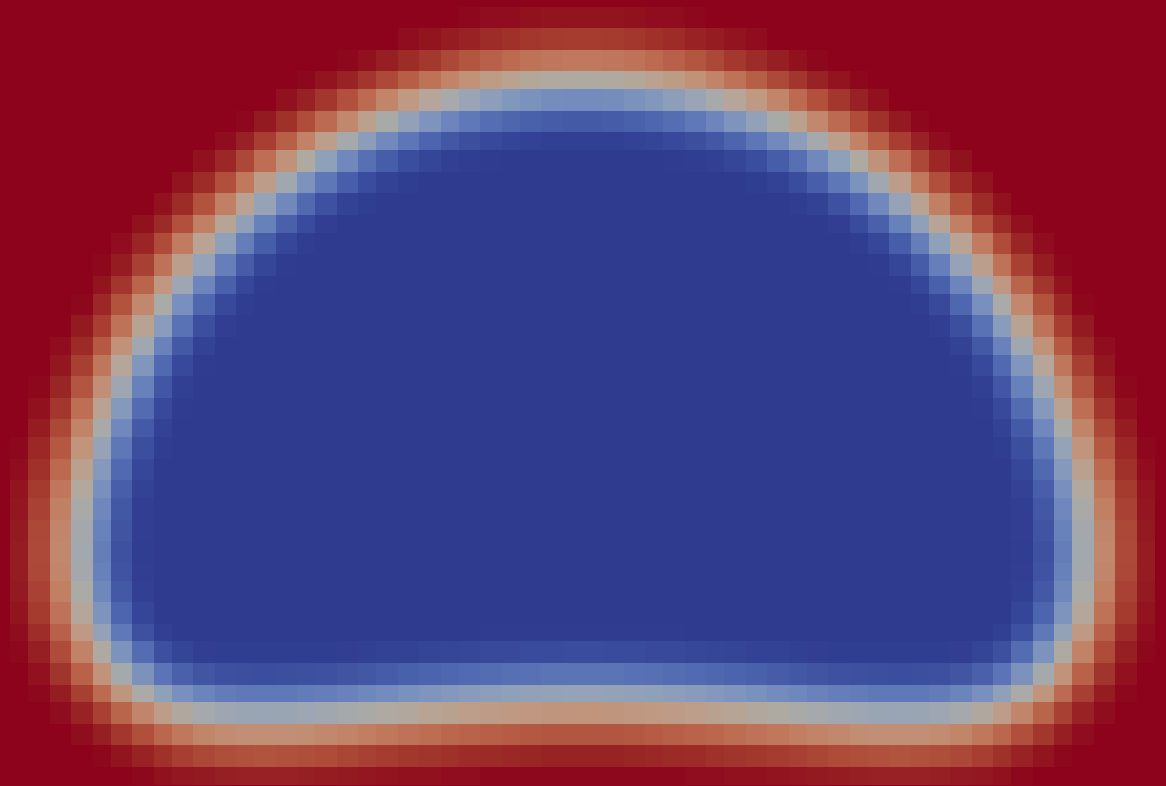
Convolution in Multiphase Flow Simulations

Improving Interface Curvature Estimations

E.M. Spaans

Delft University of Technology



TU Delft

Convolution in Multiphase Flow Simulations

Improving Interface Curvature Estimations

by

E.M. Spaans

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 4, 2018 at 2:00 PM.

Student number:	4429710	
Project duration:	September 4, 2017 – July 4, 2018	
Thesis committee:	Prof. dr. ir. C. Kleijn,	TU Delft, supervisor
	Dr. D. van der Heul,	TU Delft, supervisor
	Ir. K. van As,	TU Delft, supervisor
	Prof. dr. ir. A. Heemink,	TU Delft
	Dr. S. Kenjeres,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft Delft University of Technology

Abstract

The accurate approximation of the surface tension force is paramount for continuum surface models in the field of computational fluid dynamics for multiphase flow where surface tension is relevant. This involves being able to accurately calculate the curvature at the interface. This study focuses on the use of convolution in smoothing the VOF colour field in order to obtain better approximations of the curvature.

Given the sudden jump in values of the VOF colour field, the calculation of its derivative for the curvature is sensitive to errors, given the large values of high order terms that determine the truncation error. To deal with this problem, convolution of this abruptly varying field can be used to create a smoother transition. The curvature approximation of a circular interface improved as the support of the convolution was increased. It was proven analytically that, for these interfaces, the original curvature is retrieved from the convoluted field. Interfaces along which the curvature varies were also considered, and it was found that there is a critical convolution support that minimizes the error in the curvature, given that the choice of the support length can modify the curvature that is estimated.

An algorithm was implemented in OpenFOAM that calculates the convolution of the VOF colour field. The resulting smoothed field was then used to calculate the curvature, which is needed for the surface tension force of the system. The simulations of a two-dimensional rising bubble resulted in more accurate results for the circularity and the rising velocity, when compared to the original OpenFOAM implementation with no smoothing. With the convolution algorithm, the terminal velocity deviated only 0.01% from a well-accepted benchmark case, a great improvement when compared to the 4.2% difference when no smoothing was used. However, simulations of a static bubble in zero-gravity rapidly resulted in unphysical flow, manifested as a wavy interface, when a convolution support larger than 2 cells was chosen. An improvement of the estimation of the surface tension force direction may be needed for this behaviour to disappear.

Contents

1

Introduction

Fluid-fluid multiphase flow is concerned with the study of the motion and interaction of two or more immiscible fluids. These interactions are ubiquitous in nature, e.g. rainfall, ocean waves and underground water circulation. It is also of great importance for industry, e.g. fuel sprays for combustion engines, ink jet systems, cavitation and evaporation [1], and spray drying [2]. Understanding the underlying dynamics is crucial for the prediction of the behaviour of such systems.

Multiphase flow is an important component in the field of computational fluid dynamics (CFD), which intends to predict the motion of a multiphase system by numerically approximating the solution of the governing equations. Despite its broad use and the impact it has had on the field, it remains a challenge to accurately estimate the evolution of such flow. There are several methods that have been developed to solve these problems, and to keep track of the interface. The most common ones include Volume of Fluid (VOF) methods [3], Height Functions (HF) [4] and Level Set (LS) [5]. Some approaches also involve a combination of two methods, in order to take advantage of the qualities of each one [6]. The current study focuses on the VOF method.

In some cases, surface tension plays an important role in the development of the flow. Here, one of the main issues is how to model the surface tension that arises at a curved interface between fluids. Brackbill et al. [7] proposed the Continuum Surface Force (CSF) model, which models the surface tension as a force spread over a region of finite thickness. It is the most accepted model to resemble the action of surface tension within the VOF formulation. A key factor in the modelling of this force is being able to correctly approximate the curvature of the interface, since this influences the magnitude of this force. Moreover, the interface normal vector is important for the direction of the force.

The main advantages of the VOF method are its mass conserving property [8], and its straightforward implementation. Nevertheless, it suffers from poor curvature estimations, as the VOF describes an abruptly varying field. The curvature is estimated from finite differences of this field, and the magnitude of high order terms, which determine the accuracy of the curvature, are large for this steep field. This generally leads to inaccurate surface forces, and the generation of so-called *spurious currents* [7] (non-physical velocities which arise as a consequence of the numerical mismatch of the surface tension force and the pressure). To address this issue, various efforts have been made to improve the curvature estimations. Some methods involve a reconstruction of the interface based on the volume fractions [9]. Another approach is to smooth the abruptly varying VOF field, and calculate the curvature based on this new field. Lafaurie et al. [10] developed the Laplacian filter in order to achieve this smoothing. Among others, convolution is also used to achieve this [11, 12], but there are still numerical challenges encountered with this smoothing method.

OpenFOAM is a well-known open-source CFD solver [13]. Currently, it estimates the curvature by using the steep VOF field. There are contributions from the CFD community that have implemented a Laplacian filter [14], which has led to a reduction of spurious currents. Nevertheless, these are in many cases still significant in magnitude. This study focuses on the impact of using convolution to smooth the VOF field on curvature estimations for the simulation of two-phase flow in OpenFOAM. To achieve this, a convolution algorithm was implemented.

The following questions will be addressed:

1. *How does convolution affect the estimation of the interface curvature?* Convolution as a smoother may change the local value of the curvature. This effect will be considered for an interface with constant curvature

(circle), and an interface along which the curvature varies (ellipse).

2. *Does an improved curvature estimation by convoluting the VOF field yield more accurate multiphase simulations in OpenFOAM?* Test cases of a static bubble in zero-gravity and a rising bubble are considered. In the former, the spurious currents are quantified, and in the latter the rising velocity and circularity. The results are compared with benchmark cases.

Chapter 2 reviews the physics of two-phase flow relevant to this study, along with a description of the numerical model. Chapter 3 explains how convolution works, and the convolution filters that will be considered. Next, Chapter 4 evaluates the effect of convolving the VOF field for interfaces of various shapes, followed by the results of simulations of a static and a rising bubble case in OpenFOAM in Chapter 5. Chapter 6 analyses the unexpected interface wave that emerged during the simulation when the convolution method was used, and Chapter 7 finalizes by presenting the conclusions of the study.

2

Principles of Two-Phase Flow Modelling

In this chapter, the principles of two-phase flow modelling are introduced. The physics of two-phase immiscible, incompressible flow is outlined in Section 2.1, whereafter the discretisation of the domain with the use of the Volume of Fluid (VOF) method is presented in Section 2.2. Section 2.3 follows with an overview of the mathematical model that describes the presented flow, where the Continuum Surface Force (CSF) model is introduced. The chapter concludes by presenting the equations for the calculation of the curvature in Section 2.4.

2.1. Physics of Two-Phase Flow

The theory of two-phase flow attempts to describe the evolution of two immiscible fluids brought into contact. It studies, among other aspects, the displacement and interaction of these two phases. In order to correctly predict how a specific initial distribution of two fluids will develop in time, fundamental physical laws are used. To this end, Newton's second law and the laws of conservation of mass and energy are applied to the system. The current study focuses only on the spatial displacement of the two-phase flow, such that thermal energy transfer is ignored in the present analysis. It is assumed that the fluids are Newtonian, incompressible and immiscible. The main focus is on fluid droplets immersed in a second fluid.

2.1.1. Governing Equations

Consider a domain with two fluids separated by an interface, as shown in Figure 2.1.

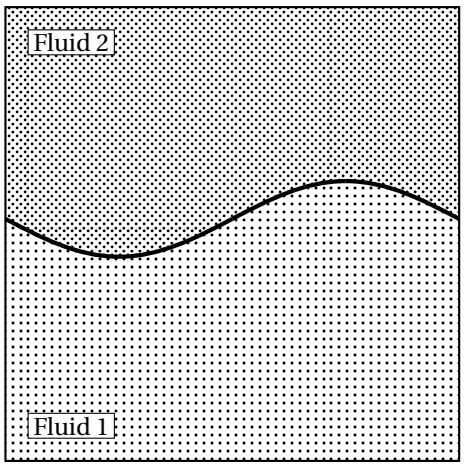


Figure 2.1: Illustration of a two-phase flow distribution.

The momentum equation that arises from the application of Newton's second law of motion to a Newtonian, incompressible volume element in fluid i at either side of the interface results in the well-known

Navier-Stokes momentum equation [2]

$$\rho_i \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) = \mu_i \nabla^2 \mathbf{u} - \nabla p + \sum \mathbf{f}_{\text{ext}}, \tag{2.1}$$

where ρ is the density, $\mathbf{u}$ the velocity, μ the dynamic viscosity and $\mathbf{f}_{\text{ext}}$ a force per unit volume. The first term on the left-hand side of Equation (2.1) accounts for the local transient variation of the velocity field and the second term for the changes due to the motion of the fluid. The first two terms on the right-hand side represent the stresses present within the fluid, whereas the last term includes all the external forces that act on the system. This normally is formed by the gravitational force $\mathbf{f}_{\text{g}} = \rho \mathbf{g}$, with $\mathbf{g}$ the acceleration vector due to gravity, and in the present research the volumetric surface tension force $\mathbf{f}_{\text{s}}$ is also included. This is further explained in Section 2.3.

With the law of conservation of mass, the so-called continuity equation for a volume element can be derived, which in the case of incompressible flow is given by [2]

$$\nabla \cdot \mathbf{u} = 0. \tag{2.2}$$

Equations (2.1) and (2.2) together allow solving for the unknown velocity field $\mathbf{u}$ and the pressure p. Due to the discontinuity of fluid properties ρ and μ at the interface, Equation 2.1 needs to be solved for fluid 1 and fluid 2 separately, given appropriate boundary conditions. One of the conditions is for the normal and tangential direction of the velocity to be the same at the interface for both fluids. Secondly, the normal and tangential (also called shearing) stresses should be balanced at the interface. An approach to solving the system as a whole is presented in Section 2.3.

2.1.2. Surface Tension

The concept of surface tension plays an important role in understanding the interface development between two fluids in a two-phase system. When considering the composition of a fluid at a microscopic level, the cohesion forces between molecules is what keeps them together. For molecules located at the inner part of a fluid, on average these forces are the same in all directions, such that the net force on each molecule is zero (without considering any other external forces). However, for molecules located at the interface of the fluid, there is an imbalance in the forces at one side of the molecule, due to the fact that there are no alike molecules with which the molecule can bind with. This results in a net force pointing inwards, towards the fluid, which for a curved interfaces gives a pressure jump across the interface [15]. This pressure jump is given by the Young-Laplace equation:

$$\Delta p = \sigma \kappa, \tag{2.3}$$

where σ is the surface tension and κ the curvature. This pressure jump needs to be included in Equation (2.1), but need special treatment due to the locality of the force and the discontinuity in the pressure across the interface. This will be treated in Section 2.3.

2.1.3. Interface Shape

The shape that a bubble or droplet acquires in a specific two-phase flow setting depends on the forces that act on these particles, and the particle properties. This defines a wide variety of possible shapes a fluid can attain. In order to classify these into categories, and be able to predict the occurrence of each of these categories, important dimensionless quantities are used.

Reynolds number (Re): describes the ratio between inertia forces and viscous forces,

$$\frac{\rho_o d_i v_i}{\mu_o}. \tag{2.4}$$

Eötvös number (Eo): describes the ratio between gravitational forces and surface tension forces,

$$\frac{g d_i^2 \Delta \rho}{\sigma}. \tag{2.5}$$

Morton number (M): correlates the parameters of the two-phase flow,

$$\frac{g\mu_o^4\Delta\rho}{\rho_o^2\sigma^3}, \tag{2.6}$$

where ρ is the density, d is the characteristic diameter, v is the rising velocity of the droplet or bubble, μ is the dynamic viscosity and σ is the surface tension [16]. The indices i and o stand for *inner* and *outer*, whether it is a droplet surrounded by gas or a bubble surrounded by liquid.

Figure 2.2 depicts the different shape regimes attained by gravity driven two-phase flow given different values of the dimensionless quantities.

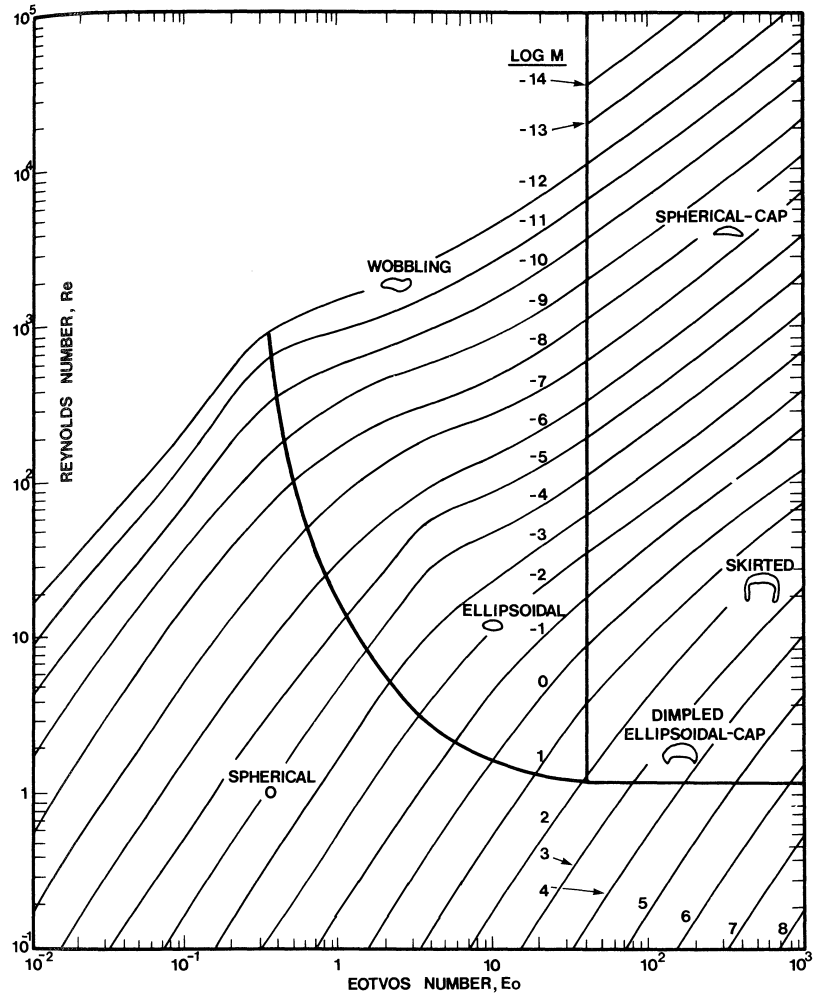


Figure 2.2: Shape regimes for two-phase flow in the presence of a gravitational field.[2]

Note how these numbers can influence the shape of a droplet or bubble. Consider for instance a combination of a relatively low Eo (around 0.3), as well as a low Re (around 1). Given that surface tension forces are more relevant than the gravitational forces, and the fact that viscous forces are approximately in balance with the inertia forces, the equilibrium shape constitutes a sphere. The external forces are not big enough to deform the droplet or bubbles to more complicated shapes, and the minimization of the energy results in the minimization of the surface area (such that the surface tension force is reduced), which is attained by a spherical shape. This situation occurs when the length is scaled down, since this results in a linear scaling of the tension forces, whereas other forces decrease at a higher rate [17]. The bottom-left region of Figure 2.2 depicts this behaviour.

More complex shapes occur as the external forces become significant, which occurs at higher Re and Eo. In these situations, the surface tension is no longer able to counteract the inertial/gravitational forces which act on the bubble or droplet, which results in ellipsoids, and caps with different shapes, as observed

in the right and upper regime of the presented figure. Note that the shape of a bubble evolves in time, as the properties of the bubble or droplet such as velocity changes over time, which results in a variation of for instance Re, and a shift in the shape regimes depicted in Figure 2.2. For extreme values of Re (generally for values over 10^4), the flow becomes turbulent, and viscous forces cannot keep up with the external flow field. In some of these cases, the single bubble or droplet we had in our system may breakup into parts. Similar consequences are observed when surface tension forces are not able to keep up with the buoyancy forces as a result of the gravitational influence, for high Eo.

For the modelling of this flow and to solve the discontinuities in fluid properties for Equation 2.1, the Volume of Fluid method is used, which is introduced next.

2.2. Volume of Fluid

The Volume of Fluid (VOF) method was introduced by Hirt and Nichols [3] as a way of tracking free boundaries, i.e. regions in space at which discontinuities occur. In the present case, this method is used to track the interface of two-phase flow. The function that characterizes this method is called the *colour function* γ:

$$\gamma = \begin{cases} 1, & \text{fluid 1,} \\ 0, & \text{fluid 2.} \end{cases} \tag{2.7}$$

To define the *VOF colour function*, the domain is first discretised into a mesh of any shape or size. For the present study, a 2D Cartesian mesh is considered, where the cell size in the x and y direction, Δx and Δy, are the same. This cell size will be called h. Consider now a two-phase flow distribution, where the interface between fluid 1 and fluid 2 is modelled. The value of the VOF colour function for each mesh element of this discretisation is defined as the fraction of fluid 1 present in the control volume of the grid element:

$$c_{ij} = \frac{\int\limits_{\text{cell(i,j)}} \gamma \, \mathrm{d}V}{\int\limits_{\text{cell(i,j)}} \mathrm{d}V}, \tag{2.8}$$

where (i, j) represent the cell indices in the x and y direction, respectively. In other words, c will attain a value of 1 in those mesh elements that are completely covered by fluid 1, whereas a value of 0 means that the mesh element is only covered by fluid 2. Mesh elements located along the interface between the fluid will receive values between 0 and 1, proportional to the amount of fluid 1 present in the cell. This definition is illustrated in Figure 2.3, where the value of the VOF colour function for a circle in a 2D Cartesian discretised domain is indicated in each cell.

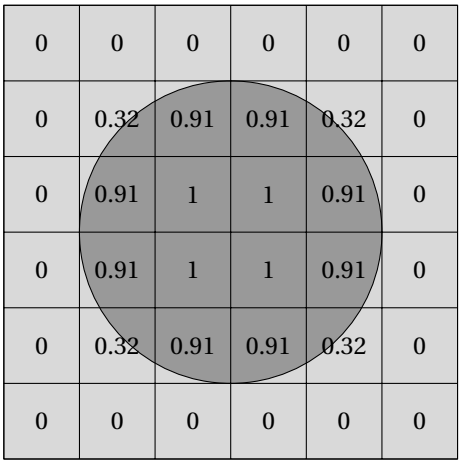


Figure 2.3: Values of the VOF colour function with a two decimal precision for a circle in a 2D Cartesian domain. The darker region represents fluid 1.

To find the approximate location of the interface given the VOF colour function, the normal direction to the boundary needs to be found. This is given by the direction in which the colour function changes the

most [3], since this is the direction of the gradient, which by definition is perpendicular to the tangent of the point on the interface. In this context, the normal of a cell (i, j) at the interface is therefore given by

$$\mathbf{n}_{ij} = \nabla^h c_{ij}, \tag{2.9}$$

where ∇^h emphasizes that the derivatives are calculated by a finite difference method. Dividing by $|\nabla^h c|$ would result in the unit normal vector $\hat{\mathbf{n}}_{ij}$. Together with the value of c in the present cell, the interface can be approximated. The major advantages of the VOF method include the relatively easy implementation for arbitrary mesh shapes [18], and the fact that only one value per cell needs to be stored to have information about the interface location. Moreover, the mass conservation property of this method is one of its principal assets [8]. However, because of the discrete nature of this field, the computation of the derivative of this function needs to be treated carefully.

To follow the evolution of the VOF colour function c in time, the field is advected with the transport equation, which in integral form is

$$\int \left[\frac{\partial c}{\partial t} + \nabla^h \cdot (c\mathbf{u}) \right] \mathrm{d}V = 0. \tag{2.10}$$

In the remainder of this and the following chapters, the subscript h will be omitted, where it should be clear from the context that finite difference methods should be used for the discrete valued function c.

2.3. Mathematical Model

After discretising the domain, the momentum and continuity equations (2.1) and (2.2) are solved for each volume element. As mentioned in Section 2.1, the jump in fluid properties at the interface posed a problem for solving Equation 2.1 as a whole. To solve this, for cells containing an interface, cell properties are calculated by a weighted average of the two fluids, with the weights given by the VOF colour function. In this way, the mean value across the control volume is used, and a gradual transition between fluid properties is obtained. The density and dynamic viscosity can be therefore written as

$$\rho = c\rho_1 + (1-c)\rho_2, \tag{2.11}$$

$$\mu = c\mu_1 + (1-c)\mu_2. \tag{2.12}$$

Due to the non-linearity of the Navier-Stokes momentum equation, only very simple scenarios can be solved analytically, and one has to recur very quickly to numerical methods to find the velocity and pressure of the setting.

Surface tension, which was introduced in Section 2.1.2, is modelled as an external force in the momentum equation. Therefore, an extra term $\mathbf{f}_s$ must be included on the right-hand side of Equation (2.1) to account for the surface tension force. In the VOF method, this force can be modelled as a volumetric force, i.e. acting over a finite region in space rather than over the infinitesimally thin surface. The most accepted method to model the surface tension that arises at the interface between two fluids is the Continuum Surface Force (CSF) model, developed by Brackbill et al. [7]. The subtlety of their approach was to spread the force resulting from surface tension over a region of finite thickness, instead of using its value as a boundary condition. In this way, the combination of the two fluids being considered can be seen as a whole, with surface tension acting as an external force over some small finite region in the domain. To this end, we need a characteristic function that distinguishes the fluids, and that changes continuously over the region where the surface tension force is spread. In the present study, this characteristic function is given by the VOF colour function, c. With this characteristic function, the volumetric surface tension force can be expressed as [7]

$$\mathbf{f}_s = \sigma\kappa\nabla c, \tag{2.13}$$

with σ the surface tension coefficient and κ the curvature of the interface. σ is assumed to be constant in the observed computational cell. The calculation of this curvature is treated in Section 2.4.1. Note that a force of this magnitude correctly models the pressure jump from the Young-Laplace equation (2.3). The locality of $\mathbf{f}_s$ is ensured by the gradient of the colour function. Since this function only varies at a region around the interface, it evaluates to zero elsewhere, where surface tension does not come into action. Figure 2.4 depicts the CSF approach. The region around the interface where the force is present is shown in white, and some vectors representing the direction of the force are given.

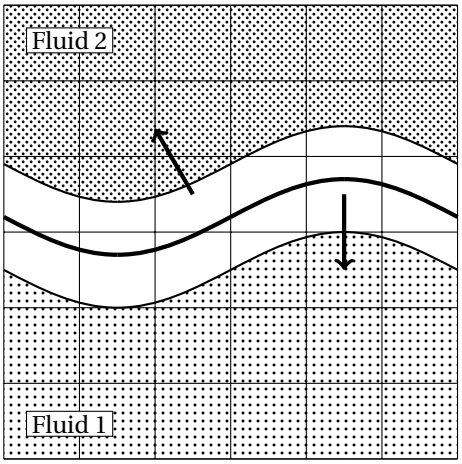


Figure 2.4: Illustration of a discrete two-phase flow distribution with the CSF approach to surface tension modelling. As an example, the force is represented as a vector acting on cell centres for two cells. This image is based on Brackbill et al. [7].

In conclusion, the final model that describes the current problem is given by the following system of equations:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t}+\mathbf{u}\cdot\nabla\mathbf{u}\right)=\nabla^2(\mu\mathbf{u})-\nabla p+\rho\mathbf{g}+\sigma\kappa\nabla c \tag{2.14}$$

$$\nabla\cdot\mathbf{u}=0, \tag{2.15}$$

with appropriate boundary and initial conditions, and where the interface is tracked with Equation (2.10). The fluid properties are given by Equations (2.11) and (2.12). After the discretisation of the domain, this system is solved in each computational cell. The main focus of this research lies on the calculation of the interface curvature κ. Accurate estimations of this parameter are needed in order to obtain accurate pressure and velocity fields from the equations describing the system, as noted by Francois et al. [9]. Failing to do so, the solutions become unphysical, due to the presence of the so-called *spurious currents*.

2.4. Curvature

2.4.1. Curvature Calculation

The curvature κ of an interface is a measure of the variation of the unit normal vector to the interface. Given a curve that is parametrised in terms of a variable t: $(x(t), y(t))$, where t runs along the arc length, the exact curvature can be calculated with

$$\kappa_{\text{exact}}=\frac{x'y''-y'x''}{(x'^2+y'^2)^{3/2}}. \tag{2.16}$$

In terms of the normal vector, it can be computed as the negative of the divergence of the unit normal vector [7]:

$$\kappa=-\nabla\cdot\hat{\mathbf{n}}. \tag{2.17}$$

Brackbill et al. [7] proposed a reformulation of this equation that produced better results:

$$\kappa=\frac{1}{|\mathbf{n}|}\left[\left(\frac{\mathbf{n}}{|\mathbf{n}|}\cdot\nabla\right)|\mathbf{n}|-\nabla\cdot\mathbf{n}\right]. \tag{2.18}$$

It was argued that, in this way, the contributions to the curvature came principally from where the gradient was maximum, as a consequence of differentiating the unnormalized normal vector. Consider the definition of $\mathbf{n}$ in terms of c in Equation (2.9). In that context, the normal vector was only defined at the interface. If one extends this definition over the whole domain, cells around the interface also receive a normal vector (the derivative is still non-zero here), and the rest of the cells' normal vector is zero. Then, one can define the normal vector in terms of the VOF field by

$$\kappa=\frac{1}{|\nabla c|^2}\left(\frac{\nabla c}{|\nabla c|}\cdot\nabla\right)\nabla c-\frac{1}{|\nabla c|}\nabla\cdot\nabla c. \tag{2.19}$$

For the two dimensional Cartesian case, this reduces to evaluating the following equation:

$$\kappa = \frac{1}{|\nabla c|^3}\left[\left(\frac{\partial c}{\partial x}\right)^2 \frac{\partial^2 c}{\partial x^2} + \left(\frac{\partial c}{\partial y}\right)^2 \frac{\partial^2 c}{\partial y^2} + 2\frac{\partial c}{\partial x}\frac{\partial c}{\partial y}\frac{\partial^2 c}{\partial x \partial y}\right] - \frac{1}{|\nabla c|}\left[\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2}\right]. \tag{2.20}$$

Since the VOF colour function is discrete, finite difference methods need to be applied in order to calculate the derivatives in Equation 2.20. To this end, a central difference stencil is used for the first derivative with a step size of $h = \Delta x = \Delta y$. For the second derivative, a central difference is applied twice with a step size of $h = \Delta x/2 = \Delta y/2$.

As mentioned earlier, direct calculation of the curvature with the VOF colour function results in high curvature errors. These errors arise mainly from the second order finite difference evaluation of the steep colour function in the equation for the curvature, as noted by Cummins et al. [8]. This occurs as a result of the large magnitude of higher order derivatives of the VOF colour function, which determine the magnitude of the truncation error of the finite difference approximation of the derivative. Moreover, the fact that the VOF tries models a discontinuous colour function γ poses problems for the differentiability of this field. There are several methods discussed in the literature to reduce these issues, amongst which the process of convolution stands out. The idea behind this is to create a smoother colour function, such that the transition region of the VOF field at the interface is increased. In this way, the field that describes the interface varies more gently, and the magnitudes of high order derivatives is decreased. The process of convolution is explained in further detail in Chapter 3.

2.4.2. Curvature Error

To determine the proximity of the calculated curvature values to the actual values, the L_2 and L_∞ norms will be used, which define the average curvature error across the entire interface and the maximum value this error attains, respectively. These error measures are given by

$$L_2 = \sqrt{\frac{\sum_{i=1}^{N}(\kappa_i - \kappa_{i,\text{exact}})^2}{N}} \tag{2.21}$$

and

$$L_\infty = \max_{i\in\{1,\dots,N\}} |\kappa_i - \kappa_{i,\text{exact}}|, \tag{2.22}$$

where the summation index i runs along the cells that form the interface, κ_i is the curvature calculated with the colour field c and $\kappa_{i,\text{exact}}$ is the exact curvature in the centre of an interface cell i, given in Equation (2.16).

3

Convolution Filters

The accurate calculation of the curvature of the interface by using the discrete VOF colour function is a challenge due to its steep, discontinuous nature. To improve this, convolution filters can be applied to the colour function in order to smooth out the values of this function. There are many parameters worth of consideration and some requirements needed when using a convolution filter. The principle of convolution will be introduced in Section 3.1. Section 3.2 mentions some requirements for convolution filters and introduces the convolution filters that are used the most in literature. Then, Section 3.3 provides a visualisation of the impact of convolution on a VOF colour field c. Section 3.4 concludes with an analytical derivation of the curvature estimation as a result of using convolution.

3.1. Convolution: the Principle

Convolution is an operation that builds a function by combining two other functions over a certain domain. It is calculated by multiplying the function that will be convoluted, say g, by shifted versions of a convolution kernel K, and finally integrating over space. In this way, a new function $\tilde{g}$ is constructed, where $\tilde{g}(\mathbf{x})$ becomes a weighted combination of different values $g(\mathbf{y})$, with $\mathbf{y}$ in the vicinity of $\mathbf{x}$. These weights are determined by the choice of the kernel K, which should be normalized. The extent to which values of $\mathbf{y}$ that are in the neighbourhood of $\mathbf{x}$ will be considered for the convolution integral are determined by what is known as the kernel support, which will be represented by Ω. For $\mathbf{y} \notin \Omega$, $K(\mathbf{y}) = 0$. The convolution operation is expressed by the $*$ sign, and is defined by

$$\tilde{g}(\mathbf{x}) = g(\mathbf{x}) * K(\mathbf{x}) = \int_{\Omega} g(\bar{\mathbf{x}}) K(\mathbf{x} - \bar{\mathbf{x}}) \, \mathrm{d}\bar{\mathbf{x}}. \tag{3.1}$$

The integral is limited to Ω since the integrand evaluates to zero outside this domain. Convolution integrals are extensively used in the current field of study to smooth an otherwise discontinuous function, as in the case of the VOF method explained in Section 2.2. The abrupt nature of the colour function c imposes difficulties when trying to calculate the derivative of this function, which is needed for computing the curvature of the interface between the two fluids. As Brackbill et al. [7] noted, errors of the magnitude of the curvature are present if the colour function c is not smoothed before calculations for the curvature are performed. By convoluting the colour function over a finite support, the volume fractions in the mesh elements are spread out, such that a gentle transition between cell values is obtained. This, in turn, ensures a more accurate estimation of the curvature at the interface cells.

Despite improvement in the curvature calculation when using the convoluted colour field, the error after convolution is found to diverge when the mesh is refined. The error in the curvature calculation starts to increase for a mesh size corresponding to about 9 cells per radius of a circle, even for triangular and polygonal meshes, as confirmed by Evrard et al. [19]. This effect will be further investigated, by considering all the factors that have impact on the convolution.

The question now arises as to how one should choose the appropriate kernel and its support. These choices, and examples of typical convolution kernels, are discussed in the next section.

3.2. Convolution Filters

3.2.1. Kernel Requirements

Williams et al. [12] established five conditions which must be taken into account when choosing a convolution kernel K:

1. The support of the kernel is compact: $|\Omega| < \infty$.
2. The kernel is spherically symmetric: $K(\mathbf{x}) = K(r)$, with r the Euclidean distance from the centre of the kernel.
3. The kernel monotonically decreases with increasing distance r from the centre of the convolution: $K(r_1) > K(r_2)$, for $r_1 < r_2$, and $r_1, r_2 \leq |\Omega|$.
4. The kernel is smooth enough: $K \in C^k$, for $k \geq 3$.
5. The kernel is normalized: $\int_\Omega K(\mathbf{x})\ \mathrm{d}\mathbf{x} = 1$.

As noted later in Section 3.4, it is also necessary for the convolution kernel to be zero at the boundary. This does not follow directly from the kernel requirements, but was required for the calculations of that section. The distance from the convolution centre to the boundary of the support is denoted by δ, which will also be referred to as the support. The choice of this δ is pivotal for the resulting field. If chosen too large, the obtained smoothed value is not representative for its location, and the interface region of the multiphase flow becomes too spread out. This causes problems when interfaces of two or more different bubbles approach each other, since then the mesh element values from one bubble impact the convoluted colour field of another one, which is not physically valid. Therefore, the decision of δ should also be based on the current physical location of the interface. However, if δ is too small, the effect of the convolution may not be significant enough to smooth the colour function. For this reason, the decision of this parameter needs to be investigated.

Based on the conditions mentioned above, Williams et al. [12] formulated the K_6 and the K_8 convolution kernels, both of which satisfy the five mentioned requirements, as well as the added requirement that it needs to be zero on the boundary ($r = \delta$).

3.2.2. Filters

Sixth-order kernel K_6

This kernel is given by

$$K_6(r,\delta) = \begin{cases} A[1-(r/\delta)^2]^3, & r < \delta \\ 0, & r \geq \delta. \end{cases} \tag{3.2}$$

The normalization constant A ensures that condition 5 is attained. With respect to the desired δ, Williams et al. [12] recommend $\delta \geq 4h$, with h the cell size.

Eighth-order kernel K_8

Following the same structure as the sixth-order kernel, the eighth-order kernel is given by

$$K_8(r,\delta) = \begin{cases} A[1-(r/\delta)^2]^4, & r < \delta \\ 0, & r \geq \delta, \end{cases} \tag{3.3}$$

where A again serves as a normalization constant.

Cosine kernel $K_{\cos}$

Peskin [20] introduced a convolution kernel based on a cosine function which he used for studying the flow of blood in the heart to smooth the velocity field and boundary forces. His kernel definition is given by

$$K_{\cos}(r,\delta) = \begin{cases} \frac{1}{2\delta}[1+\cos(\pi r/\delta)], & r < \delta \\ 0, & r \geq \delta. \end{cases} \tag{3.4}$$

For this kernel to comply with the requirements mentioned above, a normalization constant can be introduced in the definition of the kernel.

In order to visualize how the kernels distribute the weights, Figure 3.1 displays the value of each kernel as a function of the distance r from the centre of the filter.

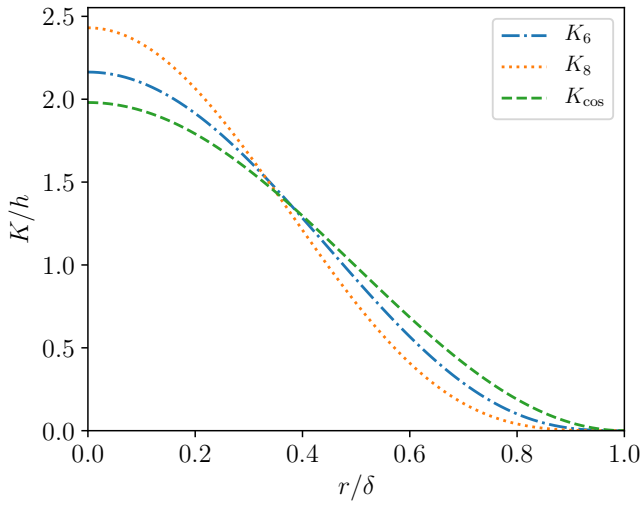


Figure 3.1: Values of the normalized K_6, K_8 and $K_{\cos}$ convolution filters as a function of distance r from the centre of the filter relative to the convolution support δ.

3.2.3. Discretisation of the Filters

The general principle of convolution and some examples of convolution kernels for a continuous domain have been considered. However, the current study is based on the use of discrete convolution, given that the VOF colour function c is discrete-valued, i.e. each cell value represents a region in space. The modification for a discrete domain is explained next.

Discrete Convolution

The principle of convolution for the discrete case is the same as the continuous case: a weighted average of neighbouring cells around the convolution centre determines the value in any cell. The integral in Equation (3.1) becomes a sum, and the kernel K must now also be made discrete. For the two dimensional case, the discrete convolution is therefore given by the next equation (square brackets are used to point out the discrete nature of the domain):

$$\tilde{g}[x,y] = \sum_{\bar{x}=-k_x}^{k_x} \sum_{\bar{y}=-k_y}^{k_y} g[i,j]K[x-\bar{x},y-\bar{y}]. \tag{3.5}$$

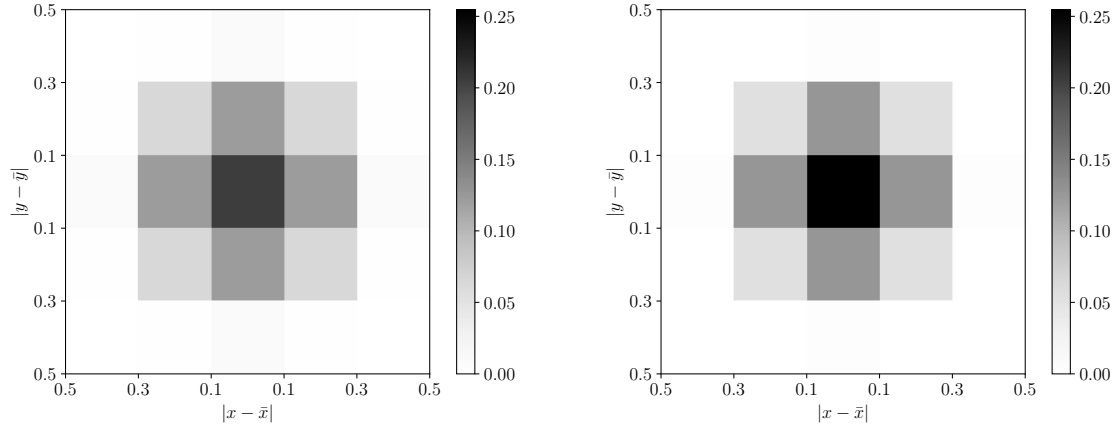
Again, the convolution is limited to a specific region, which is now given by points between $[-k_x,k_x] \times [-k_y,k_y]$, since the kernel is zero outside.

Discrete Kernels

The convolution kernels defined above are given as continuous functions of the Euclidean distance r from the centre of the kernel. For the discrete domain of the current study, the values of the kernel should only be assigned to isolated points in the neighbourhood of the value to be smoothed. To correct for this discreteness, the kernel adopts a grid-like format, with the same cell size as the one given by the domain grid. A value is assigned to each grid cell, obtained by evaluating the continuous kernel at the centre of the cell. In this way, a midpoint rule approximation of the convolution is applied. This approximation provides a second order convergence when integrating over a continuous function. However, in this case the discontinuous VOF field is integrated. It is plausible to assume that such a convergence rate will not be achieved. Improving the accuracy when integrating a discontinuous function by quadrature rules is a challenge.

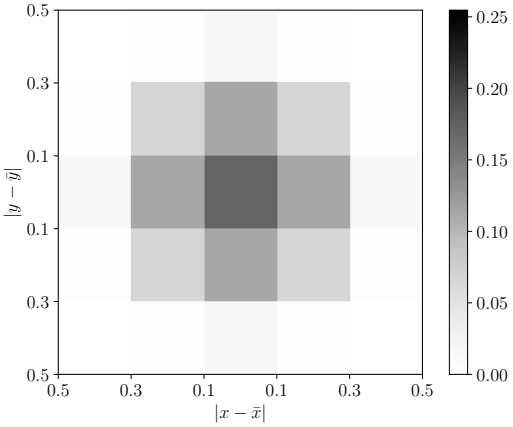
For the discrete kernels to comply with the symmetry requirement, the number of cells of the kernel grid in any dimension should be odd and equal, such that the centre of the kernel is always given by a cell centre rather than the boundary between cells. This ensures that neighbouring cells which are equally far from the cell to be smoothed receive equal weights. For the continuous convolution kernels, a constant A was introduced to ensure normalisation, and could be computed as the inverse of the magnitude of the convolution integral. In the discrete domain, a finite number of weights are assigned, and A becomes the inverse sum of these weights. This guarantees that no cell in the domain attains a VOF value above unity.

To gain some insight on how the convolution filters influence the value to be smoothed, Figure 3.2 portrays the weight distribution of the K_6, K_8 and $K_{\cos}$ discrete kernels in a colour plot. The grid is of unit length in both dimensions, and the support Ω covers a distance $\delta = 0.5$ in the radial direction from the centre ($k_x = k_y = 0.5$). A choice of five cells per dimension was made. For mesh elements of which the centre is located further away from the centre of the kernel than the value of δ, the weight becomes zero. For this reason, the corner elements of the filters for this plot have no influence in the convolution process.



(a) K_6 kernel.

(b) K_8 kernel.



(c) $K_{\cos}$ kernel.

Figure 3.2: Sixth order, eighth order and cosine convolution kernels. The colour bars haven been normalized to span the same interval. For the exact value of the weights in each cell, see Appendix A.

It is evident from this figure, and also from the plot of the weights for the continuous case in Figure 3.1, that the K_6 convolution kernel puts more weight to mesh elements located further away from the centre when compared to the K_8 kernel. The $K_{\cos}$ kernel exploits this effect even more. In this way, the eighth order convolution kernel has a much more centralized effect when smoothing the colour function, such that this function remains localized and closer to the region where there is an interface. From Figure 3.2, it is also evident that the K_8 kernel has a steeper transition region. This effect becomes even more significant when the support length δ decreases, which gives large values of the derivative of the kernel. Therefore, as Williams et al. [12] noted, a large number of points is needed within the support of the kernel to calculate the derivative

of the kernel, which was a step needed in the hybrid method they implemented for the curvature calculation. The curvature calculation in this work is based solely on the difference equations once the colour function has been smoothed, such that this unwanted effect does not have an impact.

There is one more filter worth of consideration, which is discrete by nature. This is the Laplacian filter, and is presented next.

Laplacian filter

The Laplacian filter, implemented first by Lafaurie et al. [10], introduces a smoothed field $\tilde{c}$ based on the original field c defined as

$$\tilde{c} = \frac{\sum_{f=1}^{n} c_f A_f}{\sum_{f=1}^{n} A_f}, \tag{3.6}$$

where f is the index that runs through all the faces surrounding the mesh element, and A_f is the corresponding face area. The values at the faces of the mesh element boundaries are calculated by means of linear interpolation with each of the neighbouring mesh elements. This filter can also be used iteratively, such that after a second implementation of the filter, the second order neighbouring cells are also taken into account. This concept is illustrated in Figure 3.3 for a 2D Cartesian mesh.

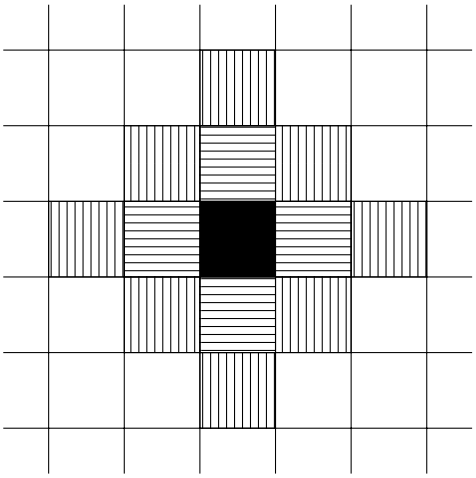
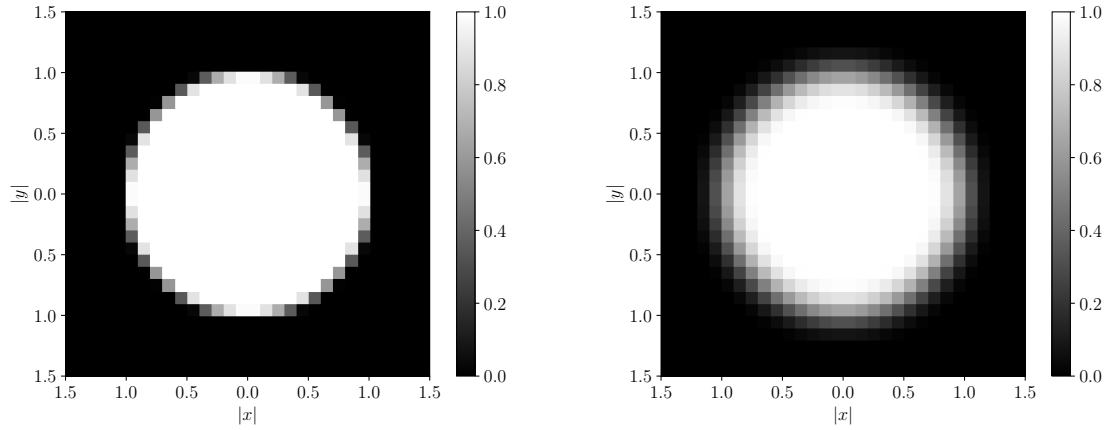


Figure 3.3: Mesh elements considered for the Laplacian filter.

The cell that will be smoothed is the central black one. Let n_{lap} be the number of iterations used. The horizontal pattern fills the first order cells, which are taken into account when $n_{\text{lap}} = 1$. Furthermore, a vertical pattern fills the cells that are then also considered when the filter is applied with $n_{\text{lap}} = 2$. Ubbink [15] suggests two implementations of the Laplacian filter to smooth the colour function, such that all neighbours depicted in Figure 3.3 are used. In this case, the weight of the central cell corresponds to 5/16, first order neighbours receive 1/8 and second order neighbours 3/128.

3.3. Example: Convolution of Circle

To visualise the impact that convolution has on a VOF field, consider the colour plots in Figure 3.4, where convolution is applied to the VOF field of a circle. The left-most image is the unconvoluted field, whereas for the right-most image, a K_6 convolution kernel was applied to smooth the colour function. It is evident what the impact of the filtering is. The interface now seems to be spread over a larger region, and computational cells that formed the interface from the original VOF field now have a smoother transition at both sides of the interface.



(a) (b)

Figure 3.4: Colour plots of (a) the original (unconvoluted) VOF field and (b) the VOF field convoluted with a K_6 kernel. The radius of the circle is 1, and the surrounding fluid was extended 0.5 at each axis. $h = 0.1$ and $\delta = 3h$.

3.4. Estimating the Normal Vector with a Convoluted VOF Field

In this section, an analytical expression for the normal vector estimation given the convoluted VOF field will be derived. After this, it will be shown that this vector correctly points in the radial direction for a locally circular interface and is independent of δ, concluding that the exact interface curvature can be retrieved in the case of a circular interface. For the purposes of the derivation, a continuous domain is considered.

3.4.1. General Case

Consider an interface of random shape as shown in Figure 3.5, where the extent of the convolution is given by the smaller circle. Let A be the region in space that belongs to the central fluid 1, and Ω the part of the domain that is covered by the convolution filter.

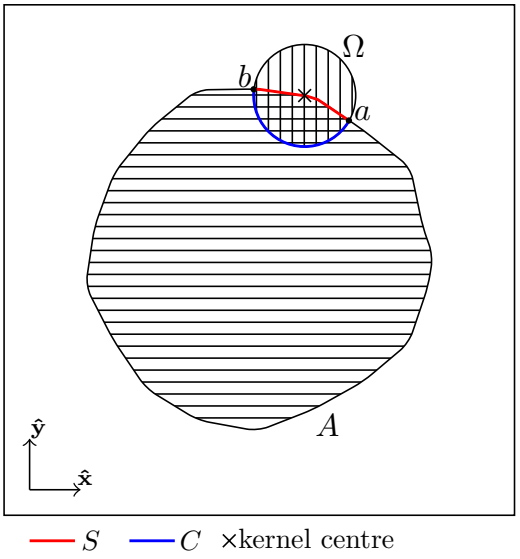


Figure 3.5: Visualisation of the relevant components when estimating the normal vector by using the convoluted colour field $\tilde{c}$.

The exact colour field c can be formulated as

$$c(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in A \setminus \partial A, \\ 1/2, & \mathbf{x} \in \partial A, \\ 0, & \mathbf{x} \notin A, \end{cases} \tag{3.7}$$

∂A being the boundary of A. The interface of the shape is defined as the contour line $c = 1/2$. Let $\hat{\mathbf{n}}$ be the unit vector normal to this contour line. The exact curvature is then obtained from

$$\kappa = \left| \frac{\mathrm{d}\hat{\mathbf{n}}}{\mathrm{d}s} \right|, \tag{3.8}$$

with s the arc length. An estimate of the normal vector needed for the curvature calculation can be obtained from the convoluted colour field $\tilde{c}$, given by the continuous version of Equation 2.9. Using the definition of convolution in Equation 3.1 we get

$$\begin{aligned}
\mathbf{n}_{\text{est}}(\mathbf{x}) &= \nabla_{\mathbf{x}} \tilde{c}(\mathbf{x}) && (3.9)\\
&= \nabla_{\mathbf{x}} \int_{\Omega(\mathbf{x})} c(\bar{\mathbf{x}}) K(\mathbf{x}-\bar{\mathbf{x}}) \, \mathrm{d}\bar{\mathbf{x}} && (3.10)\\
&= \frac{\partial}{\partial x} \left[\int_{\Omega(\mathbf{x})} c(\bar{\mathbf{x}}) K(\mathbf{x}-\bar{\mathbf{x}}) \, \mathrm{d}\bar{\mathbf{x}} \right] \hat{\mathbf{x}} + \frac{\partial}{\partial y} \left[\int_{\Omega(\mathbf{x})} c(\bar{\mathbf{x}}) K(\mathbf{x}-\bar{\mathbf{x}}) \, \mathrm{d}\bar{\mathbf{x}} \right] \hat{\mathbf{y}}, && (3.11)
\end{aligned}$$

where $\nabla_{\mathbf{x}}$ denotes differentiations with respect to $\mathbf{x}$, and where the coordinates are $\mathbf{x} = x\hat{\mathbf{x}} + y\hat{\mathbf{y}}$ and $\bar{\mathbf{x}} = \bar{x}\hat{\mathbf{x}} + \bar{y}\hat{\mathbf{y}}$. The subscript *est*, referring to the estimation of the normal vector, will be dropped to avoid too many subscripts, but it may be clear that it refers to an estimate. Define $H(\mathbf{x},\bar{\mathbf{x}}) = c(\bar{\mathbf{x}})K(\mathbf{x}-\bar{\mathbf{x}})$ and consider the x component of the normal vector, n_x. Using Reynolds Transport Theorem with respect to the variable x, n_x becomes

$$n_x(\mathbf{x}) = \int_{\Omega(\mathbf{x})} \left[\frac{\partial H(\mathbf{x},\bar{\mathbf{x}})}{\partial x} + \nabla_{\bar{\mathbf{x}}} \cdot (H(\mathbf{x},\bar{\mathbf{x}})\mathbf{v}) \right] \mathrm{d}\bar{\mathbf{x}}, \tag{3.12}$$

with

$$\mathbf{v} = \frac{\mathrm{d}\bar{\mathbf{x}}}{\mathrm{d}x}. \tag{3.13}$$

Splitting the integrand and applying the divergence theorem to the second integral gives

$$n_x(\mathbf{x}) = \int_{\Omega(\mathbf{x})} \frac{\partial H(\mathbf{x},\bar{\mathbf{x}})}{\partial x} \, \mathrm{d}\bar{\mathbf{x}} + \int_{\partial\Omega(\mathbf{x})} H(\mathbf{x},\bar{\mathbf{x}})(\mathbf{v}\cdot\hat{\mathbf{n}}) \, \mathrm{d}\bar{\mathbf{x}}, \tag{3.14}$$

with $\hat{\mathbf{n}}$ is the unit vector perpendicular to the boundary of the domain $\partial\Omega(\mathbf{x})$. Remember that the kernel was chosen such that it evaluates to zero at the boundary. Therefore,

$$H(\mathbf{x},\bar{\mathbf{x}}) = 0 \quad \text{for} \quad \bar{\mathbf{x}} \in \partial\Omega(\mathbf{x}), \tag{3.15}$$

and the second term in Equation (3.14) vanishes. We are left with

$$n_x(\mathbf{x}) = \int_{\Omega(\mathbf{x})} \frac{\partial}{\partial x} [c(\bar{\mathbf{x}})K(\mathbf{x}-\bar{\mathbf{x}})] \, \mathrm{d}\bar{\mathbf{x}}, \tag{3.16}$$

where the definition of H was substituted back again. The performed calculation is equally valid for the y component of the normal vector, such that

$$\begin{aligned}
\mathbf{n}(\mathbf{x}) &= \left[\int_{\Omega(\mathbf{x})} \frac{\partial}{\partial x} [c(\bar{\mathbf{x}})K(\mathbf{x}-\bar{\mathbf{x}})] \, \mathrm{d}\bar{\mathbf{x}} \right] \hat{\mathbf{x}} + \left[\int_{\Omega(\mathbf{x})} \frac{\partial}{\partial y} [c(\bar{\mathbf{x}})K(\mathbf{x}-\bar{\mathbf{x}})] \, \mathrm{d}\bar{\mathbf{x}} \right] \hat{\mathbf{y}} && (3.17)\\
&= \int_{\Omega(\mathbf{x})} \left[\frac{\partial}{\partial x} [c(\bar{\mathbf{x}})K(\mathbf{x}-\bar{\mathbf{x}})]\hat{\mathbf{x}} + \frac{\partial}{\partial y} [c(\bar{\mathbf{x}})K(\mathbf{x}-\bar{\mathbf{x}})]\hat{\mathbf{y}} \right] \mathrm{d}\bar{\mathbf{x}} && (3.18)\\
&= \int_{\Omega(\mathbf{x})} \nabla_{\mathbf{x}} [c(\bar{\mathbf{x}})K(\mathbf{x}-\bar{\mathbf{x}})] \, \mathrm{d}\bar{\mathbf{x}} && (3.19)\\
&= \int_{\Omega(\mathbf{x})} c(\bar{\mathbf{x}}) \nabla_{\mathbf{x}} K(\mathbf{x}-\bar{\mathbf{x}}) \, \mathrm{d}\bar{\mathbf{x}}. && (3.20)
\end{aligned}$$

Recall that $c(\bar{\mathbf{x}}) = 0$ for $\bar{\mathbf{x}} \notin A$. Therefore, the integrand only has a non-zero value for $\bar{\mathbf{x}} \in A \cap \Omega$, and we get

$$\mathbf{n}(\mathbf{x}) = \int_{A\cap\Omega} \nabla_{\mathbf{x}} K(\mathbf{x}-\bar{\mathbf{x}}) \, \mathrm{d}\bar{\mathbf{x}}. \tag{3.21}$$

Using that $\nabla_{\mathbf{x}} K(\mathbf{x}-\bar{\mathbf{x}}) = -\nabla_{\bar{\mathbf{x}}} K(\mathbf{x}-\bar{\mathbf{x}})$,

$$\mathbf{n}(\mathbf{x}) = -\int_{A\cap\Omega} \nabla_{\bar{\mathbf{x}}} K(\mathbf{x}-\bar{\mathbf{x}})\ \mathrm{d}\bar{\mathbf{x}}. \tag{3.22}$$

Let $\mathbf{a} \in \mathbb{R}^2 \setminus \mathbf{0}$ be a constant vector. Using the identity $\nabla\cdot(K\mathbf{a}) = \mathbf{a}\cdot(\nabla K) + K(\nabla\cdot\mathbf{a})$, and the fact that $\mathbf{a}$ is constant, it holds that $\nabla\cdot(K\mathbf{a}) = \mathbf{a}\cdot(\nabla K)$, since the divergence of $\mathbf{a}$ vanishes. Using this identity, we can rewrite our expression for $\mathbf{n}(\mathbf{x})$ in Equation 3.22, introducing a constant vector $\mathbf{a}$:

$$\mathbf{a}\cdot\mathbf{n}(\mathbf{x}) = -\mathbf{a}\cdot\int_{A\cap\Omega} \nabla_{\bar{\mathbf{x}}}[K(\mathbf{x}-\bar{\mathbf{x}})]\ \mathrm{d}\bar{\mathbf{x}} \tag{3.23}$$

$$= -\int_{A\cap\Omega} \mathbf{a}\cdot\nabla_{\bar{\mathbf{x}}}[K(\mathbf{x}-\bar{\mathbf{x}})]\ \mathrm{d}\bar{\mathbf{x}} \tag{3.24}$$

$$= -\int_{A\cap\Omega} \nabla_{\bar{\mathbf{x}}}\cdot[K(\mathbf{x}-\bar{\mathbf{x}})\mathbf{a}]\ \mathrm{d}\bar{\mathbf{x}}. \tag{3.25}$$

The divergence theorem now gives us that

$$\mathbf{a}\cdot\mathbf{n}(\mathbf{x}) = -\int_{\partial(A\cap\Omega)} K(\mathbf{x}-\bar{\mathbf{x}}(l))\mathbf{a}\cdot\hat{\mathbf{n}}_\partial\ \mathrm{d}l, \tag{3.26}$$

where $\hat{\mathbf{n}}_\partial$ is the unit normal vector to $\partial(A\cap\Omega)$, the boundary of $A\cap\Omega$, and $\mathrm{d}l$ is the arc length. This boundary is the union of the curves S and C in Figure 3.5. It is again used that the kernel is zero at the boundary of the convolution support. Therefore, the integrand in Equation 3.26 vanishes for the blue curve C in Figure 3.5, and we obtain

$$\mathbf{a}\cdot\mathbf{n}(\mathbf{x}) = -\mathbf{a}\cdot\int_S K(\mathbf{x}-\bar{\mathbf{x}}(l))\hat{\mathbf{n}}_\partial\ \mathrm{d}l, \tag{3.27}$$

where the vector $\mathbf{a}$ was taken out of the integral because it is constant. Rearranging gives

$$\mathbf{a}\cdot\mathbf{n}(\mathbf{x}) + \mathbf{a}\cdot\int_S K(\mathbf{x}-\bar{\mathbf{x}}(l))\hat{\mathbf{n}}_\partial\ \mathrm{d}l = 0 \tag{3.28}$$

$$\mathbf{a}\cdot\left(\mathbf{n}(\mathbf{x}) + \int_S K(\mathbf{x}-\bar{\mathbf{x}}(l))\hat{\mathbf{n}}_\partial\ \mathrm{d}l\right) = 0. \tag{3.29}$$

For this to hold for any non-zero constant $\mathbf{a} \in \mathbb{R}^2$, we must have that

$$\mathbf{n}(\mathbf{x}) = -\int_S K(\mathbf{x}-\bar{\mathbf{x}}(l))\hat{\mathbf{n}}_\partial\ \mathrm{d}l. \tag{3.30}$$

Equation 3.30 provides an analytical expression for the estimation of the normal vector based on the convoluted colour field. The exact outcome of this calculation is dependent on the local interface topology. When an interface is locally circular, this expression can be simplified. This will be done next.

3.4.2. Interface with a Locally Constant Curvature

Consider an interval S of an interface where the local curvature can be considered constant. Figure 3.6 illustrates this scenario, and presents the relevant parameters. Given the symmetry of this interval, it is useful to use polar coordinates.

Let the convolution support be such that $\delta < R$. The points $\mathbf{x}$ on the interval S can be parametrised as

$$\mathbf{x}(\theta) = R\sin(\theta)\hat{\mathbf{x}} + R\cos(\theta)\hat{\mathbf{y}}. \tag{3.31}$$

The exact unit local normal vector is computed directly from this parametrisation, and results in

$$\hat{\mathbf{n}}_{\mathrm{loc}}(\theta) = \sin(\theta)\hat{\mathbf{x}} + \cos(\theta)\hat{\mathbf{y}} = \hat{\boldsymbol{\epsilon}}(\theta). \tag{3.32}$$

The exact curvature is given by $\kappa_{\mathrm{exact}} = 1/R$. The unit tangential vector to the interface is $\hat{\boldsymbol{\omega}}(\theta)$. The unit normal vector $\hat{\mathbf{n}}_\partial$ in Equation 3.30 can be expressed in terms of the unit vectors $\hat{\boldsymbol{\epsilon}}$ and $\hat{\boldsymbol{\omega}}$ as

$$\hat{\mathbf{n}}_\partial(\phi,\theta) = \cos(\phi)\hat{\boldsymbol{\epsilon}}(\theta) + \sin(\phi)\hat{\boldsymbol{\omega}}(\theta), \tag{3.33}$$

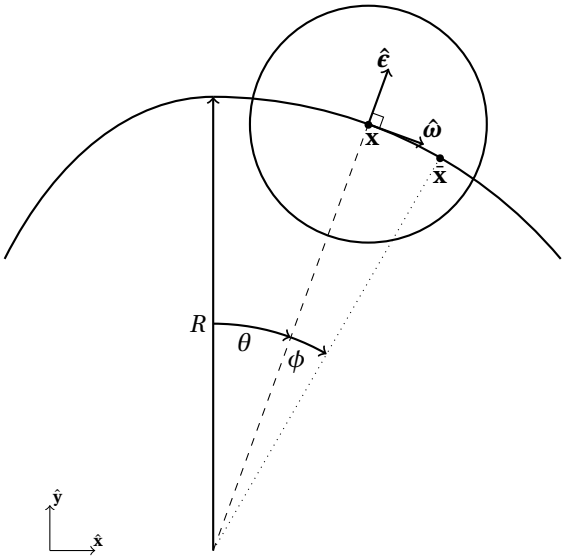


Figure 3.6: Visualisation of the relevant variables when estimating the interface normal of a locally circular interface.

where θ specifies the angle at which the centre of the convolution is located, and ϕ the angle with respect to θ. The argument of K in Equation 3.30 only depends on the distance between the centre of the convolution $\mathbf{x}(\theta)$ and points at the interface $\bar{\mathbf{x}}(\theta,\phi)$. The argument as a function of ϕ can therefore be expressed as

$$|\mathbf{x}-\bar{\mathbf{x}}| = |2R\sin(\phi/2)|. \tag{3.34}$$

Combining Equations (3.33) and (3.34) into (3.30) gives

$$\mathbf{n}(\theta) = -\int_{\phi_-}^{\phi_+} K(|2R\sin(\phi/2)|)\left[\cos(\phi)\hat{\boldsymbol{\epsilon}}(\theta)+\sin(\phi)\hat{\boldsymbol{\omega}}(\theta)\right] R\,\mathrm{d}\phi \tag{3.35}$$

$$= -\left[\int_{\phi_-}^{\phi_+} K(|2R\sin(\phi/2)|)\cos(\phi)\; R\,\mathrm{d}\phi\right]\hat{\boldsymbol{\epsilon}}(\theta) - \left[\int_{\phi_-}^{\phi_+} K(|2R\sin(\phi/2)|)\sin(\phi)\; R\,\mathrm{d}\phi\right]\hat{\boldsymbol{\omega}}(\theta), \tag{3.36}$$

where $\mathrm{d}l = R\,\mathrm{d}\phi$ is the arc length, and ϕ_+ and $\phi_- = -\phi_+$ are the angles with respect to θ at which the convolution boundary intersects the interface, for which the condition $\delta < R$ is needed. The integrand of the $\hat{\boldsymbol{\omega}}$ component is odd around $\phi = 0$, and therefore integrates to zero. The integrand of the $\hat{\boldsymbol{\epsilon}}$ component is even around $\phi = 0$ and non-zero, since the convolution kernel is positive in its interior. Thus, it integrates to a non-zero constant, such that

$$\mathbf{n}(\delta,\theta) = \gamma(\delta)\hat{\boldsymbol{\epsilon}}(\theta). \tag{3.37}$$

The constant γ is dependent on δ because it determines the integration limits ϕ_- and ϕ_+. With this, it has been proven that for a locally circular interface the normal vector calculated with the convoluted colour field correctly points in the direction perpendicular to the interface. For a completely circular interface, this means that the direction of the estimated normal vector at each point on the interface is the same as the direction of the exact perpendicular vector. For the curvature from Equation 3.8 it means that

$$\kappa = \left|\frac{\mathrm{d}\hat{\mathbf{n}}(\delta,\theta)}{\mathrm{d}s}\right| \tag{3.38}$$

$$= \left|\frac{1}{\gamma(\delta)}\frac{\mathrm{d}\mathbf{n}(\delta,\theta)}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}s}\right| \tag{3.39}$$

$$= \frac{1}{R}, \tag{3.40}$$

since $s = \theta R$. This matches the exact curvature κ_{exact}, and is independent of the convolution support δ.

Concluding remarks
It was shown that, for a circular interface, the estimation of the curvature from the convoluted colour field is the same as the exact value, independent of $\delta < R$. For shapes where the interface cannot be approximated as circular, the estimated normal vector will acquire a component in the $\hat{\boldsymbol{\omega}}$ direction which will be dependent on δ and on the local interface shape. For that curvature, this means that the variation of the unit normal vector along the interface differs from the exact variation, failing to obtain the exact curvature.

4

Impact of Convolution on Curvature Calculations

In order to test the performance of the different convolution filters and the influence of the kernel parameters, a program that calculates a discrete convolution integral was implemented in Python. The performance was checked by calculating the error in the curvature after the VOF field was smoothed by means of convolution. The implementation is described in Section 4.1. Section 4.2 presents the results of a circular shape, for which the interface curvature is constant. Next, Section 4.3 shows the same calculations, but now for an ellipse, for which the interface curvature is not constant. In this chapter, time is not considered, and all the calculations are two-dimensional.

4.1. Implementation

The implemented program in Python calculates the VOF colour field for a particular interface shape, carries out the convolution and calculates the curvature. To this end, the convolution operation is performed with the different kernels described in Section 3.2. Convolution is performed by element-wise multiplication of two matrices, the VOF colour field and shifted versions of the convolution kernel, which is also instantiated in matrix-form. Problems with calculating convolution at the boundary of the domain were avoided by extending the surrounding fluid to such an extent that it did not affect the convolution of the cells for which convolution actually changes the colour field (i.e. the walls of the domain can be thought of as being located at infinity).

The program accepts random shape dimensions, as well as grid resolution. For ease of implementation, the computational grid is aligned with the chosen shape, having an even amount of cells fill the shape both in the x and y direction. This allows for the calculation of the colour field only in the first quadrant, after which the obtained field is mirrored in both the x and y direction. After convolution, the curvature is calculated using Equation (2.20). Central differencing schemes are used for the first and second derivative, as explained in Section 2.4.1. The values of the curvature are then compared with the exact curvature, using the measures introduced in Section 2.4.2. The cells that are considered in this error measure are the ones located at a band around the interface which is $3h$ wide, and centred at the interface. This represents the region where the surface tension force from the Continuum Surface Force model typically acts (refer to Section 2.3).

4.2. Constant Curvature: Circle

A circular interface is a good starting point for the current study, because the VOF implementation is rather straightforward and it allows for a proper visualisation of the underlying effect of convolution. This shape serves as a test case of constant curvature along the interface, which will then be compared with a case of varying curvature in Section 4.3.

4.2.1. Initialization

The VOF colour field c for a circle was calculated exactly based on the overlap of the circle with each of the grid cells, given the geometry of the setting and the grid being Cartesian. A circle can be parametrised as

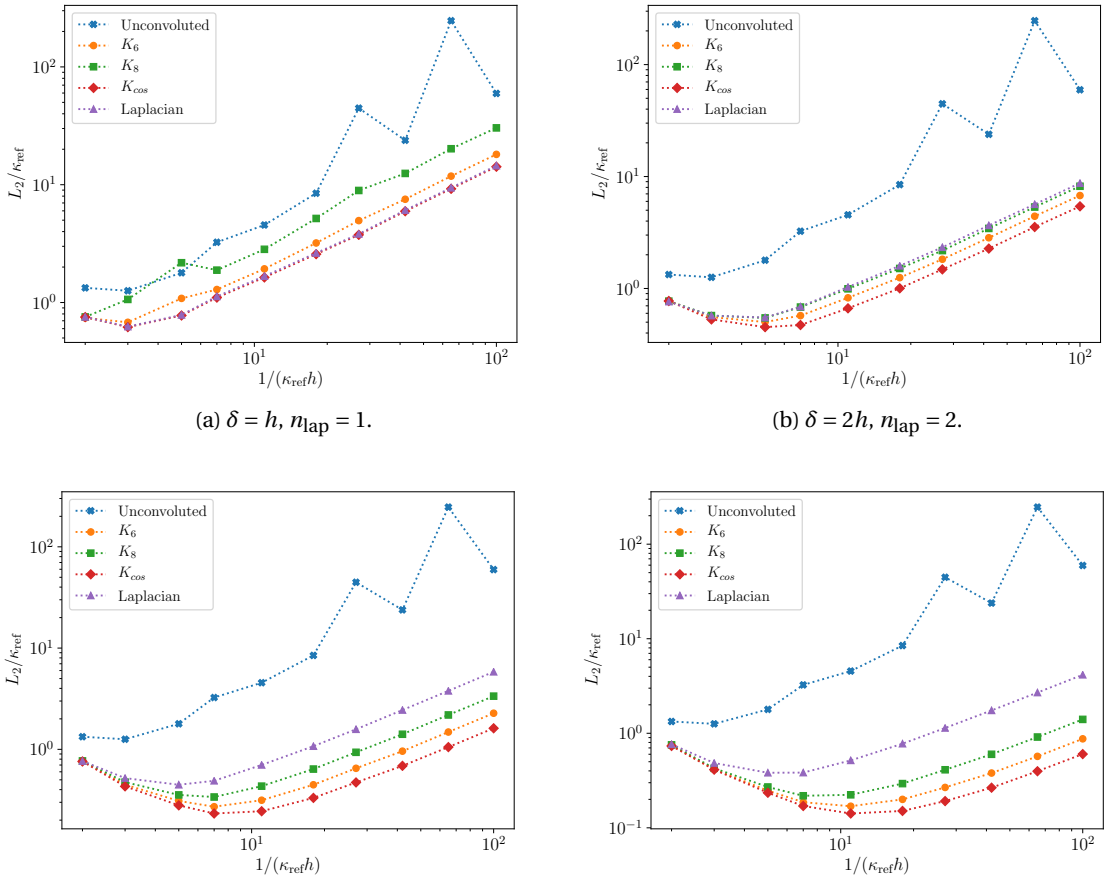
$$x(\theta) = R\cos(\theta) \tag{4.1}$$
$$y(\theta) = R\sin(\theta), \tag{4.2}$$

where R is the radius of the circle and $\theta \in [0, 2\pi)$. With this parametrisation, the exact curvature of a circle is found by substituting the corresponding expressions in Equation (2.16), which results in

$$\kappa_{\text{exact,c}} = \frac{1}{R}. \tag{4.3}$$

4.2.2. Curvature Error

The error in the curvature was calculated as a function of the mesh size, for a circle with $R = 1$. Ten logarithmically-spaced mesh sizes were chosen between 2 and 100. The convolution support δ was varied between h and $4h$, as well as the Laplacian filter iteration number n_{lap} from 1 to 4, in order to observe the impact on the error given the cell size. The results for the mean curvature error are shown in Figure 4.1, and the maximum error for the same settings in Figure 4.2.



(a) $\delta = h$, $n_{\text{lap}} = 1$.

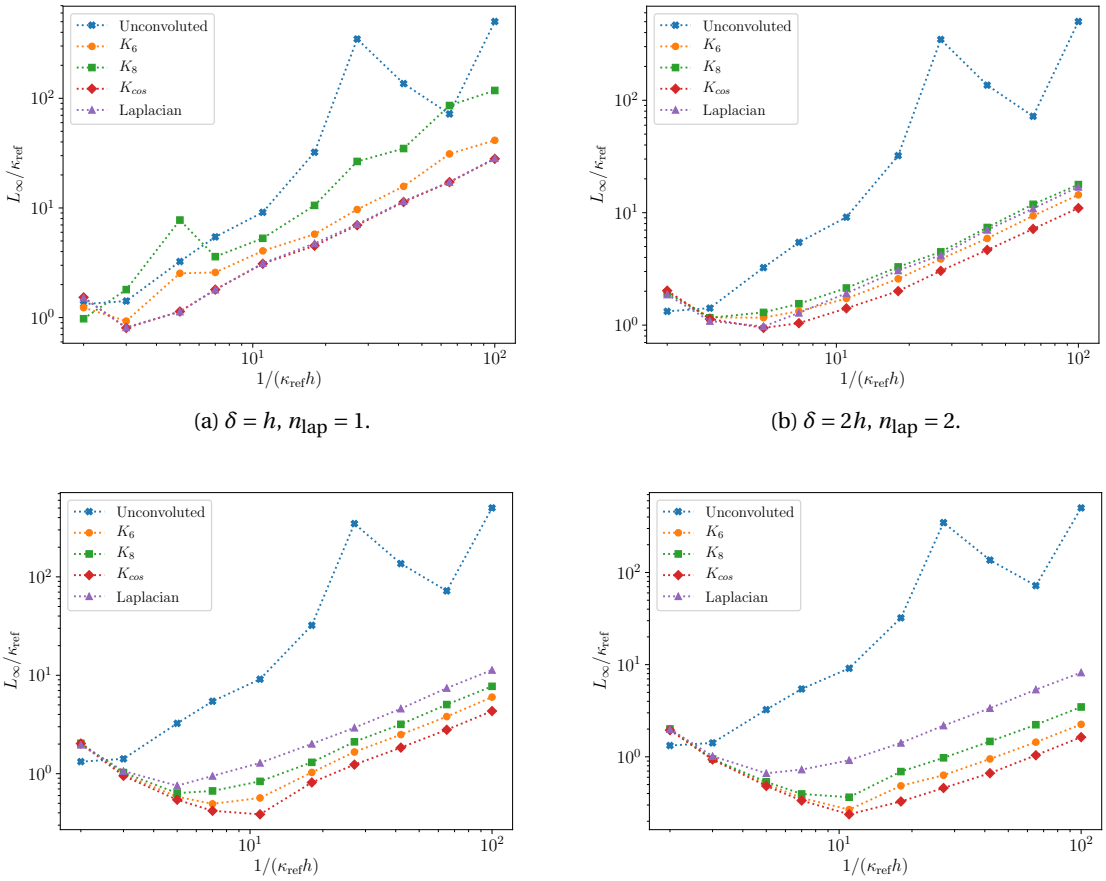
(b) $\delta = 2h$, $n_{\text{lap}} = 2$.

(c) $\delta = 3h$, $n_{\text{lap}} = 3$.

(d) $\delta = 4h$, $n_{\text{lap}} = 4$.

Figure 4.1: Curvature errors L_2 for different convolution supports δ or Laplace filter iterations n_{lap}. The curvature calculated with the unconvoluted VOF is shown for reference in each sub-figure. Following the same approach as Evrard et al. [19], each axis is normalized with the reference curvature, which for the case of a circle is simply the constant curvature $\kappa_{\text{exact}} = 1/R$.

The x-axis quantifies the number of cells per radius of the interface, so the error in the curvature decreases up to a point as the mesh becomes finer, whereafter it monotonically starts to increase (this will be called the



(a) $\delta = h$, $n_{\text{lap}} = 1$.

(b) $\delta = 2h$, $n_{\text{lap}} = 2$.

(c) $\delta = 3h$, $n_{\text{lap}} = 3$.

(d) $\delta = 4h$, $n_{\text{lap}} = 4$.

Figure 4.2: Similar to Figure 4.1, but now the L_∞ curvature errors are shown on the vertical axis.

turning point). This holds for all kernel supports. This behaviour is the same that is observed by other studies [19]. It is worth noting that all the implemented kernels have the same divergent trend. This is not surprising, since the same method of smoothing is being used in each case, so the trend should be similar. The number of cells per radius at the turning point increases as the support of the convolution kernel is extended. For a support of h, at approximately 3 cells per radius the curvature error starts to increase, and for a support of $2h$, $3h$ and $4h$ it becomes around 4, 7 and 11 cells per radius, respectively. This is attributed to the fact that a greater kernel support implies a smoother transition between the values of the colour function, which implies that the differentiation that is applied when calculating the curvature is less prone to errors. Note how the curvature error of the unconvoluted field is very susceptible to errors. As the mesh is refined, the curvature error follows an upward trend, with evident spikes that demonstrate the unstable nature of the curvature when the colour field is not smoothed.

The difference between the kernels given a fixed convolution support is only how the weights of the filter are distributed, and this accounts for a shift in the error of the curvature for each kernel. Besides Figure 4.1a, where the convolution support was set to h, the mean curvature errors are distributed in the same way for each kernel. In each case, the Laplacian filter has the highest error, followed by the K_8, K_6 and K_{cos} kernels. Recall from Figure 3.1 that the K_{cos} was the filter that extended its weight the most to neighbours further away, followed by K_6 and K_8, this last one being the most 'localised' one. The consequence of giving more weight to far neighbours is comparable to extending the kernel support. This effect was analysed in the previous paragraph. Therefore, the K_{cos} kernel provides a smoother change of the colour field at the interface, which is reflected in having a smaller curvature error. It can be concluded that the difference in the curvature error for each convolution kernel is trivial, since the nature of the curvature error follows the same trend in each case.

It seems that scaling the convolution support linearly with the cell dimension h cannot ensure convergence of the curvature error as the mesh is refined. However, it is possible for the error to converge when the mesh is refined by scaling the convolution support δ in a different way with the mesh dimension h. More specifically, for $\delta \propto h^\alpha$, with $\alpha < 2/3$. The effect of scaling δ in a non linear way with h has already been investigated [8], but may also be proven formally. Consider the second term on the right-hand-side of Equation 2.19 for the estimation of the curvature based on a smoothed colour function $\tilde{c}$:

$$\psi = -\frac{1}{|\nabla \tilde{c}|}\nabla^2 \tilde{c}. \tag{4.4}$$

It can be proven that ψ is not consistent for $\alpha \geq 2/3$.

Proposition. *Given a convoluted colour field $\tilde{c}$, the estimation of ψ is not consistent for a support of $\delta \propto h^\alpha$, $\alpha \geq 2/3$.*

Proof. Let $\delta \propto h^\alpha$ be the convolution support used to smooth c. The width of the transition region between the values of 0 and 1 of $\tilde{c}$ is proportional to $2h^\alpha$. Consider $\tilde{c}$ in terms of $\tau = r/h^\alpha$, such that the colour function becomes independent of h. By repeatedly applying the chain rule for $\tilde{c}(\tau)$, the computation of the n-th derivative of this function can be written as

$$\frac{\mathrm{d}^n \tilde{c}}{\mathrm{d}r^n} = \frac{\partial^n \tilde{c}}{\partial \tau^n} \cdot \frac{1}{h^{n\alpha}}, \tag{4.5}$$

with $\partial^n \tilde{c}/\partial \tau^n$ independent of h and bounded. Consider first the $\nabla^2 \tilde{c}$ term in ψ. Given the discreteness of the domain, finite difference methods are used. Using a central difference approach with step size h, the approximation of the second derivative of the Laplacian can be written as

$$\nabla^2 \tilde{c} = \frac{\mathrm{d}^2 \tilde{c}}{\mathrm{d}r^2} + \frac{h^2}{12}\tilde{c}^{(4)}(\xi), \tag{4.6}$$

for $\xi \in (r-h, r+h)$. The truncation error of this approximation is

$$\frac{h^2}{12}\tilde{c}^{(4)}(\xi) = \frac{h^2}{12}\frac{\partial^4 \tilde{c}}{\partial \tau^4}(\xi) \cdot \frac{1}{h^{4\alpha}} = \mathcal{O}(h^{2-4\alpha}), \tag{4.7}$$

since $\partial^4 \tilde{c}/\partial \tau^4$ is a constant. Consider now the scaling factor $1/|\nabla \tilde{c}|$ in ψ. The gradient is found by differentiating once with respect to r:

$$\frac{\mathrm{d}\tilde{c}}{\mathrm{d}r} = \frac{\partial \tilde{c}}{\partial \tau} \cdot \frac{1}{h^\alpha}. \tag{4.8}$$

Therefore,

$$|\nabla \tilde{c}| = \mathcal{O}(h^{-\alpha}) \implies \frac{1}{|\nabla \tilde{c}|} = \mathcal{O}(h^\alpha). \tag{4.9}$$

Combining the two factors results in

$$\psi = -\frac{1}{|\nabla \tilde{c}|}\nabla^2 \tilde{c} = \mathcal{O}(h^\alpha)\mathcal{O}(h^{2-4\alpha}) = \mathcal{O}(h^{2-3\alpha}). \tag{4.10}$$

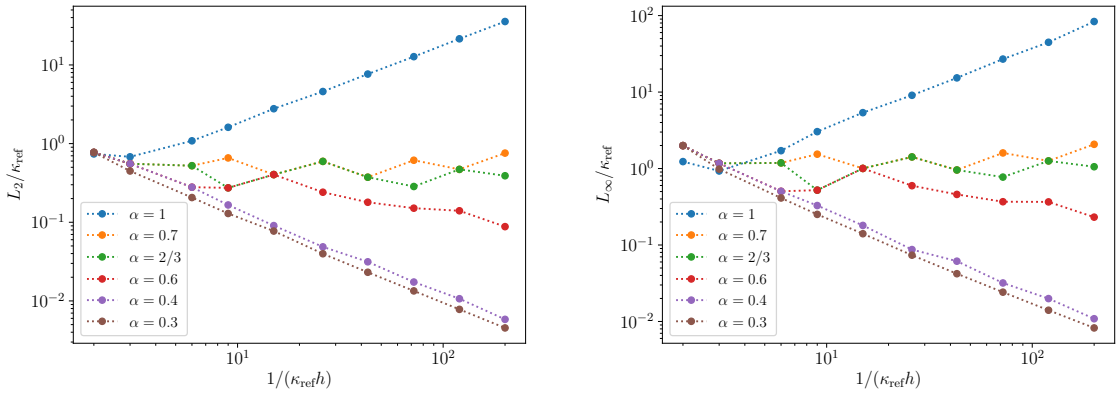
The total truncation error of the curvature estimation is therefore $\mathcal{O}(h^{2-3\alpha})$. To guarantee consistency in the estimation, it is necessary that

$$2 - 3\alpha > 0 \implies \alpha < \frac{2}{3}. \tag{4.11}$$

□

To visualize this finding, Figure 4.3 shows the mean and maximum curvature error L_2 for ten logarithmically spaced mesh sizes. The parameter that distinguishes the different lines is the scaling of the convolution support. A support of $\delta = h^\alpha$ was used for different values of α, for the K_6 convolution kernel.

Though Figure 4.3 certainly does not prove convergence, it is instructive to observe the behaviour of the curvature error as the scaling of the convolution support with respect to the cell size is varied. For $\alpha = 2/3$, the convergence seems to be $\mathcal{O}(1)$, which follows from the proven proposition. If $\alpha < 2/3$, one can expect a second order convergence, given the truncation error of a central difference approximation for a second derivative. However, Figure 4.3 seems to shows that, for $\alpha < 2/3$, no improvement from approximately a first



(a) L_2 error.

(b) L_∞ error.

Figure 4.3: Normalized curvature errors as a function of mesh size for convolution supports given by $\delta = h^\alpha$. The K_6 kernel was used.

order convergence can be obtained. As mentioned in Section 3.2.3, the midpoint rule integration over the discontinuous VOF colour field is not assumed to have the expected $\mathcal{O}(h^2)$ truncation error. Therefore, this may limit the accuracy that is obtained from the finite differencing methods for the curvature calculation.

Scaling with $\alpha < 1$ makes physical sense. Suppose a support of $\delta = h$ is chosen for convolution. When the mesh is coarse, the colour function values around the interface are weighted with cells that are relatively far away from the interface, since the mesh dimension is relatively large. When the mesh is refined, the colour function values around the interface will every time be averaged with cells that are closer to the interface, so the physical span of the convolution kernel is decreased. This means that the transition of the colour function is still very steep, such that errors in the curvature calculation are likely to occur. However, if the convolution support is scaled with $\alpha < 1$, the discrete span of the kernel increases, resulting in a smoother colour field (it is extended over a larger physical span compared to scaling with $\alpha = 1$). The degree to which one should decrease α is determined by the error term in the finite difference method used to calculate the curvature, which appears to be $\alpha = 2/3$. It is still the case that the support of the convolution in physical space decreases when refining the mesh, but the decrease with $\alpha = 2/3$ is more gradual.

4.3. Non-Constant Curvature: Ellipse

The convergence of the curvature error was investigated in Section 4.2 for an interface of constant curvature. The objective now is to examine the effect of convolution on interfaces where the curvature is not constant. The simplest shape for this scenario is an ellipse. This will help to determine what are the decisive factors when smoothing a colour field belonging to a shape where the curvature varies along the interface.

4.3.1. Initialization

The VOF colour field c of the ellipse was calculated with Monte Carlo integration. 10^5 random points were generated per cell, and the ratio of points inside the ellipse with respect to the total number determined the VOF fraction. An ellipse can be parametrized as

$$x(\theta) = a\cos(\theta), \tag{4.12}$$

$$y(\theta) = b\sin(\theta), \tag{4.13}$$

where again $\theta \in [0, 2\pi)$, and a and b are fixed. a is called the semi-major axis, and b the semi-minor axis when $a > b$, which is the case for the generated ellipses. Figure 4.4 shows the relevant parameters of the ellipse.

Given the parametrisation, the exact curvature of an ellipse at the interface is now

$$\kappa_{\text{exact,e}}(\theta) = \frac{ab}{\left[(a\sin(\theta))^2 + (b\cos(\theta))^2\right]^{3/2}}. \tag{4.14}$$

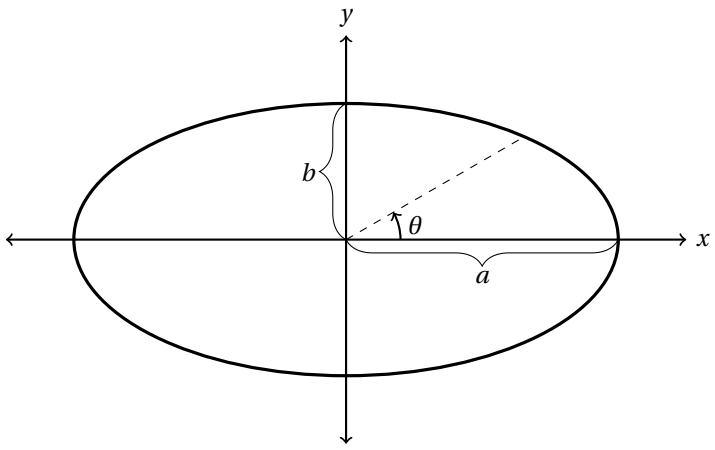


Figure 4.4: Sketch of an ellipse and its relevant parameters.

4.3.2. Curvature Error

Mean curvature error

Differing now from the approach used when considering the circle, the effect of increasing the support of the kernel for fixed cell dimensions in the curvature error will be investigated. To this end, different ellipses are generated, for which $a/b = 2$, 3, and 4. The higher the ratio a/b, the higher the curvature variation along the interface, so the effect of smoothing an interface with different curvature variations can be analysed. Without loss of generality, the K_6 kernel will be used, since it was seen that all kernels behave similarly. Eight logarithmically-spaced convolution supports were taken between h and $16h$. The results for the mean curvature error as a function of δ are shown in Figure 4.5

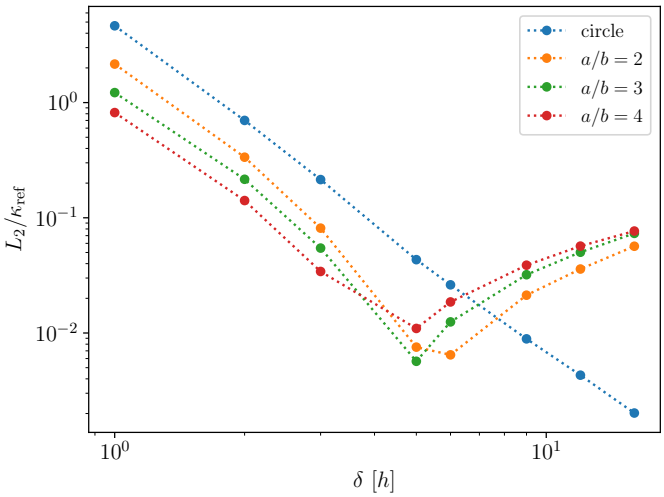


Figure 4.5: L_2 curvature error as a function of δ for ellipses with 20 cells per semi-minor axis and a varying number of cells per semi-major axis. The error of a circle with 20 cells per radius is given as a reference. The error of the ellipses were normalized with $\kappa_{\text{ref}} = a/b^2$, which corresponds to the maximum curvature, and of the circle again with $\kappa_{\text{ref}} = 1/R$, following the approach of Evrard et al. [19].

Note that Figure 4.5 portrays a different parameter on the x-axis than Figure 4.3, since it is used to analyse the impact of large convolution supports. Figure 4.5 evidences how the calculation of the curvature of a circle is improved as the kernel support increases. However, this is not the case for an ellipse. For this shape, there is a point after which increasing the kernel support makes no sense to improve the curvature calculation. Note how the curvature error of the ellipse for which the curvature varies the greatest ($a/b = 4$) is the largest. Moreover, its error appears to diverge for a lower δ than the ellipse with $a/b = 2$. This is also the case for the ellipse with $a/b = 3$. The kernel support after which the mean curvature error of the ellipse starts to diverge

changes as a function of the amount of cells that form the ellipse. In this sense, the size of the cells in the domain of a discrete ellipse tells us nothing without the actual dimensions of the ellipse; only mesh sizes relative to the ellipse dimensions provide information about what kernel support to use.

The following paragraph analyses what exactly is the determining factor of an interface that leads to discrepancies of the curvature when the colour field is smoothed by means of convolution.

Curvature error as a function of curve parameter θ

To investigate what aspect of an interface leads to high curvature errors, the difference between the exact and calculated curvature will be considered along the interface, as a function of θ. In this way, the value of θ for which the curvature error is the highest indicates the part of the interface that is more susceptible for errors in the curvature when convolution is used. In Figure 4.6, an ellipse with 20 cells per semi-minor axis is convoluted with a kernel of $\delta = 8h$. From Figure 4.5, it can be seen that the error at this point is already diverging.

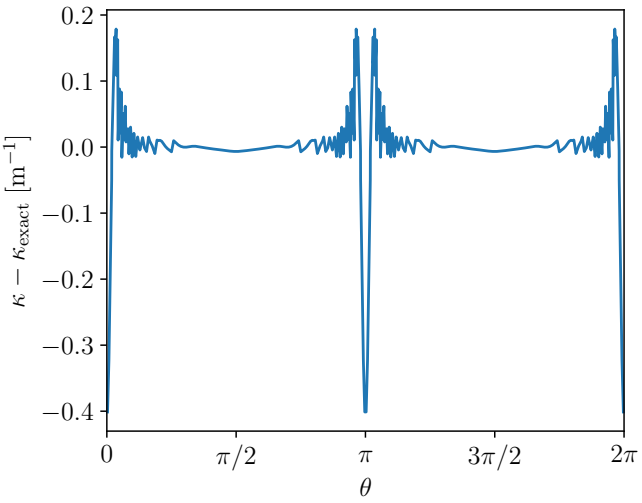
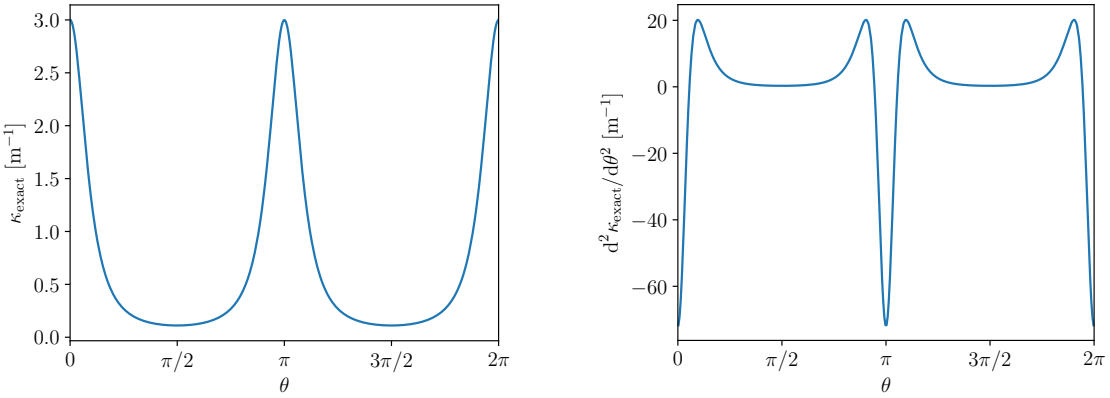


Figure 4.6: Difference between exact and estimated curvature of an ellipse as a function of θ for $\delta = 8h$. $b = 1\,\text{m}$, $a = 3\,\text{m}$.

Figure 4.6 clearly demonstrates that the major contributions to the curvature error come from the points where $\theta = 0$ and $\theta = \pi$. To further investigate what aspect of the curvature variation determines these high errors, the values of the curvature and its second derivative as a function of θ have been plotted in Figure 4.7 (the first derivative is not relevant for the analysis). It stands out how closely related the curvature error in Figure 4.6 and the second derivative of the exact curvature in Figure 4.7b are. This resemblance leads to the conclusion that it is this aspect of the interface that will determine the error in the curvature calculation of the interface after convolution with a large kernel support.

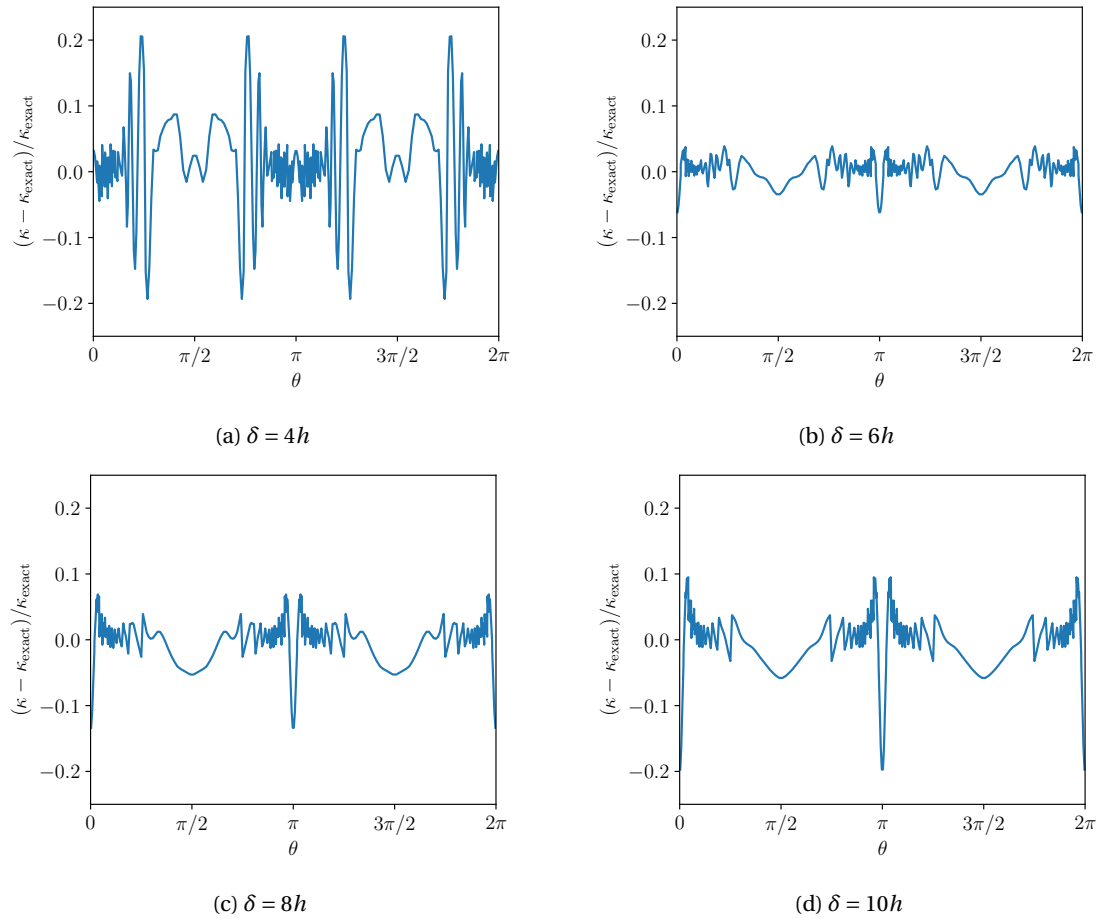
Figure 4.6 might be deceiving, because it plots the error in an absolute scale rather than a relative one. At $\theta = 0$ and $\theta = \pi$, the curvature is the highest, so maybe the relative error is unchanged as a function of the interface parameter. To analyse the evolution of the relative error as a function of the support θ when the support of the kernel is increased, Figure 4.8 plots these parameters for the same ellipse as before, changing the convolution support in each sub-figure.

Figure 4.8 evidences that the error at $\theta = 0$ and π does not always dominate. For a kernel support of $\delta = 4h$, the relative error peaks are at $\theta = \pi/4$, $3\pi/4$, $5\pi/4$ and $7\pi/4$. As the support is increased to $\delta = 6h$, the relative error is more or less evenly distributed as a function of θ. From that point on, increasing the kernel support changes the error magnitude minimally at all the values of θ but the errors at 0 and π, where the error is significantly increased. Therefore, large convolution supports fail to work for an ellipse because the error is amplified around these values of θ, leading to inaccurate estimations of the curvature.



(a) κ (b) $d^2\kappa/d\theta^2$

Figure 4.7: The exact curvature κ_{exact} and its first derivative of an ellipse as a function of θ.



(a) $\delta = 4h$ (b) $\delta = 6h$

(c) $\delta = 8h$ (d) $\delta = 10h$

Figure 4.8: The relative curvature error of an ellipse as a function of θ for different convolution supports. The y axes represent the same interval for all subplots.

Choosing the ideal support for an ellipse

In order to investigate what size the kernel should have given an ellipse and its extension in the discrete cell space, several ellipses with $a/b = 2,\ 3$ and 4 are generated, varying the number of cells per semi-minor axis b, which will be called n. For each of these domains, the colour fields are convolved with kernels varying in size from $\delta = h$ to $\delta = 18h$, in steps of h. For each mesh resolution, the normalized mean curvature error L_2 was plotted against the convolution support. Some of these plots are shown in Appendix B. It was observed that in each case, the difference between the ellipses with $a/b = 2,\ 3$ and 4 was minimal, so the influence of this parameter can be neglected. To find the ideal kernel support for an ellipse, the convolution support for which L_2 was minimal for the ellipse with $a/b = 3$ was plotted against n. The results are shown in Figure 4.9.

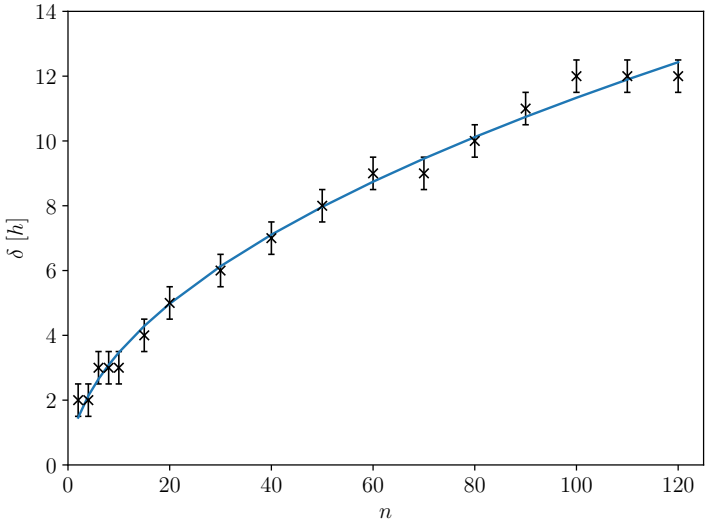


Figure 4.9: Convolution support δ for which the L_2 error in the curvature was minimal against the number of cells per semi-minor axis b, n. The best fit for the square root relation shown is $\delta = (1.15 \pm 0.03)\sqrt{n} + (-0.17 \pm 0.20)$.

The best fit for the results is a square root relation given by

$$\delta = (1.15 \pm 0.03)\sqrt{n} + (-0.17 \pm 0.20). \tag{4.15}$$

The notable step-like character of the data points at nearby values of n is because the convolution support δ can only change in unit steps, given the discrete nature of the domain. For this same reason, when choosing the appropriate convolution support given the mesh size of an ellipse, the obtained value of δ in the fit must be rounded to the nearest integer. The error bars cover an interval of δ, to account for variations in the a/b value of an ellipse. This fit can be used as a general guideline for the convolution support that must be chosen for an ellipse in a given mesh. In this way, the error in the curvature by using convolution with the current method is minimized. However, this might not be the best that can be achieved with convolution. As was analysed in the previous paragraph, the error in the curvature depends on the local interface shape. The current method utilises a global convolution support, i.e. it is the same for all cells. A different approach would be to choose a local support based on the local curvature of the interface. For this, an estimate of the curvature can be made using a global support, after which one could specialize the support per individual cell, and thereby minimize the error in the curvature. This method is computationally more costly than the current method, and was not implemented for the current research.

4.4. Concluding Remarks

In this section, two types of interfaces were considered, one with a constant interface curvature and one where the curvature varied along the interface. In the first case, it was observed that increasing the convolution support had a positive effect in decreasing the error in the curvature estimations. This is because as δ is increased, the VOF colour field around the interface becomes smoother, and the magnitude of the higher order terms that determine the truncation error in the curvature decrease. This result is accompanied by the fact that, for interfaces where the curvature is constant, the convoluted VOF field correctly preserves the

shape of the interface, independent of the chosen δ for convolution. This was analytically shown in Section 3.4.2.

The case for an interface with varying curvature is different. It is still the case that a smoother VOF field is needed for better curvature approximations. Convolution is still helpful to achieve this, but because of the interface shape, an unwanted effect occurs. As the curvature varies along the interface, convoluting the VOF field has an impact on the actual curvature that one intends to estimate. If the support is chosen too large, contour lines around the interface of the smoothed VOF field will actually change its original shape. This effect was analysed in Section 4.3.2 for an ellipse. It was found that this modification from the original shape is most significant for high values of the second derivative of the curvature with respect to the arc length.

In this sense, increasing the convolution support for convoluted VOF fields for interfaces with varying curvature has both a positive and negative impact on the curvature estimation:

1. *Positive:* the VOF field at the interface becomes smoother, decreasing the magnitude of higher order derivatives that determine the error in the estimation.

2. *Negative:* the original shape of the interface is increasingly modified.

The combined effect of these two factors determine the final accuracy of the curvature error. In Section 4.3.2, a general measure for choosing the convolution support is provided for an ellipse, which is based on the support at which the combined effect is the most favourable. The exact convolution support for a general interface above which effect 2 becomes more significant than effect 1 cannot be retrieved from this analysis, but it was evident that this point is determined by the magnitude of the second derivative of the curvature with respect to the arc length. It was here where the dominance of effect 2 over effect 1 was the largest. For future work, it would be interesting to analyse in depth the point at which effect 2 becomes more significant for a general interface shape.

5

Impact of Convolution on Two-Phase Flow Simulations

Having tested the influence of convolution merely in the curvature computation in Chapter 4, this chapter will consider its use in OpenFOAM simulations, which includes other phenomena like spurious currents. Section 5.1 describes the implemented convolution algorithm in OpenFOAM. Section 5.2 compares OpenFOAM curvature values with values from the Python code used in Chapter 4. Then, the results of performing simulations with convolution will be verified using two benchmark cases: a static bubble in Section 5.3 and a rising bubble in Section 5.4.

5.1. Algorithm

Figure 5.1 summarizes the computation of convolution in OpenFOAM. The code can be found in Appendix C.

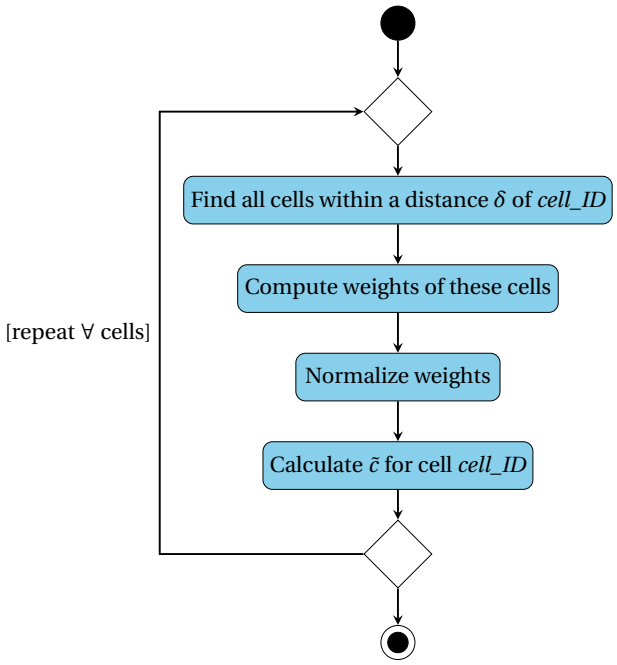


Figure 5.1: UML activity diagram of the convolution algorithm in OpenFOAM.

Comments

- *Find all cells within a distance δ of cell_ID*: this is done by means of an iterative process. Using the OpenFOAM method *cellCells()[cell_ID]*, the cell indices of the cells neighbouring *cell_ID* are obtained.

This method is used again to find neighbouring cells to the neighbours of *cell_ID*. This process is continued until the cell centres of any new neighbours and the centre of *cell_ID* are separated by a distance larger than the convolution support δ.

- *Compute weights of these cells*: the weight of each neighbouring cell is computed with the equation for the kernel, where the distance r_i is the distance between the cell centre of neighbouring cell i and *cell_ID*. In the developed program, an additional weighing with the cell volume was included. This only has an effect when the mesh size varies along the mesh. In that case, a higher weight is given to cells that occupy a larger region in space, which avoids biasing the convolution to one specific region where the mesh might have been refined. Therefore, the weight of each cell is expressed as

$$w(r_i) = w_k(r_i)V_i, \tag{5.1}$$

with w_k the unnormalized kernel weight and V the volume.

- *Normalize weights*: the normalization constant A of the convolution is given by

$$A = \sum_{i=1}^{N} w(r_i), \tag{5.2}$$

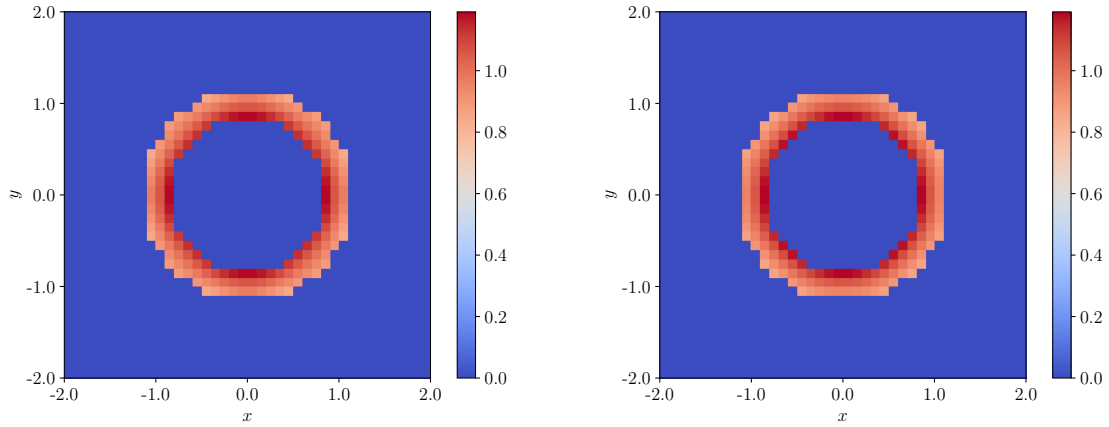
where the index i runs along all the neighbour cells. The weights calculated in the previous step are divided by A.

- *Calculate $\tilde{c}$ for cell cell_ID*: the smoothed colour field value for cell *cell_ID* is given by

$$\tilde{c}[cell\_ID] = \frac{1}{A}\sum_{i=1}^{N} c_i\, w(r_i). \tag{5.3}$$

5.2. Comparison of Curvature Calculation Between Python and OpenFOAM

The results obtained from the developed code in Python to calculate the curvature used throughout Chapter 4 will be compared to the values obtained with OpenFOAM for a given two-phase flow initial setting. The VOF colour field of a circular bubble with a radius R of 1 m and 10 cells per radius is generated with the code in Python. After this, this field is exported to OpenFOAM and used as initial condition. The colour field is smoothed with the K_6 kernel given in Equation 3.2 and a support of $\delta = 4h$. The curvature is then calculated independently with each program. Figure 5.2 depicts the calculated curvature field for both programs on a $3h$ wide band around the interface.



(a) OpenFOAM. (b) Python.

Figure 5.2: Curvature of domain cells within a 3 cell wide band centred around the interface when calculated with different programs. The convolution support is $\delta = 4h$.

The fields shown in Figure 5.2 visually show a great resemblance between the curvatures calculated with both programs. To quantify the difference for several convolution supports and mesh dimensions, Table 5.1 show the mean difference in curvature for cells in a $3h$ wide band around the interface.

Table 5.1: Mean difference in curvature values between the calculations with OpenFOAM and Python. The deviations are normalized with the reference curvature of $\kappa_{ref} = 1\,\mathrm{m}^{-1}$. The considered cells are those within a $3h$ wide band around the interface.

Cells per radius	**Convolution support** $[h]$			
	4	**6**	**8**	**10**
10	0.34	0.10	0.02	0.01
20	0.18	0.05	0.02	0.01

Large deviations are observed for coarse meshes, and the difference seems to vanish as the support increases. The observed deviation can be explained by the fact that OpenFOAM directly calculates the divergence of the normal vector based on the VOF colour field (Equation (2.17)), whereas the code that was implemented in Python uses the modification suggested by Brackbill et al., Equation (2.18). When testing these two ways of calculating, they found that the rms error was smaller when using Equation (2.18) as opposed to Equation (2.17). This difference can also be attributed to the fact that OpenFOAM uses the curvature values at cell faces, as opposed to cell centres. To achieve this, it interpolates the values from cell centres. At the end, the curvature values at cell centres are obtained by reconstructing the values from the faces. Therefore, the final curvature of each cell is not the direct curvature that is computed in Python, but an average over the curvature values in a set of nearby cells. This effect is the largest for small convolution supports, because the band around the interface where the curvature estimates are better is narrower than for larger supports.

5.3. Benchmark Case 1: Static Bubble

The impact of convolution on the complete two-phase flow model will first be considered in a case with a circular static bubble in another fluid. This is achieved by setting the gravitational acceleration g to zero, such that no external forces can make the bubble rise or sink. This case is widely used because any movement of the bubble in the simulation can be attributed to numerical discrepancies in the method of solving the equations (spurious currents), rather than the laws of physics acting on the system. Therefore, a criterion to evaluate the validity of the implemented method is observing whether the bubble gains velocity over time, and how much. The results will be compared with the benchmark simulation of Hoang et al. [21]. Although their research focuses on microchannels, the impact of smoothing with a Laplacian filter (Section 3.2.3) for a static bubble in OpenFOAM is considered, and therefore serves as a reference point for the current study.

5.3.1. Case Description

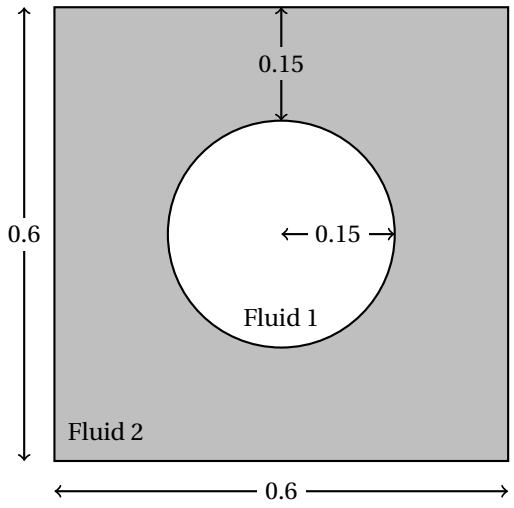


Figure 5.3: Initial setting for benchmark case 1. All the distances are in millimetres.

Table 5.2 presents the properties of the fluids.

For the circular bubble, $R = 150 \cdot 10^{-6}\,\mathrm{m}$, and the radius covers 25 computational cells, such that $h = 6 \cdot 10^{-6}\,\mathrm{m}$. Furthermore, the total domain spans an area of $4R \times 4R$, and the bubble is initially centred. The

Table 5.2: Fluid properties for benchmark case 1. ρ is the density, μ the dynamic viscosity, ν the kinematic viscosity and σ the surface tension.

Property	**Fluid 1**	**Fluid 2**
ρ $[10^3\,\mathrm{kg\,m^{-3}}]$	0.5	1
μ $[10^{-3}\,\mathrm{Pa\,s}]$	2.5	1
ν $[10^{-6}\,\mathrm{m\,s^{-1}}]$	5	1
σ $[10^{-3}\,\mathrm{N\,m^{-1}}]$	23.6	

approach Hoang et al. [21] used to initialize the VOF colour field was to start with a rectangular bubble, and let the bubble relax to its final shape by running the simulation for some time. In the current study, the *exact* VOF colour field is used as initial condition. This field is calculated externally with the existing Python code and then used as initial condition.

As in the benchmark case, OpenFOAM's *PISO* scheme is used. First order implicit Euler schemes are used for the transient terms, and OpenFOAM's *interfaceCompression* is used for the discretisation of the VOF colour field, with the *MULES* solver [13]. The rest of the spatial discretisations are done with the second order *van Leer* scheme. The maximum Courant number was set to 0.3, which sets an upper bound for the time steps in the discrete integration methods. A fixed zero contact angle was chosen as boundary condition of the colour field the walls, and a uniform value of 1 at the top and bottom. No slip conditions were applied at the walls and bottom: a uniform value of zero for the velocity and zero gradient for the pressure. Finally, the top of the domain had a uniform value of zero for the pressure and zero gradient for velocity.

5.3.2. Results

As in the benchmark case, the maximum velocity magnitude present in the domain is tracked over time. This velocity is scaled with what is called the capillary velocity, and the time is scaled with the capillary time:

$$u_{\mathrm{cap}} = \frac{\sigma}{\mu_1},\ t_{\mathrm{cap}} = \frac{\mu_1 R}{\sigma}. \tag{5.4}$$

These are used to normalize the corresponding axes to non dimensional quantities. For this case, $u_{\mathrm{cap}} = 23.6\,\mathrm{m\,s^{-1}}$ and $t_{\mathrm{cap}} = 6.36\,\mu\mathrm{s}$. Figure 5.4 shows the results for a convolution support of $\delta = h$, and $2h$.

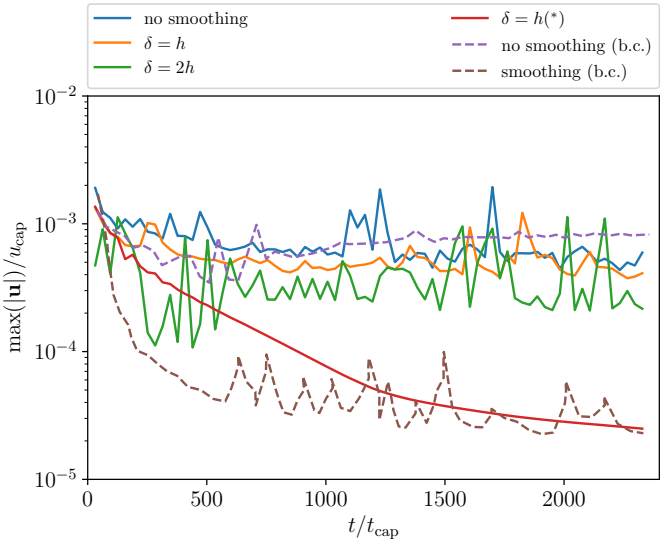


Figure 5.4: Plot of the maximum spurious velocity as a function of time, normalized with the capillary velocity and capillary time, respectively. The results from Hoang et al. [21] are given as a reference, and are marked as (b.c.) for benchmark case. (*)Case with adjusted solver settings.

The obtained maximum spurious velocity without smoothing of the current study are very similar to those

from the benchmark case. This means that the results can be compared meaningfully, since the starting point of the spurious currents is similar. Qualitatively, a support of $2h$ delivers the best results when trying to reduce spurious currents. When the support was extended beyond this support, the maximum spurious velocities started increasing again, and were left out of the plot for visualisation reasons. A simulation was also performed with adjusted solver settings, restricting the maximum time step to a maximum of $\Delta t = 1 \cdot 10^{-5}$ to ensure stability of the simulation. This reduced the spurious currents by more than an order of magnitude when compared to the case with the same convolution support but no explicit time constraint.

In order to quantify the impact of convolution, a similar approach to the benchmark case is adopted. The following measure is defined:

$$\epsilon_{\mathrm{sv},\lambda} = \frac{\left(\int_t \max(|\mathbf{u}|)\, dt\right)_{\delta=\lambda h}}{\left(\int_t \max(|\mathbf{u}|)\, dt\right)_{\delta=0}}, \tag{5.5}$$

which serves as an average value of the spurious velocity displayed in Figure 5.4. Table 5.3 presents the magnitudes of this measure for convolution supports of $\delta = \lambda h$, for $\lambda = 1$, 2, 3 and 4, along with the same measure obtained by Hoang et. al for a λ number of iterations of their Laplacian smoother, $\epsilon_{\mathrm{bc},\lambda}$.

Table 5.3: Normalized spurious velocity average until $t/t_{\mathrm{cap}} = 2350$ for different convolution supports

λ	0	1	2	3	4
$\epsilon_{\mathrm{sv},\lambda}$	1	0.75	0.54	0.75	1.91
$\epsilon_{\mathrm{bc},\lambda}$	1	0.14	0.078	0.072	0.07

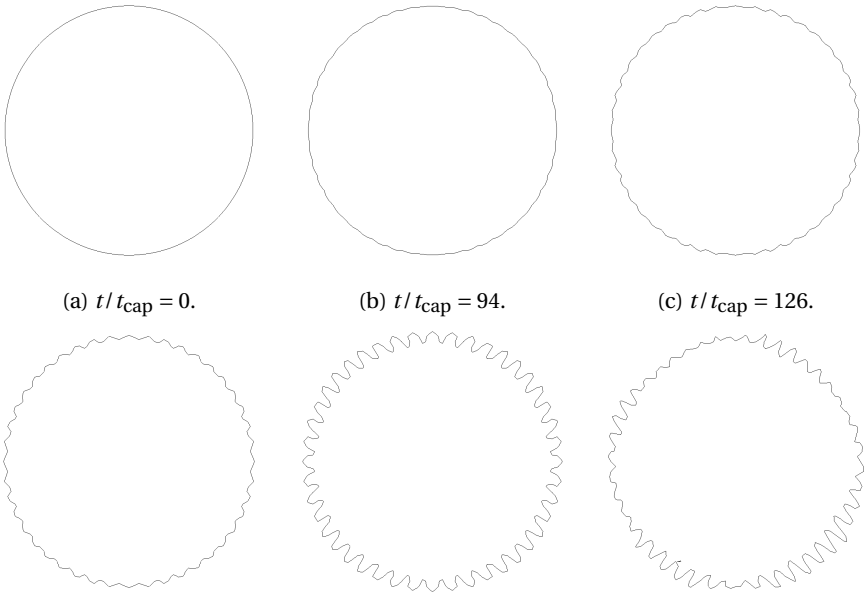
The benchmark case smoother provides an average reduction in the spurious velocities of a factor 13 for a smoothing with two iterations, after which the improvement in spurious currents was minimal. When convolution is used, the minimum in the average spurious velocity over the displayed time is achieved with a convolution support of $\delta = 2h$, after which the spurious currents start to increase again. The spurious velocities were decreased by approximately a factor 1.9 in this case, significantly less than what was achieved with the Laplacian smoother.

A remarkable difference in the results is that the Laplacian smoother keeps reducing the maximum spurious velocities as the number of iterations is increased. However, this is not the case with convolution. As the support is increased after $\delta = 2h$, the maximum spurious velocities start rising again, resulting in poorer results. This is very surprising, since the analysis in Section 4.2 provided the insight that the curvature estimations for the case of a circle became more accurate as the convolution support was increased. This is clearly not the case here. By visualising the VOF field of the simulation, it became evident why the spurious velocities were so significant with high convolution supports. Figure 5.5 depicts contour plots of the VOF colour field at several times during the first half of the simulation.

While smoothing with a Laplacian filter correctly maintains the circular shape that a bubble in the current scenario should have, convolution appears to generate a wave on the interface. Already after short times of the simulation, a clear distortion is visible. The amplitude of this distortion for a convolution support of $\delta = 4h$ grows until approximately $t/t_{\mathrm{cap}} = 346$. After this, the colour field is so distorted that it starts to oscillate in random directions, resulting in asymmetric contour lines, as observed for $t/t_{\mathrm{cap}} = 1006$. It makes no sense to examine the effect at larger times, because the evolution of the simulation from this point on is quite unpredictable and nonsensical.

Considering this phenomenon physically, surface tension would ensure that these waves, where the curvature is relatively high, damp out. However, the curvature is being calculated with the smoothed VOF colour field, where these high curvature areas have been smoothed out as a consequence of convolution. So surface tension is greatly underestimated, and these waves remain on the interface. Figure 5.6 depicts how the high curvatures vanish when convolution is applied.

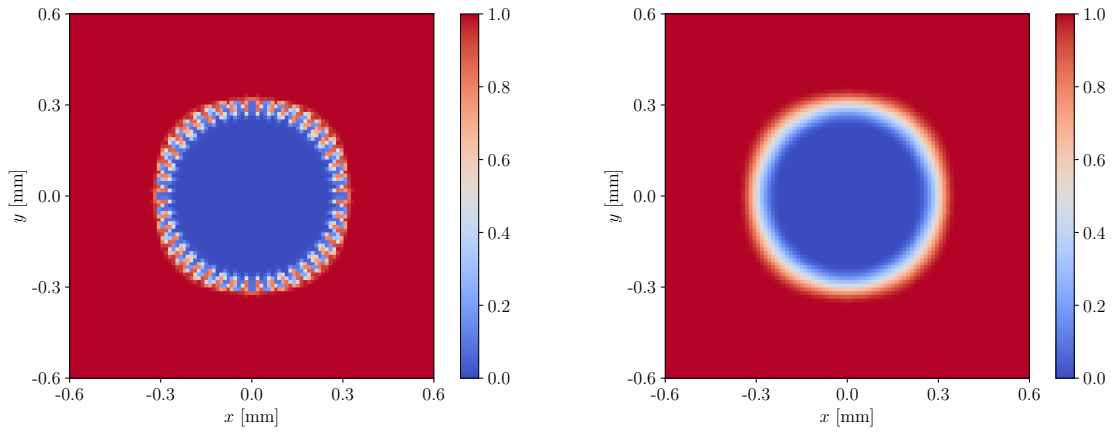
It was observed that the amplitude of these waves is proportional to the convolution support, and that the wavelength increases as the support is increased. At $\delta = h$, they are not even visible and therefore do not affect the VOF colour field considerably at all. As the support increases, the distortion on the field becomes more significant. Despite providing some improvement in the maximum spurious velocity of a static bubble, there is an underlying problem with the convolution algorithm in OpenFOAM. This phenomenon will be analysed in more detail in Section 6.



(a) $t/t_{\text{cap}} = 0$. (b) $t/t_{\text{cap}} = 94$. (c) $t/t_{\text{cap}} = 126$.

(d) $t/t_{\text{cap}} = 157$. (e) $t/t_{\text{cap}} = 346$. (f) $t/t_{\text{cap}} = 1006$.

Figure 5.5: Contour plots of the VOF colour field at $c = 0.5$ for different times using convolution with $\delta = 4h$.



(a) Unsmoothed VOF colour field. (b) VOF colour field smoothed with $\delta = 4h$.

Figure 5.6: Colour plots of the VOF colour field and its smoothed counterpart.

5.4. Benchmark Case 2: Rising Bubble

Having considered the static case as first benchmark case, the next step is to evaluate the use of convolution for a rising bubble. The gravitational force is now present as an external force. Therefore, the simulated bubble will have a physical rising velocity. The considered benchmark case is the first case described by Hysing et al. [22]. This case refers to a rising bubble with small density and viscosity ratios. They compared the results from simulations of three different research groups that use finite element methods, two of which use the Level-Set Method and the third uses Arbitrary Lagrangian-Eulerian. The benchmark quantities that will be considered are the mean rising velocity and circularity. The results for the three different groups considered in the reference study were very well in agreement. Although they did not study a case where the VOF method is used, it serves as a numeric reference for the mentioned quantities, given that three different, independent codes obtained virtually identical results.

5.4.1. Case Description

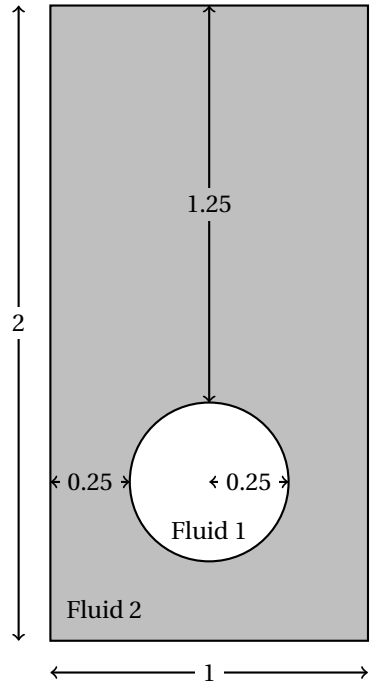


Figure 5.7: Initial setting for benchmark case 2. All the distances are in metres.

Table 5.4 presents the properties of the fluids and the simulation.

Table 5.4: Simulation and fluid properties for benchmark case 2. ρ is the density, μ the dynamic viscosity, ν the kinematic viscosity, σ the surface tension and g the gravitational acceleration.

Property	**Fluid 1**	**Fluid 2**
ρ $[10^3\,\mathrm{kg\,m^{-3}}]$	0.1	1
μ [Pa s]	1	10
ν $[10^3\,\mathrm{m\,s^{-1}}]$	0.1	0.1
σ $[10^{-3}\,\mathrm{N\,m^{-1}}]$	24.5	
g $[\mathrm{m\,s^{-2}}]$	0.98	
Eo	9	
M	$3.6\cdot10^{-3}$	

A circular bubble is initialized with the exact VOF colour field, provided by the program implemented in

Python. This bubble has $R = 0.25\,\text{m}$, and the domain covers a region of $2R \times 4R$, as shown in Figure 5.7. The benchmark case considered several different cell dimensions, but observed little difference for a cell size finer than $h = 1/80\,\text{m}$, which is why this cell dimension is chosen for the current study. This corresponds to 20 cells per radius for the initial condition.

As in the static bubble case, the *PISO* algorithm is used for pressure-velocity coupling. As for the boundary conditions, a zero gradient is used for all the faces for the colour function. No slip condition is applied to the velocity for the top and bottom walls, and slip for the lateral walls. Finally, a uniform zero dynamic pressure is applied to all the faces as boundary condition.

5.4.2. Results

The comparison with the benchmark case is made with time traces of the mean rising velocity and circularity of the bubble.

Circularity

The circularity is defined as

$$\phi = \frac{P_\text{a}}{P_\text{b}}, \tag{5.6}$$

where P_a is the perimeter of an area equivalent circle and P_b is the actual perimeter of the bubble. These values are extracted after the simulation with the *contour* filter available in *paraview*. With this tool, a contour line along a value of the colour field of 0.5 is chosen, which approximates the location of the interface. The length of this contour line then provides the value of the perimeter of the bubble. The area A occupied by the bubble can be obtained by integrating the colour field. With this value, the perimeter of an area equivalent circle is obtained with

$$P_\text{a} = 2\sqrt{A\pi}. \tag{5.7}$$

Figure 5.8 shows the results of simulations performed without smoothing and with convolution supports of $\delta = h$, $3h$ and $5h$, and close-ups around pertinent moments in time.

All of the performed simulations predicted a circularity that is lower than the reference case. The circularity of the bubble approaches the one from the benchmark case as the support of the convolution kernel is increased. For no smoothing, the final circularity predicted by OpenFOAM deviates around 1% from the benchmark case, whereas a convolution with $\delta = 5h$ reduces this to around 0.4%. Convolving with a single cell of support does not seem to improve the results. This is very different from the results obtained for a spurious bubble, where it was observed that instabilities in the simulation actually increased after a convolution support of $\delta = 2h$. For the case of a rising bubble, there is no visible wave present on the interface that distorts the VOF colour field. To see this, Figure 5.9 depicts the evolution of the interface by plotting contour plots at different times.

The difference in shape for the simulation without smoothing and with a convolution support of $\delta = 4h$ is minimal but noticeable, and is in closer agreement with the considered benchmark case. However, if the support is increased too much, the resulting shapes of the bubble become unphysical. This starts at a convolution support of $\delta = 5h$. Figure 5.10 presents the final rising shape which arises when a convolution support of $\delta = 8h$ is used. Similar results where observed by Denner and van Wachem [11] when large convolution stencils were used.

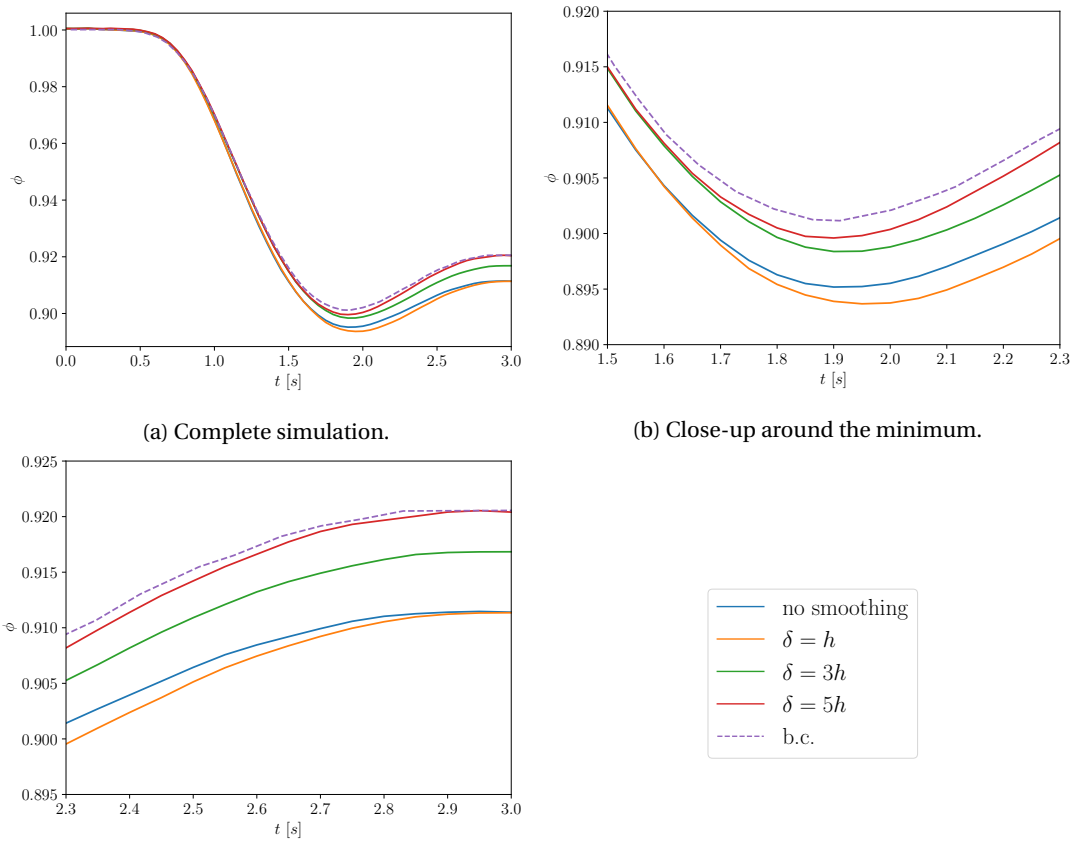
In Section 4.3, it was analysed why there was a limit above which convolution does not improve curvature calculations for an interface with varying curvature, which can explain the obtained results. By approximating the obtained final shape to an ellipse, there are approximately 16 cells per semi-minor axis. Based on Equation 4.15, the ideal convolution support for this scenario would be $\delta = 4h$. This gives a reasonable approximation of the actual ideal support, which is $5h$, considering that the actual shape is not precisely an ellipse.

Rising Velocity

The rising velocity is given by

$$u_\text{r} = \frac{\int_{\Omega_1} |\mathbf{u}| \, \mathrm{d}\mathbf{x}}{\int_{\Omega_1} \mathrm{d}\mathbf{x}}, \tag{5.8}$$

where Ω_1 is the region occupied by fluid 1. To calculate this value, a third-party library extension called *swak4foam* is used. Figure 5.11 plots the rising velocity as a function of time for the same cases considered in Figure 5.8.



(a) Complete simulation.

(b) Close-up around the minimum.

(c) Close-up around final value.

Figure 5.8: Circularity as a function of time of the considered rising bubble and close-ups around relevant moments in the shape development. The results from Hysing et al. [22] are given as reference, indicated by b.c.

These results are indeed what one would expect after having observed the circularity plots in Figure 5.8. Again, the rising velocities approach the results of the benchmark case as the convolution support is increased, for supports greater than $\delta = h$. Since the circularity of the simulated bubbles was consistently increased as the support was incremented up to $\delta = 5h$, this indicates that the cross-sectional area of the rising bubbles decreased as a function of convolution support. Discarding the possibility of bubble shapes that are elongated in the rising direction (this is not physically possible for the current case), a circle would be the shape with least cross-sectional are, which corresponds to a circularity of 1. Therefore, since the circularity increased as the convolution support did when compared to the case with no smoothing, the cross-sectional area decreased. This means that the drag force that counteracts the buoyancy force is smaller, which finally results in a higher rising velocity. The calculated terminal velocity for no smoothing differs 4.2% from the benchmark case, and this is greatly reduced to 0.01% for a convolution support of $\delta = 5h$. As with the circularity, it makes no sense to consider the cases where a large convolution support distorts the development of the bubble, since it results in unphysical rising velocities.

Concluding remarks

Simulations of the rising bubble case with the use of convolution resulted in circularities and rising velocities that were in better agreement when compared to the reference data, up to a limit of a convolution support of $\delta = 5h$. The values where still a little underestimated. It is worth mentioning that the results of the benchmark case were obtained with Finite Element Methods, which are a higher order method than the currently used Finite Volume Method. This might explain the underestimation.

Unfortunately the results cannot be compared with experimental data, because the case considered is two dimensional, and a 3D case would take too long for the present work. However, given that Hysing et al. evaluated the algorithms of three independent research groups, and their results were very similar, the data presented in their study is a good numerical reference of what the circularity and rising velocity of this

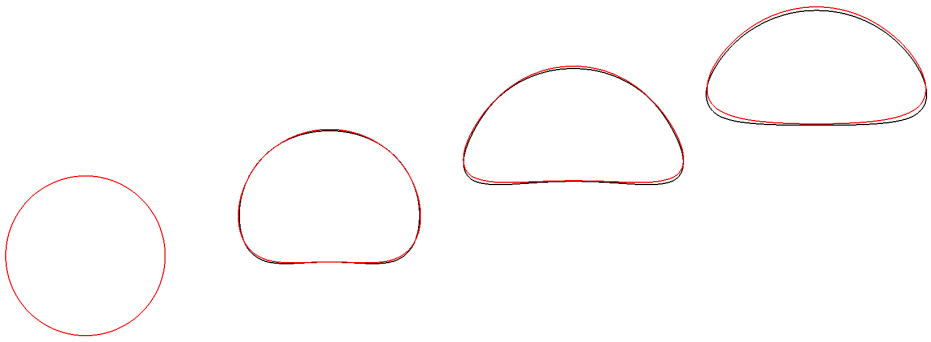


Figure 5.9: Shape evolution of an initially circular bubble for a simulation without smoothing (black) and with convolution of support $\delta = 4h$ (red).

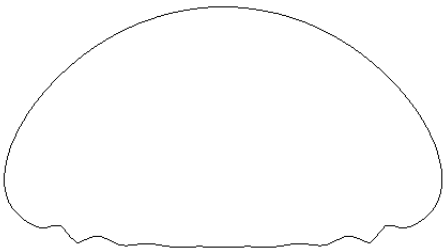
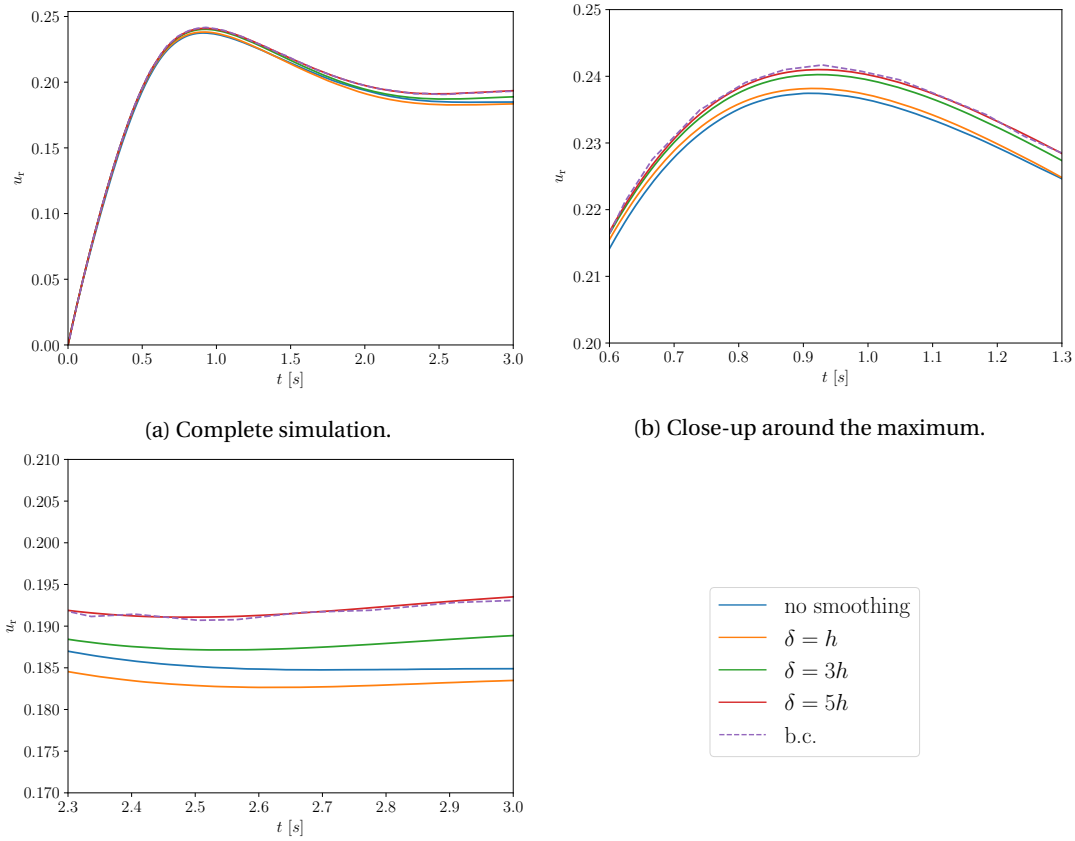


Figure 5.10: Bubble shape at $t = 3\,\mathrm{s}$ when a support of $\delta = 8h$ is used.

hypothetical test case should be.

The unexpected interface wave that emerged for a zero-gravity static bubble was not visible for the rising bubble. The current case has a Eo of 9, which means that the gravitational forces are much more significant that the surface tension forces. Therefore, the inaccuracies of the surface tension estimation have a smaller impact on the results of the simulation when compared to the static case, where Eo was 0.



(a) Complete simulation.

(b) Close-up around the maximum.

(c) Close-up around the terminal velocity.

Figure 5.11: Rising velocity as a function of time and close-ups around important moment during the simulation. The results from Hysing et al. [22] are given as reference, indicated by b.c.

6

Analysis of the Interface Wave Induced by the Convolution Method

In Section 5.3.2 it was observed that increasing the convolution support beyond $\delta = 2h$ was detrimental for the simulation of a static bubble. A wave formed on the interface that distorted the VOF colour field to such an extent that the spurious velocities greatly increased. This chapter attempts to investigate the factors that may have influence on the generation of such a wave. The same static bubble benchmark case as in Section 5.3 will be considered.

6.1. Parameters Relevant to Simulation

6.1.1. Curvature

Section 4.2.2 described how the error in the curvature estimation changes as a function of the convolution support. It was found that, for a circle, the curvature error decreased as the support δ increased. Figure 6.1 confirms that this is also the case for the curvature calculation in OpenFOAM.

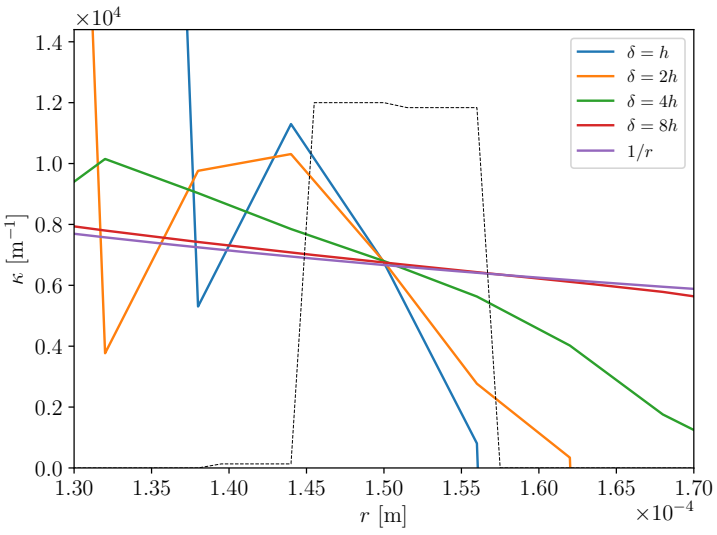


Figure 6.1: Curvature as a function of the distance r to the centre of the bubble for different convolution supports. The line is taken along the positive x-axis. The exact value of $1/r$ is plotted as reference. The dotted line is a scaled version of the surface tension force, which indicates the region where the curvature actually has an impact on the flow.

As the convolution support is increased, the curvature in the region around the interface better approximates the expected value of $1/r$. This behaviour was the same for lines at 8 equidistant angles between 0 and $\pi/2$ with respect to the x-axis. Therefore, regardless of the alignment of the grid with respect to the interface, convolution has a positive impact on the curvature calculation.

Given that the convolution method correctly predicts the curvature, and that using this method results in an interfacial wave, which does not occur without smoothing or when using a Laplacian smoother, Figure 6.2 compares the computed curvature of those methods. The Laplacian smoother uses 8 iterations, such that the smoothed VOF colour fields of the different smoothers are similar in terms of dispersion of the original colour field around the true interface.

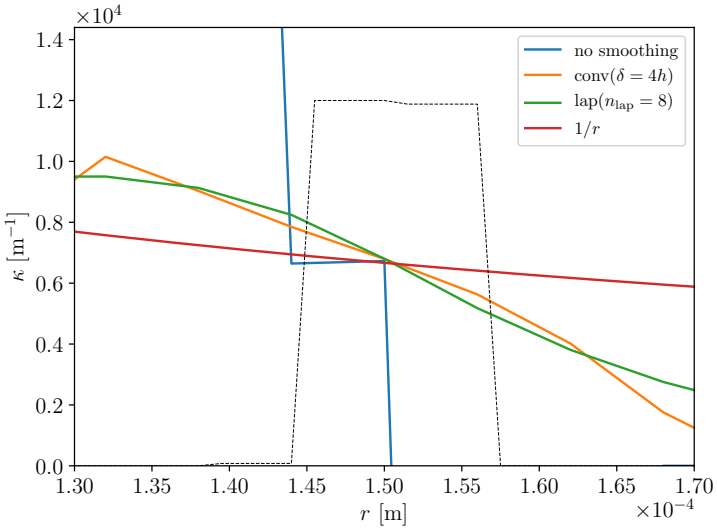


Figure 6.2: Similar to Figure 6.1, but now including the results of a simulation without smoothing and one with a Laplacian smoother.

It can be seen that the Laplacian filter correctly improves the curvature estimations as well, and does not differ significantly from the convolution smoother. The unsmoothed case provides the right curvature exactly where the interface is located, but fails to come close to the real curvature for the region around the interface. Surface tension is present in this region, so it is crucial to correctly estimate the curvature here as well. Therefore, the overall correction in the curvature for both smoothed cases when compared to the unsmoothed case is significant. Again, this same behaviour was observed for lines at 8 different angles from 0 to $\pi/2$, indicating independence of grid alignment.

6.1.2. Pressure

Two fluids experience a pressure jump at the interface separating the fluids. According to the Young-Laplace equation (2.3), this pressure jump should be $\Delta p = \sigma\kappa$. For the current case, the pressure jump should be $\Delta p = 157.3\,\text{Pa}$, given that the exact curvature is $\kappa = 1/R$. To evaluate whether this is achieved, simulations for cases with smoothing (convolution and Laplacian) and without smoothing were performed for the benchmark case and two further mesh refinements. The pressure jump was determined as the difference in pressure between the centre of the bubble and a point in the field far from the bubble. Table 6.1 summarizes the pressure jumps for the considered cases.

Table 6.1: Pressure jumps in Pa for three different smoothing options and cell dimensions.

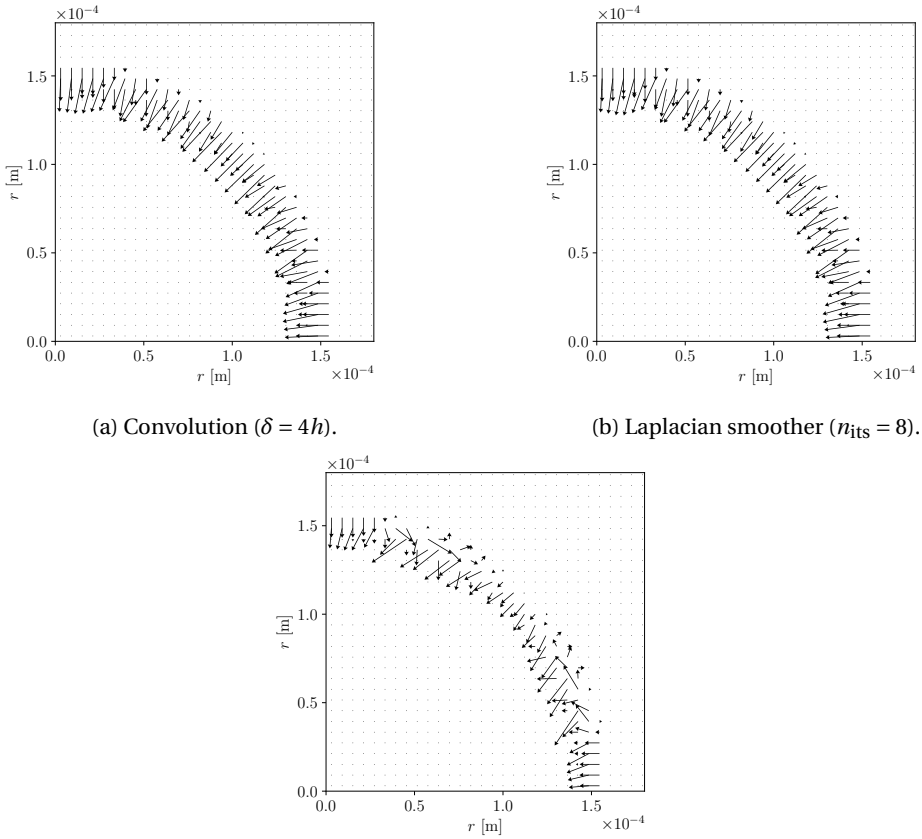
Case	**Cells per radius** R/h		
	25	**50**	**75**
No smoothing	127	133	136
Convolution	153	154	153
Laplacian	152	153	152

It was observed that all the simulations underestimate the true value of the pressure jump, and that further refining the mesh does not yield an improvement for the cases where smoothing was applied. The case with convolution seems to be the one that is the closest to the expected value, but does not differ much from the case with the Laplacian filter. No conclusion can be drawn by merely looking at the pressure jump, be-

cause it is the pressure gradient in combination with the surface tension force that predicts the flow. This is considered in Section 6.1.3

6.1.3. Surface Tension Force

The curvature has a direct impact on the resulting surface tension force that arises at a region around the interface. For the static bubble case, this force, which is modelled as a volumetric force, should compensate for the pressure jump between the considered fluids. Only then will the Navier-Stokes equations predict a zero velocity as solution. The two important factors to consider for a force are its magnitude and direction. Ideally, it should point radially inward. Figure 6.3 plots the vector field of the surface tension force in the first quadrant of the bubble.



(a) Convolution ($\delta = 4h$).

(b) Laplacian smoother ($n_{\text{its}} = 8$).

(c) No smoothing

Figure 6.3: Plot of the vector field of the surface tension force for different smoothers.

The direction of the surface tension force for the unsmoothed case is very distorted along the interface. This improves with the use of either smoother, but there are still vectorial components pointing in a direction other than the radial. The current study focused on the improvement of curvature calculations, which directly affects the magnitude of the force. The direction is given by the normal vector, defined as the gradient of the VOF colour field, as described in Equation 2.13. This normal vector is calculated with the unsmoothed VOF colour field, and is thus very sensitive to errors. The model of the current study also uses this approach, which is why the vector fields in Figure 6.3 are not directed perfectly in the radial direction. Even then, the normal vector seems to be approached better when a smoother is used. This can be explained by considering how OpenFOAM calculates the direction of the surface normal force.

In order to solve for the velocity and pressure fields, calculations in OpenFOAM are performed with respect to the cell faces. The surface tension force is therefore defined on cell faces (as a surface field) rather than cell centres (as a volume field). To achieve this, once the curvature is calculated at cell centres, these values are interpolated to the faces. This is done by averaging the curvature values of the two cells that are connected to a face. The gradient of the VOF colour field is calculated by projecting the centre value to the

faces. The two surface fields are multiplied with each other and scaled with the surface tension σ to obtain the surface tension force, at the cell faces. Finally, a reconstruction procedure is used to obtain fields that represent the volume of the cell, by averaging the face values. Each face represents a vector in the direction normal to the face, with the magnitude given by the face value. In this way, a vector is obtained for the body force that models the surface tension in each cell. Therefore, the direction of the force is affected by the magnitude of the curvature, since it has a direct impact on the face values, which then determine the length of the force component in the direction normal to the interface. This can explain why the direction of the surface tension force is better approximated with the smoothers despite calculating the normal vector with the gradient of the unsmoothed VOF colour field.

An interesting approach would be to calculate the normal vector needed for the direction of the surface tension force with the smoothed VOF colour field. Figure 6.4 plots the vector field of the surface tension force calculated in this way, for the same case and a convolution support of $\delta = 4h$.

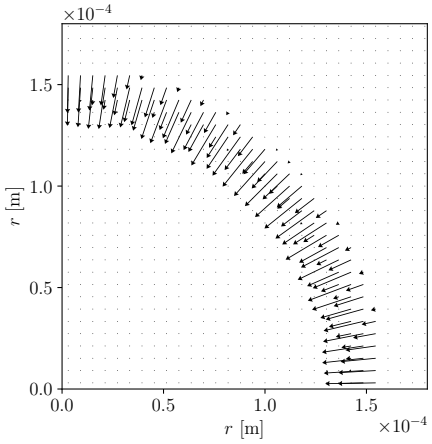


Figure 6.4: Vector field with the same magnitude as Figure 6.3a, but with the direction determined by the gradient of the smoothed VOF colour field. This is calculated outside OpenFOAM.

There is an evident improvement in the direction of the surface tension force when the normal vector is calculated with the smoothed VOF colour field. This gradient was calculated post processing from the cell centre values of this smoothed field. Simulations in OpenFOAM might benefit from this approach, regardless of the interpolation and reconstruction processes inherent to the solver.

The difference in magnitude of the surface tension force between smoothing with convolution and a Laplacian filter was small along the interface, but is definitely present. Figure 6.5 plots the magnitude of the surface tension force F_s along 16 lines with equidistant angles between 0 and $\pi/2$. The contribution to this magnitude comes from the interface cells located along the chosen lines.

The magnitude of the surface tension force oscillates with respect to the angular direction, which could generate an interfacial wave. However, the interfacial wave appeared only in the simulations where convolution was used, not with the Laplacian filter. Figure 6.5 shows that the Laplacian smoother also suffers from this oscillation as a function of θ, and therefore this distortion cannot explain why the convolution smoother has this problem.

The difference in magnitude of the surface tension force along the angular direction for a given smoother can be attributed to discretisation. Line integrals are performed for different angles, and some angles might provide a line that covers more computational cells where surface tension is present than others. This was nevertheless the best way of comparing the surface tension forces along the interface for the smoothers. Figure 6.5 therefore serves merely as a comparison between Laplacian smoothing and convolution, and has no absolute meaning. No conclusions can therefore be drawn about a variation of the surface tension force along the interface. What can be said is that a difference in the magnitude of the surface tension force, in combination with a poor estimation of the interface normal, could lead to unphysical interface development.

6.2. Conclusions

The analysis of the curvature, surface tension force and pressure values for simulations with different smoothers did not provide any reason for the appearance of an interface wave that would uniquely be present in the con-

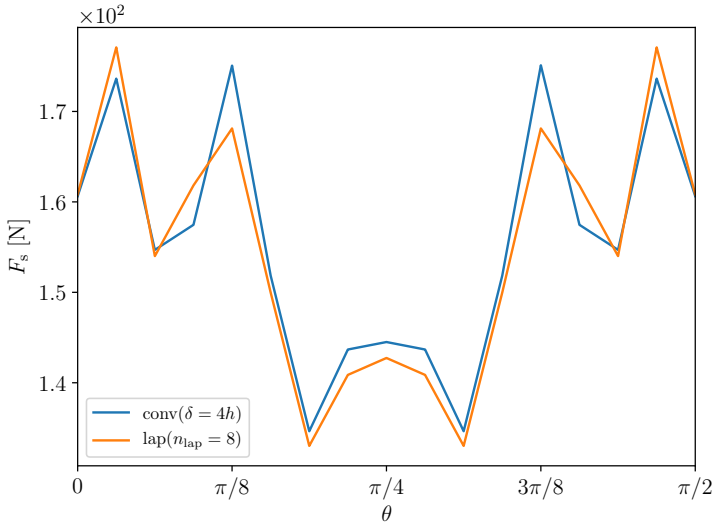


Figure 6.5: Magnitude of the surface tension force F_s along line with an angle of θ with respect to the x-axis for two different smoothers.

volution method. Curvature estimations are indeed improved when convolution is used to smooth the VOF colour field, but the resulting effect on the surface tension force, and therefore also on the time evolution of the bubble, does not benefit from this. Besides the curvature factor in the surface tension force, it is also important that the normal vector is estimated appropriately, since this will influence the direction of the force. The direction of the surface tension force did not vary significantly between simulations with the Laplacian smoother and convolution. However, this direction could be better approximated if the smoothed VOF colour field is used in the computations of the force direction instead of the otherwise abruptly varying VOF colour field.

It is evident that finding the origin of the formation of the wave at the interface requires a more extended analysis of the influencing factors, and an evaluation of the methodology used by OpenFOAM to estimate the factors present in the surface tension body force, including the impact of their interpolation and reconstruction methods to switch between cell centre values and face values. Any uneven compensation of forces for the static case may lead to the formation of a wave, where the direction is as important as the magnitude to obtain accurate results. Regardless, for the mere purpose of improving the curvature calculations, smoothing by means of convolution is advantageous.

7

Conclusions and Recommendations

7.1. Conclusions

In this study, the effect of smoothing the VOF colour field by means of convolution was investigated. Since this colour field describes an abruptly varying discrete function across the interface, calculations of the second order spatial derivative for curvature estimations often lead to high errors. These errors, in turn, introduce discrepancies in the surface tension forces along the interface, resulting in unphysical flows (known as spurious currents). Convolution is used to smooth the colour function, to obtain a more gentle transition of the function values and thereby improve the curvature estimations. The goal was ultimately to understand how convolution can influence these estimations, and the impact of the convolution parameters on the resulting field.

Convolution was first used to smooth a VOF field representing a circular interface, for which the curvature is constant. It was found that the curvature estimations improved as the convolution support increased. A uniform discretised grid was used with equal cell lengths in both dimensions, h. When the length of the convolution support was scaled as a linear function of h, the curvature diverged as the mesh was refined. It was found that if the length of the convolution support is scaled with $h^{2/3}$, the curvature converges to the theoretical value. Moreover, it was analytically proven that for interfaces with constant curvature, the original interface shape is conserved in the convoluted VOF field.

When using convolution for VOF fields representing interfaces with varying curvature, the behaviour was different. The simplest non-constant interface curvature shape was chosen: an ellipse. In contrast to the circular shape, there is an ideal convolution support to minimize the errors in the curvature for an ellipse: convolution of the VOF field acts as an averaging process of values from nearby cells, with the consequence that the true interface shape may change. Above this ideal convolution support, this change in shape becomes more significant than the advantage of having a smooth VOF field for the curvature calculation, for which the magnitude of the high order derivative that determine the truncation error are small. This effect was found to be the most significant at high values of the second order derivative of the interface curvature with respect to the polar angle, as confirmed by an experimental study. A correction for the change in interface shape may be needed in order to use convolution supports larger than the ideal length. A numerical study was performed on how to choose the ideal convolution support for an ellipsoidal shape, with several ratios of major to minor axis. The curvature errors are minimized if the convolution support is chosen as a function of the square root of the number of computational cells per semi-minor axis of the ellipse. The impact of ratios of major to minor axis on this ideal value was minimal.

The goal of improving the curvature calculations with convolution was to predict two-phase flow simulations in OpenFOAM more accurately, since this solver uses the unsmoothed VOF field for the curvature estimations. Moreover, it has an implementation of another smoother, the Laplacian filter, with which to compare the results. For a static bubble in zero-gravity, spurious velocities were reduced with small convolution supports, but resulted in an increasingly unphysical flow, visible as a wavy interface, when the support was extended beyond three computational cells. In these cases, the curvature was estimated properly, but this nevertheless had a negative impact on the simulations. There was no clear cause found for this behaviour, especially since the use of a Laplacian smoother did not suffer from the mentioned problem. The case of a rising bubble in a fluid with low density and viscosity ratios was also tested, where external forces are now

active. The circularity and rising velocity of an initially-circular bubble as a function of time were obtained more accurately when convolution was applied than without it. The terminal velocity of the simulation with convolution differed only 0.01% from a well-known benchmark case, whereas no convolution resulted in a deviation of 4.2%.

7.2. Recommendations

For further research, there are two topics that could be examined more carefully. First, a thorough analysis of the curvature calculation for interfaces with non-constant curvature can be done in order to determine for which convolution support the error is minimized. For this, both the impact of having a smoother field and changing the local shape of the interface with convolution need to be considered. Second, the improvement of the approximation of the surface tension in OpenFOAM can be looked into with more detail. The normal vector of the force is still determined with the unsmoothed VOF colour field, and therefore does not always point in the radial direction. This could improve the results from this study and maybe provide a reason for generation of an interface wave. Finally, the impact of the interpolation and reconstruction methods used by OpenFOAM should be investigated more deeply. These are used to retrieve field values at either cell centres or faces, and a repetitive use of these functions may affect the locality of the calculated field.

A

Filter weights

0	0.002	0.010	0.002	0
0.002	0.064	0.121	0.064	0.002
0.010	0.121	0.205	0.121	0.010
0.002	0.064	0.121	0.064	0.002
0	0.002	0.010	0.002	0

(a) K_6 kernel.

0	0	0.004	0	0
0	0.054	0.127	0.054	0
0.004	0.127	0.255	0.127	0.004
0	0.054	0.127	0.054	0
0	0	0.004	0	0

(b) K_8 kernel.

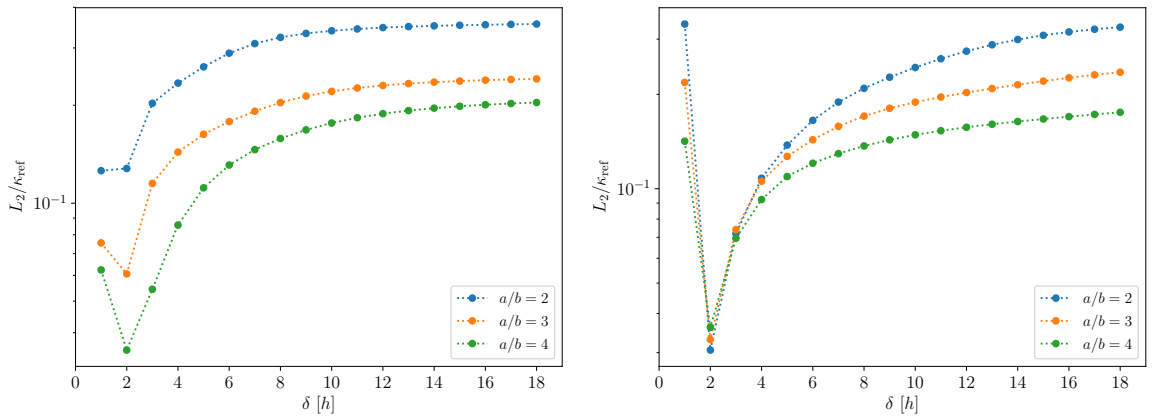
0	0.005	0.016	0.005	0
0.005	0.068	0.113	0.068	0.005
0.016	0.113	0.172	0.113	0.016
0.005	0.068	0.113	0.068	0.005
0	0.005	0.016	0.005	0

(c) $K_{\cos}$ kernel.

Figure A.1: Numerical values to three decimals of precision of the weights the filters K_6, K_8 and $K_{\cos}$ assign to its neighbouring cells for $\delta = 0.5$, where each mesh element size is 0.2×0.2.

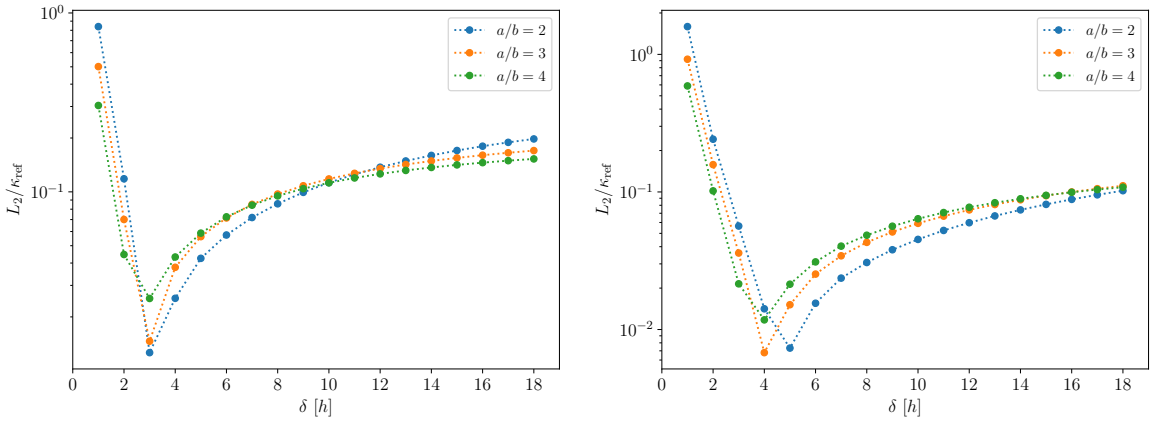
B

Plots of Curvature Error as a Function of Convolution Support for Different Mesh Dimensions



(a) 2 cells per b

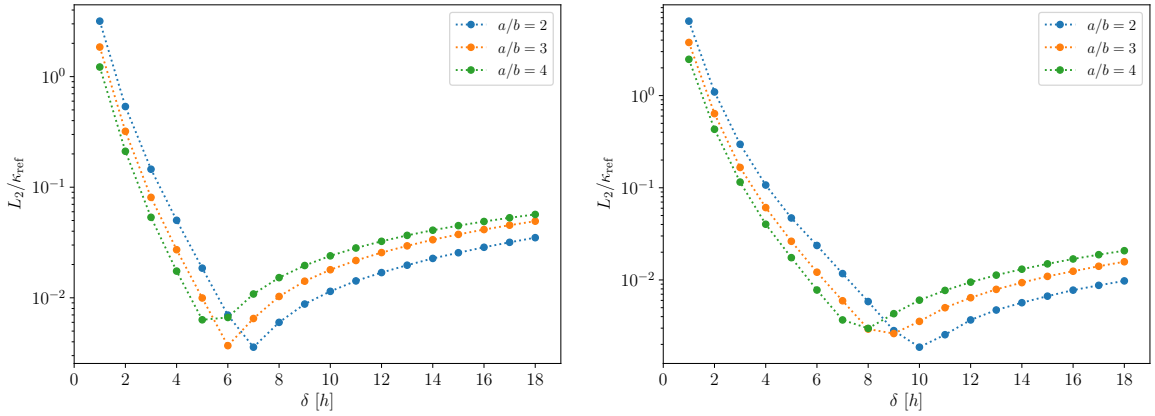
(b) 4 cells per b



(c) 8 cells per b

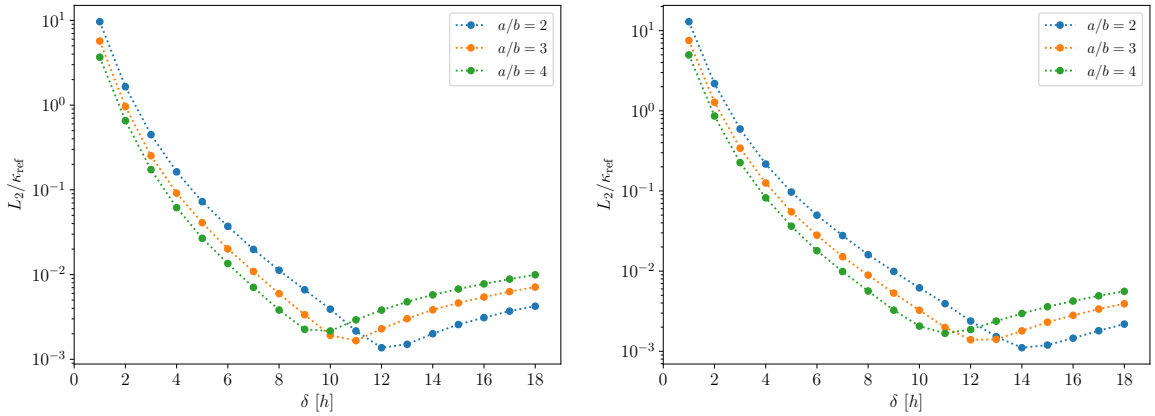
(d) 15 cells per b

Figure B.1: Curvature error of three ellipses with $a/b = 1$, 2 and 3 normalized with $\kappa_{\text{ref}} = a/b^2$ as a function of the convolution support δ for different mesh dimensions.



(a) 30 cells per b

(b) 60 cells per b



(c) 90 cells per b

(d) 120 cells per b

Figure B.2: Curvature error of three ellipses with $a/b = 1$, 2 and 3 normalized with $\kappa_{\text{ref}} = a/b^2$ as a function of the convolution support δ for different mesh dimensions.

C

Convolution Code

C.1. convolution.H

```
#ifndef smoothers_convolution_H
#define smoothers_convolution_H

#include "weightedSmootherKernel.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
namespace smoothers
{

/*---------------------------------------------------------------------------*\
                           Class normal Declaration
\*---------------------------------------------------------------------------*/

template<class Type>
class convolution
:
    public weightedSmootherKernel<Type>
{
    // Private data

        scalar convSupportConstant_;
        word kernelName_;
        typedef scalar (*KernelPtr)(scalar, scalar);
        KernelPtr kernel_;

        // Private Member Functions
        // Define a struct to store cell labels and their corresponding weights
        struct cellsAndWeights
        {
            DynamicList<label> neighbours;
            DynamicList<scalar> weights;
        };


        // General functions
        cellsAndWeights findConvolutionCellsAndWeights(const GeometricField<Type,
            ↪ fvPatchField, volMesh>& fld, int celli) const;

        inline scalar computeDistance(vector a, vector b) const { return mag(a-b); };

        inline bool insideKernel(scalar distance, scalar convSupport_) const { return (
            ↪ distance <= convSupport_); };

        bool contains(DynamicList<label>& list, scalar value) const;

```

```
47          Type convolutionOneCell(const GeometricField<Type, fvPatchField, volMesh>& fld,
                ↪  struct cellsAndWeights& convolutionCellsAndWeights) const;
48
49          // Convolution kernels
50          static inline scalar degree6Kernel(scalar distance, scalar convSupport_) {
                ↪ return pow(1-pow(distance/convSupport_,2),3); };
51
52          static inline scalar degree8Kernel(scalar distance, scalar convSupport_) {
                ↪ return pow(1-pow(distance/convSupport_,2),4); };
53
54          static inline scalar cosKernel(scalar distance, scalar convSupport_) { return
                ↪ (1+cos(M_PI*distance/convSupport_))/(2*convSupport_); };
55
56  public:
57
58      //- Runtime type information
59          TypeName("convolution");
60
61      // Constructors
62
63          //- Construct from dictionary
64          convolution
65          (
66                          const word& name,
67                          const dictionary & dict
68          );
69
70      //- Destructor
71                  virtual ~convolution(){}
72
73
74      // Member Functions
75
76                  virtual tmp<GeometricField<Type, fvPatchField, volMesh>> smoothen(const
                        ↪  GeometricField<Type, fvPatchField, volMesh>& fld) const;
77                  virtual tmp<GeometricField<Type, fvPatchField, volMesh>> smoothen(
78                          const GeometricField<Type, fvPatchField, volMesh>& fld,
79                          const tmp<volScalarField>& tweight
80                  ) const;
81  };
82
83
84  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
85
86  } // End namespace smoothers
87  } // End namespace Foam
88
89  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
90
91  #ifdef NoRepository
92      #include "convolution.C"
93  #endif
94
95  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
96
97  #endif
98
99  // ************************************************************************* //
```

C.2. convolution.C

```
1   #include "convolution.H"
2   #include "fvcWeightedAverage.H"
3
4   namespace Foam
5   {
6
7   // * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * //
8
9   template<class Type>
```

```
Foam::smoothers::convolution<Type>::convolution
(
        const word& name,
        const dictionary& dict
)
:
        weightedSmootherKernel<Type>(name, dict),
        convSupportConstant_(dict.lookupOrDefault<label>("convSup",0)),
    kernelName_(dict.lookupOrDefault<word>("kernel", "K6")),
    kernel_(NULL)
{
        if (convSupportConstant_ <= 0)
        {
        // Default value still needs to be adjusted
                int convSupportConstantDefault = 2;
                WarningInFunction
                        << "Specified convSup = " << convSupportConstant_ << "." << nl
                        << "    " << "This value must be positive. Assuming the default
                            ↪  value " << convSupportConstantDefault << " instead." <<
                            ↪ endl;
                convSupportConstant_ = convSupportConstantDefault;
        }

    if (kernelName_ == "K6")
    {
        kernel_ = &degree6Kernel;
    } else if (kernelName_ == "K8")
    {
        kernel_ = &degree8Kernel;
    } else if (kernelName_ == "Kcos")
    {
        kernel_ = &cosKernel;
    } else
    {
        FatalErrorInFunction << "Specified kernel " << kernelName_ << " for convolution
            ↪  not valid." << nl
            << "Valid kernel names are:\n\tK6\n\tK8\n\tKcos" << exit(FatalError);
    }
}

// * * * * * * * * * * * * * * * * * * Private Member Functions  * * * * * * * * * * * * * *
    ↪ * //
template<class Type>
typename Foam::smoothers::convolution<Type>::cellsAndWeights
Foam::smoothers::convolution<Type>::findConvolutionCellsAndWeights(const GeometricField
    ↪ <Type, fvPatchField, volMesh>& fld, int celli) const
{
    // Returns a list of lists, with the neighbour indices that are used in the
        ↪ convolution in the first list and their unnormalized weights in the second
        ↪ list
    // Each cell is weighted with the kernel weight and a cell volume weight, to give a
        ↪  larger importance to bigger cells
    // Dynamic list is used to store the cells for convolution. A rough estimate of the
        ↪  amount of neighbours of celli is given by convolutionCellsSize=(2*
        ↪ convSupport_)^d/fld.mesh().V()[celli], with d the dimensions of the
        ↪ simulation.

        // Access reference to cell centres and neighbours
        const volVectorField& cellCentres = fld.mesh().C();
        const labelListList& cellNeighbours = fld.mesh().cellCells();
        const scalarField& cellVolumes = fld.mesh().V();

        // Estimate local cell separation by averaging distances to cell neighbours
        scalar avgCellSep = 0;
        forAll(cellNeighbours[celli], counter)
        {
            avgCellSep += computeDistance(cellCentres[celli], cellCentres[
                ↪ cellNeighbours[celli][counter]]);
        }
        avgCellSep /= cellNeighbours[celli].size();

```

```
69          scalar convSupport_ = (convSupportConstant_+0.5)*avgCellSep;
70
71          int convolutionCellsSize = pow(convSupport_*2, 2);
72          DynamicList<label> convolutionCells(convolutionCellsSize);
73          convolutionCells.append(celli);
74          DynamicList<scalar> weights(convolutionCellsSize);
75          scalar weightCentreCell = kernel_(0, convSupport_)*cellVolumes[celli];
76          weights.append(weightCentreCell);
77
78          // Keep track of the sum of the weights for normalization
79          scalar totalWeight = weightCentreCell;
80
81          DynamicList<label> newCells;
82          newCells.append(celli);
83          int newCellsSize = 2;
84
85          // Loop until no new neighbours are added
86          while (newCellsSize != 0)
87          {
88              // Create a copy of the new cells list to avoid looping over a list that
                    ↪ changes size in the loop iterations
89              DynamicList<label> newCellsCopy = newCells;
90
91              // Clear the newCells list to fill it up again with the neighbours from
                    ↪ last iteration, which are now stored in newCellsCopy
92              newCells.clear();
93              forAll(newCellsCopy, cellj)
94              {
95                  const labelList neighbours = cellNeighbours[newCellsCopy[cellj]];
96                  forAll(neighbours, cellk)
97                  {
98                      scalar distance = computeDistance(cellCentres[celli], cellCentres[
                            ↪ neighbours[cellk]]);
99                      if (insideKernel(distance, convSupport_) && !(contains(
                            ↪ convolutionCells, neighbours[cellk])))
100                     {
101                         convolutionCells.append(neighbours[cellk]);
102                         scalar kernelWeight = kernel_(distance, convSupport_)*
                                ↪ cellVolumes[cellk];
103                         weights.append(kernelWeight);
104                         totalWeight += kernelWeight;
105                         newCells.append(neighbours[cellk]);
106                     }
107                 }
108             }
109             // If no new neighbours are added, newCellsSize becomes zero and the while
                    ↪ loop is broken
110             newCellsSize = newCells.size();
111         }
112
113         // Normalize weights
114         forAll(weights, cellm)
115         {
116             weights[cellm] /= totalWeight;
117         }
118
119         // Make a struct containing neighbours indices and their corresponding weights
120         cellsAndWeights convolutionCellsAndWeights;
121         convolutionCellsAndWeights.neighbours = convolutionCells;
122         convolutionCellsAndWeights.weights = weights;
123         return convolutionCellsAndWeights;
124 }
125
126 template<class Type>
127 bool Foam::smoothers::convolution<Type>::contains(DynamicList<label>& list, scalar
        ↪ value) const
128 {
129     // Returns true if value is contained in list, false otherwise
130     bool isContained = false;
131     forAll(list, item)
132     {
```

```
133         if (value == list[item])
134         {
135             isContained = true;
136             break;
137         }
138     }
139     return isContained;
140 }
141 
142 template<class Type>
143 Type Foam::smoothers::convolution<Type>::convolutionOneCell
144 (
145         const GeometricField<Type, fvPatchField, volMesh>& fld,
146         struct cellsAndWeights& convolutionCellsAndWeights
147 ) const
148 {
149     Type smoothValue(Zero);
150     forAll(convolutionCellsAndWeights.neighbours, celli)
151     {
152         smoothValue += fld[convolutionCellsAndWeights.neighbours[celli]]*
            ↪ convolutionCellsAndWeights.weights[celli];
153     }
154     return smoothValue;
155 }
156 
157 // * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * * //
158 
159 template<class Type>
160 tmp<GeometricField<Type, fvPatchField, volMesh>>
161 Foam::smoothers::convolution<Type>::smoothen(const GeometricField<Type, fvPatchField,
    ↪ volMesh>& fld) const{
162 
163         if(!this->weight_.valid()){
164                 FatalErrorInFunction
165                         << "Attempted to use smoothen(fld) with a smoother instance
                            ↪ that does not own its own weightFunction." << nl
166                         << "The function smoothen(fld, weight) should have been used
                            ↪ instead." << nl
167                         << "This indicates an error in the programming logic."
168                         << abort(FatalError);
169         }
170 
171         // Compute weightFactor
172         tmp<volScalarField> tweight = this->weight_->weight(fld.mesh());
173 
174         // Smoothen field
175         return smoothen(fld,tweight);
176         // tweight is automatically cleared (out-of-scope)
177 
178 }
179 
180 
181 template<class Type>
182 tmp<GeometricField<Type, fvPatchField, volMesh>>
183 Foam::smoothers::convolution<Type>::smoothen(
184         const GeometricField<Type, fvPatchField, volMesh>& fld,
185         const tmp<volScalarField>& tweight // Not yet implemented
186 ) const
187 {
188     Info << "(convolution) Smoothing(fld,tweight) " << fld.name() << "." << endl;
189     tmp<GeometricField<Type, fvPatchField, volMesh>> tfldSmooth
190     (
191         new GeometricField<Type, fvPatchField, volMesh>
192         (
193             IOobject
194             (
195                 "convolution("+fld.name()+')',
196                 fld.instance(),
197                 fld.mesh(),
198                 IOobject::NO_READ,
199                 IOobject::NO_WRITE
```

```
200             ),
201             fld.mesh(),
202             fld.dimensions()
203         )
204     );
205
206     GeometricField<Type, fvPatchField, volMesh>& fldSmooth = tfldSmooth.ref();
207
208     forAll(fld, celli)
209     {
210         struct cellsAndWeights convolutionCellsAndWeights =
                ↪ findConvolutionCellsAndWeights(fld, celli);
211         fldSmooth[celli] = convolutionOneCell(fld, convolutionCellsAndWeights);
212     }
213
214     typename GeometricField<Type, fvPatchField, volMesh>::
215     Boundary& bav = fldSmooth.boundaryFieldRef();
216
217     forAll(bav, patchi)
218     {
219         bav[patchi] = fld.boundaryField()[patchi];
220     }
221
222     fldSmooth.correctBoundaryConditions();
223
224     return tfldSmooth;
225 }
226
227 // ************************************************************************* //
228
229 } // End namespace Foam
230
231 // ************************************************************************* //
```

Bibliography

[1] C.E. Brennen. *Fundamentals of Multiphase Flow*. Cambridge University Press, 2005. ISBN: 9781107717671. URL: https://books.google.nl/books?id=zM5VAgAAQBAJ.

[2] R. Clift et al. *Bubbles, Drops, and Particles*. Academic Press, 1978. ISBN: 9780121769505. URL: https://books.google.nl/books?id=n8gRAQAAIAAJ.

[3] C.W. Hirt and B.D. Nichols. "Volume of fluid (VOF) method for the dynamics of free boundaries". In: *Journal of Computational Physics* 39.1 (1981), pp. 201 –225. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(81)90145-5.

[4] Malik Mayank, Fan Eric Sheung-Chi, and Bussmann Markus. "Adaptive VOF with curvature-based refinement". In: *International Journal for Numerical Methods in Fluids* 55.7 (2007), pp. 693–712. DOI: 10.1002/fld.1490.

[5] Mark Sussman, Peter Smereka, and Stanley Osher. "A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow". In: *Journal of Computational Physics* 114.1 (1994), pp. 146 –159. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1994.1155.

[6] Mark Sussman. "A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles". In: *Journal of Computational Physics* 187.1 (2003), pp. 110 –136. ISSN: 0021-9991. DOI: https://doi.org/10.1016/S0021-9991(03)00087-1.

[7] J.U. Brackbill, D.B. Kothe, and C. Zemach. "A continuum method for modeling surface tension". In: *Journal of Computational Physics* 100.2 (1992), pp. 335 –354. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(92)90240-Y.

[8] Sharen J. Cummins, Marianne M. Francois, and Douglas B. Kothe. "Estimating curvature from volume fractions". In: *Computers & Structures* 83.6 (2005). Frontier of Multi-Phase Flow Analysis and Fluid-Structure, pp. 425 –434. ISSN: 0045-7949. DOI: https://doi.org/10.1016/j.compstruc.2004.08.017.

[9] Marianne M. Francois et al. "A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework". In: *Journal of Computational Physics* 213.1 (2006), pp. 141 –173. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2005.08.004.

[10] Bruno Lafaurie et al. "Modelling Merging and Fragmentation in Multiphase Flows with SURFER". In: *Journal of Computational Physics* 113.1 (1994), pp. 134 –147. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1994.1123.

[11] Fabian Denner and Berend G. M. van Wachem. "Fully-Coupled Balanced-Force VOF Framework for Arbitrary Meshes with Least-Squares Curvature Evaluation from Volume Fractions". In: *Numerical Heat Transfer, Part B: Fundamentals* 65.3 (2014), pp. 218–255. DOI: 10.1080/10407790.2013.849996.

[12] M. W. Williams, D. B. Kothe, and E. G. Puckett. "Accuracy and Convergence of Continuum Surface Tension Models". In: *Fluid Dynamics at Interfaces*. Univ. Press, 1998, pp. 294–305.

[13] The OpenFOAM Foundation. *OpenFOAM v4 User Guide*. 2015. URL: https://cfd.direct/openfoam/user-guide-v4/.

[14] K. van As. "kva_interfaceProperties". In: *GitHub repository* (2017). URL: https://github.com/floquation/OF-kva_interfaceProperties.

[15] Onno Ubbink. "Numerical prediction of two fluid systems with sharp interfaces". PhD thesis. London: Imperial College London, Jan. 1997.

[16] L.P.B.M. Janssen and M.M.C.G. Warmoeskerken. *Transport Phenomena Data Companion*. 3rd ed. Delft: VSSD, 2006. ISBN: 9789071301599.

[17] Suman Chakraborty. "Surface-Tension-Driven Flow". In: *Encyclopedia of Microfluidics and Nanofluidics*. Ed. by Dongqing Li. New York, NY: Springer New York, 2015, pp. 3170–3186. ISBN: 978-1-4614-5491-5. DOI: 10.1007/978-1-4614-5491-5_1510.

[18] Fabian Denner. “Balanced-force two-phase flow modelling on unstructured and adaptive meshes”. PhD thesis. London: Imperial College London, Aug. 2013.

[19] Fabien Evrard, Fabian Denner, and Berend van Wachem. “Estimation of curvature from volume fractions using parabolic reconstruction on two-dimensional unstructured meshes”. In: *Journal of Computational Physics* 351.Supplement C (2017), pp. 271 –294. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2017.09.034.

[20] Charles S. Peskin. “Numerical analysis of blood flow in the heart”. In: *Journal of Computational Physics* 25.3 (1977), pp. 220 –252. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(77)90100-0.

[21] Duong A. Hoang et al. “Benchmark numerical simulations of segmented two-phase flows in microchannels using the Volume of Fluid method”. In: *Computers & Fluids* 86 (2013), pp. 28 –36. ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2013.06.024.

[22] Hysing S. et al. “Quantitative benchmark computations of two-dimensional bubble dynamics”. In: *International Journal for Numerical Methods in Fluids* 60.11 (2008), pp. 1259–1288. DOI: 10.1002/fld.1934.