SINTEF

NTNU – Trondheim
Norwegian University of
Science and Technology

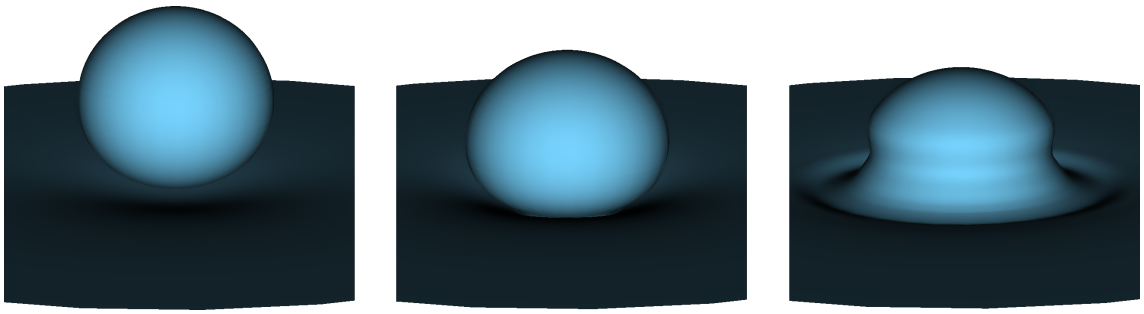
SINTEF Energy Research
NTNU Department of Physics

Master's Thesis

The local level-set extraction method for robust calculation of geometric quantities in the level-set method, with application to droplet collisions in natural gas condensation

Author:
Åsmund Ervik

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



Update titlepage

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 Ingve Simonsen, who has been my supervisor at the Department of Physics, and 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.

Double-check Masters/Master's and authors/author's

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 copied from 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.

*Todo list

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. 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 og utvides enkelt til tredimensjonale simuleringer.

Ved hjelp av denne metoden simuleres 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. Spesielt betraktes aksesymmetriske simuleringer av en vanndråpe som faller gjennom dekan og slår seg sammen med en dyp dam av vann. Resultater fra simuleringene sammenliknes med eksperimentelle resultater som finnes i literaturen. Det presenteres også rent geometriske resultater som validerer resultatene av den nye metoden, samt tredimensjonale resultater for en statisk overflatekonfigurasjon.

New section

Abstract

The level-set method is an implicit interface tracking method that can be used in two or more dimensions. The method is popular e.g. in computer vision, and as here, in simulations of two-phase flow. 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, and is easily extended to three-dimensional simulations.

Using 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. In particular, axisymmetric simulations of a water droplet falling through decane and merging with a deep pool of water are considered. The results of simulations are compared to experimental results in the literature. 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.

New section

Contents

Page

Nomenclature

$\nabla \cdot \mathbf{u}$	The divergence of the vector field $\mathbf{u}$.	
$\nabla \mathbf{u}$	The Jacobian matrix of the vector field $\mathbf{u}$.	
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
Δx	The grid spacing used in spatial discretization.	
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 $\int f(x)\delta(x)dx = f(0)$.	
$\phi(\mathbf{x})$	The level-set function. Γ is the set of $\mathbf{x}$ such that $\phi(\mathbf{x}) = 0$.	
$\operatorname{sgn}(x)$	The signum function, $\operatorname{sgn}(x) = x/\operatorname{abs}(x)$. The value at zero is $\operatorname{sgn}(0) = 0$.	
$\mathbf{n}$	Normal vector to the surface.	
$\mathbf{t}$	Tangent vector to the surface.	
CFL	Courant-Friedrichs-Lewy	
CNG	Compressed Natural Gas	
GFM	Ghost-Fluid Method	
LNG	Liquefied Natural Gas	
LSM	Level-Set Method	
PDE	Partial Differential Equation	
SSP	Strong Stability Preserving	
TVD	Total Variation Diminishing	
WENO	Weighted Essentially Non-Oscillatory	

§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 [1], 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 [2], 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 reduction in greenhouse gas emissions that results from using LNG instead of compressed natural gas (CNG) or other petroleum-based fuels. It has been estimated [3] 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 long-term research programme called "Enabling Low-Emission LNG Systems", which among other things aims to obtain a better understanding of fluid dynamics at the microscale through numerical modelling and experiments. The work presented here 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 is 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, and the code will therefore be referred to as the level-set code. The LSM is very general, and apart from fluid dynamics it has been used for modeling everything from tumor growth [4] to wildland fire propagation [5] and computer RAM production [6]. For a good introduction to the LSM, see e.g. [7]. The LSM originated from the seminal article by Osher and Sethian [8].

In two dimensions, the level-set method can easily be visualized, as in Figure 1 on the next page. In this figure, the interface is between the gray and white areas in the right half. To track this interface in 2D, a function $\phi(x)$ in a (2+1)D space (metallic gray) is used, shown in the left half, and the interface is represented by the level set $\Gamma = \{x|\phi(x) = 0\}$. $\phi(x)$ is the level-set function, and the name of this function should now be 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. 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 this chapter, a general introduction to the problem is given. In Chapter 2, the theory of two-phase incompressible flow, the LSM and numerical methods is given. In Chapter 3, the current 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. Relevant parts of numerical codes used in simulations are

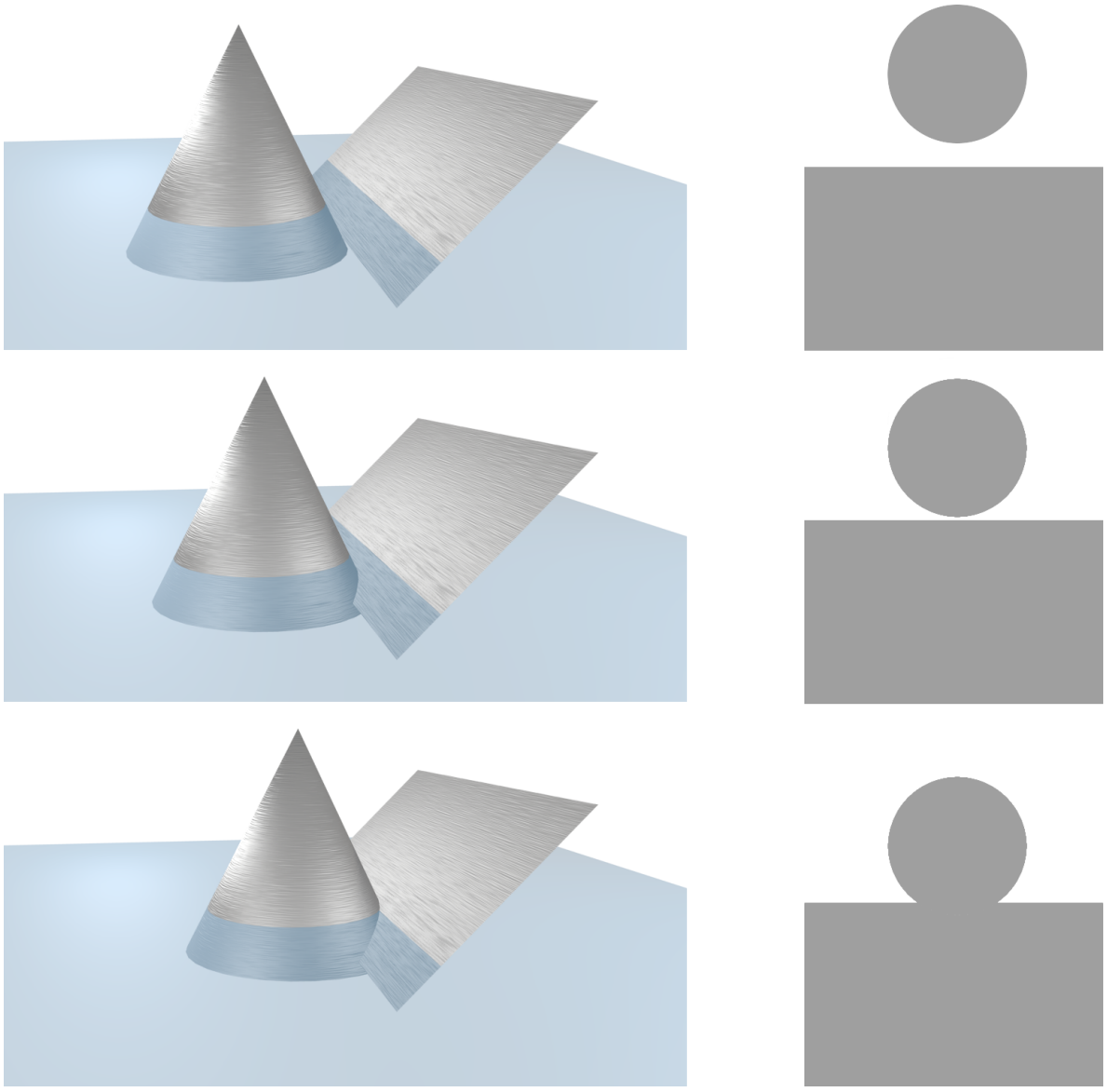


Figure 1: Illustration of interface tracking using the level-set method. The interface is between the gray and the white area on the right-hand side of the image, and the level-set function is shown in gray on the left-hand side. Also shown on the left-hand side, in blue, is the plane $\phi(\mathbf{x}) = 0$.

presented in the appendices.

§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

Old chapter, but some new parts(indicated).

THE THEORY OF THE LSM is a large subject which cannot possibly be completely reviewed here. For an excellent and extensive review of the LSM and its applications, see the review article by Osher and Fedkiw [7]. 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

New section

THE LSM IS ONE OF THE MORE SUCCESSFUL interface tracking methods used in computational physics. Since its introduction by Osher and Sethian in [8], 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. Comparing the LSM to other interface tracking methods, such as the Front Tracking Method where the interface is represented by 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 tricks have been invented to circumvent this, e.g. the HCR-2 reinitialization method[9], 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. [10], 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[11].

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.

[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.

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 $S \times \mathbb{R}$. (It should be noted that ϕ is an invertible mapping for most $\mathbf{x}$, but there exists a subset of S where this is not the case if more than one body is present in S.) As an example of this with $S = \mathbb{R} \times \mathbb{R}$, the level-set function ϕ representing a circle would be a cone standing on its tip, with an angle of 90° at the tip. The cone is of course a 2D surface embedded in 3D Euclidean space.

With this picture of a cone standing on its tip, 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. This is simply the normal vector to the circle, or indeed to any interface, when evaluated at the interface. If 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

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

In [12], 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."

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 [13]. However, when curvature calculations are combined with topological changes, things are not so rosy, as is seen further down. The standard discretization of the curvature is (see e.g. [14])

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

Second half of section in new.

Here, e.g. ϕ_x denotes the derivative of ϕ, calculated using standard central differences.

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. [15].

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 is fulfilled, namely that the interface is the same before and after reinitialization. The reinitialization procedure was introduced by Sussman, Smereka and Osher in [15], and consists in solving the PDE

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

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 [16]. Second, it may distort the interface due to grid errors.

The PDE-based approach introduced in [15] 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[15].

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 $S(\phi) = \frac{\phi}{\sqrt{\phi^2 + 2\Delta x^2}}$.[2] $\mathcal{G}(\phi)_i$ is the discretization of the so-called Godunov Hamiltonian, discussed in e.g. [15]. 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:

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

[2] 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.

Last part of section is new

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. 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 a single phase fluid flow, but we are interested in more complicated problems. However, it turns out that by a clever trick the two-phase problem is equivalent to the single-phase Navier-Stokes equations, with an additional force term which is singular and zero outside the interface, 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. This is explained in greater detail in [17, Chapter 2.2], and is a big advantage: we can use existing methods for the one-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), which unfortunately counteracts some of the improvements that come from using Equation (12). In particular, the introduction of a ghost fluid means we still have to solve the Navier-Stokes equations twice, both for the real fluid and the ghost fluid.

Double-check this is correct

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. [18].

It should also be noted that while Equations (10) to (11) appears to be coordinate-free, this is not the case. Perhaps the easiest way to see this is to compare the operator ∇^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

$$x = \rho\cos(\theta) = \rho \tag{13}$$
$$y = \rho\sin(\theta) = 0 \tag{14}$$
$$z = \zeta \tag{15}$$

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 [19]

$$\nabla^2_{\text{cart}} = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2} \tag{16}$$
$$\nabla^2_{\text{axi}} = \frac{\partial^2}{\partial \rho^2} + \frac{\partial^2}{\partial \zeta^2} + \frac{1}{\rho}\frac{\partial}{\partial \rho} \tag{17}$$

where an additional term contributes in the axisymmetric case. This difference is not purely mathematical, but physical as well. It has e.g. been noted that turbulence is a different phenomenon in 2D than in 3D, with energy dissipation being more nonlocal in 2D [20].

Part about 2D/3D is new

2.4 The Ghost-Fluid Method

THE DISCONTINUITY OF E.G. ρ 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. [17], yield Equations (18) to (22), where $[a]$ denotes the difference in a across the interface.

$$[\mathbf{u}] = 0, \tag{18}$$
$$[p] = 2[\mu]\mathbf{n}\cdot\nabla\mathbf{u}\cdot\mathbf{n} + \sigma\kappa, \tag{19}$$
$$[\mu\nabla\mathbf{u}] = [\mu]\Big((\mathbf{n}\cdot\nabla\mathbf{u}\cdot\mathbf{n})\mathbf{nn} + (\mathbf{n}\cdot\nabla\mathbf{u}\cdot\mathbf{t})\mathbf{nt} - \tag{20}$$
$$(\mathbf{n}\cdot\nabla\mathbf{u}\cdot\mathbf{t})\mathbf{tn} + (\mathbf{t}\cdot\nabla\mathbf{u}\cdot\mathbf{t})\mathbf{tt}\Big) - (\mathbf{t}\cdot\nabla_s\sigma)\mathbf{tn}, \tag{21}$$
$$[\nabla p] = 0. \tag{22}$$

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 phi, and that the jump is $[a] = a^+ - a^-$, where a^+ lies 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 [21]. The Ghost-Fluid Method (GFM) avoids this by implementing the jump conditions implicitly. The GFM was introduced in [22] for the Euler equations, and extended to the Navier-Stokes equations in [21].

As the name suggests, the GFM introduces a ghost fluid. The purpose of this ghost fluid is to represent both fluids everywhere in the computational domain. When we momentarily pretend that both fluids exist everywhere in the domain, we simply solve all equations for both fluids, then update the position of the interface, and use the sign of the level set function to select which solution is the physical one at each point. The jump conditions are enforced onto the ghost fluid using interpolation.

Improve this section

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

$$\frac{\partial u(x)}{\partial x} \approx \frac{u(x+\frac{1}{2}\Delta x) - u(x-\frac{1}{2}\Delta x)}{\Delta x}. \tag{23}$$

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 [23] for details and justification of this scheme. WENO schemes are known to be computationally efficient in addition to being robust[23]. 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 [23] for these.

When the temporal solution of a PDE is considered, similar problems arise. We consider 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

$$\frac{\partial u(t)_i}{\partial t} = f\left(u(t)_i, u(t)_{i+\frac{1}{2}}, u(t)_{i-\frac{1}{2}}, \dots\right) = f(u). \tag{24}$$

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 $\mathcal{O}(\Delta t^5)$.

The problem with the classical RK method and other related RK methods is that they may produce unstable solutions in the presence of shocks. To avoid 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 one time integration 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, at the expense of being more lengthy to describe and implement. For examples of low-storage methods and a good discussion of SSP-RK methods in general, see [24]. The methods described here are not low-storage methods in an effort to avoid too many technicalities.

In order to specify the SSP-RK methods used here, we use the convenient notation afforded by the Butcher tableau. In Table 1, a general four-stage RK method is shown, together with the Butcher tableau representing this method. The coefficients describing the method are divided into three groups, the a_{ij} which are highlighted in green, and the b_i which are highlighted in red. The third group of coefficients, c_i, are not used when $f(u)$ does not depend explicitly on time, as is the case here.

Table 1: LEFT: A four-stage RK method. RIGHT: The corresponding Butcher tableau, a simple way to organize the coefficients. In this case, $f(u,t) = f(u)$ is assumed, so the coefficients c_i, used for specifying the time $f(u,t)$ should be evaluated at in each stage, are irrelevant.

$$\begin{aligned}
k_1 &= \Delta t f(u) \\
k_2 &= \Delta t f(u + a_{21}k_1) \\
k_3 &= \Delta t f(u + a_{31}k_1 + a_{32}k_2) \\
k_4 &= \Delta t f(u + a_{41}k_1 + a_{42}k_2 + a_{43}k_3) \\
u^{n+1} &= u^n + b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4)
\end{aligned}$$

0				
c_2	a_{21}			
c_3	a_{31}	a_{32}		
c_4	a_{41}	a_{42}	a_{43}	
	b_1	b_2	b_3	b_4

The Butcher tableaus for the two SSP-RK methods used here are given in Table 2. The more accurate of these methods, the third-order SSP-RK 3()4, is given in Table 2 b). This is used when solving the Navier-Stokes equations. The less accurate second-order SSP-RK 2()4 given in part a) of the table 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 [25].

Table 2: **S**trong **S**tability-**P**reserving Runge Kutta schemes used for solving the differential equations at hand.

(a) SSP four-stage second-order RK

0				
1/3	1/3			
2/3	1/3	1/3		
1	1/3	1/3	1/3	
	1/4	1/4	1/4	1/4

(b) SSP four-stage third-order RK

0				
1/2	1/2			
1	1/2	1/2		
1/2	1/6	1/6	1/6	
	1/6	1/6	1/6	1/2

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. See [26, 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 [17] and [27]. 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 [23] and [25]. This concludes the theory chapter. The next chapter introduces and motivates the proposed method for robust calculation of geometric quantities.

§3 Local level-set extraction - the LOLEX method

Contents

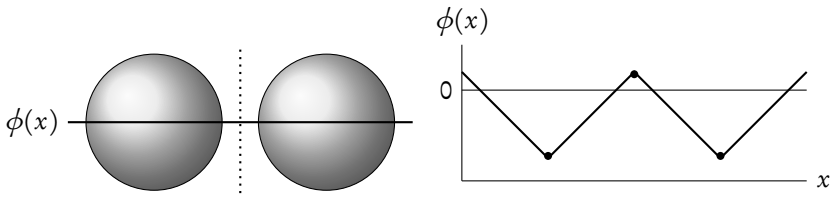
> A good theory sticks its neck out by making claims that can, at least in principle, be found to be wrong.
>
> Richard Feynman

Proofread chapter

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) and in calculating the surface force in Equation (12). 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 (18). 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 (4) and Equation (3) 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.[3] For a graphical depiction of the problem, see Figure 2. When this happens, the curvatures and normal vectors will be erroneous, sometimes sufficiently so to make the simulations crash.



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

Figure 2: (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)

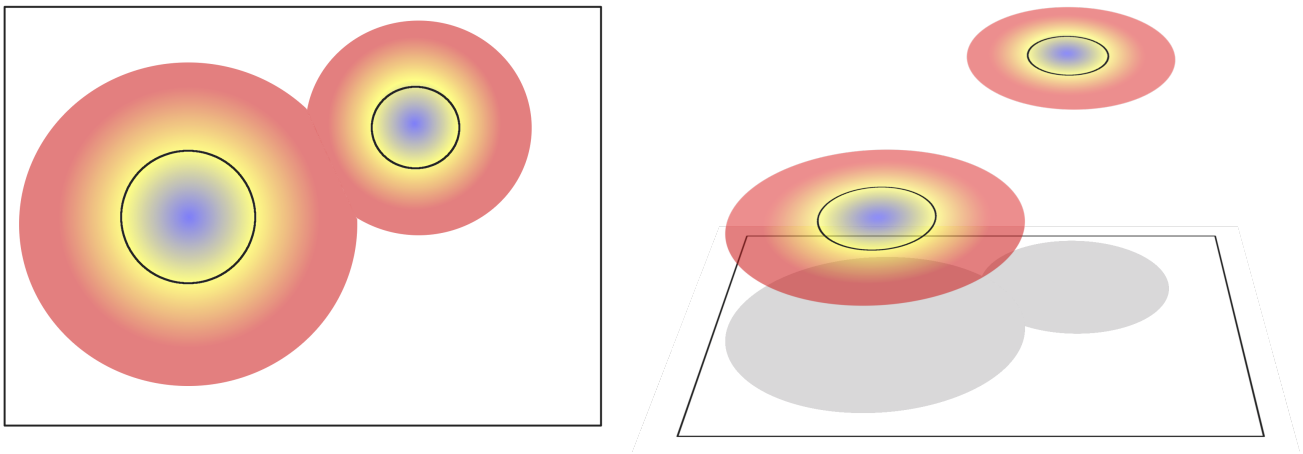
Several approaches have been used to remedy this flaw. The first approach to this problem is described by Smereka in [12]. He describes the problem briefly, and increases the numerical smoothing in the curvature discretization to lessen the effect. This is not an optimal solution, and Smereka notes on one

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

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.

The earliest non-smearing approach, by Macklin and Lowengrub [4], uses a modification of the directional differences for points close to kinks, along with a curve fitting scheme. This has been elaborated further by the same authors. Further improvements to this method, and adaptations to the framework used in the present level-set codes, have been developed by Lervåg [28],[29]. 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 [30], 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. For a review and comparison of these methods, see the article by Lervåg and Ervik [31].



(a) Original level-set function

(b) Extracted level-set functions

Figure 3: 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”.

Improve figure, make realistic

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.

We give here an outline of the following sections: we start by giving a more thorough account of the Salac and Lu method, upon which the current method is based, in Section 3.2. Some concepts from the Macklin and Lowengrub method are also introduced, since they are used in the current method.

The motivation for the current method is then explained. 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. The actual code used is presented in the appendices; 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 current method

THE IDEA OF SALAC AND LU is simple when compared to the curve-fitting scheme used by Macklin and Lowengrub and later by Lervåg. 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 trivially extendable 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 4, 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 4 is taken from the author's project thesis, [32, 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 5, copied from [4, Figure 6].

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 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 methods 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 [4]. It is defined as

$$Q(\mathbf{x}) = |1 - \nabla\phi(\mathbf{x})|, \tag{25}$$

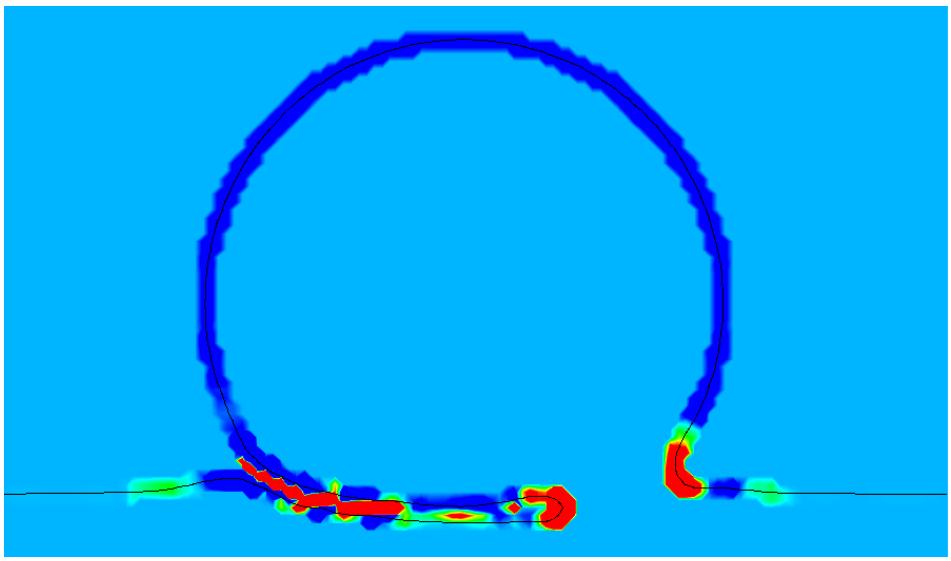


Figure 4: The curvature field plotted in the final frame before the simulation crashed. Note the red curvature field inside the air finger between the drop and the pool, which is incorrect. The color should be green in this area.(Figure best viewed in color)

Improve figure w lighter background

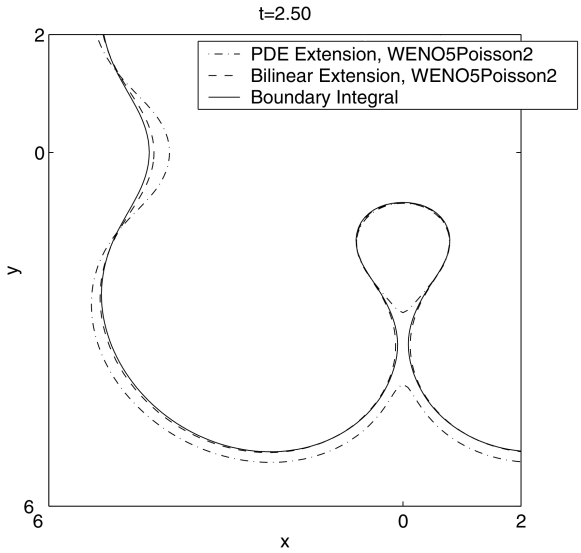


Figure 5: Another typical situation where the Salac and Lu method would fail. Figure copied from [4], showing the interface between a tumor and healty tissue. The dash-dotted line is the least accurate solution.

i.e. the deviation of ϕ from a signed distance function. If $Q(\mathbf{x}_{i,j,k}) > \eta$, we use the present method for the grid point $\mathbf{x}_{i,j,k}$. 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 [33]. 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 severly restrict the use of 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 6. 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 bands indicate 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.

Check consistent use of "present method"/"LOLEX method"

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 the appendices, 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 ($Q(\mathbf{x}_{i,j}) < \eta$) 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.
  - ↪ 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 the internal points of all bodies other than this one
  - ↪ 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.

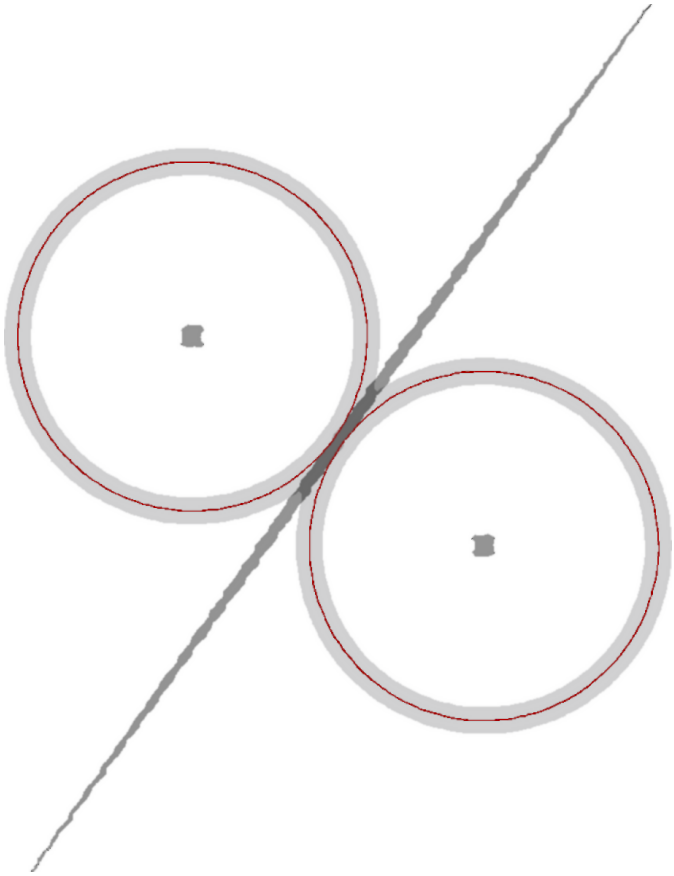


Figure 6: Illustration of the quality function and the narrow band level-set method. The red lines indicate $\phi = 0$, the light grey bands indicate the narrow band where $|\phi| < \epsilon$, and the medium grey bands indicate where $Q(\mathbf{x}) > \eta$. The intersection of the light and the medium grey bands, the dark grey area, indicates where the LOLEX method will be used.

The steps in this algorithm that perhaps 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` etc. are all set to 7 in the following. Their values are independent of the global grid size, denoted by `imax`, `jmax`, `kmax`.

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 short outline, or because they have not been properly motivated yet. The steps that will be considered is 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 appendices. 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.

In practice, a Gaussian is used to decide which bodies are discarded. A normalized Gaussian is precomputed and stored in an array which is the same size as the `bodies` array, centered at the centre of the local area, with 1/4 the width of the local area as the standard deviation. The values in this array are summed up for the points in the current body. If these sum up to less than a given value, the current body is discarded. A threshold of `0.11` is used here. For removed bodies, the points in the body are marked as `removed`. Points not inside a body are marked as `nobody`.

Verify final threshold. Difference 2D/3D? Gaussian/constant weight?

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 vectors for this point is unimportant, so we set the curvature equal to zero. If all bodies are removed, but the central point is close to an interface, an error message is printed saying that something has gone wrong. This has not happened during the simulations reported here.

verify $2\Delta x$

A 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, which is too few to cause memory problems.

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 from [34], which is slightly modified. This procedure was also used in the author's project thesis [32], and the remainder of this subsection is copied from that report with minor changes.

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.

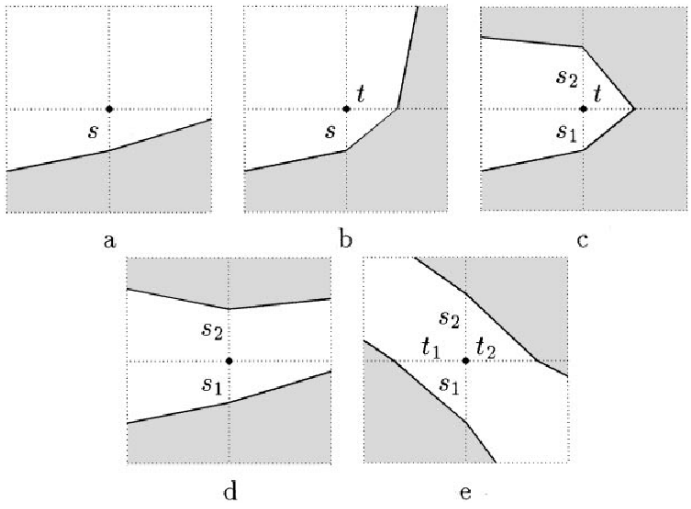


Figure 7: The five cases considered in the Adalsteinsson et al. method. Note that here, the body displayed in gray is just one of the bodies close to the central point. This implies that the fifth case is not relevant in this context: if the interfaces in **e** belong to two different bodies, we are 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 [34])

The procedure in [34] 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 7, copied from [34].

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

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

$$s = \Delta x + \phi(i-1, j) \tag{26}$$

where we have assumed that $(i-1, j)$ 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 (Δx) 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 distorted.

b The interface crosses two of the lines, and these two lines make out a corner of the 2×2 grid in Figure 7. 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

$$\left(\frac{d}{s}\right)^2 + \left(\frac{d}{t}\right)^2 = 1. \tag{27}$$

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

$$d = \frac{st}{\sqrt{s^2 + t^2}} \quad . \tag{28}$$

If we have $s^2 + t^2 = 0$, then $s = t = 0$, so it is obvious from Figure (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

$$\left(\frac{d}{\min(s1, s2)}\right)^2 + \left(\frac{d}{t}\right)^2 = 1. \tag{29}$$

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; this is not considered in further detail here.

Consider the 3D case

3.4.3 Extrapolation

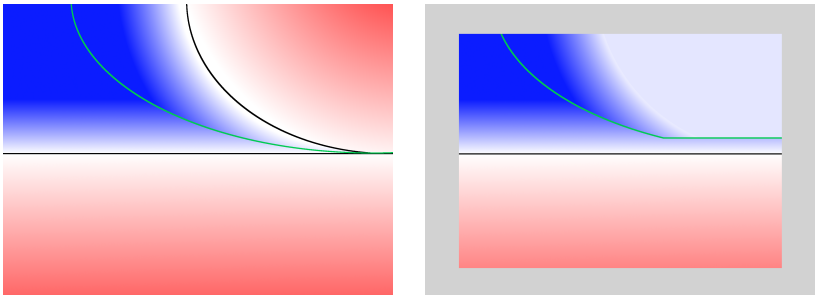
After the interior of the `locphi` array has been filled, the ghost cells must be filled before we can reinitialize the local ϕ. A first approach was to use linear extrapolation, which should work well since ϕ is a linear function in 1D. However, it turns out that this approach does not work. A fundamental 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 8(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 8(d) and (f).

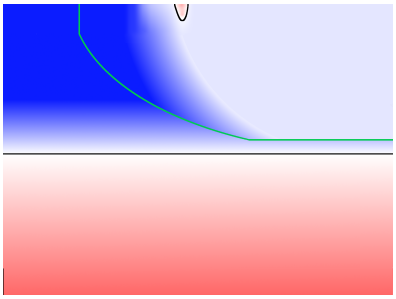
We describe further the parts of Figure 8. In (a), a zoom in on the global level set of a droplet touching a pool is shown. In (b), the local level set of the lower body (the pool) is shown after extraction and explicit reconstruction. Here, the values on the edges are not set, indicated in green. In (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 8 is an illustration ignoring grid effects, a “real-life” example of a zeroth-order extrapolated local level set is given in Figure 9 (a). This figure also illustrates how the corner cells are handled.

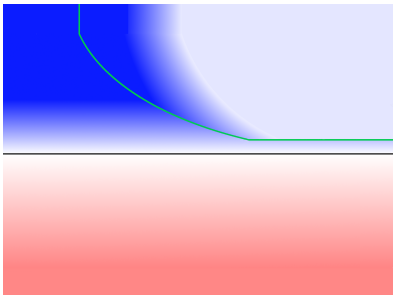


(a) Zoom in on global level set

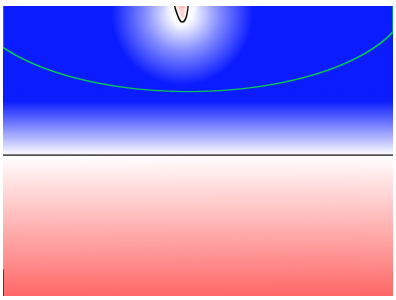
(b) Extracted local level set



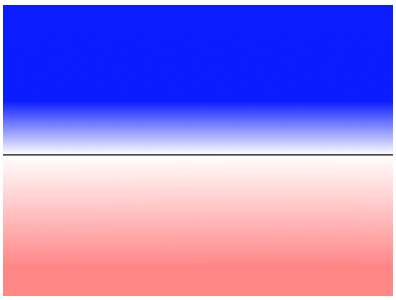
(c) First order extrapolated



(d) Zeroth order extrapolated



(e) First order, reinitialized



(f) Zeroth order, reinitialized

Figure 8: Extraction, extrapolation and reinitialization of the local level set is shown, for the lower body in Figure (a). Figure (a) is a zoom-in of Figure 6. 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.)

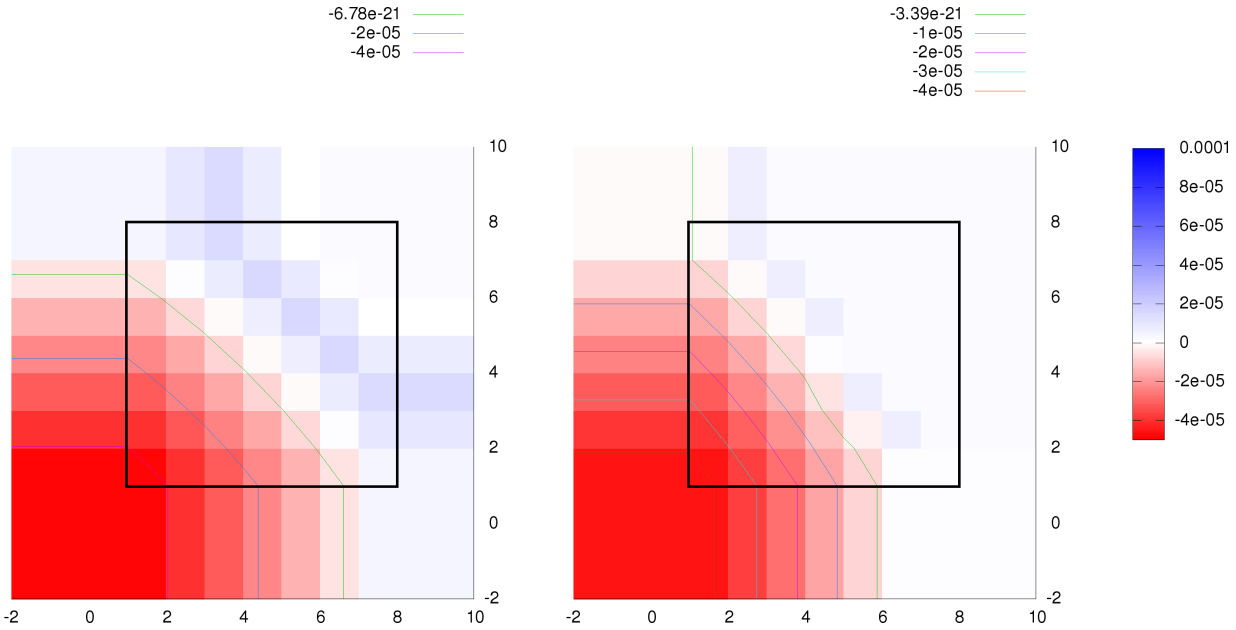


Figure 9: Zeroth-order extrapolated local level set, part of a circle. In this case, there are 7 internal cells in either direction, and three ghost cells outside this. The black square frames the internal cells. In (a), the distance from the interface to the other body is $2.8\Delta x$. In (b), the distance is $1.1\Delta x$. It is seen that the zero level set in (b) is slightly deformed.

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. However, the curvature and normal vectors calculated further down are sufficiently accurate that no more time was spent pursuing this alternative 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, however, straightforward.

When reinitializing, we require at least some points on either side of the interface with decent ϕ-values. 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.

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.

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 9 (b). This is a similar case to that in Figure 9 (a), but the distance to the other body 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 a different level set, we simply add a 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 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 gives a much more accurate representation of the interface. This will, in turn, give a much more accurate curvature.

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 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.

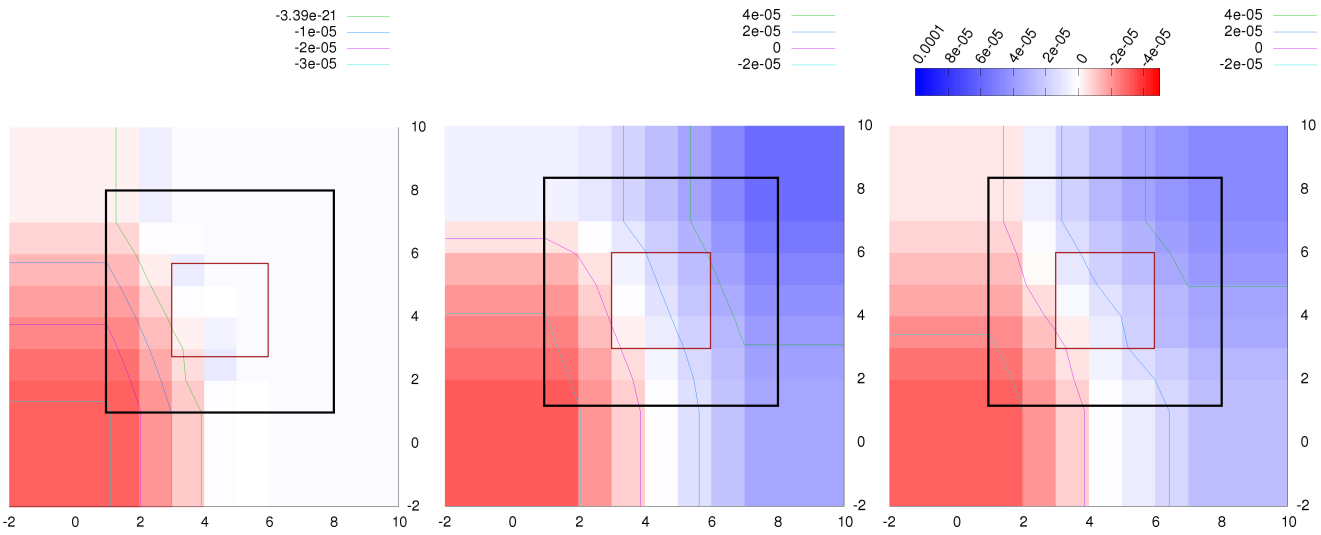
The fact that specific routines had to be written for local level-set reinitialization enabled some simplifications in this code. The ordinary reinitialization uses routines that are shared with other PDE solvers, and thus they contain cases (e.g. in the choice of Runge-Kutta method) that are irrelevant for local level-set reinitialization. In the routines for local level-set reinitialization, these cases are removed. For reinitializing the local level-set, the SSP-RK 4()2 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 than can be varied. An overview of these is given here, along with the values used presently, and sensible ranges for some of them.

Make parameter list

To conclude this section, the entire main routine of the present method is included as well. It can be



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

Figure 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 $\phi = -1.0\Delta x$. (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.

seen in the appendices. 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 motivation behind this method and the advantages compared to other methods have been given. The present method is said to handle bodies folding back onto themselves, as well as being easily extended to three dimensions. The origin of the ideas behind the method have been explained, and the technical details have been given. The descriptions should be sufficient for anyone wanting to implement the LOLEX method. The parameters of the method, which may be chosen with varying freedom, have also been given. Results, both for static and dynamic simulations, are given in the next chapters.

§4 Results from LOLEX calculations on static cases

Contents

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

Proofread chapter

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, and from the Salac and Lu method and the Lervåg method which is a variant of Macklin and Lowengrubs method.

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[32].

Include initial tests

Write section on computational expense of LOLEX

4.1.1 Averaging the curvature values

After all the previous steps, 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 central point by d_i. The geometrically weighted average is then given by

$$\kappa = \frac{d_1\kappa_2 + d_2\kappa_1}{d_1 + d_2}. \tag{30}$$

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

$$\kappa = \frac{d_1 + d_2}{d_1/\kappa_2 + d_2/\kappa_1}. \tag{31}$$

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 [35]. In the present case, large values are often incorrect, since the errors produced by the ordinary method are $\mathcal{O}(1/\Delta x)$ [4]. 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 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.

$$\kappa = \frac{\kappa_1 + \kappa_2}{2}. \tag{32}$$

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

$$\kappa = \frac{d_1^2\kappa_2 + d_2^2\kappa_1}{d_1^2 + d_2^2}. \tag{33}$$

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

$$\kappa = \lim_{n\to\infty} \frac{d_1^n\kappa_2 + d_2^n\kappa_1}{d_1^n + d_2^n}, \tag{34}$$

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

$$\kappa = \begin{cases} \kappa_1 & \text{if } d_1 < d_2 \\ \kappa_2 & \text{if } d_1 > d_2 \\ \frac{1}{2}(\kappa_1 + \kappa_2) & \text{if } d_1 = d_2 \end{cases}. \tag{35}$$

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.

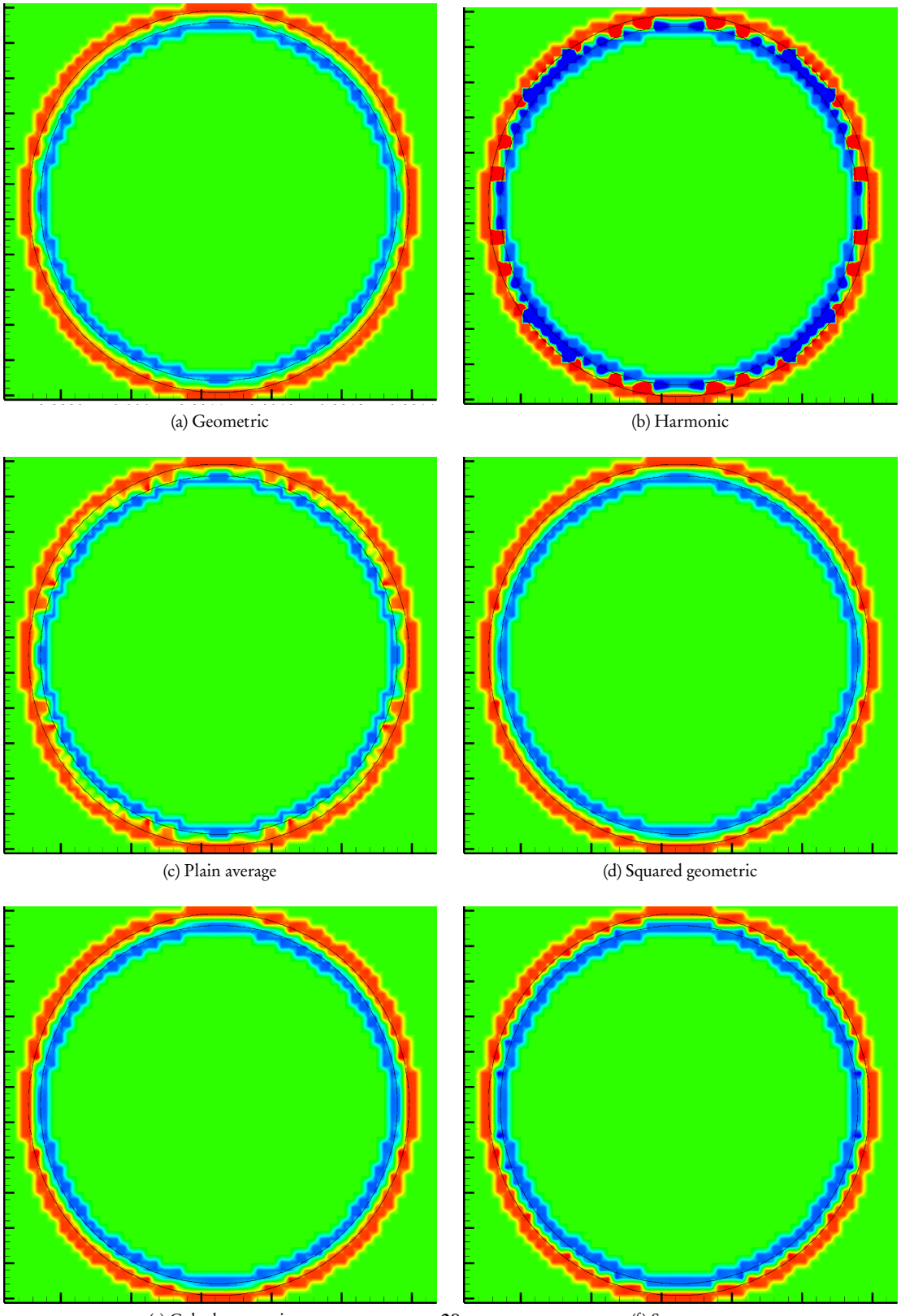
PROOF. We want to show that the limit in Equation (34) is Equation (35). Suppose that $d_1 > d_2$, the argument is identical in the opposite case and for equality. We can divide by d_1 in the nominator and denominator in Equation (34), and defining $a = d_2/d_1 < 1$ we are left with

$$\kappa = \lim_{n\to\infty} \frac{1^n\kappa_2 + a^n\kappa_1}{1^n + a^n} = \frac{1\kappa_2 + 0\kappa_1}{1+0} = \kappa_2 \tag{36}$$

■

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 the first test, the width of the ring was set to `1.5 dx`.

The results of the test are shown in Figure 11. 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. It is seen that the harmonic average performs worst of all. 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.



(a) Geometric (b) Harmonic

(c) Plain average (d) Squared geometric

(e) Cubed geometric (f) Supremum

Figure 11: Comparison of different ways to average the curvature

In the line plots, the interpolated curvature values are shown for 16 points equidistantly spaced on a 4*dx interval along the interface. These plots are shown in Figure 12 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 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 13. From this it seems that $n = 5$ is a good choice, slightly better than the geometric average. 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 has a problem at this point. The interpolation routine uses the marching squares algorithm, which has well-known problem in some cases [36]. 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 [12]. The interpolation is only used here in order to compare the averaging methods. Thus no interpolation errors affect the physical results.

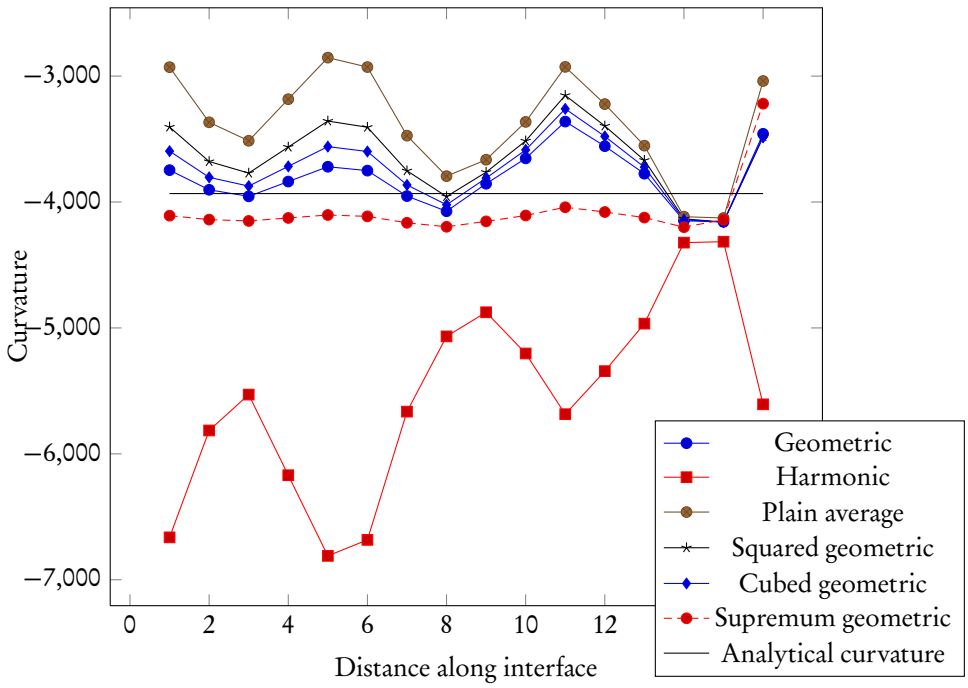


Figure 12: Lineplots of curvature along the interface for different averaging methods.

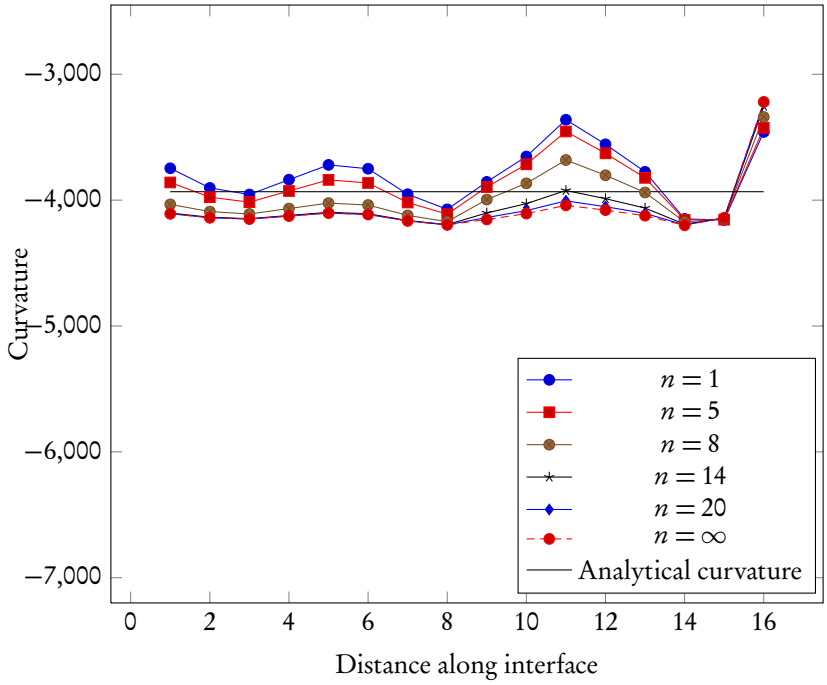


Figure 13: Lineplots of curvature along the interface for different exponent n in the geometric weighting.

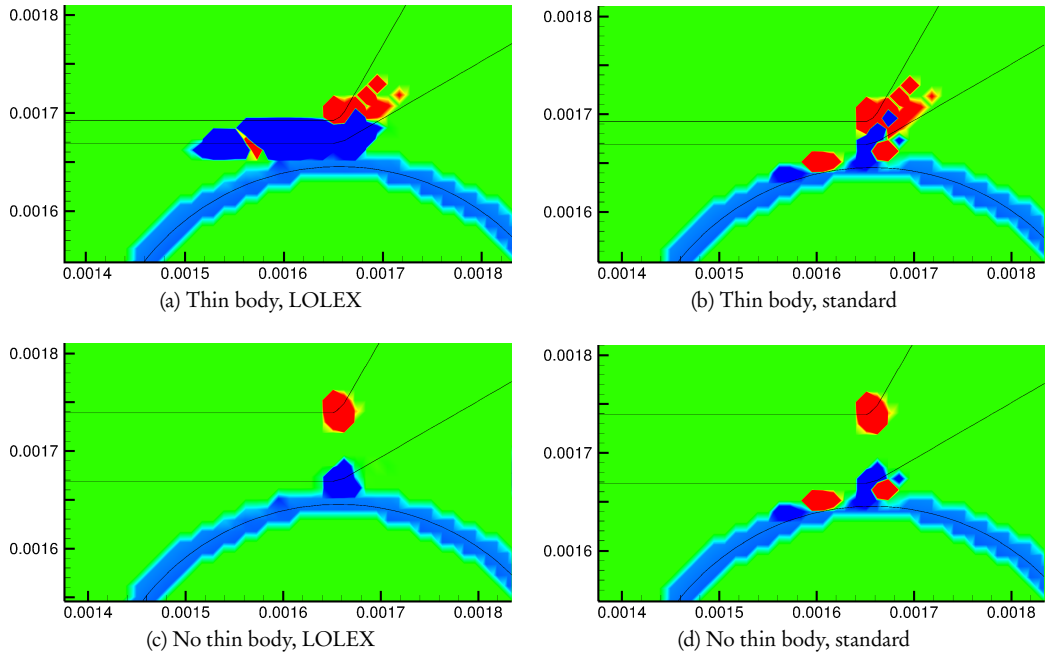
4.2 A problem with thin bodies

A PROBLEM WITH THIN BODIES and the level-set method was discussed in the author's project thesis [32]. 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 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. 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 greatly. 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. It does not occur in the similar but much more important situation when two bodies are close, 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 were thinner than `2*dx`, the curvature becomes incorrect due to the reinitialization deforming the thin bodies. The resulting error is shown in Figure 14.



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

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

Figure 14: 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)).

In Figure 14 (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 14 (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 15, reinitialization of such a thin body can result in a deformation inducing an incorrect curvature. Note in Figure 14 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 bodies, 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 effects in the

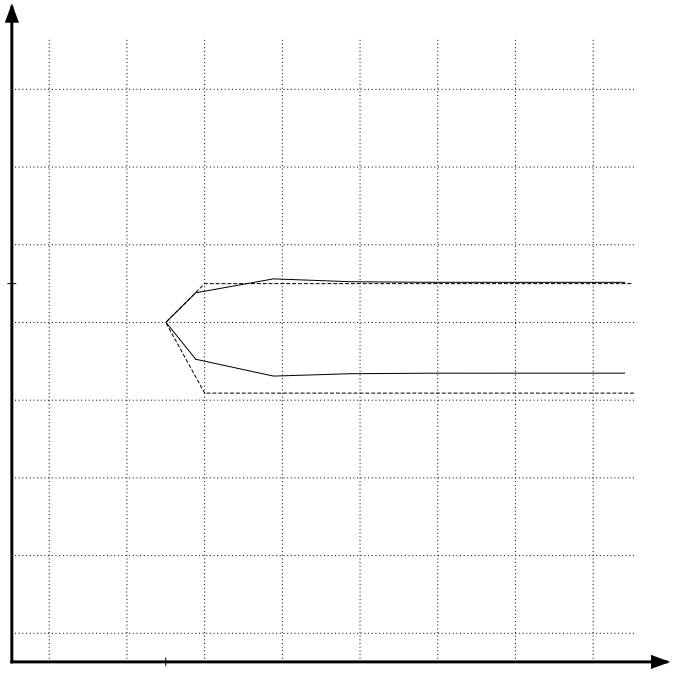


Figure 15: Thin body deformed by reinitialization. The dotted line shows the initial interface, where ϕ is a signed distance function. The full line shows the result of reinitialization. In theory these should overlap. The width of the initial body is $1.45\Delta x$.

Make figure less tall

literature, although several authors (e.g. Smereka in [12]) 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 5.5.

To summarize this section, the LOLEX method tends to give large errors 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.

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 [29] 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 distance from the grid point to both interfaces is exactly equal.

Using this selection of the proper normal vector, the normal vectors were calculated for the test case discussed previously in Figure 11. In this case, the width of the thin air-film was $1.6\Delta x$. The results for all four methods are shown in Figure 16.

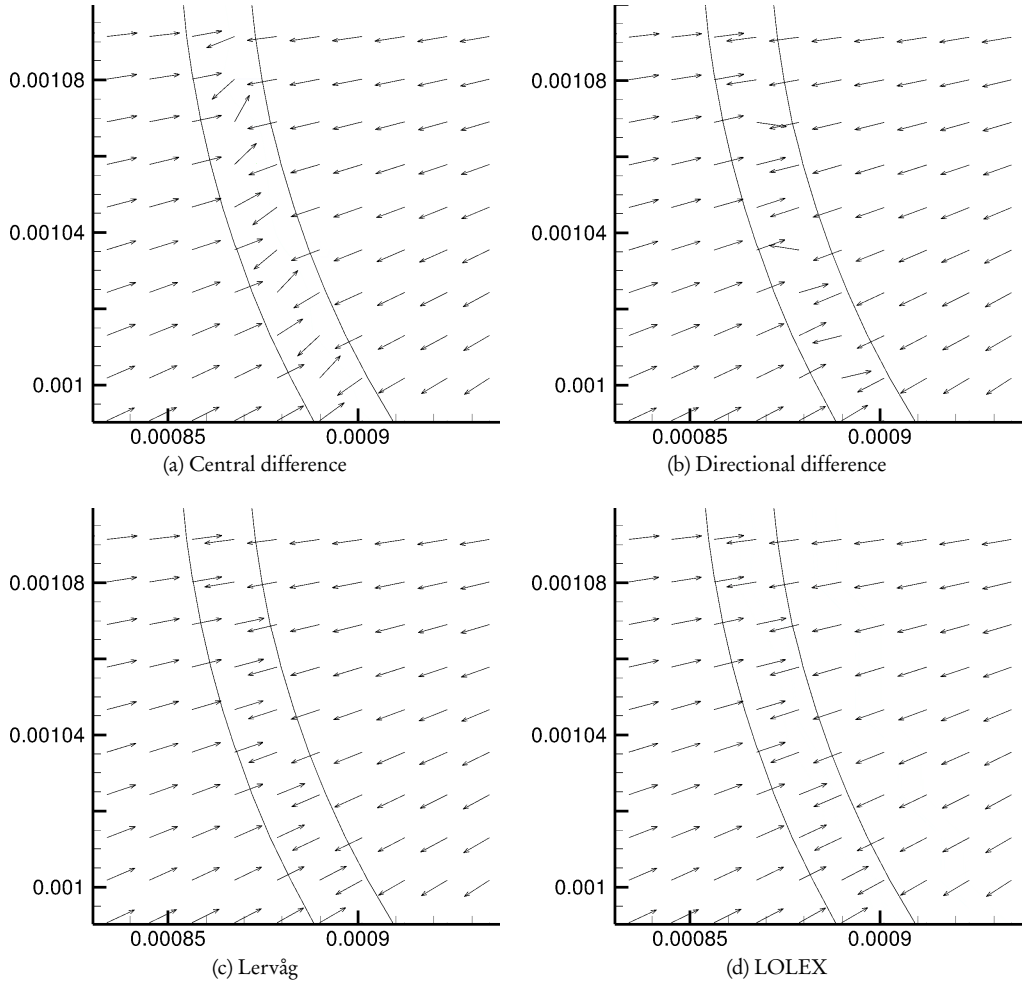
Perhaps the most surprising result of this is that the directional difference method is not much better than the central difference method. This is what prompted the use of curve fitting methods; Macklin and Lowengrub initially used directional differences and additional grid refinement in [4], but switched to curve-fitting methods in [37]. As is seen in Figure 16 (c), the curve fitting method (the method by Lervåg is used here) gives 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 the figures and zooming in alot; then a minute difference can be seen, as in Figure 17. Here, the normal vectors calculated by the LOLEX method have been colored dark red, and the ones calculated by the Lervåg method have been colored dark blue. 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 [38] for a review of various methods including the use biquadratic surface and of splines. The conclusion of [38] 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 9 (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[39]. 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.



(a) Central difference (b) Directional difference

(c) Lervåg (d) LOLEX

Figure 16: Comparison of normal vector calculations using different methods.

The results of the proof-of-concept implementation are shown in Figure 18. 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 18 (a), the standard 27-point stencil due to Kang et al. [14] is used. In Figure 18 (b), the LOLEX method is used. 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 18 (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

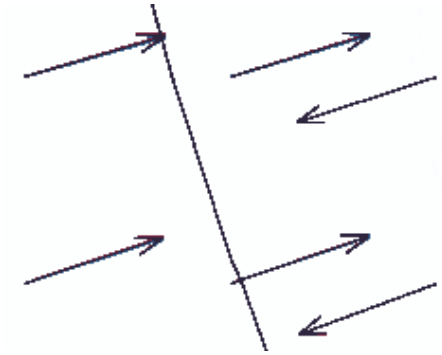
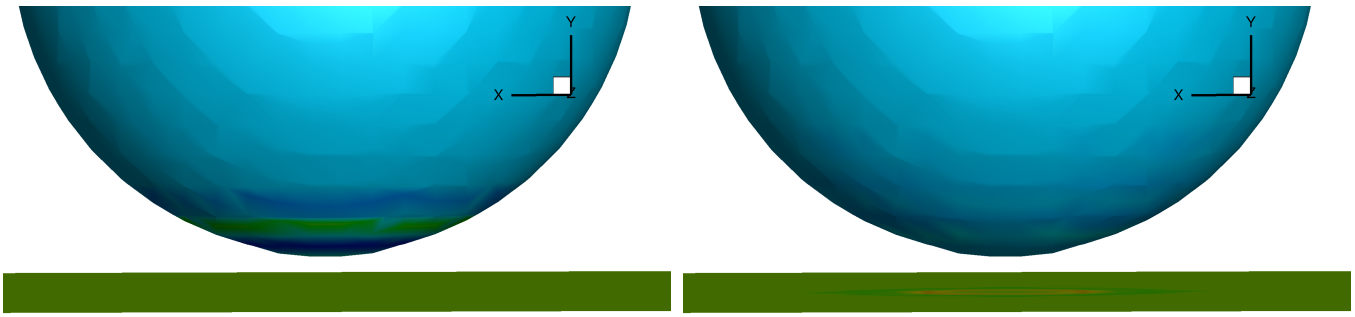


Figure 17: Comparison of the Lervåg method and the LOLEX method for calculating normal vectors. The LOLEX normal vectors are colored dark red, the Lervåg normal vectors are colored dark blue. A minute difference can be seen.

much better than the standard method close to kinks in the level-set function. There is still a small error, and part of this could be due to some elusive bug(s), but some deviation has to be expected since we are reconstructing information that is missing from the level-set function. A deviation of this magnitude is unlikely to have a large impact on simulations, in contrast to the errors from the standard discretization.



(a) Standard discretization

(b) LOLEX method

Figure 18: 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 yellow ring on the green plane in (b) is a bug which has not been fixed.

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 [30], but it is not known how (or if) they handle problems like that illustrated in Figure 4. 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 is probably due to the computational costs of 3D simulations. This cost is such that highly accurate (i.e. interface tracking) 3D

simulations are not common. As an example, the 2D simulations reported by Macklin and Lowengrub in [4] were performed on supercomputer hardware; this indicates that 3D simulations are still too demanding to perform. The 3D simulations reported by Salac and Lu in [30] consider motion under mean curvature, which is computationally much less expensive than fluid flow simulations. However, 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 beginning to come within reach. When this happens, a method such as the present one will be necessary in order to get accurate results.

§5 Results from 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

Proofread chapter

5.1 Summary of previous simulations

Skip this section, has not been rewritten yet

THE RESULTS OF PREVIOUS SIMULATIONS performed with the present codes are reported here. These were completed during the author's specialization project in the fall of 2011, and the results reported here are from the author's project report.

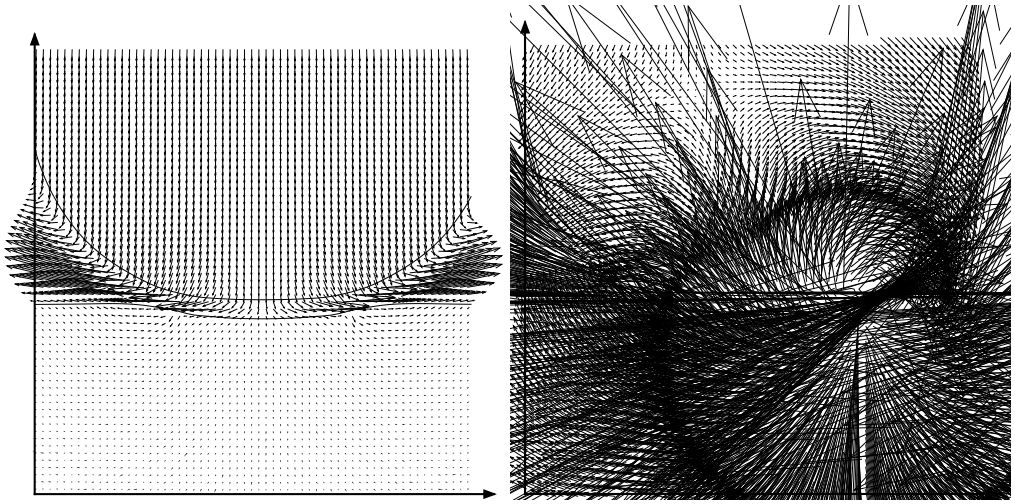
5.1.1 Droplet colliding with pool

WITH THIS MOTIVATION, several numerical simulations of droplets colliding with pools were performed using the new curvature calculation method. The basic case considered was a 0.6 mm diameter droplet of methanol falling through air onto a pool of methanol from a height of 0.65 mm. A second case was also considered, where the drop fell from a height of 2.5 mm. The droplet reached a speed of 0.09 m/s in the first case, and 0.27 m/s in the second case, before colliding with the liquid pool. Such velocities are in the lower end of the range of velocities studied experimentally. These cases were also chosen because the old method fails to produce any interesting results here. The simulation of the first case crashes when the droplet is $2.6\Delta x$ away from the pool, or after 0.01193 s.

The crash in this simulation using the old method is caused by the erroneous curvature field, which induces an unphysical pressure field. A typical crash is shown in Figure 19. In Figure 20, 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 does 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.

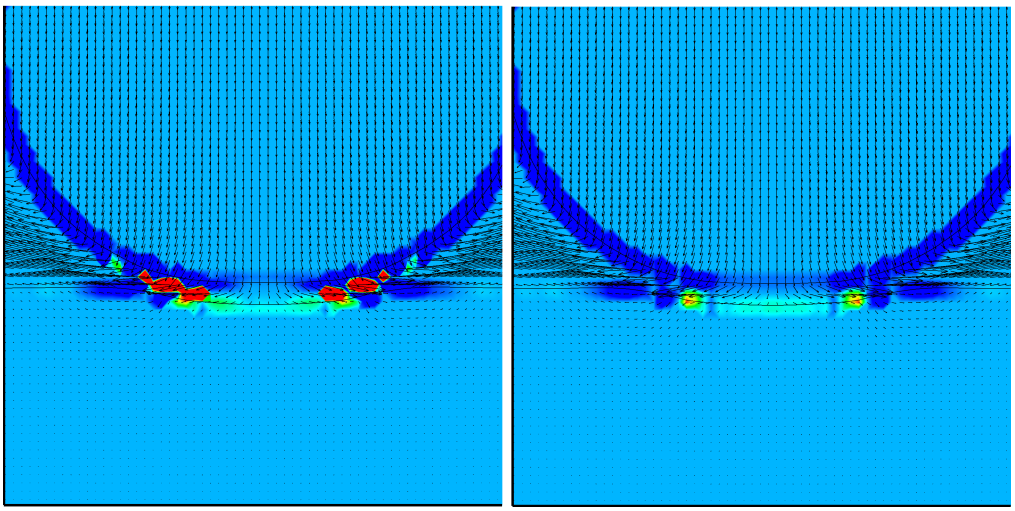
For completeness, some further properties of the simulations above are provided here. As stated previously, the only difference between the two simulations is the curvature calculation method. The simulations were performed on a uniform 162×242 grid, which represented a 1.5 mm×2.25 mm domain. [4] The droplet had a diameter of 0.6 mm, and was placed 0.65 mm above the pool. The density of methanol was set to 792 kg/m^3, while the surrounding air density was 1.2 kg/m^3. The dynamic viscosity

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



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

Figure 19: The droplet colliding with the pool, showing how early the old 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.



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

Figure 20: 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.

was $560 \cdot 10^{-6}$ Pa s for methanol and $20 \cdot 10^{-6}$ Pa s for air. The surface tension was $2250 \cdot 10^{-6}$ N/m, and standard gravity (9.81 m/s^2) was used. No-slip conditions were enforced on all boundaries.

Considering more numerical settings, the ILUK-preconditioned GMRES Poisson solver from the PETSc library was used, with a desired residual of $1.0 \cdot 10^{-12}$. 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.[5] The HCR-2 method was used for the reinitialization. A CFL-number of 0.3 was used for the Navier-Stokes solver, while a CFL-number of 1.0 was used for the reinitialization and for extrapolation of the velocity and curvature. 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 (or rather think that it has crashed). For details of the CFL condition used, see [14]. 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. Apart from the settings mentioned here, the standard settings of the level-set codes were used; there are too many to list them all, but the important ones are given.

As stated, the simulation using the Salac and Lu method does not fail where the ordinary method does, but this simulation also failed some time later, at 0.0123 s. This is shown in Figure 21, 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 21, 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. This has not been investigated in further detail here, but will perhaps be considered in future work.

Since it seems somewhat unphysical that the merger of the droplet and the pool happens all the way to one side, possible causes for this behaviour were investigated. Two questions were considerd: 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? Closer inspection of the frames just before merging, as in Figure 22, gives an answer to the first question. It seems that the culprit may again 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. This is caused by the same phenomenon as described earlier in **??**, in **??**. Note that in Figure 22, 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.

The answer to the second question seems to be that instabilities arising in the simulation break the initial symmetry of the problem. This is discussed in more detail further down, in **??**.

5.1.2 Diagonal simulation of droplet colliding with pool

ATTEMPTING TO WORK AROUND THE PROBLEM of merging at one side, a new simulation was performed where the physical situation was rotated 45° counter-clockwise relative to the grid. The idea was that this would make the droplet merge with the pool at the middle, making the air

[5] 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.

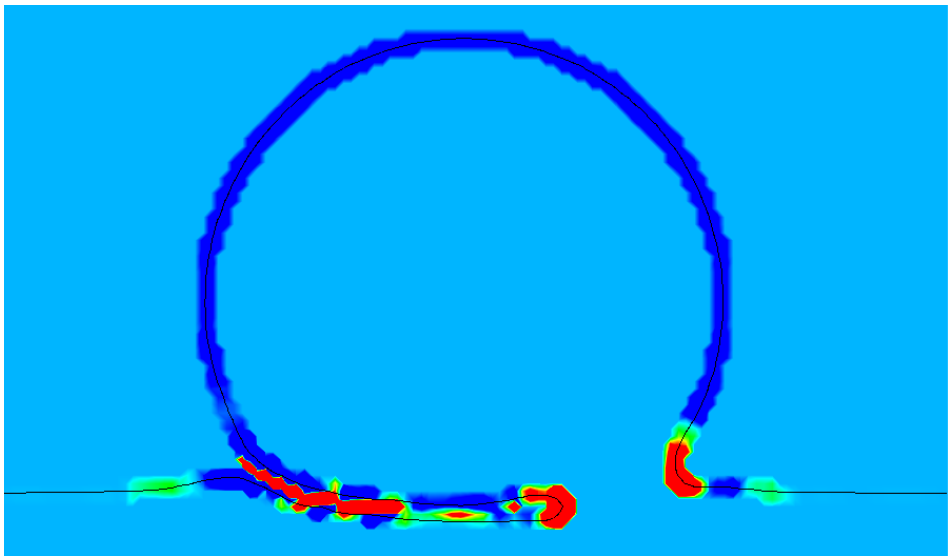


Figure 21: The curvature field plotted in the final frame before the simulation crashed. Note the red curvature field inside the air finger between the drop and the pool, which is incorrect. Note also the wave in the pool surface that can be seen on the left-hand side.

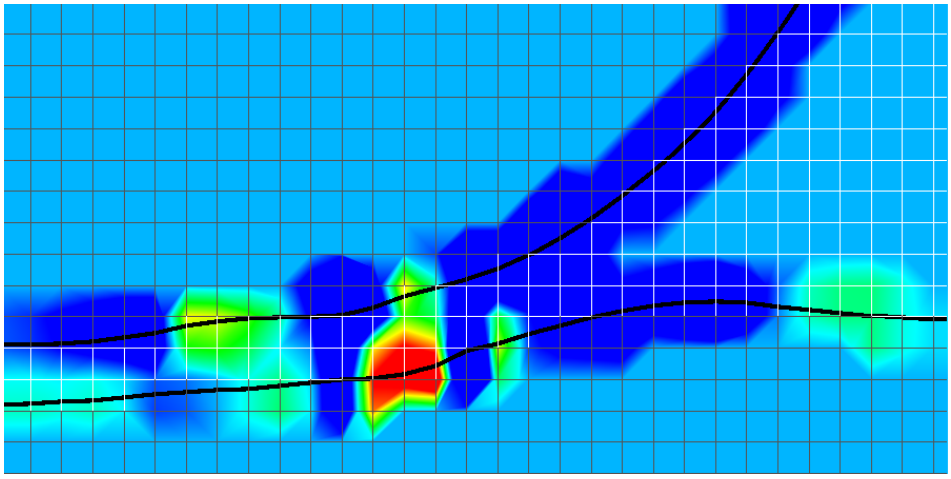
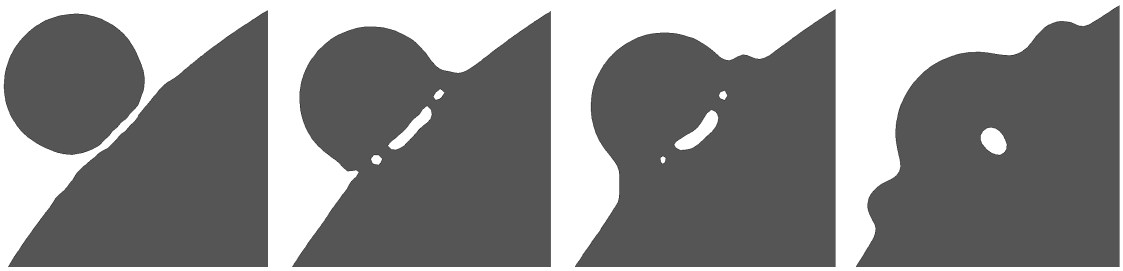


Figure 22: A zoomed in plot of the curvature field, before the droplet has merged into the pool. The grid is shown, and is colored white in air and dark gray in the fluid. It is seen that the interfaces have become blocky, following the edges of grid cells instead of being smooth.

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 are just simple mirroring, this becomes a small problem, as mirroring boundary conditions force the pool to intersect the domain boundaries at 90°. This induces 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 23.



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

Figure 23: The droplet colliding with the pool in the diagonal simulation. This plot is zoomed in on the droplet, the dark gray indicates methanol and the white indicates air.

In this case, the simulation does not crash, and can be continued for as long as one wants. After the final frame shown in Figure 23, the waves on the right and left 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. It is not yet possible for the numerical simulations and the experimental conditions to meet in the middle with regards to the impact velocity, so it is not known for sure at this time whether "no jetting" is correct in this case.

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 in **??**, 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. It is therefore not known whether such bubbles form. However, no extensive literature search has been performed to see if other experimental data is available to confirm or deny the existence of such bubbles. This may be done as part of future work, particularly if such bubbles are seen to form in the numerical results for many different initial conditions.

A final interesting feature of this simulation that is worth mentioning is the two waves produced on the pool surface by the droplet. Such waves are well-known from experiments, as is seen below in Section 6.2, 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 21. 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 23 (d).

5.2 Results from LOLEX on previous cases

USING THE LOLEX METHOD ON THESE CASES solves some problems

Complete methanol-in-air simulations, report results

5.3 Stability considerations for methanol-in-air case

New section

STABILITY IS ALWAYS IMPORTANT in numerical simulations, but one cannot always get what one wants. In the methanol-in-air case considered previously, the results are not stable. Particularly 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.

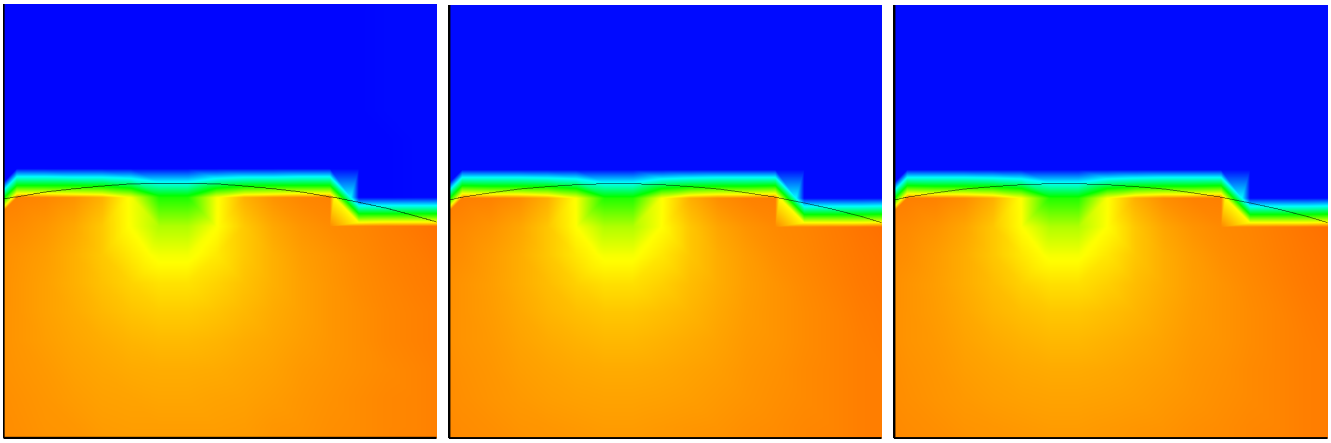
As the pressure field is the most unstable, considering the numerical methods used for calculating the pressure is an obvious starting point. In the present work, the pressure is calculated by solving a Poisson equation, the result of using a projection method to decouple the incompressible Navier-Stokes 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, and stands for Incomplete Lower-Upper preconditioning. These different solvers were tested on the methanol-in-air case, the results are shown in Figure 24.

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 24 all used a CFL number of 0.3, which is already quite low. Decreasing it further to 0.1 gave the results in Figure 25. It is seen in this figure that decreasing the CFL number reduces the erroneous pressure, but does not remove it completely. Reducing the CFL number further would be too computationally intensive.

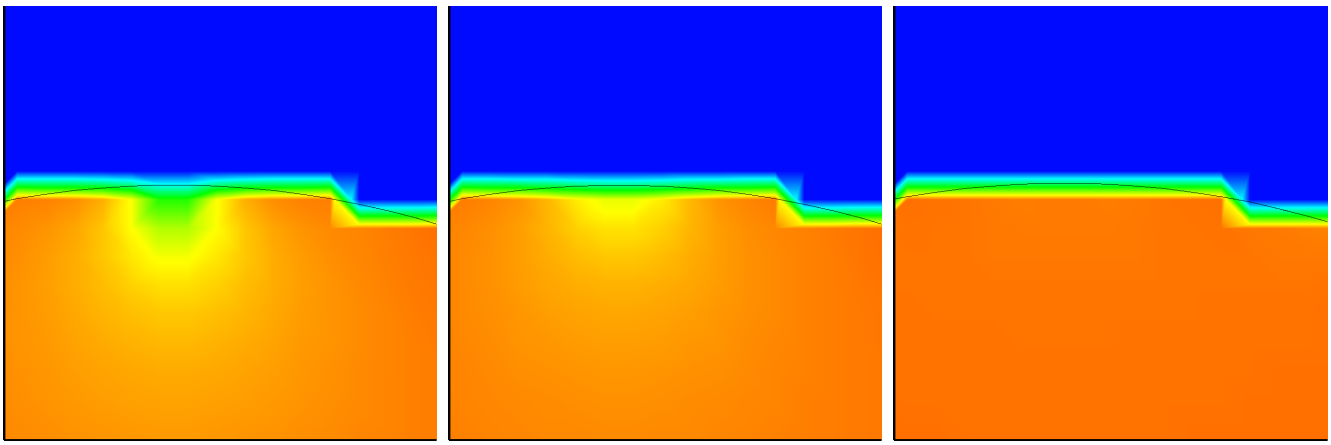
The conclusion of this is that the methanol-in-air case is too unstable for the present codes. The (particle) Reynolds number in this case is $Re \approx 430$, and the Weber number is $We \approx 2.1$, at the time of impact with the surface. The density ratio between the two fluids is $760:1$. This Reynolds number is quite high, as is the density ratio, so new simulations were considered with a lower Reynolds number and density ratio.

Report switch to water-in-decane, experiments there



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

Figure 24: Comparison of different Poisson solvers on an unstable case. The top of a falling bubble of methanol in air is shown, colored according to the pressure. The variation in pressure inside the bubble is unphysical, appearing and disappearing again in just 50 time steps.



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

Figure 25: 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 prior to 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.

5.4 Results from LOLEX on new cases

CONSIDERING THE WATER-IN-DECANE case presented above, several simulations were performed using the given data for fluid densities etc. Some simulations were performed to match the experiments in [40], where the bubble is gently placed onto the interface. Other simulations were performed using a higher initial fall height, such that the droplet has a non-zero velocity when impacting the surface.

These simulations, with a lower density difference and higher viscosity in the surrounding fluid, proved to be much more stable than the methanol-in-air simulations. The pressure field is shown in Figure 26. 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.

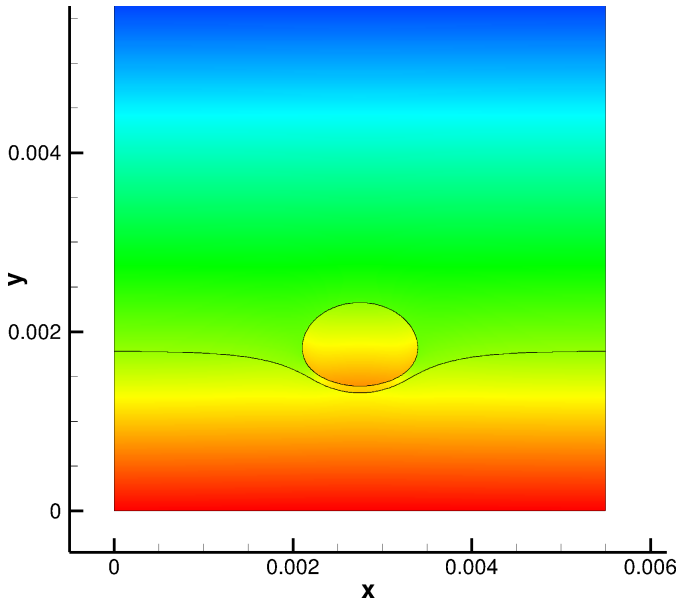


Figure 26: Contour plot of the pressure field just before merging. The pressure field is seen to be symmetric and sensible.

After this test case was seen to be stable, at least regarding the pressure, the effects of grid refinement and domain width was studied.

Complete diagonal simulations, report results

Jetting simulation: pressure wave! Magic happens! Is it physical or numerical? We don't know...

Complete water-decane simulations, report results

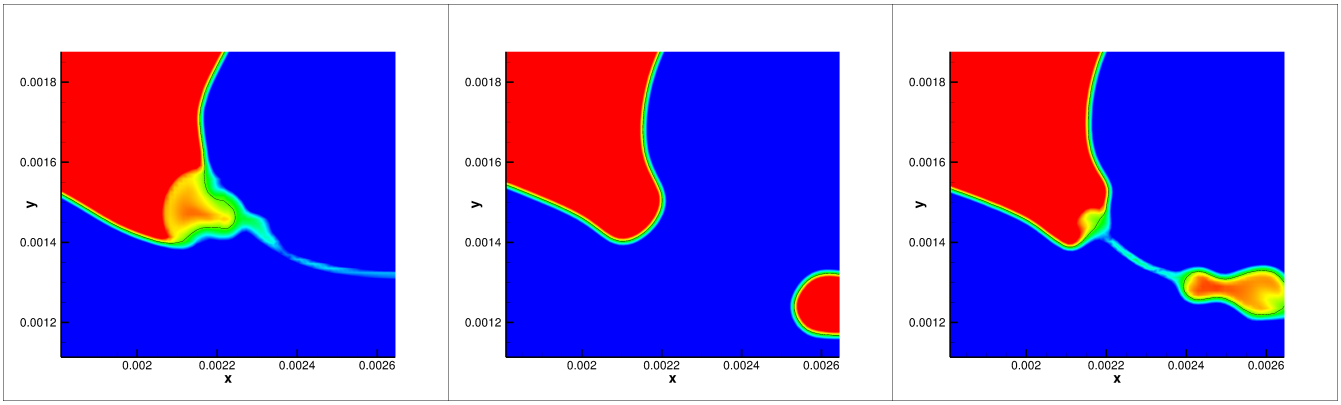
Complete axisymmetric simulations, report results

Discuss difference 2D/axisymmetric, Venturi effect

5.5 Effects of reinitialization on merging

New section

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 27. Here the level-set function of a droplet merging with a pool is shown for the 2D case. The 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. 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. 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. In Figure (b), reinitialization was used every 50 time steps. Completely turning off reinitialization turns out to cause its own problems, as seen in Figure (c). Thus, reinitialization should be performed about every 50-100 time steps. Tests using every 50 and every 100 time steps did not show any difference.



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

Figure 27: 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.

Discuss importance of accurate normal vectors for reinitialization, show results

§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

Write summary of water-decane experiments

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. 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. [40]. These experiments consider fluid-fluid interfaces between water and various light fluids, e.g. decane. Falling drops of water are seen to merge with or bounce on a pool of water. These experiments were the inspiration for the water-in-decane simulations above.

The main experimental technique used for studying droplet-pool impacts is high-speed shadowgraphy using a laser as light source. For a detailed description of such an experimental setup, the reader is referred to the PhD thesis by He Zhao [41]. If nothing else is stated, the images in this chapter are from this PhD thesis. The purpose of this chapter is to briefly demonstrate the main situations that are observed experimentally, and to compare with the current numerical simulations where possible. The various cases are sorted after the perceived difficulty in simulating them numerically; the first case has already been simulated, and the last case is probably too difficult to simulate using the present numerical methods.

6.2 Droplet merging with pool

The first class of experiments presented here is that of droplets merging completely with the pool of liquid. This corresponds to the numerical simulations presented in the previous chapter. According to the flow characterization presented in [41, 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 [41] 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. Part of the experimental observations of this droplet are shown in Figure 28. Comparing this to Figure 23, 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 Figure 23 (d) and in the middle frame in Figure 28 is encouraging. The formation of a surface wave also looks to be similar.

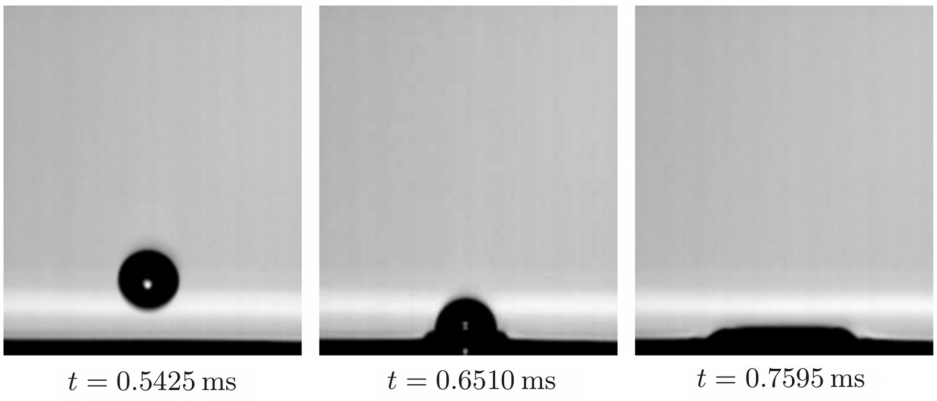


Figure 28: High-speed imagery of a 0.3 mm diameter methanol droplet merging completely with a pool of methanol at a velocity of 2.2 m/s. Figure from [41, Figure 5.11].

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 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.

6.3 Droplet partially merging with pool

All numerical simulations presented in this work end up with the droplet merging completely with the pool, or the simulation crashing before this can be established. 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 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 29. 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 the previous chapter are overcome, and would be a very nice result if the numerical simulations mathced the experimental results.

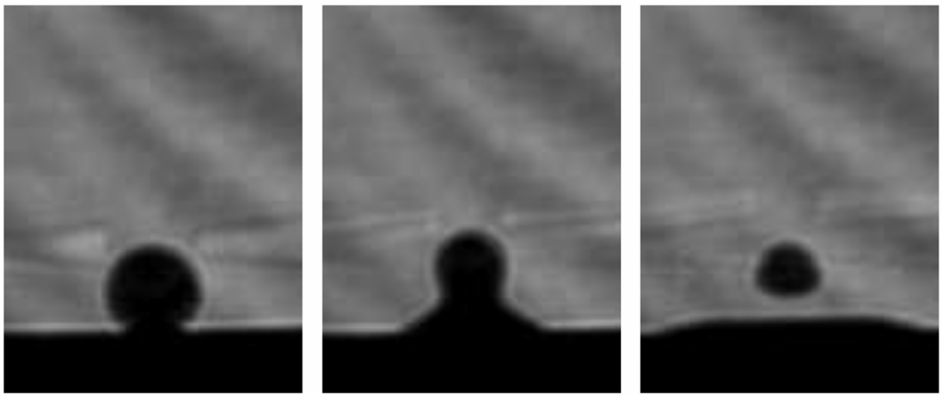


Figure 29: High-speed imagery of a 0.12 mm diameter water droplet merging partially into a pool of water, at a velocity of 0.29 m/s. The droplet pinches off, and leaves a smaller droplet behind. Figure from [41, Figure 5.17].

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. With this fact in mind, the author is not very optimistic about attaining such results numerically, but an attempt will perhaps be made. The experimental result presented in Figure 30 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, underscoring the author's pessimism towards simulating this case numerically.

6.4 Droplet bouncing off the pool

THE FINAL EXPERIMENTAL CASE presented here is the droplet bouncing off the pool. The author is near-certain that this result cannot be replicated numerically 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 (it is e.g. treated as incompressible) [42]. 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

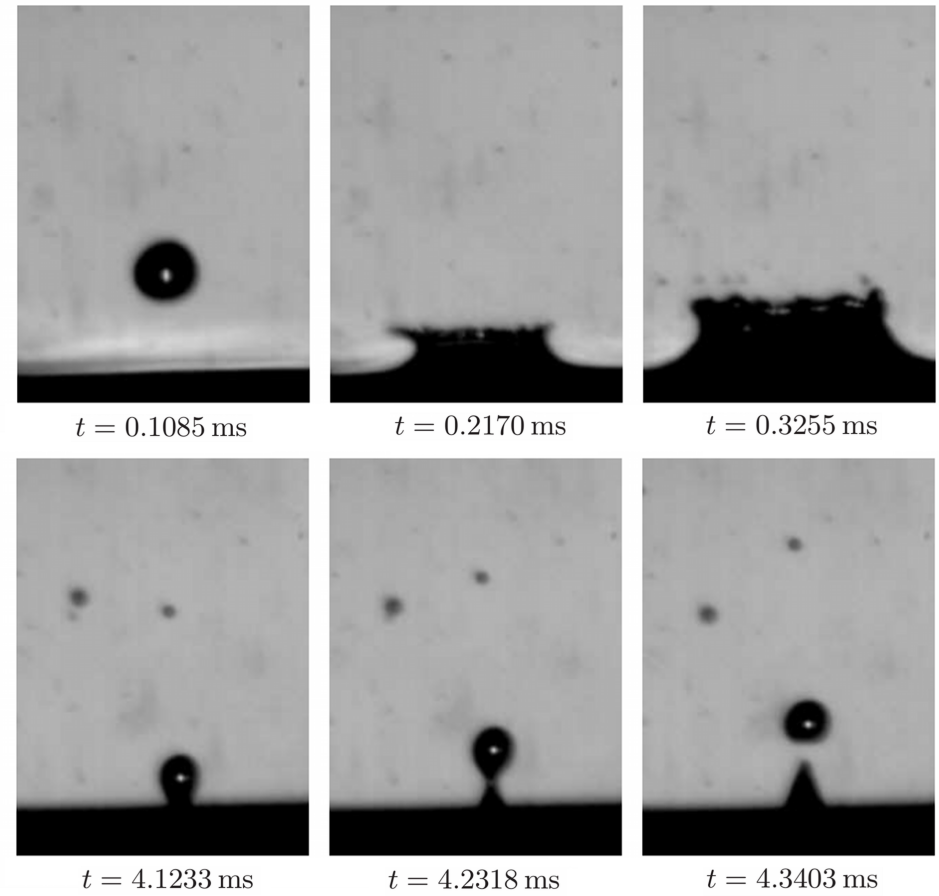


Figure 30: High-speed imagery of a 0.26 mm diameter n-pentane droplet impacting a pool of n-pentane at 5.9 m/s. After some time, the surface of the pool rebounds and ejects a jet. Note that some time has passed between the first and the second row of images. Figure from [41, Figure 5.6].

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 31. 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 a remarkable feat.

6.5 Concluding remarks

THE EXPERIMENTAL RESULTS presented in this section span a range of initial conditions that is currently larger than that available to numerical simulations. Contrasting with this is the ultimate,

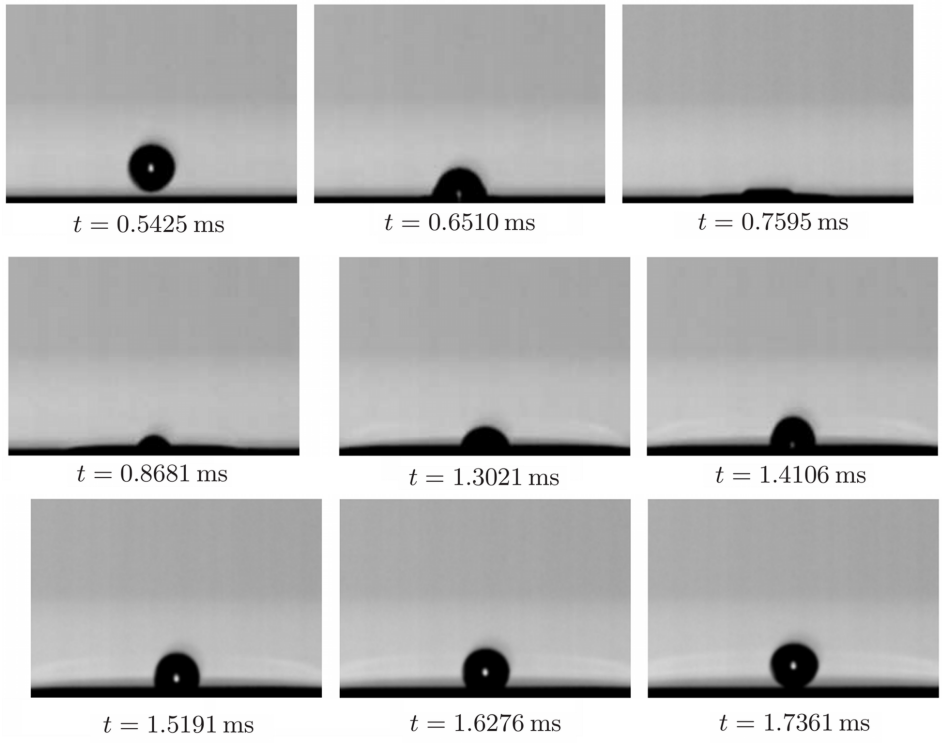


Figure 31: High-speed imagery of a 0.24 mm diameter 1-propanol droplet impacting a 1-propanol pool at 1.14 m/s, and bouncing back again. Note that several frames have been removed between the first and second frames in the second row, as indicated by a larger gap. Figure from [41, Figure 5.14].

perhaps optimistic goal of the numerical effort: to replace experimental work, which is costly and time-consuming, requires attention to HSE for most working fluids, and provides much less 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.

§7 Concluding remarks and future prospects

7.1 Conclusions

WITH THE PRESENT NUMERICAL SIMULATIONS ...

Fix lettrine indentation for W and A.

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

Mention increasing resolution of ϕ while keeping N-S resolution low

Mention filtering approach as in e.g. Vliet and Verbeek, SCIA93LVPV.pdf

References

[1] Norwegian Petroleum Directorate. FactPages V.2 (2011). Available online. [1]

[2] Gisvold, M. Eventyret på Melkøya. *Gemini Research Magazine* (2004). Avaliable online. [1]

[3] Tamura, I. *et al.* Life cycle CO2 analysis of LNG and city gas. *Applied Energy* **68**, 301 – 319 (2001). Available online. [1]

[4] 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. [1, 3.1, 3.2, 3.3, 5, 4.1.1, 4.3, 4.4]

[5] 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. [1]

[6] 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. [1]

[7] Osher, S. & Fedkiw, R. P. Level set methods: An overview and some recent results. *Journal of Computational Physics* **169**, 463 – 502 (2001). Available online. [1, 2]

[8] 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. [1, 2.1]

[9] Hartmann, D., Meinke, M. & Schröder, W. The constrained reinitialization equation for level set methods. *Journal of Computational Physics* **229**, 1514 – 1535 (2010). Available online. [2.1]

[10] 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. [2.1]

[11] 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. [2.1]

[12] Smereka, P. Semi-implicit level set methods for curvature and surface diffusion motion. *Journal of Scientific Computing* **19**, 439–456 (2003). Available online. [2.1, 3.1, 4.1.1, 4.2]

[13] 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. [2.1]

[14] 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. [2.1, 4.4, 5.1.1]

[15] Sussman, M., Smereka, P. & Osher, S. A level set approach for computing solutions to incompressible two-phase flow. Technical Report, Department of Mathematics, UCLA (1994). Available online. [2.2, 2.2, 2.2]

[16] Chopp, D. L. Computing minimal surfaces via level set curvature flow. *Journal of Computational Physics* **106**, 77 – 91 (1993). Available online. [2.2]

[17] Lervåg, K. Y. *Simulation of two-phase flows with varying surface tension*. Master's thesis, NTNU (2008). Available online. [2.3, 2.4, 2.6]

[18] 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. [2.3]

[19] Rottmann, K. *Matematisk Formelsamling* (Spektrum Forlag, 2003). [2.3]

[20] Kraichnan, R. H. Inertial-range transfer in two- and three-dimensional turbulence. *J. Fluid Mech.* **47**, 525–535 (1971). Available online. [2.3]

[21] 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. [2.4]

[22] 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. [2.4]

[23] Jiang, G.-S., Shu, C.-W. & L, I. Efficient implementation of weighted ENO schemes. *J. Comput. Phys* **126**, 202–228 (1996). Available online. [2.5, 2.6]

[24] 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. [2.5]

[25] 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. [2.5, 2.6]

[26] Versteeg, H. & Malalasekera, W. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method (2nd Edition)* (Prentice Hall, 2007), 2 edn. [2.5]

[27] Hansen, E. B. *Numerical simulation of droplet dynamics in the presence of an electric field*. Ph.D. thesis, NTNU (2005). [2.6]

[28] 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. [3.1]

[29] Lervåg, K. Y. Calculation of the interface curvature and normal vector with the level-set method (2012). To be submitted. [3.1, 4.3]

[30] Salac, D. & Lu, W. A local semi-implicit level-set method for interface motion. *Journal of Scientific Computing* **35**, 330–349 (2008). Available online. [3.1, 4.4]

[31] Lervåg, K. Y. & Ervik, Å. Curvature calculations for the level-set method. In *ENUMATH 2011 Proceedings Volume* (Leicester, England, 2011). [3.1]

[32] 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. [3.2, 3.4.2, 4.1, 4.2]

[33] Adalsteinsson, D. & Sethian, J. A. A fast level set method for propagating interfaces. *Journal of Computational Physics* **118**, 269 – 277 (1995). Available online. [3.3]

[34] 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. [3.4.2, 7, 3.4.2]

[35] Ferger, W. F. The nature and use of the harmonic mean. *Journal of the American Statistical Association* **26**, pp. 36–40 (1931). Available online. [4.1.1]

[36] Newman, T. S. & Yi, H. A survey of the marching cubes algorithm. *Computers and Graphics* **30**, 854 – 879 (2006). Available online. [4.1.1]

[37] 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. [4.3]

[38] 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. [4.4]

[39] 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. [4.4]

[40] 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. [5.4, 6.1]

[41] Zhao, H. *An Experimental Investigation of Liquid Droplets Impinging Vertically on a Deep Liquid Pool*. Ph.D. thesis, NTNU (2009). [6.1, 6.2, 28, 29, 30, 31]

[42] Qian, J. & Law, C. K. Regimes of coalescence and separation in droplet collision. *Journal of Fluid Mechanics* **331**, 59–80 (1997). Available online. [6.4]

Appendices

The routines presented here constitute the main routines of the current method. Their interdependence is shown in Figure 32 for the case of curvature calculation. For the routines `boun_extra_small`, `boun_extra_large`, `patdown` and `reinit_smallphi`, only interfaces are given (Listing 4) since they have been described previously.

Appendix A

Flowchart for LOLEX curvature calculations

Update the code in appendices to the final version

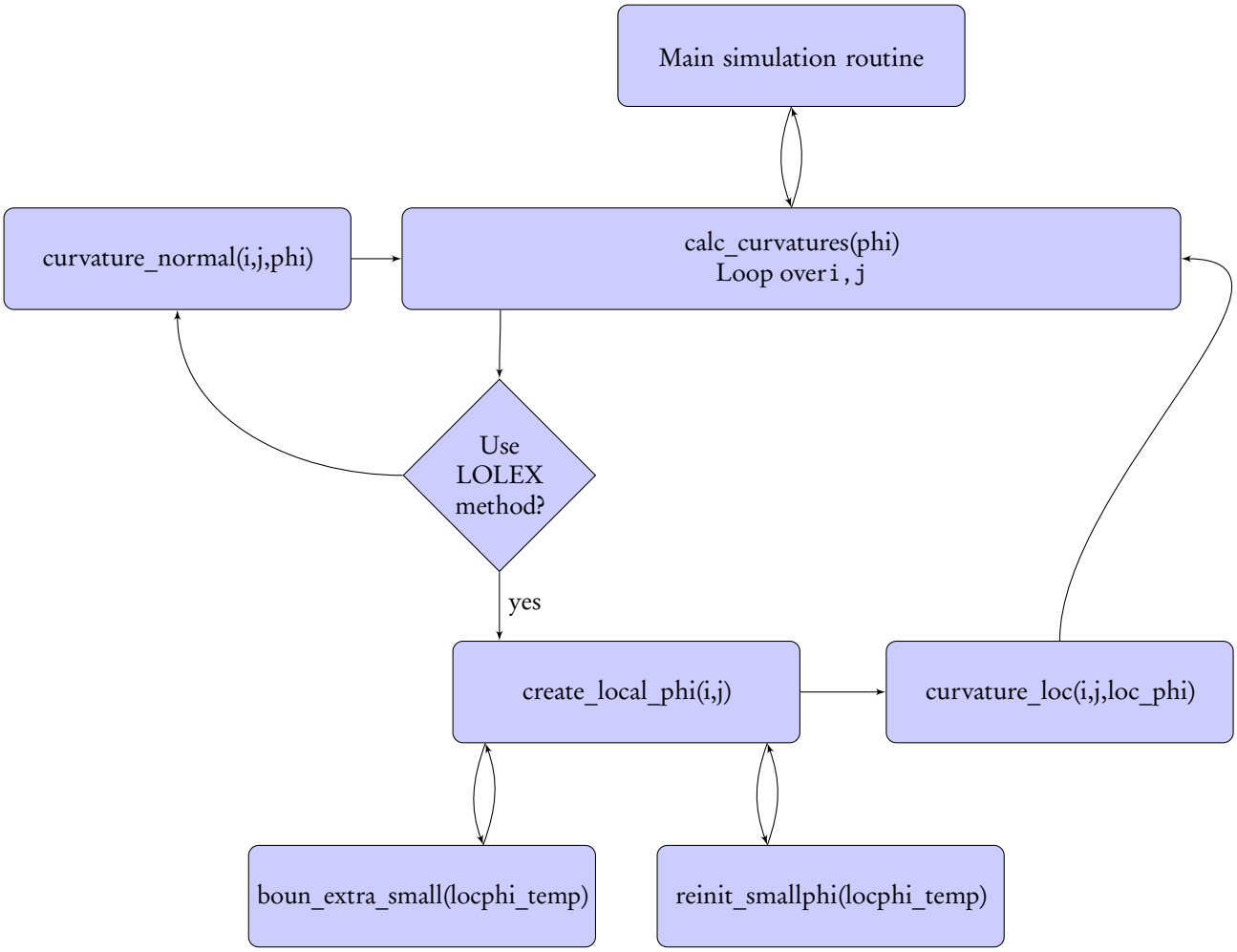


Figure 32: Flowchart of curvature calculation routines.

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 to 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
        !
        ! Present 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.
  implicit none
  integer, intent(in)   :: i,j
  !Local variables
  integer :: i0,j0,bodyno,neigh_bodies,pointcase,neighs,n,s,e,w,&
             body,i1,j1,radi,bodypoints,j2
  integer, parameter :: unchecked=-10, nobody=-5, dep=1, indep=0, removed=-1
  integer, dimension(ilmax,jlmax) :: bodies, depend
  real, dimension(0:ilmax+1,0:jlmax+1) :: 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,k).
  !
  ! lookphi is phi in the square we look at. radi is the "radius" of square
  radi=(ilmax-1)/2 + 1 ! + 1 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 = 1, jlmax
    do i0 = 1, ilmax
      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 three points, forget it
          if (bodypoints < 3) then
            ! All the body is within the 5x5 square around i0,j0
            ! Remove all markings with bodyno in this square.
            !write(*,*) "Removing body of size", bodypoints
            do j1 = max(1,j0-2), min(jlmax,j0+2)
              do i1 = max(1,i0-2), min(ilmax,i0+2)
                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.
  locphi_temp(:,:,:,:) = 2*dx(1)
```

```
    !
    ! Loop over the bodies present (main loop)
    do body = 1, loc_bodies
      ! Loop over the 7x7 square, and copy phi where appropriate.
      do j0 = 1, jlmax
        do i0 = 1, ilmax
          if (bodies(i0,j0) == body) then
            ! Inside this body, copy
            locphi_temp(i0,j0,body) = lookphi(i0,j0)
          elseif (bodies(i0,j0) == nobody .and. depend(i0,j0) == indep) then
            ! Outside bodies, and independent region, copy
            locphi_temp(i0,j0,body) = lookphi(i0,j0)
            !
          elseif (depend(i0,j0) == dep) then
            !
            ! Explicit reconstruction following Adalsteinsson et. al.
            !
            pointcase = 0
            approx_d = 0
            ! find out which case this is
            neighs = 0
            n = 0; s=0; e=0; w=0; ! north, south, east, west
            ! Check points north and south
            do i1 = max(1,i0-1), min(ilmax,i0+1), 2
              if (bodies(i1,j0) == body) then
                neighs = neighs + 1
                if (i1 == max(1,i0-1)) then
                  s = 1
                else
                  n = 1
                end if
              end if
            end do
            ! Check points east and west
            do j1 = max(1,j0-1), min(jlmax,j0+1), 2
              if (bodies(i0,j1) == body) then
                neighs = neighs + 1
                if (j1 == max(1,j0-1)) then
                  w = 1
                else
                  e = 1
                end if
              end if
            end do
            ! Set pointcase
            ! Case 1 (Case a)
            if (neighs == 1) then
              pointcase = 1
            else if (neighs == 3) then
              pointcase = 3
            else if (neighs == 2) then
              if (n+e == 2 .or. n+w == 2 .or. s+e == 2 .or. s+w == 2) then
                ! Corner case
                pointcase = 2
              else
                pointcase = 4
              end if
            end if
            !
            ! Compute the distance
            if (pointcase == 1) then
              ! Set (i1,j1) to the relevant neighbour
```

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

```
241             if (w==1) j1 = j0-1
                ! Calculate s1 and s2, take their minimum.
243             s1 = lookphi(i0,j0)/(lookphi(i0,j0)-lookphi(i1,j0))
                if (i1 /= i0) i1 = i0+1
245             if (j1 /= j0) j1 = j0+1
                s2 = lookphi(i0,j0)/(lookphi(i0,j0)-lookphi(i1,j0))
247             approx_d = min(s1,s2)
              else
249             write(*,*) "Error: independent point was marked as dependent"
                approx_d=0.5
251           end if
              !
253           ! Set the approximate distance as the value in localphi
              !write (*, *) "Distance set to", approx_d*dx(1)
255           locphi_temp(i0,j0,body) = approx_d*dx(1)
            else if (bodies(i0,j0) == removed) then
257           ! A body has been removed. This value will be overridden by reinit
              ! anyway, so the default 2*dx(1) is okay. Do nothing.
259         else
              ! This means that we hit a point belonging to the other body. Do
261           ! nothing.
            end if
263       end do
        end do
265     !
        if (loc_bodies > 1 .or. lremoved_body) then
267       call boun_extra_small(locphi_temp(:,:,body))
          call reinit_smallphi(locphi_temp(:,:,body))
269     end if
      end do
271   !
      ! After finishing this, including reinitialization, we do not need the full
273   ! 7x7 square any more. The kinks in phi have been removed, so we can revert
      ! to 3x3.
275   !
      loc_phi(:,:,:) = locphi_temp(3:5,3:5,:)
277 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
  real, dimension(0:ilmax+1,0:jlmax+1), intent(in) :: phi
  integer, dimension(ilmax,jlmax), intent(inout) :: bodies
  integer, intent(inout) :: bodypoints
  integer, intent(in) :: i0, j0, bodyno
  ! Local variables
  integer :: i,j
  integer, parameter :: unchecked=-10
  !
  ! Increment our counter of points in this body
  bodypoints = bodypoints + 1
  ! Loop over all the eight neighbours of (i0,j0)
  do i = max(1,i0-1), min(ilmax,i0+1)
    do j = max(1,j0-1), min(jlmax,j0+1)
      ! Skip the centre of the 9-cell square
      if ( i /= i0 .or. j /= j0) then
        if ( bodies(i,j) == unchecked .and. phi(i,j) < 0.0 ) then
          ! Mark this cell as belonging to body no. bodyno
          bodies(i,j) = bodyno
          ! Call this subroutine again on this interesting cell
          call patdown(phi,bodies,bodyno,bodypoints,i,j)
        end if
      end if
    end do
  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
    logical :: locrein=.true.,  l_updated_s=.false.
    real :: cfl_reinit=0.5, tau_reinit=8.0
    integer :: maxit_reinit=3
  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
  integer,save :: index_debug=0
  !
  ! 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);
    ! Weighted harmonic mean:
    !curvature_loc = (phi1+phi2)/(phi1/k2 + phi2/k1)
    ! Weighted arithmetic mean:
    curvature_loc = (phi1*k2 + phi2*k1)/(phi1+phi2)
  else
    write(*,*) "Error!!! Curvature average does not support @>@2 or 0 bodies!"
    write(*,*) "Returning zero curvature, results will be wrong!"
    curvature_loc = 0
  end if
  !
end function
```