NTNU – Trondheim
Norwegian University of
Science and Technology

The Local Level-Set Extraction Method for Robust Calculation of Geometric Quantities in the Level-Set Method

Åsmund Ervik

Master of Science in Physics and Mathematics
Submission date: July 2012
Supervisor: Ingve Simonsen, IFY
Co-supervisor: Svend Tollak Munkejord, SINTEF Energi AS

Norwegian University of Science and Technology
Department of Physics

SINTEF

NTNU – Trondheim
Norwegian University of
Science and Technology

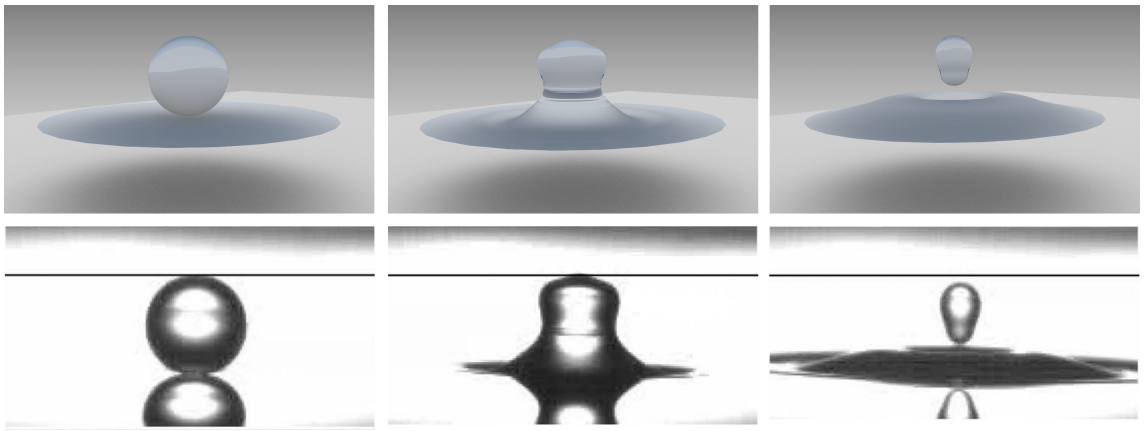
SINTEF Energy Research
NTNU Department of Physics

Master's Thesis

The local level-set extraction method for robust calculation of geometric quantities in the level-set method

Author:
Åsmund Ervik

Supervisors:
Svend Tollak Munkejord (SINTEF)
Ingve Simonsen (NTNU)



Trondheim, July 2012

This document is typeset in 10pt URW Garamond and 10pt Inconsolata.

Preface

This thesis will be submitted for the degree of Master of Science and Technology (Sivilingeniør), concluding my education at NTNU. It is the result of my work during the spring semester of 2012, formally at the Department of Physics. My work has been carried out on behalf of SINTEF Energy Research, under the supervision of Dr. Svend Tollak Munkejord, to whom I am grateful for the support and guidance he has generously provided.

I also want to thank Professor Ingve Simonsen, who has been my supervisor at the Department of Physics, and Mr. Karl Yngve Lervåg at the Department of Energy and Process Engineering, for valuable discussions and suggestions. Finally, I want to thank my wife Elisa for her support and patience during this time, and my daughter Aurora for letting me sleep enough and for being a source of inspiration.

This work is part of the Enabling Low-Emission LNG Systems project at SINTEF Energy Research, and I wish to acknowledge the contributions of GDF SUEZ, Statoil, and the Petromaks programme of the Research Council of Norway (193062/S60).

An electronic version of this document is available at `http://www.pvv.org/~asmunder/mastersthesis.pdf`. This version has numerous hyperlinks, e.g. to most of the articles in the reference list, and some figures may benefit from being able to zoom in on them.

The author's specialization project report, which was completed in January 2012, was on a similar topic concerning level-set methods and curvature calculations. Some sections in this thesis are based on that report, particularly in the Introduction and Theory chapters. An electronic version of the project report is available at `http://www.pvv.org/~asmunder/projectreport.pdf`.

Åsmund Ervik
July 13, 2012

Sammendrag

Level set-metoden er en implisitt metode for å representere og spore en overflate i to eller flere dimensjoner. Metoden er mye brukt bl.a. i datagrafikk, eller som her, ved simulering av tofasestrømning. Motivasjonen bak simuleringene som foretas her er at man ønsker å øke forståelsen av de komplekse tofasefenomenene som forekommer i varmevekslere som benyttes til flytendegjøring av naturgass, f.eks. samspillet mellom dråper og væskefilmer.

En av de største fordelene med level set-metoden er at den håndterer endringer i overflatens topologi på en naturlig måte. I denne oppgaven diskuteres beregning av krumningen og normalvektorene til en overflate representert med level set-metoden. Krumning og normalvektorer beregnes vanligvis med sentraldifferansestensiler, men denne standardmetoden bryter sammen når overflaten endrer topologi, f.eks. når to dråper kolliderer og slår seg sammen. Det har tidligere blitt utviklet flere metoder for å håndtere dette problemet. I denne oppgaven presenteres en ny metode som er en videreutvikling av tidligere metoder. Den nye metoden håndterer mer generelle tilfeller enn tidligere metoder. I motsetning til enkelte tidligere metoder beholdes den implisitte representasjonen av overflaten, slik at metoden enkelt kan utvides til tredimensjonale simuleringer, noe som demonstreres her.

I korthet består den nye metoden i å ekstrahere en eller flere lokale level set-funksjoner for overflater som er i nærheten av punktet som betraktes, for så å ekstrapolere disse og reinitialisere dem for å fjerne knekker. Disse benyttes så til å beregne krumningen og normalvektoren i punktet som betraktes. Dersom det er flere lokale level set-funksjoner, benyttes et vektet gjennomsnitt for krumningen, mens normalvektoren til den nærmeste overflaten blir brukt.

Ved hjelp av denne nye metoden har det blitt simulert flere tilfeller av tofasestrømning som er relevante for å forstå flytendegjøring av naturgass. Den nye metoden muliggjør simuleringer som er mer generelle enn tidligere simuleringer. Det har blitt utført en todimensjonal simulering av en metanol-dråpe med diameter på 0,6 mm som faller gjennom luft og slår seg sammen med en dyp dam av metanol. Den nye metoden gav gode resultater i dette tilfellet, men ufysiske oscillasjoner i trykkfeltet gjorde at dette tilfellet er dårlig egnet til sammenlikning med eksperimentelle resultater.

Flere liknende tilfeller med mye lavere tetthetsforskjell mellom de to fluidene har også blitt betraktet; i disse tilfellene oppførte trykkfeltet seg fysisk, men resultatene er i mindre grad overførbare til å forstå flytendegjøring av naturgass, og er heller egnet til å validere resultatene av den nye metoden. Spesielt har det blitt betraktet en aksesymmetrisk simulering av en vanndråpe med diameter på 0,11 mm i dekan som slår seg sammen med en dyp dam av vann. Resultatene fra denne simuleringen viser svært god overenstemmelse med eksperimentelle data. Det ble også forsøkt gjort

simuleringer av en større dråpe, men for dette tilfellet var det ikke mulig å gjennomføre simuleringen med tilstrekkelig oppløsning til å gjenskape det fysiske resultatet, grunnet økningen i regnetid dette ville ha medført.

Det har også blitt utført beregninger av rent geometriske resultater som validerer resultatene av den nye metoden, samt tredimensjonale resultater for en statisk overflatekonfigurasjon som demonstrerer at metoden er enkelt utvidbar til høyere dimensjoner.

Abstract

The level-set method is an implicit interface capturing method that can be used in two or more dimensions. The method is popular e.g. in computer graphics, and as here, in simulations of two-phase flow. The motivation for the simulations performed here is to obtain a better understanding of the complex two-phase flow phenomena ocurring in heat exchangers used for liquefaction of natural gas, including the study of droplet-film interactions and coalescence.

One of the main advantages of the level-set method is that it handles changes in the interface topology in a natural way. In the present work, the calculation of the curvature and normal vectors of an interface represented by the level-set method is considered. The curvature and normal vectors are usually calculated using central-difference stencils, but this standard method fails when the interface undergoes a topological change, e.g. when two droplets collide and merge. Several methods have previously been developed to handle this problem. In the present work, a new method is presented, which is a development on existing methods. The new method handles more general cases than previous methods. In contrast to some previous methods, the present method retains the implicit formulation and can easily be extended to three-dimensional simulations, as demonstrated in this work.

Briefly, the new method consists in extracting one or more local level sets for bodies close to the grid point considered, reinitializing these local level sets to remove kinks, and using these to calculate the curvature and normal vector at the grid point considered. For the curvature, multiple values are averaged, while for the normal vector, the one corresponding to the closest interface is selected.

With this new method, several two-phase flow simulations are performed that are relevant for understanding the liquefaction of natural gas. The new method enables simulations that are more general than previous ones. A two-dimensional simulation was performed of a 0.6 mm diameter methanol droplet falling through air and merging with a deep pool of methanol. The new method gave good results in this case, but unphysical oscillations in the pressure field rendered this result unsuitable for comparison with experimental results.

Several similar cases with significantly lower density differences between the two fluids were also considered; in these cases, the pressure field behaved physically, but the results are less applicable to the understanding of natural gas liquefication, and better suited for validation of the new method. In particular, an axisymmetric simulation of a 0.11 mm diameter water droplet in decane merging with a deep pool of water has been considered. The results of this simulation show a very close agreement with experimental data. Attempts were also made to simulate a larger droplet, but in this case finer grids were needed than what could be achieved here due to the computational time cost of grid refinement.

Purely geometrical results are also presented in order to validate the results of the new method, and three-dimensional results are given for a static interface configuration, demonstrating that the method is easily extended to higher dimensions.

Contents

Nomenclature

$\nabla \cdot \mathbf{u}$	The divergence of the vector field $\mathbf{u}$.	
$\nabla \mathbf{u}$	The Jacobian matrix of the vector field $\mathbf{u}$.	
Δt or `dt`	Size of time step used in temporal solution.	s
Δx or `dx`	The grid spacing used in spatial discretization. Similar for y and z.	m
κ	The curvature of the interface. It may vary along the interface.	1/m
μ	Dynamic viscosity of a fluid.	Pa·s
ν	Kinematic viscosity of a fluid. $\nu = \mu/\rho$.	m^2/s
ρ	Density of a fluid.	kg/m^3
σ	The surface tension. It may vary along the interface.	N/m
$\mathbf{u}(\mathbf{x})$	Velocity field of a fluid.	m/s
$p(\mathbf{x})$	Pressure of a fluid.	Pa
v	Normal velocity of the interface. $v = \mathbf{n} \cdot \mathbf{u}$.	m/s
$Q(\mathbf{x})$	The quality function, measuring the deviation of ϕ from a signed-distance function.	
Γ	The interface between two fluids. This is the zero level set of ϕ.	
$\delta(\mathbf{x})$	The Dirac delta "function", the distribution with the property $\int f(x)\delta(x)dx = f(0)$.	
ϵ	A small quantity used to avoid dividing by zero.	
η	Threshold used for the quality function.	
γ	The viscosity ratio for a drop relative to the ambient fluid.	
$\phi(\mathbf{x})$	The level-set function. Γ is the set of $\mathbf{x}$ such that $\phi(\mathbf{x}) = 0$.	
$\mathrm{sgn}(x)$	The signum function, $\mathrm{sgn}(x) = x/\lvert(x)\rvert$. The value at zero is $\mathrm{sgn}(0) = 0$.	
$\mathbf{n}$	Normal vector to the surface.	
$\mathbf{t}$	Tangent vector to the surface.	

CFL	Courant-Friedrichs-Lewy
CG	Conjugate Gradient
CNG	Compressed Natural Gas
GFM	Ghost-Fluid Method
GMRES	Generalized Minimal Residual
GPU	Graphics Processing Unit
LNG	Liquefied Natural Gas
LOLEX	Local Level-set Extraction
LSM	Level-Set Method
PDE	Partial Differential Equation
RK	Runge-Kutta
SSP	Strong Stability Preserving
TVD	Total Variation Diminishing
VOF	Volume-of-Fluid
WENO	Weighted Essentially Non-Oscillatory
`ilmax`	Number of local grid points in the LOLEX method. Similarly, `jlmax` and `klmax`.

§1 Introduction

LIQUEFIED NATURAL GAS has in recent years become a major Norwegian export item. In 2010, the Snøhvit field alone produced 5 billion standard cubic meter oil equivalents of natural gas [32], which was processed at the Melkøya production facility. The heat exchangers operating at Melkøya can liquefy 11 000 tonnes of natural gas every 24 h [11], with a final LNG temperature of −162 °C. These numbers indicate that large amounts of energy are being used for the liquefaction of natural gas. As a consequence, reducing energy use in the liquefaction process can lead to large financial savings. It will also improve the reductions in greenhouse gas emissions that result from using LNG instead of compressed natural gas (CNG) or other petroleum-based fuels. It has been estimated [45] that the contribution to greenhouse gas emissions from the liquefaction stage is around 20 g/kWh of CO_2 equivalents. This is roughly the same as for the final combustion stage, so there is a large potential for reduction.

While some reductions in energy consumption can be achieved using standard engineering methods, a deeper fundamental understanding of the processes in a liquefaction heat exchanger is needed in order to further reduce the energy usage. A better understanding may also reduce the downtime of liquefaction systems due to transient operating conditions. To this end, SINTEF Energy has a five-year research programme called “Enabling Low-Emission LNG Systems”, which among other things aims to obtain a better understanding of LNG fluid dynamics through numerical modelling and experiments. The present work is a part of this programme, and aims to improve the existing numerical codes. In particular, the aim is to improve parts of the code that are relevant to colliding drops and to drops colliding with films. Such cases are of high interest when attempting to better understand gas liquefaction.

The numerical codes presently being used have been developed to simulate two-phase fluid flow. In order to simulate two fluids interacting at a detailed level, it is paramount to know at all times where the boundary between the two fluids is located. In the present code, the Level-Set Method (LSM) is used to track the interface. The LSM is very general, and apart from fluid dynamics it has been used for modeling everything from tumor growth [24] to wildland fire propagation [26] and computer RAM production [27]. For a good introduction to the LSM, see e.g. [33]. The LSM originated from the seminal article by Osher and Sethian [34].

In two dimensions, the level-set method can easily be visualized, as in Figure 1.1 on the next page. In this figure, the interface is between the grey and white areas in the right half. To track this interface in 2D, a function $\phi(x)$ is used, shown on the left in metallic grey. The interface is then represented by the level set $\Gamma = \{x | \phi(x) = 0\}$. $\phi(x)$ is the level-set function, and the reason for the name of this function should now be

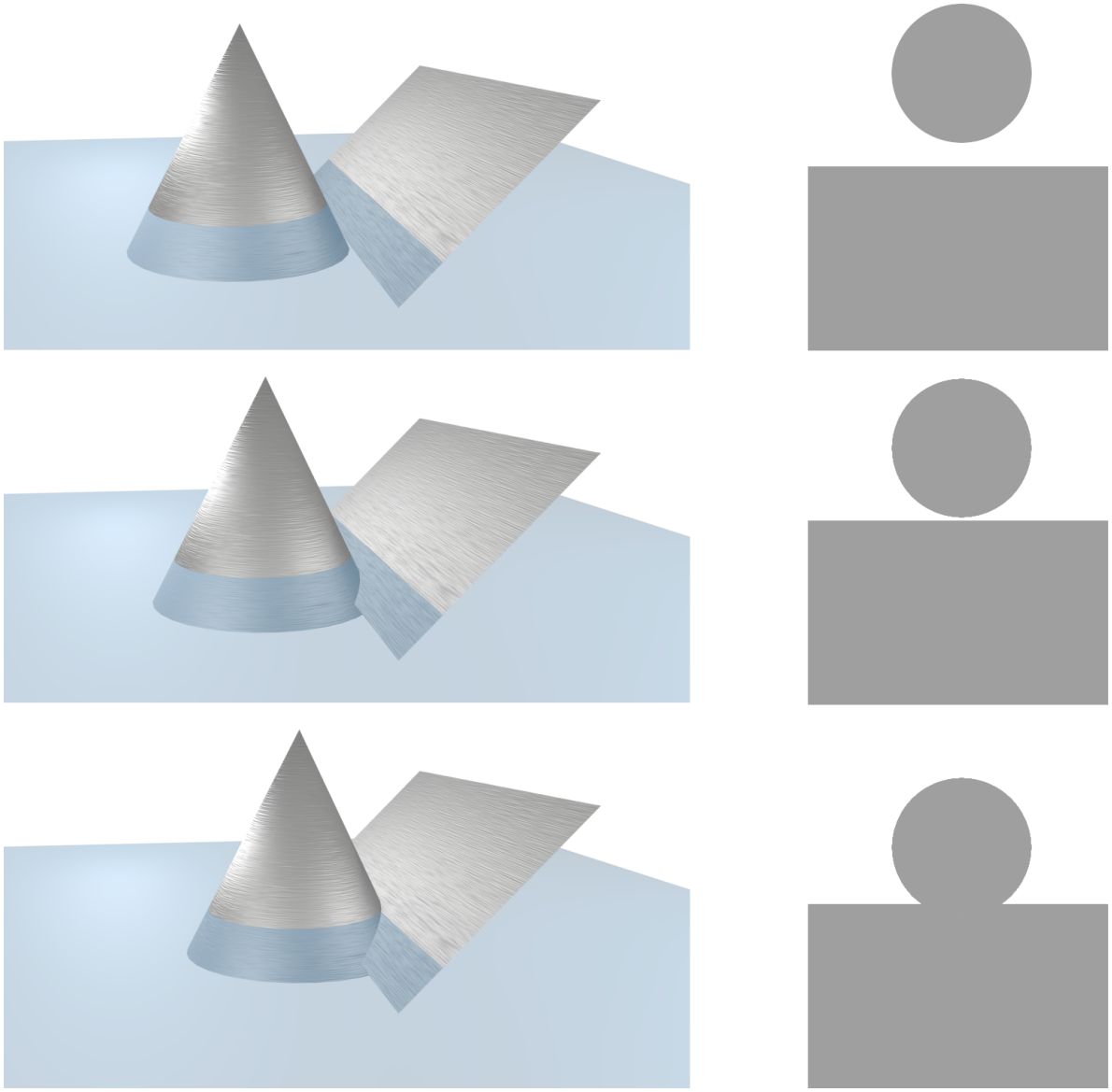


Figure 1.1: Illustration of interface tracking using the level-set method. The interface is between the grey and the white area on the right-hand side of the image, and the level-set function is shown in grey on the left-hand side. Also shown on the left-hand side, in blue, is the plane $\phi(\mathbf{x}) = 0$.

self-evident.

In two-phase flow simulations using the LSM, accurate interface curvature and normal vector information is vital in order to get good results. Standard methods exist for calculating these geometric quantities, but fail when the interface topology changes, e.g. when two drops collide and merge. The present work proposes a new method for calculating these quantities, which is an extension of previous methods. The proposed method handles general interface configurations and topology changes, and extends easily to three dimensions.

The outline of this report is as follows: In Chapter 2, the theory of two-phase incompressible flow, the LSM and numerical methods is given. In Chapter 3, the proposed method is presented in detail. In Chapter 4, the method is validated on geometric test cases, and the results are compared to other methods. In Chapter 5, the results of two-phase flow simulations using the current method are reported. In Chapter 6, experimental results in the literature are reviewed and compared to the simulation results. Finally, in Chapter 7, some concluding remarks are offered and future prospects are discussed.

§2 Theory of the level-set method and two-phase fluid simulation

Contents

> Even if there is only one possible unified theory, it is just a set of rules and equations. What is it that breathes fire into the equations and makes a universe for them to describe?
>
> Stephen Hawking

THE THEORY OF THE LEVEL-SET METHOD is a large subject which cannot possibly be completely reviewed in detail here. For an extensive review of the LSM and its applications, see the article by Osher and Fedkiw [33]. In this chapter, a brief introduction to the LSM will be given, along with an overview of how the method is coupled to the physics of multiphase flow. Special emphasis will be given to the topics of reinitialization and of curvature calculation, which are important in this work. A short introduction to the numerical methods used is also given.

2.1 The Level-Set Method

THE LSM IS ONE OF THE MORE SUCCESSFUL interface capturing methods used in computational physics. Since its introduction by Osher and Sethian in [34], it has been used for numerous physical applications, as well as in computer graphics.[1]

Perhaps the main virtue of the LSM is how intuitive it is; in 2D it can easily be explained to anyone with a basic knowledge of multivariate calculus. This simplicity stems from the implicitness of the LSM. This also makes the numerical implementation of the LSM relatively easy. When comparing the LSM to other interface tracking methods, such as the Front Tracking Method where the interface is represented by

[1] Industrial Light+Magic has used the LSM in several Hollywood blockbusters, e.g. *Star Wars: Episode III* and the *Pirates of the Caribbean* movies, to create realistic ocean animations with effects like water spray.

piecewise continuous functions, the simplicity becomes especially clear.

The main disadvantage of the LSM, on the other hand, is that it is not a conservative method. During the course of a simulation, a fraction of fluid 1 may be converted to fluid 2 in an unphysical fashion. Various methods have been invented to circumvent this, e.g. the HCR-2 reinitialization method[14], so it is only a small effect presently. Interface tracking methods may be conservative; an example of this is the Volume-of-Fluid (VOF) method, but then they typically have other disadvantages. In the VOF method, for instance, the advection equation cannot easily be solved, necessitating the use of interface reconstruction methods. See e.g. [37], where particularly Figure 11 illustrates the challenges presented by this fact. Recent efforts have attempted to join the LSM and VOF in order to get the benefits of both methods; this approach seems to be fairly successful[43].

To expand on the simplistic presentation of the LSM given in the introduction, we present the formal definitions here. Let Γ be the interface between two fluids, e.g. air and water. The interface Γ has codimension 1 to the space S we are working in, that is, in 3D the interface has two dimensions. S is the physical domain where the fluids under study are confined, e.g. a cubic box. Furthermore, a physical interface such as the one between air and water clearly has an inside and an outside, so Γ is an orientable surface. To represent this interface, we define a *level-set function* $\phi : S \to \mathbb{R}$ with the property

$$\Gamma = \{\mathbf{x} \mid \phi(\mathbf{x}) = 0\}. \tag{1}$$

This only defines the value of ϕ at the interface Γ. Away from the interface, we have not specified what ϕ is yet. As we only care about the value at the interface, we have some freedom here, but the common choice is a signed distance function. Thus ϕ is fully specified by

$$\phi(\mathbf{x}) = \begin{cases} -\text{dist}(\mathbf{x}, \Gamma) & \text{if } \mathbf{x} \text{ is inside } \Gamma, \\ \text{dist}(\mathbf{x}, \Gamma) & \text{if } \mathbf{x} \text{ is outside } \Gamma. \end{cases} \tag{2}$$

Here, the function $\text{dist}(\mathbf{x}, \Gamma)$ is the shortest distance from the point $\mathbf{x} \in S$ to the interface Γ. As ϕ is a mapping from $S \to \mathbb{R}$, it can be embedded as a surface in $S \times \mathbb{R}$. As an example of this with S equal to 2D Euclidean space, the level-set function ϕ representing a circle in 2D can be viewed as a cone with an angle of 90° at the tip, as seen in Figure 1.1[2]. The cone is of course a surface embedded in 3D Euclidean space.

With this picture of the level set function as a cone, another virtue of the level-set approach can be nicely illustrated. If we look at $\nabla\phi$, we known that this vector will point in the direction that ϕ increases the most. For the cone, this is simply the normal vector to the circle, or in general to any interface, when evaluated at the interface. If

[2]The tip angle is smaller than 90° in Figure 1.1 for aesthetic reasons

ϕ is not exactly a signed distance function, we need to normalize $\mathbf{n}$, so it is given by

$$\mathbf{n} = \frac{\nabla\phi}{|\nabla\phi|}. \tag{3}$$

From this, the curvature is given by the well-known formula

$$\kappa = \nabla \cdot \mathbf{n} = \nabla \cdot \left(\frac{\nabla\phi}{|\nabla\phi|} \right). \tag{4}$$

With suitable discretizations of the derivatives involved, these quantities are easy to calculate numerically. This is often quoted as one of the nice features of the LSM, along with e.g. the very natural way the method handles topological changes [36]. However, when curvature calculations are combined with topological changes, things are not so rosy, as is seen in the following. The standard discretization of the curvature is (see e.g. [18])

$$\kappa = \frac{\phi_{xx} + \phi_{yy}}{(\phi_x^2 + \phi_y^2 + \epsilon)^{1/2}} - \frac{\phi_x^2\phi_{xx} + \phi_y^2\phi_{yy} + 2\phi_x\phi_y\phi_{xy}}{(\phi_x^2 + \phi_y^2 + \epsilon)^{3/2}} \tag{5}$$

Here, e.g. ϕ_x denotes the first derivative of ϕ in x-direction, calculated using standard central differences.

In [41], Smereka notes regarding curvature that "One of the major advantages of level-set methods is their ability to easily handle topological changes. However for this problem we have found this not to be the case."

2.2 Advection and reinitialization of ϕ

From the defining Equation (2), ϕ is intialized at the start of a simulation. For a given velocity field $\mathbf{u}$, ϕ should be transported so that the interface follows the flow. This is done by solving the advection equation,

$$\frac{\partial\phi}{\partial t} = v|\nabla\phi| = -\mathbf{u}\cdot\nabla\phi. \tag{6}$$

Here v is the velocity normal to the interface, and the first equality is a Hamilton-Jacobi type equation. The second equality follows from the normal velocity being $v = \mathbf{u}\cdot\mathbf{n}$ and inserting the expression for $\mathbf{n}$ given in Equation (3). This equation is not justified here, see e.g. [44].

Solving this advection equation will of course result in transportation of the interface Γ and of the level-set function ϕ. But it also has a side effect: it will stretch and compress the level-set function, making it different from a signed distance function. Over time, this makes the LSM less accurate in tracking the position of the interface.

If we imagine one extreme of stretching, $|\nabla\phi| \to 0$, this loss of accuracy is easy to understand: for such a very flat ϕ, adding a small numerical error to ϕ will displace the interface position by a very large amount. Because of this we want to keep ϕ equal to a signed distance function, which brings us to the topic of *reinitialization* of ϕ.

Reintialization is, as the name suggests, to reset the value of ϕ to an intial condition of some sort. The initial condition given in Equation (2) is a signed distance function, so with reinitialization we want to transform an arbitrary ϕ into a signed distance function with the same zero level set. It is essential that this last criterion be fulfilled, namely that the interface is the same before and after reinitialization. The reinitialization procedure was introduced by Sussman, Smereka and Osher [44], and consists in solving the PDE

$$\frac{\partial\phi}{\partial t} + \operatorname{sgn}(\phi)(|\nabla\phi| - 1) = 0 \tag{7}$$

to steady state. Intuitively, one might reinitialize by simply computing the signed distance to the interface for all grid points. This approach has two problems: first, it is very slow, requiring $\mathcal{O}(n^3)$ operations (in 3D) even after some clever optimizations [4]. Second, it may distort the interface due to grid errors.

The PDE-based approach introduced in [44] is much faster computationally, and avoids problems with grid errors. It is justified by the fact that the signed distance function is the unique viscosity solution of the Eikonal equation, $|\nabla\phi| = 1$, with the initial value of ϕ at the interface. With this in mind, solving the Eikonal equation for N pseudo-time steps will ensure that ϕ is a signed distance function $C \cdot N$ space steps away from the interface, where C is the CFL-number used when solving Equation (7) numerically. This is so because the characteristics of Equation (7) originate at the interface, a very useful property of this equation[44].

This equation is discretized using the advanced methods presented below in Section 2.5. To avoid these technicalities here, we consider instead the 1D version with a forward Euler integration in τ. The extension to higher-order methods is straightforward, although tedious. The forward Euler integration gives

$$\phi_i^{\nu+1} = \phi_i^{\nu} - \Delta t \; S(\phi_i^0)\, \mathcal{G}(\phi)_i, \tag{8}$$

where $S(\phi)$ is the smoothed signum function given by $S(\phi) = \frac{\phi}{\sqrt{\phi^2 + 2\Delta x^2}}$.[3] $\mathcal{G}(\phi)_i$ is the discretization of the so-called Godunov Hamiltonian, discussed in e.g. [44]. The details of $\mathcal{G}(\phi)_i$ are not important for the purposes of this discussion, but since there exist at least two subtly different versions both in use, it is given here:

$$\mathcal{G}(\phi)_i = \begin{cases} \sqrt{\max(a_+^2, b_-^2)} - 1 & \text{if } \phi_i^0 > 0 \quad , \\ \sqrt{\max(a_-^2, b_+^2)} - 1 & \text{if } \phi_i^0 < 0 \quad , \end{cases} \tag{9}$$

[3] Here Δx is a small numerical constant to avoid dividing by zero. The grid spacing is a common choice.

where $a_+ = \max(a,0)$, $a_- = \min(a,0)$, and $a = D_x^- \phi_i$ (the backward difference in x-direction) and $b = D_x^+ \phi_i$ (the forward difference in x-direction). What is important to notice here is the -1-term which is not included by some authors, who prefer to include it explicitly in Equation (8). $\mathcal{G}$ is written here in a slightly awkward form in order to make the generalization to 2D and 3D obvious by analogy to the Euclidean distance formula.

When solving this in practice, higher order versions of the forward and backward differences are used. The fourth- and fifth-order WENO discretizations introduced in Section 2.5 are commonly used. These are upwinding discretizations, and need accurate normal vectors at the interface in order to correctly determine which direction is upwind. This will prove to be a crucial point further down.

2.3
The Navier-Stokes equations

THE NUMEROUS INTERESTING PHYSICAL SITUATIONS that can be studied with the level-set codes used here all share few common traits: they involve two incompressible fluids, these fluids do not mix, and they are both Newtonian fluids. In each phase, the density and viscosity is assumed constant, but they may of course vary between phases. Further it is assumed that the temperature is constant, that no chemical reactions happen etc. Together, these assumptions give the governing equations of the system being studied, namely the celebrated Navier-Stokes equations:

$$\nabla \cdot \mathbf{u} = 0 \tag{10}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{\nabla p}{\rho} + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{11}$$

Here $\nu = \mu/\rho$ is the kinematic viscosity, while μ is the dynamic viscosity. ρ is the density, $\mathbf{u}$ is the velocity field and p is the pressure. $\mathbf{f}$ is any external force, such as gravity, and may be zero.

These equations hold for single phase fluid flow, but we are interested in more complicated problems. However, it turns out that by adding an additional force term, which is singular and zero outside the interface, the two-phase problem is equivalent to the single-phase Navier-Stokes equations. This surface force term is given by

$$\mathbf{f}_s(\mathbf{x},t) = \int_\Gamma \mathbf{f}_{\text{sfd}}(\mathbf{s},t)\delta(\mathbf{x} - \mathbf{x}_I(s))d\mathbf{s}, \tag{12}$$

where $\mathbf{f}_{\text{sfd}}$ is a surface-force density dependent on the actual system, and $\mathbf{x}_I(s)$ is a parametrization of the interface. Note that in level-set contexts, it is not necessary to

parametrize the interface in the actual code, since $\phi(\mathbf{x})$ stores the distance to the interface. Note that the delta function must be regularized, since the numerical resolution is finite. There are several ways to do this; here we use Equation (1.23) from [33], which is

$$\delta(x) = \begin{cases} 0 & \text{if } |\phi| > \epsilon \\ \frac{1}{2\epsilon}\left(1 + \cos\left(\frac{\pi\phi}{\epsilon}\right)\right) & \text{else} \end{cases} \tag{13}$$

where $\epsilon = 1.5\Delta x$ is used here. This will be referred to as the standard smeared delta function.

This surface force term is explained and justified more thoroguhly in [20, Chapter 2.2], and is a big advantage: we can use existing methods for the single-phase problem! However, we still have to take care of physical properties that vary between the two fluids. This is done using the Ghost-Fluid Method (GFM) in the present work; other alternatives include the Continuum Surface Force (CSF) method.

Another problem when solving the Navier-Stokes equations is the pressure term. The pressure is coupled back to the velocity, so a decoupling is needed in order to facilitate the numerical solution. This is made possible by the Helmholtz-Hodge theorem. After the decoupling, the pressure is given by a Poisson equation, and the resulting system of equations can be solved numerically. For an introduction to the application of the Helmholtz-Hogde theorem in projection methods for incompressible flows, see e.g. [5].

It should be noted that the notation used for operators in Equations (10) to (11) is a little imprecise and must be interpreted correctly. In particular, when using some non-Cartesian coordinate systems (like spherical coordinates), using the divergence operator $\boldsymbol{\nabla}\cdot$ on a vector field is not equal to the inner product of the $\boldsymbol{\nabla}$ operator with the vector field. For this reason, some prefer the notation **div** and **curl**. Here, the operator notation is used, and it is understood that $\boldsymbol{\nabla}\cdot$ means the divergence operator etc.

This brings us to another point, namely the difference between 2D and 3D axisymmetry. Naively, one might think that a symmetric 2D simulation can simply be revolved 180° around the symmetry axis to give a 3D axisymmetric result. This is not the case. Perhaps the easiest way to see the difference is to compare the operator $\boldsymbol{\nabla}^2$ for a 2D Cartesian coordinate system and for a 3D axisymmetric coordinate system with a fixed azimuthal angle $\theta = 0$. By fixing θ, we indicate that none of the physical quantities depend on θ. The 2D coordinates (x, z) are then in correspondence with the

axisymmetric coordinates (ρ, ζ), where we use the convention

$$x = \rho\cos(\theta) = \rho \tag{14}$$
$$y = \rho\sin(\theta) = 0 \tag{15}$$
$$z = \zeta \tag{16}$$

This coincidence does not, however, mean that the physics is the same in these two systems. Writing out the Laplacian using both Cartesian and axisymmetric coordinates, we obtain [39]

$$\nabla^2_{\text{cart}} = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2} \tag{17}$$
$$\nabla^2_{\text{axi}} = \frac{\partial^2}{\partial \rho^2} + \frac{\partial^2}{\partial \zeta^2} + \frac{1}{\rho}\frac{\partial}{\partial \rho} \tag{18}$$

where an additional term contributes in the axisymmetric case.

2.4 The Ghost-Fluid Method

THE DISCONTINUITY OF ρ AND μ from one fluid to another leads to several jump conditions that can be derived for physical properties across the interface. This is reminiscent of the similar situation in electrodynamics, where one constructs Gaussian pill-boxes and Amperian loops in order to find the correct jump conditions. Similar considerations here, see e.g. [20], yield Equations (19) to (23), where $[a]$ denotes the difference in a across the interface.

$$[\mathbf{u}] = 0, \tag{19}$$
$$[p] = 2[\mu]\mathbf{n}\cdot\nabla\mathbf{u}\cdot\mathbf{n} + \sigma\kappa, \tag{20}$$
$$[\mu\nabla\mathbf{u}] = [\mu]\Big((\mathbf{n}\cdot\nabla\mathbf{u}\cdot\mathbf{n})\mathbf{nn} + (\mathbf{n}\cdot\nabla\mathbf{u}\cdot\mathbf{t})\mathbf{nt} - \tag{21}$$
$$(\mathbf{n}\cdot\nabla\mathbf{u}\cdot\mathbf{t})\mathbf{tn} + (\mathbf{t}\cdot\nabla\mathbf{u}\cdot\mathbf{t})\mathbf{tt}\Big) - (\mathbf{t}\cdot\nabla_s\sigma)\mathbf{tn}), \tag{22}$$
$$[\nabla p] = 0. \tag{23}$$

Here $\mathbf{n}$ is the normal vector to the surface and $\mathbf{t}$ is the tangent vector, and products like $\mathbf{nn}$ denote the outer (tensor) product. Correspondingly, $\nabla\mathbf{u}$ does not indicate the divergence, but the Jacobian tensor. Thus e.g. $\mathbf{n}\cdot\nabla\mathbf{u}\cdot\mathbf{n}$ is a scalar. Note that the sign convention for $\mathbf{n}$ and the definition of $[a]$ have to be consistent, if so these equations

hold regardless of the sign convention. The convention used here is that normal vectors point towards increasing ϕ, and that the jump is $[a] = a^+ - a^-$, where a^+ is evaluated further towards increasing ϕ than a^-.

These jump conditions have to be imposed on the numerical solutions somehow. Early attempts at this used the jump conditions explicitly for 1D simulations, but these approaches are cumbersome to implement in 2D or higher dimensions [8]. The Ghost-Fluid Method (GFM) avoids this by implementing the jump conditions as a modification of the difference stencils close to the interface, enforcing the jump conditions given above. The GFM was introduced in [7] for the Euler equations, and extended to the Navier-Stokes equations in [8].

2.5 Numerical methods

When solving the PDEs introduced in the previous sections on a computer, both the spatial discretizations and time integration (e.g. Runge-Kutta) schemes for evolving the system in time have to be chosen with care. Since the cases considered here involve two fluids in contact, there is always a contact discontinuity in the pressure, density etc. Such discontinuitites are problematic for both the spatial and time discretizations, and must be handled with care.

Consider first the space discretizations, using as an example the discretization of $\frac{\partial u}{\partial x}$ in 1D. A very common choice is the central difference

$$\frac{\partial u(x)}{\partial x} \approx \frac{u(x+\frac{1}{2}\Delta x) - u(x-\frac{1}{2}\Delta x)}{\Delta x}. \tag{24}$$

This is a second-order accurate discretization, and it is obviously a linear scheme. From this fact it follows, using Godunov's theorem, that the discretization may introduce spurious oscillations when discontinuities are present. Godunov's theorem is a no-go theorem stating that linear schemes for solving PDE's can be *at most* first order accurate if we demand that they do not introduce new extrema (i.e. oscillations).

Several approaches have been developed to circumnavigate this restriction, such as the use of flux limiters. In the present numerical codes, a discretization scheme known as WENO-5 is used, see [17] for details and justification of this scheme. WENO schemes are known to be computationally efficient in addition to being robust[17]. WENO stands for Weighted Essentially Non-Oscillatory, and is an improvement to the earlier Essentially Non-Oscillatory (ENO) schemes. To avoid too many technicalities, the reader may think of the ENO scheme as an interpolation method where an r-point stencil is chosen around the point x such that the resulting interpolating polynomial introduces the least oscillations. This can be done e.g. using divided differences. As the

name suggests, this results in a solution with essentially no oscillations, although they cannot be completely avoided as Godunov's theorem dictates.

The drawback of the ENO method is that it uses the least oscillating interpolation also when the solution is smooth, where it is possible to achieve higher precision with the same amount of computation. This is because when choosing between all possible r-point stencils around x, we have used $2r-1$ points in the intermediate calculations. Away from discontinuities, it should be possible to utilize all these points. This is the improvement made by the WENO scheme.

The WENO scheme extends the ENO approach by using a weighted average of all the interpolations available to the ENO scheme. The weighting is such that when the function to be interpolated is smooth, the weighting is close to the optimal choice, giving a $2r-1^{\text{th}}$ order interpolation. When the function to be interpolated has a discontinuity, it essentially falls back to the ENO scheme which gives a r^{th} order interpolation. This is the elegance of the WENO schemes, that one can easily compute which ENO discretizations are nice at a given point. No more technical details will be given here about niceness estimates etc., the reader is again referred to [17] for these.

When the temporal solution of a PDE is considered, similar problems arise. We consider one of the most commonly used class of methods for "integrating" a PDE one step in time, namely the Runge-Kutta methods. For the purposes of this discussion, assume that the spatial discretization has already been done, so that the PDE is of the form

$$\frac{\partial u(t)_i}{\partial t} = f\left(u(t)_i, u(t)_{i+\frac{1}{2}}, u(t)_{i-\frac{1}{2}}, \dots\right) = f(u). \tag{25}$$

Here, the last equality defines the shorthand $f(u)$. The $\dots$ indicates all the spatial evaluations involved in the spatial discretization. The Runge-Kutta method (or RK method) then estimates the solution at the next time step, $t+\Delta t$, using s evaluations of the function $f(u)$. From this, we say that a given RK method is an s-stage method. An exampe is the classical RK method, which is a fourth-order four-stage RK method, where fourth order means that the error made in each step is $\mathcal{O}(\Delta t^5)$.

A potential problem with the classical RK method and other related RK methods is that they may produce unstable solutions in the presence of discontinuities. To minimize this, a special class of RK methods known as Strong Stability-Preserving (SSP-RK) methods are used. The SSP-RK methods come from the theory of Total Variation Diminishing (or TVD) solutions of hyperbolic PDEs. TVD solutions are first-order Euler integrations of the PDE where the spatial discretization is designed to keep the conservation property, e.g. conservation of momentum, of the original PDE. This is done by requiring that going to the next time step does not increase the total variation of the solution. SSP-RK methods are extensions of this technique, where the RK method is designed to still keep this conservation property while simultaneously

allowing for a more accurate solution than the first-order Euler method. The SSP-RK methods used in practice belong to a further sub-class of methods called low-storage methods. As the name implies, these use less memory than more straight-forward alternatives. For examples of low-storage methods and a good discussion of SSP-RK methods in general, see [19].

The more accurate of the SSP-RK methods used here, the third-order SSP-RK 3()4, is used when solving the Navier-Stokes equations. The less accurate second-order SSP-RK 2()4 is used for e.g. solving the reinitialization equation, where high accuracy is not as important. The reader may wonder why a four-stage method is used, if the desired accuracy is only second order. This is because the SSP coefficient, a number describing the maximum time step size in an analogous way to the CFL condition, is much higher for the optimal four-stage method – 3 – than the optimal two-stage method – 1. This means that a time step three times as large can be used while still preserving the SSP property. In general, the SSP coefficient for an optimal s-stage method is $(s-1)$. This is discussed in much greater detail in [42].

In computational fluid dynamics, the use of a CFL criterion to dynamically determine the time step size Δt is common. This criterion is usually formulated as $\Delta t f(\Delta x) \leq C$, where $f(\Delta x)$ is some function of the step size, and C is known as the CFL number. The idea behind this is to use as large a Δt as possible without the solution becoming unstable (or unphysical). Using the largest possible Δt reduces the computation time, but will also reduce the negative effects of any numerical smearing that is introduced to handle discontinuities. For e.g. hyperbolic systems, the CFL criterion can be derived in a rigorous fashion, and the system is known to be stable if this criterion is fulfilled. For the Navier-Stokes equations, however, it is not so straightforward, particularly for multiphase flow. In the present work, we use the CFL criterion derived by Kang et al. in [18]. It is given as

$$\Delta t \left(\frac{(C_{\text{cfl}} + V_{\text{cfl}}) + \sqrt{(C_{\text{cfl}} + V_{\text{cfl}})^2 + 4(G_{\text{cfl}})^2 + 4(S_{\text{cfl}})^2}}{2} \right) \leq C \tag{26}$$

where C_{cfl} is called the convective CFL number, V_{cfl} is called the viscous CFL number, G_{cfl} is the gravitational CFL number, and S_{cfl} is the surface tension CFL number. C is the total CFL number, and will be referred to as just the CFL number in the following.

The partial CFL numbers are given as

$$C_{\text{cfl}} = \left(\frac{|u|_{\max}}{\Delta x} + \frac{|v|_{\max}}{\Delta y} + \frac{|w|_{\max}}{\Delta z} \right) \tag{27}$$

$$V_{\text{cfl}} = \max\left\{ \frac{\mu_1}{\rho_1}, \frac{\mu_2}{\rho_2} \right\} \left(\frac{2}{(\Delta x)^2} + \frac{2}{(\Delta y)^2} + \frac{2}{(\Delta z)^2} \right) \tag{28}$$

$$G_{\text{cfl}} = \sqrt{\frac{|g|}{\Delta y}} \tag{29}$$

$$S_{\text{cfl}} = \sqrt{\frac{\sigma|\kappa|}{\min\{\rho_1, \rho_2\}(\min\{\Delta x, \Delta y, \Delta z\})^2}} \tag{30}$$

The subscripts 1 and 2 here refer to the different fluids, and the grid is assumed to be uniform. If a non-uniform grid is used, a maximum over all grid points of quantities involving the grid spacing must be used as well. Using these definitions, the actual CFL number used in simulations depends on how stable the case being considered is. Kang et al. use a CFL number of 0.5 in [18]; in the simulations reported in the following, CFL numbers ranging from 1 to 0.1 are used.

This completes the discussion on solving the Navier-Stokes equations and the level-set equations, e.g. for reinitialization. In addition to this, a Poisson equation for the pressure must be solved at each time step, as indicated in Section 2.3. Solving this equation is done using the PETSc library, www.mcs.anl.gov/petsc/, which provides several Poisson solvers. These include methods based on the well-known conjugate gradient (CG) method or the generalized minimal residual (GMRES) method. The GMRES method is generally more robust than the CG method, but unless otherwise stated, the simulations performed here are not particularly sensitive to the choice of Poisson solver.

A final feature of the present codes that is worth mentioning is the use of a staggered grid. In simple terms, this means that scalar values are stored at cell centers and vector values are stored at cell faces. This commonly used approach makes some computations less cumbersome, since the vectorial values are often needed at the cell faces, e.g. we need the value $u_{i+1/2,j}$. It also avoids checkerboarding of the pressure, a problem with unstaggered grids where the pressure becomes oscillatory and unphysical. The checkerboarding problem can also be avoided for an unstaggered grid by using Rhie-Chow interpolation [38]. See [51, Section 6.2] for a more detailed introduction to the use of staggered grids.

2.6
Final remarks

FURTHER DETAILS OF THE METHODS used here, e.g. a thorough derivation of the jump conditions across the interface, can be found in [20] and [13]. As the methods to be discussed in this thesis mainly focus on the level-set aspect of modelling, the theory of incompressible two-phase flow is not reviewed in great depth here; on this topic, said references provide more detail. A more detailed exposition of the numerical methods used can be found in [17] and [42]. This concludes the theory chapter. The next chapter introduces and motivates the proposed method for robust calculation of geometric quantities.

§3 The local level-set extraction method

Contents

> A good theory sticks its neck out by making claims that can, at least in principle, be found to be wrong.
>
> Richard Feynman

3.1 Introduction

CALCULATING THE CURVATURE of the interface between two phases is important, since its value is used e.g. in the Ghost Fluid Method (GFM) (or other methods of enforcing the jump conditions). Recalling the discussion of the GFM in Section 2.4, it is seen that the curvature κ determines part of the jump in the pressure across the interface, Equation (19). In a similar fashion, the normal vectors to the interface are important, e.g. when advecting the level-set function and when reinitializing it. Calculating these geometric quantities is straighforward in theory, using Equation (3) and Equation (4) to compute them from the level-set function.

However, as is often the case, in practice it is not so straightforward. The problems arise when the distance between two interfaces is of the order Δx. In this case the kink in ϕ, which has to exist between the interfaces, will often cause the derivatives of ϕ to become undefined.[4] For a graphical depiction of the problem, see Figure 3.1. When this happens, the curvatures and normal vectors will be erroneous, sometimes sufficiently so to make the simulations crash.

Several approaches have been used to remedy this flaw. The first approach to this problem is described by Smereka in [41]. He describes the problem briefly, and increases the numerical smoothing in the curvature discretization to lessen the effect. This is not an optimal solution, and Smereka notes on one of the simulations with merging interfaces that "most of the area loss occurs at the topology change". He also notes that he is not satisfied with this approach.

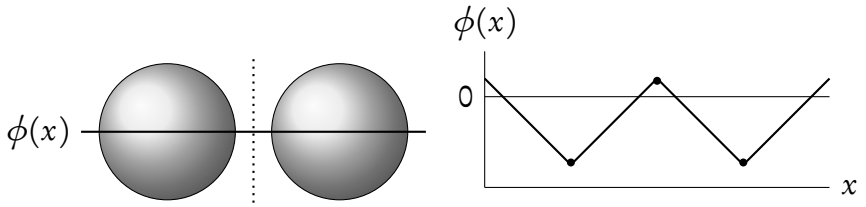
[4] In the numerical sense, undefined does not necessarily mean infinite, but rather large, erroneous values.

The earliest non-smearing approach, by Macklin and Lowengrub [24], uses a modification of the directional differences for points close to kinks, along with mesh refinement for these points. The same authors introduced a curve-fitting method instead in [25], which is said to be an improvement on the directional differences and a simplification. Further improvements to this method, and adaptations to the framework used in the present level-set codes (i.e. calculating the curvature at the grid points, not at the interface), have been developed by Lervåg [21],[22]. These methods work well, but are difficult to extend to 3D due to the use of curve-fitting.

An alternative approach to the problem is due to Salac and Lu [40], and will be referred to as the Salac and Lu method. In essence, this approach extracts the bodies represented by the level-set function, such that each body (e.g. drop) is momentarily given its own version of the computational domain. In this dedicated version, ϕ does not represent any other bodies that can induce kinks, and this temporary ϕ can be reinitialized and the geometric quantities can be calculated without problems. The author likes to think of this as a layer-based approach, in analogy with layers used in popular image editing software, as illustrated in Figure 3.2. For a review and comparison of these methods, see the article by Lervåg and Ervik [23].

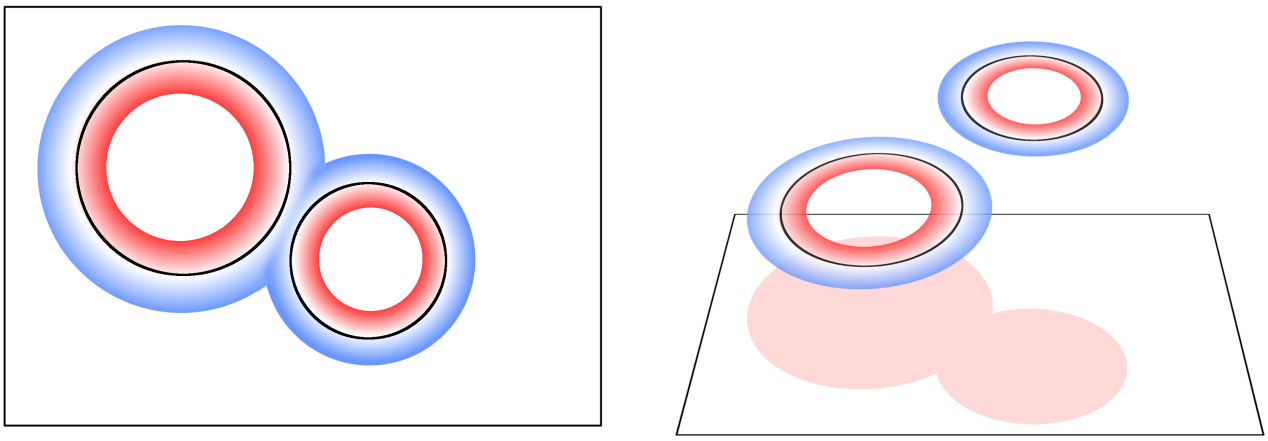
The method considered here is a further development of the Salac and Lu method. It is referred to as the local level-set extraction method, or LOLEX method in short. The reason why the Salac and Lu method is insufficient in some cases, as well as the details of the present method, is given below.

An approach which has not been considered here, or by others in the context of level-set methods as far as the author is aware, is the use of filtering. Vliet and Verbeek [50] study the estimation of curvature from a discretely sampled greyscale image, using derivative-of-Gaussian filters, and obtain good results. A similar approach could perhaps be made to work for curvature calculation in the level-set method, but it is not known if such an approach could be made to reliably calculate normal vectors.



(a) Droplets in near contact (b) A slice of the level-set function

Figure 3.1: (a) Two droplets in near contact. The dotted line marks a region where the derivative of the level-set function is not defined. (b) A one-dimensional slice of the level-set function. The dots mark points where the derivative of ϕ is not defined. (Figure due to Karl Yngve Lervåg)



(a) Original level-set function

(b) Extracted level-set functions

Figure 3.2: Illustration of the Salac and Lu approach. In Figure (b), two level set functions are shown in different layers. These two have been extracted from the single level-set function in Figure (a). The zero level-set is highlighted in black, as well as the edges of the computational domain. In both cases, the level-set function is only shown for a region around the zero level-set, inside the commonly employed "computational tubes".

We give here an outline of the following sections: we start in Section 3.2 by motivating the present method. Following this, the idea behind the LOLEX method is described in Section 3.3, and detailed explanations, pseudo-code and figures are presented in Section 3.4 to illustrate the details of the method. Some concepts from the Macklin and Lowengrub method are also introduced, since they are used in the current method. The actual code used is presented in Appendices A-F; the programming language used is Fortran 90/95. We finally give a brief retrospect of this chapter along with some closing remarks. Results obtained with the LOLEX method are presented in the next chapters.

3.2 Motivation of the LOLEX method

THE IDEA OF SALAC AND LU is simple when compared to the curve-fitting scheme used by Macklin and Lowengrub [24] and later by Lervåg[21],[22]. This simplicity is more in keeping with the "spirit" of the level-set method: the LSM is

an implicit alternative to front tracking methods that employ curve fitting, and this implicitness makes extending to higher dimensions straightforward. In the same fashion, the Salac and Lu method is easily extensible to 3D, while the methods employing curve fitting are not. There are, however, some drawbacks to the Salac and Lu method as well.

The primary issue stems from the fact that the Salac and Lu method is aware of the global topology of the interface. A problematic area, with a kink in the level set function close to $\phi = 0$, can be caused either by two bodies in close proximity or by a single body folding back onto itself. In the latter case, as illustrated in Figure 3.3, the Salac and Lu method falls back to the standard discretization, and the calculated curvature will be erroneous. This may seem like an edge case not worth considering, but simulations have shown that this often happens, e.g. when a falling droplet merges into a pool. When the curvature was calculated in such a case using the Salac and Lu method, the curvature error was sufficient to crash the simulation. For details of this particular case, the reader is referred to Section 5.1. Figure 3.3 is taken from the author's project report, [6, Section 5.2]. Another situation where this would often be the case is in tumor simulations like those performed by Macklin and Lowengrub, as seen in Figure 3.4, copied from [24, Figure 6].

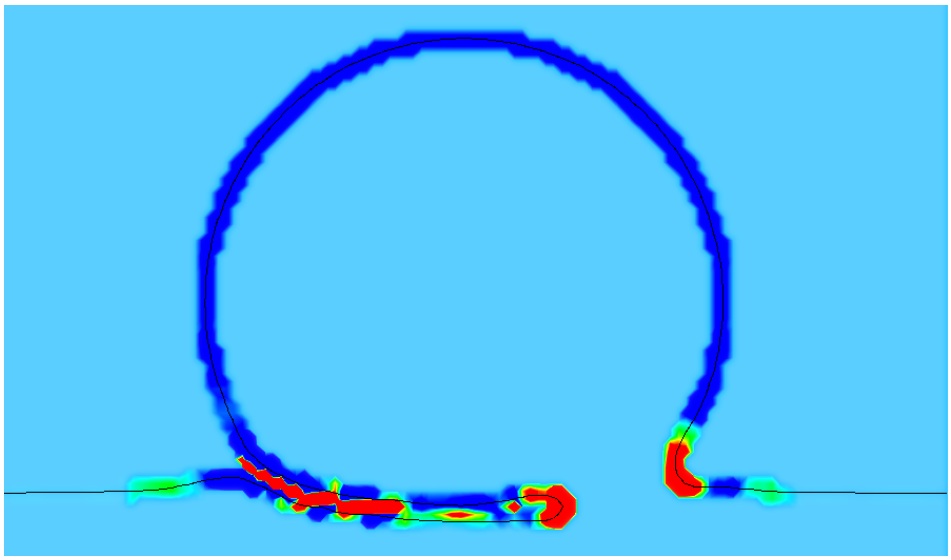


Figure 3.3: The curvature field plotted for the Salac and Lu method in the final frame before this simulation crashed. Note the red curvature field inside the air finger between the drop and the pool, which is incorrect. The color should be green in this area. (Figure best viewed in color)

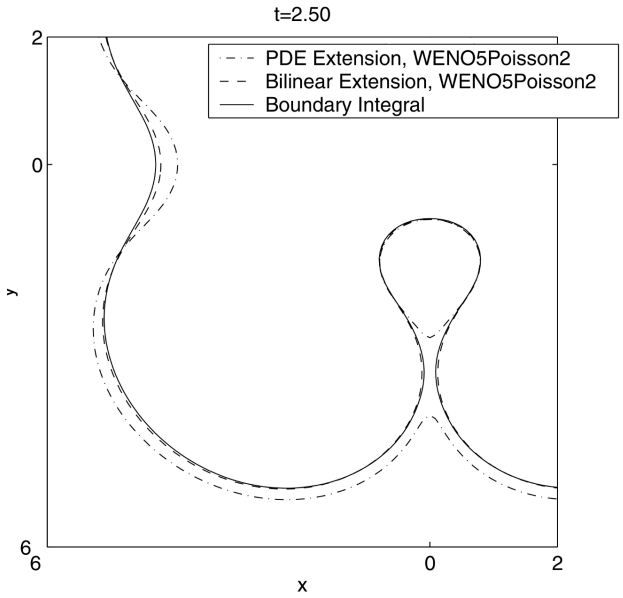


Figure 3.4: Another typical situation where the Salac and Lu method would fail. Figure copied from [24], showing the interface between a tumor and healty tissue. The dash-dotted line is the least accurate solution.

3.3 The idea of the LOLEX method

THE METHOD PRESENTED HERE tries to combine the best of the Salac and Lu approach with the best of the Macklin and Lowengrub method. As illustrated in the previous section, the Salac and Lu method is aware of the global topology of the interface, which is problematic in some cases. The Macklin and Lowengrub method does not have this problem, as its curve fitting considers only the local area, but as previously stated it does not extend easily to 3D. An obvious workaround to the "global awareness" is to make the Salac and Lu method consider only the local topology; say, a $10\times10\times10$ cube around the point where we calculate the curvature.

Since the Salac and Lu method relies on reinitialization to remove kinks, a potential problem with this approach is computational efficiency. Since reinitialization is somewhat computationally expensive, and we would do it on e.g. a $10\times10\times10$ grid for each point along the interface, the method would quickly become slow. To avoid problems with this, we want to use the ordinary method as much as possible, only resorting to the LOLEX method when we have to. This means using it when there are kinks in the level-set function close to the interface. To easily identify kinks, we use

the quality function $Q(\mathbf{x})$ which was introduced by Macklin and Lowengrub in [24]. It is defined as

$$Q(\mathbf{x}) = |1 - \nabla\phi(\mathbf{x})|, \tag{31}$$

i.e. the deviation of ϕ from a signed distance function. If $\max(Q(\mathbf{x}_{i,j,k})) > \eta$ for $\mathbf{x}_{i,j,k}$ in a 3×3×3 cube around the current grid point, we use the LOLEX method. A value of $\eta = 0.005$ is used here, and is seen to perform well. In addition to this, the current work uses the "narrow band" level-set method introduced in [2]. This means that quantities such as the curvature are only calculated in a narrow band around the zero level set, where they are needed. Together, these two conditions significantly restrict the use of the present method compared to the use of the normal method. In a typical falling drop simulation, the present method will only be used in a small percentage of the total number of time steps, and even then, it will typically not be used for all points along the interface. This means the computational (in)efficiency of the present method has a low impact on the total runtime of a simulation. Note, however, that the improvement afforded by the LOLEX method is still large: it only takes a few erroneous curvature values in one time step to completely ruin the results of a simulation.

An illustration of both the narrow band level-set method and the quality function is given in Figure 3.5. In this figure, the red lines indicate the zero level set, the light grey bands indicate the narrow band around $\phi = 0$, say where $|\phi| < \epsilon \approx \Delta x$, and the medium grey band indicates where $Q(\mathbf{x}) > \eta$. The intersection of the light and the medium grey bands indicate where the present method will be used; as expected, this is where the two bodies are close together.

Having briefly presented the idea behind the present method and the scope in which it will be used, we give here a step-by-step outline of it. See also Appendices A-F, where the full code is given, along with a flowchart illustrating the interdependence of the routines in the case of curvature calculations. 2D notation is used here, since it is easier to read than 3D notation. All steps are easily extendable to 3D.

↪ Loop over the computational domain using indices `i,j`

↪ If ($\mathbf{x}_{i,j}$ not close to interface) do nothing

↪ Else if ($Q(\mathbf{x}_{n,m}) \leq \eta \ \forall (n,m) \in [i-1, i+1] \times [j-1, j+1]$) use ordinary method

↪ Else use LOLEX method:

  ↪ Copy ϕ in a `[-1,ilmax+2]*[-1,jlmax+2]` square around `i,j` into the `lookphi` array.

  ↪ Identify the bodies present in the `[0,ilmax+1]*[0,jlmax+1]` square, store this in the `bodies` array.

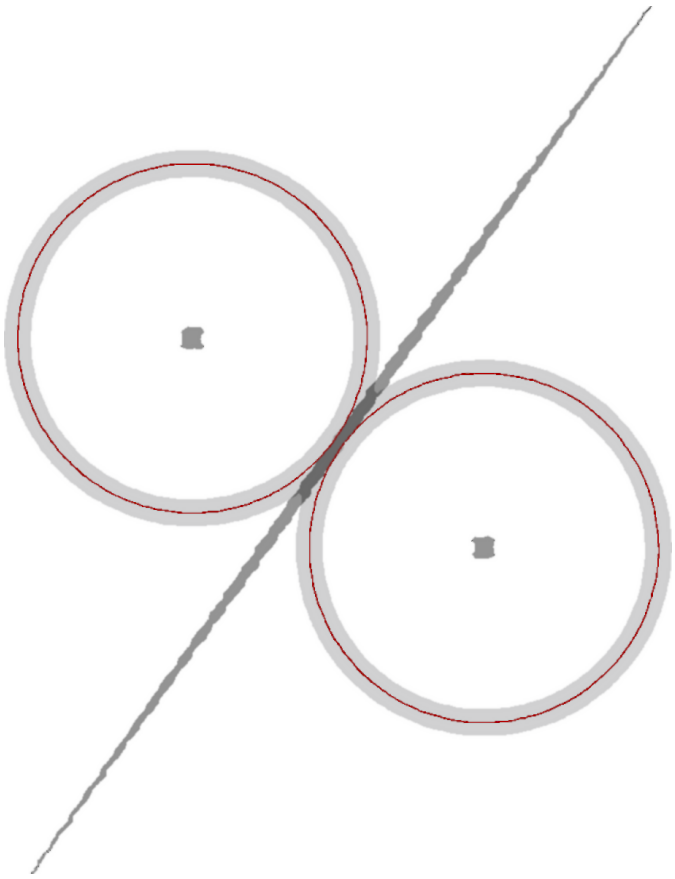


Figure 3.5: Illustration of the quality function and the narrow band level-set method. The red lines indicate $\phi = 0$, the light grey bands indicate the narrow band where $|\phi| < \epsilon$, and the medium grey band indicates where $Q(\mathbf{x}) > \eta$. The intersection of the light and the medium grey bands, the dark grey area, indicates where the LOLEX method will be used.

↪ For each body, extract the relevant part of the `lookphi` array into `locphi(:,:,bodyno)`. This array has 3 ghost cells on the boundary outside `ilmax*jlmax`; these are not used until the extrapolation further down. Extracting means

- copying `lookfi` for the internal points of *this* body
- copying `lookfi` for external points that are not next to more than one body
- explicitly reconstructing the signed distance for external points that are next to more than one body

  - setting a value of `2*dx` for all other points

↪ Once the `locphi` array has been filled for all bodies, the values are extrapolated into the ghost cells. The extrapolation is zeroth-order, as will be explained further down.

↪ The `locphi` array is then reinitialized for all bodies. This erases the problematic kink, as well as the value of `2*dx` which was set previously. Thus this value is unimportant, as long as it is > 0.

↪ Using these local ϕ's, the curvature and normal vectors can be calculated for each body. The multiple curvatures are then combined using a weighted average. For the normal vector, the closest body dictates which one to use.

The steps in this algorithm that warrant further comments are: identifying the bodies present, explicitly reconstructing the signed distance, extrapolating to the ghost cells, and reinitializing. These will be considered further in the next section and subsections. The quantities `ilmax`, `jlmax` and `klmax` represent the number of grid points, in the x, y and z directions respectively, of the *local* grid. The values of `ilmax`, `jlmax`, `klmax` are all set to 7 in the following. Their values are independent of the global grid size.

3.4 Details of the method

SOME STEPS OF THE ALGORITHM OUTLINED need further explanations. This is either because they are too technical to be fully described in the previous short outline, or because they have not been properly motivated yet. The steps that will be considered are identifying the bodies present (Section 3.4.1), explicitly reconstructing the signed distance (Section 3.4.2), extrapolating to the ghost cells (Section 3.4.3), and reinitializing (Section 3.4.4).

3.4.1 Identifying the bodies present

We start by explaining the procedure used to identify the bodies present. Here a recursive routine is used, which starts at a seed point in a body and iterates through the entire body, marking it as such in the `bodies` array. This routine is called `bodyscan` in the code and flowcharts, see Appendix D. The `bodies` array starts with a value of `unchecked`, and bodies found are marked using increasing integers, i.e. the first body found is marked as 1. The recursive subroutine will have marked the entire first body when its first call returns.

After the subroutine returns, we check if the present body is large enough to keep, or if it should be discarded. The reasoning behind discarding some bodies is twofold: a body occupying only a few cells in the local area will not be accurately represented, and will give erroneous values of the curvature and normal vectors. Furthermore, if it is small, it cannot be close to the present point, which is at the centre of the local area. Thus it is not important, since there must be another body close to the central point. If all bodies present are far away from the central point, we would not be using the LOLEX method in the first place. This assumes that no global bodies have only a few internal points, but such bodies are not properly resolved anyway. For such small bodies, we fall back to the standard method. For removed bodies, the points in the body are marked as `removed`. Points not inside a body are marked as `nobody`.

Several methods were tested for determining which bodies are to be discarded. The simplest method is to count the number of points in a body, and discard it if this number is less than some threshold. This method worked well, using a threshold of 25 cells for a 9×9 `bodies` array in 2D and 122 cells for a $9\times9\times9$ `bodies` array in 3D. These thresholds were chosen after tests with varying thresholds. They are large enough so that small bodies leading to erroneous values were discarded, but not so large as to remove bodies which are close to the central point (and thus important).

Another method tested was to use Gaussian weighting, instead of a constant weighting as described previously. To do this, a normalized Gaussian is precomputed and stored in an array of the same size as the `bodies` array. The standard deviation used was $1/4$ the width of the array. The idea behind this is that a body with 10 cells along the edge of the local domain is less important than a body with 10 cells stretching towards the centre of the domain. This was tested and found to work as well as a constant weighting in 2D, using a threshold of 0.11. No benefits of this approach were seen. In 3D, a Gaussian weighting could not be made to work, as it would discard some bodies that should have been kept, and keep some bodies that should have been discarded, no matter how the threshold was tweaked. The result of this is that a constant weighting was used for 3D. For 2D, both methods were equally good, and the Gaussian weighting was used. It is possible that some combination of standard deviation and threshold for the Gaussian could be made to work in 3D, but this was not investigated further, as the constant weighting was seen to work well.

For some points, it may happen that all bodies are removed from the local area. In this case, we check the distance from the central point in the local area to the closest interface. This is given by the value of the level-set function at this point. If this value is larger than $2\Delta x$, the curvature value and normal vector for this point are unimportant, so we set the curvature equal to zero and use the standard method for calculating normal vectors. This is consistent with the narrow-band method used here. If all bodies are removed, but the central point is close to an interface, we fall back to using the standard discretizations and print a warning message to the user about this. This has

happened a few times in tests, typically when a merging or splitting produces a satellite droplet which only has a few internal points, i.e. it is too small to be properly resolved by the grid. Such satellite droplets usually disappear quickly due to the mass loss effect in the level-set method, which is particularly strong for small bodies. Thus their effect on the rest of the flow field is (usually) negligible.

A final point to note about the routine given here is that even though a recursive subroutine is used, memory usage will not be problematic. This is because the routine operates on a small array whose size is independent of the grid size. In 3 dimensions and with the presently used size of the local area, the array `bodyscan` would have `11*11*11` = 1331 elements. This routine can maximally be called 1331 times, so this will not cause memory problems, although it could probably become too large to fit in the CPU cache thus slowing down the method. This has not been tested here, as the 3D calculations are only considered as a proof-of-concept, and have not been optimized for speed. In 2D the memory use is naturally much smaller.

3.4.2 Explicit reconstruction of the signed distance

For some points with $\phi > 0$, two or more bodies are within Δx of the point. This means that the value of ϕ is probably incorrect, since it has to be the distance to two separate bodies at the same time. We will call such points "dependent points". Because ϕ is likely incorrect for dependent points, we discard its value, and instead explicitly reconstruct the distance to the relevant interface. The procedure used is due to Adalsteinsson et al. [1], and it is is slightly modified, as described further down.

The reader is reminded that our objective is to find the distance to the relevant interface for a dependent point. This dependent point lies right next to two interfaces, but when we perform this calculation our interest is only one of these interfaces, so the other body is removed. Note that the signed distance is always positive for exterior points, so it is just the normal distance.

The procedure in [1] is as follows. The point (i, j) which we are considering is next to the interface of current interest. We ignore all other interfaces. Up to rotational symmetry, there are four possible cases. Adalsteinsson et. al. have five possible cases in their approach, which is more general, but applied in this context the fifth case never happens. This consitutes the aforementioned small modification. The cases are shown in Figure 3.6, copied from [1].

We examine the four relevant cases (**a** to **d**) more closely:

a The interface crosses one of the lines from (i, j) to its four neighbours. In this case, we use the distance to the interface along this line as our distance. This distance is given by

$$s = \Delta y + \phi(i, j-1) \tag{32}$$

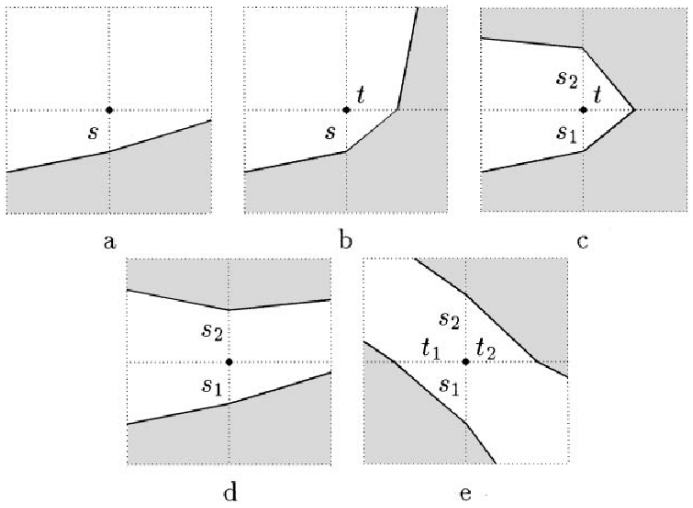


Figure 3.6: The five cases considered in the Adalsteinsson et al. method. Note that here, the body displayed in grey is just one of the bodies close to the central point. This implies that the fifth case is not relevant in this context: if the interfaces in **e** belong to two different bodies, we are only interested in one of them, so the situation is reduced to **b**. If they both belong to the same body, there cannot be two bodies next to the center point, so we have an independent point and would not be using this method in the first place. (Figure copied from [1])

where we have assumed that $(i, j-1)$ is the neighbour on the other side of the interface. Since this neighbour is an internal point, it has $\phi < 0$. The distance to the interface is the distance to the neighbouring grid point (Δy) minus the distance from that grid point to the interface, which gives this formula. It is best to use only the ϕ-value inside the body, since it is less likely to be distorted.

b The interface crosses two of the lines, and these two lines make out a corner of the 2×2 grid in Figure 3.6. In this case we use the shortest distance to the straight line between the two points of intersection. The distance d is given by the formula

$$\left(\frac{d}{s}\right)^2 + \left(\frac{d}{t}\right)^2 = 1. \tag{33}$$

As long as $s^2 + t^2 \neq 0$ this equation can be solved, and the positive solution is

$$d = \frac{s\,t}{\sqrt{s^2 + t^2}} \quad . \tag{34}$$

If we have $s^2 + t^2 = 0$, then $s = t = 0$, so it is obvious from Figure 3.6 (b) that the distance to the interface is $d = 0$.

c The interface crosses three lines. We construct the two straight lines between the points of intersection, and use the shortest distance to either of these two lines, given by

$$\left(\frac{d}{\min(s1, s2)}\right)^2 + \left(\frac{d}{t}\right)^2 = 1. \tag{35}$$

d The interface crosses two lines. These lines are on opposite sides of the point (i, j). In this case, we use the shortest of the two distances, so $d = \min(s1, s2)$.

As a side remark, we mention that case **a** is the most common, case **b** is less common, and cases **c** and **d** have not been observed during typical droplet simulations.

These formulae can be extended to three dimensions, where the possible cases are more numerous. In 3D, the central point has two additional neighbours. This means there are more variations in addition to the cases considered above.

3.4.3 Extrapolation

After the interior of the `locphi` array has been filled, the ghost cells must be filled before we can reinitialize the local ϕ. A first approach was to use linear extrapolation, which should work well since ϕ is a linear function in 1D. However, it turns out that this approach does not work. A fundamental property of the reinitialization equation (7) is that its characteristics originate at the interface $\phi = 0$. This is why the present method (and the Salac and Lu method) works - we only need a few cells directly next to the interface to have the correct value of ϕ, and reinitialization will fix the rest. It also means that reinitialization will never move the position of the interface, which is a desireable property in general.

The problem with linear extrapolation occurs when we extrapolate starting on the opposite side of the kink from the interface. In this case, the values of the local ϕ are tending towards 0 from above, which means that extrapolation can reintroduce the other body (which we removed in the first place). When this happens, reinitialization cannot fix the values beyond the kink, since it cannot move the interface reintroduced by extrapolation. This situation is shown in Figure 3.7(c) and (e), and thus we cannot use linear extrapolation.

A straightforward alternative is to use a zeroth-order extrapolation. This means simply copying the values along the edges into the ghost cells. It is obvious that this will never cross $\phi = 0$, so reinitialization is able to works as intended. An example of this is shown in Figure 3.7(d) and (f).

We describe further the parts of Figure 3.7. In (a), a zoom in on the global level set of a droplet touching a pool is shown. In (b), the local level set of the lower body (the pool) is shown after extraction and explicit reconstruction. Here, the values on the edges

are not set, indicated in grey. In (c), the same is shown using first-order extrapolation, and in (d) with zeroth-order extrapolation. In (e), the first-order extrapolated ϕ is shown reinitialized, and in (f) the zeroth-order extrapolated ϕ is shown reinitialized. Note in particular that in (e), a kink still exists after the entire procedure (green line), so the geometric quantities calculated could still be wrong if the derivatives cross the kink.

As Figure 3.7 is an illustration ignoring grid effects, a "real-life" example of a zeroth-order extrapolated local level set is given in Figure 3.8 (a). This figure also illustrates how the corner cells are handled. Assuming three ghost cells in both directions, there are nine ghost cells in one corner. Using zeroth-order extrapolation, these cells all get the value from the corresponding corner of the internal grid.

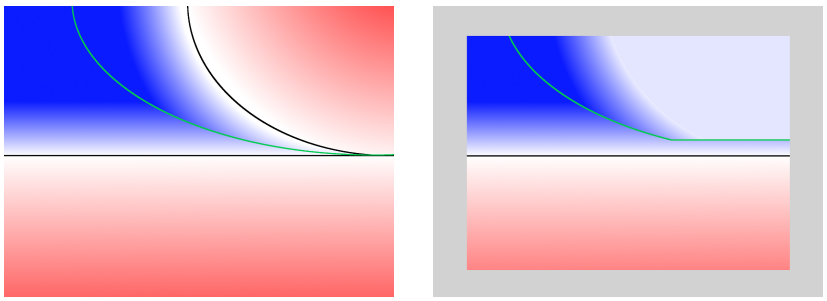
A possible improvement over the zeroth-order extrapolation used here would be to use values from the global ϕ where it is possible, i.e. inside the body considered and in outside regions where the quality function is below the threshold η. Where this is not possible, one would fall back to the zeroth-order extrapolation. This would probably increase the accuracy of the curvature and normal vectors, but has not been considered in detail here.

3.4.4 Reinitialization

When the extracted local level-set has been extrapolated, it must then be reinitialized before the geometric quantities are calculated. This is essential in order to have good values of the level-set function outside the interface. The entire LOLEX method hinges on the fact that reinitialization restores the local level-set to a signed distance function, so that ordinary discretizations will not give errors. This is not entirely straightforward, however.

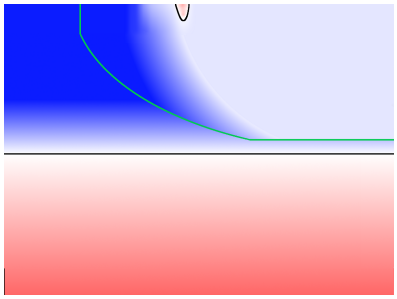
When reinitializing, we require at least some points on either side of the interface with decent ϕ-values, i.e. ϕ being the signed distance to the interface. In addition to this, we need to know the smeared sign function, and most crucially, the normal vectors at the interface. Thus we are faced with a bootstrapping problem: accurate normal vectors are required in order to accurately calculate the normal vectors. This is only a problem when the global interfaces are very close; when there is a moderate distance (i.e. more than one grid point between the interfaces), the normal vectors can be calculated at the interface using the local level-set.

The solution to this conundrum is to exploit the redundant information which is stored in the level set function. To illustrate this redundancy, imagine that you are walking along a normal vector to the interface. At each grid point you pass, you are told the current distance to the interface. As long as you do not pass any kinks, this information is redundant: using the value at the first grid point you pass, you can calculate the value at the next grid point, and the one after that, given that you know the grid spacing.

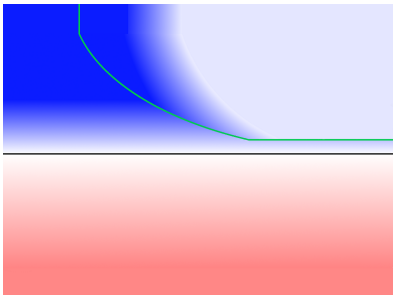


(a) Zoom in on global level set

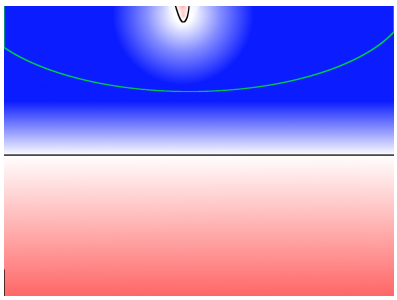
(b) Extracted local level set



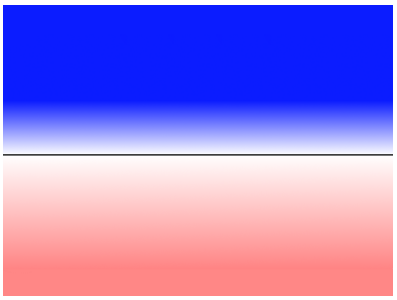
(c) First order extrapolated



(d) Zeroth order extrapolated



(e) First order, reinitialized



(f) Zeroth order, reinitialized

Figure 3.7: Extraction, extrapolation and reinitialization of the local level set is shown, for the lower body in Figure (a). Red indicates a negative value, blue a positive value, and white indicates zero. The green lines indicate kinks in the level set function, and the black lines are the zero level sets. A detailed explanation of the figures is given in Section 3.4.3. (Figure best viewed in color.)

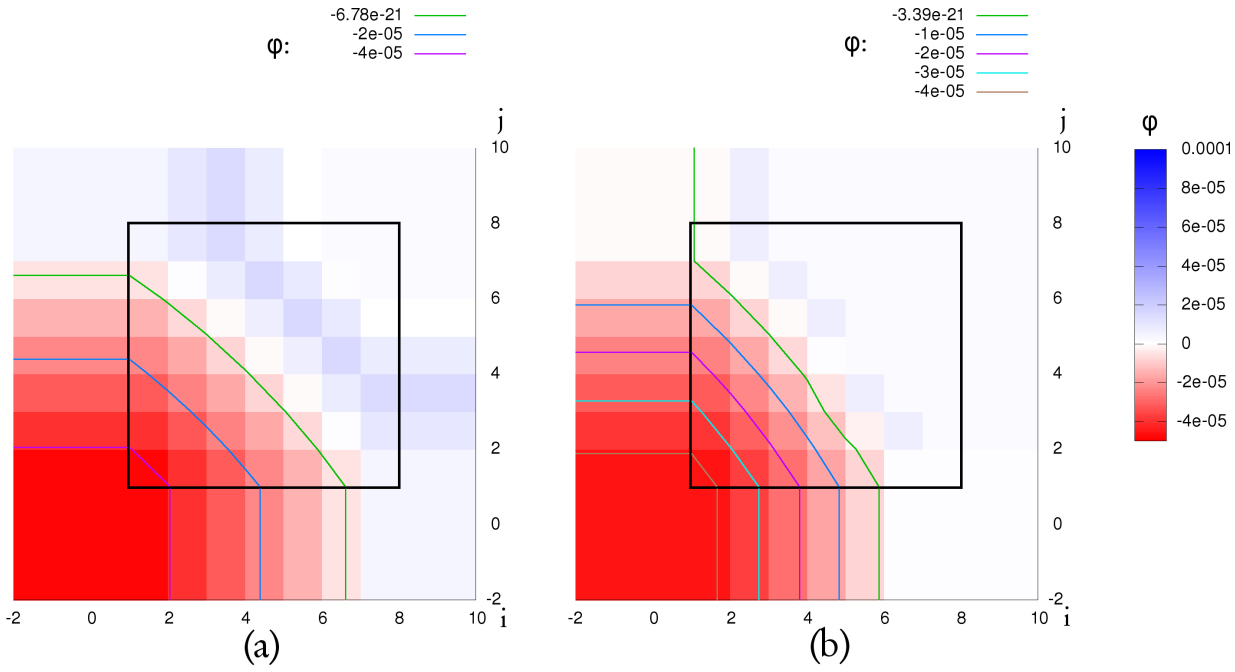


Figure 3.8: Zeroth-order extrapolated local level set, part of a circle. In this case, there are 7 internal cells in either direction, and three ghost cells outside this. The black square frames the internal cells. In (a), the distance from the interface to the other body is $2.8\Delta x$. In (b), the distance is $1.1\Delta x$. It is seen that the zero level set in (b) is slightly deformed.

What this means for the present case is that we have information inside the current body that we can use. Most importantly, we can calculate the normal vectors without problem for internal points. This means that we can reinitialize a level set different from $\phi = 0$, e.g. $\phi = -0.8\Delta x$, and get essentially the correct ϕ afterwards. We are not guaranteed to get exactly the correct ϕ, but seeing as we cannot obtain the correct ϕ anyway, we will settle for a good approximation. An illustration of this in 1D is shown in Figure 3.9, where the extracted local level-set function ϕ is shown in red. Note that e.g. the value of $\nabla\phi$ at the grid point 0^2, shown with a blue line, is much closer to 1 than the value at the grid point 0^1, shown with an orange line. When the lower level set is used, we momentarily move the interface further to the left in this figure, so the grid point 0^2 is closest to the interface. It is obvious that we have a better chance of restoring a signed distance function with the correct location of the interface if we reinitialize from the lower level set. [5]

[5]The values of $\nabla\phi$ are calculated here using central differences, but the error will still persist to some degree when e.g. a more accurate WENO discretization is used instead.

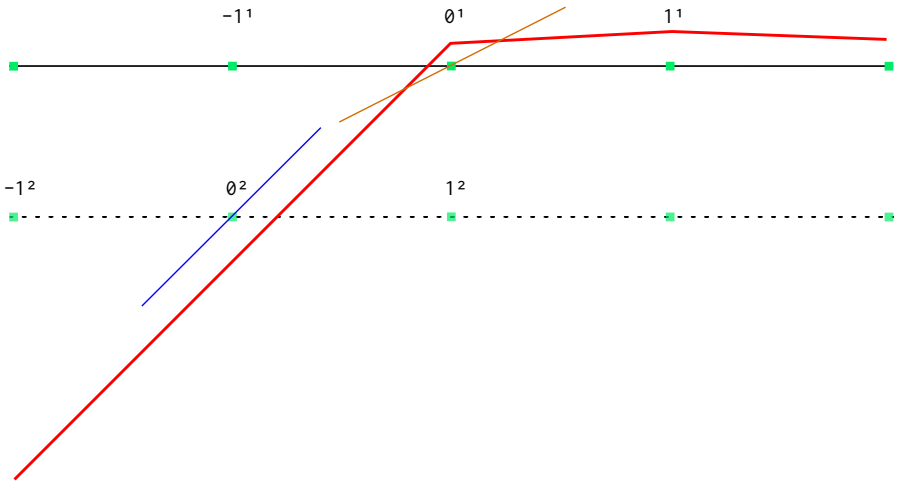
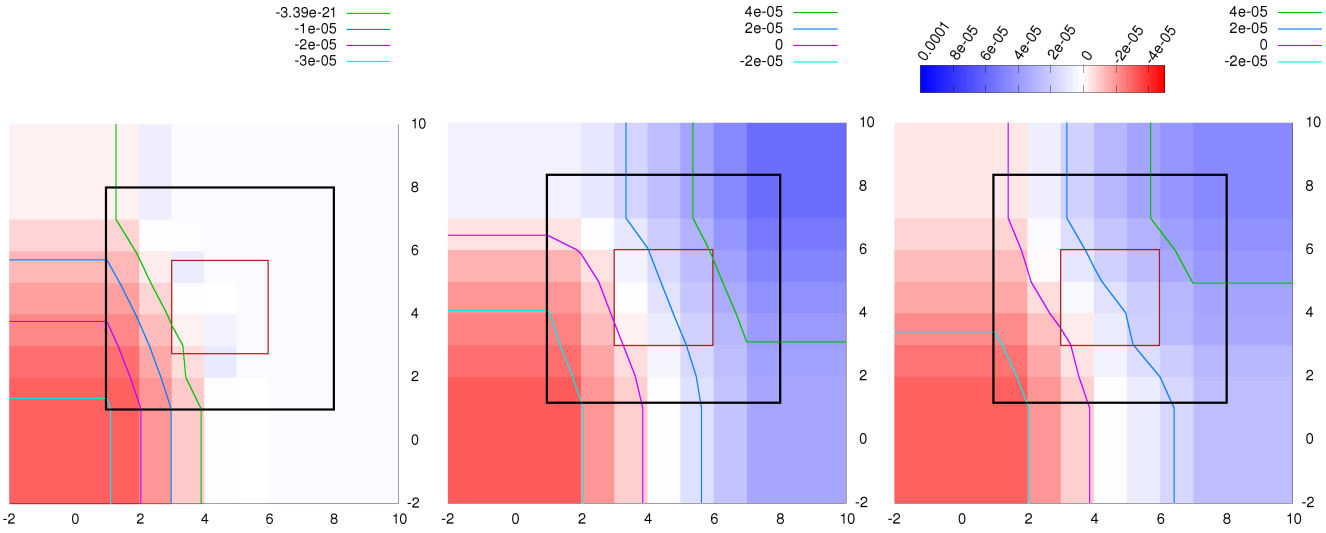


Figure 3.9: Why we reinitialize from a lower level set: At the lower level set, indicated by the dotted line, values of e.g. $\nabla\phi$ are more accurate at the grid point which is closest to the red line than for the zero level set. The red line indicates the local level-set function ϕ.

The value of $-0.8\Delta x$ used here gives the most accurate results. If the value is too close to zero, the benefit of reinitializing from a lower level set is reduced. However, if the value is too large, we risk having this lower level set too close to the edges of the local domain, and we increase the potential error caused by reinitializing from a different level than zero.

Another problem solved by this is the fact that the values directly outside the zero level set may be incorrect in some cases. In particular, this happens when an outside grid cell is not flagged as dependent, but its value of ϕ still deviates from that dictated by a signed distance function. Tests have shown that this sometimes occurs, and that it distorts the zero level set somewhat. An example of this is shown in Figure 3.8 (b). This is a similar case to that in Figure 3.8 (a), but the distance to the other body (not shown) has been reduced from 2.8 Δx to 1.1 Δx.

Reinitializing from a different level may sound somewhat complicated to do, but the implicit formulation springs to the rescue again. To reinitialize from a lower level set, we simply add a positive constant to ϕ at every local grid point, call the reinitialization routine on this ϕ, and then subtract the same constant from the reinitialized ϕ. The effect of this is illustrated in Figure 3.10, which is an extreme case. Here, reinitialization of two very close bodies has distorted the global level-set function close to and outside the interfaces. The reinitialized local level-set function is also wrong, but the one which is reinitialized from a lower level set gives a more accurate representation of the interface. This will, in turn, give a significantly more accurate curvature. A plot



(a) Before (b) From $\phi = -\Delta x$ (c) From $\phi = 0$

Figure 3.10: The LOLEX method on a global level set which is distorted due to reinitialization of very close bodies. (a) Local ϕ before reinitialization. (b) Local ϕ reinitialized from $\phi = -0.8\Delta x$. (c) Local ϕ reinitialized from $\phi = 0.0$. The black square indicates the boundary to the ghost cells, and the red square indicates the 3×3 central points that are used in the final curvature calculation.

of the curvature calculated with and without this improvement is shown in the next chapter, in Figure 4.4, where it is seen that improvement is large, particularly when two interfaces are close.

In order to reinitialize the local level-set, some modifications were made to the existing reinitialization routines. The most obvious modification was to make them work on the smaller grid containing the local level set. In addition to this, the normal vectors and the smeared sign function are typically stored in global arrays. This is because they are used for several things in addition to reinitialization, e.g. when advecting the interfaces. In the local level-set reinitialization, this is unnecessary since the values are not to be reused in other parts of the code. For this reason, the code calculating the smeared sign function and the normal vectors was inlined in the routine that calculates the right-hand-side of the reinitialization PDE.

The fact that specific routines had to be written for local level-set reinitialization enabled some simplifications in this code. The ordinary reinitialization uses routines that are shared with other PDE solvers, and thus they contain cases (e.g. in the choice of Runge-Kutta method) that are irrelevant for local level-set reinitialization. In the routines for local level-set reinitialization, these cases are removed. For reinitializing

Table 1: Parameters used in the LOLEX method, along with values used and sensible ranges. For the curvature weighting exponent, see Section 4.1.1.

Parameter	Value	Sensible range
Local grid size	7	5–11
η	0.005	0.01–0.001
2D discard threshold	30.9 %	30–32 %
3D discard threshold	16.7 %	16–17 %
Reinit. level set	$-0.8\Delta x$	-1.0 to $-0.5\Delta x$
Curvature weighting exponent	5	1, 4–7

the local level-set, the SSP-RK 2()4 method is used, along with the WENO-5 method. For details of these see Section 2.5.

3.4.5 Parameters used presently

In the LOLEX method as presented in the previous sections, there are a number of parameters that can be varied. An overview of these is given here, along with the values used presently, and sensible ranges, in Table 1.

To conclude this section, the entire main routine of the LOLEX method is included as well. It can be seen in Appendices A-F. This code has been given some cosmetic alterations to make it more readable, but is otherwise the same as that used to produce the results shown here.

3.5 Summary

In this chapter the presently used LOLEX method has been described in detail. The method is used for grid points where the level-set function deviates from being a signed distance function, where it extracts one or more local level sets, removes any kinks in these by use of reinitialization, and finally uses these local level sets to calculate the curvature and normal vectors. If there are multiple local level sets, the curvature values are averaged, and the normal vector corresponding to the closest interface is used.

The method is motivated in that it is more general than the previous method by Salac and Lu, handling bodies which fold back onto themselves, and it extends more easily to 3D than the previous methods by Macklin and Lowengrub and by Lervåg, which use curve fitting schemes. A description of the LOLEX method has been given

in sufficient detail for anyone wanting to implement it. The parameters of the method are given in Table 1. Results, both for static and dynamic simulations, are given in the next chapters.

§4 LOLEX calculations on static cases

Contents

> However beautiful the strategy, you should occasionally look at the results
>
> Winston Churchill

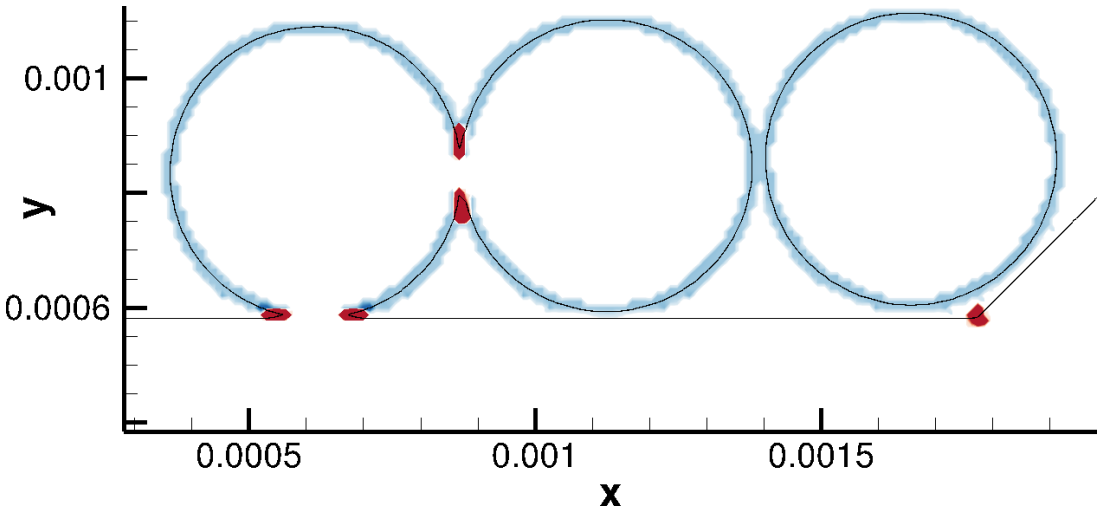
USING THE LOLEX METHOD described in the previous chapter, we present here some calculations of the curvature and normal vectors for static cases with no flow, i.e. purely geometric results. These results are compared with the results from ordinary discretizations, analytical results, and results from the Salac and Lu method [40] and the Lervåg method [21],[22] which is a variant of Macklin and Lowengrub's method [25].

4.1 LOLEX curvature calculations

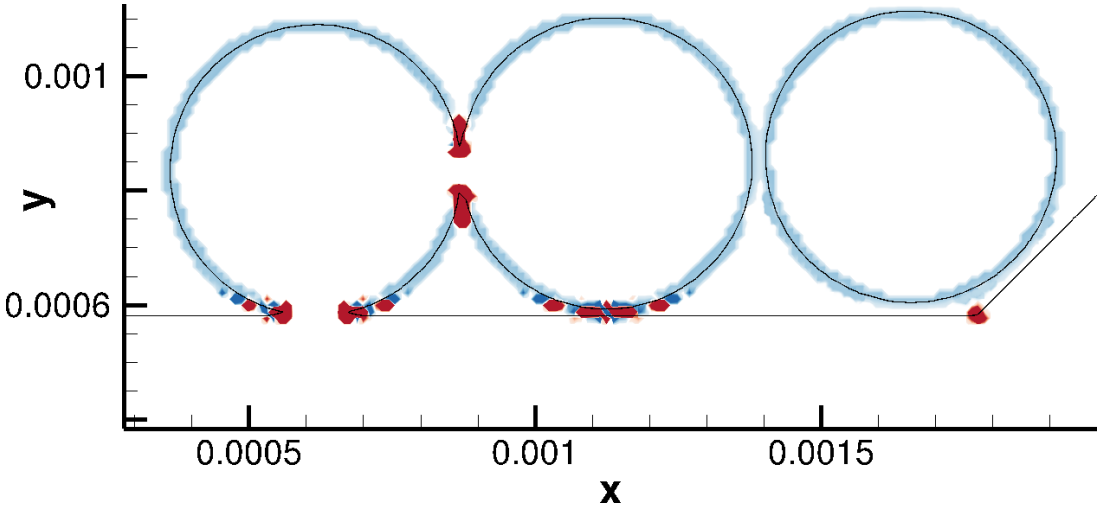
THE RESULTS OF THE LOLEX METHOD on geometric test cases are presented here. Initial tests were made to ensure the LOLEX method reproduces the results of the Salac and Lu method, which has been tested previously in the author's project report[6], and to see whether it solves the problem it was designed to solve.

The first test case considered was designed to replicate typical situations that occur in simulations of droplet collisions, and will be referred to here as the curvature test case. The results for this test case are shown in Figure 4.1 for the LOLEX method, the Salac and Lu method, and the standard method. In this figure, the interfaces are shown as black lines, and the color indicates the curvature. The background curvature of 0 is indicated in white, blue indicates a negative curvature and red indicates a positive curvature.

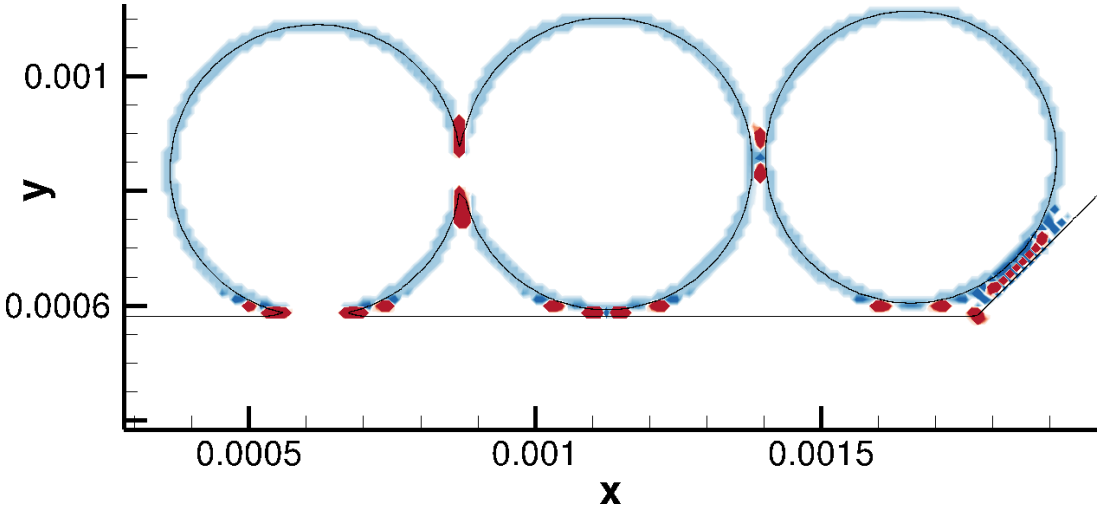
As is seen in this figure, the LOLEX method outperforms both the Salac and Lu method and particularly the standard method. The difference between the LOLEX method and the Salac and Lu method is mainly seen where the second circle from the left comes close to the lower flat interface. The flat interface, as well as the first and the



(a) LOLEX method



(b) Salac and Lu method



(c) Standard method

Figure 4.1: Comparison of curvature calculation methods for a geometric test case.

second circle from the left, have been joined into one body in order to illustrate the problem with the Salac and Lu method. Note that the red dots seen for the LOLEX method are correct, in these places the curvature is very high.

Some tests were also made to determine how much more computationally expensive the LOLEX method is, compared to the standard discretization. For this purpose, the curvature test case with static interface configurations and no flow was used on a 200×200 grid. The LOLEX method was activated in 603 of the cells. The time used for one curvature calculation (i.e. setting the curvature field in the entire domain) was determined. The code was compiled using the `gfortran` compiler with identical optimizing flags for both methods. The standard method used $3.00 \cdot 10^{-3}$ s. The LOLEX method used $3.48 \cdot 10^{-1}$ s. Thus the LOLEX method has a two orders of magnitude longer runtime than the standard method. This is as expected, since the LOLEX method uses reinitialization, which is much more computationally expensive than simply calculating central differences.

As mentioned previously, this slowdown will not have a large impact on the total simulation runtime. One reason for this is that other routines are equally (or even more) time consuming. To illustrate this, the curvature test case was run for 10 time steps using the LOLEX method. In this case, the flow is zero everywhere, but the Navier-Stokes equations are still solved, as is the Poisson equation for the pressure, and reinitialization is performed every 3 time steps. These settings are typical of a two-phase flow simulation. In this case, the total runtime was 152 s, while the LOLEX method only used 16.4 s in total (41 calls to the curvature calculation).

While a 10% runtime increase may seem significant, the negative impact of the LOLEX method is further reduced in real simulations. In simulations, the method will typically be active for only a fraction of the total time steps, around the time when interfaces merge or split. If it is active in e.g. 10% of all time steps, which is more than in simulations reported in the following, this means the negative impact on the simulation runtime is just 1%. All in all, this means that the LOLEX method has a negligible impact on the total runtime of e.g. a droplet collision simulation.

4.1.1 Curvature averaging methods

After extracting the local level sets using the LOLEX method, we end up with two values of the curvature in the case with two bodies present in the local ϕ. Since we cannot have a multiply-defined curvature in the present codes, an average has to be used.

The first method of averaging considered is a geometrically weighted average. After the construction of two local ϕs, we know the distance to both interfaces. Denote the curvature of the i^{th} interface by κ_i, and the distance from the i^{th} interface to the

current point by d_i. The geometrically weighted average is then given by

$$\kappa = \frac{d_1\kappa_2 + d_2\kappa_1}{d_1 + d_2}. \tag{36}$$

An alternative to this is the harmonic weighted average. With the same notation as before, this average is given by

$$\kappa = \frac{d_1 + d_2}{d_1/\kappa_2 + d_2/\kappa_1}. \tag{37}$$

The potential benefit of using a harmonic weighted average over a geometrically weighted average is that the harmonic average gives less importance to larger values. In the present case, large values are often incorrect, since the errors produced by the ordinary method are $\mathcal{O}(1/\Delta x)$ [24]. When considering a circle, the grid resolution would typically be set so that the radius is $> 10\Delta x$, in order to avoid errors. Thus a typical erroneous value is at least an order of magnitude larger than a typical correct value, and so it could make sense to use a harmonic average.

Further averages can also be considered. One alternative is to use a plain average, i.e.

$$\kappa = \frac{\kappa_1 + \kappa_2}{2}. \tag{38}$$

Another option is to amend the geometric average by squaring d_i in both the nominator and denominator,

$$\kappa = \frac{d_1^2\kappa_2 + d_2^2\kappa_1}{d_1^2 + d_2^2}. \tag{39}$$

Taking this further, d_i can be raised to the power 3, 4 etc. The higher the power, the more weight is given to the curvature corresponding to the closest interface. We will call this the squared geometric weighting, *et cetera*, even though strictly speaking it is only geometric when the exponent is 1. If we take the limit

$$\kappa = \lim_{n\to\infty} \frac{d_1^n\kappa_2 + d_2^n\kappa_1}{d_1^n + d_2^n}, \tag{40}$$

we end up selecting just the curvature corresponding to the closest interface, i.e.

$$\kappa = \begin{cases} \kappa_1 & \text{if } d_1 < d_2 \\ \kappa_2 & \text{if } d_1 > d_2 \\ \frac{1}{2}(\kappa_1 + \kappa_2) & \text{if } d_1 = d_2 \end{cases}. \tag{41}$$

We will call this the supremum average. To a physicist, the statement above is immediately true. Suppose then, for the sake of argument, that you are a mathematician.

PROOF. We want to show that the limit in Equation (40) is Equation (41). Suppose that $d_1 > d_2$, the argument is identical in the opposite case and for equality. We can divide by d_1 in the nominator and denominator in Equation (40), and defining $a = d_2/d_1 < 1$ we are left with

$$\kappa = \lim_{n\to\infty} \frac{1^n\kappa_2 + a^n\kappa_1}{1^n + a^n} = \frac{1\kappa_2 + 0\kappa_1}{1+0} = \kappa_2 \tag{42}$$

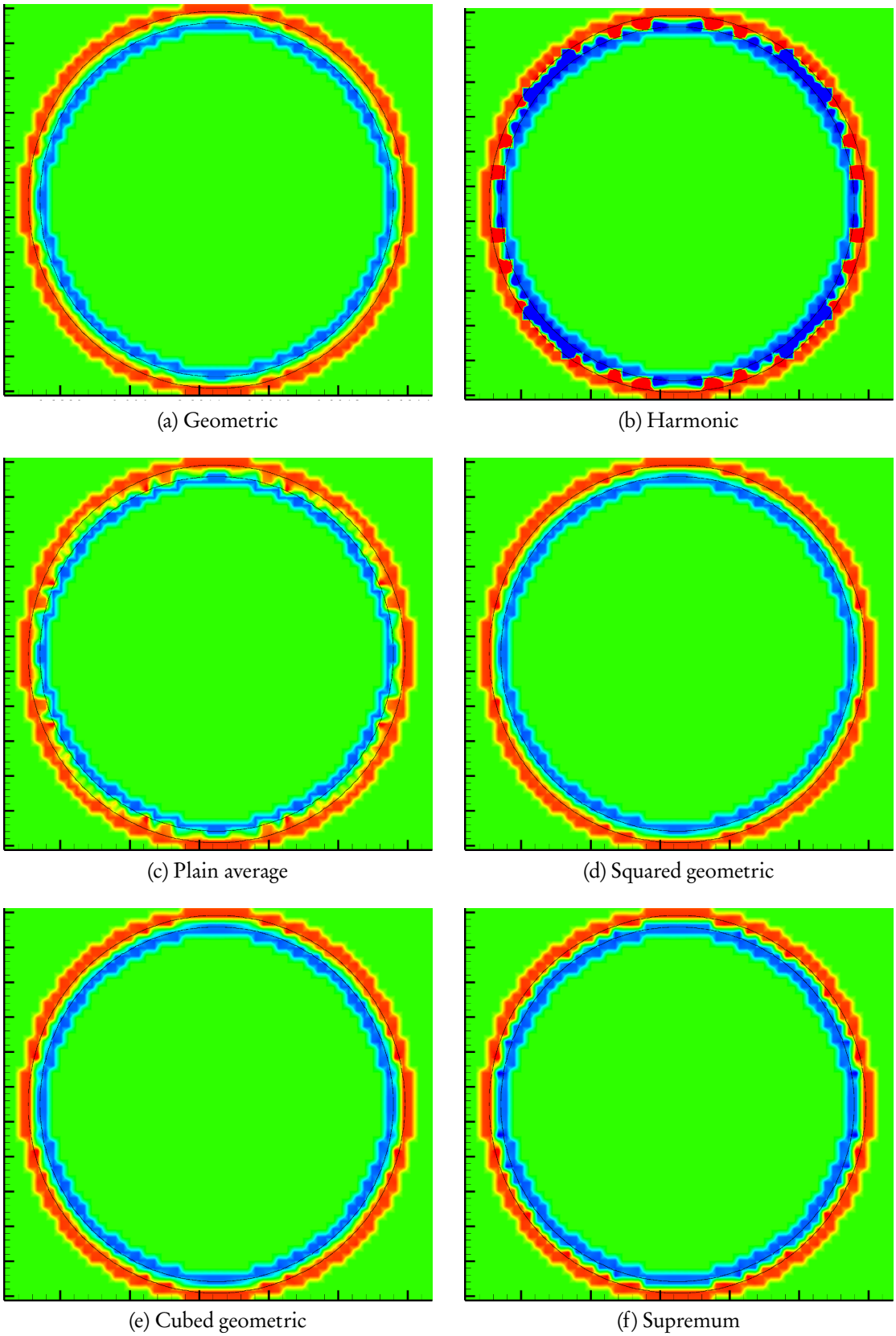
■

4.1.2 Comparison of curvature averages

To test the various averages in practice, all these methods were used to calculate the curvature field for a geometric test case. In this test case, a thin ring of fluid 1 is constructed inside fluid 2. The width of the thin ring is varied in order to test the method at different interface separations. This test case was chosen because it reveals anisotropies and grid effects easily. The curvature of a circle is simply a constant, the inverse of the radius, so the correct curvature is known. For this test, the width of the ring was set to $1.5\Delta x$. It is noted that in simulations of droplets merging with a pool, a thin film such as this is often formed. The LOLEX method is activated if the width of this thin film is less than roughly $5\Delta x$, and the interfaces start to merge when the distance is around $1.4\Delta x$. The precise values depend on the position of the interface relative to the grid. This means that $1.5\Delta x$ is roughly the most demanding case occuring in simulations.

The results of the test are shown in Figure 4.2. In this figure, blue represents a negative and red a positive curvature, while green means zero curvature. With the definition used here, the curvature of the inner interface is negative, and that of the outer interface is positive. Note that the curvature should be constant along an interface for this case. It is seen that the harmonic average performs worst of all, with large curvatures oscillations along both interfaces. It is also seen that the results are fairly isotropic, as they should be, but there are some grid effects. This is unavoidable, as we do not have enough grid points to accurately store the curvature. It is somewhat difficult to tell which average performs best, so line plots of the curvature along the interface are given as well.

In the line plots, the interpolated curvature values are shown for 16 points equidistantly spaced on a $4\Delta x$ interval along the interface. These plots are shown in Figure 4.3 and give good insights into the performance of the various averages. First of all, it is confirmed that the harmonic average is the worst, while the plain average is the next-worst. Second, the supremum average is the least noisy, but it overestimates the correct curvature value in all but one point (note that the values are all negative). Overall, the supremum or the geometric average are the best candidates, lying on either side of the correct value. This fact, along with the tendency of improvement when



(a) Geometric (b) Harmonic

(c) Plain average (d) Squared geometric

(e) Cubed geometric (f) Supremum

Figure 4.2: Comparison of different ways to average the curvature

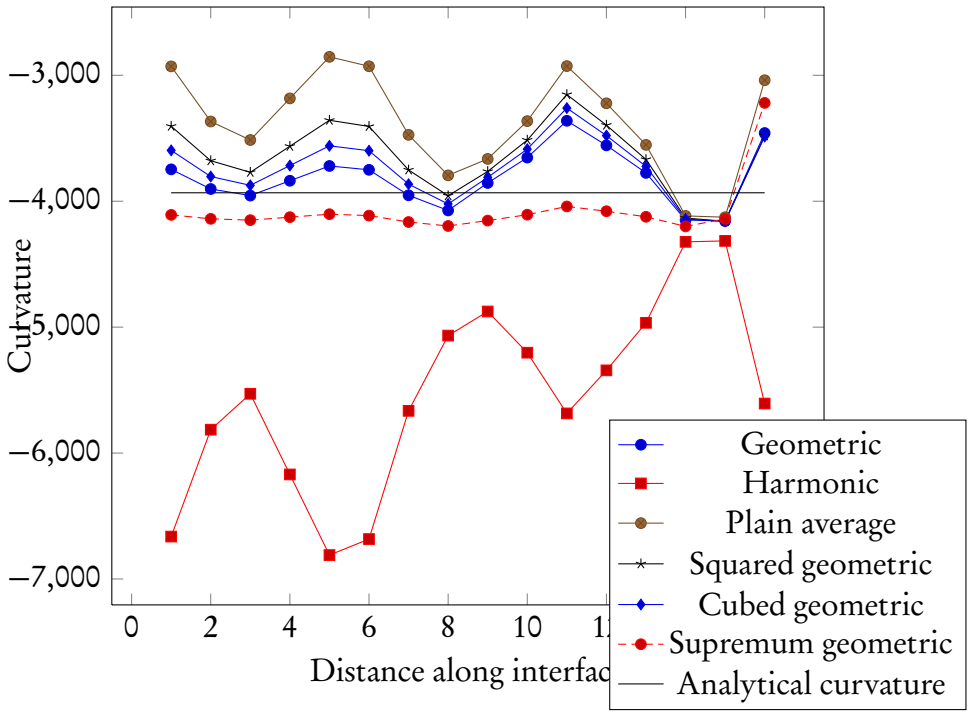


Figure 4.3: Lineplots of curvature along the interface for different averaging methods.

going from squared to cubed, prompted a search for some exponent $n > 3$ which has low variation and gives a good average value.

Testing with exponents $n = 5, 8, 14, 20$ gave the results shown in Figure 4.4. From this it seems that $n = 5$ is a good choice, slightly better than the geometric average. To be more precise, the L^1 norm of the error compared to the analytical curvature is shown in Table 2 for the data points used in this figure. $n = 5$ has the lowest error using this norm, which is a sensible one to use here, since it sums up the error in all points. Using e.g. the L^2 norm would give a larger penalty to the largest errors, and a lower penalty to those who give the wrong average value.

Furthermore, it is reassuring that as $n \to \infty$, the results converge towards the supremum average. It is also seen that at the final three points, all exponents give almost the same result. This is because the interpolation routine used when plotting has a problem at this point. The interpolation routine uses the marching squares algorithm, which has a well-known problem in some cases [31]. This effect dominates in the final three points, particularly the second-to-last one.

This does not, however, affect the simulations. In simulations, only the curvature values at grid points are used, as described in [41]. The interpolation is only used here for plotting purposes, in order to compare the averaging methods. Thus no

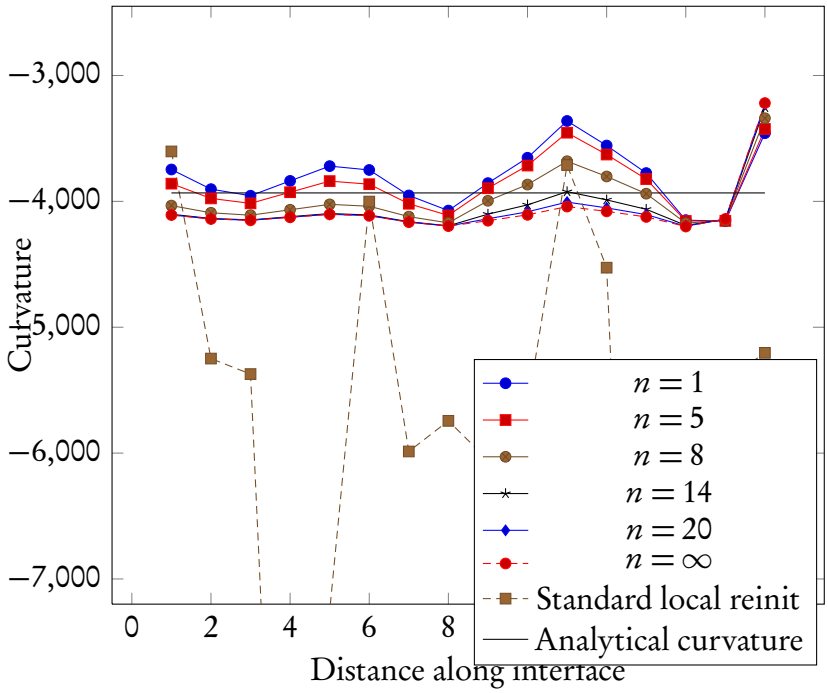


Figure 4.4: Lineplots of curvature along the interface for different exponent n in the geometric weighting.

Table 2: L^1 norm of the error in curvature calculated with different averages relative to the analytical curvature of the circle.

Averaging	L^1 Error
$n = 1$	3269
$n = 5$	2735
$n = 8$	2759
$n = 14$	3210
$n = 20$	3533
$n = \infty$	3668

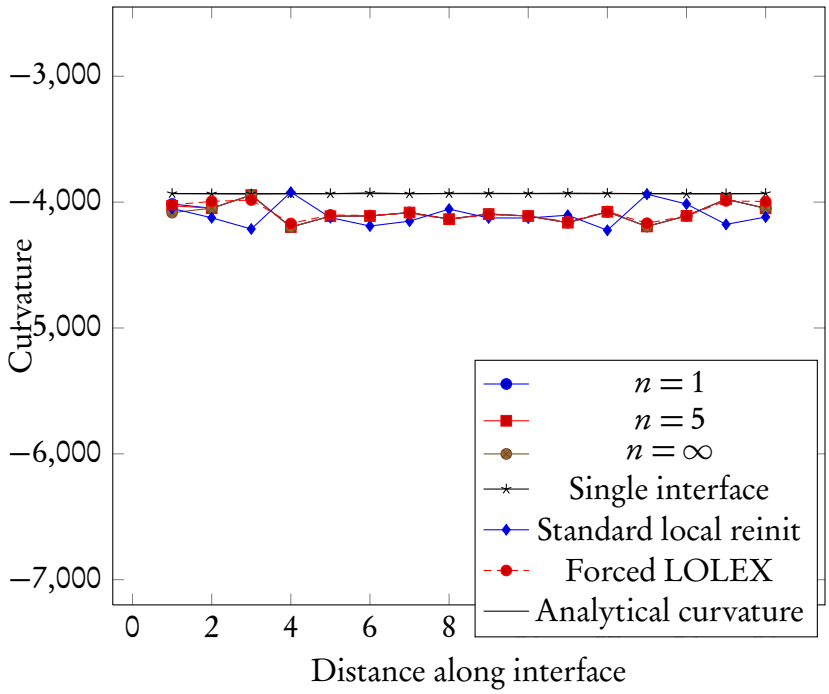


Figure 4.5: Lineplots of curvature along the interface using different averages, when the distance between interfaces is $2.2\Delta x$. Here it is seen that the difference between the averages is much smaller than for the previous line plots.

interpolation errors affect the physical results.

Also shown in Figure 4.4 is the curvature calculated when reinitializing the local level set from $\phi = 0$ instead of $\phi = -0.8\Delta x$, as discussed in Section 3.4.4. It is seen in this figure that for close interface separations, the improvement yielded by reinitializing from a lower level set is very large.

It should be noted that the curvature averaging method is only important when interfaces are very close, such as here, where the separation is $1.5\Delta x$. When the interfaces are further apart, the averages all give essentially the same result. This is shown in Figure 4.5, where the geometric ($n = 1$), $n = 5$ and supremum ($n = \infty$) averages are compared for the same case as previously, but where the interface separation is increased to $2.4\Delta x$.

Also shown in this figure (black stars) is the curvature calculated using the standard method when the interface separation is large ($10\Delta x$), and the curvature calculated when reinitializing the local level set from $\phi = 0$ instead of $\phi = -0.8\Delta x$ (blue diamonds). It is seen that for large interface separations, where there are no kinks close to the interface, the standard method is very accurate. The LOLEX method, however, tends to overestimate the curvature. One could perhaps suspect that reinitializing from

a lower level-set was the culprit, but as is seen in the figure, reinitializing the local level set from $\phi = 0$ does not alter this situation. Comparing this to Figure 4.4, it is seen that reinitializing from a lower level set is most important when the interfaces are very close, which makes sense, as a larger interface separation means that the values of ϕ directly outside the interface are much closer to being a signed distance function.

It turns out that extrapolation and reinitialization of the local level-set is the source of this overestimation. To test this, a large interface separation ($10\Delta x$) was used, where the standard method performs well, but in this case use of the LOLEX method was forced for all grid points. In addition to this, the local ϕ was limited to a maximum value of $1.2\Delta x$ outside the interface, replicating a typical local ϕ for close bodies after extraction and reconstruction. This ϕ was extrapolated and reinitialized, and the curvature was then calculated. The result is shown in Figure 4.5 (red dots), where the same overestimation is seen. Not limiting the local ϕ to a maximum value gives the same result. This indicates that extrapolation and reinitialization of the local ϕ produces a constant error of 2-4% in the curvature.

With this result at hand, it is no longer clear that using the $n = 5$ average is the best option, as indicated above. It is possible that using the supremum average is more correct; however, the constant error was discovered after most simulations had been completed, so this has not been investigated in detail here due to time constraints. It is probable that if the constant error is reduced or removed, the supremum average will outperform the other averages.

To verify that extrapolation and reinitialization was the cause of this error, rather than some subtle bug, the same test with a large interface separation and forced LOLEX method was considered again. The local level-set function was plotted in the radial direction before limiting (i.e. the correct level-set function), after limiting the maximum value to $1.2\Delta x$ and after reinitializing the limited level-set function. This is shown in Figure 4.6.

As is seen in this figure, extrapolation and reinitialization cannot reconstruct exactly the correct ϕ. The error is $\pm 1\%$ at the edges, and between 0.5 % and 0.1 % in the middle. It is important to note that the error is not constant, so it will affect derivatives of ϕ. In a discretization like that for the curvature (Equation (5)), this error will compound, so it is reasonable that this induces a curvature error of 2–4%.

A possible way of reducing this constant error is to improve the extrapolation method, copying from the local level-set for internal points and external points with good values according to the quality function, as indicated above. Another possibility is to use a more accurate reinitialization procedure, e.g. the HCR-2 method due to Hartmann et al. [14]. This has not been done here, but will perhaps be considered in future work.

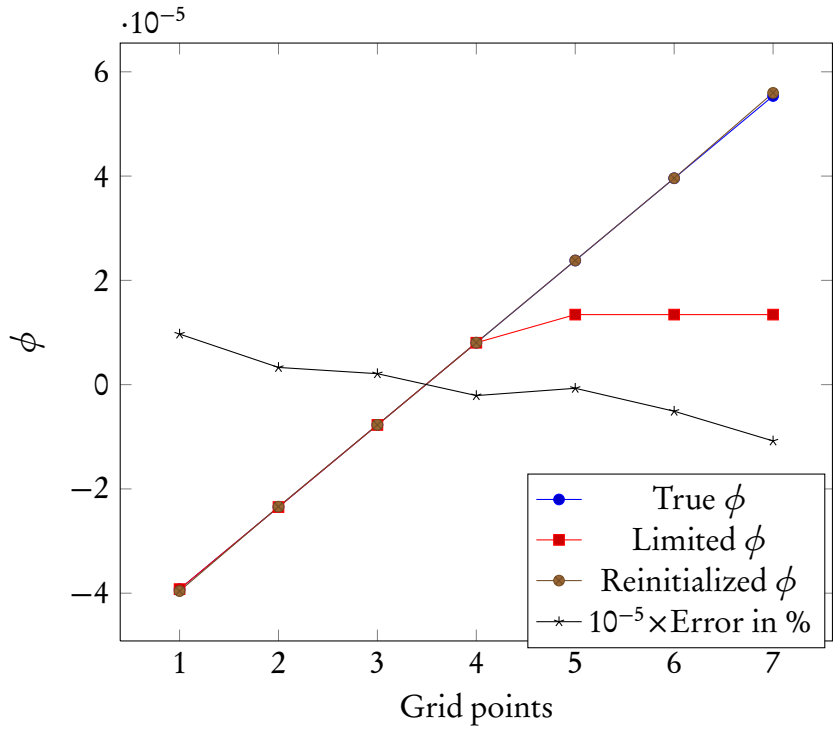


Figure 4.6: Plots of the level-set function in radial direction for a circle. The blue dots show the true level-set function, the red squares show a limited level-set function emulating an extracted local level-set in the LOLEX method, and the brown dots show the resulting level-set function after reinitialization. The black stars show the error, i.e. (Trueϕ − Reinitializedϕ)/Trueϕ, as percentages multiplied by 10^{-5}. This means that the numbers on the y-axis also indicate the error in %; the error is 1 % in the first grid point and so forth.

4.2 A problem with thin bodies

A PROBLEM WITH THIN BODIES and the level-set method was discussed in the author's project report [6]. In essence, what happens in this case is that a thin body with only one interior point cannot be stored correctly. Since we demand that this interior point stores the signed distance to two interfaces at the same time, we are bound to get an error if the distance to the two interfaces is not equal. If only a low amount of reinitialization is used, this error is small, and the interfaces move mostly correctly according to the advection equation (6). By a low amount, we typically

mean that reinitialization is performed every 20-100 time steps; the exact number depends on the specific case. By a large amount of reinitialization, we typically mean that reinitialization is performed every 1-5 time steps, or maybe even in every substep of a multistep Runge-Kutta method. Other factors also affect the "amount" of reinitialization. The number of pseudo-time steps and the pseudo-CFL number used when solving Equation (8) can be varied. One can also use a convergence criterion to stop reinitialization before the specified number of pseudo-time steps, reducing the amount of reinitialization.

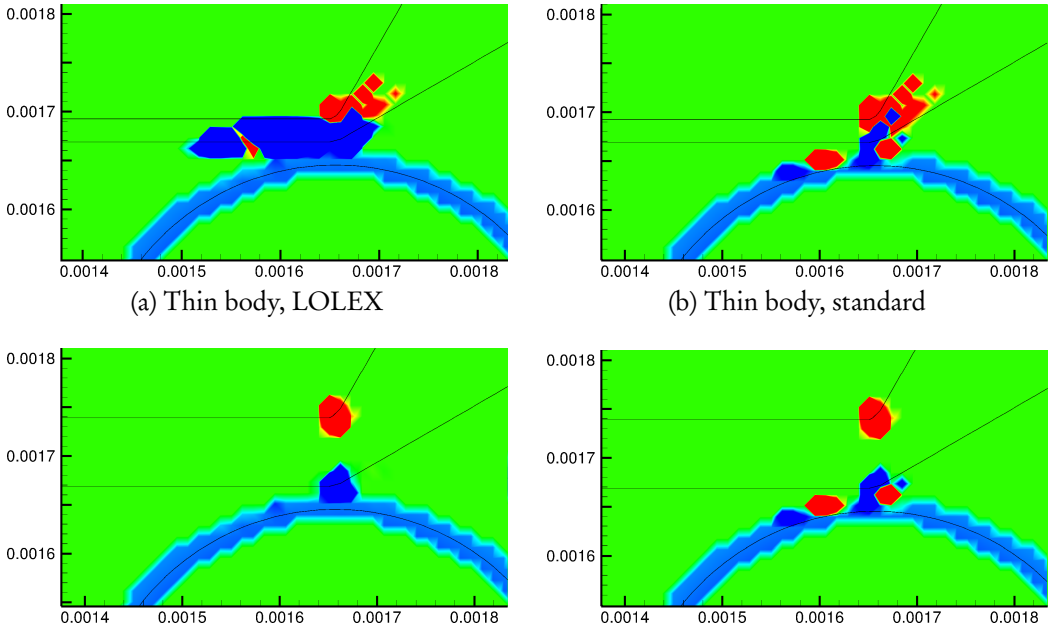
In the LOLEX method, a large amount of reinitialization is used to remove kinks in ϕ. This will introduce a large error for such thin bodies. It should be stressed that this only holds when a body is thin, i.e. when a body contains only one interior point somewhere, and when this thin body is close another body. It does not occur in the similar but much more important situation when two bodies are close (so that there is a thin film between them), since in that case we remove one of the bodies before reinitializing.

Since this problem only occurs with bodies that are too thin to be resolved, it is not a significant problem in realistic simulations. In fact, it was only discovered using an artificial stress-test of the curvature calculation. In that test case, thin films were used. If these films are thinner than `2*dx`, the curvature becomes incorrect due to the reinitialization deforming the thin body. The resulting error is shown in Figure 4.7.

In Figure 4.7 (a) and (b), a thin body is placed close to a circle and the curvature is calculated using the LOLEX method and the standard method, respectively. Both methods give erroneous values. In Figure 4.7 (c) and (d), the thin body has been widened. In (c), the LOLEX method gives the correct results everywhere, while in (d) the standard method gives incorrect results in the area between the bodies.

As seen in Figure 4.8, reinitialization of such a thin body can result in a deformation inducing an incorrect curvature. Note in Figure 4.7 that the problem occurs only when a thin body is close to another body. In the areas where the thin body is far from other bodies, even though use of the LOLEX method is triggered by the kink inside the thin body, the single body is not reinitialized and the calculated curvature is correct (the curvature of a straight line is zero).

The conclusion of all this is that thin bodies (i.e. bodies with only one internal point in some direction) should be avoided. This agrees with common sense, since such bodies are not resolved properly, and a finer grid must be used if one wants to work with thinner bodies. However, the possible effect of reinitialization on the interaction of e.g. colliding droplets is a related effect in a physically interesting case that should be more closely examined. Since the ϕ representing a thin film between two drops is just the ϕ of a thin body turned upside-down, it is reasonable to think that the effect occurs for thin films as well. The author is not aware of any thorough studies of such



(a) Thin body, LOLEX (b) Thin body, standard

(c) No thin body, LOLEX (d) No thin body, standard

Figure 4.7: Illustration of the curvature error when using the LOLEX method on thin bodies. When thin bodies are present, both LOLEX and the standard method give the wrong result. When no thin bodies are present, the LOLEX method gives the correct result while the standard method gives the wrong result (the two lower red dots in (d)).

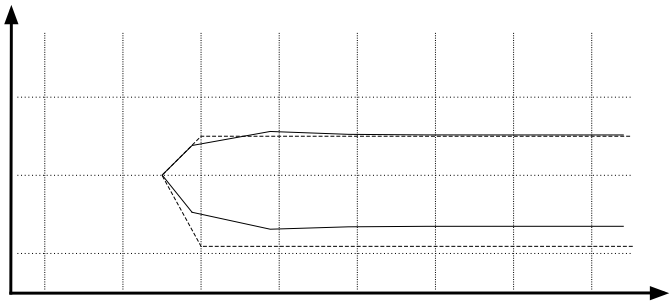


Figure 4.8: Thin body deformed by reinitialization. The dotted line shows the initial interface, where ϕ is a signed distance function. The full line shows the result of reinitialization. In theory these should overlap. The width of the initial body is $1.45\Delta x$.

effects in the literature, although several authors (e.g. Smereka in [41]) mention that reinitialization is turned off in their simulations of collisions.

In this light, it would be instructive to test the effect of reinitialization on the coalescence of droplets, to determine whether this affects the behaviour. Reinitialization should in theory not have any adverse effect on the motion of colliding droplets (or any bodies), but theory is often different from practice. Some tests with different amounts of reinitialization are therefore performed further down in Section 5.4.

To summarize this section, the LOLEX method tends to give errors as large as the standard method for bodies that are not properly resolved by the level-set method. This is not a problem *per se*, since further grid refinements are necessary in such cases anyway, and then the problem goes away. However, the author feels that any differences between the LOLEX method and the standard discretizations should be pointed out clearly.

4.3 LOLEX normal vector calculations

ACCURATE NORMAL VECTORS close to the interface are crucial to level-set simulations. The importance in reinitialization has been suggested above, coming from the fact that normal vectors are used both in finding the upwind direction and in calculating the right-hand side of Equation (8). Normal vectors are equally important in calculating the extension velocity, where an error would lead to the interface not moving according to the flow.

The Lervåg method is a variant of the Macklin and Lowengrub method, using different interpolating functions.

In this section we present results of accurate normal vector calculations using the LOLEX method on several geometric test cases. These results are compared both to the standard central-differences discretization, to a directional-differences discretization and to the curve-fitting method of Lervåg[22].

When calculating the normal vectors with the LOLEX method, we are again faced with two values at each point. In this case, however, it does not make sense to average the values, as with the curvatures. Instead we choose the normal vector corresponding to the closest interface. If the distances are exactly equal, we pick the normal vector corresponding to the first body. In this case, there is no correct answer, but picking a normal vector which is at least correct for one body should be better than using one which is correct for neither. This has been discussed by Lervåg in the context of curve fitting methods; he reports in [22] that the least-squares quadratic curve fitting employed by Macklin and Lowengrub gives a normal vector which is incorrect for both interfaces in the case where the distances from the grid point to both interfaces is exactly equal.

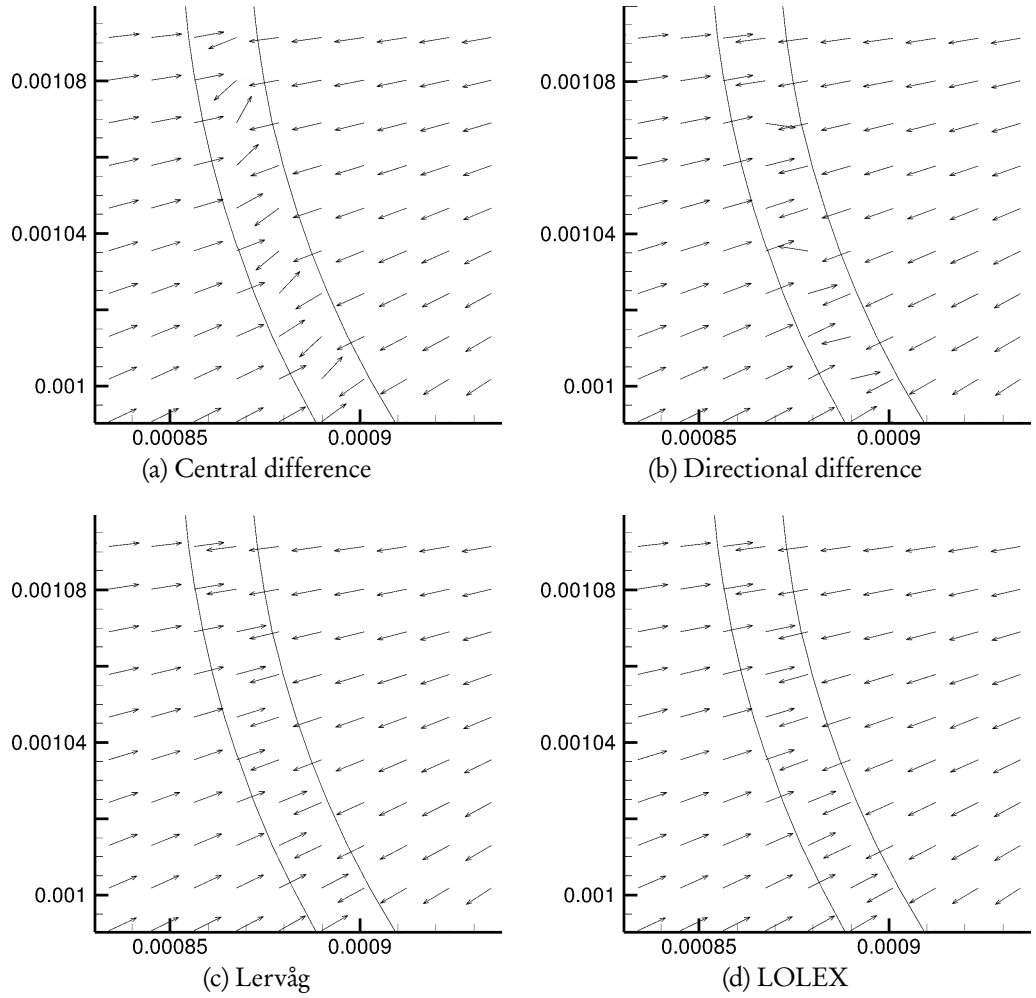


Figure 4.9: Comparison of normal vector calculations using different methods.

Using this selection of the proper normal vector, the normal vectors were calculated for the thin circle test case discussed previously in Figure 4.2. In this case, the width of the circle was $1.6\Delta x$. The results for all four methods are shown in Figure 4.9.

Perhaps the most surprising result of this is that the directional difference method is not much better than the central difference method. This is partly what prompted the use of curve fitting methods; Macklin and Lowengrub initially used directional differences and additional grid refinement in [24], but switched to curve-fitting methods in [25]. As is seen in Figure 4.9 (c), curve fitting methods (the method by Lervåg is used here) give the correct result. In (d), we see that the LOLEX method also gives the correct result. It is impossible to distinguish the results in (c) and (d) without overlaying

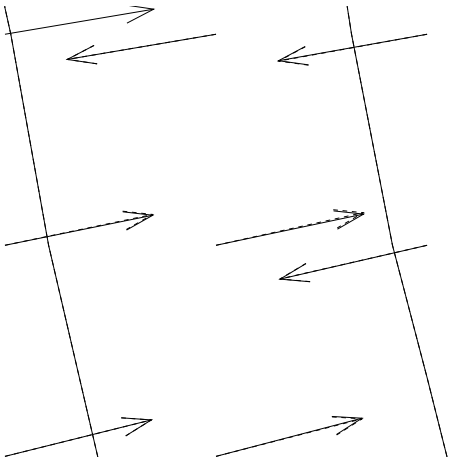


Figure 4.10: Comparison of the Lervåg method and the LOLEX method for calculating normal vectors. The LOLEX normal vectors are shown as dotted lines, the Lervåg normal vectors are shown as full lines. A minute difference can be seen.

the figures and zooming in a lot; then a minute difference can be seen, as in Figure 4.10. Here, the normal vectors calculated by the LOLEX method are shown as dotted lines, and the ones calculated by the Lervåg method are shown as full lines. The difference seen is too small to have any impact on the result of simulations.

Even though it is easy to see that the improved normal vectors look correct, it is nevertheless reassuring to see two completely different methods give essentially exactly the same result.

4.4 LOLEX curvature calculations in 3D

As pointed out several times already, the main advantage of the present LOLEX method over the Macklin and Lowengrub method is that it scales easily to 3D. This is because the present method retains the implicitness of the level-set method. A 3D extension of the Macklin and Lowengrub method, on the other hand, would fit a local surface to the point of interest. Curvature estimation in 3D based on local surface fitting has long been a topic of research in computer vision, see [9] for a review of various methods including the use biquadratic surface and of splines. The conclusion of [9] is that these methods are very sensitive to numerical noise (in their context, sensor noise). In the current case, noise is to be expected, as can be seen in Figure 3.8 (b). Due to this fact, methods in computer vision that avoid local

surface fitting and calculate only the sign of the curvature have been introduced, since this quantity can be calculated more reliably[46]. This is not a viable alternative in two-phase flow simulations as reported here.

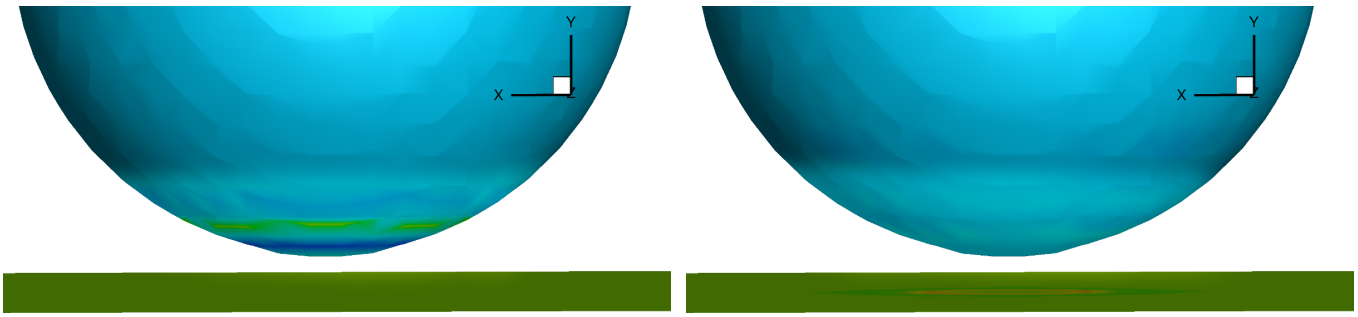
By contrast, the LOLEX method scales easily to 3D. Having stated this multiple times, it is time for the author to put his money where his mouth is. A proof-of-concept implementation was completed in 4 days, given the already working 2D code with LOLEX and a 3D code where the standard curvature discretization and routines used by the LOLEX method (e.g. reinitialization) were present.

The results of the proof-of-concept implementation are shown in Figure 4.11. In this case, a bubble is shown above a plane, with distance 1.2 Δx at the closest. The grid is $50\times50\times50$, and the bubble radius is 12.5 Δx. The surfaces are colored according to the curvature (interpolated to the surface). In Figure 4.11 (a), the standard method is used. In 3D, this is the 27-point stencil due to Kang et al. [18]. In Figure 4.11 (b), the LOLEX method is used to extract a local level set, and the curvature is then calculated using the same 27-point stencil on this local level set. It is seen that the LOLEX method performs much better than the standard discretization in areas where the bubble and plane are in close proximity. The thin, yellow ring shown on the plane in Figure 4.11 (b) is the result of a bug which was not fixed in the proof-of-concept version here. Bugfixing takes a lot of time, time which in this case is better spent setting up and running more physically interesting simulations in 2D.

The analytical curvature in this case is -10. The standard discretization performs well away from kinks, where the variation in curvature is at most $\pm0.2\%$. Close to the kink, the standard discretization has errors of $\pm80\%$. The LOLEX method has the same variation as the standard method away from kinks, while the variation is $\pm2\%$ close to the kink. Thus it is seen that the 3D LOLEX method performs much better than the standard method close to kinks in the level-set function. There is still a small error, of the same size as reported above in 2D, so the error is probably caused by reinitialization here as well. A deviation of this magnitude is unlikely to have a large impact on simulations, in contrast to the errors from the standard discretization.

To the author's knowledge, improved curvature calculations in three spatial dimensions that handle general topologies have not been reported before in the literature. Salac and Lu report results of 3D simulations in [40], but it is not known how (or if) they handle problems like that illustrated in Figure 3.3, i.e. a body folding back onto itself. They also do not discuss the problem of needing good normal vectors at the interface in order to solve the reinitialization equation.

The fact that such calculations have not been reported previously may be due to the computational costs of 3D simulations. This cost is such that highly accurate (i.e. interface tracking) 3D simulations have only recently become possible, e.g. in [43] where the authors use both parallel codes and adaptive grid refinement. The tumor-tissue simulations considered by Macklin and Lowengrub in [24] are demanding



(a) Standard discretization (b) LOLEX method

Figure 4.11: Comparison of the standard 3D curvature discretization (a) and the 3D LOLEX method (b). The surfaces are colored according to the curvature, and the standard method is seen to give the wrong result close to the kink, as the sphere should be uniformly colored. The shadow used to give an impression of 3D is removed for the lower part of both spheres, so that the color difference can be seen. This is why the lower part of both spheres looks brighter than the upper part. The orange ring on the green plane in (b) is a bug which has not been fixed.

enough that supercomputer hardware is used even though the problem is still 2D. The 3D simulations reported by Salac and Lu in [40] consider motion by mean curvature, which is computationally much less expensive than fluid flow simulations. Given the current developments toward petascale supercomputers, and particularly the rapid evolution in GPU-accelerated solvers, dynamic 3D level-set simulations of colliding bodies are going to become more commonplace. When this happens, a method such as the present one will be necessary in order to get accurate results.

4.5 Concluding remarks

In this section, results of using the LOLEX method have been presented and compared with results of other methods for geometric test cases. The LOLEX method has been found to perform well in all test cases, with the minor exception of thin bodies which are close to other bodies. This exception is not relevant for physically interesting simulations. The computational efficiency of the LOLEX method has been investigated, and it was concluded that for typical body-collision simulations, the performance impact is negligible. Various methods for averaging the curvature of two close bodies have been investigated, and a compromise between a geometric weighted

average and supremum average was found to perform best. 3D curvature calculations using the LOLEX method are reported, and the method is seen to give errors of ±2% while the standard method gives errors of ±80%, on a test case where a bubble was placed $1.2\Delta x$ away from a flat interface.

§5 Dynamic simulations using LOLEX

Contents

When I meet God, I am going to ask him two questions: Why relativity? And why turbulence? I really believe he will have an answer for the first.

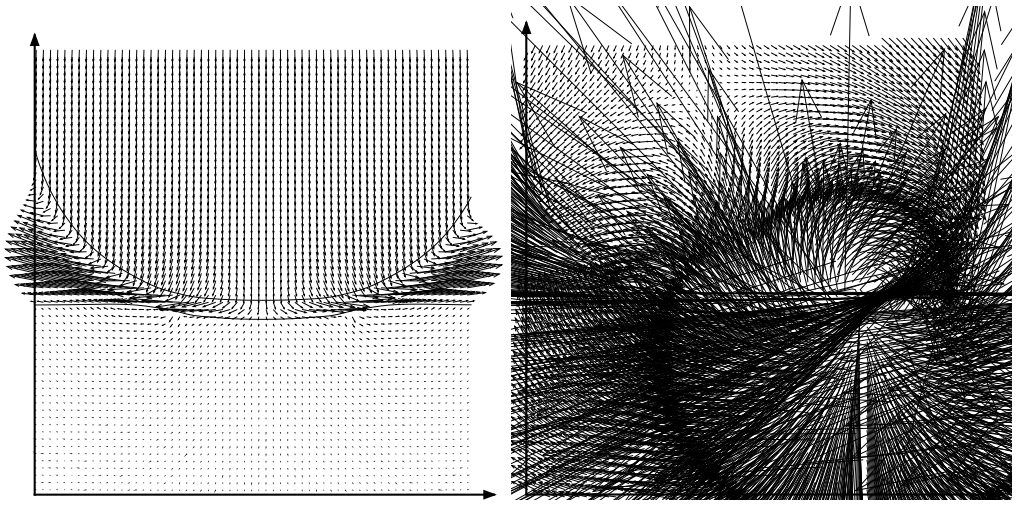
Werner Heisenberg

5.1 Summary of previous simulations

THE RESULTS OF SOME SIMULATIONS performed with the present codes are recalled here. These were completed during the author's specialization project in the fall of 2011, and the results here are from that project report[6]. In these simulations, either the standard method or the Salac and Lu method was used to calculate the curvature.

5.1.1 Droplet colliding with pool

PERHAPS THE MAIN CASE considered in the project report was a 0.6 mm diameter droplet of methanol falling through air onto a pool of methanol from a height of 0.65 mm. The droplet reached a speed of 0.09 m/s before colliding with the liquid pool. This velocity is in the lower end of the range of velocities studied experimentally. The case was chosen because the standard method fails to produce any interesting results here. The simulations were performed in 2D, because the corresponding axisymmetric case could not be simulated in a reasonable amount of time, i.e. less than a month of computation time. This is because the mean curvature of a circular rod is exactly half the mean curvature of a sphere with the same radius. In turn this means that the pressure due to surface tension of a droplet in 3D is twice the pressure due to surface tension of a droplet in 2D. As is discussed below, high pressure and density differences across an interface are hard to simulate in a stable fashion, requiring a larger computation time.



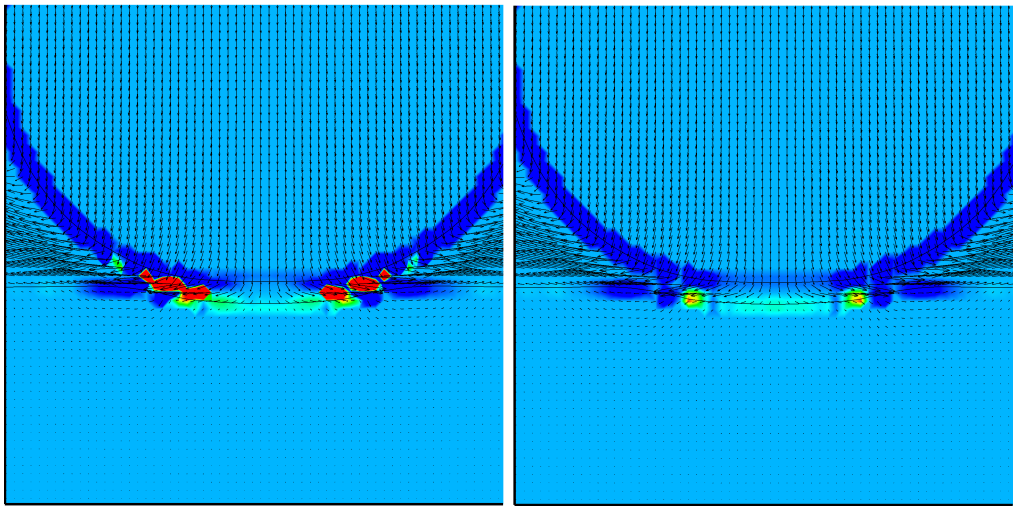
(a) Frame just before crash (b) Frame of crash

Figure 5.1: The droplet colliding with the pool, showing when the standard method crashes. The distance from the droplet to the pool is 2.6 Δx in these figures. The velocity field is plotted, and is seen to diverge spectacularly in Figure b). The same scale is used for the vector arrows in both plots.

The simulation of this case with the standard method crashed when the droplet was $2.6\Delta x$ away from the pool, or after 0.01193 s. The crash in this simulation using the standard method is caused by the erroneous curvature field, which induces an unphysical pressure field. A typical crash is shown in Figure 5.1. In Figure 5.2, the curvature field is plotted alongside the (correct) curvature field that is calculated by the Salac and Lu method. This demonstrates that the curvature is the culprit here, since the simulation using the Salac and Lu method did not crash at this point, and the curvature calculation is the only difference between the two. Note that the positive curvature indicated in two places (red areas) for the Salac and Lu method is indeed correct. Note also that in these simulations, the standard method was used for calculating the normal vectors, as improved normal vectors were not considered in the project report.

Some settings in the codes are the same for all dynamic simulations performed in the project report and in the current thesis. The values of these are given here in Table 3.

For completeness, some further properties of the simulations above are provided here. As stated previously, the only difference between the two simulations is the curvature calculation method. No-slip conditions were enforced on all boundaries.



(a) Old method (b) Salac and Lu method

Figure 5.2: Why the ordinary method fails: the curvature field is plotted in the last frame before the crash, for the old method and for the Salac and Lu method (which does not crash in its next frame). The light blue background indicates zero curvature, the dark blue indicates a negative value, and red indicates a positive value. The curvature calculated by the Salac and Lu method is correct.

Further values are given in Table 4[6]. At the moment before impact, the Reynolds number was $\mathrm{Re} \approx 430$ and the Weber number was $\mathrm{We} \approx 2.1$.

Reinitialization of the level-set function was performed every time step, using a maximum of 50 pseudo-time steps, but typically around 5 steps were necessary to reach convergence of the reinitialization equation.[7] There is some potential for trouble in the CFL condition for the Navier-Stokes equations, as the CFL-coefficient depends on the curvature field. This means that an erroneous curvature field, with κ becoming orders of magnitude too large, could force the time step to become extremely small, which means that the code will crash. The CFL condition used here is given in Equation (26), and the contribution from the curvature is given in Equation (30). The total runtime for the simulation was 11 days 4 hours on an Intel Core2 Q8400 processor, running on a single core as the present codes are not parallelized.

As stated, the simulation using the Salac and Lu method does not fail where the

[6] Using a thin domain such as here means that the edges of the domain influence the falling droplet, but a significantly larger domain makes the computation too lengthy. The small domain used here already means the runtime is measured in days and weeks.

[7] When ϕ deviates from a signed distance function less than a given tolerance up to N grid cells out from the interface, we say that the reinitialization equation has converged.

Table 3: Table of settings common to all simulations performed here

Property	Value
Boundary ghost cells	3
Grid stretching	No
Tangential velocity of walls	0.0 m/s
Pressure reference method	Explicit
Interface-capturing method	GFM
Delta function	Standard smeared
Use extrapolated velocity for interface advection	Yes
Update smeared sign function at every pseude-time step	Yes
CFL number for velocity extrapolation	1.0
CFL number for curvature extrapolation	1.0
Calculate curvature everywhere	No
Discretization of convective Navier-Stokes terms	WENO
Discretization in reinitialization equation	WENO
Discretization in level-set advection equation	WENO
Runge-Kutta method for Navier-Stokes	SSP-RK 3()4
Runge-Kutta method for reinitialization	SSP-RK 2()4
Runge-Kutta method for velocity extrapolation	SSP-RK 2()4

ordinary method does, but this simulation also failed some time later, at 0.0123 s. This is shown in Figure 5.3, where the velocity field does not go crazy, so it has not been plotted. In this simulation, the droplet does begin merging with the pool, and the result looks reasonable up until the simulation fails. The merging of the droplet and the pool happens almost completely at the right hand side, creating a thin finger of air between the droplet and the pool. This is thought to be the cause of the failure, for two reasons. First of all, after the merger there is only one body present in the calculation domain, and the Salac and Lu method cannot work since it requires at least two bodies to be present. Thus, the ordinary method is used for curvature calculation, and the result is incorrect, as is seen in Figure 5.3, where the large red area is indicating an incorrect curvature field. This will again induce an unphysical pressure, which may have caused the crash. Secondly, as mentioned, the CFL condition used includes the curvature field. As the curvature field is erroneous in this case, the CFL condition could also be the cause of the crash. The reasoning behind this is that the CFL condition tries to enforce a very precise calculation using very small time steps, but the intitial value used has very incorrect values of e.g. the curvature field. When the method is given an erroneous input, it will of course give an erroneous output. It would perhaps be better if the method used large time steps in this case, reducing the influence of the erroneous

Table 4: Table of physical properties and numerical settings used for the methanol-in-air case.

Property	Value	Unit
Domain size	1.5×2.25	mm^2
Droplet diameter	0.6	mm
Fall height	0.65	mm
Methanol		
ρ	792	kg/m
μ	$560 \cdot 10^{-6}$	Pa s
Air		
ρ	1.2	kg/m
μ	$20 \cdot 10^{-6}$	Pa s
Surface tension	$2250 \cdot 10^{-6}$	N/m
Gravity	$-9.81\hat{y}$	m/s^2
Numerical settings		
Grid size	162×242	
Poisson solver	ILU-GMRES	
Poisson solver residual	$1.0 \cdot 10^{-12}$	
CFL number	0.3	
Level-set reinitialization	Every time step	
Maximum pseudo-time τ	4.0	
HCR-2 reinitialization	On	
Reinitialization CFL number	1.0	
Level-set advection discretization	First-order upwind	

curvature. If the simulation continues for a few large time steps without crashing, level-set advection will give a situation where the curvature field is less singular, enabling the simulation to continue past this problem. The influence of curvature on the CFL condition has not been investigated in further detail here.

Since it seems somewhat strange that the merging of the droplet and the pool happens all the way to one side, possible causes for this behaviour were investigated. Two questions were considered: first of all, why does the merger happen at the side (and not the middle), and second of all, why does it not happen at both sides (symmetrically) in this symmetric case?

It turns out that off-center merging has been reported in the literature. The ratio between the viscosity of the two fluids, γ, can be used to characterize whether off-center merging happens or not. γ is defined as the dynamic viscosity of the drop divided by the viscosity of the ambient. With a $\gamma \gtrsim 1$, off-center merging is expected[30]. This

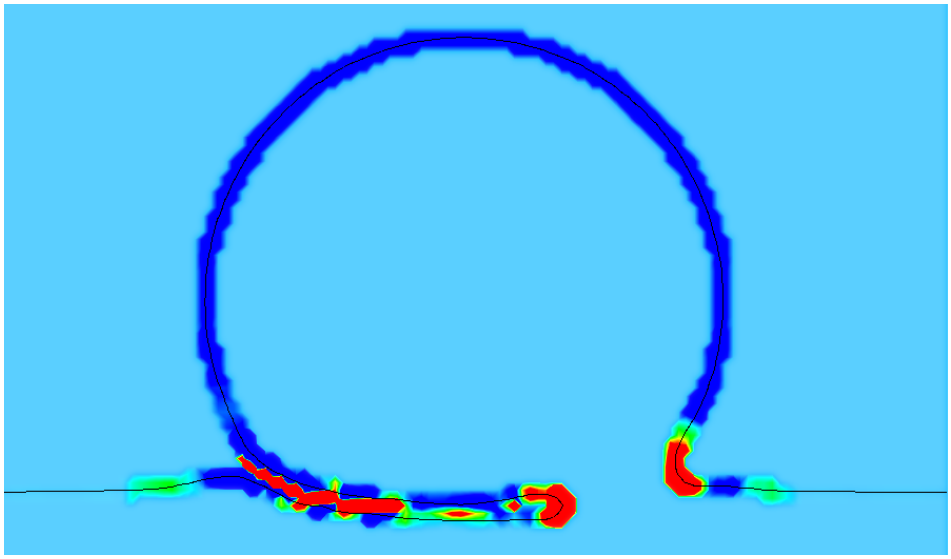


Figure 5.3: The curvature field plotted in the final frame before the simulation crashed. Note the red curvature field inside the air finger between the drop and the pool, which is incorrect. Note also the wave in the pool surface that can be seen on the left-hand side.

result holds for a low Reynolds number flow; it is not known whether this result holds for large Reynolds numbers as in the present case. In the present case, $\gamma = 28$, so off-center merging could perhaps be reasonable. The result reported in [30] is also for 3D; it is not known if the corresponding threshold is the same in 2D.

The off-center merging could also be a grid effect: the width of the air film is only one or two grid cells, which causes the interfaces to become less smooth, even blocky-looking, as seen in Figure 5.4. Note that in this figure, the part of the grid that is in air is colored white, and the part that is in methanol is colored dark grey. The blockiness is more pronounced at the sides, where the interface is not parallel to the y-direction, as it is in the middle. As is indicated further down, this effect is due to incorrect normal vectors, since the standard method of calculating normal vectors were used here.

The answer to the second question seems to be that instabilities arising in the simulation break the initial symmetry of the problem. If the instabilities were to be removed, a symmetric merging should be the result.

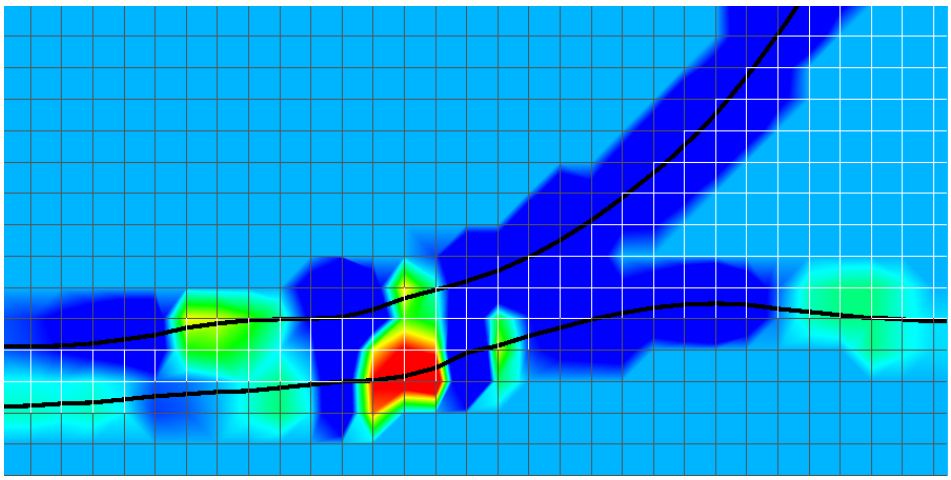


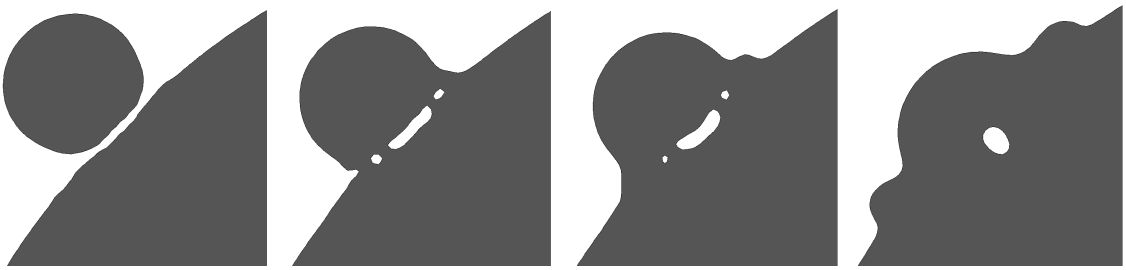
Figure 5.4: A zoomed in plot of the curvature field, before the droplet has merged into the pool. The grid is shown, and is colored white in air and dark grey in the fluid. It is seen that the interfaces have become blocky, following the edges of grid cells instead of being smooth. This is chiefly caused by incorrect normal vectors.

5.1.2 Diagonal simulation of droplet colliding with pool

In an attempt to work around the problem of merging at one side, a simulation was also performed in the project report where the physical situation was rotated 45° counter-clockwise relative to the grid. The idea was that if the off-center merging was a grid effect, this would make the droplet merge with the pool at the middle. This should in turn make the air fingers between the droplet and the pool shorter and less problematic. In this simulation, the edges of the pool intersect the edges of the computational domain at 45° as well. Since the boundary conditions used for the level-set function were just simple mirroring, this became a small problem, as mirroring boundary conditions will after some time make the pool intersect the domain boundaries at 90°. This induced a wave motion in the surface of the pool, making a direct comparison to the previous case more complicated. However, this motion became damped out almost completely during the fall of the drop, so the difference should not be very important.

While this approach was not successful in making the droplet and pool merge at the middle, it was nonetheless more of a success than the previous simulation. In this case, the droplet merged at both the left and the right end, trapping an air bubble below it. This is shown in Figure 5.5.

In this case, the simulation did not crash, and could be continued for as long as one wants. After the final frame shown in Figure 5.5, the waves on the right and left



(a) $t = 0.0296$ (b) $t = 0.02995$ (c) $t = 0.03035$ (d) $t = 0.03085$

Figure 5.5: The droplet colliding with the pool in the diagonal simulation. This plot is zoomed in on the droplet, the dark grey indicates methanol and the white indicates air.

continue outwards, and the drop merges completely with the pool. The simulation was continued until $t = 0.05$ s to see if anything interesting happens, e.g. a secondary jetting, but the result is just wave motion in the pool. This could be due to effects of the narrow domain, but for low kinetic energies such as here, the experimental results do not show jetting either. This is due to viscosity damping out the comparatively small kinetic energies before jetting can occur.

The reader may also have noticed that three air bubbles were in fact produced between the methanol droplet and the pool, but that the two smallest air bubbles disappeared. This is due to these small air bubbles being at the limit of the grid resolution, with one of them containing three grid points and the other four. As described previously, such thin structures are destroyed by the reinitialization. The largest air bubble, however, remains.

It is an interesting question whether the formation of such bubbles is physically realistic, or even typical, in droplet-pool collisions. The experimental techniques used by the team at SINTEF Energy Research for imaging such collisions, presented further down, do not permit a view of the interior of the fluids in motion. The PIV technique used by e.g. Mohamed-Kassim and Longmire, whose results are reported further down, does permit an interior view, but has mostly been used to study liquid/liquid systems. This is because the tracer particles used cannot be suspended in a gas. NMR imaging techniques consitute a new non-invasive tool for imaging multiphase flow, but have traditionally not been fast enough for studying fluid flow phenomena. This is currently changing, with studies in the literature using NMR for imaging falling liquid drops[12], and more recently for ultrafast imaging of bubbles in multiphase flow[47]. In this context, ultrafast means a reported sampling rate of 188 velocity field maps per second, which is slow compared to contemporary high-speed photography techniques reaching well into kHz sampling speeds, but is only an order of magnitude away from being

useful for studying liquid-in-liquid collisions. If the progress in this field continues at the current rate, detailed NMR imaging of droplet-pool merging for liquid-in-gas cases would be possible in a few years time. It will be interesting to see what new techniques eventually reveal about droplet collisions. In a recent paper, Thoroddsen et al. report for the first time a turbulent phenomenon occuring between a droplet and the underlying pool which it has collided with at high Reynolds number[49]; a feature of the flow which was not previously known and not expected *a priori*. It will be interesting to see whether nature has more surprises in store for us as both experimental and numerical techniques continue to improve their accuracy.

A final interesting feature of the present simulation that is worth mentioning is the waves produced on each side of the pool surface by the droplet. Such capillary waves are well-known from experiments, as is seen further down, and it is reassuring to see that the simulation reproduces them. These waves start forming before the droplet merges into the pool, and can be seen also in the previous simulation, e.g. on the left side in Figure 5.3. The first, smaller waves form due to the air between the droplet and the pool pushing the water in the pool away. After the droplet has merged with the pool, the small waves are dominated by larger capillary waves resembling those seen in dam-break simulations, as in Figure 5.5 (d).

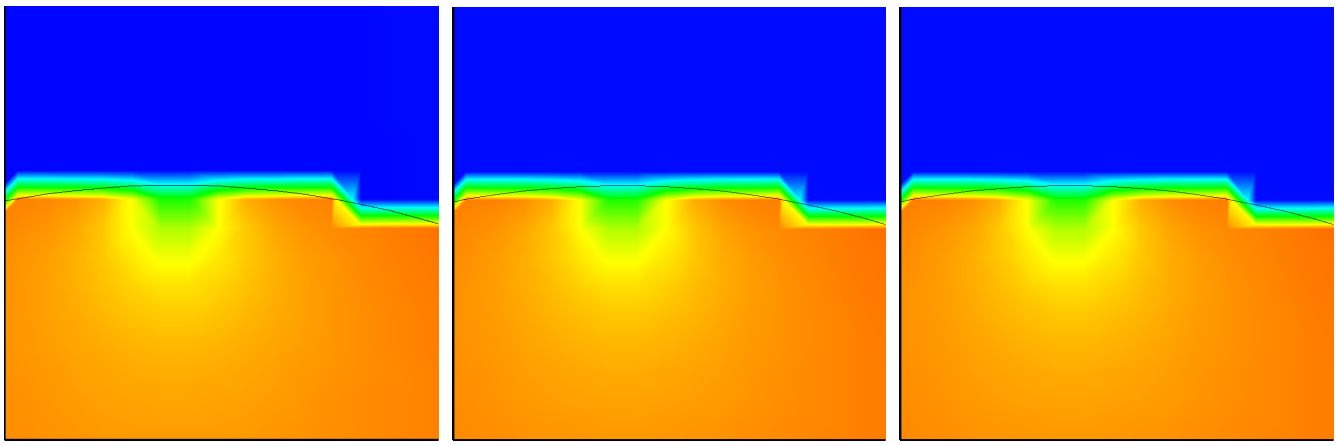
5.2 LOLEX on previous cases

USING THE LOLEX METHOD ON THESE CASES solves some problems, but does not improve the stability. In this section, we first discuss some attempts to improve the stability of the methanol-in-air case. Following this, we consider results of using the LOLEX method on this case.

Stability is always important in numerical simulations, but one cannot always get what one wants. In the methanol-in-air case considered in Section 5.1, the results are not stable. Especially the pressure field has large oscillations, suggesting that the results of this case are not to be trusted too much. Some effort was made to understand the cause of these instabilities, and whether they could be mitigated in a reasonable way.

Calculating the pressure here is numerically more complicated and time-consuming than for compressible flow.

As the pressure field is the most unstable, considering the numerical methods used for calculating the pressure is an obvious starting point. In the present work, the pressure is calculated by solving a Poisson equation, the result of using a projection method to decouple the incompressible Navier-Stokes equations. When solving this Poisson equation, the freely available PETSc library is used, with several different solvers appropriate to this equation. The alternatives include ILU-GMRES, ILU-BCGS, and the HYPRE-BoomerAMG. ILU denotes the choice of preconditioner,



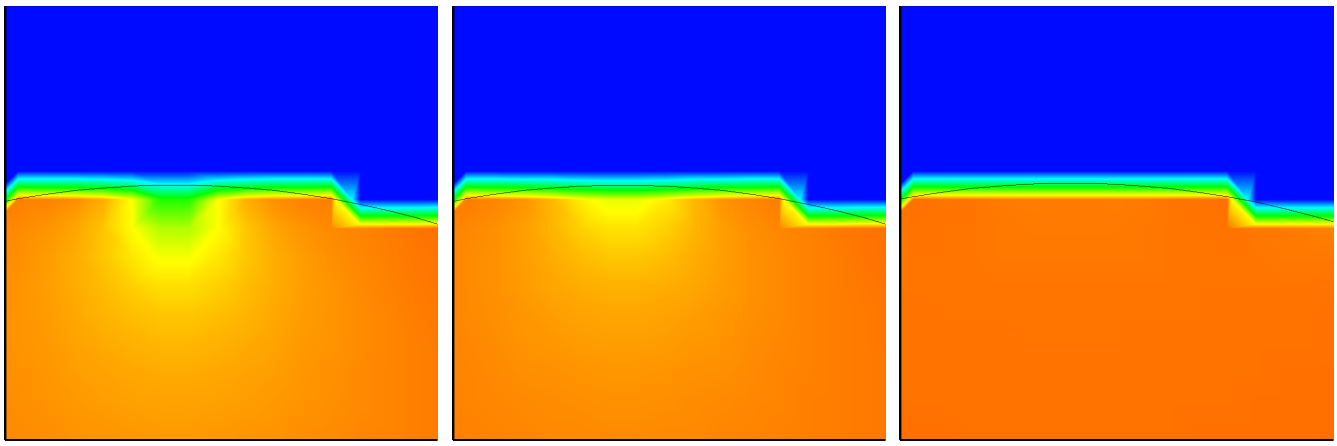
(a) GMRES (b) BCGS (c) HYPRE-BoomerAMG

Figure 5.6: Comparison of different Poisson solvers on the methanol-in-air case. The top of a falling droplet of methanol in air is shown, colored according to the pressure. The droplet has only fallen for 1.45 ms, and the speed is 0.01 m/s. The variation in pressure inside the droplet is unphysical, appearing and disappearing again in just 50 time steps. The droplet is almost exactly circular, and small enough that the gravitationally induced pressure gradient is negligible compared to the surface tension, so the pressure should be uniform inside it.

and stands for Incomplete Lower-Upper preconditioning. These different solvers were tested on the methanol-in-air case, the results are shown in Figure 5.6. In these figures, a zoom in on the top of the methanol droplet is shown. The droplet has fallen through air for 1.45 ms before this snapshot, and the droplet speed is 0.01 m/s, i.e. not very high. The droplet is still completely circular.

As is seen from this figure, the choice of Poisson solver does nothing to reduce the instabilities. This also rules out effects of the preconditioner, since the HYPRE-BoomerAMG method does not use ILU preconditioning. It is noted that the GMRES method used much more CPU time than the other two methods, as is expected.

Further investigations were made, varying the pressure reference method, the amount of reinitialization and the CFL number. These simulations were all performed using the HYPRE-BoomerAMG Poisson solver. None of these provided any improvement, except for decreasing the CFL number. The simulations in Figure 5.6 all used a CFL number of 0.3, which is already quite low. Decreasing it further to 0.1 gave the results in Figure 5.7. It is seen in this figure that decreasing the CFL number reduces the erroneous pressure oscillation, but does not remove it completely. Reducing the



(a) CFL 0.3 (b) CFL 0.1 (c) Previous frame

Figure 5.7: Effect on stability of reducing the CFL number. The CFL number is reduced from 0.3 to 0.1, which reduces the instability somewhat, but does not remove it. The frame after the one in (a) and (b) is shown in (c); here, no trace of the instability is seen. The frame prior to (a) and (b) is identical to (c) with regards to the pressure.

CFL number further would be too computationally intensive. It should be noted here that for the current system (i.e. the Navier-Stokes equations), the CFL condition is not a rigorous condition on convergence, but rather a touch-and-go criterion. Thus it is not surprising that instabilites can arise even for CFL numbers below 1.

A final attempt at stabilizing the code was made after some discussions with Karl Yngve Lervåg, who suggested that using a more diffusive spatial discretization in solving the advection equation, Equation (6), could improve the stability. This means using e.g. a forward upwind discretization instead of the WENO discretization used elsewhere in this work. This turned out to work fairly well, making the case more stable, but not completely so.

Using this more diffusive level-set advection, the methanol-in-air case was simulated using the LOLEX method for curvature and normal vector calculation. The results are shown in Figure 5.8. The pressure in these results is still oscillatory, but there is a result (as opposed to a crash in previous attempts). The curvature and normal vectors behave well, and the resulting interface motion looks sensible. In particular, the capillary waves (indicated by arrows) look like those seen in experiments. The merging between the droplet and the pool occurs before the pool is significantly deformed, again in agreement with experiments. It should be noted that the (particle) Reynolds number in this case is $\mathrm{Re} \approx 430$, and the Weber number is $\mathrm{We} \approx 2.1$, at the time of impact with the surface. The density ratio between the two fluids is $760 : 1$.

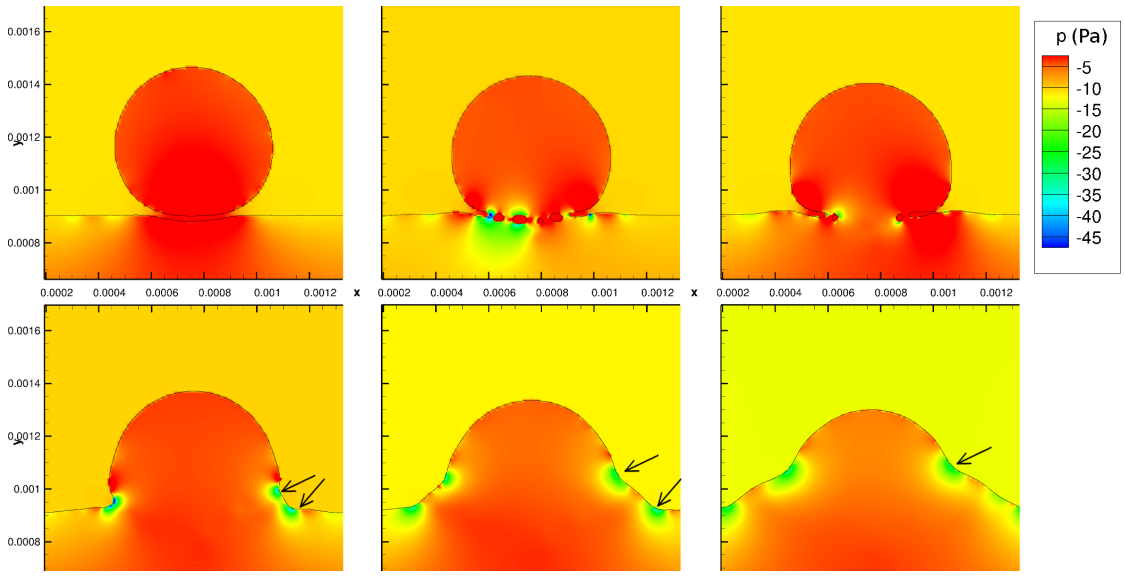


Figure 5.8: Methanol droplet merging with pool of methanol. The pressure field is indicated in color. There are still pressure instabilities, as can be seen, but not as bad as in previous cases. Apart from the problematic pressure, the merging is seen to progress reasonably, with two capillary waves seen on each side (indicated in arrows on the right side). The snapshots are 3 ms apart.

The conclusion of this section is that the methanol-in-air case cannot be simulated in a stable way using the present codes, and that while some results can be produced, it is difficult to know how much these results can be trusted. An effort is currently underway to improve the stability of the present codes. The Reynolds number for this case is quite high, as is the density ratio, so it is reasonable that this case is hard to simulate. Sussman et al. note in [43] that large density ratios are numerically demanding, and suggest that improved methods like their hybrid level-set volume-of-fluid method are necessary. A fix of such a magnitude is too large to be contemplated during the course of this Master's thesis. Instead, we consider a case which is less demanding to simulate numerically.

In order to get more stable results for the purposes of the present discussion, new simulations were considered with a lower Reynolds number and density ratio. These liquid-in-liquid simulations do not have the same immediate relevance to the study of natural gas condensation as e.g. the methanol-in-air case considered here, but they enable simulations where the results can be trusted to a larger degree. Confirming that the LOLEX method gives good results on liquid-in-liquid phenomena when compared to experimental results means that when the stability issues are worked out in the future, liquid-in-gas phenomena can be reliably simulated as well.

5.3 LOLEX on 2D liquid/liquid cases

In order to simulate a liquid-in-liquid case, the experiments reported in [3] were used as a starting point. In these experiments, a bubble is gently placed on the interface, and merging happens. With these data for fluid properties, some simulations were performed where the droplet falls some distance before impacting the interface. These simulations with falling droplets are more relevant to the ultimate goal here, which is the study of droplet-pool interactions in natural-gas condensation.

The experiment in [3] that we will compare to considers a water droplet in a mixture of 80% decane and 20% polybutene. This case will be referred to as the water-in-decane case for brevity. 2D simulations of the water-in-decane case, with an initial falling height of 1.41 times the droplet diameter, were considered first. These simulations were chosen in order to verify the stability of this case, as well as for studying grid effects. The result was that the lower density difference and higher viscosity in the surrounding fluid provided a much more stable case than the methanol-in-air simulations. An example of the pressure field in this case is shown in Figure 5.9. It is symmetric and looks like it should. It is also free of oscillations from timestep to timestep, in contrast to the methanol-in-air simulations. Details of the employed parameters are given in Table 5.

After this test case was seen to be stable, the effects of grid refinement and domain width were studied. The results are shown in Figure 5.10 for an increased domain width and an increased grid refinement, relative to Figure 5.9. It is seen that increasing the domain width does not significantly affect the result, but increasing the grid refinement does. For this reason, the following simulations are run with the finest grid possible which still gives a simulation time which is feasible. It is noted that the runtime for the finest grid used here was 256580 s, or almost 3 days, up to the point of merging. When one wants to study the behaviour during and after merging as well, this simulation takes about a week to run, as the CFL condition decreases Δt significantly during the merging, when interface velocities become large. For this reason, the fine grid used in Figure 5.10 (c) was used in the following. It is noted that the grid used here is much finer than in the methanol-in-air case. This is another benefit of using denser and more viscous fluids here, where in this case the CFL condition becomes a less severe restriction on the grid refinement for a given total computation time than in the methanol-in-air case.

Using the fine grid described previously, both 2D and 2D axisymmetric simulations were performed. For 2D, the merging and subsequent behaviour is shown in Figure 5.11. In this simulation, the merging is seen to happen off-center. Using the previously

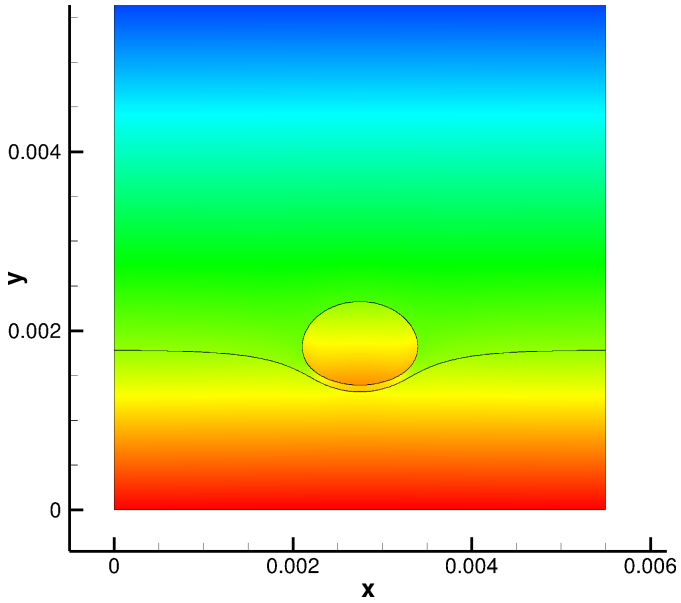
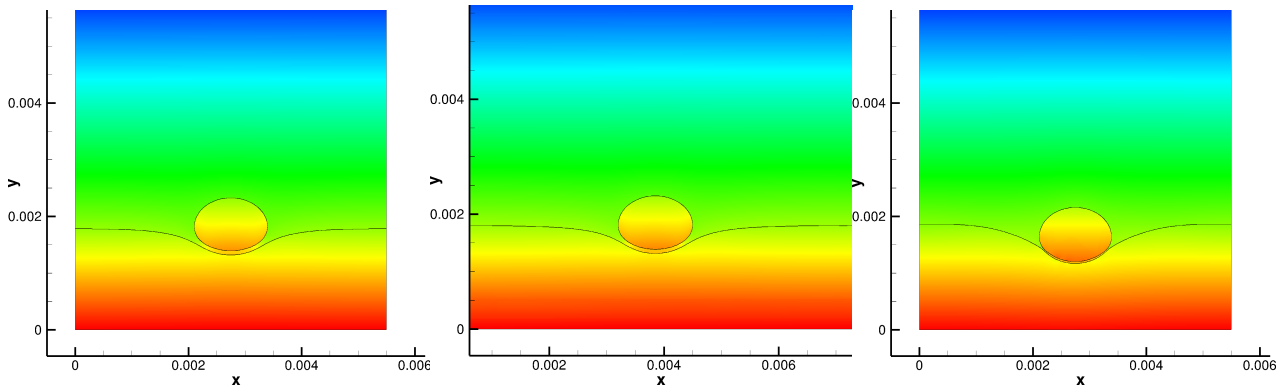


Figure 5.9: Contour plot of the pressure field just before merging, at $t = 0.09$ s after the droplet started to fall. The pressure field is seen to be symmetric and sensible.



(a) Base case (b) Wider domain (c) Finer grid

Figure 5.10: The effect of grid refinement and domain width on the water-in-decane case. All cases are shown at the same time ($t = 0.09$ s) after starting to fall. In the base case, a 350×390 grid is used to represent a 5.5×6.1 mm^2 domain. In (b), this is extended to a 490×390 grid representing 7.7×6.1 mm^2, giving a wider grid with the same size of Δx. In (c), the same domain size as (a) is used, but the grid refinement is increased to 490×546.

Table 5: Table of physical properties and numerical settings used for the water-in-decane case. Domain size and grid refinement are given for the coarsest grid.

Property	Value	Unit
Domain size	5.5×6.1	mm^2
Droplet diameter	1.1	mm
Fall height	1.56	mm
Water		
ρ	1000	kg/m
μ	$1\cdot10^{-3}$	Pa s
Decane		
ρ	760	kg/m
μ	$2\cdot10^{-3}$	Pa s
Surface tension	$2.97\cdot10^{-3}$	N/m
Gravity	$-9.81\hat{y}$	m/s^2
Numerical settings		
Grid size	350×390	
Poisson solver	HYPRE-BoomerAMG	
Poisson solver residual	$1.0\cdot10^{-12}$	
CFL number	0.8	
Level-set reinitialization	Every other time step	
Maximum pseudo-time τ	12.0	
HCR-2 reinitialization	On	
Reinitialization CFL number	0.8	
Reinitialization discretization	WENO	
Level-set advection discretization	WENO	

defined viscosity ratio, with $\gamma = 0.5$ here, indicates that the merging should not be off-center. In this simulation, the Reynolds number at the time of merging is $\mathrm{Re} \approx 1.1$, so the use of γ to determine whether off-center merging should happen could be reasonable. Then again, this simulation is two-dimensional, and the three-dimensional results reported in [30] are not guaranteed to hold here.

It is also seen in Figure 5.11 that the droplet rests on the interface for some time before the merging happens. This fact provides an additional clue as to why the merging is off-center. Looking at the flow field, it is seen that the emptying of the thin film between the droplet and the pool is a contributing factor to off-center merging. In Figure 5.12, the vector field of the flow in and around the thin film is shown. In this figure, the flow is strongest at the place where the merging subsequently happens. This flow will induce a Venturi-like phenomenon, which pulls the interfaces together

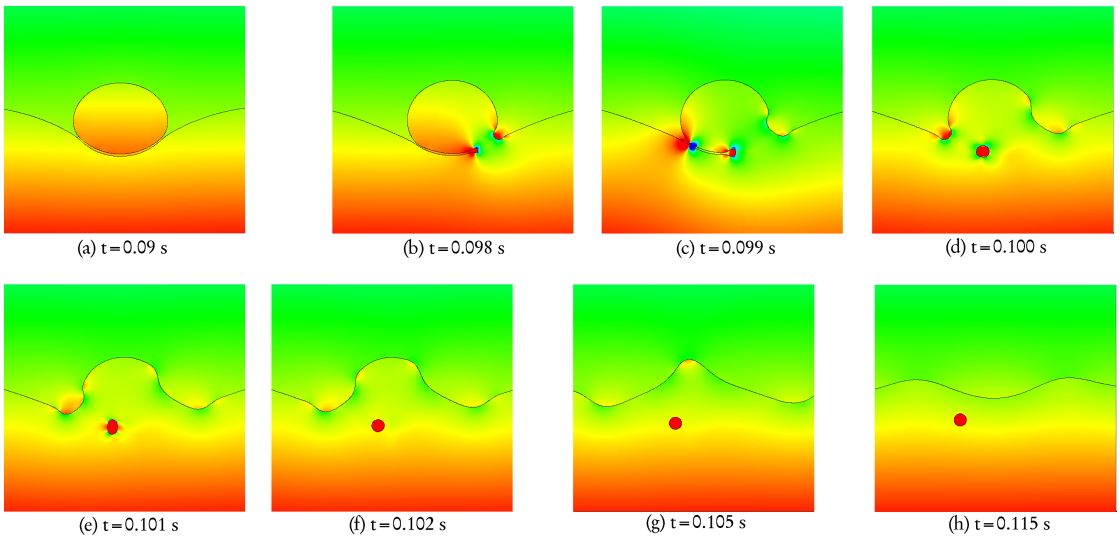


Figure 5.11: The behaviour during and after merging for the water-in-decane case in 2D, using the fine (350×390) grid . The times are indicated below each frame. The pressure field is indicated in color, and is stable and looks sensible. Note that increased spacing between frames indicates that a longer time interval has passed between these frames.

where the flow is strongest, further increasing the velocities at this point, *et cetera*. Further studies can be made into this effect, but this is outside the scope of the present work. Our main objective at this point is to compare the results obtained using the LOLEX method to experimental results. As 2D axisymmetric simulations can be compared directly with 3D experiments, they are more interesting for this purpose. Such cases will be considered next, after a brief detour into some considerations about reinitialization.

5.4 Effects of reinitialization on merging

REINITIALIZATION IS A NECESSARY COMPONENT of level-set simulations, but it is also a source of errors; particularly mass loss if excessive amounts is used. In the context of colliding bodies, it causes another problem: when the initial merging of bodies occurs, the surface tension induces large interface speeds. An example of this is shown in Figure 5.13. Here the level-set function of a droplet merging with a pool is shown for the 2D case described in the previous section. The

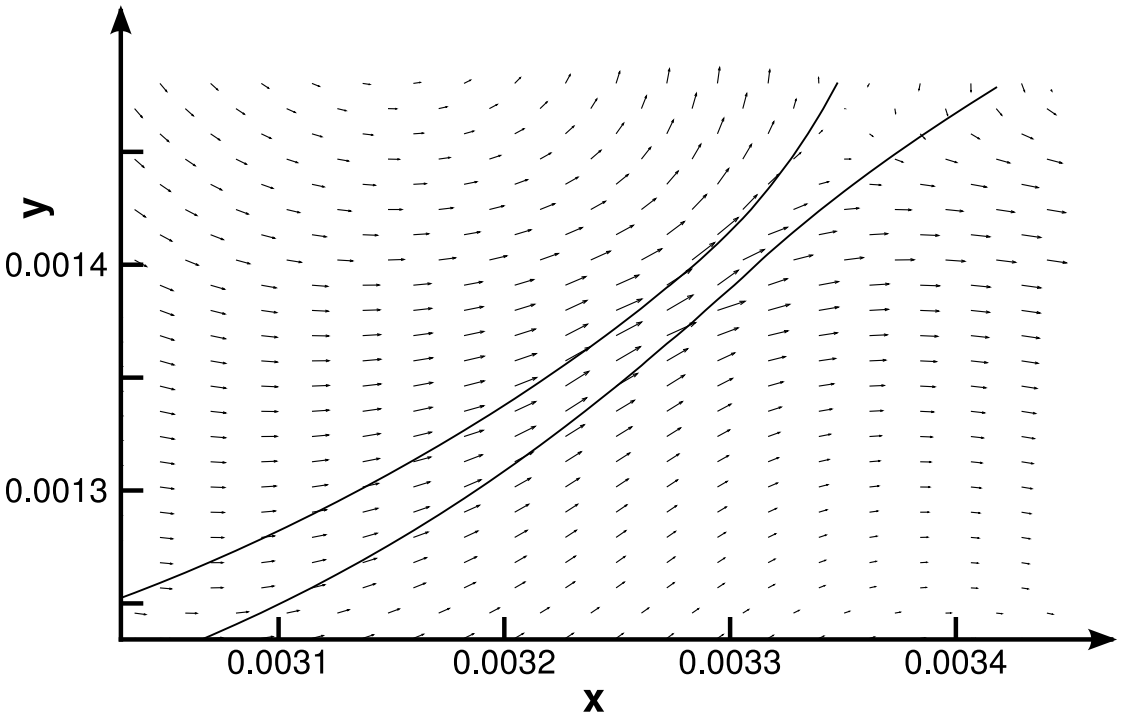
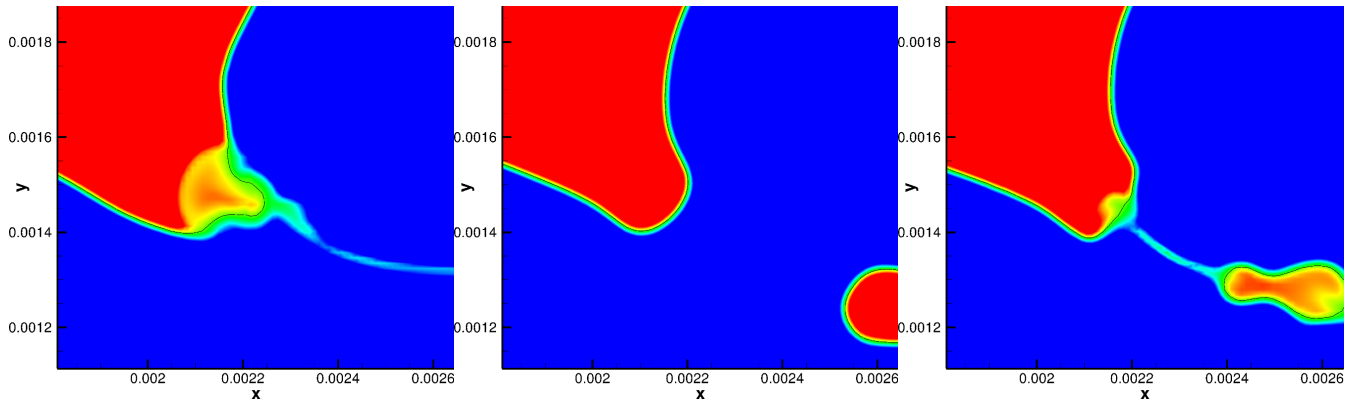


Figure 5.12: The flow field inside and around the thin film is shown for one of the sides. It is seen that the flow is strongest around the point where merging subsequently happens. Note that vectors are only shown for every other grid point.

color indicates the values of ϕ, and the color map is set such that the color should be either red (outside droplet) or blue (inside droplet) when more than Δx away from the interface.

In this figure, it is seen that the ϕ in Figure (a) which has been reinitialized frequently is incorrect, and that it fails to realize the entrained bubble seen in Figure (b). In Figure (a), reinitialization was used every time step and the reinitialization equation was solved for 15 pseudo-time steps or until it had converged. This is a large amount chosen in order to illustrate the problem; in practice it is more typical to reinitialize around every 10 time steps for cases such as this one, where the interface moves only a tiny fraction of a grid spacing from one time step to another. In other cases, where the CFL restriction is less severe, it is common to reinitialize every time step or every other time step.

In Figure (b), reinitialization was used every 50 time steps, again using up to 15 pseudo-time steps. In this case, the level-set function behaves as it should. Completely turning off reinitialization turns out to cause problems again, as seen in Figure (c). This is to be expected; if reinitialization was not necessary, it would not have been



(a) Heavy reinitialization (b) Light reinitialization (c) No reinitialization

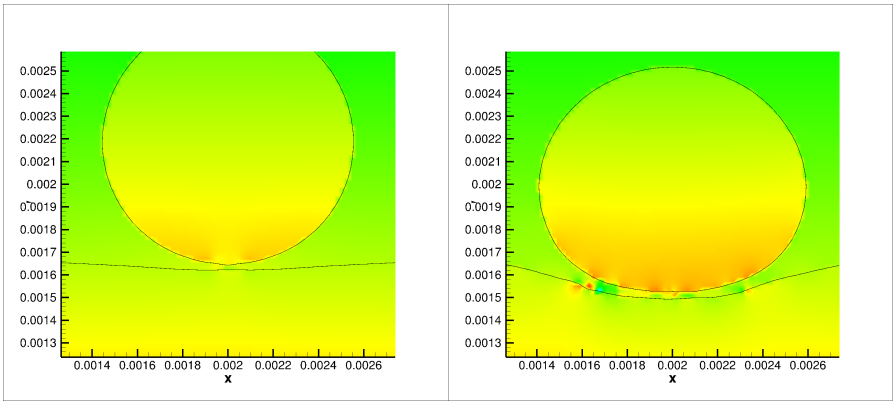
Figure 5.13: Level set function of a droplet merging with a pool. In (a), reinitialization was used every time step. In (b), reinitialization was used every 50 time steps. In (c), no reinitialization was used. The color indicates the value of ϕ, and should be either blue or red when more than Δx away from the interface.

invented in the first place. Note that it is not certain that the result in Figure (b) is physically correct, but since the level-set function does not deviate significantly from a signed-distance function in this case, as opposed to in Figure (a) and (c), the result in Figure (b) is trusted to a larger degree. The conclusion of this is that reinitialization should be performed about every 50-100 time steps for this case. Tests using every 50 and every 100 time steps did not show any difference.

In addition to this, there is another issue with reinitialization and advection that should be discussed here. When the standard method is used to calculate the normal vectors, the resulting vectors can be very far from normal to the interface, as is seen in Figure 4.9 (a). Since the normal vectors are used both for reinitialization and advection, this will result in advection and reinitialization deforming the interface significantly e.g. when simulating merging. This effect is seen above in Figure 5.4 on page 63.

To demonstrate both this effect and the effect of an erroneous curvature field, two simulations of the water-in-decane case were performed with a fall height of $2\Delta x$ or 0.04 mm, where one simulation used the LOLEX method for calculating curvature and normal vectors, and the other simulation used the standard method for both. The results of this test are shown in Figure 5.14.

The first thing to note in this figure is that when the standard method is used, the merging happens significantly later than with the LOLEX method. This is because the erroneous curvature field induces an artificially high pressure between the droplet and the pool. The erroneous curvature field also breaks the symmetry of this case, since the curvature errors are not symmetric.



(a) LOLEX method (b) Standard method

Figure 5.14: Comparison of the LOLEX method and the standard method for the water-in-decane case with almost zero fall height. In Figure (a), the interface and pressure field are shown for the LOLEX method in the frame just before merging, which is at $t = 0.01$ s. In Figure (b), the same are shown for the standard method in the frame before merging, which is at $t = 0.027$ s.

The effect of incorrect normal vectors can also be seen in Figure 5.14 (b), where the interfaces are slightly blocky in the region where the droplet and pool are close. A close up of this is shown in Figure 5.15, where the normal vectors are also plotted for every fourth grid point.

These figures give a clear indication of the errors that occur when using the standard method, and offer a convincing argument in favour of the LOLEX method.

5.5 LOLEX on axisymmetric liquid/liquid cases

With a good idea of the effects of reinitialization, further simulations were performed using 2D axisymmetry, so that the resulting data could be revolved around one axis to give 3D data. These simulations can be compared directly to experiments when the initial conditions and physical properties are matched. However, as it turns out, it is difficult to find experimental cases matching conditions that the present method is able to reliably simulate.

As stated previously, an attempt was made to perform axisymmetric simulations of liquid-in-gas cases, but due to the increased pressure from surface tension, these cases could not be simulated in a reasonable amount of time using the present

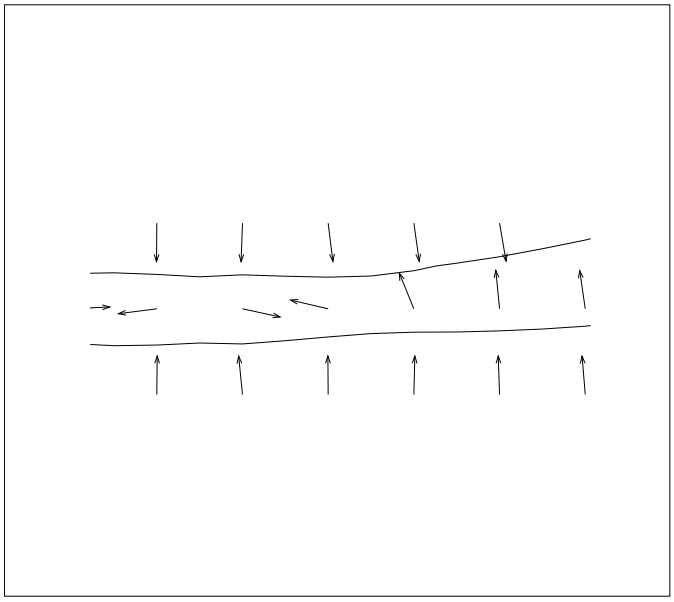


Figure 5.15: Close up of the interfaces in Figure 5.14 (b), with normal vectors also shown. It is seen that the interfaces are deformed due to the incorrect normal vectors.

method.

However, axisymmetric simulations were performed for several liquid-in-liquid cases. Among these are the most direct comparison to an experimental study, namely the water-in-decane case described by Chen et al.[3]. This case was set up to exactly match the experimental case, with settings as reported above in Table 5 on page 71 , except for the falling height. In this case, the water droplet was placed $2.0\Delta x$ above the pool, and allowed to merge with the pool due to the effects of gravity. This case has been simulated before using the present codes, where a specific distance from the droplet to the pool was set to minimize the errors caused by the standard curvature calculation method, such that the simulation was not significantly affected by the curvature errors. The results here were obtained without having to set a specific distance, and they are shown in Figure 5.16.

The results of this simulation are impressive in that they match experimental results very closely, as is shown in the following in Section 6.5 on page 88. It is also reassuring to see that both merging and splitting can be reliably simulated, as these are often-touted benefits of using the level-set method.

Additional simulations were performed using the same physical properties, but larger falling heights. Experiments corresponding to this are not reported in the literature, as far as the author is aware, but successful simulations of such cases indicate that the LOLEX method has opened the door for simulations of merging for liquid-

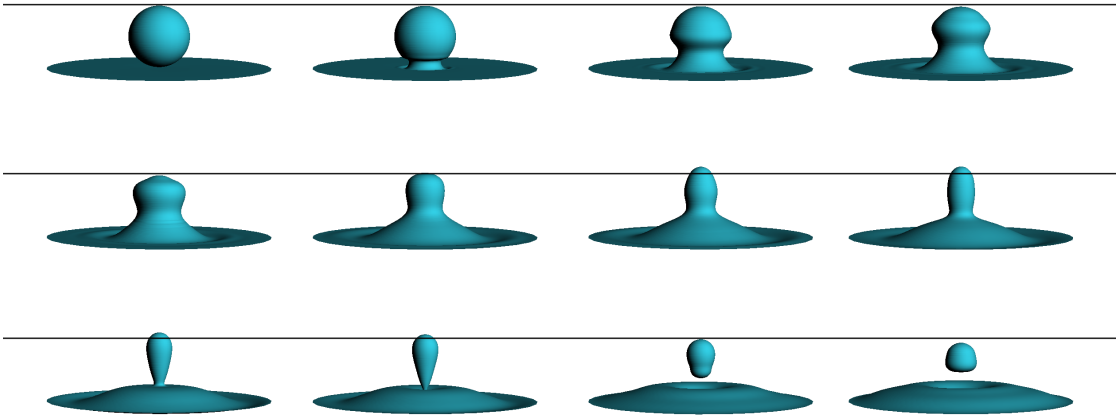


Figure 5.16: Axisymmetric simulation of a water droplet immersed in a decane-polybutene mix, merging with a deep pool of water. The capillary waves travelling up the bubble after merging are clearly seen. After some time, pinch-off occurs, and a new droplet is formed. The black lines indicate the top of the droplet just before merging.

in-gas cases, when the stability concerns detailed previously in Section 5.2 have been addressed. The results of this for a falling height of 1 droplet diameter are shown in Figure 5.17. For this case, no jetting occured, as the droplet velocity from falling caused the entire droplet to merge with the pool. The simulation can be continued for as long as one wants without problems.

While simulating this droplet falling onto the pool, a case was encountered in which the merging causes an unphysical pressure wave, and a subsequent jetting. As stated, this case is unphysical, but the result serves as an example indicating that some care must still be taken when simulating such collisions. In this case, the level-set reinitialization was only performed every 100 time steps, causing the problem. Reinitializing every 40 time steps instead avoids the problem. It is illustrated in Figure 5.18, where the pressure wave as well as the interface behaviour is shown. The morale here is that even though the present method is becoming fairly robust, the onus is still on the researcher to know when unphysical behaviour has occured.

Some experimental results were found in the literature concerning falling liquid drops in a liquid with subsequent merging with a deep pool. These results are due to Mohamed-Kassim and Longmire [29]. In these experiments, a droplet which is ten times larger in diameter than in the Chen et al. experiments was used. This droplet fell from a large height, giving it terminal speed before merging with the pool. The liquids used were slightly different from those used by Chen et al. due to the fact that

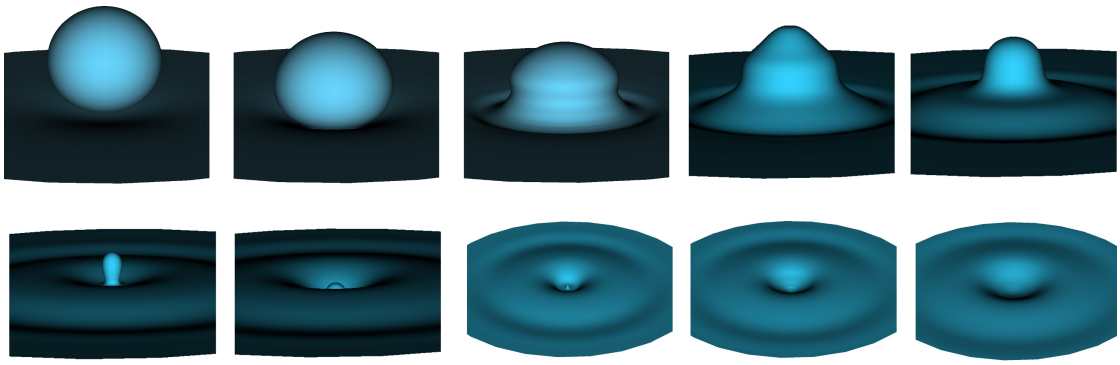


Figure 5.17: The same case as in the previous figure, but with a larger fall height. This droplet was started one droplet diameter above the pool. No pinch-off, as for the previous case, is seen. The perspective is altered in the final three frames in order to permit a view of all the surface.

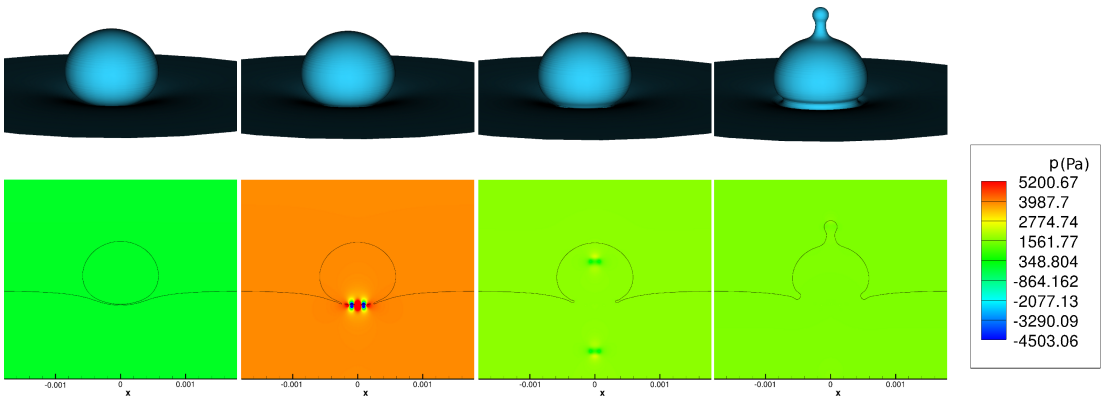


Figure 5.18: A droplet merging into a pool produces an unphysical pressure wave, causing the top of the droplet to jet upwards. The pressure field is shown in the lower half of the figure, where a wave originating at the time of merging can be seen.

Mohamed-Kassim and Longmire used an experimental technique where the refraction indices of the two liquids had to be closely matched. The details of the case are given in Table 6. An attempt was made to simulate this case; the results are shown in Figure 5.19, where the interfaces in a slice through the droplet and pool are shown.

The result of this simulation is that the droplet merges with the pool much earlier than seen in experiments. The experimental results are shown in Section 6.5, Figure 6.7. This discrepancy is probably due to the effects of the thin film between the droplet and the pool being underestimated in simulations. In experiments, the width of this

Table 6: Table of physical properties and numerical settings used when attempting to simulate the Mohamed-Kassim and Longmire case.

Property	Value	Unit
Domain size	2.75×6.10	cm^2
Droplet diameter	1.1	cm
Fall height	2.35	cm
Water-Glycerine		
ρ	1128	kg/m
μ	$6.3 \cdot 10^{-3}$	Pa s
Mineral oil		
ρ	949	kg/m
μ	$19 \cdot 10^{-3}$	Pa s
Surface tension	$29.1 \cdot 10^{-3}$	N/m
Gravity	$-9.81\hat{y}$	m/s^2
Numerical settings		
Grid size	245×546	
Poisson solver	ILU-BCGS	
Poisson solver residual	$1.0 \cdot 10^{-12}$	
CFL number	0.7	
Level-set reinitialization	Every 40 time steps	
Maximum pseudo-time τ	12.0	
HCR-2 reinitialization	On	
Reinitialization CFL number	0.8	
Reinitialization discretization	WENO	
Level-set advection discretization	First-order upwind	

film is very thin; Mohamed-Kassim and Longmire are unable to determine exactly how thin it is, as it is thinner than 100 μm which is the limit of their camera resolution. The grid used here is almost of the same resolution, with $\Delta x = 112\mu$m. Increasing the number of grid cells by an order of magnitude or more is unfeasible in the scope of this work. Another effect that could contribute in this case is that for very thin films, the continuum description afforded by the Navier-Stokes equations is no longer a good model. It should be noted, however, that the first two frames in Figure 5.19 agree well with the first two frames in Figure 6.7 on page 91. This indicates that with a realistic model for thin film effects and a refined grid, one could hope to get good results for this case.

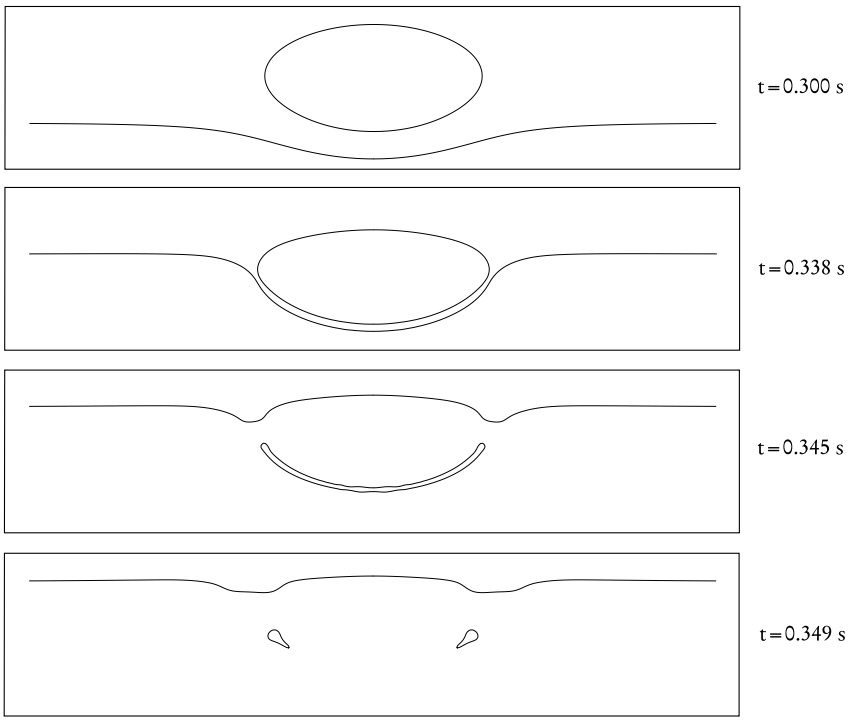


Figure 5.19: Attempt at simulating the droplet-pool case studied by Mohamed-Kassim and Longmire. In this case, the droplet is ten times larger than in the previous simulations reported. This simulation does not agree with experiments, as the merging here happens much earlier than in experiments.

§6 Comparison to experimental data

Contents

> It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong.
>
> Richard Feynman

6.1 Introduction

As seen in the previous chapter, numerical simulations of two-phase flow with merging is still difficult and computationally intensive. The experimental results available, by contrast, are numerous and accurate.

The first class of experiments discussed here come from the experimental group at SINTEF Energy Research and NTNU's Department of Energy and Process Engineering, particularly from the PhD thesis and postdoctoral work by He Zhao[52]. These experiments have attempted to characterize the different flow regimes that are interesting in the context of LNG condensation. The experiments have been performed with various liquids falling through various gases. These experiments are relevant for the methanol-in-air simulations reported above.

The second class of experiments is due to Chen et al. [3] and Mohamed-Kassim and Longmire [30]. These experiments consider fluid-fluid interfaces between water and various light fluids, e.g. decane. Drops of water were made to rest on a pool of water, with a thin film of light fluid between, and subsequent mergings were recorded. These experiments were the inspiration for the water-in-decane simulations above. In the zero-impact-velocity cases, a partial coalescence phenomenon was reported which was self-affine in nature, with partial droplets splitting off and remerging up to eight times [30]. Mohamed-Kassim and Longmire also consider the case where a droplet falls through the ambient liquid before merging with an interface [29].

The experimental techniques used for studying droplet-pool impacts include high-

speed shadowgraphy using a laser as light source, ordinary high-speed photography, and particle image velocimetry (PIV) measurements which can determine the fluid flow inside droplets.

The purpose of this chapter is to briefly describe the main situations that are observed experimentally, and to compare with the current numerical simulations where possible.

6.2 Methanol droplet merging with pool

THE SIMPLEST METHANOL DROPLET EXPERIMENTS presented here is that of droplets merging completely with the pool of liquid. This corresponds to the numerical simulations of methanol in air presented in the previous chapter. According to the flow characterization in [52, Chapter 5.1], complete merging is the expected behaviour for the velocities considered in the numerical simulations presented previously. The closest experimental result considered in [52] to the numerical simulation is with a 0.3 mm diameter methanol droplet, impacting the pool at 2.2 m/s. This is a smaller droplet, but it travels faster, and it has a higher kinetic energy. This droplet has $\mathrm{Re} \approx 930$ and $\mathrm{We} \approx 510$. Part of the experimental observations of this droplet are shown in Figure 6.1. Comparing this to Figure 5.8, the results look somewhat similar. Since the numerical simulation droplet is moving slower, the time between images is larger, but the qualitative agreement looks good. In particular, the fact that the top of the droplet is essentially undisturbed both in the fifth frame in Figure 5.8 and in the middle frame in Figure 6.1 is encouraging. The formation of a surface wave also seems to be similar.

To further study the agreement of the simulations with experimental results, a simulation should be performed with exactly the same parameters as used in this experiment. The reason why this has not been done presently, is that the time it would take to simulate a fall giving a 2.2 m/s collision velocity is much too large to be feasible using the present code. It is not possible to start the droplet close to the interface with the correct, large velocity field, so the droplet must be started at a large height above the interface and then fall under gravity. This would also require that the stability problems described previously are addressed.

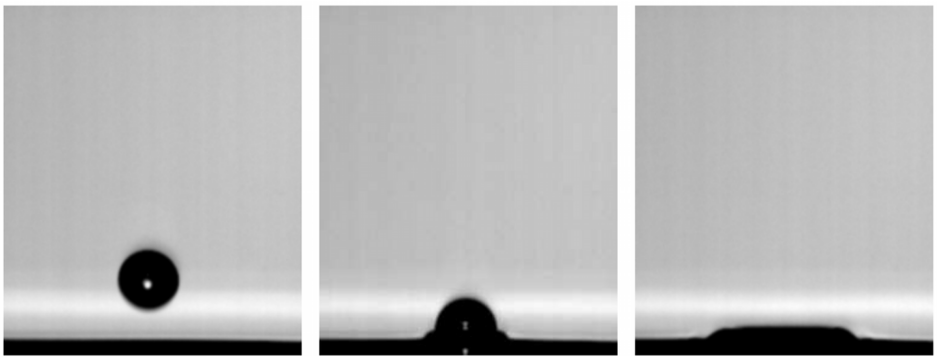


Figure 6.1: High-speed imagery of a 0.3 mm diameter methanol droplet merging completely with a pool of methanol at a velocity of 2.2 m/s. Figure from [52, Figure 5.11].

6.3 Methanol droplet partially merging with pool

All numerical simulations presented in this work for liquid-in-gas cases end up with the droplet merging completely with the pool. Experimentally, however, a well-defined range of initial conditions give rise to partial merging, either with the drop pinching off during merging, or with jetting from the waves induced in the pool. Reproducing either of these two scenarios for a liquid drop falling from a non-zero height through a gas would be a major milestone. The experimental results presented here are intended to demonstrate how these phenomena look, giving a clearer view of future hopes and prospects.

The first sub-case of partial merging, which is pinch-off of the merging drop, happens for flow regimes where the kinetic energy is not very high. This suggests that simulating such cases should not be very demanding. However, the experimental results showing this type of merging are for fluids like water, which have high surface tension. This is a complicating factor in the numerical simulations, as higher surface tension is more demanding to simulate numerically. It also produces a higher pressure inside the droplet, so the situation where the drop merges with the pool becomes more demanding to simulate as well. With sufficiently small time steps, such simulations may be carried out, but it would be preferable to find experimental cases for pinch-off with fluids like methanol or *n*-pentane. An experimental case with a pinch-off is shown in Figure 6.2. In this case, a 0.12 mm water droplet impacts with a pool of water at 0.29 m/s. A simulation like this could probably be carried out if the obstacles presented in

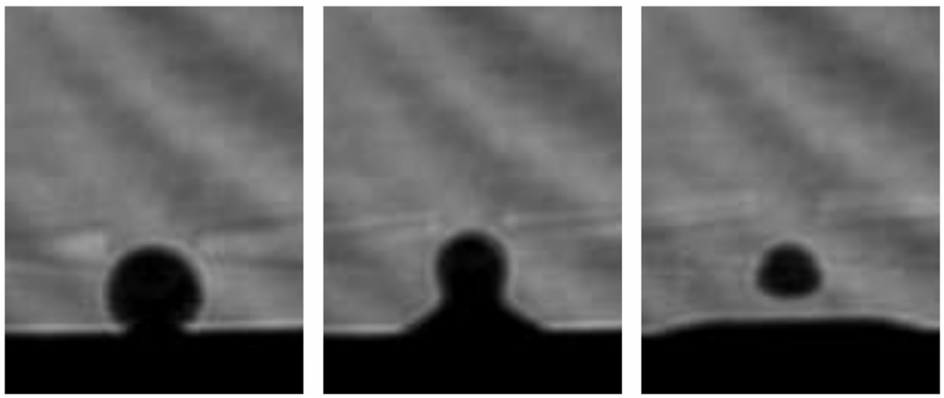


Figure 6.2: High-speed imagery of a 0.12 mm diameter water droplet merging partially into a pool of water, at a velocity of 0.29 m/s. The droplet pinches off, and leaves a smaller droplet behind. Figure from [52, Figure 5.17].

the previous chapter are overcome, and would be a very nice result if the numerical simulations mathced the experimental results.

The second sub-case of partial merging is called jetting, where the droplet first merges completely with the pool, and a jet then emerges from the pool. Such cases happen for much higher kinetic energies than the first sub-case, and will be more demanding to simulate numerically, not only because of the long fall times needed but also because the velocities involved are one or two orders of magnitude larger than for the numerical simulations presented here. The experimental result presented in Figure 6.3 showing such a case is for a 0.26 mm diameter n-pentane droplet impacting a pool of n-pentane at 5.9 m/s. Note that in this figure, a significant time period has passed between the first and the second row of images. During this period, there is a lot of motion inside the pool of liquid n-pentane, but this is not visible to the high-speed camera, so these removed frames are not very interesting. The first row of frames indicates that this collision is pretty violent, with complex phenomena occuring at small spatial scales, so a very fine grid (or adaptive grid refinement) would be necessary to capture this behaviour.

Another aspect which could perhaps become important is that the gas is treated as incompressible in the present work. It has been shown experimentally [10] that in some cases of jetting where an object collides with a liquid pool at high Reynolds number, a supersonic shock can occur in the gas even when the object is only falling at 2 m/s. This can obviously not be accurately modeled using an incompressible treatment of the gas.

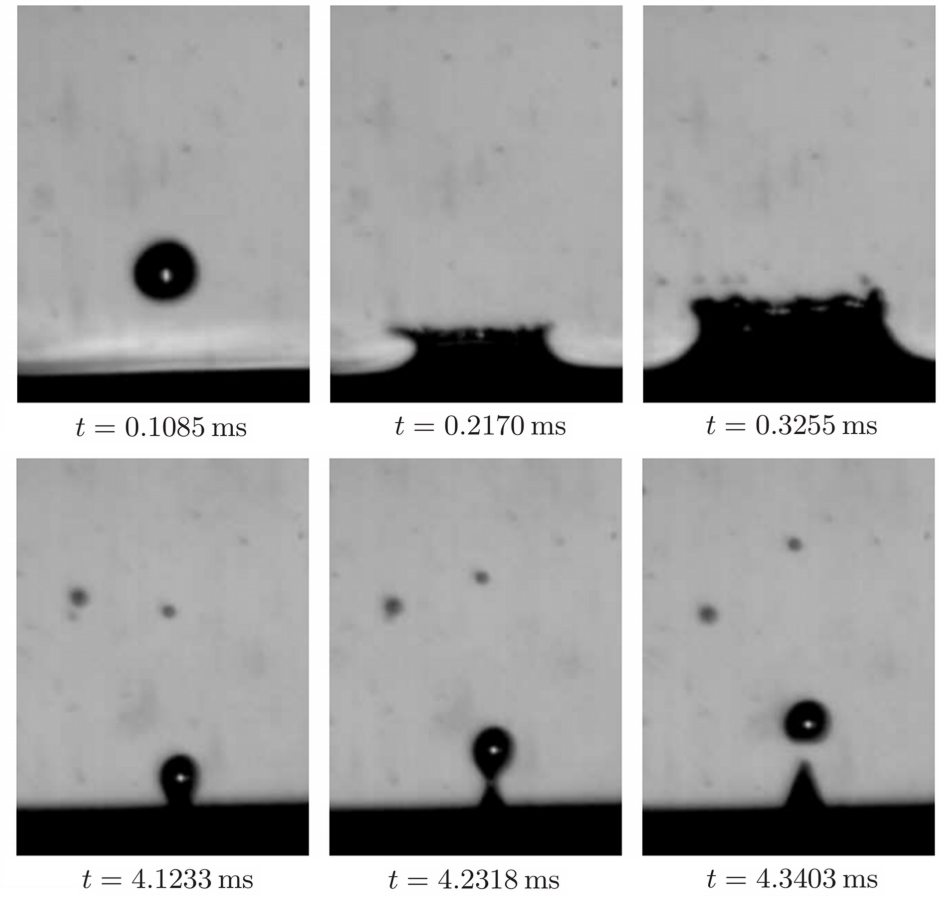


Figure 6.3: High-speed imagery of a 0.26 mm diameter n-pentane droplet impacting a pool of n-pentane at 5.9 m/s. After some time, the surface of the pool rebounds and ejects a jet. Note that some time has passed between the first and the second row of images. Figure from [52, Figure 5.6].

6.4 Methanol droplet bouncing off the pool

THE FINAL EXPERIMENTAL CASE presented here for methanol-in-air is the droplet bouncing off the pool. The author is unsure whether this result can be replicated accurately using the present methods, because this phenomenon requires a large surface tension, and because this phenomenon is believed to be greatly dependent on the properties of the surrounding gas, which is not very accurately modeled

at present. There is also no model for the surface film behaviour in the present codes.

Pan and Suga [35] note that their simulations of bouncing agree with experiments at high Weber numbers (i.e. We $\gtrsim 1$), but fail to do so at low Weber numbers, where the physics in the gas film between colliding objects is no longer accurately modeled by the continuum formulation used in the Navier-Stokes equations. The present case has We ≈ 10, so if the same criterion holds for the droplet-pool collisions here as for the droplet-droplet collisions studied by Pan and Suga, one would expect to get decent results in this case.

In addition to this, the gas film between the droplet and the pool is very thin, and a very fine grid will be required to model this accurately. This last obstacle could be partially overcome by using a non-uniform grid, which is possible in the present codes.

In spite of these challenges, the experimental result is still presented here. In this case, the droplet and pool contained 1-propanol, and the 0.24 mm diameter droplet impacted the pool at 1.14 m/s. The velocity of the droplet bouncing up again was 0.29 m/s, and the droplet emerges completely from the pool and hangs in the air for some time, before falling back down again. This is shown in Figure 6.4. Note that in this figure, several frames have been removed between the first and second picture in the second row; in these frames, not much is happening that is visible to the imaging apparatus. This experimental result is truly fascinating, in that the entire drop emerges back out of the pool after first having almost merged with it. A replication of this result numerically would be remarkable.

6.5

Decane droplet in water merging with decane pool

THE SECOND CLASS OF EXPERIMENTS consider two immiscible liquids, where a droplet of the heaviest liquid is placed in the lightest liquid above a pool of the heaviest liquid. In the most common experiments, the droplet is made to rest on the pool, and then merging happens after some time when the thin film between the droplet and the pool has drained sufficiently. There are also some experimental results due to Mohamed-Kassim and Longmire for a larger falling height[29], corresponding to the simulations in Section 5.3 with a substantial fall height.

The zero fall height simulations reported by Chen et al. [3] and by Mohamed-Kassim and Longmire [30] are described here first. In these experiments, a droplet of a heavy fluid is gently placed on a flat interface between the same heavy fluid (bottom)

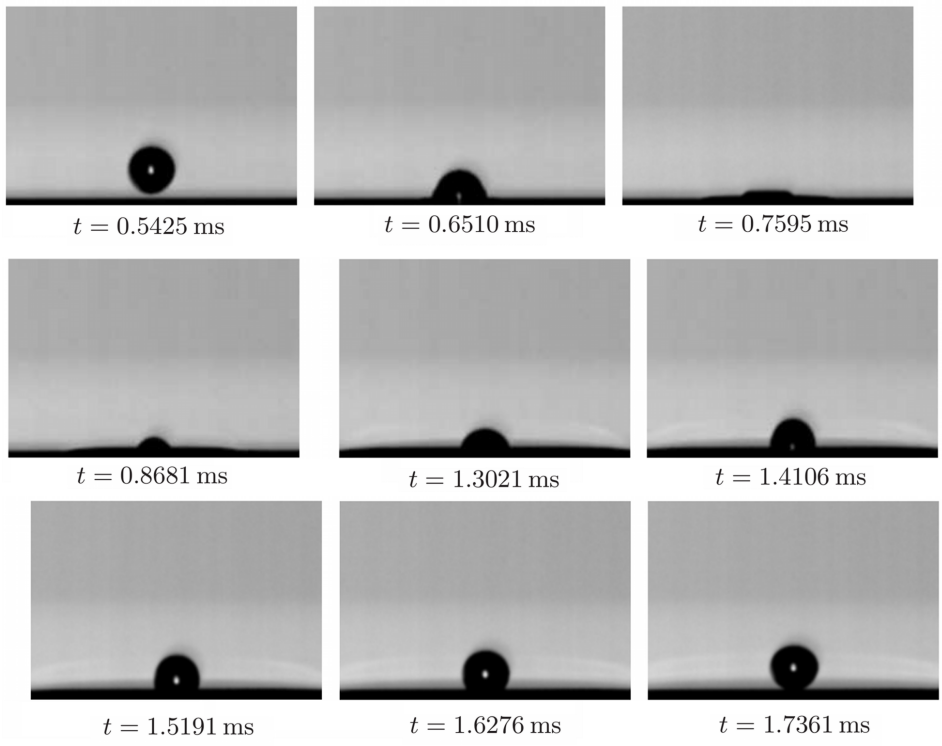
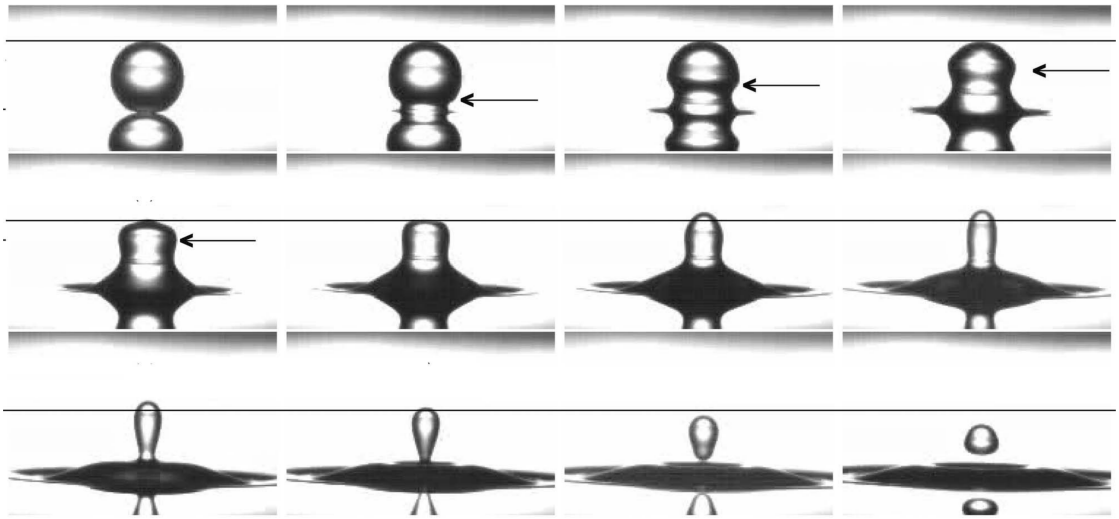


Figure 6.4: High-speed imagery of a 0.24 mm diameter 1-propanol droplet impacting a 1-propanol pool at 1.14 m/s, and bouncing back again. Note that several frames have been removed between the first and second frames in the second row, as indicated by a larger gap. Figure from [52, Figure 5.14].

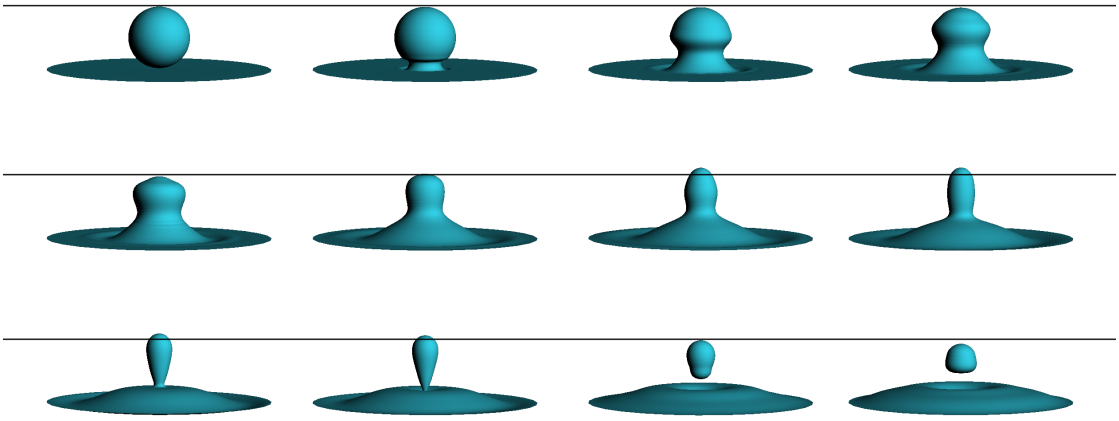
and a light fluid (top). As gravity affects the droplet, a thin film is formed between the droplet and the interface. This thin film drains, and after some time the film ruptures and the droplet merges with the interface.

The point of merging varies between the Chen et al. experiments and the Mohamed-Kassim and Longmire experiments. The former report merging at the central point, while the latter report an off-axis merging. This is probably because Mohamed-Kassim and Longmire consider larger droplets, and also because the tracer particles used in their PIV method are randomly distributed, and may affect the rupture of the thin film.

In the present simulations, the point of merging is decided mainly by grid effects when the droplet deforms the interface forming a thin film. It would be desireable to have a better model of the thin film effects, which some authors have noted may deviate from that predicted by capillary wave theory for fluids like decane[28]. A model for the influence of surfactants at the interface is available in the codes used



(a) Experimental result



(b) Simulation result

Figure 6.5: A water droplet merging with a water pool. The surrounding fluid is 20% polybutene in decane. Snapshots are taken 542 μs apart, the arrow indicates the capillary wave. Figure (a) is the experimental result from [3], Figure (b) is the simulation result presented previously in Section 5.5.

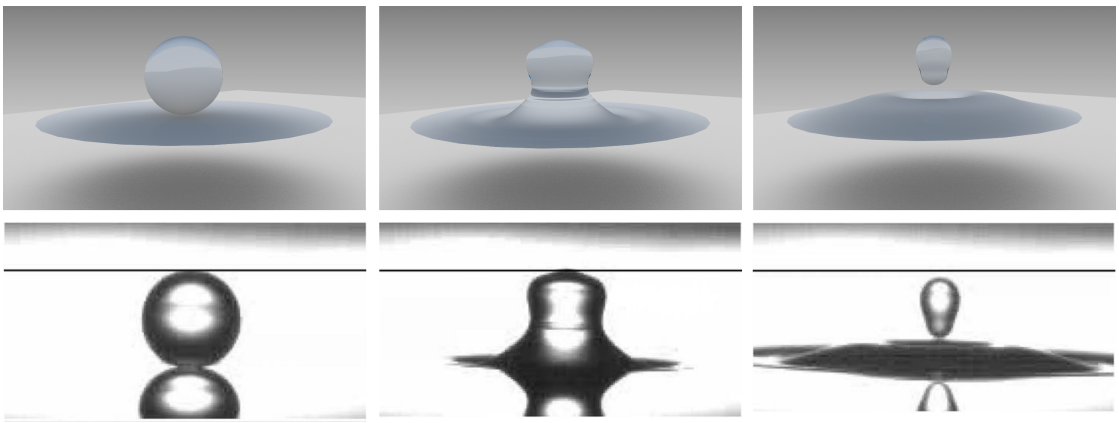


Figure 6.6: Closer comparison of simulation and experimental results for frames 1, 5 and 11 of Figure 6.5. In this figure, the simulation results have been raytraced to give the surface the physical appearance of water. Some small differences can be seen; this is due to the fact that experimental images and simulation frames are not taken at exactly the same time.

here, but has not been used in the present work. Experimental investigations have shown that surfactant effects can influence coalescence phenomena significantly[16]; the present results correspond to no surfactants, but adding surfactants can stabilize the physical situation[16].

With the present methods, we must simply hope that precisely what happens at the time of merging does not significantly affect the following behaviour. Comparing Figure 5.16, shown here again in Figure 6.5 (b), with Figure 6.5 (a) indicates that at least in this case, the numerics and experiments agree very nicely. This result has been produced previously using the present codes, by setting a specific initial distance from the droplet to the pool, such that merging starts quickly, and the curvature error becomes small enough not to significantly affect the result[48]. The present result is obtained without having to do this, and the fact that results calculated using the LOLEX method agree closely with experimental results is very reassuring.

In Figure 6.5 (b), the physical constants, droplet diameter etc. are all set to match those reported in [3]. In particular, the droplet diameter is 1.1 mm. For three of the frames in Figure 6.5, namely frame 1, 5 and 11, a closer comparison of the experimental and simulation result is shown in Figure 6.6. In this figure, the simulation results have been raytraced using the optical properties of water. This has not been done for all frames, since the rendering is fairly time consuming; the time to raytrace these three frames was a little over 2 hours, excluding the time spent importing the surface into the raytracing program and setting up the rendering.

In the experiments in [30], a 10 times larger droplet diameter is used than in the

experiments by Chen et al., as well as slightly different fluid properties. This leads to a situation where the droplet deformes the interface and rests on it for some time before merging. A simulation to compare with this was not performed here. Instead, an attempt was made to simulate the case in [29], where the same large droplet diameter is considered, and the droplet is also made to fall some distance before merging with the interface. This situation is more similar to the liquid-in-gas cases that are interesting in the study of LNG condensation than the cases where the drop is placed gently on the interface.

The results of this simulation are shown in Figure 5.19, and compared with the experiments in Figure 6.7. In experiments, the droplet deformed the interface significantly before merging, and the droplet rests on the pool indefinitely wihtout merging. In simulations, the merging happens at an early point in time during the initial deformation of the pool. This indicates that such a case, with a large falling droplet, cannot be accurately modeled using the present method. This is probably due to the thin film effects being very important in this case, as well as any surfactants. As these effects are not modelled here, the results do not agree.

6.6 Concluding remarks

THE EXPERIMENTAL RESULTS presented in this section span a range of initial conditions that is currently larger than that available to the present method. Contrasting with this is the ultimate, perhaps optimistic goal of the numerical effort: to replace experimental work, which is costly and time-consuming, requires attention to HSE for many working fluids, and provides less detailed data about what is going on (particularly inside the fluids). The experimental results that have been presented here represent both short- and long-term goals for the numerical simulations to reach for, and it will be interesting to see how far it is possible to go using the present methods.

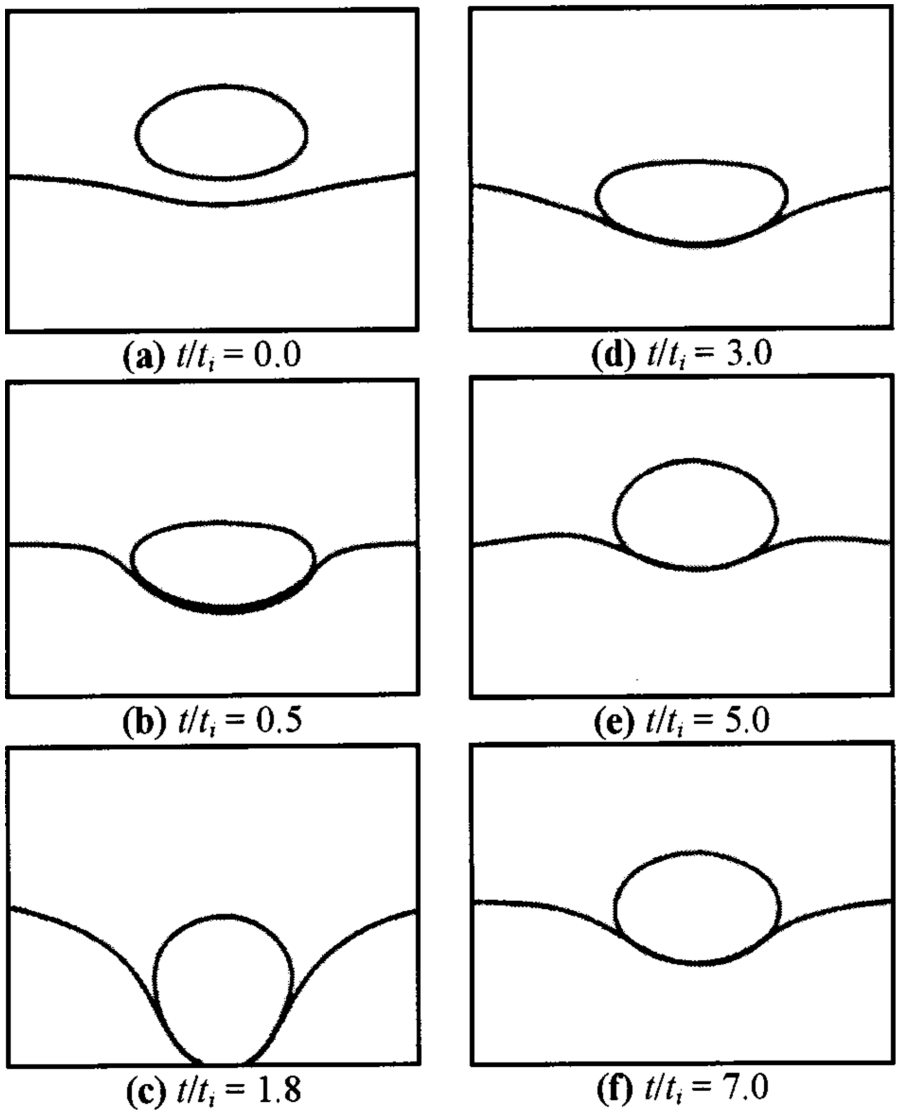


Figure 6.7: Snapshots of a droplet of water and glycerin falling through a mineral oil, and then colliding with a pool. The droplet deforms the pool significantly, before the pool bounces back up again. The droplet does not merge with the pool. Figure taken from [29].

§7 Concluding remarks and future prospects

7.1 Conclusions

In this thesis, the LOLEX method for robust calculation of geometric quantities has been introduced. The method is used here in simulations of two-phase fluid flow relevant for understanding LNG condensation, e.g. a droplet colliding with a deep pool, but the method is generally applicable to all level-set simulations where curvature and normal vectors are calculated while the interface topology changes.

Introducing the problem at hand, the theory behind two-phase flow simulations and the level-set method has been reviewed. The LOLEX method introduced here has been motivated in that it is more general than earlier methods, and can easily be extended to higher dimensions. The method has been described in detail; briefly, it consists in extracting one or more local level sets for bodies close to the grid point considered, reinitializing these local level sets to remove kinks, and using these to calculate the curvature and normal vector at the grid point considered. For the curvature, multiple values are averaged, while for the normal vector, the one corresponding to the closest interface is selected.

Following the description of the method, it has been verified using a number of geometric test cases with static interface configurations. The method is seen to perform significantly better than the standard method, with curvature errors of around 5 % in the worst case, where the standard method gives errors of $\mathcal{O}(1/\Delta x)$ or up to several 1000%. The error in normal vectors is too small to be accurately quantified. The LOLEX method is also applied to the 3D test case of a droplet close to a flat interface, in order to demonstrate that the method can easily be extended to 3D. The results in this 3D case were obtained using a proof-of-concept code which was implemented in 4 days.

After the verification of the LOLEX method on geometric test cases, physically interesting simulations are considered. Both a liquid-in-gas case, using methanol and air, and several liquid-in-liquid cases are considered. The liquid-in-gas case is seen to suffer from instabilities, particularly in the pressure, and while some results can be obtained, the accuracy of these is unknown. The liquid-in-liquid cases considered, on the other hand, suffer no stability problems, and the results can be trusted to a larger degree. Axisymmetric simulations of a water droplet in decane merging with a pool of water are performed, and the results match experimental observations closely. Simulations are also performed of a falling droplet merging with a pool; these cases are relevant

to the study of LNG condensation phenomena, but the simulations performed here cannot be directly compared with experiments in the literature. In particular, for large droplets, the thin film effects that occur just before merging are very important. These effects are not accurately modeled in the present codes, and the finest grid available for a feasible computation time is still too coarse to replicate the thin film widths seen in experiments. For small falling droplets, no experimental results for liquid-in-liquid cases could be found in the literature.

Finally, a review of relevant experimental cases from the literature is given. These include experiments with methanol droplets falling through air, as in simulations, but the Reynolds numbers attained in experiment are too high to be reliably simulated within a reasonable timeframe. Experiments are also reported for liquid-in-liquid cases; here, the simulations and experiments agree closely for small droplets where thin film effects are unimportant.

7.2 Future prospects

AT PRESENT, it seems the LOLEX method is the only general method for improved calculation of geometric quantities that also scales easily to 3D. As has been clearly indicated by the results presented here, the improvement offered by the method is significant, enabling new simulations that were previously unavailable for numerical study with the present codes. When the stability concerns outlined previously have been adressed, the LOLEX method is set to enable detailed simulations of coalescence phenomena that are highly interesting when attempting to understand the complex phenomena occuring in natural gas liquefaction.

Some recommendations are made here for future work on the present codes. As the LOLEX method solves the problems with erroneous curvatures and normal vectors, the main obstacle to simulations of liquid-in-gas cases is the stability of the method. For large density differences, this is still an active area of research, and it is not known what the best solution to this problem is. One possible approach is a hybrid level-set volume-of-fluid method, as has recently been demonstrated with good results by Sussman et al. [43].

An approach which has not been considered in this thesis is the use of filtering. It is possible that e.g. a Gaussian filter can be applied in some way to make the curvature less noisy. This could be particularly interesting for small separations between two bodies, when the curvature is the most noisy.

Improvements can still be made to the LOLEX method as well. The error in curvature reported here is up to 5 %, and while this is significantly better than the standard method in problematic areas, it is still an error. One possible solution to

this is an improved reinitialization procedure on the local grid, maybe in combination with an improved extrapolation method. This will perhaps be considered in future work.

A further recommendation is to consider solving the level-set equations on a finer grid than that of the Navier-Stokes equations [15]. In typical simulations, the level-set reinitialization and advection take only a fraction of the total time per time step, so it could be possible to increase the resolution of the interface capturing method without adding significantly to the total simulation runtime.

Given the overall present state of the codes used here, in which the LOLEX method has been implemented, and the ongoing effort to increase the stability of the codes, the author is looking forward to seeing increasingly complex simulations of coalescence phenomena; such simulations will perhaps reveal more of Mother Nature's secrets.

References

[1] Adalsteinsson, D. & Sethian, J. A. The fast construction of extension velocities in level set methods. *Journal of Computational Physics* **148**, 2 – 22 (1999). Available online.

[2] Adalsteinsson, D. & Sethian, J. A. A fast level set method for propagating interfaces. *Journal of Computational Physics* **118**, 269 – 277 (1995). Available online.

[3] Chen, X., Mandre, S. & Feng, J. J. Partial coalescence between a drop and a liquid-liquid interface. *Physics of Fluids* **18**, 051705 (2006). Available online.

[4] Chopp, D. L. Computing minimal surfaces via level set curvature flow. *Journal of Computational Physics* **106**, 77 – 91 (1993). Available online.

[5] Denaro, F. M. On the application of the Helmholtz–Hodge decomposition in projection methods for incompressible flows with general boundary conditions. *International Journal for Numerical Methods in Fluids* **43**, 43–69 (2003). Available online.

[6] Ervik, Å. Curvature calculation for two-phase fluid interfaces using the level-set method. Project Report, Norwegian University of Science and Technology (NTNU) (2012). Available online.

[7] Fedkiw, R. P., Aslam, T., Merriman, B. & Osher, S. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the Ghost Fluid Method). *Journal of Computational Physics* **152**, 457 – 492 (1999). Available online.

[8] Fedkiw, R. P. & Liu, X. D. The Ghost Fluid Method for viscous flows. Presented at the "Solutions of PDE" Conference in honour of Prof. Phil Roe (1998). Available online.

[9] Flynn, P. & Jain, A. On reliable curvature estimation. In *Computer Vision and Pattern Recognition, 1989. Proceedings CVPR '89., IEEE Computer Society Conference on*, 110 –116 (1989). Available online.

[10] Gekle, S., Peters, I. R., Gordillo, J. M., van der Meer, D. & Lohse, D. Supersonic air flow due to solid-liquid impact. *Phys. Rev. Lett.* **104**, 024501 (2010). Available online.

[11] Gisvold, M. Eventyret på Melkøya. *Gemini Research Magazine* (2004). Avaliable online.

[12] Han, S.-I., Stapf, S. & Blümich, B. NMR imaging of falling water drops. *Phys. Rev. Lett.* **87**, 144501 (2001). Available online.

[13] Hansen, E. B. *Numerical simulation of droplet dynamics in the presence of an electric field*. Ph.D. thesis, NTNU (2005).

[14] Hartmann, D., Meinke, M. & Schröder, W. The constrained reinitialization equation for level set methods. *Journal of Computational Physics* **229**, 1514 – 1535 (2010). Available online.

[15] Herrmann, M. Refined level set grid method for tracking interfaces. Technical Report, Center for Turbulence Research (2005).

[16] Hodgson, T. & Lee, J. The effect of surfactants on the coalescence of a drop at an interface i. *Journal of Colloid and Interface Science* **30**, 94 – 108 (1969). Available online.

[17] Jiang, G.-S., Shu, C.-W. & L, I. Efficient implementation of weighted ENO schemes. *J. Comput. Phys* **126**, 202–228 (1996). Available online.

[18] Kang, M., Fedkiw, R. P. & Liu, X.-D. A boundary condition capturing method for multiphase incompressible flow. *Journal of Scientific Computing* **15**, 323–360 (2000). Available online.

[19] Ketcheson, D. I. & Robinson, A. C. On the practical importance of the SSP property for Runge–Kutta time integrators for some common Godunov-type schemes. *International Journal for Numerical Methods in Fluids* **48**, 271–303 (2005). Available online.

[20] Lervåg, K. Y. *Simulation of two-phase flows with varying surface tension*. Master's thesis, NTNU (2008). Available online.

[21] Lervåg, K. Y. Calculation of interface curvature with the level-set method. In *Sixth National Conference on Computational Mechanics MekIT'11* (Trondheim, Norway, 2011). Available online.

[22] Lervåg, K. Y. Calculation of the interface curvature and normal vector with the level-set method (2012). To be submitted.

[23] Lervåg, K. Y. & Ervik, Å. Curvature calculations for the level-set method. In *ENUMATH 2011 Proceedings Volume* (Leicester, England, 2011).

[24] Macklin, P. & Lowengrub, J. Evolving interfaces via gradients of geometry-dependent interior Poisson problems: application to tumor growth. *Journal of Computational Physics* **203**, 191 – 220 (2005). Available online.

[25] Macklin, P. & Lowengrub, J. An improved geometry-aware curvature discretization for level set methods: Application to tumor growth. *Journal of Computational Physics* **215**, 392 – 401 (2006). Available online.

[26] Mallet, V., Keyes, D. & Fendell, F. Modeling wildland fire propagation with level set methods. *Computers and Mathematics with Applications* **57**, 1089 – 1101 (2009). Available online.

[27] Melicher, V., Cimrak, I. & Keer, R. V. Level set method for optimal shape design of MRAM core. Micromagnetic approach. *Physica B: Condensed Matter* **403**, 308 – 311 (2008). Proceedings of the Sixth International Symposium on Hysteresis Modeling and Micromagnetics. Available online.

[28] Mitrinović, D. M., Tikhonov, A. M., Li, M., Huang, Z. & Schlossman, M. L. Noncapillary-wave structure at the water-alkane interface. *Phys. Rev. Lett.* **85**, 582–585 (2000). Available online.

[29] Mohamed-Kassim, Z. & Longmire, E. K. Drop impact on a liquid–liquid interface. *Physics of Fluids* **15**, 3263–3273 (2003). Available online.

[30] Mohamed-Kassim, Z. & Longmire, E. K. Drop coalescence through a liquid/liquid interface. *Physics of Fluids* **16**, 2170–2181 (2004). Available online.

[31] Newman, T. S. & Yi, H. A survey of the marching cubes algorithm. *Computers and Graphics* **30**, 854 – 879 (2006). Available online.

[32] Norwegian Petroleum Directorate. FactPages V.2 (2011). Available online.

[33] Osher, S. & Fedkiw, R. P. Level set methods: An overview and some recent results. *Journal of Computational Physics* **169**, 463 – 502 (2001). Available online.

[34] Osher, S. & Sethian, J. A. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics* **79**, 12 – 49 (1988). Available online.

[35] Pan, Y. & Suga, K. Numerical simulation of binary liquid droplet collision. *Physics of Fluids* **17**, 082105 (2005). Available online.

[36] Peng, D., Merriman, B., Osher, S., Zhao, H. & Kang, M. A PDE-based fast local level set method. *Journal of Computational Physics* **155**, 410 – 438 (1999). Available online.

[37] Pilliod, J. E., Jr. & Puckett, E. G. Second-order accurate volume-of-fluid algorithms for tracking material interfaces. *Journal of Computational Physics* **199**, 465 – 502 (2004). Available online.

[38] Rhie, C. M. & Chow, W. L. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal* **21**, 1525–1532 (1983).

[39] Rottmann, K. *Matematisk Formelsamling* (Spektrum Forlag, 2003).

[40] Salac, D. & Lu, W. A local semi-implicit level-set method for interface motion. *Journal of Scientific Computing* **35**, 330–349 (2008). Available online.

[41] Smereka, P. Semi-implicit level set methods for curvature and surface diffusion motion. *Journal of Scientific Computing* **19**, 439–456 (2003). Available online.

[42] Spiteri, R. J. & Ruuth, S. J. A new class of optimal high-order strong-stability-preserving time discretization methods. *SIAM Journal on Numerical Analysis* **40**, pp. 469–491 (2003). Available online.

[43] Sussman, M., Smith, K., Hussaini, M., Ohta, M. & Zhi-Wei, R. A sharp interface method for incompressible two-phase flows. *Journal of Computational Physics* **221**, 469 – 505 (2007). Available online.

[44] Sussman, M., Smereka, P. & Osher, S. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics* **114**, 146 – 159 (1994). Available online.

[45] Tamura, I., Tanaka, T., Kagajo, T., Kuwabara, S., Yoshioka, T., Nagata, T., Kurahashi, K. & Ishitani, H. Life cycle CO_2 analysis of LNG and city gas. *Applied Energy* **68**, 301 – 319 (2001). Available online.

[46] Tang, C.-K. & Medioni, G. Curvature-augmented tensor voting for shape inference from noisy 3D data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **24**, 858 –864 (2002). Available online.

[47] Tayler, A. B., Holland, D. J., Sederman, A. J. & Gladden, L. F. Exploring the origins of turbulence in multiphase flow using compressed sensing MRI. *Phys. Rev. Lett.* **108**, 264505 (2012). Available online.

[48] Teigen, K. E., Munkejord, S. T. & Bjørklund, E. A computational study of the coalescence process between a drop and an interface in an electric field. In *CFD2008 - 6th International Conference on Computational Fluid Dynamicics in the Oil and Gas, Metallurgical and Process Industries* (Trondheim, Norway, 2008).

[49] Thoroddsen, S. T., Takehara, K., Etoh, T. G., Popinet, S., Ray, P., Josserand, C., Zaleski, S. & Thoraval, M.-J. von Kármán vortex street within an impacting drop. *Phys. Rev. Lett.* **108**, 264506 (2012). Available online.

[50] van Vliet, L. J. & Verbeek, P. W. Curvature and bending energy in digitized 2D and 3D images. In *Proceedings of the 8th Scandinavian Conference on Image Analysis*, 1403–1410 (1993).

[51] Versteeg, H. & Malalasekera, W. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method (2nd Edition)* (Prentice Hall, 2007), 2 edn.

[52] Zhao, H. *An Experimental Investigation of Liquid Droplets Impinging Vertically on a Deep Liquid Pool*. Ph.D. thesis, NTNU (2009).

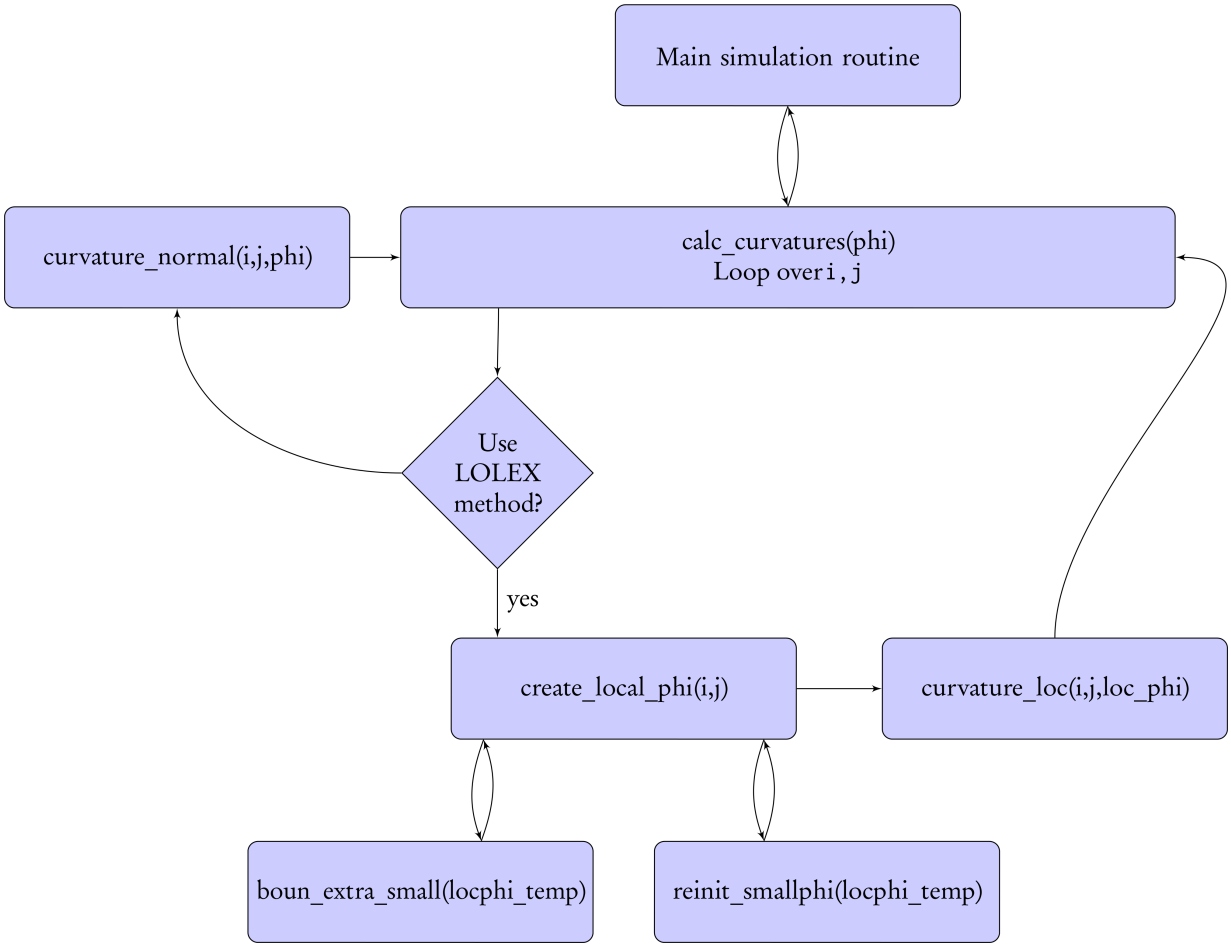


Figure H.1: Flowchart of curvature calculation routines.

Appendices

The routines presented here constitute the main routines of the current method. Their interdependence is shown in Figure H.1 for the case of curvature calculation. For the routines `boun_extra_small`, and `reinit_smallphi`, only interfaces are given (Listing 4) since these are standard numerical methods.

Appendix A
Flowchart for LOLEX curvature calculations

Appendix B
Main curvature calculation routine

Listing 1: Main routine

```
subroutine calc_curvatures(phi)
  ! Main curvature calculation routine
  implicit none
  real, dimension(ib1:ibn,jb1:jbn), intent(in) :: phi
  ! ib1=1-3, ibn=imax+3. 3 ghost cells are required by WENO-5.
  integer :: i,j
  !
  ! Initialize curvature field:
  w(:,:,nwcur)=0.0
  curv_max=0.0
  !
  ! When using the present method, we need to have values of phi beyond
  ! ib1:ibn etc.
  phi_larger(ib1:ibn,jb1:jbn)=phi !Module variable
  ! Extrapolate further to the ghost cells
  call boun_extra_large(phi_larger)
  !
  ! Calculate the curvature at all points on the grid close to an interface
  do j=1,jmax
    do i=1,imax
      if (.not. lcalc_all_curvs) then
        !
        ! Only calculate curvatures near interfaces
        if (sign(1.0,minval(phi(i-1:i+1,j-1:j+1))) &
            == sign(1.0,maxval(phi(i-1:i+1,j-1:j+1)))) cycle
      endif
      !
      if (maxval(w(i-1:i+1,j-1:j+1,nwlsq))>lls_eta) then
        !
        ! LOLEX method
        call create_local_phi(i,j)
        w(i,j,nwcur)=curvature_loc(i,j,loc_phi)
        !
      else
        !
        ! Ordinary method
        w(i,j,nwcur)=curvature_local(i,j,phi(i-1:i+1,j-1:j+1))
      endif
      !
      ! Store the max curvature for use e.g. in time-step estimation
      !
      curv_max=max(curv_max,abs(w(i,j,nwcur)))
    enddo
  enddo
end subroutine calc_curvatures
```

Appendix C
Main LOLEX routine

Listing 2: Routine that builds the local ϕ

```
subroutine create_local_phi(i,j)
  ! Create local level set functions at and around the cell (i,j)
  ! We create one for each body present in the domain.
  use grid, only: ilmax,jlmax,ilb1,ilbn,jlb1,jlbn,nbord
  implicit none
  integer, intent(in)   :: i,j
  !Local variables
  integer :: i0,j0,bodyno,neigh_bodies,pointcase,neighs,no,so,ea,we,&
             body,i1,j1,radi,bodypoints,j2
  integer, parameter :: unchecked=-10, nobody=-5, dep=1, indep=0, removed=-1
  integer, dimension(ilmax,jlmax) :: depend
  real, dimension(0:ilmax+1,0:jlmax+1) :: bodies
  real, dimension(-1:ilmax+2,-1:jlmax+2) :: lookphi
  real, dimension(ilb1:ilbn,jlb1:jlbn,loc_maxbodies) :: locphi_temp
  real :: s1,s2,t1,approx_d
  logical :: lremoved_body
  !-------------------------------------------------------------------------
  !
  loc_phi(:,:,:,:)=1e10 ! Module variable
  locphi_temp(:,:,:,:)=1e10
  !
  ! First of all, we need to determine how many bodies are present in the
  ! square surrounding our point (i,j).
  !
  ! lookphi is phi in the square we look at. radi is the "radius" of square
  radi=(ilmax-1)/2 + 2 ! + 2 gives us boundaries for phi outside 7x7 square
  lookphi(:,:)=phi_larger(i-radi:i+radi,j-radi:j+radi) ! Module variable
  !
  ! Bodyscan routine: creates bodies(:,:) array.
  ! Mark all cells as unchecked
  bodies(:,:) = unchecked
  ! bodyno is used to count body 1, body 2 etc.
  bodyno = 0
  lremoved_body = .false.
  do j0 = 0, jlmax+1
    do i0 = 0, ilmax+1
      if (bodies(i0,j0) == unchecked) then
        if (lookphi(i0,j0) < 0) then
          bodyno = bodyno + 1
          bodies(i0,j0) = bodyno
          bodypoints = 0
          ! Find all points in this body:
          call patdown(lookphi,bodies,bodyno,bodypoints,i0,j0)
          ! If this body has less than 25 points, forget it
          if (bodypoints < 25) then
            ! Remove all markings with bodyno in this square.
            !write(*,*) "Removing body of size", bodypoints
            do j1 = 0, jlmax+1
              do i1 = 0, jlmax+1
                if (bodies(i1,j1) == bodyno) then
                  bodies(i1,j1) = removed
                end if
              end do
```

```
            end do
            ! Decrement bodyno, so that this body is forgotten.
            bodyno = bodyno - 1
            ! Set boolean to say that a body has been removed
            lremoved_body = .true.
          end if
        else
          bodies(i0,j0) = nobody
        end if
      end if
    end do
  end do
  ! Store the number of bodies found
  loc_bodies = bodyno
  !
  ! Now we construct one phi for each body present.
  ! The next sub-task in this is to classify whether this point is
  ! "dependent" or "independent". We skip this if only one body has been
  ! found.
  !
  depend(:,:) = indep
  if (loc_bodies > 1 .or. lremoved_body) then
    do j0 = 1, jlmax
      do i0 = 1, ilmax
        ! Only check cells outside a body (excludes removed bodies as well)
        if (bodies(i0,j0) < -1) then
          ! Loop over all neighbours of this point. If we find more than one
          ! different positive value in the bodies(:,:) array, this point is
          ! dependent.
          neigh_bodies=0
          do j1 = max(1,j0-1), min(jlmax,j0+1)
            ! Skip centre point:
            if (j1 /= j0) then
              ! If this neighbour is inside a body (possibly a removed body):
              if (bodies(i0,j1) > 0 .or. bodies(i0,j1) == removed) then
                ! If we have already found another neighboring body:
                if (neigh_bodies /= 0 .and. bodies(i0,j1) /= neigh_bodies) then
                  ! Mark as dependent
                  depend(i0,j0) = dep
                end if
                neigh_bodies=bodies(i0,j1)
              end if
            end if
          end do
          ! Same procedure for i direction
          do i1 = max(1,i0-1), min(ilmax,i0+1)
            if (i1 /= i0) then
              if (bodies(i1,j0) > 0 .or. bodies(i1,j0) == removed) then
                if (neigh_bodies /= 0 .and. bodies(i1,j0) /= neigh_bodies) then
                  depend(i0,j0) = dep
                end if
                neigh_bodies=bodies(i1,j0)
              end if
            end if
          end do
        end if
      end do
    end do
  end if
  !
  ! For each body present, we now extract the relevant phi.
  !
```

```
      ! Set a sensible initial value of phi.
117   locphi_temp(:,:,:,:) = 2*dx
      !
119   ! Loop over the bodies present (main loop)
      do body = 1, loc_bodies
121     ! Loop over the 7x7 square, and copy phi where appropriate.
        do j0 = 1, jlmax
123       do i0 = 1, ilmax
            if (bodies(i0,j0) == body) then
125           ! Inside this body, copy
              locphi_temp(i0,j0,body) = lookphi(i0,j0)
127         elseif (bodies(i0,j0) == nobody .and. depend(i0,j0) == indep) then
              ! Outside bodies, and independent region, copy
129           locphi_temp(i0,j0,body) = lookphi(i0,j0)
              !
131         elseif (depend(i0,j0) == dep) then
              !
133           ! Explicit reconstruction following Adalsteinsson et. al.
              !
135           pointcase = 0
              approx_d = 0
137           ! find out which case this is
              neighs = 0
139           n = 0; s=0; e=0; w=0; ! north, south, east, west
              ! Check points north and south
141           do i1 = max(1,i0-1), min(ilmax,i0+1), 2
                if (bodies(i1,j0) == body) then
143               neighs = neighs + 1
                  if (i1 == max(1,i0-1)) then
145                 s = 1
                  else
147                 n = 1
                  end if
149             end if
              end do
151           ! Check points east and west
              do j1 = max(1,j0-1), min(jlmax,j0+1), 2
153             if (bodies(i0,j1) == body) then
                  neighs = neighs + 1
155               if (j1 == max(1,j0-1)) then
                    w = 1
157               else
                    e = 1
159               end if
                end if
161           end do
              ! Set pointcase
163           ! Case 1 (Case a)
              if (neighs == 1) then
165             pointcase = 1
              else if (neighs == 3) then
167             pointcase = 3
              else if (neighs == 2) then
169             if (n+e == 2 .or. n+w == 2 .or. s+e == 2 .or. s+w == 2) then
                  ! Corner case
171               pointcase = 2
                else
173               pointcase = 4
                end if
175           end if
              !
177           ! Compute the distance
```

```
            if (pointcase == 1) then
179           ! Set (i1,j1) to the relevant neighbour
              i1 = i0; j1 = j0;
181           if (n==1) i1 = i0-1
              if (s==1) i1 = i0+1
183           if (w==1) j1 = j0-1
              if (e==1) j1 = j0+1
185           ! The distance is given by this formula:
              approx_d = dx+lookphi(i1,j1)
187         else if (pointcase == 2) then
              ! find the relevant corner, extract data from this later
189           i1 = i0; j1 = j0;
              if (n==1) i1 = i0-1
191           if (s==1) i1 = i0+1
              if (w==1) j1 = j0-1
193           if (e==1) j1 = j0+1
              ! Corner is now given by (i,j).
195           ! Compute s1 and t1
              s1 = dx+lookphi(i1,j0)
197           t1 = dx+lookphi(i0,j1)
              ! Compute the distance given by Eq. (20) in report
199           if (s1**2 + t1**2 > 0) then
                approx_d = s1*t1/sqrt(s1**2 + t1**2)
201           else
                approx_d = 0
203           end if
            else if (pointcase == 3) then
205           ! Set (i1,j1) to the neighbour which is _not_ relevant
              i1 = i0; j1 = j0;
207           if (n==0) i1 = i0-1
              if (s==0) i1 = i0+1
209           if (w==0) j1 = j0-1
              if (e==0) j1 = j0+1
211           ! Go opposite side from (i1,j1) to find t1
              if (i1<i0) i1=i0+1
213           if (i1>i0) i1=i0-1
              if (j1<j0) j1=j0+1
215           if (j1>j0) j1=j0-1
              t1 = dx+lookphi(i1,j1)
217           ! Now find s1 and s2:
              if (i1/=i0) then
219             i1 = i0
                j1 = j0 - 1
221           else
                j1 = j0
223             i1 = i0 - 1
              end if
225           s1 = dx+lookphi(i0,j1)
              if (i1 /= i0) i1 = i0+1
227           if (j1 /= j0) j1 = j0+1
              s2 = dx+lookphi(i1,j0)
229           ! Take the minimum of s1 and s2 in s1, then use formula from case
              ! 2 above
231           s1 = min(s1,s2)
              if (s1**2 + t1**2 > 0) then
233             approx_d = s1*t1/sqrt(s1**2 + t1**2)
              else
235             approx_d = 0.0
              end if
237         else if (pointcase == 4) then
              i1 = i0; j1 = j0;
239           ! Set (i,j) to one of the interesting neighbours, the other will
```

```
              ! be on the opposite side
              if (n==1) i1 = i0-1
              if (w==1) j1 = j0-1
              ! Calculate s1 and s2, take their minimum.
              s1 = lookphi(i0,j0)/(lookphi(i0,j0)-lookphi(i1,j0))
              if (i1 /= i0) i1 = i0+1
              if (j1 /= j0) j1 = j0+1
              s2 = lookphi(i0,j0)/(lookphi(i0,j0)-lookphi(i1,j0))
              approx_d = min(s1,s2)
            else
              write(*,*) "Error: independent point was marked as dependent"
              approx_d=0.5
            end if
            !
            ! Set the approximate distance as the value in localphi
            !write (*, *) "Distance set to", approx_d*dx(1)
            locphi_temp(i0,j0,body) = approx_d*dx(1)
          else if (bodies(i0,j0) == removed) then
            ! A body has been removed. This value will be overridden by reinit
            ! anyway, so the default 2*dx(1) is okay. Do nothing.
          else
            ! This means that we hit a point belonging to the other body. Do
            ! nothing.
          end if
        end do
      end do
      !
      ! Extrapolate and reinitialize the body
      if (loc_bodies > 1 .or. lremoved_body) then
        call boun_extra_small(locphi_temp(:,:,body))
        locphi_temp(:,:,body)=locphi_temp(:,:,body)+0.8*dx(i)
        call reinit_smallphi(locphi_temp(:,:,body))
        locphi_temp(:,:,body)=locphi_temp(:,:,body)-0.8*dx(i)
      end if
    end do
    !
    ! After finishing this, we do not need the full square any more. The
    ! kinks in phi have been removed, so we can revert to 3x3.
    xlmin=(ilmax-1)/2
    ylmin=(jlmax-1)/2
    loc_phi(:,:,:) = locphi_temp(xlmin:xlmin+2,ylmin:ylmin+2,1,:)
    !
    ! If we have removed down to 0 bodies, we return the central part of the
    ! global phi instead.
    if (loc_bodies == 0) then
      loc_phi(:,:,1,1) = lookphi(xlmin:xlmin+2,ylmin:ylmin+2)
    end if
    !
  end subroutine create_local_phi
```

Appendix D
patdown routine

Listing 3: Marking a body

```
recursive subroutine patdown(phi,bodies,bodyno,bodypoints,i0,j0)
  ! Input variables
  use grid, only: ilosamax,jlosamax
  implicit none
  real, dimension(-1:ilosamax+2,-1:jlosamax+2,1), intent(in) :: fi
  integer, dimension(0:ilosamax+1,0:jlosamax+1), intent(inout) :: bodies
  integer, intent(inout) :: bodyno, bodypoints
  integer, intent(in) :: i0, j0
  ! Local variables
  integer :: i,j
  integer, parameter :: unchecked=-10
  !
  ! Increment our counter of points in this body
  bodypoints = bodypoints + 1
  ! Loop over four closest neighbours of (i0,j0):
  i=i0
  do j = max(0,j0-1), min(jlosamax+1,j0+1)
      ! Skip the centre of the square:
      if ( j /= j0) then
        !
        ! If this cell has not been checked before, and if fi < 0
        if ( bodies(i,j)==unchecked .and. fi(i,j,1) < 0.0 ) then
          !
          ! The cell (i,j) is interesting.
          ! Mark this cell as belonging to body no. bodyno
          bodies(i,j) = bodyno
          !
          ! Call this subroutine again on this interesting cell
          call patdown(fi,bodies,bodyno,bodypoints,i,j)
          !
        end if
        !
      end if
  end do
  j=j0
  do i = max(0,i0-1), min(ilosamax+1,i0+1)
      if ( i /= i0) then
        if ( bodies(i,j)==unchecked .and. fi(i,j,1) < 0.0 ) then
          bodies(i,j) = bodyno
          call patdown(fi,bodies,bodyno,bodypoints,i,j)
        end if
      end if
  end do
end subroutine patdown
```

Appendix E
Interfaces to routines not listed

Listing 4: Interfaces to routines not listed in full here

```
interface
  subroutine boun_extra_small(b)
    ! Extrapolate the scalar b from inner domain to the ghost cells
    ! across the boundary.
    ! This version has inner domain 1:ilmax and total domain ilb1:ilbn. It
    ! extrapolates to zeroth order, i.e. just copies outwards.
    implicit none
    real, dimension(ilb1:ilbn,jlb1:jlbn), intent(inout) :: b
    integer :: i,j
  end subroutine boun_extra_small
  !
  subroutine reinit_smallphi(fi)
    ! "Main" routine for reinitialization of local phi. Returns
    ! a reinitialized local phi.
    implicit none
    real, dimension(ilb1:ilbn,jlb1:jlbn), intent(inout) :: fi
    ! Local variables
    real :: cfl_reinit=0.5, tau_reinit=8.0
    integer :: maxit_reinit=15
  end subroutine reinit_smallphi
end interface
```

Appendix F
Curvature averaging routine

Listing 5: Curvature calculation with averaging

```
function curvature_loc(i,j,loc_phi)
  implicit none
  integer, intent(in)   :: i,j
  real, intent(in), dimension(3,3,loc_maxbodies) :: loc_phi
  real                  :: curvature_loc
  !
  ! Local variables
  real, dimension(loc_maxbodies) :: curv
  real :: k1,k2,phi1,phi2
  integer :: body,toexp
  !
  ! If there is just one body, we do ordinary calculation. If there are two
  ! bodies, we use a weighted average.
  do body = 1, loc_bodies
    curv(body) = curvature_local(i,j,loc_phi(:,:,body))
  end do
  if (loc_bodies == 1) then
    curvature_loc = curv(1)
  else if (loc_bodies ==2) then
    k1=curv(1); k2=curv(2); phi1=loc_phi(2,2,1); phi2=loc_phi(2,2,2);
    toexp=5 ! Exponent in curvature average
    curvature_loc = (fi1**toexp*k2 + fi2**toexp*k1)/(fi1**toexp+fi2**toexp)
  else
    if (locsalac_bodies == 0 .and. locsalac_fi(2,2,1,1) > 2*dx) then
      ! If this happens, we have removed down to 0 bodies, but it does not
      ! matter since we are far away from any interface. Set curvature equal
      ! to zero.
      curvature_loc = 0
    else
      ! This means that there is a small body which cannot be resolved by the
      ! grid. Fall back to the standard curvature method and warn user
      curvature_locsalac = curvature_local(i,j,loc_phi(:,:,1))
      write(*,*) "LOLEX method encountered small body"
      write(*,*) "Falling back to standard method"
    end if
  end if
  !
end function
```