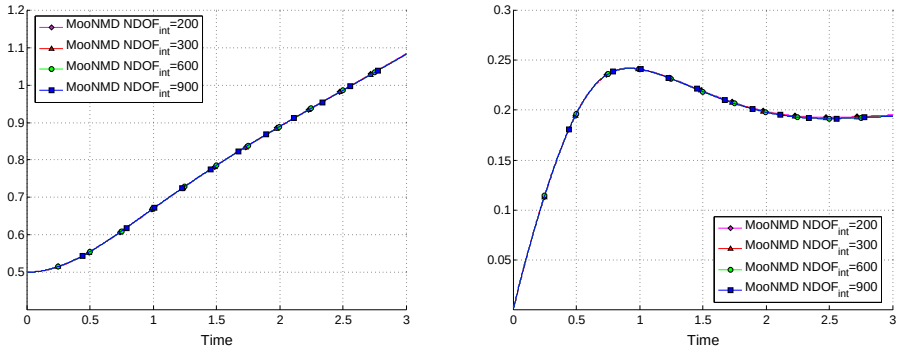
Table 6.10: Relative error norms and convergence orders for test case 1 and group 3 (MooNMD).

That this method had very small error levels, even on the coarsest grids, is apparent from Figures 6.10(a) and 6.11, which depict the circularity, center of mass, and the rise velocity. There is no real visible evidence that any of the curves differ, even for the coarser grids, until one looks much closer (see Figure 6.10(b)). Then it is possible to see that each grid refinement produces results that are closer to the curve corresponding to the computation on the finest grid.



(a) Circularity (b) Close-up of the circularity

Figure 6.10: Temporal evolution of the circularity (MooNMD Case 1).



(a) Center of mass (b) Rise velocity

Figure 6.11: Center of mass and rise velocity (MooNMD Case 1).

Table 6.11 shows the time and values of the minimum circularity, maximum rise velocity, and maximum position of the center of mass achieved during the simulations. For the finest grid a minimum circularity of 0.9013 was measured at time t=1.9. The rise velocity showed a very stable maximum, almost irrespective of grid level, with a value of 0.2417 recorded at times around 0.92. At the final time the center of mass of the bubble had reached a position of 1.0817.

$NDOF_{int}$	200	300	600	900
ϕ_{min}	0.9022	0.9018	0.9014	0.9013
$t\|_{\phi=\phi_{min}}$	1.8630	1.8883	1.9013	1.9000
$V_{c,max}$	0.2418	0.2417	0.2417	0.2417
$t\|_{V_c=V_{c,max}}$	0.9236	0.9236	0.9214	0.9239
$y_c(t=3)$	1.0833	1.0823	1.0818	1.0817

Table 6.11: Minimum circularity and maximum rise velocity, with corresponding incidence times, and the final position of the center of mass for test case 1 and group 3 (MooNMD).

6.3.4 Overall results for test case 1

Here the results from all groups computations on the finest grids are compared starting with the bubble shapes shown in Figure 6.12. No significant differences can really be seen at all and one would thus expect the computed benchmark quantities to be similarly close.

The curves for the circularity shown in Figure 6.13(a) does not reveal any significant differences between the groups. Only in the enlarged section around the minimum (Figure 6.13(b)) is it possible to see some separation between the curves. The curves of groups 1 (TP2D) and 3 (MooNMD) agree best, while the minima calculated by the 2nd group (FreeLIFE) is somewhat offset. This is also reflected by the actual values shown in Table 6.12. From there it is possible to conclude that the minimum circularity will have a value of 0.9012±0.0001 and occur around t=1.9.

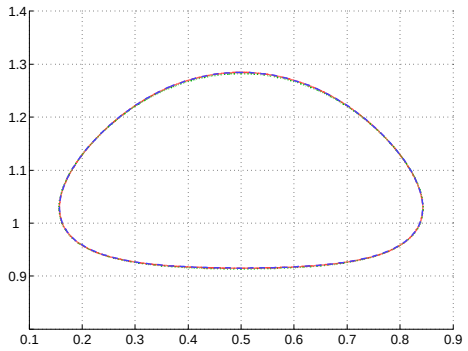
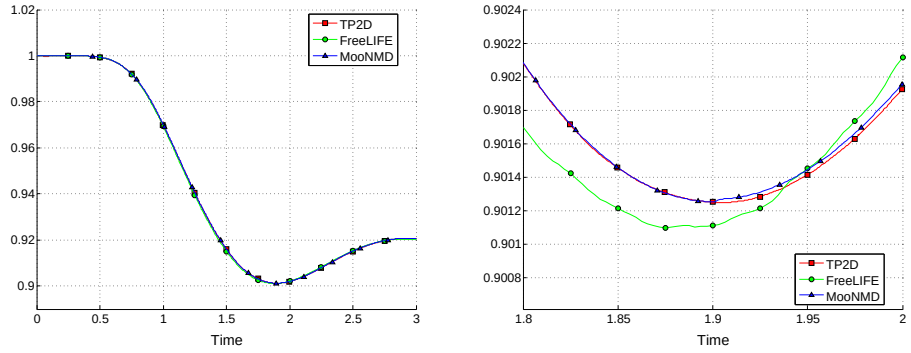


Figure 6.12: Bubble shapes at the final time (t=3) for test case 1 (TP2D (solid red), FreeLIFE (dotted green), and MooNMD (dashed blue)).

Group	1	2	3
ϕ_{min}	0.9013	0.9011	0.9013
$t\|_{\phi=\phi_{min}}$	1.9041	1.8750	1.9000
$V_{c,max}$	0.2417	0.2421	0.2417
$t\|_{V_c=V_{c,max}}$	0.9213	0.9313	0.9239
$y_c(t=3)$	1.0813	1.0799	1.0817

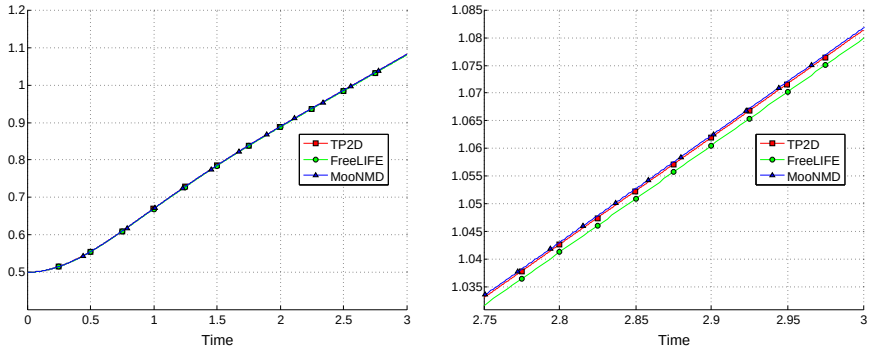
Table 6.12: Minimum circularity and maximum rise velocity, with corresponding incidence times, and the final position of the center of mass for test case 1 (all groups).



(a) Circularity (b) Close-up of the circularity

Figure 6.13: Circularity for test case 1 (all groups).

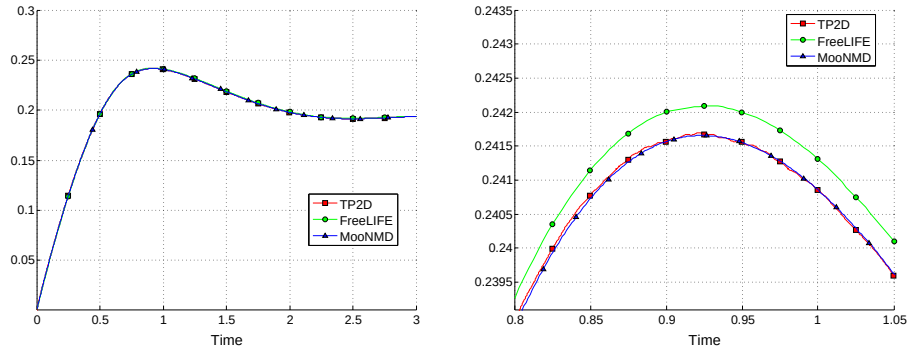
The time evolution of the center of mass (Figure 6.14) essentially shows the same behavior as the circularity. The curves for groups 1 and 3 agree well while the bubble of the second group seems to rise with the same speed but has been somewhat delayed (negative offset). From Table 6.12 one can see that the center of mass of the bubble reaches a position of 1.081±0.001 at the final time.



(a) Center of mass (b) Close-up of the center of mass

Figure 6.14: Center of mass for test case 1 (all groups).

The last benchmark quantity to be examined for this test case is the mean rise velocity of the bubble, which is shown in Figure 6.15. Again the curves for groups 1 (TP2D) and 3 (MooNMD) agree very well while the curve for FreeLIFE (group 2) is slightly positively offset. The overall maximum rise velocity has a magnitude of 0.2419±0.0002 and occurs between times t=0.921 and t=0.932.



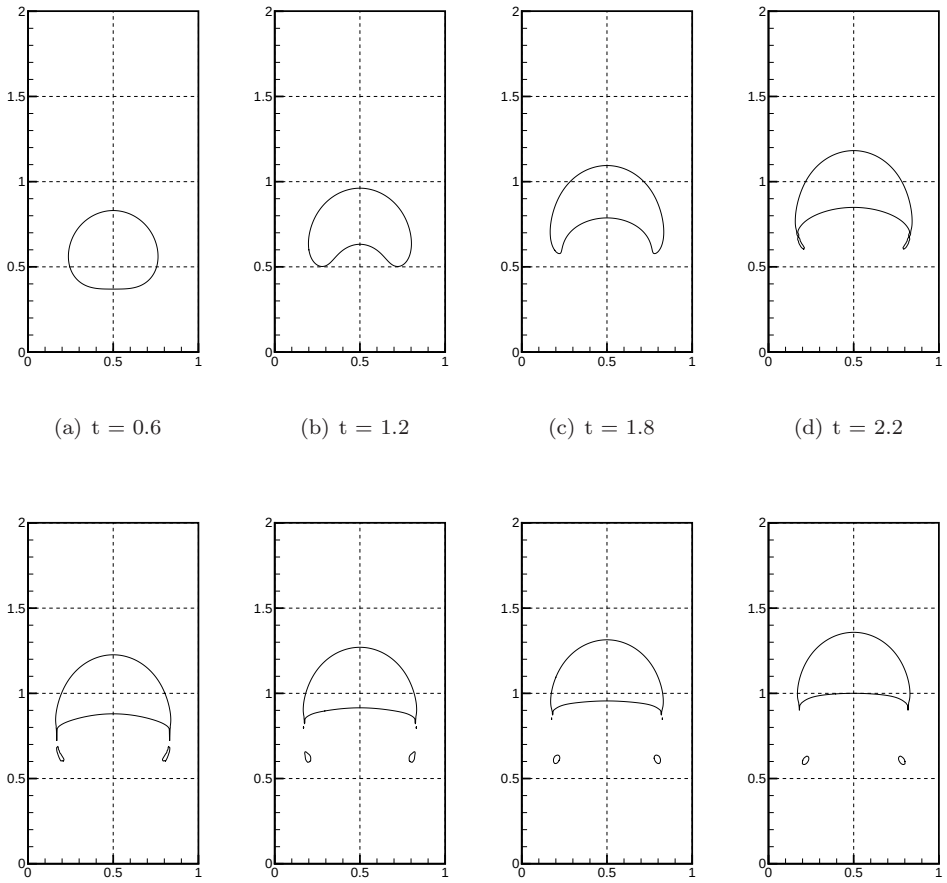
(a) Rise velocity (b) Close-up of the rise velocity

Figure 6.15: Rise velocity for test case 1 (all groups).

To summarize, preliminary studies of test case 1 have been conducted which have made it possible to establish a target reference range for each of the benchmark quantities. However, the three different codes did not agree perfectly and one must conclude that numerical simulation of a single rising bubble, undergoing quite moderate deformation, is still not a trivial task.

6.4 Results for test case 2

Figure 6.16 shows snapshots of the time evolution of the bubble (computed by group 1 on a $h = 1/160$ grid). Although the bubbles in both test cases rise with approximately the same speed, the decrease in surface tension causes this bubble to assume a more convex shape and develop thin filaments which eventually break off. The time of break up is in this simulation predicted to occur between t=2.2 and 2.4, as is evident from Figures 6.16(d) and 6.16(e). After the break up small satellite droplets trail the bulk of the main bubble, which eventually assumes the shape of a dimpled cap.



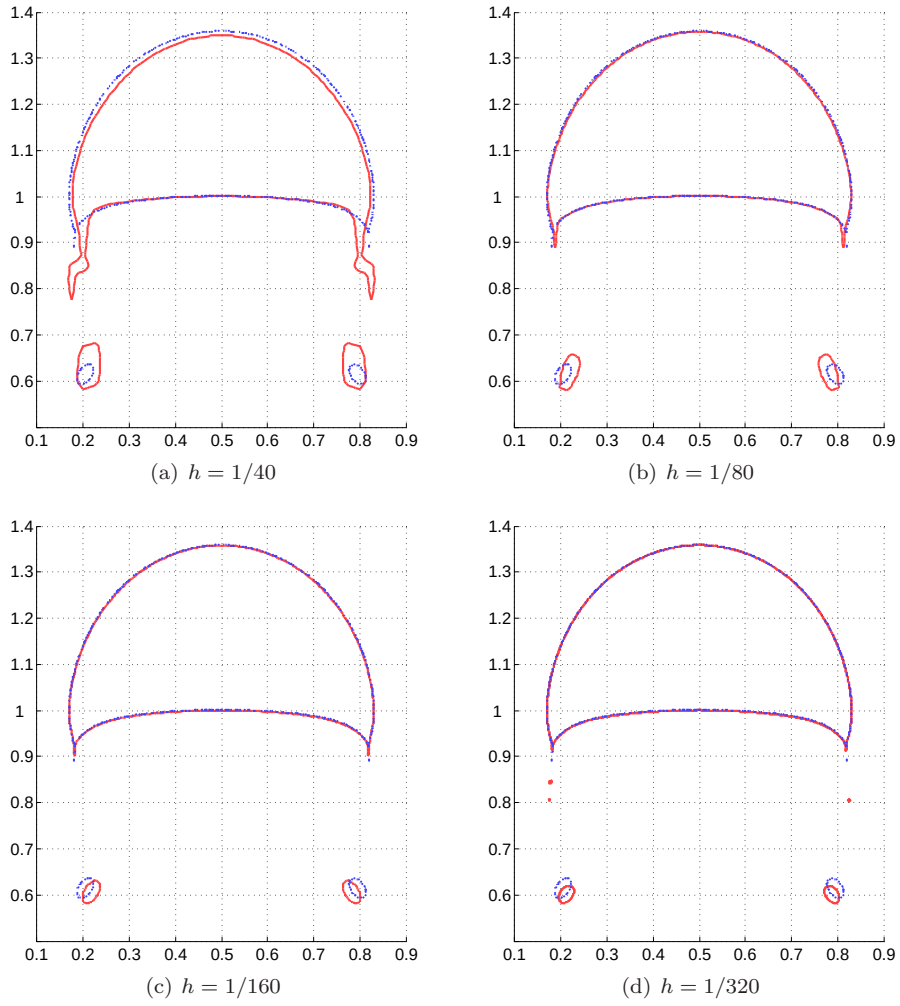
(a) t = 0.6 (b) t = 1.2 (c) t = 1.8 (d) t = 2.2

(e) t = 2.4 (f) t = 2.6 (g) t = 2.8 (h) t = 3.0

Figure 6.16: Typical time evolution of the interface for test case 2.

6.4.1 Group 1: TP2D

The bubble shapes at the final time (t=3), computed by the TP2D code of group 1, are shown in Figure 6.17. First of all one can see that the simulation on the coarsest grid produced a rather unphysical break up behavior, producing sharp edged trailing filaments (Figure 6.17(a)). The shapes computed on the finer grids did not have these filaments and seemed to converge for the main bulk bubble. The two small satellite droplets were apparently the most difficult to correctly capture since their shape and position even differed on the two finest grids (Figure 6.17(d)).



(a) $h = 1/40$ (b) $h = 1/80$

(c) $h = 1/160$ (d) $h = 1/320$

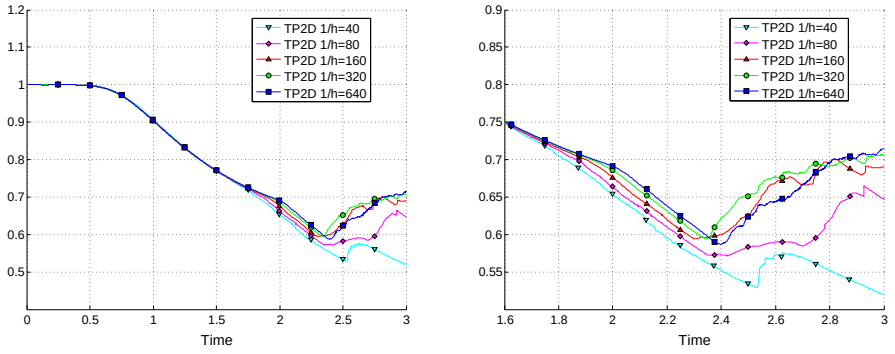
Figure 6.17: Test case 2 bubble shapes for the TP2D code at time t=3. Coarse grid solutions (solid red) compared to the shape computed on the finest grid $h = 1/640$ (dashed blue).

Table 6.13 shows the simulation statistics and timings for this test case and the TP2D code. The CPU times have increased by more than a factor of two compared to the timings from the first test case, shown in Table 6.3. This is mostly due to the increased number of multigrid iterations required to solve the pressure Poisson equation which now converges more slowly due to the 100 times larger jump in the density.

$1/h$	NEL	NDOF	NTS	CPU	CPU/TS
40	3200	19561	1920	410	0.2
80	12800	77521	3840	4351	1.1
160	51200	308641	7680	47919	6.2
320	204800	1231681	15360	713816	46.5
640	819200	4920961	30720	2967465	96.6

Table 6.13: Simulation statistics for test case 2 and group 1 (TP2D).

The circularity shown in Figure 6.18(a) is constant until t=0.5. It then decreases more or less linearly until somewhere between t=2.2 and t=2.6 where there is a sharp inflection point (see Figure 6.18(b)). This point should be very close to the time of break up since the thin elongated filaments, due to the high curvature and surface tension, shrink quite rapidly thereafter. This is also consistent with Figures 6.16(d)-6.16(f). The curves for all grid levels agree very well until t=1.7 where they start to deviate from each other. Although the deviations are also apparent from looking at the values of the minima in Table 6.14, the numbers point towards a minimum circularity of 0.59±0.05 occurring between times 2.3 and 2.4.



(a) Circularity (b) Close-up of the circularity

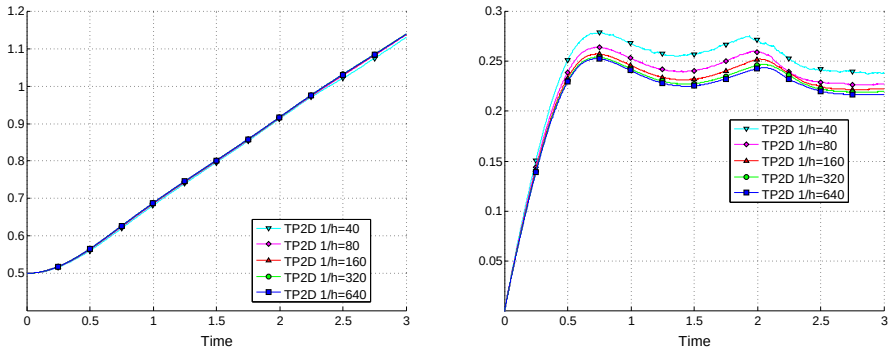
Figure 6.18: Temporal evolution of the circularity (TP2D Case 2).

The time evolutions of the center of mass and mean rise velocity of the bubble are shown in Figures 6.19(a) and 6.19(b), respectively. The center of mass moves similar to the first test case, reaching a slightly higher position of 1.138 at the end of the simulation (Table 6.14). There are virtually no differences between the curves for the different grids. For the mean rise velocity on the other hand one can see that the curves corresponding to simulations computed on coarser grids

$1/h$	40	80	160	320	640
ϕ_{min}	0.5193	0.5717	0.5946	0.5943	0.5869
$t\|_{\phi=\phi_{min}}$	3.0000	2.4266	2.2988	2.3439	2.4004
$V_{c,max\ 1}$	0.2790	0.2638	0.2570	0.2538	0.2524
$t\|_{V_c=V_{c,max\ 1}}$	0.7641	0.7250	0.7430	0.7340	0.7332
$V_{c,max\ 2}$	0.2749	0.2597	0.2522	0.2467	0.2434
$t\|_{V_c=V_{c,max\ 2}}$	1.9375	1.9688	2.0234	2.0553	2.0705
$y_c(t=3)$	1.1303	1.1370	1.1377	1.1387	1.1380

Table 6.14: Minimum circularity and maximum rise velocities, with corresponding incidence times, and the final position of the center of mass for test case 2 and group 1 (TP2D).

differ quite much from, but also converge nicely toward, the fine grid solutions. Instead of the single velocity maximum found in the first test case there are now two, the first occurring at time 0.7332 with a magnitude of 0.2524 and the second one at t=2.0705 with a slightly smaller magnitude of 0.2434. Lastly note that it is not possible to see when the break up occurs for these two benchmark quantities in contrast to the circularity.

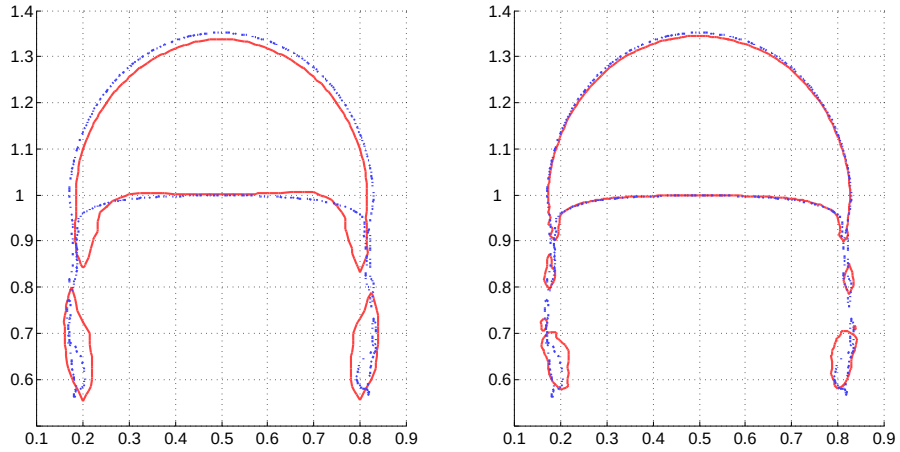


(a) Center of mass (b) Rise velocity

Figure 6.19: Center of mass and rise velocity (TP2D Case 2).

6.4.2 Group 2: FreeLIFE

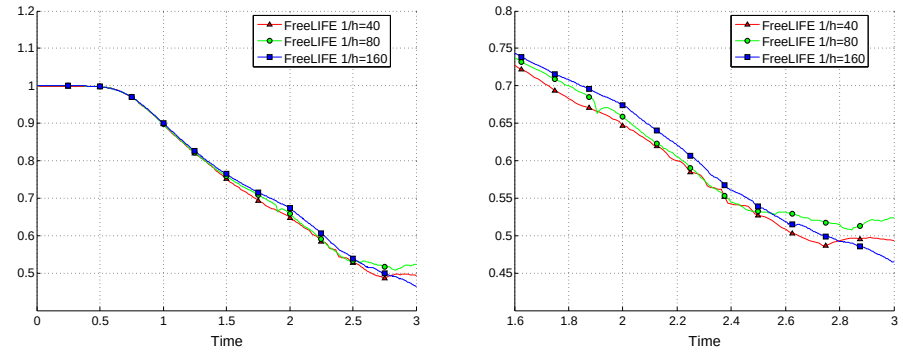
The bubble shapes at the final time (t=3) computed with the FreeLIFE code on the three grid levels $h = 1/[40, 80, 160]$ are presented in Figure 6.20. Although some sharp edged filaments are present, despite refining the grids, the shapes do seem to converge towards the solution obtained on the finest grid. The main bulk of the bubble appears to be the easiest to capture correctly, showing only minor visible differences between the two finest grids (Figure 6.20(b)).



(a) $h = 1/40$ (b) $h = 1/80$

Figure 6.20: Test case 2 bubble shapes for FreeLIFE at time t=3. Coarse grid solutions (solid red) compared to the shape computed on the finest grid $h = 1/160$ (dashed blue).

The curves for the circularity (Figures 6.21(a) and 6.21(b)) agree well and show a typical convergence behavior up to t=1.8 after which the bubble breaks up and no convergence trend can be seen anymore. Since the thin filaments do not retract after break up has occurred there is no clear inflection point which could indicate the time of break up. The minimum circularity can thus be found towards the very end of the simulations, as can be seen in Table 6.15.



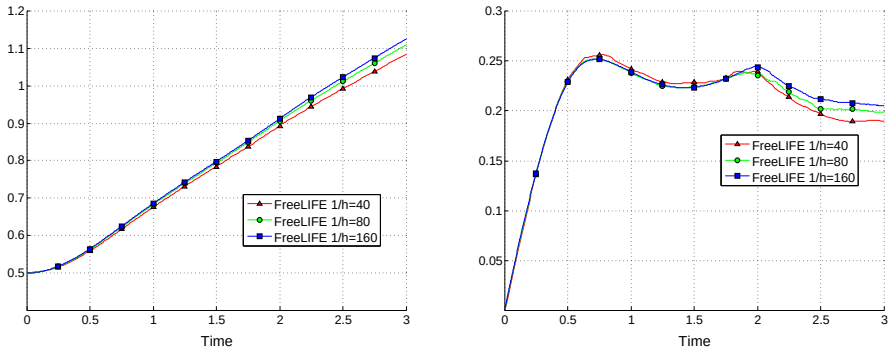
(a) Circularity (b) Close-up of the circularity

Figure 6.21: Temporal evolution of the circularity (FreeLIFE Case 2).

$1/h$	40	80	160
$\not\epsilon_{min}$	0.4868	0.5071	0.4647
$t\|_{\not\epsilon=\not\epsilon_{min}}$	2.7500	2.8438	3.0000
$V_{c,max\ 1}$	0.2563	0.2518	0.2514
$t\|_{V_c=V_{c,max\ 1}}$	0.7750	0.7188	0.7281
$V_{c,max\ 2}$	0.2397	0.2384	0.2440
$t\|_{V_c=V_{c,max\ 2}}$	1.9875	1.9062	1.9844
$y_c(t=3)$	1.0843	1.1099	1.1249

Table 6.15: Minimum circularity and maximum rise velocities, with corresponding incidence times, and the final position of the center of mass for test case 2 and group 2 (FreeLIFE).

The vertical position of the center of mass, shown in Figure 6.22(a), converges better than the circularity. The thin filaments apparently do not influence the overall movement too much since the curves are still approximately linear. A position of 1.1249 is reached by the bubble at the end of the simulation on the finest grid (Table 6.15). A very good agreement can be seen for the curves describing the rise velocity up until the first maximum, occurring at t=0.7281 with a magnitude of 0.2514 on the finest grid (Figure 6.22(b) and Table 6.15). From then on the curve corresponding to the simulation on the coarsest grid starts to show a somewhat irregular and oscillatory behavior. The other two curves corresponding to the finest grids keep in close contact until the second maximum from where all curves show minor irregularities (most likely due to some oscillations in the velocity field close to the interface).

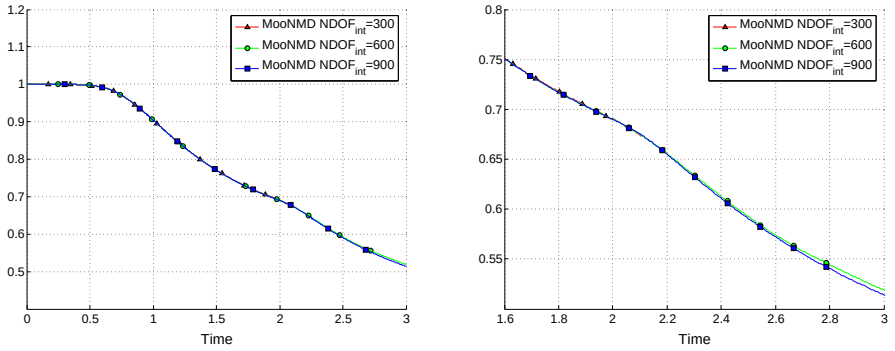


(a) Center of mass (b) Rise velocity

Figure 6.22: Center of mass and rise velocity (FreeLIFE Case 2).

6.4.3 Group 3: MooNMD

For the third group (MooNMD) the results are as consistent as for the first test case with one exception, the simulation on the coarsest grid ($\text{NDOF}_{int} = 300$) failed at t=2.1 due to the formation of very computationally unfavorable cell shapes in the thin filamentary regions. The Lagrangian approach used here could not treat break up automatically, and thus the bubble kept deforming more and more. This is evident from the curves for the circularity which after the initial period decreases monotonically (Figures 6.23(a) and 6.23(b)).



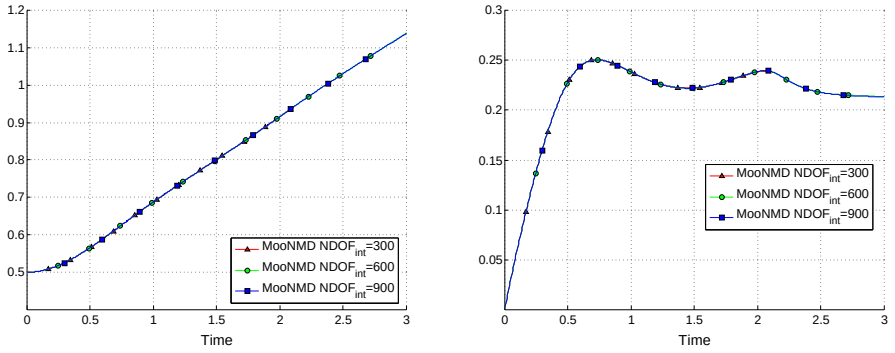
(a) Circularity (b) Close-up of the circularity

Figure 6.23: Temporal evolution of the circularity (MooNMD Case 2).

NDOF_{int}	300	600	900
ϕ_{min}	-	0.5191	0.5144
$t\|_{\phi=\phi_{min}}$	-	3.0000	3.0000
$V_{c,max\ 1}$	0.2503	0.2502	0.2502
$t\|_{V_c=V_{c,max\ 1}}$	0.7317	0.7317	0.7317
$V_{c,max\ 2}$	0.2390	0.2393	0.2393
$t\|_{V_c=V_{c,max\ 2}}$	2.0650	2.0600	2.0600
$y_c(t=3)$	-	1.1380	1.1376

Table 6.16: Minimum circularity and maximum rise velocities, with corresponding incidence times, and the final position of the center of mass for test case 2 and group 3 (MooNMD).

Figure 6.24(a) shows the linear time evolution of the center of mass. The curves agree completely irrespective of grid level and from Table 6.16 it is possible to see that the center of the bubble reaches a height of 1.1376 at the end of the simulation. Table 6.16 also shows very consistent results for the two maxima found in the mean rise velocity. The first maximum occurred at t=0.7317 with a magnitude of 0.2502 while the second peak came later at t=2.06 with a velocity of 0.2393. These results are also perfectly mirrored by the plotted curves in Figure 6.24(b).



(a) Center of mass (b) Rise velocity

Figure 6.24: Center of mass and rise velocity (MooNMD Case 2).

6.4.4 Overall results for test case 2

To compare the results for the second test case, a rising bubble with a significantly lower density compared to that of the surrounding fluid, the bubble shapes computed by the different codes are plotted against each other in Figure 6.25. It is evident that although all codes predict a similar shape for the main bulk of the bubble, there is no agreement with respect to the thin filamentary regions. The two first codes (TP2D and FreeLIFE) can handle break up automatically but do not, with the employed discretizations in space and time, agree what happens after. Since no criteria for the break up of the bubble has been implemented in the third code (MooNMD), the long thin trailing filaments remain intact.

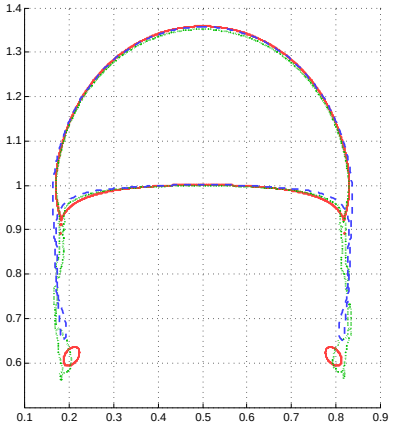
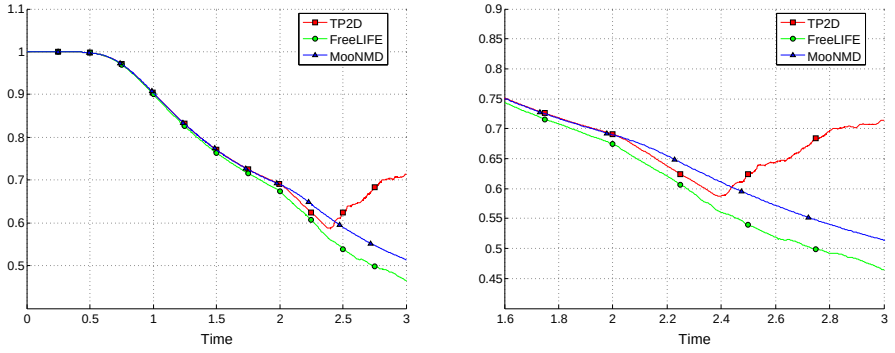


Figure 6.25: Bubble shapes at the final time (t=3) for test case 2 (TP2D (solid red), FreeLIFE (dotted green), and MooNMD (dashed blue)).

The circularity for all groups (Figure 6.26(a)) agree very well with each other until about t=1.75 after which significant differences start to appear. The main difference is that the circularity predicted by the TP2D code (group 1) starts to increase after the break up due to the retraction of the filaments (Figure 6.26(b)). Table 6.17 clearly shows how there is no real agreement between the codes concerning the minimum circularity.



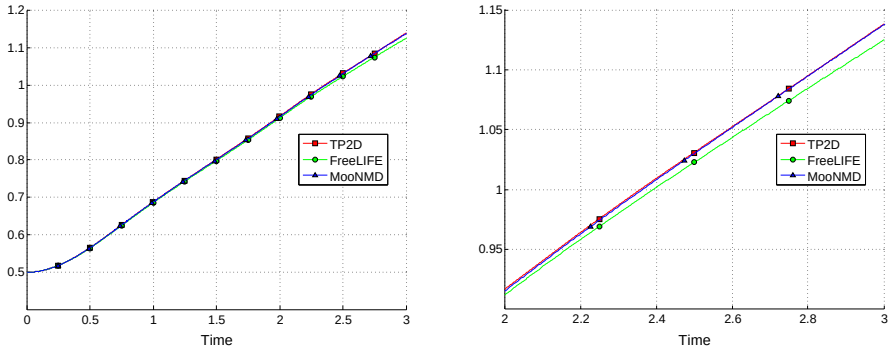
(a) Circularity (b) Close-up of the circularity

Figure 6.26: Circularity for test case 2 (all groups).

Group	1	2	3
ϕ_{min}	0.5869	0.4647	0.5144
$t\|_{\phi=\phi_{min}}$	2.4004	3.0000	3.0000
$V_{c,max1}$	0.2524	0.2514	0.2502
$t\|_{V_c=V_{c,max1}}$	0.7332	0.7281	0.7317
$V_{c,max2}$	0.2434	0.2440	0.2393
$t\|_{V_c=V_{c,max2}}$	2.0705	1.9844	2.0600
$y_c(t=3)$	1.1380	1.1249	1.1376

Table 6.17: Minimum circularity and maximum rise velocities, with corresponding incidence times, and the final position of the center of mass for test case 2 (all groups).

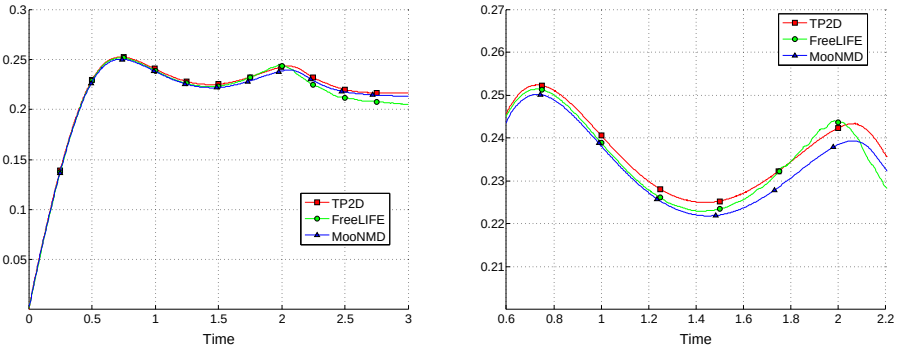
The vertical movement of the center of mass, shown in Figures 6.27(a) and 6.27(b), is predicted very similarly for all groups. Surprisingly the break up does not in this case influence the overall averaged quantities to a significant degree. The estimated final position of the center of mass is 1.37±0.01 (Table 6.17). However, this value is quite meaningless since the final shape the bubble most likely will assume is unknown.



(a) Center of mass (b) Close-up of the center of mass

Figure 6.27: Center of mass for test case 2 (all groups).

Lastly the time evolution of the mean rise velocity is examined. There is also here a quite good agreement between the different codes. The first maximum is predicted to have a magnitude of 0.25±0.01 and occur around t=0.73±0.02 (Table 6.17). The prediction of this maximum should be quite trustworthy since break up has not yet occurred and the curves are quite close to one another (Figure 6.28(a)). The second maximum is more ambiguous but should most likely have a somewhat smaller magnitude and occur around t=2.0 (Figure 6.28(b)).



(a) Rise velocity (b) Close-up of the rise velocity

Figure 6.28: Rise velocity for test case 2 (all groups).

6.5 Summary of the benchmarks

Benchmark studies are valuable tools for the development of efficient numerical methods. A well defined benchmark does not only assist with basic validation of new methods, algorithms, and software components but can also help to answer more fundamental questions, such as *"How much numerical effort is required to attain a certain accuracy?"*, which would allow for rigorous comparison of different methodologies and approaches.

Two benchmark test cases have been introduced and studied by conducting extensive preliminary computations. Both test cases concern the evolution of a single bubble rising in a liquid column while undergoing topological change. For the first test case the shape deformation is quite moderate while the second bubble deforms significantly and even breaks up after some time. In addition, a number of benchmark quantities have been defined which allow for easier evaluation and comparison of the computed results since they can be used for strict validation in a "picture norm" free form. They include the circularity and the center of mass, which both are topological measures, and also the mean rise velocity of the bubble. In future benchmarks it would be interesting to additionally track more complex quantities, such as force measures which involve derivatives of the dependent variables and the discontinuous pressure.

The preliminary studies showed that it was possible to obtain very close agreement between the codes for the first test case, a bubble undergoing moderate shape deformation, and thus establish reference target ranges for the benchmark quantities. The second test case proved far more challenging. Although the obtained benchmark quantities were in the same ranges they did not agree on the point of break up or even what the bubble should look like afterwards, rendering these results rather inconclusive. To establish reference benchmark solutions including break up and separation will clearly require much more intensive effort by the research community.

Chapter 7

State of the art

In this chapter the state of the art of mathematical software tools for simulating flows with immiscible fluids will be examined. Two commercial codes, the very general and flexible PDE solving package Comsol Multiphysics and the truly dedicated CFD solver Ansys Fluent, will be compared with the academic TP2D code which is developed according to the principles presented in this thesis. The chapter will conclude with a discussion on techniques and methodologies which have a strong potential to improve both accuracy and efficiency for multiphase flow simulations.

7.1 Commercial software tools

Commercial software tools are widely used by engineers in the industry to simulate various physical processes. Except for monetary cost, they offer many benefits over academic tools; commercial codes are reasonably easy to use, are documented extensively, have good user support, and usually include tried and tested algorithms which produce good results with "engineering precision". However, what is usually not known is how accurate these codes really are on an absolute level, and what performance can be expected for a given problem. This will, within the context of two-phase flows, be examined in the following by simulating one of the previously described benchmarks with two different commercial codes.

7.1.1 Comsol Multiphysics

The Comsol Multiphysics software suite (previously marketed under the name Femlab) is a finite element package for solving coupled systems of partial differential equations. Although the software is very user friendly, has a nice graphical user interface (GUI), and allows for almost arbitrary PDE problems to be postulated, the fully coupled approach and heavy dependence on direct solvers limits its practical use to rather small problem sizes. Despite this, Comsol Multiphysics was applied to the first benchmark test case (described in Chapter 6) in order to get a feeling for what a general commercial simulation tool, not optimized for CFD, can accomplish.

The following simulations were performed with version 3.3a of Comsol Multiphysics and the conservative level set application mode included in the corresponding Chemical Engineering Module [73, 74]. A purely Cartesian tensor product grid was employed in the calculations with continuous biquadratic and discontinuous linear finite element basis functions, the Q_2P_1 Stokes elements, discretizing the velocity and pressure. The level set field was correspondingly discretized with Q_2 basis functions.

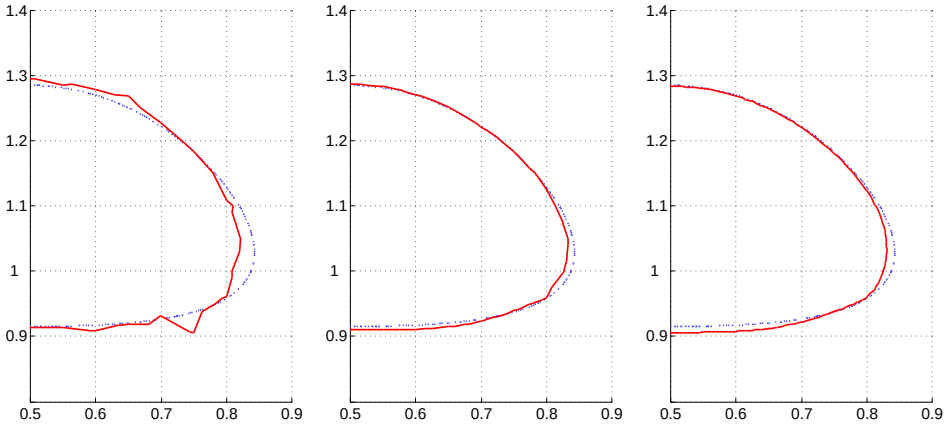
All the following computations (also including those of Fluent and TP2D for comparison purposes) were performed on a computer with a 2.0 GHz Intel Core2Duo processor for which simulation statistics are given in Table 7.1. The first column $1/h$ shows the number of cells in the x-direction corresponding to the level of grid refinement. The number of elements is denoted by NEL, the total number of degrees of freedom by NDOF, and the number of time steps by NTS. The computational effort required can be seen from the peak memory consumption in Megabytes (MEM) and the required time to complete each simulation (CPU).

$1/h$	NEL	NDOF	NTS	MEM	CPU
20	800	13163	277	175	101
40	3200	51923	115	446	483
60	7600	116283	120	890	986
80	12800	206243	120	1568	2361

Table 7.1: Simulation statistics and timings for Comsol Multiphysics.

Although a comprehensive selection of iterative linear solvers is included in Comsol Multiphysics, the default and most robust choice is to use a direct solver, in this case UMFPACK [21]. The magnitude of the peak memory consumption, although scaling linearly with the number of degrees of freedom, was very high due to the fully coupled approach. It was in fact impossible to obtain a solution for anything larger than a 80×160 grid, even when switching to the iterative solvers, which either failed to converge or still tried to allocate too much memory. The time stepping scheme on the other hand worked very well, only requiring about 120 time steps to complete each simulation for all but the coarsest grid. The underlying algorithm employed the variable order DAE solver DASPK (where up to fifth order accuracy was allowed) [11].

The bubble shapes at the final time (t=3) can be seen in Figure 7.1. The results computed on the finer grids look quite good and believable in the picture norm. The shape for the coarsest grid 20×40 (Figure 7.1(a)) exhibits a very jagged contour, but is otherwise reasonably well aligned with the reference shape. Refining the grid improved the shape, but seemed to result in slightly more rounded contours than that of the reference bubble.



(a) 20×40 (b) 40×80 (c) 80×160

Figure 7.1: Bubble shapes computed with Comsol Multiphysics on different grids (solid red), and a reference solution (dashed blue).

The use of the defined benchmark quantities makes it easier to spot convergence trends, and the computed circularity, a measure of shape deformation, is thus compared against an established reference curve (Figure 7.2). The results for the two coarsest grids, 20×80 and 40×80, shows very oscillatory behaviors for which the means deviate significantly from the reference curve. The curves corresponding to the two finer grids behave better but do not converge towards the reference solution after t=2.5. The computed circularity increases towards the end of the simulation instead of flattening out indicating a less elongated shape which is consistent with Figures 7.1(b) and 7.1(c).

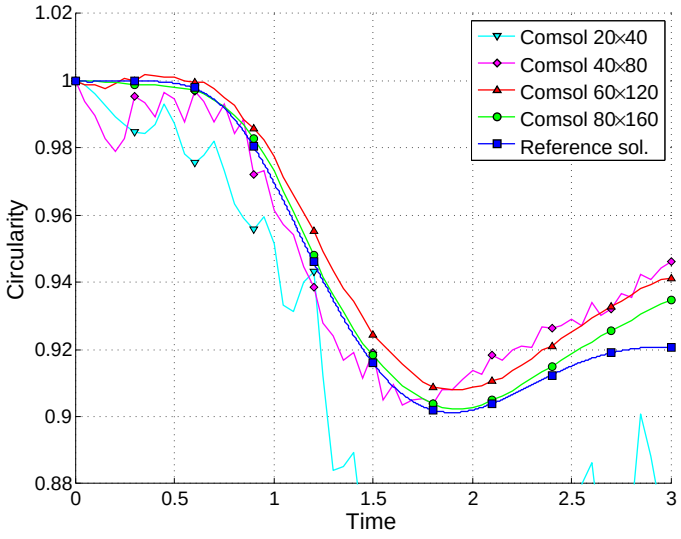


Figure 7.2: Computed circularity for Comsol Multiphysics.

Table 7.2 shows the minimum circularity ϕ_{min} with corresponding incidence times $t|_{\phi=\phi_{min}}$ and also the time averaged relative error of the circularity $|e_\phi|$. Except for the very coarsest grid, the minimum circularity is quite close to the reference value, with a relative error of $1.0{\cdot}10^{-3}$ on the finest grid. The corresponding incidence times fluctuate somewhat and one would ideally like to have more data for finer grid levels to really be able to establish a convergence trend. The time averaged relative error is three times larger than that of the minimum on the finest grid, due to the divergence towards the end of the simulation.

$1/h$	ϕ_{min}	$t\|_{\phi=\phi_{min}}$	$\|e_\phi\|$
20	0.8402	2.25	$3.6{\cdot}10^{-2}$
40	0.9034	1.65	$1.0{\cdot}10^{-2}$
60	0.9081	1.90	$8.0{\cdot}10^{-3}$
80	0.9022	1.95	$2.9{\cdot}10^{-3}$
$Ref.$	0.9012	1.90	

Table 7.2: Minimum circularity with corresponding incidence time, and time averaged error for Comsol Multiphysics. (The line $Ref.$ shows the reference values.)

To summarize, these tests have shown that although the Comsol Multiphysics package can simulate two-phase flows some real difficulties do exist. Firstly, the results do not seem to converge towards the correct solution for longer time periods and, secondly, the general solver approach consumed far too much memory to be able to run simulations with even moderately dense grids.

7.1.2 Ansys Fluent

The CFD package Fluent has frequently been marketed as "the world leader in Computational Fluid Dynamics" [32] and "a state-of-the-art computer program for modeling fluid flow" [31], and thus has a lot to live up to. Fluent includes a comprehensive set of models to treat various flow related phenomena such as heat transfer, turbulence, combustion and chemical reactions, population balances, and also multiphase flow. Flows with immiscible fluids are treated with the Eulerian volume of fluid (VOF) methodology which employs the use of a volume fraction function indicating the relative amounts of the fluids present in each cell.

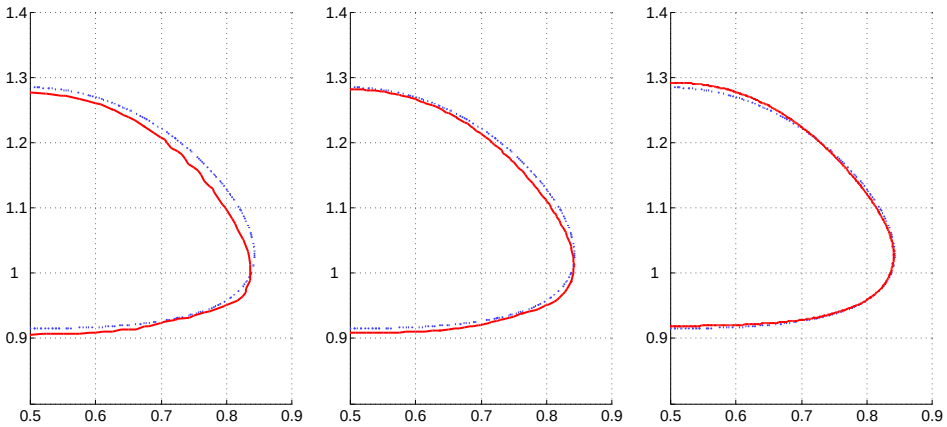
Fluent employs the traditional finite volume discretization in space with unknowns located at the cell centers. In the time domain there are a number of discretization schemes to choose from, of which the recommended implicit Fractional step operator spitting scheme has been used in the following tests. This scheme, which is a form of projection method, effectively separates the solution of the pressure from the velocity calculations, thus saving computational effort. To solve the arising linear equation systems Fluent employs an algebraic multigrid approach (AMG). In the following, identical benchmark tests were performed with version 6.3 of Fluent as previously done with Comsol Multiphysics.

The simulation statistics and timings for Fluent can be seen in Table 7.3. Compared to Comsol Multiphysics, Fluent allocated significantly less memory and allowed the use of finer grids. However, it should be pointed out that since Fluent uses cell centered degrees of freedom the total number of unknowns is four times fewer than used by the Comsol software for a grid of the same size (Comsol Multiphysics also allows for higher accuracy with the employed Q_2 finite element discretization). The time steps were selected so that the capillary time step restriction was respected. Although the calculations for a given grid consumed less CPU time than for Comsol Multiphysics, it does not say anything about the level of accuracy achieved.

$1/h$	NEL	NDOF	NTS	MEM	CPU
40	3200	12800	150	96	45
80	12800	51200	480	111	254
160	51200	204800	1200	210	2106
320	204800	819200	3000	439	21091

Table 7.3: Simulation statistics and timings for Fluent.

Figure 7.3 shows the computed bubble shapes at the final time (t=3). The computation on the coarsest grid produced a result which deviated significantly from the reference bubble and one can see that there is a lot of room for improvement (Figure 7.3(a)). Refining the grids allowed the simulations to converge towards the reference shape, which is evident from Figures 7.3(b) and 7.3(c). It is in fact quite hard to see any significant differences between the reference solution and the shape computed on the finest grid (320×640).



(a) 40×80 (b) 80×160 (c) 320×640

Figure 7.3: Bubble shapes computed with Fluent on different grids (solid red), and a reference solution (dashed blue).

Since the final shape was accurately computed one might expect that the preceding temporal evolution also is correct. However, if one looks at the curves for the circularity (Figure 7.4) one can see that this is not the case. Although mesh independent solutions are obtained from the two finest grids, they do not converge towards the reference solution. It is apparently a period around the maximum deformation, between t=1.2 and t=2.5 (corresponding to the point of minimum circularity), that causes the most trouble for Fluent.

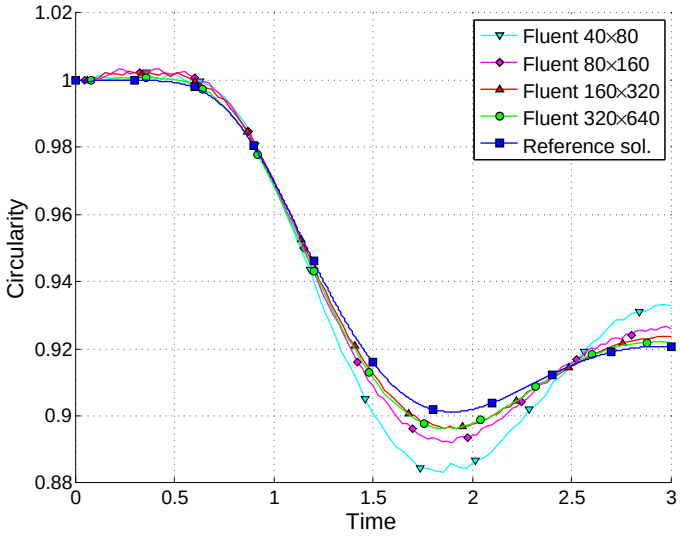


Figure 7.4: Computed circularity for Fluent.

The maximum errors and also the time averaged errors of the circularity are quite large as can be seen from Table 7.4. The values from the two finest grids show that a mesh independent solution has indeed been obtained. However, this solution does not coincide with the reference solution towards which convergence has completely stopped. The minimum circularity is predicted to occur slightly too late with a smaller value than expected. Comparing these errors with those produced by Comsol Multiphysics (Table 7.2) one can see that both codes achieve quite similar levels of accuracy. Fluent has a slight advantage in the averaged norm while the Comsol software produces better values for the minimum circularity.

$1/h$	ϕ_{min}	$t\|_{\phi=\phi_{min}}$	$\|e_\phi\|$
40	0.8834	1.86	$8.2{\cdot}10^{-3}$
80	0.8922	1.90	$4.3{\cdot}10^{-3}$
160	0.8962	1.92	$2.4{\cdot}10^{-3}$
320	0.8963	1.92	$2.3{\cdot}10^{-3}$
$Ref.$	0.9012	1.90	

Table 7.4: Minimum circularity with corresponding incidence time, and time averaged error for Fluent. (The line $Ref.$ shows the reference values.)

To conclude the study of commercial simulation tools recall the curves for the circularity (Figures 7.2 and 7.4). It unfortunately seems that if one desires a good final solution one should choose Fluent, and if one prefers a better intermediate solution one should choose Comsol Multiphysics. Is it not possible to have both? To try to answer this let's proceed by examining a code belonging to a different class of software tools.

7.2 Academic software tools

Academic software often utilizes the newest and most experimental algorithms in contrast to commercial tools which mostly include tried and tested routines. It is interesting to see if there exist any performance differences between a newly developed academic tool and what the commercial enterprises offer. The same tests used to evaluate the commercial simulation tools are therefore also applied to an academic dedicated two-phase flow code.

7.2.1 TP2D

The TP2D code is a Fortran 77 realization of the concepts described in Chapters 2–5. An overview of the solver structure with a break down of the relative computational costs of the involved components can be found in Appendix A. The employed code was identical to the one used to produce the benchmark results in Chapter 6 but in this case tuned to run as fast as possible without sacrificing too much accuracy.

The resulting bubbles shapes and curves for the benchmark quantities were next to identical to the ones given in Section 6.3.1 in Chapter 6 and will thus not be repeated here. However, the simulation statistics and timings differed and are shown in Table 7.5. The most notable difference, compared to the benchmark runs (Table 6.3), is that the number of time steps could be reduced significantly, leading to a corresponding decrease in computational effort. The required CPU time was also smaller than for the commercial tools with respect to both grid size and the number of degrees of freedom (compare Table 7.5 with Tables 7.1 and 7.3).

$1/h$	NEL	NDOF	NTS	MEM	CPU
40	3200	19561	150	15	15
80	12800	77521	450	55	185
160	51200	308641	1000	212	1674

Table 7.5: Simulation statistics and timings for TP2D.

The resulting errors together with reference values are shown in Table 7.6. From these error levels it can be seen that TP2D produces very accurate results, and most notably is that even for the very coarsest grid the error was significantly smaller than anything that the commercial tools could achieve. This can be seen more clearly from Figure 7.5 which plots the averaged error in the circularity against the CPU time for all codes.

$1/h$	ϕ'_{min}	$t\|_{\phi=\phi'_{min}}$	$\|e_{\phi'}\|$
40	0.9002	1.88	$7.2{\cdot}10^{-4}$
80	0.9007	1.88	$2.8{\cdot}10^{-4}$
160	0.9010	1.91	$1.9{\cdot}10^{-4}$
$Ref.$	0.9012	1.90	

Table 7.6: Minimum circularity with corresponding incidence time, and time averaged error for TP2D. (The line $Ref.$ shows the reference values.)

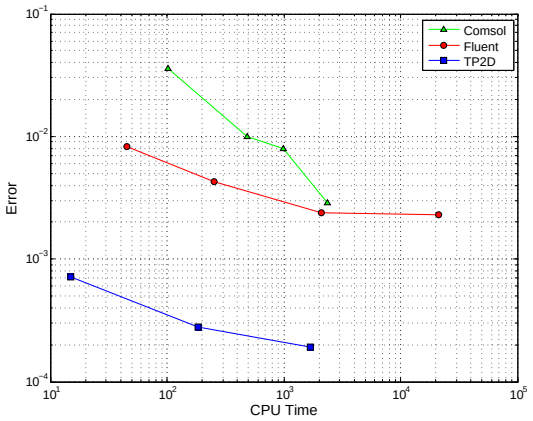


Figure 7.5: Averaged error in the circularity vs. CPU time.

To summarize, these small comparisons have shown that a properly designed academic tool can effectively compete with and also beat commercial tools with respect to both computational time and accuracy. Although the TP2D code did not run magnitudes faster than either Fluent or Comsol Multiphysics this is something that most certainly can be improved upon in the future. Techniques for achieving this will be explored in the following section.

In order to show that the TP2D code has been designed to be able to treat a very general class of two-phase simulations, not just a single rising bubble, three additional examples are shown below. First is a simulation with nine randomly placed bubbles which rise and merge with a free surface (Figure 7.6). The simulation is clearly quite dynamic involving a lot of merging of interfaces. Figure 7.7 shows a corresponding simulation with 25 symmetrically placed bubbles which as they rise arrange themselves in a somewhat unordered pattern. The last example shows how the finite element approach can be applied to more complicated geometries, in this case a liquid film flows around a half circle and forms a long filament which eventually breaks up into drops (Figure 7.8).

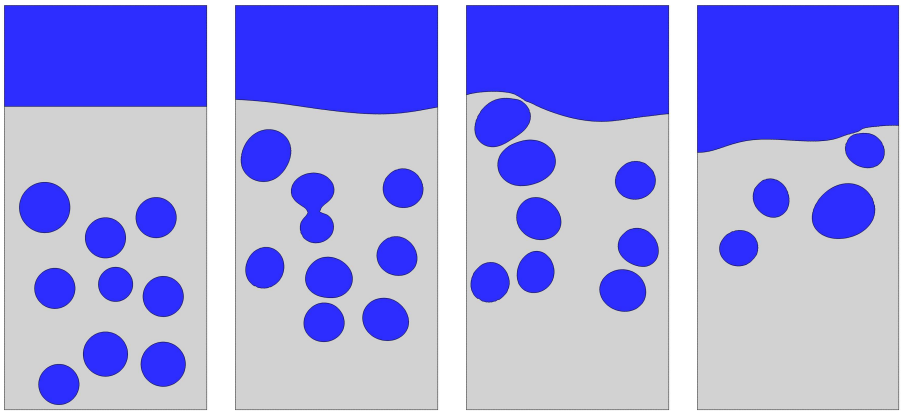


Figure 7.6: Simulation of nine randomly sized and placed rising bubbles merging with a free surface.

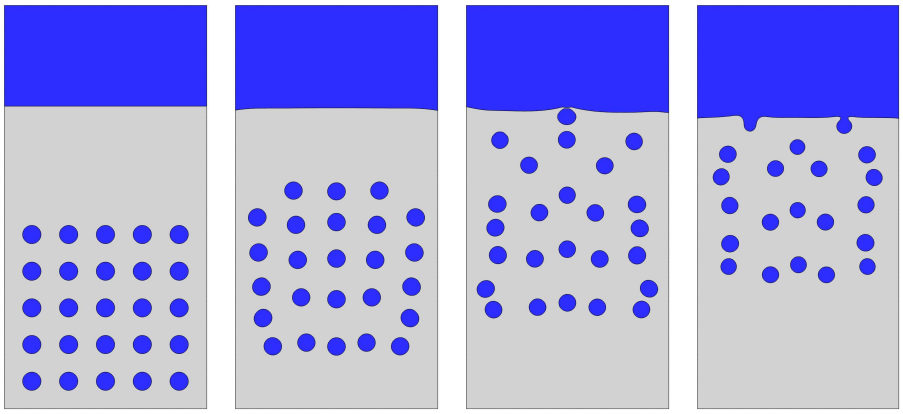


Figure 7.7: Simulation of 25 equally sized and symmetrically placed rising bubbles merging with a free surface.

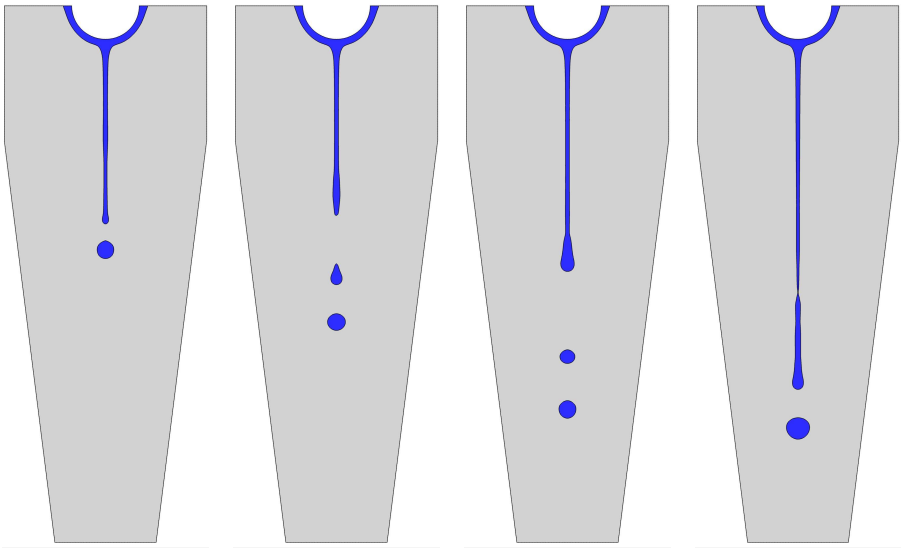


Figure 7.8: Simulation of drop formation from a liquid stream.

7.3 Future prospects

This section presents a discussion on various methods and algorithms which have a strong potential to bring future improvements, in both accuracy and efficiency, to the presented methodology. Many of the methods described in the following have already been successfully implemented, but not thoroughly tested, in the TP2D code. The discussions will therefore only include a few preliminary computational examples.

7.3.1 Grid adaption

Local grid adaption techniques can be particularly useful for Eulerian simulations with interfaces. It is reasonable to assume that a large part of the errors in the computations arises from regions where the interfaces and thus also discontinuities exist. The extent to which the interfaces can be resolved is directly proportional to the local grid size and hence the idea of locally adapting the grids and thereby decreasing the corresponding error.

There are two main approaches to applying grid adaption of which the first is to locally refine cells where needed, so called 'h-adaption'. This approach will invariably generate an increased number of degrees of freedom, and also hanging nodes when working with quadrilateral and hexahedral cells. Another drawback is that the underlying grid topology and matrix structure will be changed, potentially leading to performance penalties. The second approach, called grid deformation or 'r-adaption', consists of redistributing the existing grid points to increase accuracy while at the same time preserving the number of nodes and cells. The following discussion will be limited to this methodology since it does not possess the drawbacks of 'h-adaption', is theoretically easy to implement, and does not require significant changes to an existing code.

Grid deformation

The grid deformation method essentially involves the construction of a transformation ϕ, from the computational space ξ to the physical space $x = \phi(\xi)$ [40]. There are two basic types of approaches to accomplish this. First, there are locally based approaches where x is computed via minimization of a variational form. The minimization procedure often requires the solution of nonlinear systems which can be costly. The second group of grid deformation approaches are velocity based which means that a mesh velocity field is computed which transports the grid nodes to their new locations. There are several advantages to the latter method of which the main ones are the following:

- Only the solution of linear Poisson problems on fixed meshes are required.
- Monitor functions can be constructed quite freely; either directly from error distributions or by choosing other error indicators.
- Mesh tangling can be prevented.
- The mesh topology is never changed through the deformation procedure.

Given the area distribution of the undeformed mesh $g(x)$, and a monitor function/size distribution $f(x)$ for the target grid, then the transformation ϕ can be computed with the velocity based method from the following four steps:

1. Compute the two scale factors c_f and c_g for the given monitor function $f(x) > 0$ and the area distribution $g(x)$ so that

$$c_f \int_\Omega \frac{1}{f(x)}\, dx = c_g \int_\Omega \frac{1}{g(x)}\, dx = |\Omega|,$$

where $\Omega \subset \mathbb{R}^d$ is the computational domain. Let $\tilde{f}$ and $\tilde{g}$ denote the reciprocals of the scaled functions f and g, that is,

$$\tilde{f}(x) = \frac{c_f}{f(x)}, \qquad \tilde{g}(x) = \frac{c_g}{g(x)}.$$

2. Compute a grid-velocity vector field $v : \Omega \to \mathbb{R}^d$ by satisfying the following linear Poisson equation

$$-\mathrm{div}(v(x)) = \tilde{f}(x) - \tilde{g}(x), \;\; x \in \Omega, \quad \text{and} \quad v(x) \cdot \mathbf{n} = 0, \;\; x \in \partial\Omega,$$

where $\mathbf{n}$ is the outer normal vector of the domain boundary $\partial\Omega$, which may consist of several boundary components.

3. For each grid point x, solve the following ODE system

$$\frac{\partial \varphi(x,t)}{\partial t} = \eta(\varphi(x,t),t), \quad 0 \le t \le 1, \quad \varphi(x,0) = x,$$

with

$$\eta(y,s) := \frac{v(y)}{s\tilde{f}(y) + (1-s)\tilde{g}(y)}, \quad y \in \Omega, \;\; s \in [0,1].$$

4. Evaluate the following transformation to get the new updated grid points

$$\phi(x) := \varphi(x,1).$$

For more detailed discussions regarding various implementation issues with this grid deformation method the reader is referred to Grajewski et al. [40]. In addition, further improvements can potentially be achieved by aligning the cell edges with the internal interfaces [48].

Monitor function

The monitor function f should describe the absolute mesh size distribution of the target grid. Typically f is chosen as

$$f(x) = min\,\{1, max\,\{expr, \epsilon\}\}$$

so that f is bounded between $[\epsilon, 1]$ with *expr* determining the grid size distribution in the transition area.

The proper choice of monitor function expression *expr* is an essential part of maximizing the effect of grid deformation, since the movement of grid points should result in a reduction of error. It is reasonable to choose *expr* from information gathered from error indicators, defect vectors, and solution gradients.

For problems containing moving interfaces, such as two-phase flows, there is another option, and that is to use the assumption that errors will be introduced mainly from the interface regions. This assumption is particularly valid for Eulerian interface tracking methods where the interfaces are regularized and smoothed out over several cells. The monitor function expression can thus be constructed by taking the shortest distance to the interface, that is a distance function, which thus concentrates the grid points near the interface. Another option is to use information about the curvature of the interface κ since regions where curvature is high naturally need higher resolution. A linear combination can possibly be the best option, that is

$$f = f\left(\epsilon, |\phi(x)|, \kappa(x)\right).$$

This approach of constructing the monitor function makes the level set method ideally suited to use as an interface tracking algorithm, since both the distance function and curvature are easy to recover and are also defined globally.

Example: Static bubble

To test the influence of grid deformation on a simple fluid flow problem the static bubble test case used in the Chapter 5 is revisited. This test models a perfectly stationary two dimensional bubble with radius $r = 0.25$ in a unit square which, according to the Laplace-Young law, should have a pressure inside equal to $p_{in} = p_{out} + \sigma/r$. The velocity field should ideally be zero everywhere since the bubble is stationary, but due to the numerical discretization so called spurious velocity currents will appear in the vicinity of the interface.

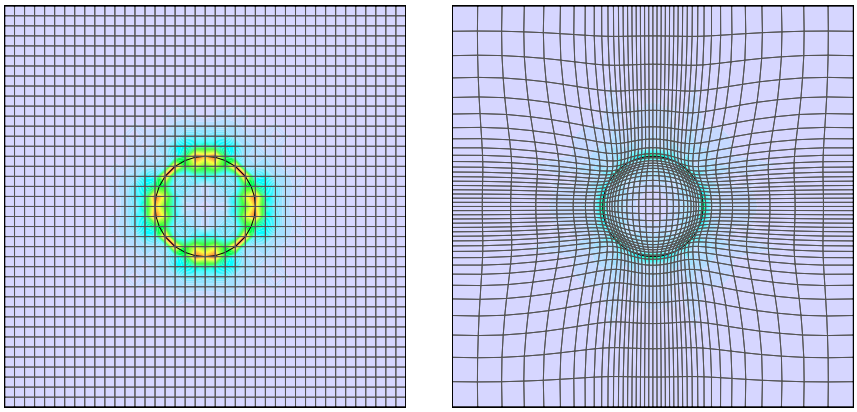


Figure 7.9: Magnitude of spurious velocities for a static bubble with grid deformation (right) and without (left).

Figure 7.9 shows the resulting magnitude of the velocity field with and without grid deformation (the same scaling is used in both figures). Both the spatial extent and magnitude of the spurious velocities decrease as the cells are concentrated around the interface. In Table 7.7 the velocity and pressure errors in the maximum norm are given. With the adapted grid the velocity error was reduced by a factor of 2–3 and the pressure with factor of 2–5. It is hard to

judge the convergence properties with grid deformation since the cell sizes are not controlled in an absolute way. The most convincing way to illustrate the potential of the this method is to look at the pressure jump in Figure 7.10 which clearly shows how much sharper it becomes with an adapted grid.

Grid level	3	4	5	6
Tensor product grid				
U error	$3.7 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$5.6 \cdot 10^{-3}$
P error	5.583%	1.433%	0.212%	0.037%
Adapted grid				
U error	$2.0 \cdot 10^{-2}$	$7.3 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$1.8 \cdot 10^{-3}$
P error	2.969%	0.261%	0.042%	0.013%

Table 7.7: Velocity and pressure errors with and without grid adaption.

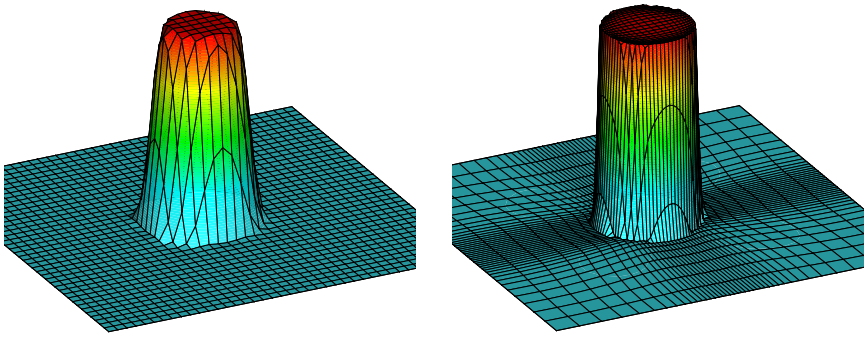


Figure 7.10: Pressure field for a static bubble with (right) and without (left) grid deformation.

7.3.2 ALE formulation

The grid deformation methodology, although having potential to significantly increase accuracy while keeping the computational costs low, introduces an additional difficulty for time dependent problems. Since the grid nodes are moving the computed solution values will not match up with the new node positions. One approach to account for this discrepancy is to use interpolation or some form of projection. This will in most cases introduce various degrees of interpolation errors which with time can accumulate and become quite significant.

A more common approach is what is referred to as the Arbitrary Lagrangian Eulerian method, or ALE for short. The essence of the ALE method is to reformulate the governing equations so that they account for the grid movement in an independent way. The easiest and most convenient way to do this is to

simply subtract the grid velocity $\mathbf{w}$ from the convective velocities, which for a typical discretized problem will look like

$$\begin{aligned}\left[M^{n+1} + \Delta t A^{n+1}(\mathbf{w}^{n+1})\right] u^{n+1} &= b^{n+1}, \\ b^{n+1} &= M^{n+1}u^n + \Delta t f^{n+1}.\end{aligned} \tag{7.1}$$

Here M^l is the mass matrix corresponding to the domain Ω_l, and $A^l(\mathbf{w})$ represents a system matrix. For a typical convection and diffusion problem in non-conservative formulation, discretized with the finite element method, $A^l(\mathbf{w})$ takes the form

$$A^l(\mathbf{w}) = \int_{\Omega^l} \nu \nabla v_1 \cdot \nabla v_2 + ((\mathbf{u} - \mathbf{w}) \cdot \nabla)\, v_1 v_2 \, d\Omega$$

where ν represents the diffusion coefficient, $\mathbf{u}$ the convective velocities, and v_1 and v_2 the finite element trial and test function spaces, respectively.

The ALE scheme (7.1) is simple to implement but only the first order equivalent of the backward Euler time stepping method [33, 70]. More accurate schemes requires a more elaborate construction, as for example the following conservative 2nd order scheme [8]

$$\begin{aligned}\left[M^{n+1} + 0.5\Delta t \left(A^{n+1}(\mathbf{w}^{n+1}) - M_w^{n+1}(\mathbf{w}^{n+1})\right)\right] u^{n+1} &= b^{n+1}, \\ b^{n+1} = \left[M^n - 0.5\Delta t \left(A^n(\mathbf{w}^{n+1}) - M_w^n(\mathbf{w}^{n+1})\right)\right] u^n & \\ + \ 0.5\Delta t \left(f^{n+1} + f^n\right). &\end{aligned} \tag{7.2}$$

Here $M_w^l(\mathbf{w})$ is a mass matrix weighted with the divergence of the grid velocity, that is

$$M_w^l(\mathbf{w}) = \int_{\Omega^t} (\nabla \cdot \mathbf{w}) v_1 v_2, d\Omega.$$

Example: Interface tracking

The combination of the grid deformation and ALE methods are here applied to the standing vortex test case by Rider and Kothe [86]. This interface tracking test consists of convecting a circle with radius 0.15 centered at (0.5,0.5) in a unit square. The velocity field used to deform the circle is given by

$$u = -2\sin^2(\pi x)\sin(\pi y)\cos(\pi y), \quad v = 2\sin(\pi x)\sin^2(\pi y)\cos(\pi x)$$

which deforms the interface in a circular pattern forming a thinning spiral. After the interface reaches the point of maximum deformation at $t = t_{end}/2$ the flow is reversed to attempt to exactly recover the initial state. This time-reversal is accomplished by multiplying the velocity field by $\cos(\pi t/t_{end})$. The initial and intermediate ($t = 2$) reference solutions for the following simulations with $t_{end} = 4$ can be seen in Figure 7.11.

Figure 7.12 shows the computed interface shapes for a fixed tensor product grid, and also the results for the same grid dynamically adapted with subsequent application of ALE schemes (7.1) and (7.2). The results without grid

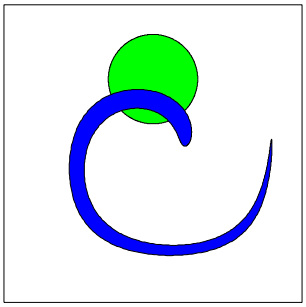


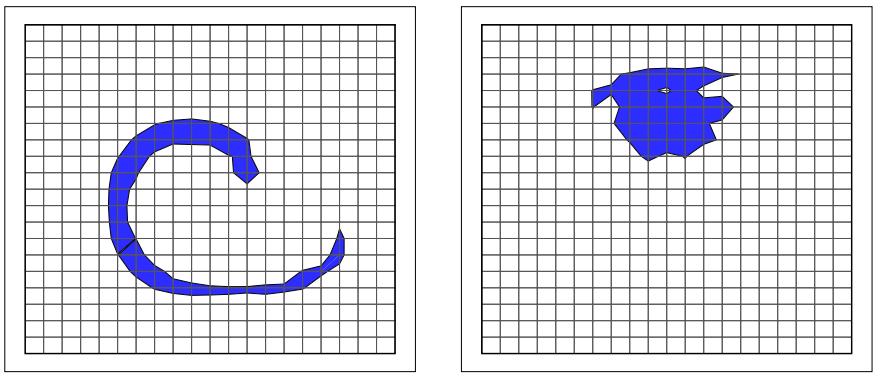
Figure 7.11: Intermediate (blue) and final (green) reference solutions for convection of a circle with a standing vortex.

adaption do not look particularly promising. The intermediate solution is not smooth and seems to have lost a bit of mass in the tail section (Figure 7.12(a)). The final solution shown in Figure 7.12(b) is also very perturbed and does not resemble a circle at all. Both calculations with grid adaption and ALE look far more promising. From comparing the intermediate results (Figures 7.12(c) and 7.12(e)) it is clear that the 2nd order scheme preserves the tail much better. As for the final shapes (Figures 7.12(d) and 7.12(f)) the 1st order scheme generates a smoother result while the shape from the 2nd order scheme is more jagged. One should add that calculations on finer grids produced much better results for the 2nd order scheme, a near perfect circle at the final time, while for the 1st order method this was not the case.

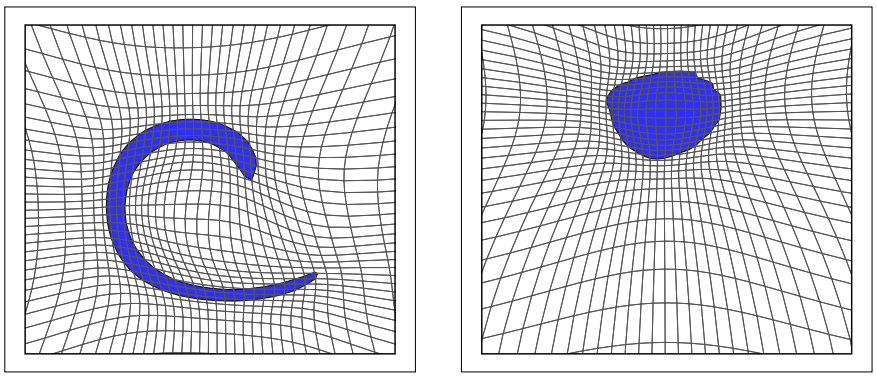
The results are quantitatively summarized in Table 7.8. One can see that while both ALE schemes improve on the final circularity (which should be equal to 1 if the initial state has been recovered perfectly), the 1st order ALE scheme actually leads to a mass loss which is greater than that for the non adapted simulations. The 1st order scheme should therefore be used with extreme caution in contrast to the 2nd order scheme which showed improvements for all grid levels.

Scheme	$1/h$	A_{err}	$\not\in(t = t_{end})$
No grid adaption	20	0.01532	0.60584
	40	0.01288	0.71031
	80	0.00390	0.96438
Grid adaption with	20	0.11688	0.96329
1st order ALE scheme	40	0.05428	0.99110
	80	0.01212	0.99829
Grid adaption with	20	0.02879	0.73490
2nd order ALE scheme	40	0.00093	0.98078
	80	0.00043	0.99881

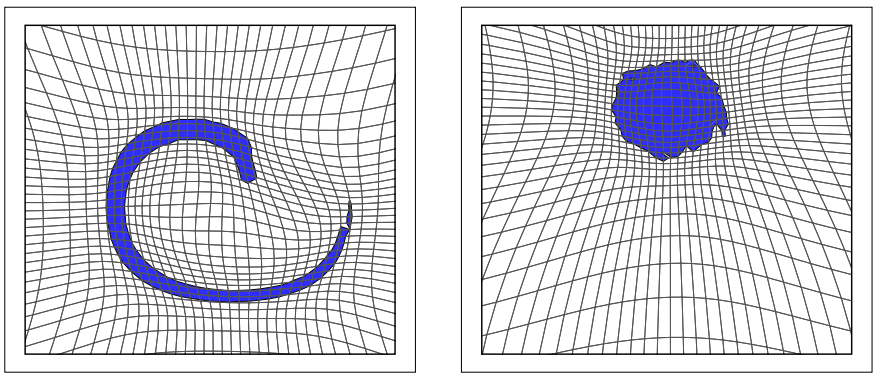
Table 7.8: Errors in mass/area (A_{err}) and the computed circularity at the final time ($\not\in(t = t_{end})$) for the standing vortex interface tracking test case.



(a) Interm. shape (without adaption) (b) Final shape (without adaption)



(c) Interm. shape (1st order ALE scheme) (d) Final shape (1st order ALE scheme)

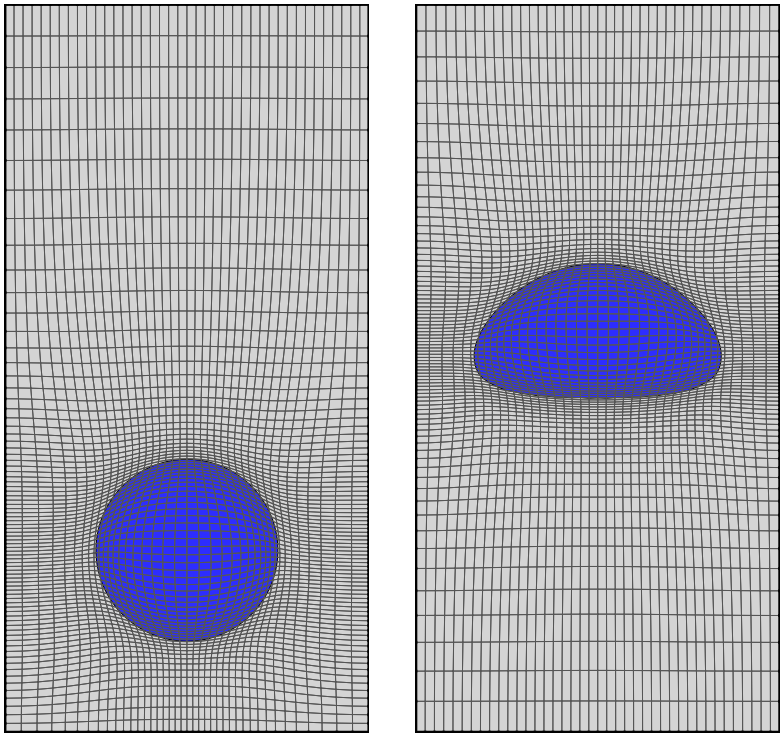


(e) Interm. shape (2nd order ALE scheme) (f) Final shape (2nd order ALE scheme)

Figure 7.12: Intermediate and final shapes for the standing vortex interface tracking test case with and without grid adaption.

Example: Rising bubble

The grid deformation approach with ALE was also applied to the first rising bubble benchmark problem. In contrast to the example above, this test also includes the flow solver which is strongly coupled with the interface tracking algorithm. The introduction of the ALE corrections are unfortunately not enough to completely account for the grid movement in this case, and one must thus introduce supplemental correction mechanisms. One example of the ensuing additional complexities is the treatment of the density and viscosity fields which now have to be interpolated after the grid has been moved. The deformed meshes and bubble shapes for the initial and final times can be seen in Figure 7.13.



(a) t=0 (b) t=3

Figure 7.13: Grid and bubble shapes for a rising bubble simulation with grid deformation and ALE.

The resulting curves for the circularity can be seen in Figure 7.14. The circularity computed with grid deformation and ALE was improved for times less than t=0.7 in comparison with the results from the non-deformed grid with the same number of cells. However, around the minimum the curve shows a slight increase in error, predicting a somewhat larger circularity. To find some reductions in accuracy is not so surprising considering the low level of error already achieved with the tensor product grid, and also the amount of additional uncertainties introduced by the grid deformation/ALE approach.

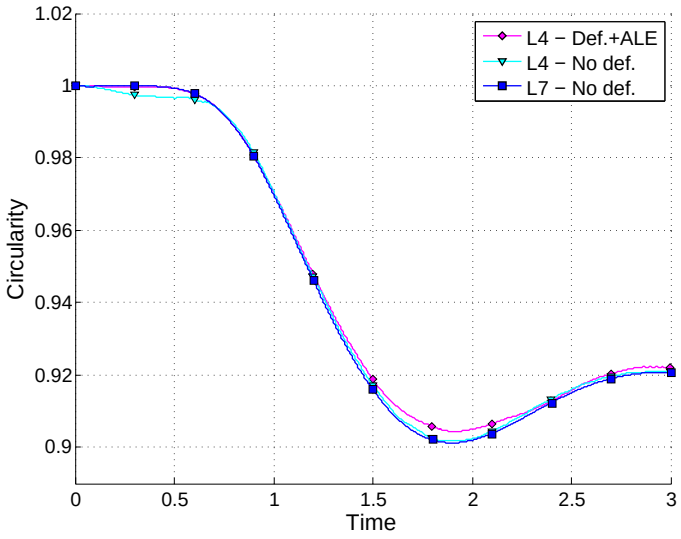


Figure 7.14: Computed circularity with and without grid deformation and ALE, for a rising bubble benchmark problem.

7.3.3 Other potentially beneficial components

Particle level set method

The particle level set method, originally introduced by Enright et al. [26], has the potential to improve the resolution significantly over the standard level set method. The idea is to add signed massless particles in a band around the interfaces which then can be passively transported by the flow. Since the particle convection generally is more accurate, and inexpensive to perform, than for the level set field one should use information from them to correct the level set function. Thus, if a particle ends up on the wrong side of the interface, the level set function is incorrectly represented on the given cell and should be reconstructed with the help of the surrounding particles. Xu implemented this approach in a code based on the presented methodology, and showed how its inclusion can result in very significant improvements in accuracy and mass conservation for pure interface tracking problems [116, 117]. The particle level set method can also theoretically be combined with the grid adaption methodology presented above to yield a method capable of resolving interfaces very accurately.

Pressure separation

Pressure separation is a technique to improve the velocity approximation in finite element solutions of the Navier-Stokes equations [23, 36]. By recognizing that the velocity error generally scales as $Ch^{k+1}(|\mathbf{u}|_{H^{k+1}} + Re|p|_{H^k})$, where C is a constant and k is the finite element approximation order, one sees that if the pressure norm or Re is large then the error will be dominated by $Re|p|_{H^k}$. The idea is now to adjust the pressure so that the corresponding norm will be small, thus one should solve the modified system

$$\begin{aligned} \rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) &= -\nabla(p - p_{sep}) + \nabla\cdot\left(\mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right) + \mathbf{f} - \nabla p_{sep} \\ \nabla\cdot\mathbf{u} &= 0 \end{aligned}$$

instead of the usual Navier-Stokes equations, where p_{sep} is a known function so that $|p - p_{sep}|_{H^k}$ is minimized. Different ways of constructing p_{sep} is given in [36] together with some simple tests proving the validity of the approach. A more comprehensive discussion with more application oriented tests can be found in Turek et al. [105] from where it is clear this technique shows good potential for single phase flows. They also applied pressure separation together with edge stabilization to a static bubble and observed improvements in both the pressure and velocity. More intensive studies must be undertaken to be able to draw conclusions on the applicability of pressure separation to general two-phase flow simulations.

Higher order elements

The methodology described in this thesis is quite general but has been implemented with first order finite elements. This discretization is very computationally efficient but does not use the finite element method to its fullest potential. Higher order elements, such as Q_2P_1 for the flow variables, allows for significantly more accurate solutions, in particular in regions where the solutions are reasonably smooth [75]. It is also desirable to use the same finite element space for the velocities and the level set field, and thereby simplifying interpolation between the two.

The simulations with Comsol Multiphysics in section 7.1.1 above, utilized this suggested approach, namely Q_2P_1 basis functions for the velocity and pressure and Q_2 for the level set field. The corresponding simulations were performed without any artificial stabilization, mass conservation, or reinitialization of the level set field. Taking this into consideration one must say that the results were surprisingly accurate and shows that, despite the performance issues, higher order elements have a great potential.

Assembly

The matrix assembly in finite element simulations is a necessary but very time consuming task, especially in multiphase flow simulations (see also Appendix A). Aside from fully optimizing the assembly routines manually in the code, one should consider what can be done to reduce the amount of required computations by streamlining the assembly algorithm on an algorithmic level. One could for example use the approach by Kirby et al. [57] who applies a heuristic graph algorithm to find redundant and similar computations and thereby reducing the total number of involved operations.

A more natural way, with respect to assembly of mass and diffusion matrices in the case of two-phase flows, is to recognize that only the cells containing and surrounding a discontinuity will need to be updated. One can thus compute the linear matrices without the discontinuities once after which it would be sufficient to correct the subset of matrix entries corresponding to cells containing interface segments [115]. In this way a lot of computational effort can be saved since it is

quite rare to have interfaces in more than 10% of all grid cells. This approach does not apply to the nonlinear convective term in the Navier-Stokes equations which always have to be reassembled (assuming that no extrapolation in time is used). In the absence of a complexity reducing method to treat this case one can always resort to parallelization.

Parallelization

With the advent of dual- and multi-core processors discussions do not in general contain the question "if to parallelize" a code anymore, but more often "how" to do it properly. There are currently two main parallelization approaches available, shared memory with OpenMP and distributed memory with MPI, which both have their respective advantages and disadvantages. OpenMP is limited to the available memory and number of processors/cores in the machine, but is easy to include in virtually any code. One has to be careful when implementing, since writing to shared variables, such as matrices in the assembly, will result in the loss of data when two or more write operations try to access the same location at the same time. MPI on the other hand does not have this problem since memory and work load is shared between a number of computers at the cost of a more complicated implementation. The application in question must eventually dictate which approach is the most suitable.

Chapter 8

Summary and outlook

In this thesis appropriate methods for numerical simulation of flows with immiscible fluids have been discussed. An approach has been presented which couples a flow solver, based on the discrete projection method, with the level set method for interface tracking. An efficient finite element discretization in space was chosen with nonconforming $\hat{Q}_1 Q_0$ Rannacher-Turek elements for the flow variables and conforming Q_1 basis functions for the level set field. In time, the implicit θ-scheme was applied, allowing for both 1st and 2nd order accuracies.

A semi-implicit approach to implementing surface tension effects was additionally presented which in contrast to traditional explicit implementations allows for significantly larger time steps to be taken. In this new method the interfaces are represented implicitly in space through regularized delta functions, which is especially attractive for higher order discretizations in 3D where explicit interface reconstruction can be very difficult. The new methodology is particularly suited for finite element discretizations combined with the level set method due to the existence of global distance, normal, and curvature fields, which are used in the construction of the regularized delta functions. The performance of the new method was tested on three numerical examples, which showed that the capillary time step restriction could be exceeded by up to 80 times in some cases.

The usual approach to validation of multiphase flow codes is to look at the "picture norm". This is rather limiting since it does not allow one to establish an absolute error level, which is needed to be able to show convergence. Two benchmark test cases have thus been defined for strict quantitative evaluation and comparison of two-phase flow codes. A benchmarking initiative has also been started for which three independent research groups have participated by calculating the benchmarks. It was shown that grid independent reference values of the defined quantities could be obtained for the simpler of the two cases, a bubble undergoing moderate deformation, while the other case including break up and separation proved far more difficult.

The usefulness of the benchmarks was shown when comparing two commercial CFD tools (Comsol Multiphysics and Ansys Fluent) with the academic TP2D code, which is based on the presented methodology. It was clear that neither of the commercial tools managed to achieve a really strong convergence towards the reference solution. The developed code was on the other hand faster and more importantly much more accurate. It is notable that it only took 15 CPU seconds for the TP2D code to compute a solution better than anything that the commercial codes could produce.

The presented methodology and code has a strong future potential to be even more accurate, with the inclusion of higher order elements, grid adaption, narrow band and particle level set, and pressure separation methods. At the same time it could be made to run much faster by optimizing the finite element assembly step. Since the developed code was restricted to two dimensions a natural step in its development is to extend it to axisymmetry and full 3D. This also holds true benchmarks which then possibly could be compared with experimental data. However, this step will invariably require the aid of parallelization, since already in 2D it can take up to 1 month to calculate a benchmark reference solution. How to best achieve speedup with parallelization is also something that is left for the future.

Appendix A

Solver structure

In Chapters 2 to 5 different components needed to numerically simulate two-phase flows have been presented and discussed. In the solution process these components need to be arranged and called upon in a suitable and efficient manner, what is called a solver structure. The following appendix will cover the construction of an appropriate solver structure, and various efficiency aspects relating to it.

A.1 Solution procedure

Both the solution of the Navier-Stokes equations and any additional interface tracking algorithm can be treated in a fully coupled way, that is to set up and solve the following large system for all unknowns:

$$\begin{cases} \rho(\phi)\left(\dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) = -\nabla p + \nabla\cdot\left(\mu(\phi)(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right) + \mathbf{f}, \\ \mathbf{f} = \rho(\phi)\mathbf{g} + \sigma\kappa(\phi)\hat{\mathbf{n}}(\phi)\delta(\phi), \\ \nabla\cdot\mathbf{u} = 0, \\ \dfrac{\partial \phi}{\partial t} + (\mathbf{u}\cdot\nabla)\phi = 0. \end{cases} \tag{A.1}$$

This may be convenient and robust, but not efficient from a computational point of view. It is generally always less costly to solve, that is to invert, a series of smaller matrix systems than a single large one. The discrete projection method presented in Chapter 3 allows for this by separating the velocity computations from the pressure.

The interface tracking algorithm, in this case the level set equation, should theoretically also be included implicitly so that the interfaces, which must move with the velocity field, are updated continuously. In practice this is usually not very convenient since the density and viscosity fields, gravity force, and surface tension in (A.1) all depend on the position of the interfaces. The velocity field in the trilinear form (4.4) convecting the interfaces will also have to be handled explicitly.

Considering all this it is quite natural to add the interface tracking algorithm explicitly, updating the positions of the interfaces after the new velocity and pressure fields have been established. This will lead to a solver structure of the following form:

Solver structure

- Initialization
- Time stepping loop:

  For $t^{n+1} \in [0, T]$, and time step $\Delta t = t^{n+1} - t^n$ do

  1. Discrete projection method:

     Given $\Gamma = \Gamma^n$ update the velocity field $\mathbf{u}^{n+1}$ and pressure p^{n+1}.
  2. Interface tracking:

     Given $\mathbf{u} = \mathbf{u}^{n+1}$ update the interface Γ^{n+1}.
  3. Postprocessing

The initialization will include input of parameters, grid generation, and allocation of memory space for arrays, vectors, and matrices. It is advantageous to include the discrete projection method and interface tracking, steps 1 and 2, modularly so that the code easily can be extended if one for example wants to include chemical reactions or heat transfer and solidification effects. Steps 1-3 may be repeated iteratively in each time step until convergence in a suitable norm has been reached. The postprocessing step contains routines for graphical and quantitative output, and time stepping control. Local grid adaption and grid deformation techniques can either be added in this step, or between steps 1 and 2, depending on which variables are the least sensitive to interpolation.

For a typical simulation the flow solver step will consume 80% or more of the total CPU time. The cost of the interface tracking can potentially be made completely negligible assuming that appropriate algorithms are chosen (for example the narrow band level set method). Let's look in more detail which components are required in the different modules and what their corresponding computational overhead is.

A.1.1 Discrete projection method

The solution of the flow variables, even when using the discrete projection method, is the most costly component and hence should be given the most attention. A corresponding module (treating step 1 above) will have the following structure:

Flow solver: Discrete projection method module

1. Assembly of mass and diffusion matrices.
2. Generation of the projection matrix.
3. Assembly of RHS (gravity and surface tension effects).
4. Assembly of convective terms and calculation of initial defect.
5. Defect correction loop for the momentum equations:
   (a) Assemble convective terms and generate iteration matrices.
   (b) Solve the linearized momentum equations.
   (c) Calculate new defect.
   (d) Check for convergence (go to step 5a if necessary).
6. Calculate RHS for the pressure Poisson equation.
7. Solve the pressure Poisson equation.
8. Update pressure and velocity fields.

In this algorithmic structure, the mass and diffusion matrices are assembled once outside the defect loop. This saves some computational effort, in direct proportion to the number of required defect iterations, compared to using a single routine to assemble everything in the defect loop. Boundary conditions must also be appropriately applied to the momentum equations at flow inlets and wall boundaries.

The relative cost of each component in the flow solver was examined by computing a typical rising bubble problem (benchmark test case 1 described in Chapter 6). Table A.1 shows the relative costs for different levels of grid refinement. The boundary condition enforcement (BC), defect calculations (DFKT), and linear combinations of vectors and arrays (LC) were not major contributing parts, together comprising 13-19% of the total computational effort. The most expensive part was instead the finite element matrix assembly (ASM) taking roughly 50% of the CPU time. Of the different assembly steps, the cost of assembling the nonlinear convective contributions were the highest since 2-3 defect iterations were required for convergence to be achieved. The solution of the momentum equations (SLV(U)) was restricted to two grid levels, and employed SOR as coarse grid solver and smoother (with 1 smoothing step). This resulted in a well balanced multigrid scheme where no component was dominating. The solution of the pressure Poisson equation on the other hand required the use of all available grid levels and 4 SOR smoothing steps which clearly is why the smoothing needed about 60% of the solving effort. All together the solvers required about 20-30% of the total computational effort of which about three quarters were due to the solution of the momentum equations.

Level	4	5	6
BC	0.3%	0.2%	0.1%
DFKT	4.5%	6.0%	7.0%
LC	8.7%	11.1%	11.0%
ASM	56.1%	50.3%	47.6%
Mass. matrix	14.7%	12.0%	11.5%
Diff. matrix	28.4%	22.1%	19.7%
Conv. matrix	44.5%	47.2%	47.5%
Linear comb.	12.4%	18.7%	21.2%
SLV(U)	17.8%	23.5%	27.3%
Smoother	26.3%	25.6%	25.6%
Solver	8.3%	7.5%	7.2%
Defect calc.	37.7%	39.4%	40.5%
Prolongation	24.6%	25.4%	25.1%
Restriction	3.0%	2.0%	1.6%
SLV(P)	9.7%	7.1%	5.6%
Smoother	57.1%	60.0%	58.4%
Solver	2.6%	1.0%	0.3%
Defect calc.	14.4%	13.1%	13.1%
Prolongation	15.4%	15.3%	16.8%
Restriction	9.1%	10.4%	11.3%

Table A.1: Distribution of computational effort for the flow module (BC-Boundary conditions, DFKT-Defect calculation, LC-Linear combinations, ASM-Matrix assembly, SLV(U/P)-Solving for U or P).

A.1.2 Interface tracking

An interface tracking module for solving the level set field is in theory quite easy to construct. The governing equation is a standard convection diffusion PDE transporting a relatively smooth scalar property, and thus no special tools need to be developed.

The most costly component will again be the assembly, even though only the convective transport operator needs to be reassembled in each time step. The velocity field in the trilinear transport operator, which in this case comes from another finite element space, should be evaluated directly in the cubature points. This is costly since it will require a higher order quadrature rule (up to fourth order in our case).

Additionally routines for reinitialization, mass conservation, and normal and curvature recovery may be added in a postprocessing step. The interface tracking module with the level set method will in detail look like the following:

Interface tracking: Level set module

1. Assembly of convection matrix.
2. Generation of RHS.
3. Calculation of initial defect.
4. Defect correction loop for the level set equations:
   (a) Assemble contributions for artificial stabilization.
   (b) Solve the level set equation.
   (c) Calculate new defect.
   (d) Check for convergence (go to step 4a if necessary).
5. Apply reinitialization and mass conservation.
6. Recover normal and curvature fields.

Note that the mass matrix is assembled once and then reused (under the assumption that the grid is fixed). The defect correction loop can also be omitted if the applied convective stabilization is linear. This scheme applied to the rising bubble test problem results in the following distribution of the computational costs (see Table A.2).

Level	4		5		6	
LC	1.1%		0.9%		0.5%	
ASM	72.3%		68.0%		63.9%	
Conv. matrix		55.4%		55.9%		49.0%
Art. Stab.		7.0%		5.1%		5.4%
Asm-LC+MV.		37.6%		39.0%		45.7%
SLV	0.9%		0.8%		0.5%	
POST	25.0%		30.0%		35.0%	
Reinit.		42.0%		54.0%		68.7%
Mass cons.		29.7%		20.5%		12.9%
Grad. recov.		14.4%		12.9%		9.8%
Interf. info		15.0%		13.9%		9.6%

Table A.2: Distribution of computational effort for the level set module (LC-Linear combinations, ASM-Matrix assembly, SLV-Solving, POST-Additional post processing).

The linear combinations (LC) and more notably the solving (SLV) steps were completely negligible with respect to computational effort. In the solution step the preconditioning matrix was approximated with the lumped mass matrix, and thus cost virtually nothing to invert. The expensive part was as expected the finite element assembly (ASM) demanding 64%-72% of the total CPU time. The cheapest part of the assembly step was the addition of artificial stabilization, in this case FEM-TVD. The assembly of the convective matrix and linear combinations/matrix vector operations, used in the assembly, equally shared the rest of the time. Of the postprocessing (POST) operations, the fast marching reinitialization required the most effort, clearly costing more as the grid was refined (scaling as $\mathcal{O}(NlogN)$). The rest of the postprocessing time was shared between the artificial mass conservation, normal and curvature recovery, and general interface related routines.

Bibliography

[1] D. Adalsteinsson and J. A. Sethian, *The fast construction of extension velocities in level set methods*, J. Comp. Phys., Volume 148, 2–22, 1999.

[2] S. Akanni, T. Larsson, and C. Bienz, *Numerical modelling of the aerodynamic flow field about a formula one car*, Fluent User Group Meeting, Germany, 2001.

[3] E. Bänsch, *Numerical Methods for the Instationary Navier-Stokes Equations with a Free Capillary Surface*, Habilitation Thesis, Universität Freiburg, 1998.

[4] E. Bänsch, *Finite element discretization of the Navier-Stokes equations with a free capillary surface*, Numer. Math., Volume 88, Issue 2, 203–235, 2001.

[5] E. Bänsch, C. P. Berg, and A. Ohlhoff, *Uniaxial extensional flows in liquid bridges*, Journal of Fluid Mechanics, Volume 521, 353–379, 2004.

[6] C. Bienz, T. Larsson, T. Sato, and B. Ullbrand, *In front of the grid - CFD at Sauber Petronas F1 leading the aerodynamic development*, 1st European Automotive CFD Conference, Bingen, Germany, 2003.

[7] H. Blum, J. Harig, S. Müller, and S. Turek, *FEAT2D Finite element analysis tools*, User Manual, Release 1.3, Heidelberg, 1992.

[8] D. Boffi and L. Gastaldi, *Stability and geometric conservation laws for ALE formulations*, Comput. Methods Appl. Mech. Engrg., Volume 193, 4717–4739, 2004, doi: 10.1016/j.cma.2004.02.020.

[9] J. U. Brackbill, D. B. Kothe, and C. Zemach, *A continuum method for modeling surface tension*, J. Comp. Phys., Volume 100, Issue 2, 335–354, 1982, doi: 10.1016/0021-9991(92)90240-Y.

[10] M. O. Bristeau, R. Glowinski, and J. Periaux, *Numerical methods for the Navier-Stokes equations. Application to the simulation of compressible and incompressible flows*, Comp. Phys., Volume 6, 73–188, 1987.

[11] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold, *Using Krylov methods in the solution of large-scale differential-algebraic systems*, SIAM J. Sci. Comput., Volume 15, 1467–1488, 1994.

[12] E. Burman and P. Hansbo, *Edge stabilization for Galerkin approximations of convection-diffusion-reaction problems*, Computer Methods in Applied Mechanics and Engineering, Volume 193, Issues 15–16, 1437–1453, 2004, doi: 10.1016/j.cma.2003.12.032 .

[13] E. Burman and N. Parolini, *Subgrid edge stabilization for transport equations*, EPFL-IACS report 09.2005, 2005.

[14] E. Burman and N. Parolini, *A new reinitialization procedure for the finite element approximation of the level set equation*, EPFL-IACS report 13.2005, 2005.

[15] D. L. Chopp, *Some Improvements of the fast marching method*, SIAM J. Sci. Comput., Volume 23, Issue 1, 230–244, 2001, doi: 10.1137/S106482750037617X.

[16] L. Chen, S. V. Garimella, J. A. Reizes, and E. Leonardi, *The development of a bubble rising in a viscous fluid*, Journal of Fluid Mechanics, Volume 387, 61–96, 1999.

[17] T. Chen, P. D. Minev, and K. Nandakumar, *A projection scheme for incompressible multiphase flow using adaptive Eulerian grid*, Int. J. Num. Meth. Fluids, Volume 45, Issue 1, 1–19, 2001, doi: 10.1002/fld.591.

[18] A. M. Christon, P. M. Gresho, and S. B. Sutton, *Computational predictability of time-dependent natural convection flows in enclosures (including a benchmark solution)*, Int. J. Num. Meth. Fluids, Volume 40, Issue 8, 953–980, 2002, doi: 10.1002/fld.395.

[19] R. Clift, J. R. Grace, and M. E. Weber, *Bubbles, Drops and Particles*, Academic Press: New York, 1978.

[20] *Millennium Prize Problems*, Available from the Clay Mathematics Institute of Cambridge, Massachusetts (CMI) site: http://www.claymath.org/millennium/.

[21] T. A. Davis, *Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method*, ACM Transactions on Mathematical Software, Volume 30, Issue 2, 196–199, 2004.

[22] D. A. Di Pietro, S. Lo Forte, and N. Parolini, *Mass preserving finite element implementations of the level set method*, Applied Numerical Mathematics, Volume 56, Issue 9, 1179–1195, 2006, doi: 10.1016/j.apnum.2006.03.003.

[23] O. Dorok, *Improved accuracy of a finite element discretization for solving the Boussinesq approximation of the Navier-Stokes equations*, Preprint 32/94, Otto-von-Guericke-Universität Magdeburg, Fakultät für Mathematik, 1994.

[24] G. Dziuk, *An algorithm for evolutionary surfaces*, Numer. Math, Volume 58, Issue 6, 603–611, 1991, doi: 10.1007/BF01385643.

[25] B. Engquist, A.-K. Tornberg, and R. Tsai, *Discretization of Dirac delta functions in level set methods*, J. Comput. Phys., Volume 207, 28–51, 2005.

[26] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell, *A hybrid particle level set method for improved interface capturing*, J. Comput. Phys., Volume 183, 83–116, 2002.

[27] D. Erickson, *Towards numerical prototyping of labs-on-chip: modeling for integrated microfluidic devices*, Microfluidics and Nanofluidics, Volume 1, Issue 4, 301–318, 2005, doi:10.1007/s10404-005-0041-z.

[28] R. Fedkiw and X.-D. Liu, *The Ghost Fluid Method for Viscous Flows*, Innovative Methods for Numerical Solutions of Partial Differential Equations, edited by M. Hafez and J.-J. Chattot, 111–143, World Scientific Publishing, New Jersey, 2002.

[29] X. Feng, Y. He, and C. Liu, *Analysis of finite element approximations of a phase field model for two-phase fluids*, Math. Comp., Volume 76, 539–571, 2007, doi: 10.1090/S0025-5718-06-01915-6.

[30] C. A. J. Fletcher, *The group finite element formulation*, Computer Methods in Applied Mechanics and Engineering, Volume 37, Issue 2, 225–244, 1983, doi: 10.1016/0045-7825(83)90122-6.

[31] *FLUENT 6.3 Getting Started Guide*, Fluent Inc. 2006.

[32] *Fluent announces first international CFD conference dedicated to the oil & gas industry*, Press release, Fluent Europe Ltd., Sheffield, UK, 7th April 2006.

[33] L. Formaggia and F. Nobile, *A stability analysis for the arbitrary Lagrangian Eulerian formulation with finite elements*, East-West J. Numer. Math., Volume 7, Issue 2, 105–131, 1999.

[34] S. Gallot, D. Hulin, and J. Lafontaine, *Riemannian Geometry, 3rd ed.*, Springer-Verlag, 2004, ISBN: 3-540-20493-8.

[35] S. Ganesan, *Finite Element Methods on Moving Meshes for Free Surface and Interface Flows*, PhD Thesis, Otto-von-Guericke-Universität, Fakultät für Mathematik, Magdeburg, 2006, published as book by docupoint Verlag Magdeburg, ISBN: 3-939665-06-1.

[36] S. Ganesan and V. John, *Pressure separation–a technique for improving the velocity error in finite element discretizations of the Navier-Stokes equations*, Applied Mathematics and Computation, Volume 165, 275–290, 2005.

[37] S. Ganesan and L. Tobiska, *Finite element simulation of a droplet impinging a horizontal surface*, Proceedings of Algoritmy, 1–11, 2005.

[38] S. Ganesan, G. Matthies, and L. Tobiska, *On spurious velocities in incompressible flow problems with interfaces*, Computer Methods in Applied Mechanics and Engineering, Volume 196, Issue 7, 1193–1202, 2007, doi: 10.1016/j.cma.2006.08.018.

[39] V. Girault and P. A. Raviart, *Finite Element Methods for Navier-Stokes equations*, Springer-Verlag, 1986.

[40] M. Grajewski, M. Koester, S. Kilian, and S. Turek, *Numerical analysis and practical aspects of a robust and efficient grid deformation method in the finite element context*, Ergebnisberichte des Instituts für Angewandte Mathematik, Nummer 294, FB Mathematik, Universität Dortmund, 2005.

[41] S. Gross, V. Reichelt, and A. Reusken, *A finite element based level set method for two-phase incompressible flows*, IGPM-Report 243, RWTH Aachen, 2004.

[42] W. Hackbusch, *Multi-Grid Methods and Applications*, Springer-Verlag, 1985, ISBN: 0-387-12761-5.

[43] M. Hanke, *Benchmarking FEMLAB 3.0a: Laminar Flows in 2D*, Royal Institute of Technology (Stockholm), Report No. 2004:01, Department of Numerical Analysis and Computer Science, Parallel and Scientific Computing Institute, 2004.

[44] E. Hinton, T. Rock, and O. C. Zienkiewicz, *A note on mass lumping and related processes in the finite element method*, Earthquake engineering and structural dynamics, Volume 4, 245–249, 1976.

[45] J. I. Hochstein and T. L. Williams, *An Implicit Surface Tension Model*, AIAA meeting papers, 599, 1996.

[46] J. Hron and S. Turek, *Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow*, Bungartz, H.-J.; Schäfer, M., Lecture Notes in Computational Science and Engineering, 53, 371–385, Fluid-Structure Interaction - Modelling, Simulation, Optimization, Springer, 2006, ISBN: 3-540-34595-7.

[47] T. J. R. Hughes and A. Brooks, *A multidimensional upwind scheme with no crosswind diffusion*, In T. J. R. Hughes, editor, Finite Element Methods for Convection Dominated Flows, AMD, Volume 34, ASME, 19–35. New York, 1979.

[48] J. M. Hyman, S. Li, P. Knupp, and M. Shashkov, *An algorithm for aligning quadrilateral grid with internal boundaries*, Journal of Computational Physics, Volume 163, 133–149, 2000, doi:10.1006/jcph.2000.6560.

[49] S. Hysing and S. Turek, *The Eikonal equation: Numerical efficiency vs. algorithmic complexity on quadrilateral grids*, Proceedings of Algoritmy, Conference on Scientific Computing, 22–31, 2005, ISBN: 80-227-2192-1.

[50] S. Hysing, *A new implicit surface tension implementation for interfacial flows*, Int. J. Num. Meth. Fluids, Volume 51, Issue 6, 659–672, 2006, doi: 10.1002/fld.1147.

[51] S. Hysing, S. Turek, D. Kuzmin, N. Parolini, E. Burman, S. Ganesan, and L. Tobiska, *Proposal for quantitative benchmark computations of bubble dynamics*, Submitted to Int. J. Num. Meth. Fluids in 2007.

[52] V. John, *Higher order finite element methods and multigrid solvers in a benchmark problem for the 3D Navier-Stokes equations*, Int. J. Num. Meth. Fluids, Volume 40, 775–798, 2002.

[53] V. John, *Reference values for drag and lift of a two-dimensional time dependent flow around a cylinder*, Int. J. Numer. Meth. Fluids, Volume 44, 777–788, 2004.

[54] V. John and G. Matthies, *Higher-order finite element discretizations in a benchmark problem for incompressible flows*, Int. J. Num. Meth. Fluids, Volume 37, Issue 8, 885–903, doi: 10.1002/fld.195.

[55] V. John and G. Matthies, *MooNMD - a program package based on mapped finite element methods*, Computing and Visualization in Science, Volume 4, Issues 2–3, 163–170, 2004, doi: 10.1007/s00791-003-0120-1.

[56] R. Kimmel and J. A. Sethian, *Computing geodesic paths on manifolds*, Proceedings of National Academy of Sciences, Volume 95, Issue 15, 8431–8435, 1998.

[57] R. C. Kirby, M. Knepley, A. Logg, and L. R. Scott, *Optimizing the evaluation of finite element matrices*, SIAM Journal on Scientific Computing, Volume 27, Issue 3, 2005.

[58] D. Kuzmin, *On the design of general-purpose flux limiters for finite element schemes. I. Scalar convection*, Journal of Computational Physics, Volume 219, Issue 2, 513–531, 2006, doi: 10.1016/j.jcp.2006.03.034.

[59] D. Kuzmin and S. Turek, *High-resolution FEM-TVD schemes based on a fully multidimensional flux limiter*, Journal of Computational Physics, Volume 198, Issue 1, 131–158, 2004, doi: 10.1016/j.jcp.2004.01.015.

[60] D. Kuzmin, R. Löhner, and S. Turek, *Flux-Corrected Transport: Principles, Algorithms, and Applications*, Scientific Computation, Springer, 2005, ISBN: 978-3-540-23730-3.

[61] R. Levy and M. Shearer, *Comparison of two dynamic contact line models for driven thin liquid films*, European Journal of Applied Mathematics, Volume 15, 625–642, 2004.

[62] Z. Li and K. Ito, *The Immersed Interface Method - Numerical Solutions of PDEs Involving Interfaces and Irregular Domains*, SIAM series: Frontiers in applied mathematics, 2006, ISBN: 0-89871-609-8.

[63] H. Liu and T. Zhou, *CFD-Based PEM fuel cell models and applications*, Technical Proceedings of the 2003 Nanotechnology Conference and Trade Show, Volume 3, 463–466, Nano Science and Technology Institute, 2003, ISBN: 0-9728422-2-5.

[64] G. Matthies, *Finite Element Methods for Free Boundary Value Problems with Capillary Surfaces*, PhD Thesis, Shaker Verlag, 2002.

[65] J. Martin and W. Moyce, *An experimental study of the collapse of liquid columns on a rigid horizontal plane*, Philos. Trans. A 244, 312–324, 1952.

[66] P. D. Minev, T. Chen, and K. Nandakumar, *A finite element technique for multifluid incompressible flow using Eulerian grids*, Journal of Computational Physics, Volume 187, Issue 1, 255–273, 2003, doi: 10.1016/S0021-9991(03)00098-6.

[67] F. Mut, G. C. Buscaglia, and E. A. Dari, *New mass-conserving algorithm for level set redistancing on unstructured meshes*, Journal of Applied Mechanics, Volume 73, Issue 6, 1011–1016, 2006.

[68] G. Nabh, *On Higher Order Methods for the Stationary Incompressible Navier-Stokes Equations*, PhD Thesis, Universität Heidelberg, Preprint 42/98, 1998.

[69] B. D. Nichols and C. W. Hirt, *Methods for calculating multi-dimensional, transient free surface flows past bodies*, Proc. First Intern. Conf. Num. Ship Hydrodynamics, Gaithersburg, ML, 1975.

[70] F. Nobile, *Numerical Approximation of Fluid-Structure Interaction Problems with Application to Haemodynamics*, PhD Thesis, Number 2458, École Polytechnique Fédérale de Lausanne (EPFL), 2001.

[71] C. E. Norman and M. J. Miksis, *Dynamics of a gas bubble rising in an inclined channel at finite Reynolds number*, Physics of Fluids, Volume 17, Issue 2, 022102, 2005, doi: 10.1063/1.1842220.

[72] S. Osher and J. A. Sethian, *Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations*, Journal of Computational Physics, Volume 79, Issue 1, 12–49, 1988, doi: 10.1016/0021-9991(88)90002-2.

[73] E. Olsson and G. Kreiss, *A conservative level set method for two phase flow*, Journal of Computational Physics, Volume 210, Issue 1, 225–246, 2005, doi: 10.1016/j.jcp.2005.04.007.

[74] E. Olsson, G. Kreiss, and S. Zahedi, *A conservative level set method for two phase flow II*, Journal of Computational Physics, Volume 225, Issue 1, 785–807, 2007, doi: 10.1016/j.jcp.2006.12.027.

[75] A. Ouazzi, *Finite Element Simulation of Nonlinear Fluids with Application to Granular Material and Powder*, PhD Thesis, University of Dortmund, 2005.

[76] A. Ouazzi and S. Turek, *Efficient multigrid and data structures for edge-oriented FEM stabilization*, Preperint No. 306, Institute of applied mathematics, Dortmund University, 2005.

[77] N. Parolini, *Computational Fluid Dynamics for Naval Engineering Problems*, PhD Thesis, Number 3138, École Polytechnique Fédérale de Lausanne (EPFL), 2004.

[78] N. Parolini and E. Burman, *A finite element level set method for viscous free-surface flows*, Applied and Industrial Mathematics in Italy, Proceedings of SIMAI 2004, 417–427, World Scientific, 2005.

[79] N. Parolini and A. Quarteroni, *Mathematical models and numerical simulations for the America's Cup*, Computer Methods in Applied Mechanics and Engineering, Volume 194, Issues 9-11, 1001–1026, 2005, doi: 10.1016/j.cma.2004.06.020.

[80] P.-O. Persson and G. Strang, *A simple mesh generator in Matlab*, SIAM Review, Volume 46, Issue 2, 329–345, 2004, doi: 10.1137/S0036144503429121.

[81] C. S. Peskin, *Numerical analysis of blood flow in the heart*, Journal of Computational Physics, Volume 25, Issue 3, 220–252, 1977, doi: 10.1016/0021-9991(77)90100-0.

[82] J. E. Pilliod, *An Analysis of Piecewise Linear Interface Reconstruction Algorithms for Volume-of-Fluid Methods*, MSc Thesis, University of California, Davis, 1992.

[83] R. Rannacher, *Incompressible viscous flows*, Encyclopedia of Computational Mechanics, Volume 3, 155–182, Wiley, 2004.

[84] Y. Renardy and M. Renardy, *PROST: a parabolic reconstruction of surface tension for the volume-of-fluid method*, Journal of Computational Physics, Volume 183, Issue 2, 2007.

[85] K. J. Ruschak, *A method for incorporating free boundaries with surface tension in finite element fluid-flow simulators*, International Journal for Numerical Methods in Engineering, Volume 15, Issue 5, 639–648, 1980, doi: 10.1002/nme.1620150502.

[86] W. J. Rider and D. B. Kothe, *Stretching and tearing interface tracking methods*, Technical Report AIAA 95-0699, AIAA, 1995.

[87] R. Sedgewick, *Algorithms, 2nd Ed.*, Addison-Wesley, 1988, ISBN: 0-201-06673-4.

[88] J. A. Sethian, *A fast marching level set method for monotonically advancing fronts*, Proceedings of National Academy of Sciences, Volume 93, Issue 4, 1591–1595, 1996.

[89] J. A. Sethian, *Level Set Methods and Fast Marching Methods*, Cambridge University Press, 1999.

[90] J. A. Sethian and A. Vladimirsky, *Fast methods for the Eikonal and related Hamilton-Jacobi equations on unstructured meshes*, Proceedings of National Academy of Sciences, Volume 97, Issue 11, 5699–5703, 2000.

[91] J. R. Shewchuk, *Triangle: engineering a 2D quality mesh generator and delaunay triangulator*, Applied Computational Geometry: Towards Geometric Engineering in Lecture Notes in Computer Science, Volume 1148, 203–222, Springer-Verlag, 1996.

[92] R. Schmachtel, *Robuste Lineare und Nichtlineare Lösungsverfahren für die Inkompressiblen Navier–Stokes–Gleichungen*, PhD Thesis, University of Dortmund, 2003.

[93] K. A. Smith and J. M. Ottino, *Simple representation of contact-line dynamics in a level-set model of an immiscible fluid interface*, Ind. Eng. Chem. Res., Volume 44, Issue 5, 1194–1198, 2005.

[94] A. Smolianski *Numerical Modeling of Two-Fluid Interfacial Flows*, PhD Thesis, Jyväskylä Studies in Computing 8, University of Jyväskylä, 2001, ISBN: 951-39-0929-8.

[95] A. Spira and R. Kimmel, *An efficient solution to the Eikonal equation on parametric manifolds*, Interfaces and Free Boundaries, Volume 6, Issue 3, 315–327, 2004.

[96] M. Sussman and E. Fatemi, *An efficient, interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow*, SIAM J. Sci. Comput., Volume 20, Issue 4, 1165–1191, 1999, doi: 10.1137/S1064827596298245.

[97] M. Sussman and P. Smereka, *Axisymmetric free boundary problems*, Journal of Fluid Mechanics, Volume 341, 269–294, 1997.

[98] M. Sussman, P. Smereka, and S. Osher, *A level set approach for computing solutions to incompressible two-phase flow*, Journal of Computational Physics, Volume 114, Issue 1, 146–159, 1994, doi: 10.1006/jcph.1994.1155.

[99] M. Sussman and E. G. Puckett, *A coupled level set and volume-of-fluid method for computing 3D and axisymmetric incompressible two-phase flows*, Journal of Computational Physics, Volume 162, 301–337, 2000.

[100] Y. R. Tsai, H.-K. Zhao, and S. Osher, *Fast sweeping algorithms for a class of Hamilton-Jacobi equations*, SIAM J. Num. Anal., Volume 41, Issue 2, 673–694, 2003, doi: 10.1137/S0036142901396533.

[101] Y. R. Tsai, *Rapid and accurate computation of the distance function using grids*, Journal of Computational Physics, Volume 178, Issue 1, 175–195, 2005, doi: 10.1006/jcph.2002.7028.

[102] S. Turek, *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*, Series: Lecture Notes in Computational Science and Engineering , Volume 6, Springer-Verlag, 1999, ISBN: 3-540-65433-X.

[103] S. Turek, *On discrete projection methods for the incompressible Navier-Stokes equations: An algorithmical approach*, Computer Methods in Applied Mechanics and Engineering, Volume 143, 271–288, 1997.

[104] S. Turek and A. Ouazzi, *Unified edge-oriented stabilization of nonconforming FEM for incompressible flow problems : Numerical investigations*, J. Numer. Math., Accepted for publication, 2007.

[105] S. Turek, A. Ouazzi, and J. Hron, *On pressure separation algorithms (PSepA) for improving the accuracy of incompressible flow simulations*, Ergebnisberichte des Instituts für Angewandte Mathematik, Nummer 349, FB Mathematik, Universität Dortmund, 2007.

[106] S. Turek and R. Rannacher, *A simple nonconforming quadrilateral Stokes element*, Numerical Methods for Partial Differential Equations, Volume 8, 97–111, 1992.

[107] S. Turek S and M. Schäfer, *Benchmark computations of laminar flow around cylinder*, Flow Simulation with High–Performance Computers II (Notes on Numerical Fluid Mechanics), Volume 52, 547–566, Vieweg, 1996.

[108] S. Turek S, M. Schäfer, and R. Rannacher, *Evaluation of a CFD benchmark for laminar flows*, Proceedings of ENUMATH, World Science Publishing, 1998.

[109] A.-K. Tornberg, *Interface Tracking Methods with Application to Multiphase Flows*, PhD Thesis, NADA, KTH, Stockholm, Sweden, 2000, ISBN: 91-7170-558-9, TRITA-NA 0010.

[110] D. L. Youngs, *An interface tracking method for a 3D Eulerian hydrodynamics code*, Technical Report 44/92/35, AWRE, 1984.

[111] P. Yue, J. Feng, C. Liu, and J. Shen, *A diffuse-interface method for simulating two-phase flows of complex fluids*, Journal of Fluid Mechanics, Volume 515, 293–317, 2004.

[112] O. Verdier, *Benchmark of FEMLAB, Fluent and ANSYS*, Preprints in Mathematical Science, 2004:6 LUFTMA-5039-2004, Lund Institute of Technology, Centre for Mathematical Sciences and Numerical Analysis, 2004.

[113] H. Wadell, Journal of Geology, Volume 41, 310–331, 1933.

[114] F. M. White, *Fluid Mechanics, 4th ed.*, McGraw-Hill 1999, ISBN: 0-07-069716-7.

[115] C. Winkelmann, Seminar presentation and internal discussions at the University of Dortmund, 2007.

[116] J. Xu, *Evaluation of Interface Tracking Schemes with Finite Element Discretizations*, MSc Thesis, Inst. for Applied Math. and Numerics, LS3, University of Dortmund, 2006.

[117] J. Xu, S. Hysing, and S. Turek, *A numerical comparison of interface tracking methods*, To be published in 2008.

[118] Z. Zhang, *Polynomial preserving gradient recovery and a posteriori estimate for bilinear element on irregular quadrilaterals*, Int. J. Numerical Analysis and Modeling, Volume 1, Number 1, 1–24, 2004.

[119] O. C. Zienkiewicz and J. Z. Zhu, *The superconvergent patch recovery and a posteriori error estimators. Part 1. The recovery technique*, Int. J. Numer. Methods Eng., Volume 33, 1331-1364, 1992.

[120] O. C. Zienkiewicz and J. Z. Zhu, *The Superconvergent patch recovery and a posteriori error estimators. Part 2. Error estimates and adaptivity*, Int. J. Numer. Methods Eng., Volume 33, 1365-1382, 1992.