PArallel, Robust, Interface Simulator (PARIS)

W. Aniszewski, T Arrufat, M Crialesi-Esposito, S Dabiri, Daniel Fuster, Y Ling, J Lu, L Malan, S Pal, R Scardovelli, et al.

PArallel, Robust, Interface Simulator (PARIS)

W. Aniszewski[a], T. Arrufat[a], M. Crialesi-Esposito[b], S. Dabiri[c], D. Fuster[a], Y. Ling[a,d], J. Lu[f], L. Malan[a,e], S. Pal[a], R. Scardovelli[g], G. Tryggvason[f], P. Yecko[h], S. Zaleski[a]

[a]*Sorbonne Université & CNRS, UMR 7190, Institut Jean Le Rond d'Alembert, F-75005, Paris, France*
[b]*CMT-Motores Térmicos, Universitat Politécnica de Valéncia, Camino de Vera, s/n, Edificio 6D, Valencia, Spain*
[c]*Mechanical Engineering, Purdue University, West Lafayette, IN, USA*
[d]*Mechanical Engineering, Baylor University, Waco, TX 76706, USA*
[e]*Mechanical Engineering, University of Cape Town, South Africa*
[f]*Mechanical Engineering, Johns Hopkins University, Baltimore, MD, USA*
[g]*DIN - Lab. di Montecuccolino, Università di Bologna, I-40136 Bologna, Italy*
[h]*Cooper Union, New York City, USA*

Abstract

Paris (PArallel, Robust, Interface Simulator) is a finite volume code for simulations of immiscible multi fluid or multiphase flows. It is based on the "one-fluid" formulation of the Navier-Stokes equations where different fluids are treated as one material with variable properties, and surface tension is added as a singular interface force. The fluid equations are solved on a regular structured staggered grid using a second-order explicit projection method. The interface separating the different fluids is tracked either using a Front-Tracking (FT) method where the interface is represented by connected marker points, or by a volume of fluid (VOF) method where the marker function is advected directly on the fixed grid. Paris is written in Fortran95/2002 and parallelized using MPI and domain decomposition. It is based on several earlier FT or VOF codes such as FTC3D, Surfer or Gerris. These codes and similar ones, as well as PARIS, have been used to simulate a wide range of multifluid and multiphase flows.

Keywords: Multiphase Flows, Multi Fluid Flows, Navier-Stokes equations, Front Tracking, Volume of Fluid, Surface Tension

PROGRAM SUMMARY

Program Title: PArallel Robust Interface Simulator — Paris
Licensing provisions: GPLv3.
Programming language: Fortran95/2002. Parallelized using MPI and domain decomposition.
Nature of problem:
Paris is a free code, or software, for the computational fluid dynamics (CFD) of multiphase flows, or computational multiphase fluid dynamics (CMFD), typically simulations of interfacial fluid flow, such as droplets, bubbles or waves as described in the book by Tryggvason, Scardovelli and Zaleski [1]. It solves the Euler or Navier-Stokes equations in the one-fluid formulation of two-phase flow, with constant surface tension. It computes complex flows such as fast atomizing jets or droplets, expanding cavitation bubble clusters and multiphase flow through porous media
Solution method:
The code mostly implements the methods described in the book by Tryggvason, Scardovelli and Zaleski [1]. Time stepping is performed using a simple second order in time predictor corrector method with an explicit projection for the pressure. Spatial discretisation is by finite volumes on a regular cuboid grid. Interface tracking is performed with either the Volume-Of-Fluid (VOF) or the Front-Tracking method. In the VOF version Paris uses either the Lagrangian Explicit advection method or the exactly mass conserving method of Weymouth and Yue [2]. The normal computation is performed using the Mixed Youngs Centered (MYC). A momentum-conserving advection method is implemented that is exactly consistent with Volume-Of-Fluid advection [3]. Curvature is computed with the Height-Function (HF) method. This is combined with the Balanced Continuous Surface Force (CSF) method to compute surface tension forces. Paris has a Lagrangian Point Particle and a Free Surface option.
Additional comments
Paris is extended from or inspired by the following codes:

- FTC3D2011 (Front Tracking Code for 3D simulations) by Gretar Tryggvason and Sadegh Dabiri
- Surfer VOF code by Stephane Zaleski, Jie Li, Ruben Scardovelli and others.
- Gerris Flow Solver by Stephane Popinet

[1] G. Tryggvason, R. Scardovelli, and S. Zaleski. *Direct Numerical Simulations of Gas-Liquid Multiphase Flows*. Cambridge University Press, 2011.
[2] G. D. Weymouth and Dick K. P. Yue. Conservative Volume-of-Fluid method for free-surface simulations on Cartesian-grids. *Journal of Computational Physics*, 229(8):2853–2865, April 2010.
[3] D. Fuster, T. Arrufat, M. Crialesi-Esposito, Y. Ling, L. Malan, S. Pal, R. Scardovelli, G. Tryggvason and S. Zaleski, A momentum-conserving, consistent, volume-of-fluid method for incompressible flow on staggered grids, *arXiv preprint* arXiv:1811.12327.

1. Introduction

Computations of the unsteady motion of multi fluid flows, where two or more immiscible fluids or thermodynamic phases flow while separated by sharp interfaces, date back to the earliest days of computational fluid dynamics (see [1, 2] for reviews). However, early simulations were restricted to relatively

small and idealized problems. As computer power has continued to grow, it has been increasingly possible to conduct Direct Numerical Simulations (DNS), defined as fully resolved and verified simulation of a validated system of equations that include non-trivial length and time scales. While such simulations are becoming increasingly common, most research groups need to devote considerable time to code development. For new groups, the need to develop a suitable simulation tool can be a significant barrier to entry. As a result, several of the authors have been involved in the development of several *free codes* such as Surfer [3], Gerris [4], Basilisk [5] and Paris [6]. In particular Paris is a free code that is intended to be relatively simple to use and modify, yet have sufficient capabilities so that it allows state-of-the-art studies of typical problems. Moroever it illustrates most of the methods described in [2]. (The latter book will be denoted "TSZ" in what follows to simplify citations.) In what follows we will often refer to "TSZ" for developments about the numerical methods.

2. Navier–Stokes equations with interfaces

2.1. Basic equations

In three-dimensional space, the locus of the interface is a smooth surface S which separates the two fluid phases. Accordingly, we consider that the interface is an object of zero thickness. This latter assumption constitutes the "sharp interface" approximation. In this approximation, the phases are implicitly located by a Heaviside function $\chi(\mathbf{x}, t)$ defined such that fluid 1 corresponds to $\chi = 1$ and fluid 2 to $\chi = 0$. Viscosity and density μ and ρ are space and time dependent and given by

$$\mu = \mu_1\chi + \mu_2(1-\chi)\,, \qquad \rho = \rho_1\chi + \rho_2(1-\chi)\,. \tag{1}$$

In the case with no phase change, mass conservation implies that the interface advances at the speed of the flow, that is

$$V_S = \mathbf{u}(\mathbf{x}, t)\cdot\mathbf{n} \tag{2}$$

where $\mathbf{u}(\mathbf{x}, t)$ is the local fluid velocity and $\mathbf{n}$ a unit normal vector perpendicular to the interface. Equivalently this condition on the interface motion can be expressed, in weak form, as

$$\partial_t\chi + \mathbf{u}\cdot\nabla\chi = 0\,, \tag{3}$$

which expresses the fact that the singularity of χ, located on S, moves at velocity $V_S = \mathbf{u}\cdot\mathbf{n}$. We refer the reader to the litterature, in particular TSZ, for additional developments on interface geometry.

For incompressible flows, which we will consider in what follows, we have

$$\nabla\cdot\mathbf{u} = 0\,. \tag{4}$$

The Navier–Stokes equations for incompressible, Newtonian flow with surface tension may conveniently be written in either conservative form, expressing the momentum balance, or in a non conservative form. The first form, using operators for notational simplicity is

$$\partial_t(\rho\mathbf{u}) = \mathcal{L}_1(\rho, \mathbf{u}) - \nabla p \tag{5}$$

where $\mathcal{L}_1 = \mathcal{L}_{\text{cons}} + \mathcal{L}_{\text{diff}} + \mathcal{L}_{\text{cap}} + \mathcal{L}_{\text{ext}}$ so that the operator $\mathcal{L}_1$ is the sum of a conservative momentum transport term, and diffusive, capillary force and external force terms. The first two terms are

$$\mathcal{L}_{\text{cons}} = -\nabla\cdot(\rho\mathbf{u}\mathbf{u})\,, \qquad \mathcal{L}_{\text{diff}} = \nabla\cdot\mathbf{D}\,, \tag{6}$$

where $\mathbf{D}$ is a stress tensor whose expression for incompressible flow is

$$\mathbf{D} = \mu\left[\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right], \tag{7}$$

where μ is computed from χ using (1). The capillary term is

$$\mathcal{L}_{\text{cap}} = \sigma\kappa\delta_S\mathbf{n} + \nabla_S\sigma\delta_S\,, \qquad \kappa = 1/R_1 + 1/R_2\,, \tag{8}$$

where σ is the (possibly non-constant) surface tension, $\mathbf{n}$ is the unit normal perpendicular to the interface, κ is the sum of the principal curvatures and δ_S is a Dirac distribution concentrated on the interface. We assume a constant surface tension value σ. Finally $\mathcal{L}_{\text{ext}}$ represents external forces. When the external force is gravity $\mathcal{L}_{\text{ext}} = \rho\mathbf{g}$.

The second, non-conservative form, is

$$\partial_t\mathbf{u} = \mathcal{L}_2(\rho, \mathbf{u}) - \frac{1}{\rho}\nabla p \tag{9}$$

where $\mathcal{L}_2 = \mathcal{L}_{\text{adv}} + \frac{1}{\rho}(\mathcal{L}_{\text{diff}} + \mathcal{L}_{\text{cap}} + \mathcal{L}_{\text{ext}})$. The first term is

$$\mathcal{L}_{\text{adv}}(\mathbf{u}) = -\nabla\cdot(\mathbf{u}\mathbf{u}) = -(\mathbf{u}\cdot\nabla)\mathbf{u}, \tag{10}$$

and the other terms are already defined above.

2.2. Boundary conditions

A major difficulty with numerical simulations of fluid flow is the correct implementation of the boundary conditions. In principle the conditions at boundaries are well defined. For viscous, incompressible fluids we require that the fluid sticks to the wall so the fluid velocity there is equal to the wall velocity

$$\mathbf{u} = \mathbf{U}_{wall}.$$

In a numerical setup, we can also impose periodic boundary conditions, as well as inflow or outflow conditions (see TSZ).

2.3. Free-surface flow

Free-surface flow is a limiting case of flow with interfaces, in which the treatment of one of the phases is simplified. For instance, for some cases of air-water flow, we may consider the pressure p in the air to depend only on time and not on space (through, say, some function $p_{\text{air}}(t)$) and the viscous stresses in the air to be negligible. The jump conditions become *boundary conditions* on the border of the liquid domain:

$$(-p + 2\mu\mathbf{n}\cdot\mathbf{D}\cdot\mathbf{n})|_S = p_{\text{air}} + \sigma\kappa \tag{11}$$

and

$$\mu\,\mathbf{t}^{(k)}\cdot\mathbf{D}\cdot\mathbf{n}\big|_S = \mathbf{t}^{(k)}\cdot\nabla_S\sigma. \tag{12}$$

for $k = 1, 2$, where $\mathbf{t}^{(1,2)}$ are two independent tangent vectors.

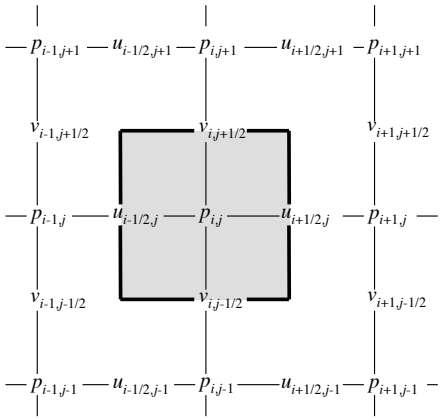

Figure 1: Representation of the staggered spatial discretisation. The pressure p is assumed to be known at the center of the control volume outlined by a thick solid line. The horizontal velocity component $u_1 = u$ is stored in the middle of the left and right edges of this control volume and the vertical velocity component $u_2 = v$ in the middle of the top and bottom edges.

3. Numerical methods implemented in the code

3.1. Spatial discretization

We assume a regular cuboid grid. A cuboid grid can be defined as a cubic grid stretched independently in the x, y and z directions, so that the centers of the cells $\Omega_{i,j,k}$ are given by the interesection set of planes $x = x_i, y = y_j, z = z_k$ and the cell boundaries are contained in the set of planes $x = x_{i+1/2}, y = y_{j+1/2}, z = z_{k+1/2}$. However in multiphase flow only the Front part of Paris can use stretched coordinates, and the VOF part uses cubic grids. We also use staggered velocity and pressure grids.

The staggered grid is represented in Figure 1. We use a finite volume discretisation of the advection equation. The corresponding control volumes of the velocity components u_m in direction m are shifted with respect to the control volume $\Omega_{i,j,k}$ surrounding the pressure p. The use of staggered control volumes has the advantage of suppressing neutral modes often observed in collocated methods but leads to more complex discretizations (see [2] for a more detailed discussion.) The control volumes for u_1 and u_2 are shown on Figure 2. This type of staggered representation is easily generalized to three dimensions. In what follows we shall use these control volumes for the velocity or momentum components.

Using the staggered grid leads to a compact expression for the continuity equation (4)

$$\frac{u_{1;i+1/2,j,k} - u_{1;i-1/2,j,k}}{\Delta x} + \frac{u_{2;i,j+1/2,k} - u_{2;i,j-1/2,k}}{\Delta y} + \frac{u_{3;i,j,k+1/2} - u_{3;i,j,k-1/2}}{\Delta z} = 0, \tag{13}$$

In what follows, we shall use the notation $f = m\pm$, with the integer index $m = 1, 2, 3$, to note the face of any control volume located in the positive or negative Cartesian direction m, and $\mathbf{n}_f$ for the normal vector of face f pointing outwards of the control volume. On a cubic grid the spatial step is $\Delta x = \Delta y = \Delta z = h$

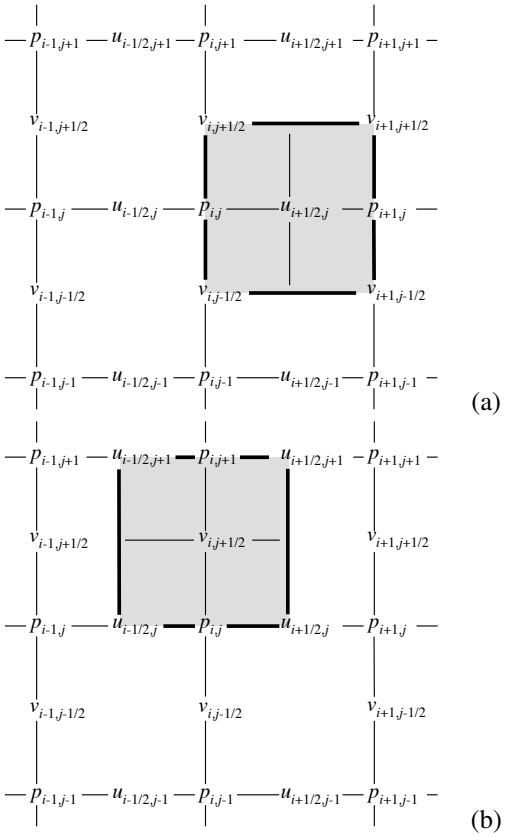

Figure 2: The control volumes for the $u_1 = u$ and the $u_2 = v$ velocity components are displaced half a grid cell to the right (horizontal velocities) and to the top (vertical velocities). Here the indices show the location of the stored quantities. Thus, half indices indicate those variables stored at the edges of the pressure control volumes.

so the continuity equation becomes

$$\nabla^h \cdot \mathbf{u} = \sum_{m=1}^{3} (u_{m+} + u_{m-})/h = 0\,, \tag{14}$$

where $u_f = u_{m\pm} = \mathbf{u} \cdot \mathbf{n}_f$ is the velocity normal to face f. The discretization of the interface location is performed using a VOF method. VOF methods typically attempt to solve approximately equation (3) which involves the Heaviside function χ, whose integral in the cell $\Omega_{i,j,k}$ defines the volume fraction $C_{i,j,k}$ from the relation

$$h^3\, C_{i,j,k} = \int_{\Omega_{i,j,k}} \chi \mathrm{d}\mathbf{x}\,. \tag{15}$$

$C_{i,j,k}$ represents the fraction of the cell labelled by i, j, k filled with fluid 1, taken to be the reference fluid. It is worth noting that in the staggered grid setup, the control volume for p is also the control volume for other scalar quantities, such as ρ and χ.

3.2. Time Marching

Time marching can be performed in a first-order or second-order manner using a small, possibliy variable time step τ so that $t_{n+1} = t_n + \tau$. We first consider the first order time stepping. The interface is advanced in time as follows. In the Front-Tracking case, we implement eq. (2). The front points $\mathbf{x}_k$ are moved as

$$\mathbf{x}_k^{n+1} = \mathbf{x}_k^n + \tau \mathbf{u}_i(\mathbf{x}_k^n, t_n) \tag{16}$$

where $\mathbf{u}_i(\mathbf{x}, t_n)$ is an interpolation of the velocity field at the location $\mathbf{x}$ (see Section 3.3 for the definition of the front points). In the Volume-Of-Fluid (VOF) case the fraction field is updated as

$$C^{n+1} = \mathcal{L}_{\mathrm{VOF}}(C^n, \mathbf{u}^n \tau/h)\,, \tag{17}$$

where $\mathcal{L}_{VOF}$ represents the operator that updates the Volume of Fluid data given the velocity field, described in detail in Section 3.4. Once volume fraction is updated, the velocity field is updated in several steps. A projection method is first used, in which a provisional velocity field $\mathbf{u}^*$ is computed. Two different versions are used. One is the "non-momentum conserving" version

$$\begin{aligned} \mathbf{u}^* &= \mathbf{u}^n + \tau \mathcal{L}^h_{\mathrm{adv}}(\mathbf{u}^n) \\ &+ \frac{\tau}{\rho^{n+1}} \left[\mathcal{L}^h_{\mathrm{diff}}(\mu^{n+1}, \mathbf{u}^n) + \mathcal{L}^h_{\mathrm{cap}}(C^{n+1}) + \mathcal{L}^h_{\mathrm{ext}}(C^{n+1}) \right], \end{aligned} \tag{18}$$

the other is a "momentum-conserving" version

$$\begin{aligned} \rho^{n+1}\mathbf{u}^* &= \rho^n \mathbf{u}^n + \tau \mathcal{L}^h_{\mathrm{cons}}(\rho^n, \mathbf{u}^n) \\ &+ \tau \left[\mathcal{L}^h_{\mathrm{diff}}(\mu^{n+1}, \mathbf{u}^n) + \mathcal{L}^h_{\mathrm{cap}}(C^{n+1}) + \mathcal{L}^h_{\mathrm{ext}}(C^{n+1}) \right]. \end{aligned} \tag{19}$$

Clearly the above operators depend on the discretization steps τ and h as well as the fluid parameters. The "momentum-conserving" $\mathcal{L}^h_{\mathrm{cons}}$ operator is described in detail in [7]. In a second step the pressure gradient corrects the velocity

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\tau}{\rho^{n+1}} \nabla^h p\,. \tag{20}$$

This constitutes the so-called projection method. The pressure is determined by the requirement that the velocity at the end of the time step must have zero divergence

$$\nabla^h \cdot \mathbf{u}^{n+1} = 0\,, \tag{21}$$

which leads to an elliptic equation for the pressure

$$\nabla^h \cdot \frac{\tau}{\rho^{n+1}} \nabla^h p = \nabla^h \cdot \mathbf{u}^*\,. \tag{22}$$

The whole set of operations above constitutes a first-order in time approximation, which can be written

$$(\mathbf{f}^{n+1}, \mathbf{u}^{n+1}) = \mathbf{L}(\mathbf{f}^n, \mathbf{u}^n) \tag{23}$$

where $\mathbf{f}^n$ is the interface data at t_n (either the front points $\mathbf{x}$ or the VOF fraction C) and $\mathbf{L}$ is the operator consisting in the steps described above. A second-order time scheme can be obtained as a first prediction

$$(\mathbf{f}^{**}, \mathbf{u}^{**}) = \mathbf{L}(\mathbf{f}^n, \mathbf{u}^n) \tag{24}$$

followed by the average

$$(\mathbf{f}^{n+1}, \mathbf{u}^{n+1}) = \frac{1}{2} \left[\mathbf{L}(\mathbf{f}^{**}, \mathbf{u}^{**}) + (\mathbf{f}^{**}, \mathbf{u}^{**}) \right] \tag{25}$$

The Paris code implements both the first-order time scheme and the second-order one above, controlled by the parameter ITIME_SCHEME.

3.2.1. Simple, non-conservative momentum advection

The non-conservative momentum advection (9) amounts to integrate over one time step the PDE

$$\partial_t u_k = \mathcal{L}_{\mathrm{adv},k}(\mathbf{u}) \tag{26}$$

where

$$\mathcal{L}_{\mathrm{adv},k} = u_j \partial_j u_k \tag{27}$$

for each value of the index k. Because of incompressiblity (4) it is equivalent to solve for a scalar field $\phi = u_k$ in the manner

$$\partial_t \phi + \nabla \cdot (\phi \mathbf{u}) = 0 \tag{28}$$

Integrating the advection equation over a control volume centered on a node of the scalar ϕ and integrating in time one obtains

$$\phi^{n+1}_{i,j,k} - \phi^n_{i,j,k} = -\sum_{\text{faces f}} F_f^{(\phi)}. \tag{29}$$

We use $F_f^{(\phi)} = \phi_f \mathbf{u}_f \cdot \mathbf{n}_f \tau/h$ as an approximation of the flux on face f. Let $u_f = \mathbf{u}_f \cdot \mathbf{n}_f$. At first order u_f is obtained by simple averaging. For example if we consider the first component of velocity $\phi = u_1 = u$, whose control volume is centered on $i + 1/2, j, k$ (see Figure 2) and the normal to the face is oriented in the x direction, the face is located at index $i + 1, j, k$. We have $u_f = u_{1,i,j,k} = \frac{1}{2}(u_{1,i-1/2,j,k} + u_{1,i+1/2,j,k})$. If on the other hand the normal to the face is oriented in the y direction, the face is located at index $i + 1/2, j + 1/2, k$ we have $u_f = u_{2,i+1/2,j+1/2,k} = \frac{1}{2}(u_{2,i,j+1/2,k} + u_{2,i+1,j+1/2,k})$.

Contrary to the estimates of u_f above, the estimation of ϕ_f may involve more complex and higher-order schemes. Indeed one-dimensional interpolation schemes incorporating flux limiters are typically used. Most of these schemes are described in TSZ together with their usage in computing the face fluxes in the bulk. We describe their general properties here shortly. Since the schemes are one-dimensional we can consider a variable ϕ defined on a regular one-dimensional grid. Specializing further the example consider that the variable ϕ, akin to the component u, takes values $\phi_{i+1/2}$ at half-integer grid point indexes. We need to estimate the flux at integer index points x_i and we thus need to predict $\phi_i = \phi_f$. An interpolation function is defined that predicts this value as a function of the four nearest points, and in an upwind manner based on the sign of u_f, given by centered interpolations. The face value ϕ_i is approximately given by

$$\phi_i = f(\phi_{i-3/2}, \phi_{i-1/2}, \phi_{i+1/2}, \phi_{i+3/2}, u_i) \tag{30}$$

where the function f is both of sufficiently high order and limits the flux. To express the function f, Paris offers a choice of the ENO, QUICK, Superbee, WENO, first-order upwind, Verstappen, or BCG schemes.

When momentum is advected in the "simple" method, ϕ in equation (29) is taken equal to one of the velocity components u_k. This method is available whether one uses the VOF method or the Front-Tracking method.

3.2.2. Conservative, VOF-consistent momentum advection

When using VOF, another momentum advection method is available that is consistent with VOF advection and which implements a conservative scheme of the form (20). This means that the same advection method is used near the interface for the VOF color function C and for the velocity $\mathbf{u}$. In other words, when there is a density jump on the interface, the discontinuity of $\rho\mathbf{u}$ is advected exactly at the same velocity as the discontinuity of $\mathbf{u}$. This can be expressed by saying that the momentum advection and the VOF advection are *consistent*. An explicitly formulated criterion for consistency is the following: if the velocity $\mathbf{u}$ is uniform, then $\rho\mathbf{u}$ remains exactly proportional to ρ. This should happen even while ρ is obtained from the VOF- advection of C using (1) and (17) and $\rho\mathbf{u}$ is obtained from the operator $\mathcal{L}^h_{\text{cons}}$. Such a "VOF-consistent" method is used since it has been empirically found by several authors that the simple advection in the previous section was often unstable at large density ratios, while consistent methods are more stable [8, 9, 10, 11, 12, 13, 14, 15].

The Paris code implements a modification of the classical momentum-preserving scheme proposed by [16] for the case of a staggered grid and Volume of Fluid (VOF) method. It is described in detail in [7]. The scheme needs to be modified from the one in the previous section only near the interface. Away from the interface, the density is constant and the scheme in (29) is already conservative. The scheme in the code comes with a set of choices for the flux limiters away from the interface which is the same as in the previous section (ENO, QUICK, Superbee, WENO, first-order upwind, Verstappen, or BCG schemes) and a different set of flux limiters near the interface which are chosen among ENO, Superbee, WENO and first-order upwind. It was found (see ref. [7]) that the most stable scheme is the combination of the CIAM method with the superbee flux limiter.

3.2.3. Implicitation of the viscous terms

The operator $\mathcal{L}^h_{\text{diff}}(\mu^{n+1}, \mathbf{u}^n)$ may be treated in part implicitly. We have

$$\mathcal{L}_{\text{diff,j}}(\mu, \mathbf{u}) = (\partial_i\mu)(\partial_i u_j + \partial_j u_i) + \mu\nabla^2 u_j \tag{31}$$

The first term on the RHS is left explicit but the second term can be made implicit by solving the linear problem

$$u^*_j = u^n_j + \tau\mu^{n+1}\nabla^{h,2}u^*_j\,. \tag{32}$$

Then the discrete operator is defined as

$$\tau\mathcal{L}^h_{\text{diff,j}}(\mu^{n+1}, \mathbf{u}^n) = (\partial_i\mu^{n+1})(\partial_i u^n_j + \partial_j u^n_i) + u^*_j - u^n_j \tag{33}$$

where u^*_j is the solution of the linear problem (32). The implicitation of the viscous terms is optional and controlled by a code parameter.

3.3. Front-Tracking

Interface tracking in the code can be performed either by Front Tracking or by the Volume-Of-Fluid method. We describe the former in this Section. Front-Tracking, in the context of simulations of two or more immiscible fluids, refers to tracking the interface separating the different fluids using moving connected marker points that represent the interface. In our implementation the marker points are connected by triangular elements, where the points are ordered in the same way for all elements, allowing us to define an "inside" and an "outside" for each element. The coordinates of the points are stored in arrays, in arbitrary order, with separate integer arrays providing pointers to the previous and next points. Thus, the points form a linked list where the location in the array provides each point with a unique ID. The elements are stored in the same way, with arrays containing pointers to the corner or node points. The coordinates of the marker points are the main quantities stored for the points, but the points also have arrays for various temporary quantities, such as velocities and the surface force. The marker points and the elements connecting the points together form the "front." In addition to pointers to their corner points, the elements also have pointers to the elements that share edges with them. These are mostly used for modifications, or reconstruction, of the front. Notice that generally only one front is needed, irrespectively of the number of distinct interfaces, and that distinct interfaces, such as in a simulations containing several bubbles and drops, for example, can have different material properties, such as surface tension. Figure 3 shows the layout of a triangulated front separating two different fluids.

As the front is deformed, stretched and compressed by the flow, the size of the elements change as points move away from, or closer to, each other. We keep the length of the edges of each elements within about a quarter to a half of the grid spacings

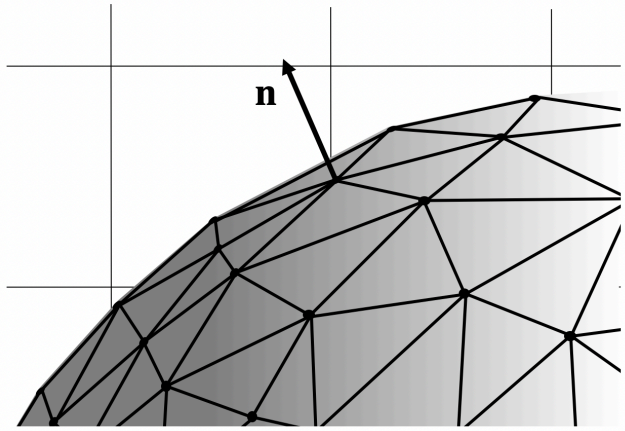

Figure 3: An interface separating two immiscible fluids represented by marker points connected by triangular elements.

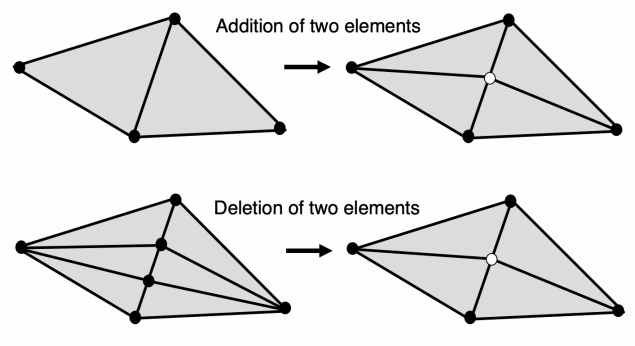

Figure 4: Restructuring of a triangulated grid by adding and deleting points and elements.

of the fluid grid, and to maintain that resolution, points and elements are dynamically added and deleted. While many strategies are possible, we add points by splitting the longest edge of an element by adding one point and two elements, and delete points by collapsing the shortest edge of an element, removing one point and two elements. The grid quality can sometimes also be improved by changing the connectivity of the elements but we generally find that doing so is not necessary. The addition and deletion of front points and elements is shown in Figure 4.

3.3.1. Connecting the front and the fluid grid

Since the Navier-Stokes equations are solved on a fixed grid, we have two grids: the front/interface grid and the fixed grid. Since the motion of the interface depends on the flow and the flow depends on where the interface is, information must be passed back and forth between the front and the fixed fluid grid. To do so we need to identify what front point is close to which fixed grid point and vice versa. For a regular structured grid, where the grid lines are straight and evenly spaced, it is straightforward to locate a point on the fixed grid that is closest to a given front point (using an INT or a MOD function), but finding the front point closest to a given grid point generally requires us to examine the distance to all the front points. Thus, it is more efficient to do all communications between the front and the fixed volume grid by looping over the front points. For periodic domains we allow the front to move out of the domain resolved by the fixed fluid grid, and use a MOD function to find the fixed grid point that would be closest to a given front point if we moved the front back into the original domain.

To transfer information between the fixed fluid grid and the moving front, we need to interpolate information from the grid and spread, or "smooth," information from the front to the fixed grid. To move the front we need the velocity at the front points and those are interpolated from the fixed grid. On a staggered grid each velocity component is interpolated separately. In general, we have:

$$\phi^l_f = \sum_{ijk} w^l_{i,j,k} \phi(i,j,k). \tag{34}$$

Here, ϕ^l_f is a quantity, such as the velocity, on the front at point l, $\phi(i,j,k)$ is the same quantity on the fluid grid, $w^l_{i,j,k}$ is the weight of each grid point with respect to front point l and the sum is over grid points "close" to the front points. Generally the same time integration method is used for the advection of the points as is used for updating the fluid velocities.

The transfer of information, such as surface tension, from the front to the fluid grid is usually referred to as smoothing, since doing so replaces a quantity defined on a sharp interface, at a front point, with a distribution on the fixed grid, where each fixed grid point receives a value according to how close it is to the front point. Unlike the interpolation of a quantity such as the velocity from the fixed grid to a front point, smoothing usually involves quantities like a force, that are given in terms of force per unit interface area on the front but must be converted to force per unit volume on the fixed grid, so that the total force is conserved. Thus, the quantity smoothed must be scaled by the ratio of the area associated with each front point divided by the volume of a fixed grid cell :

$$\phi(i,j,k) = \sum_l \phi^l_f w^l_{i,j,k} \frac{\Delta s_l}{\Delta x \Delta y \Delta z}, \tag{35}$$

where Δs is a surface area of a front element and Δx, Δy and Δz are the grid spacings.

Several interpolation/smoothing functions can be used, but in the PARIS code we use a smoother interpolation function originally introduced in [17] which involves four grid points in each coordinate direction, or 64 points total. The weights are given by

$$w^l_{i,j,k} = d(r_x) d(r_y) d(r_z), \tag{36}$$

where r_x is the scaled distance (by the grid spacing) between x^l_f and the grid line located at x_i. r_y and r_z are defined in the same way. In our case, r is given by

$$d(r) = \begin{cases} (1/4)(1+\cos(\pi r/2)), & |r| < 2, \\ 0, & |r| \geq 2. \end{cases} \tag{37}$$

Using fewer points gives a sharper transition zone but sometimes leads to wiggles, particularly for stiff problems. The interpolation function is bounded with weights that sum to one in addition to having various desirable symmetry properties. For a discussion see the reference above.

3.3.2. *Constructing the marker function*

Once the front has been moved, a marker function must be constructed on the fixed grid to assign the different material properties to each grid point. This can be done in many different ways, but one of the consideration is that fronts that are so close to each other that the flow between them is not resolved, must be handled in a plausible way. Usually this means that the marker function must retain its correct value on both sides of the double front. In the PARIS code we do this by working with the gradient of the marker function, which in the limit of a sharp interface should be a delta function defined only on the interface. The delta function is then treated in the same way as the surface tension and smoothed onto the fixed grid. Once the gradient has been smoothed onto the fixed grid, we can integrate to recover the marker function. For a staggered grid, for example

$$I_{i,j,k} = I_{i-1,j,k} + \left(\frac{\partial I}{\partial x}\right)_{i-1/2,j,k} \Delta x. \tag{38}$$

To maintain symmetry, the marker at a given point is constructed as the average of the integration from all the neighboring points, leading to a linear system that is solved iteratively. Using standard second order centered finite difference approximations, the linear system is an approximation to

$$\nabla^2 I = \nabla \cdot (\nabla I)_f, \tag{39}$$

where the subscript on the gradient of I on the right hand side means that it comes from the front. Since the right hand side is known (it is deduced from the position of the front through (38) this equation amounts to the Poisson equation

$$\nabla^2 I = \nabla \cdot G_f. \tag{40}$$

which must be solved to find I for a given G_f. In the current version this equation is solved for the entire grid, but this can easily be changed to involve only grid points next to the interface, where the value of the marker function changes as the front moves. Integrating the gradient of the marker function on the fixed grid is, of course, only one way to construct it. We have, however, found that doing so generally leads to a smooth but compact transition from one fluid to the other. In addition, since the gradients of two interfaces bounding a very thin film cancel each other when transferred to the fixed grid, the marker there will "disappear." This seems like the proper way to treat films too thin to be resolved on the fixed grid.

3.3.3. *Surface Tension*

In simulations of flows with sharp interfaces the front serves two main functions. The first is the advection of the marker function, as described above, and the second is the computation of the surface tension described below. As with most of the other operations for the front, finding the surface force can be done in several different ways. In the PARIS code we compute the force on the front and transfer it to the fixed fluid grid, where it is added to the discrete Navier-Stokes equations.

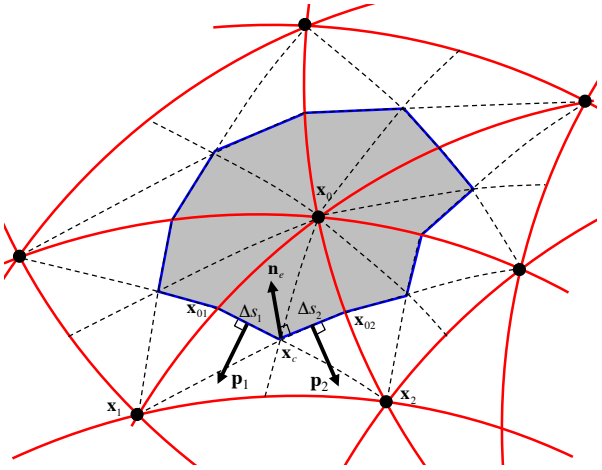

Figure 5: Computation of the surface force on a triangulated grid by integrating over the edges of an element.

To ensure that the force is conserved as we transfer it from the front and onto the fixed volume grid, we work with the total force on a small area. The total force on a small region surrounding a front point is computed by dividing each front element into three equal parts, each connected to one nodal point, and computing the pull on their side as described below. The force from each part is then added to the appropriate nodal point. When the surface force on all the elements has been computed, it is transferred to the fixed grid and converted into a force per unit volume by (35). Working with the total force on a surface element, rather than the force per area, makes it easier to ensure the total force is conserved when it is transferred between the front and the fixed volume grid.

To find the surface force we use the fact that the total force on a surface element can be found by integrating the "pull" on its edges.

$$\mathbf{f}_\sigma = \oint_{\delta A} \sigma \mathbf{m} dl. \tag{41}$$

Here $\mathbf{m}$ is a unit vector that is tangent to the interface and perpendicular to the boundary of the interface element and by keeping the surface tension coefficient σ under the integral sign we allow for variable surface tension. The benefit of using this expression is that we only need to approximate tangents on the surface—not the curvature—and that the pull on the side of one surface element is equal and opposite to the pull on the adjacent element. Thus, the surface force is conserved in the sense that an integral over a surface patch consisting of several surface elements is guarantied to give the same results as an integral over the boundaries of the whole patch. For constant surface tension coefficient, the integral over a closed surface is, in particular, guaranteed to be zero.

Figure 5 shows schematically how the integration is done.

We assume that the elements are flat and that surface tension, given at the front points, can be non-constant. It therefore has to be interpolated when approximating the integral. The force at point $\mathbf{x}_0$ is computed as the "pull" on the edges of the gray patch, surrounding it. The contribution from the element connecting points $\mathbf{x}_0$, $\mathbf{x}_1$ and $\mathbf{x}_2$ is found by first splitting it in three by connecting the centroid $\mathbf{x}_c = (1/3)(\mathbf{x}_0 + \mathbf{x}_1 + \mathbf{x}_2)$ to the mid points of the edges $\mathbf{x}_{01} = (1/2)(\mathbf{x}_0 + \mathbf{x}_1)$ and $\mathbf{x}_{02} = (1/2)(\mathbf{x}_0 + \mathbf{x}_2)$ and then finding the force on those edges by approximating the integral using a mid-point rule

$$\Delta \mathbf{f}_\sigma \approx \sigma_1 \mathbf{p}_1 \Delta s_1 + \sigma_2 \mathbf{p}_2 \Delta s_2. \tag{42}$$

Here, the pull on each element is the cross product of the outward unit normal, $\mathbf{n}_e$ and tangent vector to the edge: $\Delta s_1 \mathbf{p}_1 = \mathbf{n}_e \times (\mathbf{x}_{01} - \mathbf{x}_c)$ and $\Delta s_2 \mathbf{p}_2 = \mathbf{n}_e \times (\mathbf{x}_c - \mathbf{x}_{02})$, where Δs_1 and Δs_2 are the lengths of the edges, and σ_1 and σ_2 are the surface tensions at the midpoint of the edges. The normal is found by the normalized cross product of two of the tangent vectors to the edges of the element. After straightforward algebra, we have

$$\Delta \mathbf{f}_\sigma \approx \frac{\mathbf{n}_e}{3} \times \left\{ \sigma_1 \left[\mathbf{x}_2 - \frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_0) \right] - \sigma_2 \left[\mathbf{x}_1 - \frac{1}{2}(\mathbf{x}_2 + \mathbf{x}_0) \right] \right\} \tag{43}$$

where the interpolated surface tension at the midpoint of the edges is:

$$\begin{aligned} \sigma_1 &= \frac{1}{2} \left\{ \frac{1}{2} \left[\sigma(\mathbf{x}_0) + \sigma(\mathbf{x}_1) \right] + \frac{1}{3} \left[\sigma(\mathbf{x}_0) + \sigma(\mathbf{x}_1) + \sigma(\mathbf{x}_2) \right] \right\} \\ &= \frac{1}{12} \left[5\sigma(\mathbf{x}_0) + 5\sigma(\mathbf{x}_1) + 2\sigma(\mathbf{x}_2) \right] \end{aligned} \tag{44}$$

and σ_2 is given by a similar expression. The forces from the other elements connected to point $\mathbf{x}_0$ are found in the same way and added to give the total force on the point, that is then "smoothed" onto the fixed grid.

3.4. Volume-Of-Fluid

When the interface location is tracked by the Volume-Of-Fluid (VOF) method, a variable $C_{i,j,k}$ is initialized. It is equal to the fraction filled with fluid 1 of the cell $\Omega_{i,j,k}$. Thus fluid 1 is taken to be the reference fluid. Moreover we will use in the section a rescaling of the space and time variables so that the cell size is 1, and the time step is also 1. All velocities are then rescaled to $u' = u\tau/h$. Because of this space rescaling and in these new units, $C_{i,j,k}$ is also the measure of the volume of reference fluid in cell i, j, k.

3.4.1. Normal vector determination

The VOF method proceeds by a sequence of reconstructions and advections of C. In the reconstruction step, one attempts to find the interface geometry from Volume of Fluid data $C_{i,j,k}$. In the Paris code we use already-published methods (see for example TSZ) that have been experienced to work satisfactorily. One first determines the interface normal vector $\mathbf{n}$, then one solves the problem of finding a plane perpendicular to $\mathbf{n}$ under which one finds exactly the volume $C_{i,j,k}$. In Paris , two methods exist for normal vector determination. The most frequently used is the Mixed-Youngs-Centered Scheme (MYCS) described in TSZ. However, when a quick determination of the normal is needed and accuracy is not needed, the finite difference method $\mathbf{n} = \nabla^h C$, (the Youngs scheme) is also used.

3.4.2. Plane constant determination

Once the the interface normal vector $\mathbf{n}$ is determined, a new, colinear normal vector noted $\mathbf{m}$ and having unit "box" norm is deduced from $\mathbf{n}$, that is $\|\mathbf{m}\|_1 = |m_x| + |m_y| + |m_z| = 1$. Considering the volume $V = C_{i,j,k}$ in cell i, j, k the plane constant α is defined so that the plane

$$\mathbf{m} \cdot \mathbf{x} = \alpha \tag{45}$$

cuts exactly a volume V of the plane. The origin of the coordinate system is taken at the corner of cubic cell i, j, k with the smallest coordinate values. The reader is reminded that we used rescaled units of space, so that $0 \leq V \leq 1$ and $0 \leq \alpha \leq 1$. Then α is determined by the resolution of a cubic equation. This resolution and similar one often used in VOF are implemented in a kind of small library contained in the single file vof_functions.f90.

3.4.3. Volume initialisation

Before any VOF interface tracking is performed, the field of $C_{i,j,k}$ values must be initialized. The Paris code avoids inaccurate initializations that for example initialize a sphere as a set of $C_{i,j,k}$ values which are all 0 or 1, a so called "staircase" or "lego" initialization. There are two ways in which initialization can be improved over the lego one. In the "subgrid" initialisation, the mesh cells are subdvivided into n_I^3 subcells (where n_I is a tunable parameter, called REFINEMENT in the code). Then a "lego" initialization is performed trivially in the subcells. For example, if the initial interface is defined implicitly by the equation $\phi(\mathbf{x}) = 0$ where ϕ is a smooth implicit function (akin to a level-set function) then the trivial "lego" initialization is $c^I = \chi[\phi(\mathbf{x})]$ in the subcells where χ is the Heaviside function. The standard cell value $C_{i,j,k}$ is then determined by a summation over the subcells. In tests it was found that $n_I = 8$ was sufficient. However $n_I \geq 8$ leads to a very large number of evaluations of the function ϕ and a slow initialization. In order to avoid this, the Paris code may be linked to the Vofi library described in [18] and [19]. Then the initialization is performed using high accuracy numerical integration of the measure of the fluid volumes implicitly defined by the equation $\phi(\mathbf{x}) < 0$. The code needs to be linked to the VOFi library [19] with the shell variable HAVE_VOFI set before compilation.

3.4.4. General split-direction advection

Once the reconstruction has been performed at time t_n, it is used to obtain the approximate position of the interface, and the volumes $C_{i,j,k}$ at time t_{n+1}. The Paris code contains two advection methods, which can be selected by the user: Lagrangian Explicit (LE) with the keyword VOF_ADVECT = LE or Weymouth and Yue (WY) advection with the keyword

VOF_ADVECT = WY. The LE advection is also called "Calcul d'Interface Affine par Morceaux" (CIAM) which is french for "Piecewise Linear Interface Calculation" (PLIC) but PLIC refers to generic VOF methods with a piecewise linear reconstruction step, while CIAM refers to a specific type of advection method first described in the archival literature in [20] and classified as the "LE" method in [21]. The main advantage of both LE and WY is that they avoid overshoots ($C_{i,j,k} > 1$) and undershoots ($C_{i,j,k} < 0$). Moreover WY conserves mass to machine accuracy. These methods are described in detail in TSZ for LE/CIAM and in [22] for WY. The reader may also refer to [7] for a condensed description of both methods.

An important operation in Volume-Of-Fluid advection is "clipping". A small parameter ϵ_c is defeind for the purpose of clipping. After advection, all cells that have $C_{i,j,k} < \epsilon$ are set to 0 and all cells that have $C_{i,j,k} > 1 - \epsilon$ are set to 1. This removes some, but not all, of the wisps, floatsam and jetsam of the VOF method. In the current version of the code the default value is $\epsilon_c = 10^{-8}$. This is a rather high value and the code has been observed to function well with $\epsilon_c = 10^{-8}$ (see also [7]).

3.5. Surface Tension in Volume-Of-Fluid

3.5.1. CSF method

For simplicity, we consider only the case where σ is constant. In the Continuous Surface Force (CSF) method the capillary force $\sigma\kappa\mathbf{n}\delta_S$ given in (8) is

$$-\sigma\kappa\nabla\chi = -\sigma\kappa^h\nabla^h C. \tag{46}$$

where we have used the properties of the Heaviside function χ. One of the advantages of this formulation is that it is a "well-balanced" method (see TSZ, Chapter 7 or reference [23]).

An approximation for κ needs to be found to use the CSF method. A good estimate is obtained using so-called height functions.

3.5.2. Height Functions

We give some details about height functions since it is a relatively novel aspect of the code. Height functions were introduced in [24] and further discussed, tested and improved in several papers [25, 26]. A height function is a function on the discrete grid that gives the vertical elevation of the interface. The use of height functions greatly improves the accuracy of VOF methods since it allows us to neglect small inconsistencies in the VOF representation. On one hand a small VOF floatsam in a cell has only a very small influence on the height-function calculation. On the other hand if taken as an indication of the presence of this interface in the given cell it would create a large error on the interface location (Figure 6a). With high resolution height functions can also yield the position of the interface to fourth-order accuracy [25]. We consider for simplicity the case of an approximately horizontal interface (aligned with the x, y plane). A simple height function, rescaled by the grid size h, may be defined as

$$H = \sum_{p=k_0}^{p=k_0+n_c} C_{i,j,p}. \tag{47}$$

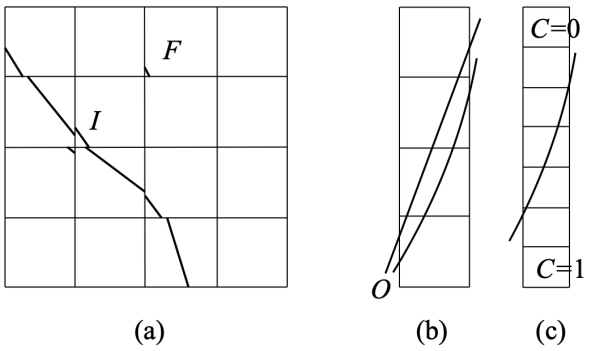

Figure 6: (a) A small floatsam F in a cell away from the interface is negligible when height functions are used to determine the location of the interface. On the other hand requiring the interface to pass near each centroid C_i has a large effect. A small inconsistency such as near point I is also ignored by the height function. (b) Two cases where the height function expression (47) is appropriate with four cells ($n_c = 3$, see text). For more vertical lines or larger curvatures the line exits the 4×1 stencil through the top and bottom and the HF method cannot be used.(c) To check the validity of a the HF calculation one needs to have one full cell ($C = 1$) below and one empty cell ($C = 0$) above, or the converse.

and is illustrated on Figure 6b. Such a height function is defined for "vertical" heights and with reference to a base at point O at the base of the bottom cell on Figure 6b. More general cases, with arbitrary orientation of the interface, are considered in Appendix A.

With height functions, the computation of the curvature may be performed by reconstructing a polynomial approximation for the height. To illustrate we take again the case of an approximately horizontal interface. Then we fit the height by

$$H(x, y) = \frac{a_1}{2}x^2 + \frac{a_2}{2}y^2 + a_3xy + a_4x + a_5y + a_6 \tag{48}$$

where the coefficients are computed using finite differences from the heights given in eq. (A.3). The curvature is then

$$\kappa = \epsilon\frac{a_1(1 + a_5^2) + a_2(1 + a_4^2) - 2a_3a_4a_5}{(1 + a_4^2 + a_5^2)^{3/2}} \tag{49}$$

where $\epsilon = 1$ if the interface is in the "canonical" position (normal pointing upwards). The above method is possible only if, in the x, y plane, all nine heights are available. If they are not, fallback methods are used, details of which are given in Appendix A.

The performance of our method for the computation of curvature is shown in Figures 7 and 8. The error was computed for a collection of diameter-to-cell-size ratios D/h. For each value of D/h the error computation was repeated for an ensemble of N sphere centers located randomly. The L_∞ norm for a given sphere center is the maximum difference between the sphere curvature and the numerically obtained curvature. The error reported on the Figures is the maximum L_∞ error for the whole ensemble of N spheres. We checked that the error varies little when N is increased above 16, and the standard test case for curvature distributed with the code uses this value of N. The differences between the error norms obtained for Basilisk and Paris are discussed in Appendix A.

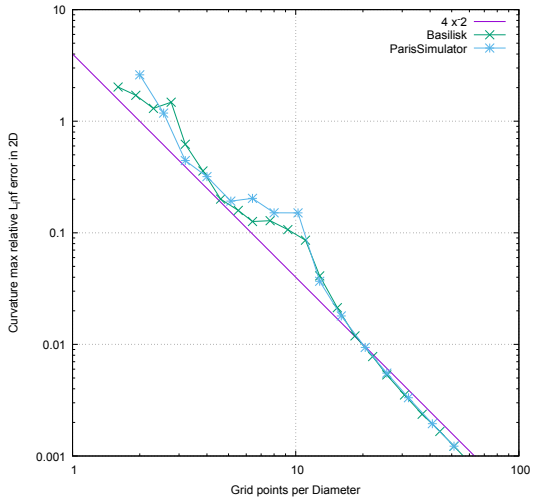

Figure 7: Maximum L_∞ error norm in two dimensions for the curvature estimated for a cylinder using the height function method in Paris and Basilisk . The mixed-height option is set in both codes.

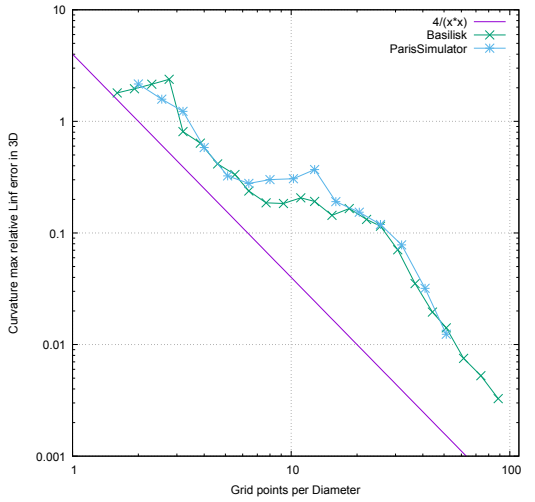

Figure 8: Maximum L_∞ error norm in three dimensions for the curvature estimated for a sphere using the height function method in Paris and Basilisk . The mixed-height option is set in both codes.

3.6. Pressure solver

3.6.1. In-code Gauss-Seidel solver

The default Poisson solver used to invert the elliptic operators appearing in eqs. (22) and (32) is a red-black Gauss-Seidel (GS) solver with overrelaxation [27]. The coefficients are arbitrary for a discrete operator of the form

$$\begin{aligned} A_{1,i,j,k}p_{i-1,j,k} &+ A_{2,i,j,k}p_{i+1,j,k} + A_{3,i,j,k}p_{i,j-1,k} \\ +A_{4,i,j,k}p_{i,j+1,k} &+ A_{5,i,j,k}p_{i,j,k-1} + A_{6,i,j,k}p_{i,j,k+1} \\ -A_{7,i,j,k}p_{i,j,k} &= A_{8,i,j,k} \end{aligned} \tag{50}$$

The coefficients verify

$$A_{7,i,j,k} = \sum_{p=1}^{6} A_{p,i,j,k} \tag{51}$$

$$A_{2,i,j,k} = A_{1,i+1,j,k} \tag{52}$$

$$A_{4,i,j,k} = A_{3,i,j+1,k} \tag{53}$$

$$A_{6,i,j,k} = A_{5,i,j,k+1} \tag{54}$$

$$\tag{55}$$

and are constructed by interpolations of $1/\rho$ (for 22) or μ (for the implicitation of the momentum diffusion). The GS solver iterates the assignment

$$\begin{aligned} p_{i,j,k} \leftarrow\ & (1-\beta)\, p_{i,j,k} + \frac{\beta}{A_{7,i,j,k}}\Big(A_{1,i,j,k}p_{i-1,j,k} \\ &+ A_{2,i,j,k}p_{i+1,j,k} + A_{3,i,j,k}p_{i,j-1,k} + A_{4,i,j,k}p_{i,j+1,k} \\ &+ A_{5,i,j,k}p_{i,j,k+1} + A_{6,i,j,k}p_{i,j,k-1} - A_{8,i,j,k}\Big) \end{aligned} \tag{56}$$

where β is an overrelaxation parameter that can be set by the user. A value of $\beta = 1.3$ is typically used.

3.6.2. HYPRE Library Multigrid solver

The HYPRE library, developed by the Lawrence Livermore National Laboratory (LLNL), is also an option to solve the elliptic equations with multigrid iterative methods. Since a structured grid is used in Paris , multiple solvers in the HYPRE library can be used. The SMG and PFMG multigrid solvers have been implemented in the code and used for large-scale simulations using up to 64,536 cores. Both SMG and PFMG are parallel semicoarsening multigrid solvers. The difference lies in that the SMG solver uses plane smoothing while the PFMG solver uses pointwise smoothing. The plane-smoothing feature makes the SMG solver more robust but less efficient. Furthermore, the scaling performance of the PFMG solver is much better than the SMG solver, since the smoothing step only involves a local stencil.

In order to take advantage of the higher efficiency of PFMG and the robustness of SMG, a solution strategy has been implemented in the code. The PFMG solver is used by default, if the iteration diverges or fails to converge within the maximum iteration number, then the code will switch to the SMG solver and redo the iteration. If the iteration converges, then the code will switch back to PFMG for the next time step. For a large-scale simulation that runs for a long time, this strategy has been shown to achieve a good performance, balancing robustness and efficiency.

The HYPRE library, at least in the versions we use, appears to control the tolerance on the residual using the L_1 norm. The code thus recomputes the residual norms and controls the accuracy using a norm of the residual chosen by the user among L_1, L_2 and L_∞.

3.6.3. In-code Multigrid solver

The code has also a native implementation of a multigrid solver for structured grids with 2^n number of points per direction. In particular the V-Cycle scheme is implemented and fully

parallelized [27]. Relaxation operations are applied starting from the finest to the coarsest first, and then from the coarsest to the finest, the number of relaxation operation being a user-adjustable parameter. One advantage of having a native multigrid solver is that it allows for an efficient solution of the Poisson equation without the necessity of having external libraries (HYPRE) installed in the system. Especially when running heavy three dimensional simulations in parallel the use of this native solver has been shown advantageous in some systems with respect to HYPRE in terms of memory manipulation.

3.6.4. GPU-accelerated solver

A GPU based solver is also available for solving the Poisson equation when a significant number of iterations is required to achieve convergence. The pressure is solved using a Jacobi method for equation (56). The need for the Jacobi method instead of a Gauss-Seidel arises because of the intrinsic nature of GPU devices. The usual domain decomposition parallelization allows the implementation of the iterative step by using a simple for loop over the indexes i, j, k. The sequentiality of the indexes cannot be achieved on GPU devices, as in this case each index combination is ideally computed simultaneously. In this sense, the larger number of iterations required by the Jacobi method is mitigated by the speed-up provided by GPUs.

The memory handling is a critical aspect in GPUs applications and it is even more critical in DNS. Although a Jacobi method intrinsically requires doubling the memory usage for the matrix p, it also enables the leanest data transfer between CPUs and GPUs, which is also a critical aspect of normal CUDA applications. A Gauss-Seidel red-black solver is in principle possible and beneficial for certain applications. In fact, let us assume that the size of the matrix p is $N_p = n_x n_y n_z$, the normal implementation of a red-black Gauss-Seidel solver would be

for all $\Omega_{i,j,k}$ cells of "red" type **do**
  compute $p_{i,j,k}$ using (56)
end for
for all $\Omega_{i,j,k}$ cells of "black" type **do**
  compute $p_{i,j,k}$ using (56)
end for
check convergence

which inherently reduces the memory usage required. On the other hand, the first **for** loop is not parallelizable in an efficient way in *CUDA*. Therefore, such an algorithm would be beneficial from a memory standpoint, but would improve the computational time only if N_p is at least 4 times greater than the number of GPU process available. As it is usually not the case, the beneficial effects of a red-black algorithm are limited, although it will be object of future studies.

The implementation of the algorithm is achieved by means of the open-source *CUDA* library for *C* developed by NVIDIA, while the intercommunication between processors is still achieved by using MPI. For this reason, an interface between *Fortran90* and *C* is created in module_CUDA.f90. By passing through the interface, each process transfers the matrices A and p to the *C/CUDA* environment (in poissonCUDA.cu) where the iteration step is performed. The boundary conditions, as well as the MPI communication, are enforced in the environment that originally created the MPI communication, hence these functions are programmed in cudaFun.f90.

3.6.5. Free surface pressure solver

A free-surface flow solver is implemented in Paris , which is designed to apply a free-surface condition as described in 2.3 for inviscid flows. For viscous cases, the stress free interface condition in (12) is not ensured, therefore the implementation is limited to inviscid cases. The free-surface solver uses the VOF method in Paris to track the interface. The flow is then solved in one phase ($\chi = 0$) using the same numerical methods as described previously. For the purpose of description the solved phase will be called liquid and the unsolved phase will be gas. The only difference between the free-surface solver and the standard one–fluid VOF approach is that the gas phase is not solved. Instead, it is assumed to have a fixed pressure that can only vary in time. The time variation in pressure is determined using a polytropic gas law

$$p_c = p_0 \left(\frac{V_0}{V_c} \right)^{\gamma} , \tag{57}$$

where V_c is the total volume of gas pocket at pressure p_c. p_0 and V_0 are respectively the reference pressure and volume of the gas phase, γ is the heat capacity ratio. The gas phase pressure along with the pressure jump due to surface tension is then applied as a Dirichlet boundary condition for the pressure in the liquid flow, as given in (11).

The method used to apply this pressure boundary condition is inspired by the idea of Fedkiw and Kang [28, 29], often referred to as the ghost fluid method. Let the time-varying pressure in the unsolved phase be p_c. Special care is required in the discretization of (22) for liquid cells near the interface. Cells that contain mostly gas are excluded from the solution, so that only cells where $C < 0.5$ are solved (here the convention is to have $C = 0$ in the liquid). Fig. 9 shows a representation of a 2D grid with a section of an interface. The grey area represents a gas-filled volume. Cells that contain a filled circle are included in the pressure solution, while cells without a marker are excluded. Since only the liquid phase is solved, only the liquid density is applied. Furthermore, for cubic cells $\Delta x = \Delta y = \Delta z = h$, where h is the constant grid spacing.

The stencil for the pressure gradient components has to be changed near the interface when a neighbouring cell falls inside the gas phase. This cell's pressure must be substituted by a surface pressure. We apply a finite difference gradient approach as Chan [30]. As an example, the approximation for the pressure gradient components for the cell with indices i and j in Fig. 10 is written

$$\nabla_x^h p_{i+1/2,j} = \frac{p_{s,i+1,j} - p_{i,j}}{\delta_{i+1/2,j}} \; ; \quad \nabla_y^h p_{i,j-1/2} = \frac{p_{i,j} - p_{s,i,j-1}}{\delta_{i,j-1/2}} , \tag{58}$$

where δ is the distance between the pressure node under consideration and the intersection with the interface. The pressure

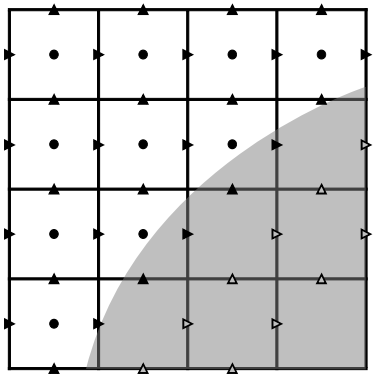

Figure 9: A 2D section of the numerical grid, showing part of a gas bubble in grey. Circles represent computational cell nodes where pressure is calculated. Triangles indicate scalar velocity components on the computational cell faces. Filled triangles indicate values which are found by solving the governing equations, while unfilled triangles represent boundary values found by extrapolation.

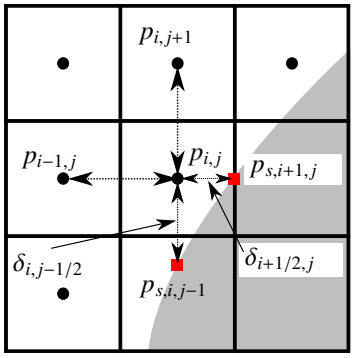

Figure 10: Discretisation of the pressure equation near the interface

p_s on the liquid side of the interface is found by adding to p_c the Laplace pressure jump. The pressure p_c inside each cavity is known from the polytropic law (57). The interface pressure in the x-direction will then be

$$p_{s,i+1,j} = p_{c,i+1,j} + \sigma \frac{\kappa_{i,j} + \kappa_{i+1,j}}{2} . \tag{59}$$

From (59) and (58) it is clear that accurate interface curvature as well as an accurate prediction of the interface location are important parameters to ensure the accuracy of the pressure solution. Since the height function is the approximate interface distance from some reference cell in a given direction, it is used for δ. When the interface configuration is such that a height cannot be obtained in the required direction, the distance is approximated by using a plane reconstruction of the interface in the staggered volume.

Extrapolation of the velocity field. The resolved velocity components right next to the interface will require neighbours in the gas phase to discretize the momentum advection term. These values in the gas phase can be seen as boundary values to the resolved velocities. In order to find neighbours in the gas phase, we extrapolate the resolved velocities similarly to Popinet [31].

After calculating the liquid velocities using standard methods in Paris , the boundary velocities in the gas phase are updated for the next time step from the closest two velocity neighbours using a linear least square fit. Let's assume the velocity field can be described as a linear combination

$$\mathbf{u}(\mathbf{x}) = \mathbf{A} \cdot (\mathbf{x} - \mathbf{x}_0) + \mathbf{u}_0 \tag{60}$$

where the components of the tensor $\mathbf{A}$ and of the vector $\mathbf{u}_0$ are the unknowns.
If we now take a 5×5 stencil around the unknown gas velocity at location $\mathbf{x}_0$, we can find the extrapolated velocity $\mathbf{u}_0$ by minimizing the functional

$$\mathcal{L} = \sum_{k=1}^{N} \left| \mathbf{A} \cdot (\mathbf{x} - \mathbf{x}_0) + \mathbf{u}_0 - \mathbf{u}_k \right|^2 \tag{61}$$

This is done first for all locations closest to the resolved velocities $\mathbf{u}_k$ ("first neighbours"), whereafter the process is repeated for the "second neighbours". Note that only resolved velocity components are included in the cost function, therefore the number of solved velocities N can vary depending on the shape of the interface. Furthermore, because of the staggered grid, only one velocity component of $\mathbf{u}_0$ is computed at any location $\mathbf{x}_0$.

Ensuring volume conservation. The extrapolation of liquid velocities into the gas phase was explained in the previous section. An additional step is required to ensure that the extrapolated velocities are divergence free. This is required to ensure that the advection of C is conservative.

A similar approach to Sussman [32] is used. Only the first two layers of cells inside the gas phase are considered and all other cells are disregarded. A 2D example is presented in Fig. 11. Similar to the projection step explained earlier, a "phantom"

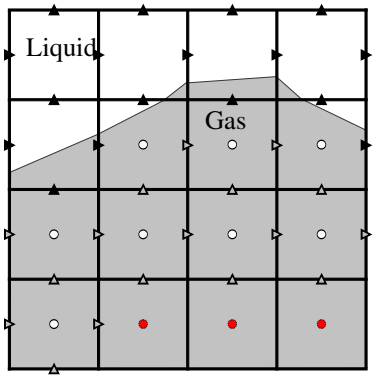

Figure 11: 2D example of the problem to correct the extrapolated velocities (unfilled triangles). A Poisson problem is solved in the cells marked with an unfilled circle.

pressure is obtained in these cells by solving a Poisson equation

$$\nabla^h \cdot \left(\nabla^h \hat{P} \right) = \nabla^h \cdot \tilde{\mathbf{u}} \,, \tag{62}$$

where $\hat{P}$ is the "phantom" pressure and $\tilde{\mathbf{u}}$ is the velocity on the faces of the first two gas neighbours. $\hat{P}$ is only calculated in the cells represented by unfilled nodes in Fig. 11. On the liquid side of these cells, the solved velocities (filled triangles) are used as a velocity boundary condition with the pressure gradient on this face set to zero. On the gas side outside the cells we consider (red filled circles), a fixed pressure is prescribed. Only the extrapolated velocities (unfilled triangles) are then corrected by the solved pressure gradient, $\nabla \hat{P}$

$$\tilde{\mathbf{u}}^{n+1} = \tilde{\mathbf{u}} - \nabla^h \hat{P} \tag{63}$$

to ensure non-divergence of velocity in the first two layers of cells just inside the gas.

For more details on the numerical method and its application in idealized micro-spall, see [33] and [34].

3.7. Solid boundaries

Solids are defined in a "block" or "Lego" manner. A domain-wide binary flag $s_{i,j,k}$ is defined that is equal to 0 inside the solid and 1 outside. A set of link-based flags s^x, s^y and s^z is also defined (a link-based array is data located on the velocity component collocation points such as $i + 1/2, j, k$). The following pseudo code is executed at initialisation:

for all i, j, k **do**
  $s^x_{i+1/2,j,k} \leftarrow s_{i,j,k}$
  $s^x_{i-1/2,j,k} \leftarrow s_{i,j,k}$
  (same for y, z directions)
end for

The indexes s^x, s^y and s^z are then used to "block" the velocity and the pressure correction on the solid region and its boundary. This is done each time the velocity is updated:

for all i, j, k **do**
  $u_{i+1/2,j,k} \leftarrow s_{i,j,k} u_{i+1/2,j,k}$
  (same for v, w components)
end for

The pressure correction should not change the velocity, so on the links a Neuman boundary condition for the pressure is established, which is equivalent to setting to zero some coefficients:

for all i, j, k **do**
  $A_{1,i,j,k} \leftarrow s^x_{i-1/2,j,k} A_{1,i,j,k}$
  $A_{2,i,j,k} \leftarrow s^x_{i+1/2,j,k} A_{2,i,j,k}$
  (same for A_3, A_4 and s^y, and for A_5, A_6 and s^z)
  $A_{7,i,j,k} \leftarrow \sum_{p=1}^{6} A_{p,i,j,k}$
end for

The solid domain can be initialized by implicit functions or by loading a file containing the $s_{i,j,k}$ data. In both cases it is important that the presence of the solid does not make the linear system (22) ill-posed. This will happen for example if the boundary conditions are Dirichlet for the velocity at the entry of a channel and the solid completely blocks the channel. There is thus a small utility program "rockread.c", distributed with the code, that checks that the fluid "percolates".

4. Testing

The testing of the code is performed automatically. The short version of testing is done by typing make test, the long version by typing make longtest. The short version takes approximately 5 minutes on a laptop with an i7 processor and the long version takes approximately 25 minutes. All the resulting tests give a report of "PASS" or "FAIL". Each test is contained in a subdirectory of the Test directory. The subdirectory corresponding to a test has a self-explanatory name, e.g. PresPoiseuille for the pressure-driven Poiseuille flow.

The tests can be divided into two categories, elementary tests which are basically sanity checks verifying that the code is not corrupted and finds elementary flows easily, and more complex test that are in some cases a verification of the code, comparing it to analytical solutions. However, the verification has not been pushed very far, since the code is an assembly of methods that have been tested extensively elsewhere, see for example TSZ for a review of these tests. The more recent methods such as the "momentum-conserving" option for velocity advection, has been tested extensively in [7] although several test cases of the method exist and will be described below.

4.1. Elementary tests

4.1.1. Poiseuille flow

An elementary Poiseuille flow [35] is tested. The simulation is set up in a $8 \times 8 \times 2$ grid in which the system reduces to a 2D planar flow in the box $(0, 1) \times (0, 1)$. The parameters are $\|\nabla p\| = \mu = \rho = 1$. The boundary conditions are set pressure on the left and right. The simulation is continued until the flow becomes stationnary, which happens with 10^{-3} accuracy around time 1. This time is reached in 1000 time steps. The explicit version of viscous diffusion is used.

This test passes if the numerical segments are tangent to the theoretical profile as seen in Figure 12. Because the Poiseuille

flow profile is quadratic, second order finite differences offer exact values for the second derivative of velocity, which ensures that the profile found is exactly a parabola. The accuracy of the amplitude of that parabola is set by the quality of the approximation of the $u = 0$ (solid wall) boundary condition. Since that is set at first order, there is a small $O(h^2)$ difference with the exact parabola.

The bottom wall tangential velocity boundary condition is $u = 0$ on $y = 0$. Since the wall is at $y_{i,1/2}$, the boundary condition is written using a ghost velocity at $y_{i,-1/2}$ that satisfies $u^{\text{ghost}}_{i,-1/2} + u_{i,1/2} = 2u^{\text{wall}}$. The boundary condition is thus implemented by writing in a "ghost layer" of the grid

$$u^{\text{ghost}}_{i,-1/2} \leftarrow 2u^{\text{wall}} - u_{i,1/2}. \tag{64}$$

The result is shown on Figure 12. It is also possible to run the

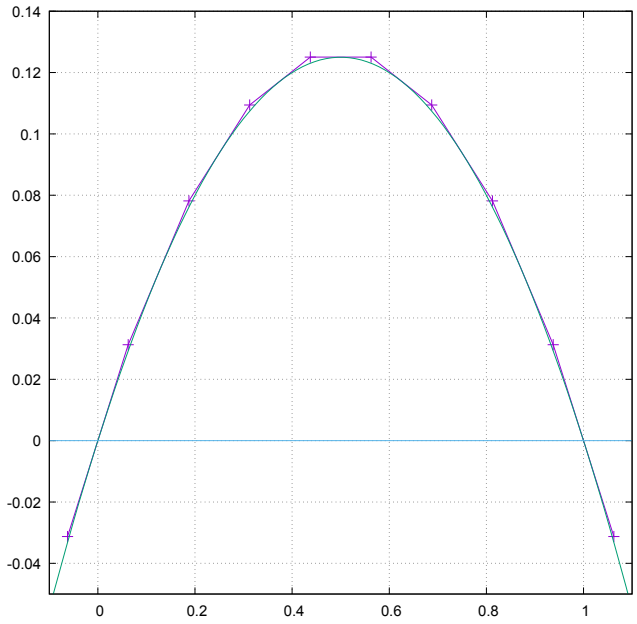

Figure 12: Poiseuille flow test case.

Poiseuille flow test in other manners, with set inflow velocity for example. It is also possible to run it with the implicit version of viscous diffusion, in which case the flow converges in a few time steps.

4.1.2. Stokes flow around a disk

A pressure driven flow around a disk of diameter $1/2$, with the other parameters as before, is set and left to evolve until steady state. The advection operator $\mathcal{L}_{\text{adv}}$ is turned off which ensures that the steady state is a Stokes flow. The explicit version of the viscous terms is used. The test resides in the test directory PresDisk. If the implicit version is used, convergence to the steady state can be faster but an error of order τ affects the solution. The result is shown on Figure 13. There are two other similar tests beyond this first one. In the second one, located

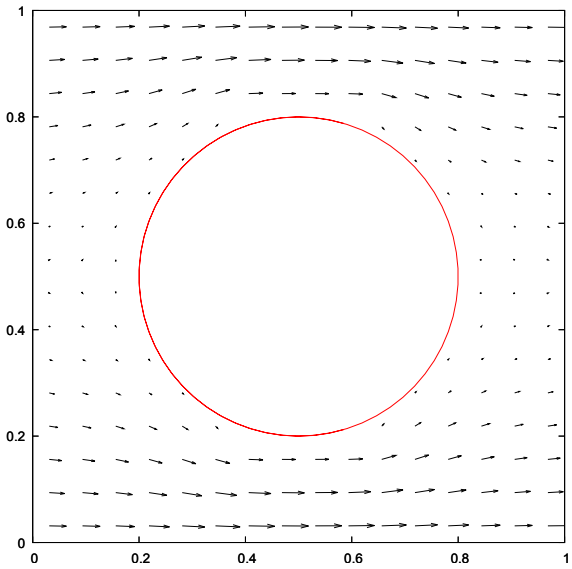

Figure 13: Stokes flow around a disk test case.

in the directory Disk, the flow is driven by a body force akin to gravity in the test directory instead of being driven by pressure. This second test is still with periodic boundary conditions. The third test has inflow and outflow conditions and is located in the test directory Inflowdisk.

4.1.3. Droplet advection

This is an elementary test to check whether the VOF method is operating normally. The final state of the field $C_{i,j,k}$ is compared to a precomputed value. The test has to be visualized "by hand" by the user, with the help of graphics software such as VISIT or PARAVIEW. One should see an underformed droplet move accross the domain.

4.1.4. Cylinder advection

This is a more sophisticated test that probes the "momentum-conserving" option. The test is described in detail in [7]. If the velocity field stays uniform and constant and as a result the droplet is advected undeformed, it means that momentum $\rho\mathbf{u}$ and density ρ are advected in lock-step, so that the operation $\rho\mathbf{u}/\rho$, at each time step, gives the constant $\mathbf{u}$.

4.1.5. Speed

This is not so much a test as a report on the code speed. However a significant drop of the code speed would be a serious issue.

4.2. Capillary wave

In this section we present results of the oscillation of planar capillary waves between two viscous fluids with equal density and viscosity in the presence of surface tension. The interface between the two fluids is slightly perturbed with a sinusoidal function of small amplitude a_0 and the initial velocity is set to

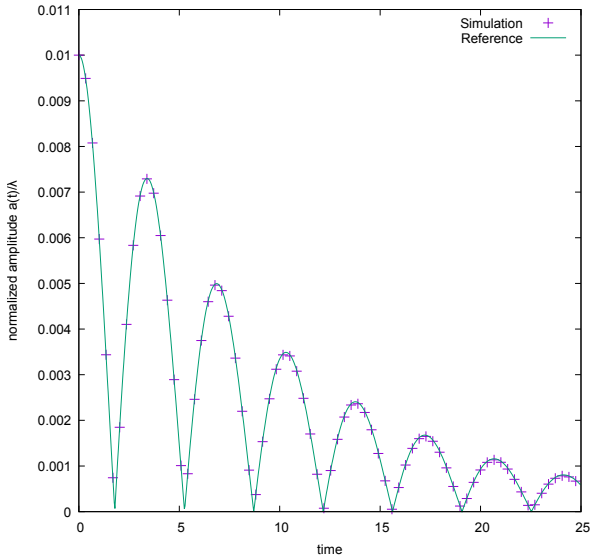

Figure 14: Comparison of the temporal evolution of the amplitude of the interface perturbation obtained numerically and the analytical initial-value solution by [36].

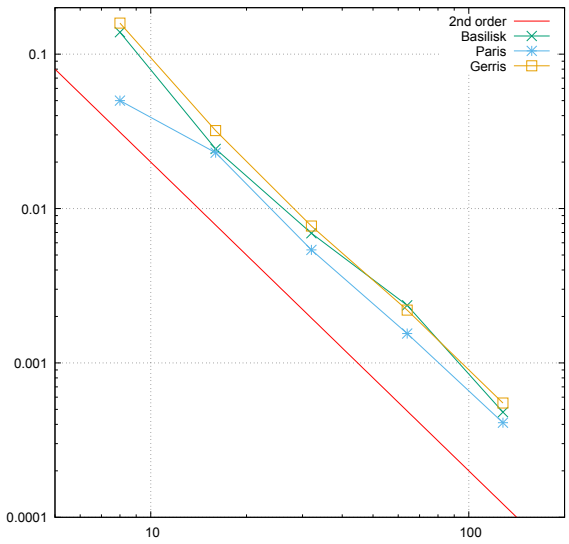

Figure 15: Relative L_2 error of the numerical solution as a function of the number of grid points per wavelength $N = \lambda/h$ for the capillary wave.

$N = \lambda/h$	8	16	32	64	128
Gerris	0.159	0.032	0.0077	0.0022	$5.5\,10^{-4}$
Basilisk	0.139	0.024	0.0069	0.0024	$4.8\,10^{-4}$
Paris	0.050	0.023	0.0054	0.0015	$4.1\,10^{-4}$

Table 1: Relative L_2 error of the numerical solution for capillary waves for various codes and numbers of grid points N per wavelength. The errors estimated by the codes have been rounded to the nearest digit. Results for Gerris are from the website http://gfs.sf.net, not from the paper [24]. Results for Basilisk, obtained by the authors from the code published on the website http://basilisk.fr, are similar.

D	μ_l	μ_g	ρ_l	ρ_g	σ
(m)	$(kg\,m^{-1}\,s^{-1})$	$(kg\,m^{-1}\,s^{-1})$	$(kg\,m^{-3})$	$(kg\,m^{-3})$	$(kg\,s^{-2})$
$3\ 10^{-3}$	10^{-3}	$1.7\ 10^{-5}$	10^3	1.2	0.0728

Table 2: Physical parameters (defined in the text) for the oscillating droplet.

zero. The solution of this problem is governed by the Laplace number which is $La = \sigma\rho\lambda/\mu^2 = 3000$, where λ is the wavelength of the sinusoidal function. Simulation are performed in a box of dimensions $L_x = \lambda$ and L_y. The results are compared to the analytical initial-value solution (AIVS) obtained in [36, 37] for small-amplitude capillary waves in viscous fluids. In the AIVS the aspect ratio L_y/L_x should be sufficiently large (at least 2) and the initial capillary wave amplitude a_0 sufficiently small. Moreover the time step τ and the tolerance of the solvers should be at convergence. We have checked that all these four parameters were at convergence for fixed h. We then look at the dependance of the remaining error on h. Figure 14 compares the temporal evolution of the amplitude of the interface perturbation with the AIVS.

To analyze the methods, the relative L_2 error norm of the difference between the numerical and the AIVS solution is computed. The results, depicted in Table 1 and Figure 15 show second order convergence for coarse grids. For very refined grids, when the error is below 1%, the accuracy on the solution is controlled by the initial amplitude of the wave. In the case of the resolution $\lambda/h = 128$, instead of a 0.01 amplitude as in [24], a 0.005 amplitude had to be used to match the AIVS.

4.3. Oscillating droplets and bubbles

A droplet with a large density ratio is initialized with a small ellipsoidal deformation. The droplet has air-water properties described in Tables 2 and 3. The initial shape, when tracked with the Front, is shown in Figure 16. This test is not designed to assess the accuracy of the code, since the methods used in the code have already been used and tested elsewhere (see for example [38] for a oscillating droplet test with Volume of Fluid methods and [39, 40] for similar tests with Front Tracking. It should however be noted that in all of these references the tests are 2D, hence easier). We thus expect the accuracy to be similar to that of already published and tested codes using similar curvature and surface tension methods. The purpose of the test is rather to ensure that the code is working as expected, and

r	m	La
ρ_l/ρ_g	μ_l/μ_g	$\sigma\rho_l d/\mu_l^2$
833.3	58.82	218400

Table 3: Dimensionless parameters for the oscillating droplet. La is the Laplace number.

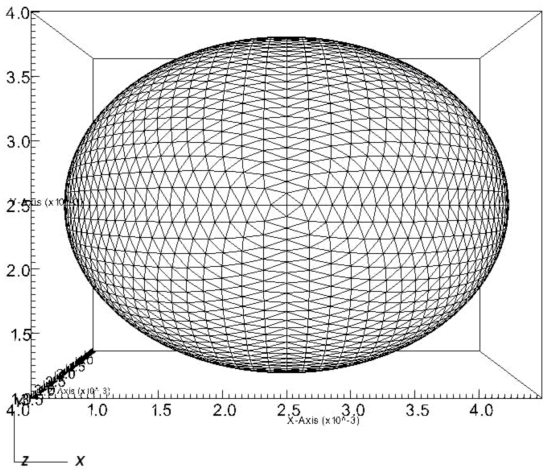

Figure 16: Initial ellipsoidal shape of the droplet or bubble with the Front.

to assess whether air water properties, which are often creating stability problems, are in fact in the stable regime of the code.

Figure 17 shows the amplitude oscillations. It shows the amplitude oscillations in time for a Droplet of $D/h = 19.2$ grid points per diameter and an initial excentricity of 0.75. The test shown is run without the momentum conserving option, which is here seen to be unnecessary for the stability, and with the mixed-height option discussed in Section Appendix A.3. The reference solution plotted alongside the test simulation result is obtained from the same VOF simulation at the larger resolution $D/h = 38.4$. Satisfying agreement is found.

Note that when this test is run automatically in the test suite, the reference solution stored in the Test/Droplet directory has been obtained by our code running in the same conditions, a device frequently used in several test cases in the code test suite. That way, one tests that the behavior of the code has not changed drastically, but not that the code (original version and current version) is correct.

We test the Front-Tracking part of the scheme by simulating the same droplet with the Front. Results are shown on Figure 18. the reference solution is obtained from the VOF simulation at larger resolution, with $D/h = 38.4$ grid points. Satisfying agreement is found.

We repeat these tests by just inverting the phases, thus initializing an air bubble inside water. The physical and the numerical parameters (such as the scheme options) are the same as before.

Figure 19 shows the amplitude oscillations, reproducing the results from the Bubble test case. It shows the kinetic energy oscillations in time for a Bubble of $D/h = 19.2$ grid points per diameter and an initial excentricity of 0.75. The kinetic energy is used this time instead of the deformation amplitude. The reference solution is again obtained from the VOF simulation at larger resolution, with $D/h = 38.4$ grid points. There is again good agreement.

We test again the Front-Tracking part of the scheme by simulating the same bubble with the Front. Results are shown on Figure 20. In this case as in the previous one the reference solu-

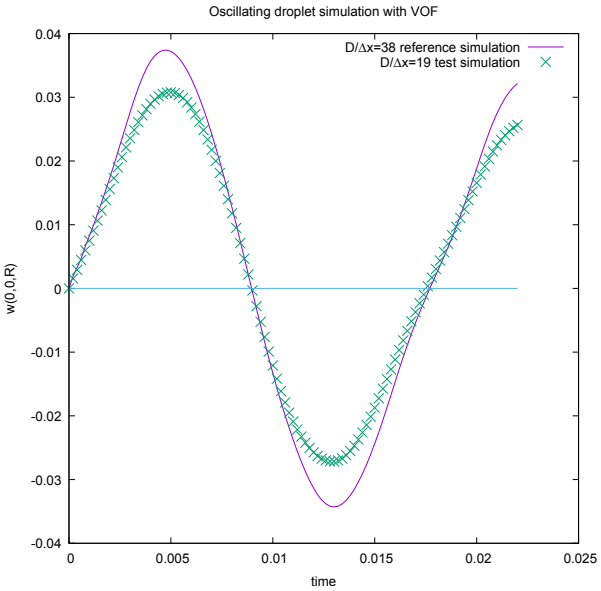

Figure 17: Amplitude of capillary oscillations of the Droplet test case.

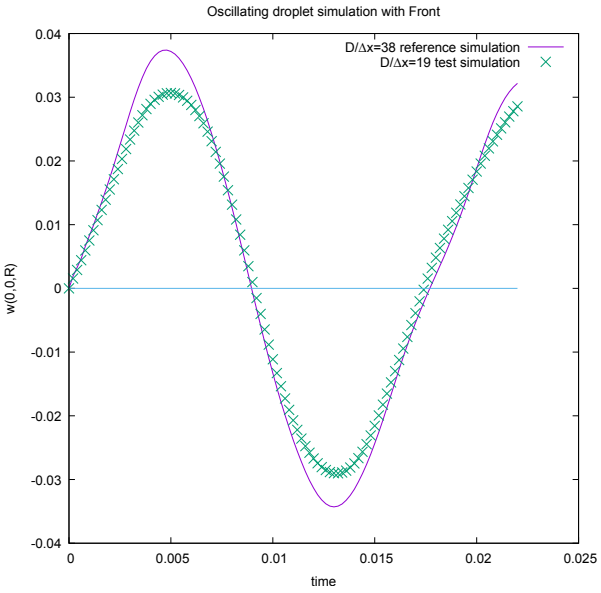

Figure 18: Amplitude of capillary oscillations of the FrontDroplet test case.

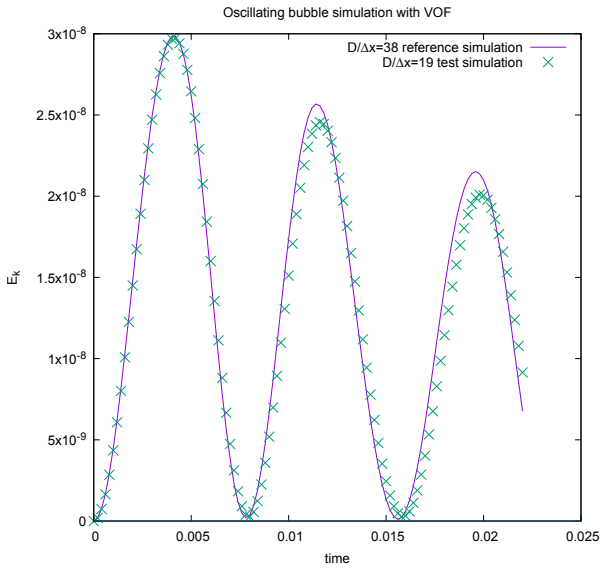

Figure 19: Kinetic Energy of capillary oscillations of the Bubble test case.

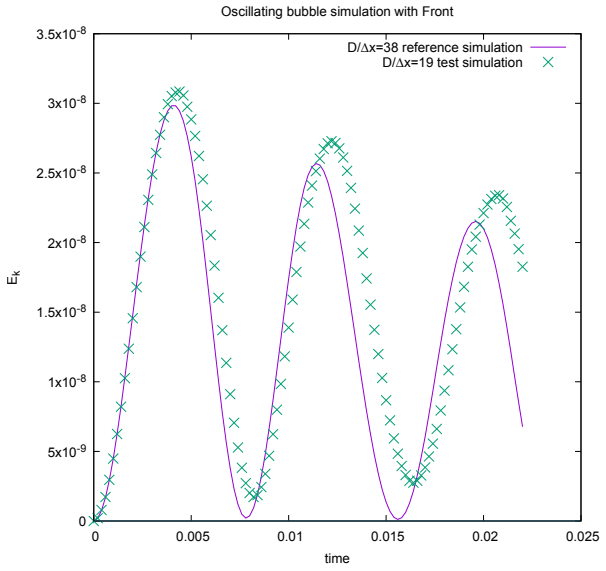

Figure 20: Kinetic Energy of capillary oscillations of the FrontBubble test case.

μ_l $(kg\,m^{-1}\,s^{-1})$	μ_g $(kg\,m^{-1}\,s^{-1})$	ρ_l $(kg\,m^{-3})$	ρ_g $(kg\,m^{-3})$
$1.0016\,10^{-3}$	$1.835\ 10^{-5}$	998.2	1.19

Table 4: Physical parameters for the raindrop test. Only the parameters that differ from Table 2 are given.

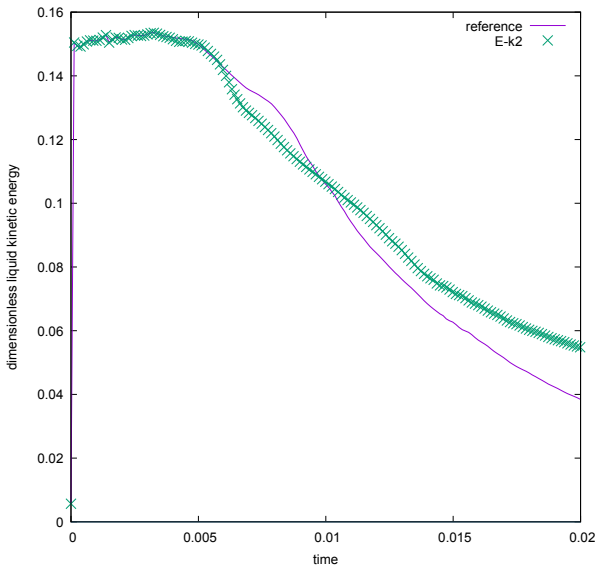

Figure 21: Kinetic Energy of a 3mm falling raindrop at low resolution $D/\Delta x = 8$.

tion is obtained from the VOF simulation with the same larger resolution $D/h = 38.4$. The agreement is satisfying but a small drift of the kinetic energy is observed.

4.4. Falling raindrop

This is an important test case, since it is very demanding for several codes. A spherical raindrop is setup with a diameter d =3mm and allowed to fall in air under gravity. The droplet should remain approximately spherical, with a pancake or bun like shape, but in many codes one finds this case difficult and spurious atomisation of the droplet is seen. For an exemple with the basilisk code see [41]. For details about the setup of the test we refer the reader to [7]. The test is performed with the parameters of Table 4. Notice that these parameters differ slightly from those of [7] as we input here the experimental values for air and water at 20deg C. The grid resolution is low $D/\Delta x = 8$, since the test is more demanding (it leads to a blowup of the code more easily) for low resolutions. Figure 21 provides the result of the Raindrop test in the code distribution. It is seen that the solution deviates somewhat from the reference, however this is not worrisome since the flow is in a regime that is very sensitive to parameters and initial conditions and any small change in the code will create a deviation of the sort. The shape of the droplet is shown in Figure 22.

Figure 22: Simulated shape of a 3mm falling raindrop at low resolution $D/\Delta x = 8$.

5. Installation and Usage

5.1. External Libraries

When solving elliptic equations, we may apply the Hypre package's [42] SMG, a semi-coarsening multigrid solver with 3D plane smoothing on structured, cuboid meshes, as mentioned in Section 3.6.2.

Installation of the static library versions (*.a files) of Hypre is controlled via the Makefile. For example, for a Linux system, this can be realized via the line

```
HYPRE_DIR = $(HOME)/some_path/hypre-2.10.0b/src/lib
```

in the Makefile. Note that the file `libHYPRE.a` is placed in the directory `HYPRE_DIR`. Consequently, the actual linking is ensured by another Makefile line:

```
HYPRE_LIBS = -L$(HYPRE_DIR) -lHYPRE
```

which is specified in the default Paris distribution. Note that the Hypre version used for the majority of Paris development has been 2.10.1; it has been tested for stability in serial and massively parallel runs [33]. In case of hypre-related problems, fallback to v. 2.10 is recommended for debugging.

VOFI [18, 19] is a library of Volume-of-fluid related procedures (see Section 3.4.3). It is an interesting option in cases where initial conditions depend strongly on a very precise interface geometry, e.g. Free Surface solutions of bubble dynamics [34, 33] and initialisation takes time compared to computation (as in the curvature test case). Linking of this library is performed in the exact same fashion as in the case of Hypre. The static library file is named `libvofi.a`, and the respective linker switch is `-lvofi`.

Seeing as Hypre and Vofi are purely optional, compilation without them is made easy. The user may set or unset the variables HAVE_VOFI and HAVE_HYPRE in his shell prior to compiling. If these variables are set the Makefile will attempt to locate the corresponding libraries. If not the compilation without the libraries proceeds. The fallback for Hypre is the built-in Gauss-Seidel solver followed by the in-code Multigrid solver, and the fallback for the Vofi are the built-in VOF initialization procedures.

5.2. Output Files

Various output formats are available in the code: VTK, SILO, and MPI I/O. While the VTK option generates ASCII files, the latter two produce binary files. The output in SILO format is done based on the SILO library developed by LLNL. For both the VTK and SILO output options, the independent parallel approach is used, namely every MPI process generates a separate file. This will become an issue for large-scale simulations using a large number of MPI processes, since creating a large number of small files simultaneously may crash the indexing server. An output option based on the MPI I/O library is implemented in PARIS for large-scale simulations, which adopts, instead, a cooperative parallel approach and creates a single file for each variable for each output. A post-processing code was developed in PARIS to convert the MPI I/O outputs and SILO files offline for visualization. This code is available in the distribution as the file util/post_utility.f90 .

5.3. Input Files

Paris requires a set of input files (in text format) to initialize and start the simulation. These files are:

- input - global and front-solver parameters.
- inputFS - Free Surface solver parameters (optional, Free Surface simulations only).
- inputlpp - Lagrangian particle module parameters (optional, implicitly requires two-phase flow, see ref. [43]).
- inputvof (optional, Volume of Fluid parameters, as above).
- inputsolids (optional, solid objects parameters).

All input files contain over 220 parameters, thus listing all of them is not practical in this paper; instead we will only point to general rules governing the use of these files. However the reader may find the default values of these parameter in the source code, usually just below the namelist instruction.

All the input lines contain the parameter specifications written as "variable = value", with values being reals, integers, string or boolean (T/F). In most cases (in the code versions distributed in main darcs tree) the variables are commented Fortran-style, i.e. in the same line, following an exclamation mark, for example:

```
MaxDt   = 5.e-5  ! maximum size of time step
dtFlag  = 2      ! 1: fixed dt; 2: fixed CFL
dt      = 1.0e-4 ! dt in case of dtFlag=1
MAXERROR= 1.0d-6 ! Residual for Poisson solver
CFL     = 0.042
```

It must be noted that the Paris source code uses the Fortran namelist input type, consequences of that being that all input files have "free format", i.e. lines can change order or be deleted. All variables are initialized to default values in the

source code. Thus, PARIS will initialize with an empty input file – however in such a case the simulation will be short. Indeed by default Nx=0 (the grid has zero points in x direction) in order to prevent simulations with some of the absurd input files that could be selected by mistake. Thus a minimum input file should contain at least a specification of Nx. For more demanding simulations, beginner users are encouraged to familiarize themselves with input file examples, such as templates found in the `Tests` sub-directory in the distribution which can be copied and modified to create new PARIS cases. Note that in the Test suite, input files are often generated from template files such as `inputfilename.template` using shell scripts.

6. Acknowledgements

We thank Dr. V. Le Chenadec, Mr. C. Pairetti, Dr. S. Popinet and Dr. S. Vincent for useful conversations on the topics of this paper.

Portions of this work were supported by National Science Foundation Grant CBET-1335913, by the ANR MODEMI project (ANR-11-MONU-0011) program and by grant SU-17-R-PER-26-MULTIBRANCH from Sorbonne Université. This work was granted access to the HPC resources of TGCC-CURIE, TGCC-IRENE and CINES-Occigen under the allocations t20152b7325, t20162b7760, 2017tgcc0080 and A0032B07760, made by GENCI and TGCC. The authors would also like to acknowledge the MESU computing facilities of Sorbonne Université. Finally, the simulation data are visualized by the software VisIt developed by the Lawrence Livermore National Laboratory.

References

[1] R. Scardovelli, S. Zaleski, Direct numerical simulation of free-surface and interfacial flow, Annu. Rev. Fluid Mech. 31 (1999) 567–603.

[2] G. Tryggvason, R. Scardovelli, S. Zaleski, Direct Numerical Simulations of Gas-Liquid Multiphase Flows, Cambridge University Press, 2011.

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

[4] S. Popinet, The gerris flow solver, http://gfs.sf.net (2001-2014).
URL `http://gfs.sf.net`

[5] S. Popinet, Basilisk, a Free-Software program for the solution of partial differential equations on adaptive Cartesian meshes (2018).
URL `http://basilisk.fr`

[6] S. Dabiri, D. Fuster, Y. S. Ling, L. Malan, R. Scardovelli, G. Tryggvason, P. Yecko, S. Zaleski, PARIS Simulator Code: A PArallel Robust Interface Simulator that combines VOF and Front-Tracking., http://parissimulator.sf.net (2012-2015).

[7] D. Fuster, T. Arrufat, M. Crialesi-Esposito, Y. Ling, L. Malan, S. Pal, R. Scardovelli, G. Tryggvason, S. Zaleski, A momentum-conserving, consistent, volume-of-fluid method for incompressible flow on staggered grids, arXiv preprint arXiv:1811.12327.

[8] M. Bussmann, D. B. Kothe, J. M. Sicilian, Modeling high density ratio incompressible interfacial flows, in: ASME 2002 Joint US-European Fluids Engineering Division Conference, American Society of Mechanical Engineers, 2002, pp. 707–713.

[9] O. Desjardins, V. Moureau, Methods for multiphase flows with high density ratio, Center for Turbulent Research, Summer Programm 2010 (2010) 313–322.

[10] M. Raessi, H. Pitsch, Consistent mass and momentum transport for simulating incompressible interfacial flows with large density ratios using the level set method, Computers & Fluids 63 (2012) 70–81.

[11] V. Le Chenadec, H. Pitsch, A monotonicity preserving conservative sharp interface flow solver for high density ratio two-phase flows, J. Comput. Phys. 249 (2013) 185–203.

[12] S. Ghods, M. Herrmann, A consistent rescaled momentum transport method for simulating large density ratio incompressible multiphase flows using level set methods, Physica Scripta 2013 (T155) (2013) 014050.

[13] G. Vaudor, T. Menard, W. Aniszewski, M. Doring, A. Berlemont, A consistent mass and momentum flux computation method for two phase flows. Application to atomization process, Computers & Fluids 152 (2017) 204–216.

[14] J. K. Patel, G. Natarajan, A novel consistent and well-balanced algorithm for simulations of multiphase flows on unstructured grids, Journal of computational physics 350 (2017) 207–236.

[15] N. Nangia, B. E. Griffith, N. A. Patankar, A. P. S. Bhalla, A robust incompressible navier-stokes solver for high density ratio multiphase flows, Journal of Computational Physics.

[16] M. Rudman, A volume-tracking method for incompressible multi-fluid flows with large density variations, Int. J. Numer. Meth. Fluids 28 (1998) 357–378.

[17] D. M. McQueen, C. S. Peskin, A three-dimensional computational method for blood flow in the heart: (ii) contractile fibers, Journal of Computational Physics 82 (1989) 289–297.

[18] S. Bnà, S. Manservisi, R. Scardovelli, P. Yecko, S. Zaleski, Numerical integration of implicit functions for the initialization of the VOF function, Computers & Fluids 113 (2015) 42–52.

[19] S. Bnà, S. Manservisi, R. Scardovelli, P. Yecko, S. Zaleski, Vofi — a library to initialize the volume fraction scalar field, Computer Physics Communications 200 (2016) 291–299.

[20] J. Li, Calcul d'interface affine par morceaux (piecewise linear interface calculation), C. R. Acad. Sci. Paris, série IIb, (Paris) 320 (1995) 391–396.

[21] R. Scardovelli, S. Zaleski, Interface reconstruction with least-square fit and split lagrangian-eulerian advection, Int J. Numer. Meth. Fluids 41 (2003) 251–274.

[22] G. D. Weymouth, D. K. P. Yue, Conservative Volume-of-Fluid method for free-surface simulations on Cartesian-grids, J. Comput. Phys. 229 (8) (2010) 2853–2865.

[23] S. Popinet, Numerical models of surface tension, Annual Review of Fluid Mechanics 50 (2018) 49–75.

[24] S. Popinet, An accurate adaptive solver for surface-tension-driven interfacial flows, J. Comput. Phys. 228 (2009) 5838–5866.

[25] G. Bornia, A. Cervone, S. Manservisi, R. Scardovelli, S. Zaleski, On the properties and limitations of the height function method in two-dimensional cartesian geometry, J. Comput. Phys. 230 (2011) 851–862.

[26] M. Owkes, O. Desjardins, A mesh-decoupled height function method for computing interface curvature, J. Comput. Phys. 281 (2015) 285 – 300. doi:http://dx.doi.org/10.1016/j.jcp.2014.10.036.
URL `http://www.sciencedirect.com/science/article/pii/S002199911`

[27] W. L. Briggs, A multigrid tutorial, SIAM Philadelphia, 1987.

[28] R. P. Fedkiw, T. Aslam, B. Merriman, S. Osher, A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method), Journal of computational physics 152 (2) (1999) 457–492.

[29] M. Kang, R. P. Fedkiw, X.-D. Liu, A boundary condition capturing method for multiphase incompressible flow, Journal of Scientific Computing 15 (3) (2000) 323–360.

[30] R. K. C. Chan, R. L. Street, A computer study of finite-amplitude water waves, J. Comput. Phys. 6 (1970) 68–94.

[31] S. Popinet, S. Zaleski, Bubble collapse near a solid boundary: a numerical study of the influence of viscosity, J. Fluid Mech. 464 (2002) 137–163.

[32] M. Sussman, A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles, J. Comput. Phys. 187 (2003) 110–136.

[33] W. Aniszewski, S. Zaleski, A. Llor, L. Malan, Numerical simulations of pore isolation and competition in idealized micro-spall process, International Journal of Multiphase Flow-doi:https://doi.org/10.1016/j.ijmultiphaseflow.2018.10.013.
URL `http://www.sciencedirect.com/science/article/pii/S030193221`

[34] L. Malan, Y. Ling, R. Scardovelli, A. Llor, S. Zaleski, Direct numerical simulations of pore competition in idealized micro-spall using the VOF method, Computers & Fluids (submitted) also available as: arXiv:1711.04561 [physics.flu-dyn].
URL `arXiv:1711.04561 [physics.flu-dyn]`

[35] P. Kundu, I. Cohen, D. Dowling, Fluid Mechanics, 891 pp, Elsevier, Amsterdam, 2012.
[36] A. Prosperetti, Motion of two superposed viscous fluids, Phys. Fluids 24 (1981) 1217–1223.
[37] F. Denner, G. Paré, S. Zaleski, Dispersion and viscous attenuation of capillary waves with finite amplitude, The European Physical Journal Special Topics 226 (6) (2017) 1229–1238.
[38] D. Fuster, G. Agbaglah, C. Josserand, S. Popinet, S. Zaleski, Numerical simulation of droplets, bubbles and waves: state of the art, Fluid Dyn. Res. 41 (6) (2009) 065001.
[39] D. J. Torres, J. U. Brackbill, The Point-Set Method: Front-Tracking without Connectivity, Journal of Computational Physics 165 (2) (2000) 620–644.
[40] U. Olgac, D. Izbassarov, M. Muradoglu, Direct numerical simulation of an oscillating droplet in partial contact with a substrate, Computers and Fluids 77 (C) (2013) 152–158.
[41] C. Pairetti, S. Popinet, S. M. Damián, N. Nigro, S. Zaleski, Bag mode breakup simulations of a single liquid droplet, 2018, 6th European Conference on Computational Mechanics (ECCM 6)7th European Conference on Computational Fluid Dynamics (ECFD 7)1115 June 2018, Glasgow, UK.
URL http://www.eccm-ecfd2018.org/admin/files/filePaper/p1694.pdf
[42] P. Sloot, C. Tan, J. Dongara, A. H. (Editors), Hypre: a library of high performance preconditioners, in: Computational Science - ICCS, Springer-Verlag, ICSS 2002, Berlin, 2002.
[43] Y. Ling, S. Zaleski, R. Scardovelli, Multiscale simulation of atomization with small droplets represented by a lagrangian point-particle model, Int. J. Multiphase Flow 76 (2015) 122 – 143. doi:http://dx.doi.org/10.1016/j.ijmultiphaseflow.2015.07.002.
URL http://www.sciencedirect.com/science/article/pii/S0301932215001524
[44] R. Scardovelli, S. Zaleski, Analytical relations connecting linear interfaces and volume fractions in rectangular grids, J. Comput. Phys. 164 (2000) 228–237.

Appendix A. Details of curvature computation from Height Functions

Appendix A.1. Height computation

The height computation is performed as described in Section 3.5.2. A more general definition than (47) is to consider for each cell $\Omega_{i,j,k}$ the possible existence of one of six height functions defined with reference to a direction $\mathbf{e}_a$, $1 \le a \le 3$, where $\mathbf{e}_a$ is one of the cartesian base vectors algined with the grid, and an orientation $\epsilon = -\text{sign},(\mathbf{e}_a \cdot \nabla C)$ (The minus sign ensures that the canonical situation where the "fluid" $C = 1$ is below the "air" $C = 0$ has $\epsilon = 1$. It also corresponds to the sign convention for the interface normal $\mathbf{n}\delta_S = -\nabla\chi$). This cell-and-orientation-dependent HF is defined as

$$H^{(a,+)}_{i,j,k} = \sum_{\text{stencil } S} C_{l,m,n} - L_{i,j,k} \tag{A.1}$$

where the sum is over all the cells in a "stencil" or "stack" S centered on containing $\mathbf{x}_{i,j,k}$ and oriented parallel to $\mathbf{e}_a$ and for the "positive" orientation ϵ. The distance $L_{i,j,k}$ is the distance $L_{i,j,k} = \epsilon(\mathbf{x}_{i,j,k} - \mathbf{x}_O) \cdot \mathbf{e}_a$ from the base of the stack to the cell center. When the orientation is $\epsilon = -1$

$$H^{(a,-)}_{i,j,k} = \sum_{\text{stencil } S} (1 - C_{l,m,n}) - L_{i,j,k} \tag{A.2}$$

and the distance $L_{i,j,k}$ is now with reference to an origin in direction $-\mathbf{e}_a$ from the cell. An example of stack is shown on Figure 6(c). This height function can be computed whenever the bottom cell and the top cell of the stack both do not contain the interface, and the interface crosses only once the intermediate cells. This can be tested by requiring that that there are cells with $C = 0$ and $C = 1$ at the top and bottom 6(c).

For a straight line interface the height function is exact. It is interesting to see how many cells are needed in the stack S to find the height for a straight line in 2D. The most "difficult" case is the 45deg case. Thus, considering a cell crossed by the interface, one seems to need to explore one cell above or one cell below that cell. With the addition of the full and empty cells this requires the *exploration* of two cells above and below the starting cell. The total number of cells for a symmetric stencil about the starting cell would thus be five, but the total number of cells in a stencil maybe as low as four. However with even a vanishing amount of curvature five cells in a symmetric stencil are not sufficient and seven cells are needed. Thus one would need to explore $N_d = 3$ cells above or below. In three dimensions the most "dangerous" cases now have the plane normal $\mathbf{n} = (1, 1, 1)$. A similar reasoning also leads to a height of seven cells for the symmetric stencil in 3D. Note that the stencil does not have to be symmetric, rather this is a accidental feature or our implemention of the method.

For each cell, it is determined whether there is a full or empty cell at a maximum distance N_d (the parameter NDEPTH in the code) above or below the cell. The value NDEPTH=3 is hard coded. In order to function in parallel, and since only two layers of celles are exchanged between MPI processes, the sum in (47) is broken in two parts, one in each processor. Then the processes exchange two informations, the "partial sums" so computed, and the lengths $L_{i,j,k}$ in expression (A.1) allowing them to reconstruct the full sum.

Appendix A.2. First pass, first attempt: full aligned heights

In the first pass, a loop is performed over all cells cut by the interface. These are defined as cells that have $0 < C_{i,j,k} < 1$. In this first pass two attempts are made. The current cell Ω_0 has grid coordinates i_0, j_0, k_0. In the first attempt, the normal $\mathbf{n}$ is estimated by MYCS. Then the grid direction $\mathbf{e}_a$ closest to the normal is determined (maximum of $\mathbf{n}.\mathbf{e}_a$). Without loss of generality consider the case $a = 3$ and consider the plane perpendicular to $\mathbf{e}_3$. This plane is then horizontal and it is the grid plane most aligned with the interface. A 3×3 planar slate of cells Σ_0, aligned with this plane, is selected containing cells Ω_{i,j,k_0} such that $i_0 - 1 \le i \le i_0 + 1$ and $j_0 - 1 \le j \le j_0 + 1$. For all these cells, either a height $H_{i,j}$ is readily available, or is searched in the above and below cells over two layers, that is for $k_0 \pm 1$ and $k_0 \pm 2$. When all nine heights are available, the coefficients of the polynomial (48) can be found using

$$\begin{aligned}
a_1 = \partial^2_{xx} H &\simeq H_{1,0} - 2H_{0,0} + H_{-1,0},\\
a_2 = \partial^2_{yy} H &\simeq H_{0,1} - 2H_{0,0} + H_{0,-1},\\
a_3 = \partial^2_{xy} H &\simeq (H_{1,1} - H_{-1,1} - H_{1,-1} + H_{-1,-1})/4,\\
a_4 = \partial_x H &\simeq (H_{1,0} - H_{-1,0})/2,\\
a_5 = \partial_y H &\simeq (H_{0,1} - H_{0,-1})/2.
\end{aligned}$$

Appendix A.3. First pass, second attempt: mixed heights

If the first attempt fails then a "mixed heights" approach is used but only if the parameter MIXED_HEIGHTS is set to 'T'. For every height $H^{(a,\epsilon)}_{i,j,k}$ that has been calculated, a point $\mathbf{x}^a_{i,j,k} = H^{(a,\epsilon)}_{i,j,k}\mathbf{e}_a + x_b\mathbf{e}_b + x_c\mathbf{e}_c$ is defined, where x_b and x_c are the cell-central coordinates in the two directions other than $\mathbf{e}_a$. Since there are six height directions, up to 54 points could be computed. However, a general orientation is computed using the MYCS normal and only the orientations compatible with that orientation are retained, which yields 27 possible points. With certain point configurations there is a risk of a degenerate case where the least-square linear operator is not invertible. This happens in particular in the set of six points obtained with combinations of $x = 0, 1$, $y = -1, 0, 1$, $z = 0$. All paraboloids of the form $z = x(1 - x)$ pass through these points. Other degeneracies are possible: points all on a circle will be fitted by infinitely many revolution paraboloids $z' = \kappa(x^2 + y^2)/2$. To avoid these degeneracies and after trial and error the minimum number of points requested is hardcoded as N_s =NFOUND_MIN +1= 7. In addition to these "mixed heights" points the centroid of the VOF face in the current cell Ω_{i,j,k_0} is added to the set of points to fit. In some cases, different directions $\mathbf{e}_a$ could yield two close points in the same cell or in a neighboring cell. Points which are closer than $h/2$ are rejected. In equations, if $\|\mathbf{x}^a_{i,j,k} - \mathbf{x}^b_{l,m,n}\| < h/2$ then one of the two points is rejected. Which point is rejected depends on the order in which points are added to the stack, which in turn depends on the order in which mixed heights are investigated, typically closest to the general orientation etc. Before the fitting is performed, an approximate normal is computed using the MYCS approach and the coordinate system is rotated so that the z axis is now aligned with the approximate normal. The rotation is optional and is controlled by the parameter DO_ROTATION. We found that performing rotation had a certain positive influence on the accuracy of the results, although it is not clear why.

By default the parameter MIXED_HEIGHTS is set to 'T' (true). This gives less accurate results in L_1 norm for curvature, but a smoother computation and as a result simulations appear to be more stable for large density ratios when the momentum-conserving scheme is not used. The results in Figure 7 are with MIXED_HEIGHTS='F' . With MIXED_HEIGHTS='T' one obtains the results of Figure A.23. The results without mixed heights and those from Basilisk are added for comparison. There is a difference remaining with the basilisk computations than we have not yet been able to explain.

Appendix A.4. Averaging scheme

A new loop over all cells cut by the interface is started. If both schemes above have failed in the current cell, an average is performed over neighboring cells that have been succesful by either method in the first pass. For each cell $\Omega_{i,j,k}$, the cubic set of neighbors

$$B_{i,j,k} = \{\Omega_{l,m,n} | i-1 \le l \le i+1, j-1 \le m \le j+1, k-1 \le n \le k+1\}$$

is defined. If at least one of the cells in $B_{i,j,k}$ has been succesful, the resulting curvature in $\Omega_{i,j,k}$ is the average curvature of these succesful cells.

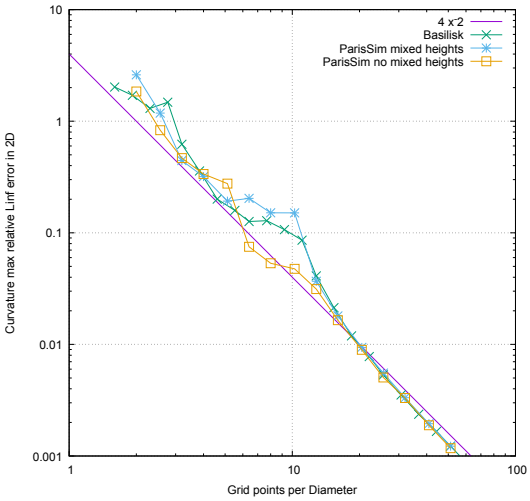

Figure A.23: Maximum L_∞ error norm in two dimensions for the curvature estimated for a cylinder using the height function method in Paris and Gerris . Two Paris results are shown, one with the mixed curvature option and one without. Averaging is used in both cases. Using the mixed-cell option yields less accurate results than not using it.

Appendix A.5. Second pass: centroid fit

A final loop on cells $\Omega_{i,j,k}$ is performed. If all three heights approaches above have failed or are not set to be used, then one falls back to a fitting of centroids. In each cell of $B_{i,j,k}$ containing an interface, the cell centroid is computed using the vof_functions microlibrary included in the code (implementing the method in [44]). Except for very small fragments, the interface must find at least five neighboring cells in addition to the current cell. This gives six points with which to fit the six parameters in expression (48).

In some cases, the fit fails because the least-square linear operator is not invertible. The code then reports in a statistical manner these failures and flags the cell as having an uncomputable curvature. A zero surface tension force is then added to the momentum.

Appendix A.6. Comparison with other implementations of height-function curvature

The accuracy contrast when modifying the mixed-heights option is even more dramatic in 3D, see Figure A.24. There is a striking drop in the L_1 error near 13 grid points per diameter. This drop is due to the averaging step in Appendix A.4. Suppressing the averaging reverts the results to accuracies comparable to those of Basilisk or slightly better.

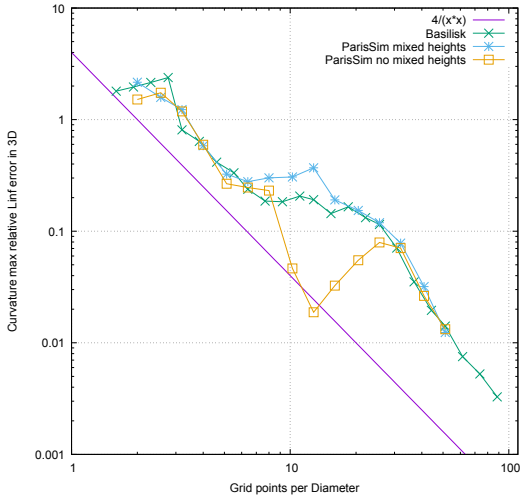

Figure A.24: Maximum L_∞ error norm in three dimensions for the curvature estimated for a sphere using the height function method in Paris and Basilisk . Two Paris results are shown, one with the mixed curvature option and one without. Averaging is used in both cases. Using the mixed-cell option yields less accurate results than not using it.